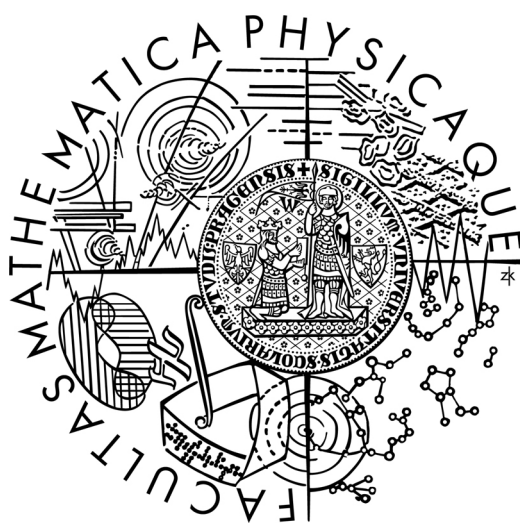


Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Martin Zeman

Využití ontologií pro GUHA procedury

Katedra softwarového inženýrství

Vedoucí diplomové práce: Mgr. Martin Ralbovský

Studijní program: Informatika

Studijní obor: softwarové systémy

Chtěl bych poděkovat zejména vedoucímu práce Mgr. Martinu Ralbovskému za skvělé vedení diplomové práce, cenné rady a připomínky a za vstřícnost při konzultacích. Dále bych chtěl poděkovat Doc. Ing. Vojtěchu Svátkovi, Dr. a Ing. Ondřeji Švábovi za poskytnutí konzultací z oblasti ontologií. A Rostislavu Taudovi za odbornou kontrolu gramatiky a stylistiky.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 13. 12. 2007

Martin Zeman

Obsah

1.	Úvod.....	6
1.1.	Cíl a struktura diplomové práce	6
1.1.1.	Cíl diplomové práce	6
1.1.2.	Struktura diplomové práce	6
1.2.	Dobývání znalostí z databází.....	8
1.3.	Metoda GUHA a GUHA procedury.....	10
1.4.	Systém Ferda Data Miner.....	10
1.4.1.	GUHA procedury implementované ve Ferdovi	12
1.5.	Ontologie.....	14
2.	Využití ontologií při aplikacích GUHA procedur v prostředí Ferda.....	15
2.1.	Využití ontologií v procesu DZD.....	15
2.1.1.	Porozumění problematice.....	15
2.1.2.	Porozumění datům.....	15
2.1.3.	Příprava dat	17
2.1.4.	Modelování.....	17
2.1.5.	Interpretace výsledků	18
2.1.6.	Využití výsledků	18
2.2.	Postupy využití ontologií (PVO) ve Ferdovi navržené v [2].....	18
2.2.1.	Mapování.....	18
2.2.2.	Využití ontologie pro identifikaci chybějících atributů (resp. sloupců).....	25
2.2.3.	Automatická tvorba atributů.....	25
2.2.4.	Vytvoření a využití skupin příbuzných atributů.....	33
2.3.	PVO, které byly v [2] shledány jako nevhodné pro implementaci	35
2.3.1.	Využití ontologie pro identifikaci redundantních atributů (resp. sloupců)	35
2.3.2.	Dekompozice 4FT úloh v závislosti na ontologii.....	35
2.3.3.	Tvorba úloh pomocí ontologie	35
2.3.4.	Propagace zjištěných znalostí do ontologie	37
3.	Volba jazyka pro reprezentaci ontologií	38
3.1.	Prezentační ontologie	38
3.2.	OWL (Web Ontology Language).....	39
3.3.	Topic Maps.....	41
3.4.	Srovnání Prezentačních ontologií, Topic Maps a OWL.....	41
3.5.	Jak v ontologii uchovat rozšiřující informace o entitách	42
4.	Implementace modulů (krabiček).....	44
4.1.	Krabička Ontologie	44
4.2.	Krabička Mapování ontologie.....	46
4.3.	Krabička Sloupec podporující ontologie.....	49
4.4.	Krabička Atribut odvozený z ontologie	50
5.	Experimentální část.....	53
5.1.	Příprava ontologie	53
5.1.1.	Ontologie UMLS_Ferda.....	53
5.2.	Mapování sloupců datového zdroje na entity ontologie.....	54
5.3.	Kategorizace atributů	55
5.4.	Sestavení úlohy	57
5.5.	Zhodnocení experimentální části	58
5.6.	ADAMEK – druhá úloha experimentální části	58

6.	Závěr.....	60
6.1.	Shrnutí vykonané práce.....	60
6.2.	Náměty na další práci.....	60
6.2.1.	Univerzální krabička pro kategorizaci atributu.....	61
6.2.2.	Automatická kombinace příbuzných booleovských atributů.....	61
6.2.3.	„Fuzzy“ ontologie.....	61
6.2.4.	„Nadkrabičky“.....	61
	Reference.....	62
	Dodatek A: Slovníček pojmů.....	64
	Dodatek B: Návod na vytvoření ontologií.....	66
	Dodatek C: Obsah příloženého CD.....	68
	Dodatek D: Vývoj jazyků pro reprezentaci ontologií.....	69

Název práce: Využití ontologií pro GUHA procedury

Autor: Martin Zeman

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí diplomové práce: Mgr. Martin Ralbovský

e-mail vedoucího: ralbovsm@vse.cz

Abstrakt:

Cílem diplomové práce je implementace dříve navržených postupů pro využití ontologií při procesu dobývání znalostí z databází (DZD). Implementace je provedena pomocí krabiček v prostředí Ferda, což je modulární prostředí na vizuální dobývání znalostí pomocí GUHA procedur. Práce diskutuje jednotlivé navržené postupy s ohledem na jejich následnou implementaci. Velká část práce je věnována výběru vhodného jazyka pro reprezentaci ontologií. Dále je popsán průběh implementace vybraných krabiček. Tyto krabičky jsou poté na názorném příkladu otestovány, je zhodnocen jejich přínos pro proces DZD a jsou navrženy možnosti následného rozvoje podpory ontologií při procesu DZD ve Ferdovi.

Klíčová slova:

dobývání znalostí, ontologie, prostředí Ferda, OWL, OWL API, metoda GUHA

Title: Usage of Ontologies for GUHA Procedures

Author: Martin Zeman

Department: Department of Software Engineering

Supervisor: Mgr. Martin Ralbovský

Supervisor's email address: ralbovsm@vse.cz

Abstract:

The goal of this master thesis is to implement some of the previously suggested ways of usage of Ontologies in Knowledge Database Discovery (KDD) process. The implementation is done in the Ferda environment, a modular environment for visual GUHA data mining. The work argues about each of the suggested ways of usage of ontologies in consideration of their successive implementation. Great part of the work is devoted to selection of the right language for representation of ontologies. Furthermore, the implementation of the selected modules in Ferda is described. The functionality of these modules is then tested on an example and there is evaluated the contribution of the modules for KDD process. At the end of the work the suggestions for future development of support of ontologies during the KDD process in Ferda DataMiner are described.

Keywords:

Data mining, ontology, Ferda environment, OWL, OWL API, GUHA method

1. Úvod

1.1. Cíl a struktura diplomové práce

1.1.1. Cíl diplomové práce

Cílem diplomové práce je rozšířit možnosti procesu zadávání úlohy pro DZD (dobývání znalostí z databází) v prostředí Ferda [1] o využití doménových znalostí zachycených ve formě ontologie. Diplomová práce navazuje na [2] a má za úkol rozšířit a implementovat postupy navržené v této práci. Součástí řešení je návrh ontologie specifické pro potřeby DZD pomocí GUHA procedur a výběr vhodné reprezentace této ontologie.

1.1.2. Struktura diplomové práce

Úkolem první kapitoly této diplomové práce je uvést čtenáře do řešené problematiky. Kromě popisu cílů a struktury práce obsahuje první kapitola popis hlavních témat, které bylo potřeba před samotnou prací nastudovat a jejichž znalost je potřebná pro pochopení dalších kapitol této práce. Těmito tématy jsou: proces dobývání znalostí pomocí metodologie CRISP-DM (kapitola 1.2), princip dobývání znalostí pomocí metody GUHA (kapitola 1.3), vizuální prostředí pro dobývání znalostí Ferda (kapitola 1.4) a teorie ontologií (kapitola 1.5).

V kapitole 2.1 jsou diskutovány možnosti využití ontologií v jednotlivých fázích procesu DZD (dle metodologie CRISP-DM). V následující kapitole 2.2 jsou pak diskutovány konkrétní návrhy postupů pro využití ontologií (označovány dále jako PVO) v prostředí Ferda. Tyto návrhy byly formulovány a diskutovány v práci [2], v níž se kromě návrhů z předcházejících prací objevily také zcela nové návrhy pro využití ontologií v procesu DZD.

Kapitola 3 úspěšně řeší nejdůležitější úkol této práce: volbu jazyka pro reprezentaci ontologií, který by vyhovoval požadavkům PVO z předcházející kapitoly.

Následující kapitola 4 popisuje moduly prostředí Ferda (tzv. krabičky), které byly v rámci této práce zvoleny k implementaci. Tato kapitola popisuje implementované krabičky, jejich funkčnost, vlastnosti, akce apod. Zároveň jsou zde popsány a diskutovány zajímavé otázky a problémy, které se v průběhu implementace jednotlivých krabiček objevily.

Kapitola 5 popisuje experimentální část diplomové práce. V rámci této kapitoly jsou testovány implementované krabičky na reálném příkladě. Tato kapitola diplomové práce může zároveň sloužit jako návod uživatelům Ferdy, kteří chtějí využít výhod nově implementovaných krabiček.

Závěrečná kapitola poskytuje souhrn vykonané práce (podkapitola 6.1) a náměty pro její využití do budoucna (podkapitola 6.2).

Pro snadnější čtenářovo pochopení je v diplomové práci použita řada ilustrativních příkladů. Mou snahou bylo, aby tyto příklady byly konzistentní, a proto se drtivá většina příkladů týká stejné dataminingové úlohy. Touto úlohou je analytická otázka 13.c. projektu STULONG [3]¹. Zdrojem dat pro tyto příklady je databáze STULONG (součást Přílohy 1 této práce).

Analytická otázka STULONG 13.c.

Podle nadváhy a krevního tlaku lze rozdělit pacienty do čtyř skupin:

- hypertonici s nadváhou
- hypertonici bez nadváhy
- normotonici s nadváhou
- normotonici bez nadváhy

Pacient je **hypertonik**,
jestliže jeho systolický krevní tlak je ≥ 140 mm Hg
a/nebo jeho diastolický krevní tlak je ≥ 90 mm Hg.

Pacient je **normotonik**,
jestliže jeho systolický krevní tlak je menší než 140 mm Hg
a zároveň jeho diastolický krevní tlak je menší než 90 mm Hg.

Pacient má **nadváhu**,
jestliže jeho BMI (body mass index) je ≥ 25 kg/m² (BMI = váha v kg / (výška v m)²).

Pacient je **bez nadváhy**, jestliže jeho BMI je ≤ 25 kg/m².

K uvedeným čtyřem skupinám pacientů se vztahuje několik analytických otázek. Jednou z nich je otázka 13.c.:

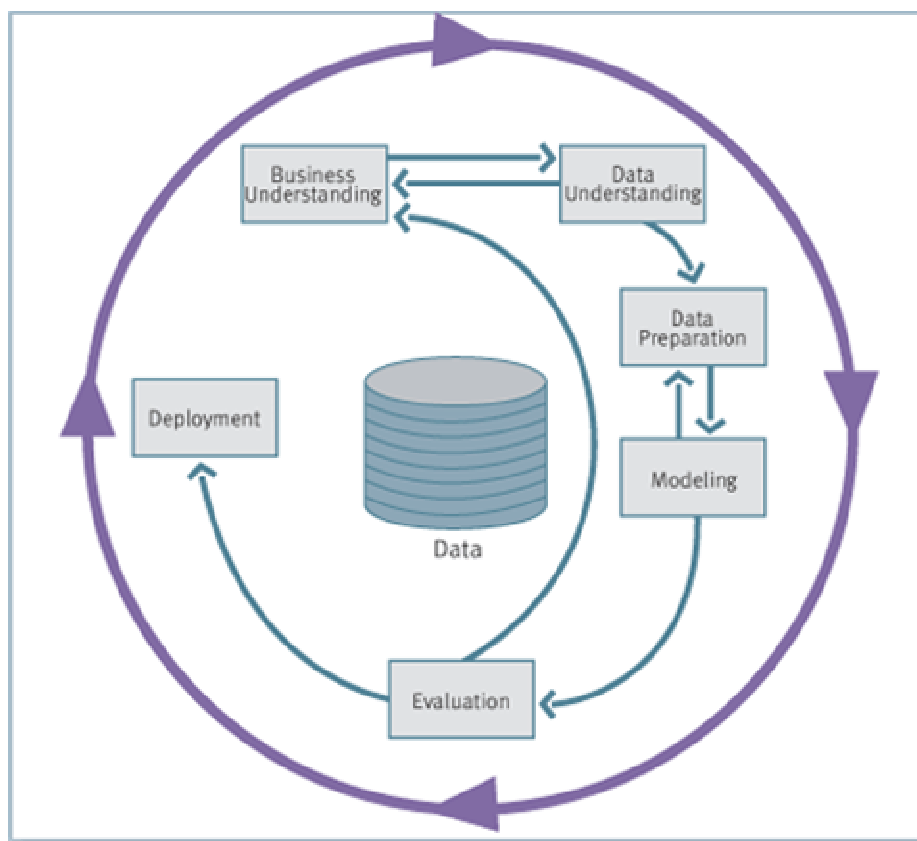
Liší se jednotlivé skupiny pacientů vzhledem ke skupině atributů kouření?

Příklad 1: Analytická otázka STULONG 13.c

¹ Studie STULONG byla realizovaná na II. Interní klinice První lékařské fakulty Karlovy Univerzity a Všeobecné Fakultní nemocnice Univerzity Karlovy, U nemocnice 2, Praha 2 (vedoucí Prof. M. Aschermann, MD, SDr, FESC), pod dohledem Prof. F. Boudíka MD, ScD, ve spolupráci s M. Tomečkovou, MD, PhD, a Ass. Prof. J. Bultasem, MD, PhD. Data byly převedeny do elektronické podoby pracovištěm EuroMISE Karlovy Univerzity a Akademie věd (vedoucí Prof. J. Zvárová, DrSc). Zdroj dat je na webových stránkách <http://euromise.vse.cz/challenge2004>. V současné době je analýza dat podporovaná grantem Ministerstva školství, mládeže a tělovýchovy České republiky č. LN 00B 107.

1.2. Dobývání znalostí z databází

Dobývání znalostí z databází (dále DZD, angl. Knowledge Discovery in Databases) je jednou z dynamicky se rozvíjejících disciplín informatiky. Cílem DZD je získávání netriviálních, skrytých a potenciálně užitečných informací z dat. Pro samotný proces dobývání znalostí byla vyvinuta řada metodologií, nejznámější a nejpoužívanější jsou metodologie 5A, SEMMA a CRISP-DM. Všechny tyto metodologie, jejichž cílem je popsat jednotlivé kroky procesu dobývání, jsou si velmi podobné. Liší se pouze v počtu a pojmenování jednotlivých fází procesu. V následujícím odstavci je popsána metodologie CRISP-DM.



Obrázek 1: Metodologie CRISP-DM (obrázek je převzatý z [4])

Metodologie CRISP-DM rozděluje celý proces dataminingového projektu do šesti základních fází, v jejichž rámci rozlišuje další kroky. Těmito fázemi jsou:

- *Porozumění problematice (Business Understanding)* – cílem úvodního kroku je celkové pochopení řešené problematiky, definování cílů (čeho chceme projektem docílit – určení správné otázky je nezbytným předpokladem k získání požadovaných informací), zvážení alternativ řešení, sestavení plánu projektu, identifikování potřebných zdrojů, identifikování rizik, která mohou ohrozit projekt, apod.
- *Porozumění datům (Data Understanding)* – v druhém kroku jde o určení dat, která budou potřeba k dobývání znalostí, jedná se jednak o definici dat a jednak o metody, jak tato data získat.
- *Příprava dat (Data Preparation)* – příprava, neboli transformace dat, bývá zpravidla nejnáročnějším úkolem při dobývání znalostí, jelikož data často nejsou v kvalitě, v jaké bychom je potřebovali, nebo někdy neexistují vůbec, je potřeba se s tím vhodně vypořádat. Tento krok obnáší různé metody, ať už

samotné získávání dat, jejich doplňování, „čištění“ (mazání nesmyslných hodnot, apod.) nebo jejich formátování.

- *Modelování (Modeling)* – tento krok bývá také překládán jako *Analytické procedury*, základem této fáze je výběr konkrétního algoritmu (obvykle jich bývá více), který bude použit pro analýzu. Důležité je určit mechanismus, kterým se následně bude testovat kvalita a správnost vytvořeného modelu (algoritmus s odpovídajícími parametry) – např. při klasifikaci je měřítkem kvality poměr správných a nesprávných klasifikací.
- *Interpretace výsledků (Evaluation)* – po nalezení kvalitního (z pohledu datové analýzy) modelu, je potřeba tento model přezkoumat z pohledu splnění obchodních cílů a detailně se zamyslet, zda nebyla opomenuta nějaká významná okolnost, která by mohla výpočet ovlivnit. S ohledem na výsledky hodnocení a revize procesu se rozhodne, zda projekt ukončit a přesunout se do fáze *Využití výsledků*, nebo zda zahájit další opakování některých fází nebo dokonce začít zcela nový dataminingový projekt.
- *Využití výsledků (Deployment)* – posledním krokem je využití výsledků, metodologie říká, že každý úspěšně ukončený projekt musí mít nějaký výstup, ať už se jedná o výstupní zprávu, nebo například vytvoření a zavedení opakujícího se dataminingového procesu.

Jednotlivé kroky procesu dobývání znalostí jsou různé časově náročné a mají různou důležitost pro úspěšné vyřešení dané úlohy. V mnoha volně dostupných zdrojích na internetu¹ se uvádí, že fáze porozumění datům zabírá často více než 80% času celého DZD projektu. V popisu procesu dobývání znalostí na [5] je doslova napsáno: „*Praktici v oboru uvádějí, že nejdůležitější je fáze porozumění problematice (80 % významu, 20 % času) a časově nejnáročnější je fáze přípravy (transformace) dat (80 % času, 20 % významu²). Překvapivě málo práce zaberou vlastní analýzy (5 % času, 2 % významu).*“ Více o CRISP-DM viz [4].

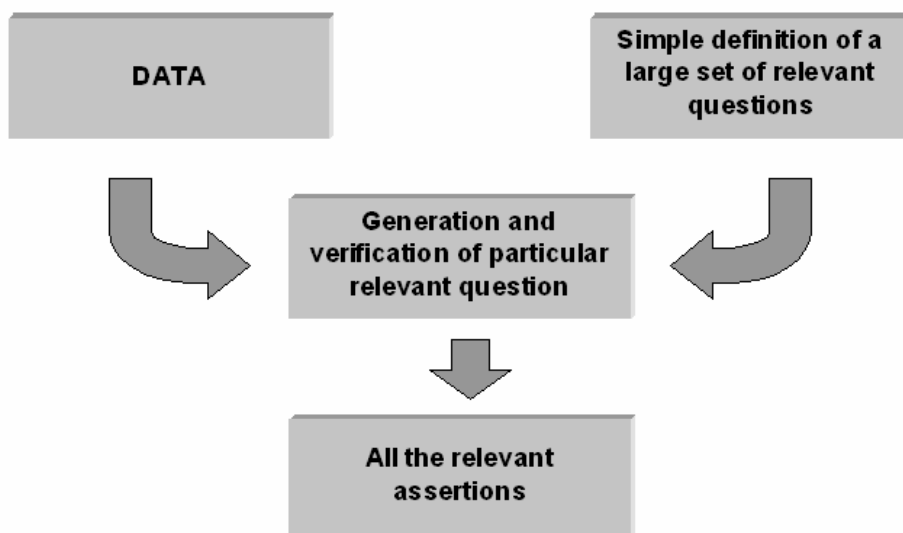
¹ Jedná se často o prezentace, které vysvětlují proces dobývání znalostí, např. http://www.cs.aau.dk/~legind/SpecCourse_Spring2007/SLIDES/Ch4_Bin.pdf, nebo http://www.mineway.de/text_1-1-4_prototyp_projektablauf_e.htm.

² Uváděné procentuální vyjádření je pouze ilustrativní, vychází ze známého Paretova pravidla 80-20 (viz např. http://en.wikipedia.org/wiki/Pareto_principle), a proto zde v součtu jednotlivé fáze přesahují 100%.

1.3. Metoda GUHA a GUHA procedury

Obsah této kapitoly je převzatý z [2]. Odpovídající kapitola v [2] výborně vystihuje základní charakteristiku metody GUHA a proto nebylo třeba jí výrazně upravovat.

GUHA (General Unary Hypotheses Automaton) je metoda pro získávání znalostí z dat. Teoretický rámec pro metodu vznikl v šedesátých letech v Československu a je popsán například v [6]. [7] poskytuje užitečný přehled literatury zabývající se GUHA metodou. Základní princip metody vyjadřuje Obrázek 2.



Obrázek 2: Princip metody GUHA

Primárním cílem metody GUHA je pomocí počítače generovat všechny hypotézy zajímavé na základě vstupních empirických dat. Zajímavost je dána logickým tvarem hypotézy a způsobem, jakým ji podporují data. Na realizaci GUHA metody se používají GUHA procedury. GUHA procedury jsou programy, jejichž vstupem jsou analyzovaná data a schematicky zapsané tvary hypotéz. GUHA procedura automaticky generuje všechny hypotézy a testuje u nich, jestli jsou pravdivé ve vstupních datech. Výstupem procedury jsou pak všechny hypotézy, které jsou pravdivé v datech a nejsou součástí jednodušší hypotézy.

Nejvýznamnější GUHA procedurou je procedura ASSOC [6], která hledá asociační pravidla. Zásadní implementací pro tuto práci je procedura 4FT, která je implementací procedury ASSOC v rámci systému Ferda Data Miner.

1.4. Systém Ferda Data Miner

Systém Ferda byl vytvořen jako softwarový projekt na MFF UK. Systém byl vyvíjen déle než dva roky a v dubnu 2006 byl jako softwarový projekt obhájen. Primárním popudem pro vznik Ferdy byl požadavek na vytvoření uživatelsky příjemnějšího prostředí pro systém LISp-Miner[8]¹. Jedním z hlavních cílů autorů Ferdy bylo dosáhnout toho, aby byl systém snadno modifikovatelný a aby do něj bylo možné jednoduše přidat novou funkčnost. Vznikla tak otevřená platforma pro dobývání znalostí². Základním kamenem této platformy je tzv. *krabičkový model*. Před pokračováním v dalším čtení textu je vhodné nejprve porozumět

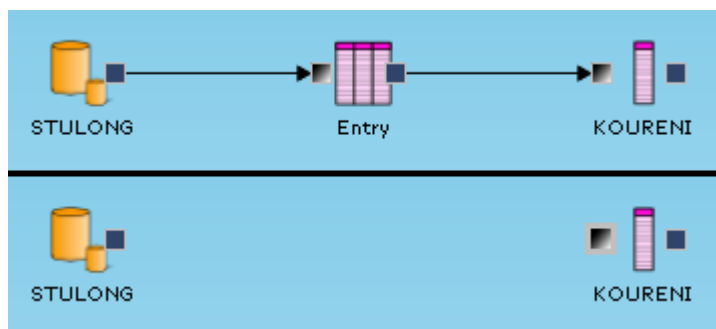
¹ LISp-Miner je akademický softwarový systém určený pro výuku a výzkum v oblasti dobývání znalostí. Tento systém byl vyvinut na VŠE v Praze a jeho vznik se datuje do poloviny devadesátých let 20. století.

² V zásadě může být do Ferdy naprogramováno cokoliv (např. krabička počítající na základě data odpovídající den v týdnu), ale je dodržována jeho tematická čistota, a tedy formální omezení na oblast dobývání znalostí.

terminologii¹ a fungování Ferdy (vhodným zdrojem těchto informací může být např. [1]). Základními stavebními kameny prostředí Ferda jsou jeho moduly, tzv. krabičky, které se na základě svého typu vzájemně propojují, čímž je sestavována samotná DZD úloha.

Krabička ve Ferdovi je základní vizuální prvek tohoto prostředí. V ostatních vizuálních systémech pro dobývání znalostí reprezentuje obvykle vizuální prvek jednu část procesu dobývání znalostí. Ve Ferdovi je tomu jinak. Krabičky ve Ferdovi mají význam funkce, v porovnání s výše zmíněnými vizuálními prvky reprezentuje krabička Ferdy jednu z mnoha funkcí jednoho vizuálního prvku. Pohlédneme-li na to z druhé strany, určitá skupina (správně spojených) krabiček reprezentuje jednu část procesu dobývání znalostí. Tato skupina krabiček (z pohledu procesu dobývání znalostí – funkcí) není na rozdíl od funkcí v klasických vizuálních prvcích pevně daná a umožňuje tak větší variabilitu samotného procesu dobývání znalostí. Tato výhoda však přináší jednu z mála nevýhod systému Ferda: menší přehlednost.

Tato nevýhoda je částečně řešena mechanismem zabalování a rozbalování zásuvek krabiček, který umožňuje skrýt krabičky vstupující do dané zásuvky. Je také možné skrýt jednotlivou krabičku. Po smazání krabičky z plochy se tato krabička zabalí do zásuvky krabičky, do které je zapojena. Vazby skryté krabičky jsou stále funkční, krabička jen není vidět, viz Obrázek 3, který ukazuje schování krabičky *Entry*.



Obrázek 3: Ukázka skrytí krabičky (před a po)

Skrývání krabiček však není ideálním řešením pro zvýšení přehlednosti. Vhodné řešení nastínil Martin Ralbovský v [2] v části diskutující požadavky prozatím hypotetického systému Ever-Miner. Uvažovaným řešením jsou tzv. „nadkrabičky“, tedy krabičky, které v sobě budou obsahovat řetězky základních krabiček. Pro „nadkrabičku“ by mohl uživatel nastavovat vlastnosti pro celou skupinu krabiček a podle nastavení vlastností by se vytvořil konkrétní řetězek krabiček, který by byl v prostředí Ferdy reprezentován jediným vizuálním prvkem. Navíc by bylo možné „vstoupit“ do „nadkrabičky“ a editovat jednotlivé krabičky uchované uvnitř. Měnit jejich vlastnosti, upravovat jejich zapojení či vytvářet nové krabičky. Bylo by ovšem nutné zajistit konzistenci zásuvek „nadkrabičky“. Celé téma „nadkrabiček“ je ještě potřeba důkladně promyslet a mohlo by být vhodným tématem nějaké další diplomové práce či třeba softwarového projektu². V souvislosti s „nadkrabičkami“ je velmi zajímavá diplomová práce Michala Kováče [9], jejíž součástí je implementace jedné takové „nadkrabičky“ – Lambda³.

V rámci softwarového projektu Ferda byla nejprve vytvořena platforma systému (jádro Ferdy) a nad ní byly následně implementovány krabičky pro podporu GUHA dobývání dat. Pro potřeby odlišení je někdy jádro systému označováno jako Ferda a jádro včetně krabiček

¹ Nejdůležitější pojmy používané v této diplomové práci jsou stručně vysvětleny v Dodatku A. Jsou mezi nimi také klíčové pojmy systému Ferda.

² „Nadkrabičky“ jsou jedním z námětů na další práci (kapitola 6.2).

³ Bohužel jádro systému Ferda prozatím neumožňuje zobrazit krabičky obsažené v této „nadkrabičce“.

pro dobývání znalostí je označováno jako Ferda DataMiner. Ovšem často se i pro Ferda DataMiner používá zkrácený pojem Ferda. Je tomu tak i v této diplomové práci.

1.4.1. GUHA procedury implementované ve Ferdovi

V práci [2], na níž tato diplomová práce přímo navazuje, byly popsány jednotlivé GUHA procedury implementované ve Ferdovi, jednalo se o 4FT, KL, CF, SD4FT, SDCF a SDKL.

Stručný popis procedur najdete v [2] i s odkazy na zdroje s detailnějšími teoretickými informacemi o těchto procedurách. Já zde ve stručnosti popíši proceduru 4FT, která poslouží jako ukázková procedura v několika příkladech této práce.

Cílem procedury 4FT (resp. GUHA procedury ASSOC) je nalezení asociačních pravidel tvaru $A \sim S/P$, kde A (antecedent), S (sukcedent) i P (podmínka) jsou booleovské atributy a \sim je 4FT kvantifikátor. Procedura funguje tak, že sestrojí čtyřpolní¹ tabulku obsahující v jednotlivých polích tabulky frekvence atributů splňujících antecedent a sukcedent (pole a), splňující antecedent, ale nesplňující sukcedent (pole b) atd. a tuto tabulku testuje vůči kvantifikátoru \sim za dané podmínky P .

	Sukcedent	\neg Sukcedent	
Antecedent	a	b	r
\neg Antecedent	c	d	s
	k	l	m

Tabulka 1: frekvenční tabulka 4FT procedury

Ve Ferdovi je pro proceduru 4FT implementováno 12 kvantifikátorů. Pro získání lepší představy o funkčnosti procedury 4FT si její aplikaci ukážeme na příkladu (Příklad 2).

¹ Odtud je odvozený název procedury Four Fold Table.

Vycházejme z analytické otázky 13.c. Pro jednoduchost se omezme na jednu její podotázku: „Existuje asociační pravidlo mezi intenzitou kouření a hypertenzí?“

Pro zodpovězení této otázky je možné použít různé kvantifikátory. V tomto příkladu použijme kvantifikátor fundované implikace. Formální zápis fundované implikace vypadá takto:

$$\text{antecedent} \Rightarrow_{p, \text{Base}} \text{sukcedent}: a / (a + b) \geq p \wedge a \geq \text{Base}$$

Fundovaná implikace je definována pro $0 < p \leq 1$ a $\text{Base} > 0$. Parametr *Base* představuje požadavek na počet objektů splňujících zároveň antecedent i sukcedent (tedy na hodnotu a ve čtyřpolní tabulce). Parametr *p* je nazýván spolehlivost (confidence) a je určující pro daný 4ft-quantifikátor. Určuje, kolik procent objektů musí vyhovovat danému 4ft-quantifikátoru, aby asociační pravidlo bylo považováno za platné v datech.

Fundovanou implikaci lze interpretovat např. takto: „Antecedent implikuje sukcedent na úrovni $100 \times p$ procent a zároveň nejméně *Base* objektů splňuje antecedent i sukcedent.“

V našem příkladu je intenzita kouření antecedent, hypertenze je sukcedent. Vytvoříme si úlohu v prostředí Ferdy, atribut intenzity kouření si rozdělíme na dvě kategorie – kuřáci a nekuřáci. Atribut diastolického a systolického tlaku si omezíme dle definice hypertenze a jejich atributy spojíme pomocí disjunkce. Zapojíme antecedent, sukcedent a krabičku fundované implikace (v níž nastavíme hodnotu parametru *p*) do krabičky 4ft-Task. Není-li zapojena krabička *Base*, není ani podmínka $a \geq \text{Base}$ testována (resp. je považována za pravdivou). Po spuštění procedury se vytvoří všechny kombinace kategorií antecedentů a sukcedentů, tzv. hypotéz (v tomto případě jsou pouze čtyři: *kuřák-vysoký systolický tlak*, *kuřák-vysoký diastolický tlak*, *nekuřák-vysoký systolický tlak* a *nekuřák-vysoký diastolický tlak*). Pro každou z těchto hypotéz se na základě dat vyplní čtyřpolní tabulka, procedura 4FT spočítá, zda hodnota *a* vyhovuje podmínce $a \geq \text{Base}$, a pokud ano, spočítá levou stranu nerovnice $a / (a + b) \geq p$.

Mějme antecedent kuřák, sukcedent vysoký systolický tlak, na základě dat z databáze STULONG (pro hodnoty tlaku jsou použity sloupce SYST1 a SYST2) vyjde následující čtyřpolní tabulka.

	Systolický tlak (≥ 140 mm HG)	Systolický tlak (< 140 mm HG)
Kuřák	361	656
Nekuřák ¹⁰	164	236

Levá strana nerovnice, $a / (a + b)$ vyjde zaokrouhleně 0,35. Dle hodnoty *p* pak procedura příslušnou hypotézu vrátí či nevrátí jako platnou hypotézu. Ale dle výsledku výše uvedeného výrazu lze říci, že na základě vstupních dat kouření neimplikuje vysoký systolický tlak.

Příklad 2: ukázka 4FT procedury

¹⁰ Jedná se o doplněk kategorie kuřák, což je nejen kategorie nekuřáků, ale navíc kategorie reprezentující pacienty, kteří neuvědli, jak intenzivně kouří.

V době psaní [2] byly pro běh jednotlivých GUHA procedur používány komponenty systému LISp-Miner, které implementovaly jednotlivé procedury (např. komponenta LISp-Mineru *4ft-Miner* implementuje GUHA proceduru ASSOC¹).

Dnešní verze Ferdy již komponenty LISp-Mineru nepoužívá. Tomáš Kuchař v rámci své diplomové práce [10] vytvořil nezávislé implementace výše uvedených GUHA procedur. Pro své implementace využil principu bitových řetězců a vstupem do jím implementovaných GUHA procedur již nejsou dílčí cedenty (jako tomu bylo u krabiček využívajících komponent LISp-Mineru), ale booleovské atributy². Navíc v rámci diplomové práce vytvořil nové a upravil stávající krabičky pro snadnější práci s atributy.

Velkým přínosem uvedené práce je výrazné rozšíření možností zadání antecedentů a sukcedentů. Jedná se zejména o možnost použití disjunkcí v cedentech. Blíže o možnostech odvozování booleovských atributů pojednává např. [10]. Pro podporu odvozování booleovských atributů byly ve Ferdovi implementovány krabičky disjunkce, konjunkce a negace, jejichž prostřednictvím je možné upravovat příslušný booleovský atribut.

1.5. Ontologie

Ontologie je prostředek, který nám umožňuje zachytit znalosti určité oblasti pro jejich další (mj. strojové) využití.

Ontologie je však velmi široký pojem. Vhodnými zdroji pro pochopení jeho šíře jsou např. [11] a [12]. Výchozí literaturou o ontologiích pro tuto práci, stejně jako pro práci, na níž tato přímo navazuje, je [13]. Použitá česká terminologie vychází z [14] a nejdůležitější pojmy týkající se ontologií jsou vysvětleny v Dodatku A této práce.

Velmi pěkné rozdělení ontologií poskytuje příslušná kapitola práce [15]. V této práci budou nadále pojmem ontologie označovány ontologie doménové³. Předmětem těchto ontologií je vždy určitá specifická věcná oblast (např. oblast medicíny). Ontologie popisuje základní pojmy této oblasti, definuje jednotlivé entity a vztahy mezi nimi. Důležitou charakteristikou ontologií (vycházející z jejich definice⁴) je to, že jsou vyjádřeny ve formálně-logickém jazyce, díky čemuž jsou interpretovatelné počítačem.

Existuje množství jazyků pro zaznamenávání ontologií (podrobněji o jazycích ontologií pojednává kapitola 3), mnoho nástrojů pro jejich tvorbu a taktéž řada aplikací, které ontologie používají. Zároveň existuje velké množství vytvořených ontologií popisujících různé domény.

Ontologie jsou v současné době jednou z hlavních oblastí výzkumu znalostního inženýrství. Zájem o ontologie je způsoben především rozvojem a výzkumem v oblasti sémantického webu [14].

¹ Nadále je v této práci pod pojmem 4FT myšlena zároveň GUHA procedura ASSOC a zároveň její nová implementace v systému Ferdy, pro potřeby této práce není nutné mezi nimi rozlišovat.

² Informace týkající se booleovských atributů jsem čerpal z práce [10] a konzultací s vedoucím diplomové práce.

³ Existují také například ontologie generické, úlohové a aplikační. Viz [15] v kapitole o ontologiích.

⁴ Jedna z definic ontologií zní takto: „Ontologie je formální specifikace sdílené konceptualizace“ s následným vysvětlením:

- formální – vyjádřená ve formálně-logickém jazyce – zpracovatelném počítačem
- sdílené – vytvořena na základě dohody více subjektů
- konceptualizace – abstraktní model určité oblasti – soubor pojmů a vztahů mezi nimi

2. Využití ontologií při aplikacích GUHA procedur v prostředí Ferda

O možnosti využití doménových znalostí při procesu dobývání znalostí pojednávají zejména práce [15], [16], [2] a [17]. Práce [15] je prvním pokusem o zmapování možností využití ontologií při DZD (návrhy jak využít ontologie při dobývání znalostí budeme v návaznosti na [2] nazývat „postupy pro využití ontologií“ a zkráceně je budeme označovat PVO). Jednotlivé PVO jsou ilustrovány na příkladu úlohy zkoumající rizikové faktory aterosklerózy u mužů středního věku (projekt STULONG [3]). Shrnutím výsledků této práce byla [16].

Na tyto práce navazuje [2] a diskutuje jednotlivé PVO. Autor poskytuje nový pohled na využití ontologií: pohled strojové využitelnosti ontologií pro podporu DZD. Vedle přínosných myšlenek k již navrženým PVO přináší autor návrhy nových PVO a předkládá také konkrétní návrhy realizace některých PVO v systému Ferda.

[17] shrnuje myšlenky předchozích prací a ilustruje jednotlivé možnosti využití ontologií při DZD na příkladu dvou praktických dataminingových úloh (jedna z oblasti medicíny, druhá z oblasti sociologie).

V podkapitole 2.2 této práce jsou diskutovány PVO z práce [2] doporučené k implementaci. Této podkapitole předchází přehled možností využití ontologií v jednotlivých fázích procesu DZD s odkazy na jednotlivé PVO. V kapitole 2.3 jsou poté komentovány PVO, které nebyly prací [2] doporučeny k implementaci.

2.1. Využití ontologií v procesu DZD

2.1.1. Porozumění problematice

Při snaze zorientovat se v nějaké oblasti může být ontologie velice platným pomocníkem, jelikož umožní porozumět názvosloví, pochopit význam jednotlivých entit, vztahů mezi nimi, atd. Strojová podpora této fáze DZD je již implementována – stačí použít vhodnou ontologii a nástroj na její čtení (např. Protégé [24]). Uvažovat o podpoře této fáze DZD v systému Ferda nemá velký smysl, jelikož Ferda tuto fázi DZD nepokrývá a ani se o to nesnaží. Nicméně z pohledu využívání ontologií je toto velmi významná oblast a bude možná hlavním důvodem pro vznik ontologií.

2.1.2. Porozumění datům

Ve fázi *porozumění datům* se ontologie stejně jako v předchozí fázi velmi dobře uplatní. V této fázi je cílem definovat data, která budou použita k zodpovězení analytických otázek. Z kvalitní ontologie je teoreticky možné vyčíst úplnou definici potřebných dat včetně jejich datového typu.

Vytvoření návrhu struktury dat by mohlo vypadat tak, že se na základě analytických otázek vyhledají v ontologii třídy odpovídající příslušným skupinám analyzovaných atributů. Tyto vybrané ontologické třídy a jejich podtřídy pak přímo určují, která data budeme pro zodpovězení vstupních analytických otázek potřebovat. Vhodně vytvořená ontologie pak může přímo poskytnout pomocné informace k určení datového typu těchto dat. Např. by mohla být v ontologii uvedena informace o kardinálnosti dané ontologické třídy. Pokud by např. byla třída *nominální*, pak by kandidátem na typ datového sloupce odvozeného z této třídy byl *string*, nebo nějaký *výčtový typ*. Naopak pokud by byla třída *kardinální*, bylo by potřeba zajistit, aby mohlo být s příslušným datovým typem nakládáno jako s kardinálním typem. Tj.

měla by na něm být definovaná relace uspořádání a míra (určující vzdálenost dvou hodnot daného typu). Vhodnými kandidáty datového typu pro atribut tvořený z kardinální třídy jsou především číselné datové typy. Představu přibližuje Příklad 3.

Zajímavým důsledkem využití ontologií v této fázi DZD je možnost postupně vylepšovat a rozšiřovat jednotlivé ontologie. Uvažujme situaci popsanou v předešlém odstavci. Snažíme se k jednotlivým atributům z analytických otázek nalézt odpovídající entity v ontologii. Předpokládejme, že takovou entitu nenalezneme, ale přesto se nám existence takové entity v dané ontologii zdá být objektivně¹ vhodná. Pak je vhodné zvážit, zda takovou entitu do ontologie nepřidat. Takovéto přidání nemusí být vždy jednoduchým krokem, ale u kvalitních a udržovaných ontologií je pravděpodobné, že bude existovat standardní proces na jejich úpravu s cílem jejich neustálého vylepšování.

Uvažujme opět náš ukázkový příklad s analytickou otázkou STULONG 13.c. Představme si, že jsme v situaci, kdy máme zatím jen sestavené analytické otázky, nemáme ještě posbíraná data a chceme zjistit, která data budeme potřebovat. Použijeme proto některou z lékařských ontologií, které pokrývají naši problematiku. Pro jednoduchost použijeme ontologii UMLS2².

Na základě naší analytické otázky víme, že budeme potřebovat určit, zda je pacient hypertonik či nikoliv. K tomu slouží hodnoty krevního tlaku, který se ovšem skládá ze dvou složek tlaku systolického a tlaku diastolického.

Vezměme si nyní např. atribut systolický tlak. Pokusme se ho najít v ontologii, abychom zjistili povahu hodnot tohoto atributu. V ontologii však entita pro systolický tlak neexistuje. Významově nejbližší je jí třída *blood pressure*, vyjadřující krevní tlak. Rozdělení krevního tlaku na systolický a diastolický je však obecně uznávanou skutečností a tyto pojmy jsou v medicíně běžně používány. Jako takové logicky patří do ontologie a jejich přidání do ontologie takovou ontologii jen zpřesní a zkvalitní³.

Na základě analytické otázky dále víme, že budeme chtít zjišťovat vliv kouření. K tomu budeme potřebovat nějaká data, která budou parametry kouření vyjadřovat. Použijeme opět danou ontologii a pokusíme se najít entity, které by vyjadřovaly kouření. Našli jsme třídu *Smoking*. Uvažujme dále, že je ontologie mnohem podrobnější a obsahuje pro třídu *Smoking* ještě další podtřídy – například: *Smoking_intensity* a *Time_of_smoking* reprezentující intenzitu kouření a dobu, jak dlouho příslušná osoba kouří. Všechny podtřídy třídy *Smoking* jsou pak vhodnými kandidáty na sloupce do vytvářené databáze. Rozhodně je vhodné se zamyslet, zda by se nám tyto informace v rámci DZD projektu nehodily.

Pokud je shledáme užitečnými, podíváme se na kardinalitu uvedené třídy. Jestliže je kardinalita *kardinální* nebo *ordinální*, je vhodné pro reprezentaci dat zvolit takový datový typ, který by implicitně vyjadřoval uspořádání daných hodnot. Např. vím, že *Smoking_intensity* je *ordinální*, pak je to příznak pro to, aby se v databázi použila nějaká číselná reprezentace tohoto parametru a nikoliv např. slovní vyjádření hodnot „*silný kuřák*“, „*slabý kuřák*“, „*nekuřák*“.

Příklad 3: Obohacení ontologie ve fázi DZD porozumění datům

¹ Slovem „objektivně“ je myšleno to, že by danou entitu nejspíše uvítali i další lidé, používající tuto ontologii

² Tato ontologie byla vytvořena v rámci práce [15] a je součástí této práce jako Příloha 2

³ Součástí této práce je také experimentální ověření funkčnosti implementovaných modulů v systému Ferda. Pro ukázání všech možností těchto modulů byla z ontologie UMLS2 vytvořena ontologie UMLS_Ferda, která mj. obsahuje navíc právě zde popsané rozdělení krevního tlaku na systolický a diastolický. Více o této ontologii a samotné experimentální části pojednává kapitola 5

Ferda je z hlediska procesu DZD používán až k sestavování úlohy a *modelování* (což je pozdější fáze procesu DZD), ale proces *porozumění datům* lze ve Ferdovi chápat také jinak. Představme si reálnou situaci, kdy by byl Ferda používán jako nástroj pro výuku dobývání znalostí (v současné době je pro výuku používán nadále systém LISp-Miner)¹. Úlohou takového nástroje je studenty seznámit se sestavováním DZD úlohy a modelováním. Vzhledem k obtížnosti fáze sběru a čištění dat je pro výuku většinou používána již připravená databáze (na VŠE v Praze se používají především medicínské databáze STULONG[3] a ADAMEK[18]). Z pohledu DZD je tak projekt dobývání znalostí připraven pro fázi dobývání znalostí. Student však před samotným sestavováním úlohy a modelováním musí proces DZD jakoby dohnat, jelikož bez porozumění datům je nemožné provádět smysluplný datamining. Budeme-li pohlížet na fázi porozumění datům z pohledu studenta, pak se nabízí podpora tohoto procesu ve formě mapování sloupečků z databáze na entity z ontologie. Návrh jak implementovat tuto podporu byl navržen v [2] a je dále rozebrán v kapitole 2.2.1 Mapování.

2.1.3. Příprava dat

Fáze *přípravy dat* obnáší sběr dat a jejich čištění. Jedná se o časově nejnáročnější fázi procesu DZD, možnosti využití ontologií pro tuto fázi jsou mizivé. Je to dáno především formou této fáze, která z velké části sestává z rutinního sběru a čištění dat, při němž nám ontologie nijak nepomůže.

2.1.4. Modelování

Fázi *modelování* lze interpretovat jako sestavení úlohy a spuštění příslušných analytických procedur. Tato fáze je hlavní fází DZD, kterou Ferda podporuje. Sestavení úlohy lze rozdělit na dvě části konstrukci atributů (tzn. rozdělení hodnot domén sloupců zkoumané datové matice do kategorií) a konstrukci úlohy (volba analytické procedury, kvantifikátoru a vytvoření cedentů z atributů)². Toto rozdělení vychází z předpokladu, že kategorizace atributů bývá pokaždé stejná a vychází z povahy dat. Je tedy možné, aby kategorizace atributu byla provedena pouze jednou v rámci celého projektu DZD.

Navíc jestliže povaha dat přímo určuje kategorizaci atributu a systém bude znát povahu dat, pak může atribut kategorizovat automaticky. Jednou z prvotních úvah jak zjistit povahu dat byla snaha určit sémantiku na základě jejich datového typu hodnot. Tak by například číselné hodnoty evokovaly *kardinální* atribut. Ovšem na jednoduchém příkladu lze ukázat, že závislost mezi datovým typem a sémantikou není jednoznačná. Uvažujme datovou tabulku se sloupcem uchovávající hodnoty poštovního směrovacího čísla. *PSC* je reprezentováno číslem, může být v databázi uloženo například jako *integer*, ale porovnání dvou hodnot *PSC* nemá žádný skutečný smysl. *PSC* tedy není *kardinální*, jak by se z datového typu dalo odvodit, ale *nominální*. Databáze tedy není zdrojem potřebných informací.

Proto se začalo uvažovat o využití jiných prostředků pro získání těchto informací a nejperspektivnějším prostředkem se zdají být právě ontologie, kde by informace o sémantice příslušné entity mohla být uložena.

Automatickou tvorbou atributů se blíže zabývá kapitola 2.2.3, kde je diskutován jeden z postupů na využití ontologií z práce [2] a to PVO Automatická tvorba atributů.

¹ Uvažovaná situace byla předestřena jen jako příklad, který má nejbližší k realitě a na kterém bude nejlépe vidět přínos podpory fáze *porozumění datům*. Obdobně lze uvažovat i o hypotetické situaci z komerční praxe, kde by v pozici studenta mohl být datový analytik, jež se poprvé setkává s databází, nad níž má provádět datamining.

² Navazují na úvahy z [2] z části pojednávající o systému Ever-Miner, v níž autor diskutuje možnosti použití Ferdy jako implementace teoretického rámce Ever-Miner.

Dalším navrženým PVO, který patří do této fáze DZD, je konstrukce dílčích cedentů. Po vytvoření nových implementací GUHA procedur ve Ferdovi se název i návrh tohoto PVO musel výrazně upravit. Podrobněji je tato úprava spolu s ukázkovým příkladem diskutována v kapitole 2.2.4.

Práce [15] navrhovala dále postup využití ontologií pro sestavení úlohy DZD. Navazující práce [2] sice tento PVO, zejména pro jeho nejasný přínos, označila za nevhodný pro implementaci ve Ferdovi, ale v kapitole 2.3, která stručně komentuje tento i další nedoporučené PVO, jsem myslím narazil na zajímavou myšlenku, která by mohla vést k nové perspektivě tohoto PVO.

2.1.5. Interpretace výsledků

Dle [17] je možné hledat zdůvodnění výsledků běhů GUHA procedur v ontologiích. Bohužel předpoklady autorů nejsou zcela správné. Autoři předpokládají, že jsou v ontologiích uloženy informace, které tam ovšem z povahy ontologií nepatří. Stejný předpoklad byl uvažován také v [15] při návrhu PVO Tvorba úloh pomocí ontologií. V kapitole 2.3.3, kde je tento PVO podrobněji rozebrán, je mj. zdůvodněno mé tvrzení o nesprávnosti výše uvedeného předpokladu a zároveň je tam nastíněna možnost (která si ovšem vyžaduje další bádání), že by se tento předpoklad mohl upravit a z něho vyplývající postupy využití ontologií by mohly být implementovatelné.

Jiným využitím ontologií v této fázi by mohlo být využití ontologie jako slovníku pro překlad názvů sloupců tabulek do pojmů ontologie. Tento způsob využití je velmi blízký využití ontologií ve fázi porozumění datům. Jestliže již rozumíme datům, teoreticky už dokážeme interpretovat výsledky bez dalšího použití ontologií. Význam použití ontologií v této fázi by však významně vzrostl, pokud by se podařilo proces DZD automatizovat natolik, že by úkol sestavit úlohu DZD přešel z analytika na stroj (o čemž je uvažováno v návrhu prozatím hypotetického systému Ever-Miner – viz příslušná kapitola v [2] nebo např. [19] či [20]).

2.1.6. Využití výsledků

Výstupem každého DZD projektu by mělo být nějaké využití výsledků, nebo aspoň nějaká výstupní zpráva. Zde jsou možnosti využití ontologií obdobné jako v předchozí fázi. V tomto kontextu je třeba zmínit nástroj LISp-Miner Report Assistant [21], vyvinutý na MFF UK, který slouží k automatizovanému vytváření výstupních zpráv dataminingových projektů. Tento nástroj však zatím automatizované využití ontologií nepodporuje.

2.2. Postupy využití ontologií (PVO) ve Ferdovi navržené v [2]

V této kapitole jsou popsány postupy využití ontologií, které byly navrženy v předchozích pracích (zejména [15] a [2]). U jednotlivých PVO je napsáno zdůvodnění, proč jsou (resp. nejsou) vhodné k implementaci. Pokud jsou shledány jako vhodné, pak jsou u nich specifikovány požadavky na implementaci. Implementace navržených krabiček je poté popsána v kapitole 4.

2.2.1. Mapování

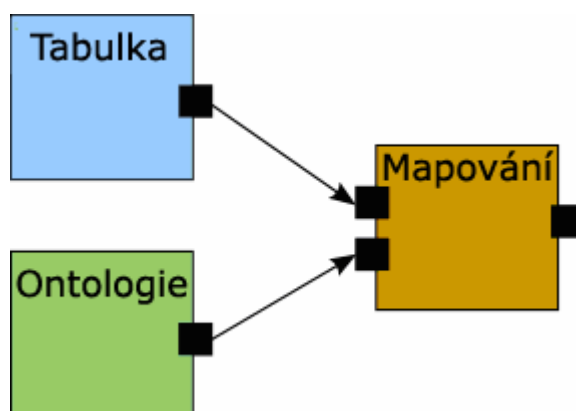
Tento PVO je navržen pro podporu fáze DZD porozumění datům. Jeho cílem je umožnit uživatelům Ferdy snadno pochopit význam jednotlivých sloupců datového zdroje. PVO vychází z představy, že nad jedním, kvalitním datovým zdrojem bádá větší skupina lidí, z nichž někteří nemají předchozí znalost problematiky a neznají význam jednotlivých dat.

Reálnost takové představy vychází z uvažovaného použití Ferdovi při výuce DZD, viz úvaha v kapitole 2.1.2.

Pokud by měl být tento PVO jediným využitím ontologií ve Ferdovi, nemělo by smysl ho implementovat. Podpora, kterou tento PVO přináší uživateli, by se dala stejně dobře zajistit jednoduchým slovníkem, kde by byl vysvětlen význam jednotlivých sloupců.

Smysluplnost implementaci tohoto PVO dodává teprve jeho využití dalšími PVO, zejména PVO Automatická tvorba atributů (kapitola 2.2.3) a PVO Vytvoření a využití skupin příbuzných atributů¹ za pomoci ontologie (kapitola 2.2.4). Pro ně je existence mapování datových sloupců na entity ontologie nezbytnou podmínkou. Primárním cílem tohoto PVO je tedy umožnit využití ontologií ve Ferdovi.

Původní krabičkový návrh tohoto PVO dle [2] zobrazuje Obrázek 4. Tento návrh počítal s vytvořením nových krabiček *Ontologie* a *Mapování* a využití již existující krabičky *Tabulka*.



Obrázek 4: Návrh zapojení krabičky *Mapování* dle [2]²

Krabička *Ontologie* má mít obdobnou funkcionalitu jako krabička *Databáze*. Krabička *Databáze* slouží k připojení se k datovému zdroji a získání informací o jednotlivých tabulkách tohoto zdroje. Obdobně by krabička *Ontologie* měla sloužit k získání veškerých informací z ontologie. Krabička *Ontologie* by měla umožňovat připojit se k ontologii na lokálním disku či na webovém serveru, načíst informace z ontologie a poskytnout tyto informace dalším krabičkám ve Ferdovi.

Vstupem pro krabičku *Ontologie* je cesta k souboru s ontologií. Výstupem krabičky by poté měla být data o ontologii ve vhodné datové struktuře³. Klíčovou otázkou, kterou je třeba zodpovědět před implementací krabičky *Ontologie* je volba jazyka pro reprezentaci ontologie. O této volbě podrobně pojednává kapitola 3.

Úkolem krabičky *Mapování* je navázání sloupců z databáze na entity z ontologie. Těmito entitami mohou být jednak třídy ontologie a jednak instance těchto tříd. Návrh v [2] předpokládá pro krabičku *Mapování* existenci dvou zásuvek. První pro *Tabulku*, do níž může být zapojena právě jedna instance krabičky typu *Tabulka*. Druhá zásuvka je určena pro krabičku *Ontologie* a dle původního návrhu tato zásuvka umožňuje zapojení více krabiček typu *Ontologie*. Při implementaci krabičky došlo posléze k omezení této zásuvky tak, že

¹ Tento PVO byl v předchozích pracích označován jako Konstrukce dílčích cedentů za pomoci ontologie, ale v důsledku implementace nových GUHA procedur ve Ferdovi (viz kapitola 1.4.1), jejichž vstupem jsou místo dílčích cedentů booleovské atributy, ztrácí původní název PVO smysl.

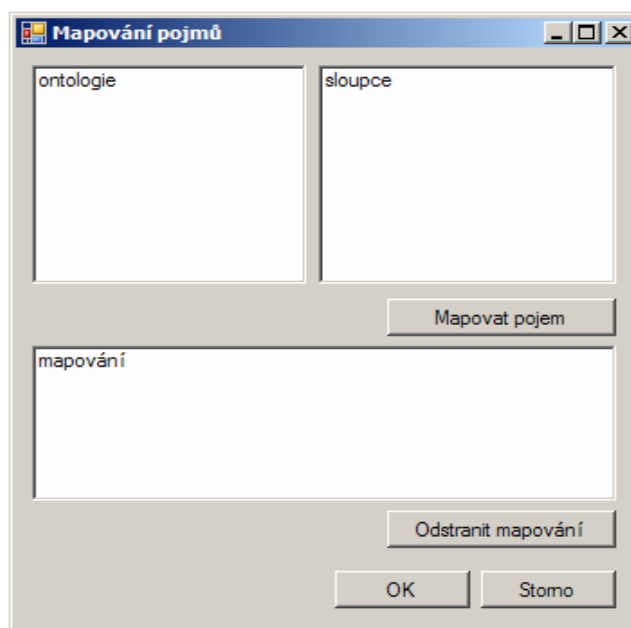
² Tento obrázek je převzat z [2], ale jsou v něm prohozeny vstupní zásuvky krabičky *Mapování* – myslím, že to takto lépe znázorňuje skutečnost, že jsou sloupce z tabulky mapovány na entity ontologie

³ Struktura musí být vhodná zejména pro middleware ICE [22], který Ferda používá.

umožňuje zapojení pouze jedné krabičky *Ontologie*. Více o implementaci krabičky *Mapování* je v kapitole 4.2.

K vytvoření mapování slouží dle návrhu [2] *modul pro nastavení*. Informace o tom, co je to *modul pro nastavení* v prostředí Ferdy, je v [1] nebo v dokumentaci k systému na [23]. Důležitým požadavkem na toto mapování je možnost jeho opakovaného využití. Proto je nutné expertem provedené mapování někde zaznamenat, aby se příště nemuselo provádět znovu. V původním návrhu bylo uložení a načtení mapování umístěno do krabičky jako *akce krabičky*, ale při implementaci byla tato funkčnost přesunuta přímo do *modulu pro nastavení*.

Práce [2] rovnou navrhla jednoduchý dialog pro mapování (viz Obrázek 5, převzatý z [2]). Tento dialog však uvažuje pouze mapování jednoho sloupce tabulky na jednu entitu ontologie (mapování 1:1). V další části se poté autor zamýšlí nad tím, jak mapovat vztahy 1:N či dokonce M:N.



Obrázek 5: Dialog navržený v [2] pro mapování pojmů ontologie a sloupců datového zdroje

Podívejme se na tyto vztahy z jiného úhlu pohledu. K čemu má sloužit ono mapování? Jak bylo zmíněno v úvodu této podkapitoly, PVO Mapování nemá sám o sobě přílišný smysl. Smysl mu dodává až implementace dalších PVO jako Automatická konstrukce atributů nebo Vytvoření a využití skupiny příbuzných atributů. Detailní popis těchto PVO je v kapitolách 2.2.3 a 2.2.4. Zde popíši tyto PVO pouze ve zkratce a s ohledem na to, jaké informace tyto PVO, resp. odpovídající krabičky potřebují z krabičky *Mapování* získat.

Cílem automatické tvorby atributu je vytvořit z jednoho sloupce tabulky atribut, k čemuž nám mají posloužit informace o mapování. PVO Automatická tvorba atributu se dotáže krabičky *Mapování*, jaké informace z ontologie přísluší danému sloupci, z něhož chceme vytvořit atribut.

Cílem druhého PVO je identifikovat skupiny významově příbuzných sloupců. K identifikaci příbuznosti opět poslouží informace z ontologie a to konkrétně příslušnost k třídě ontologie (příbuzné pak budou sloupce, které přísluší ke stejné třídě, nebo jsou potomkem společné třídy). Informace o příslušnosti sloupce k třídě ontologie poskytne opět krabička *Mapování*.

Zejména z požadavků prvního PVO vyplývá, že mapování by měla být funkce, jejíž doménou jsou sloupce tabulek databáze a rozsahem entity ontologie. Na základě tohoto požadavku můžeme přestat uvažovat o mapování M:N (a M:1).

Vztah 1:N naopak není v rozporu s uvedenými požadavky a můžeme si názorně na příkladu ukázat, že mapování 1:N má smysl. A v praxi bude tento vztah poměrně běžný.

Uvažujme náš vzorový příklad s databází STULONG a analytickou otázkou 13.c. Pro jednoduchost se omezme na jednu její podotázku: „*Existuje v datech vtaž mezi kouřením a výškou systolického tlaku?*“

V databázi STULONG, v tabulce *Entry* existují tři sloupce (*KOURENI*, *DOBAKOUR*, *BYVKURAK*), patřící do skupiny atributů kouření. Zároveň v této tabulce máme dva sloupce odpovídající naměřenému systolickému tlaku (reprezentují dvě různá měření *SYST1* a *SYST2*).

Uvažujme dále, že máme vhodnou ontologii, na níž můžeme provést mapování. Pro názornost můžeme použít ontologii UMLS_2¹ vytvořenou v rámci práce [15]. Následující odrážky znázorňují umístění entit *Smoking* a *Blood_pressure* v ontologii.

- *Activity*
 - *Individual_Behavior*
 - *Smoking*
- *Finding*
 - *Laboratory_or_Test_Result*
 - *Blood_pressure*

Chtějme nyní namapovat sloupec *DOBAKOUR* na entitu z ontologie. Význam tohoto sloupce odpovídá bezpochyby třídě *Smoking*. Jestliže by třída *Smoking* měla podtřídu *Time_of_Smoking*, pak by byl sloupec *DOBAKOUR* namapován na tuto třídu. Taková třída ale v ontologii není, a tak nám nezbyvá než sloupec *DOBAKOUR* namapovat na nejbližší příbuznou entitu, kterou je právě *Smoking*. Ze stejného důvodu budou na tuto třídu namapovány i sloupce *KOURENI* a *BYVKURAK*.

Toto mapování významově odlišných sloupců na jeden termín z ontologie by mohlo být vyřešeno také doplněním příslušných podtříd do ontologie, ale toto řešení nemusí být vždy snadné a často není žádoucí. Například při použití nějaké ontologie z webového serveru k ní například analytik provádějící mapování nemusí mít vůbec přístup pro zápis. To byl důvod, proč byl v [2] vnesen na PVO Mapování požadavek, že se při mapování nesmí modifikovat ani ontologie, ani datový zdroj.

Na druhou stranu pokud se při mapování zjistí, že by bylo vhodné nějakou třídu do ontologie přidat, může tato skutečnost iniciovat proces změny ontologie s cílem jejího vylepšení.

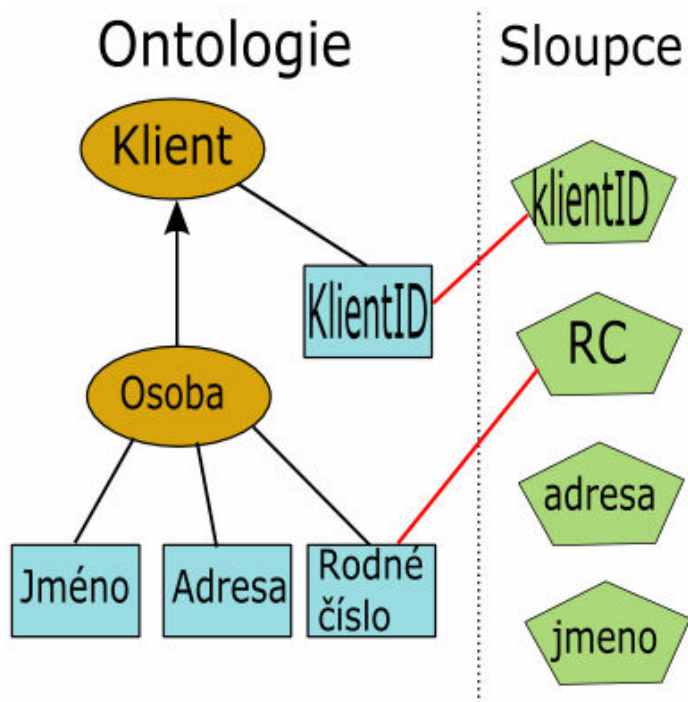
V případě sloupců kouření by se úpravou ontologie mohlo dosáhnout vazeb 1:1. Ovšem na příkladu mapování sloupců *SYST1* a *SYST2* je vidět, že mapování 1:N je zcela běžné. Oba sloupce mají stejný význam, pouze obsahují různé hodnoty. A pro přiřazení správné sémantiky těmto sloupcům je potřeba je oba namapovat na stejnou entitu ontologie. V případě ontologie UMLS_2 budou namapovány na třídu *Blood_pressure*².

Příklad 4: Mapování datových sloupců na entity ontologie vazbou 1:N

¹ Jedná se o výsek zdravotnické ontologie UMLS pokrývající entity použité v projektu STULONG. Tuto ontologii vytvořila autorka [15] v aplikaci Protégé [24]. Tato ontologie je Přílohou 2 této diplomové práce.

² V ontologii UMLS_Ferda vytvořené pro testovací účely této práce je třída *Blood_pressure* dále rozdělena na *Systolic blood pressure* a *Diastolic blood pressure*. Toto členění by však nic nezměnilo na tom, že oba sloupce *SYST1* i *SYST2* budou namapovány na stejnou entitu ontologie.

Práce [2] dále uvažuje o možnosti vizualizace mapování. Vizualizaci ilustruje Obrázek 6, převzatý z [2]. Přínosem vizualizace oproti výše navrženému dialogu je možnost zobrazit také další vazby mezi entitami ontologie, ale uvážíme-li počet entit a vazeb v běžné ontologii a počet sloupců, který se může vyskytovat v jedné tabulce, pak se dá snadno odhadnout, že by grafické mapování bylo spíše na obtíž a znemožnilo by efektivní mapování. Bylo tedy rozhodnuto, že pro mapování bude použita modifikace výše navrženého dialogu. Jak tento dialog vypadá a jak funguje je popsáno v kapitole 4.2, pojednávající o implementaci krabičky *Mapování*.



Obrázek 6: Vizualizace mapování. Ovály a obdélníky reprezentují entity ontologie, pětiúhelníky označují sloupce, šipka je dědičnost, červená čára značí již namapované vztahy.

Další zajímavou možností, kterou autor v [2] diskutuje, je vytvoření expertního systému pro automatické mapování datových sloupců a entit ontologií. Cílem tohoto expertního systému by bylo umožnit mapování i v situaci, kdy jsou datový zdroj nebo ontologie natolik rozsáhlé, že znemožňují ruční mapování. Vzhledem ke vstupním datům, které by měl tento systém k dispozici, by pracoval na principu porovnávání názvů sloupců tabulek a názvů entit ontologie (teoreticky by se daly využít například také komentáře ke sloupcům z databáze, ale obecně Ferda umožňuje napojení různých datových zdrojů, z nichž řada, např. soubory .xls, žádné komentáře ke sloupcům nemá). Již první úvahy o tomto systému odhalují obtížně řešitelné problémy.

Klíčovým problémem je nekonzistence mezi názvy sloupců v databázi a názvy entit v ontologii. Sloupce datové tabulky bývají často zkratkou nějakých slov, z níž je velice obtížné odhadnout význam daného sloupce nejen pro stroj, ale mnohdy i pro člověka. Dalším problémem může být odlišný jazyk použitý v ontologii a databázi (viz např. anglická ontologie UMLS_2 a česká databáze STULONG). Je zřejmé, že nelze zaručit 100% účinnost daného expertního systému. Vždy bude muset jeho výsledek zkontrolovat analytik či doménový expert a případné nesrovnalosti opravit a namapovat sloupce, které systém nedokázal jednoznačně přiřadit pojmu z ontologie.

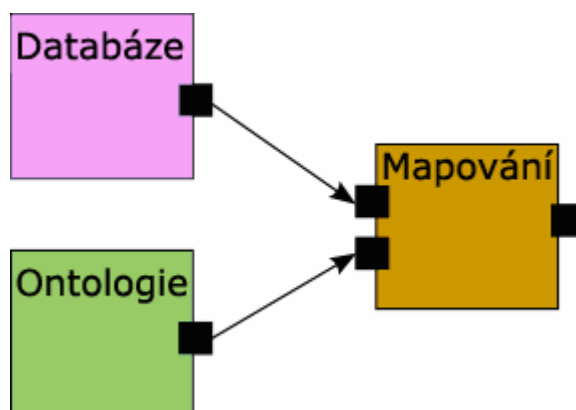
Expertní systém by mohl fungovat s velikou úspěšností v případě, že by byla daná ontologie použita již při návrhu databáze a že bylo k ní bylo přihlédnuto při volbě názvů sloupců. Ovšem taková situace je prozatím pouze hypotetická a dnešní době velmi vzdálená.

Větší efektivity než při porovnávání názvů sloupců a entit by expertní systém mohl dosáhnout při zkoušení dříve provedených mapování. Systém by čerpal vzorová mapování z nějaké databáze uložených mapování a pokoušel se v nich najít odpovídající názvy sloupců a entit, které již někdy někdo namapoval a které se zároveň vyskytují v databázi a ontologii, kterou právě používá analytik. Ovšem aby byla tato metoda efektivní pro obecné datové zdroje, musela by být uvažovaná databáze mapování velice obsáhlá. Naplnění takové databáze je nereálné. Ovšem myšlenka využití mapování, které nutně nemuselo být provedeno pro danou databázi a konkrétní ontologii byla při implementaci mapování ve Ferdovi použita (viz kapitola 4.2, část popisující načtení uloženého mapování).

Myšlenka expertního systému vychází z předpokladu, že mapování rozsáhlého datového zdroje nebo rozsáhlé ontologie je pro analytika nepřehledné ba nemožné. Podle mého názoru je však tento předpoklad chybný. Je-li ontologie vhodně navržena a strukturována, pak je možné se v ní pohodlně orientovat bez ohledu na její velikost. A pokud se analytik dobře orientuje v ontologii, pak je namapování sloupců tabulky pouze rutinní záležitostí. Tato rutina může být pochopitelně zdlouhavá, ale i při existenci expertního systému by se analytik musel věnovat ověření správnosti mapování u každého sloupce zvlášť. Navíc se lze v případě tabulky s velkým množstvím sloupců zaměřit pouze na sloupce, které potřebujeme k zodpovězení analytických otázek, tzn. není třeba mapovat všechny sloupce, ale jen ty, které potřebujeme pro další modelování.

S ohledem na stávající rozsah použití systému Ferda, však nemá smysl zvažovat implementaci expertního systému, jelikož prostor pro jeho využití je zatím v podstatě nulový. Bude-li Ferda používán, pak především pro výuku, kde jak už bylo zmíněno, je využíváno malé množství testovacích databází a jedno mapování tak může být využito opakovaně mnohokrát.

Původní návrh zapojení krabičky *Mapování* byl na základě konzultací s vedoucím diplomové práce upraven tak, že do zásuvky krabičky *Mapování* bude zapojena nikoliv krabička *Tabulka*, ale přímo krabička *Databáze*, jejímž výstupem jsou informace o celém datovém zdroji, resp. o všech tabulkách tohoto zdroje (viz Obrázek 7). Velkou výhodou tohoto nového návrhu je to, že umožňuje uživateli provést mapování na jednom místě pro celý DZD projekt.



Obrázek 7: Nový návrh zapojení krabičky *Mapování*

Tato změna návrhu způsobila posun role krabičky *Mapování*. Tento posun je blíže rozebrán v kapitole 2.2.3, týká se totiž krabiček nabízených na vytvoření, což ovšem spadá do

PVO Automatická tvorba atributů. Z pohledu PVO Mapování však funkčnost nových krabiček zůstává stejná.

2.2.2. Využití ontologie pro identifikaci chybějících atributů (resp. sloupců)¹

Cílem tohoto PVO je umožnit analytikovi objevit nedostatky jím použitého datového zdroje. Neboli identifikovat sloupce, které by z pohledu DZD mohly být užitečné, ale v datovém zdroji se nevyskytují. Otázkou je, k čemu analytikovi tato informace bude. Předchozí práce [15], [2] řadí tento PVO do fáze *Porozumění datům*. Já souhlasím, ale pouze nahlížíme-li na fázi porozumění datům z pohledu např. studenta (viz úvaha v kapitole 2.1.2), jinak se dle mého názoru jedná již o fázi *Modelování* (viz kapitola 2.1.4). Důsledek tohoto fázového posunu je zřejmý – analytik sice může získat informaci o tom, jaké sloupce by se mu ještě v datovém zdroji hodily, ale v tomto okamžiku je již většinou pozdě je získat. Tento PVO by měl vždy předcházet návrhu datového zdroje a sběru dat. I zde je možné (a vhodné) využít podpory ontologie se skutečným přínosem – podrobněji je toto využití rozebráno v kapitole 2.1.2). Z výše popsaného důvodu prozatím nebyla podpora identifikace chybějících atributů implementována, ačkoliv by to z technického hlediska nebyl žádný problém. Tato funkčnost by byla řešena nejspíše jednoduchým *modulem pro interakci* v krabičce *Mapování*, který by porovnal všechny entity ontologie s namapovanými entitami a vypsál by všechny entity, které nejsou namapované.

Nicméně [2] u tohoto PVO navrhuje ještě jinou funkčnost: seskupení příbuzných sloupců. Tato funkčnost může být užitečná pro samotné sestavování úlohy, a proto bylo rozhodnuto o její podpoře ve Ferdovi, ale nikoliv prostřednictvím speciálně navrhované krabičky, nýbrž v rámci krabičky *Mapování* (detailně je tato záležitost popsána v kapitole 4.2, která pojednává o implementaci krabičky *Mapování*).

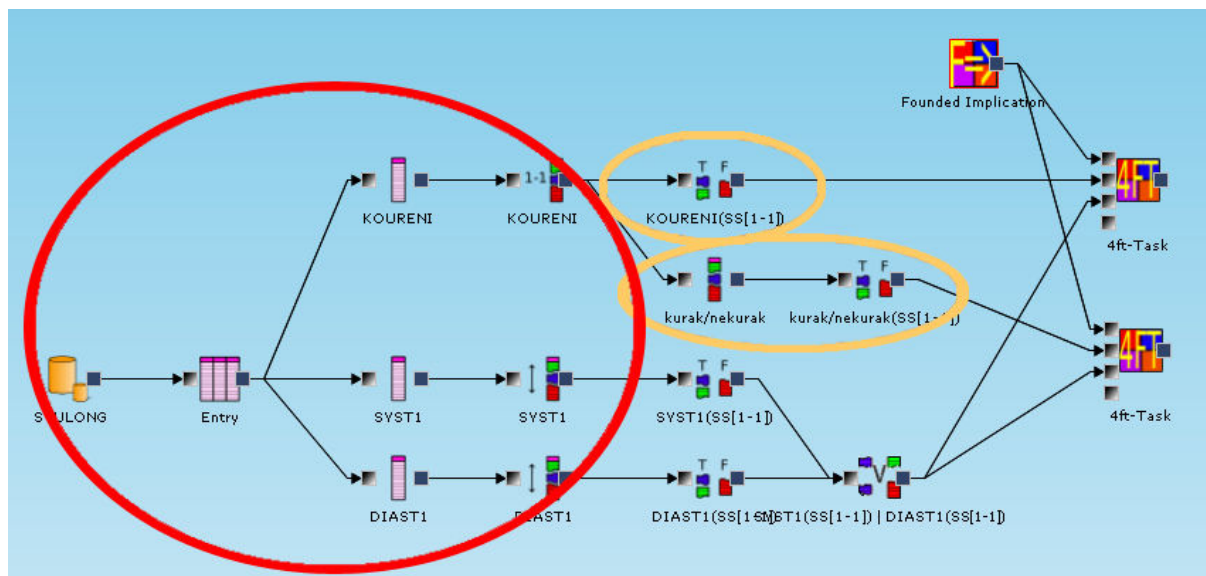
2.2.3. Automatická tvorba atributů

V kapitole 2.1.4 bylo napsáno, že sestavení úlohy DZD lze rozdělit na dvě části. Konstrukci atributů (tzn. rozdělení hodnot domén sloupců zkoumané datové matice do kategorií) a samotnou konstrukci úlohy (volba analytické procedury, kvantifikátoru a vytvoření cedentů z atributů)². Bylo také zmíněno, že toto rozdělení vychází z předpokladu, že kategorizace atributů bývá pokaždé stejná a vychází přímo z povahy dat. A budeme-li znát povahu dat, pak je možné uvažovat o automatické tvorbě atributů, což je zároveň cílem tohoto PVO.

Obrázek 8 názorně ukazuje, která část úlohy ve Ferdovi spadá do fáze tvorby atributu. Červený ovál reprezentuje část, která bude vždy stejná. Jedná se o krabičky *Databáze*, *Tabulky*, *Sloupečku* a *Atributu*. Ve žlutých oválech je poté ukázka toho, jak se pro různé úlohy využívá tohoto stejného základu. Na obrázku vidíme dvě velmi podobné úlohy, které zjišťují závislost hypertenze na intenzitě kouření. První úloha zkoumá vliv množství cigaret vykouřených za den (každá hodnota, reprezentující v případě databáze STULONG a sloupce *KOURENI* nějaký interval vykouřených cigaret, má v atributu svou kategorii). Druhá úloha pak sjednocuje všechny kuřáky do jedné kategorie a zkoumá vliv kouření (bez ohledu na jeho intenzitu) na hypertenzi.

¹ Logicky správný název tohoto PVO by měl znít Identifikace chybějících sloupců, ale v předchozích pracích o využití ontologií byl pojmenován jako Identifikace chybějících atributů. A proto je nadále používán tento mírně nepřesný název.

² Navazují na úvahy z [2] z části pojednávající o systému Ever-Miner, v níž autor diskutuje možnosti použití Ferdy jako implementace teoretického rámce Ever-Miner.



Obrázek 8: Kategorizace atributu a příprava cedentů

Obrázek však není úplně přesný, již v krabičce atributu často dochází k úpravě kategorizace pro různé úlohy a úprava parametrů této krabičky (resp. změna kategorizace) je součástí samotného hledání znalostí. Ukažme si to na příkladu.

Uvažujme dvě odlišné situace.

První vychází přímo z databáze STULONG. Ve sloupečku *KOURENI* jsou uloženy hodnoty, které reprezentují interval intenzity kouření. Např. hodnota 3 znamená, že pacient vykouří 5–14 cigaret za den. Rozhodnutí o jednotlivých intervalech bylo provedeno již před naplněním databáze (resp. již před sběrem dat), čímž byla znemožněna možnost rekategorizace atributu (nelze např. vytvořit kategorii 5–10 cigaret denně). Ačkoliv ze znalosti příslušných intervalů a jim odpovídajících hodnot ve sloupci lze říci, že čím vyšší je hodnota, tím je daný pacient silnější kuřák, není to zcela pravda (např. hodnota 6 reprezentuje kuřáky doutníků dýmek a hodnota 13 je vyhrazena pro pacienty, kteří neudalí, zda kouří či ne). Nemá tedy velký smysl jednotlivé hodnoty porovnávat, a tedy hodnoty tohoto sloupce jsou *nominální*. *Nominální* sloupce má smysl rozdělit pouze na kategorie kde každá hodnota bude odpovídat jedné kategorii (ve Ferdovi se k tomu využívá krabičky *Each Value One Category*).

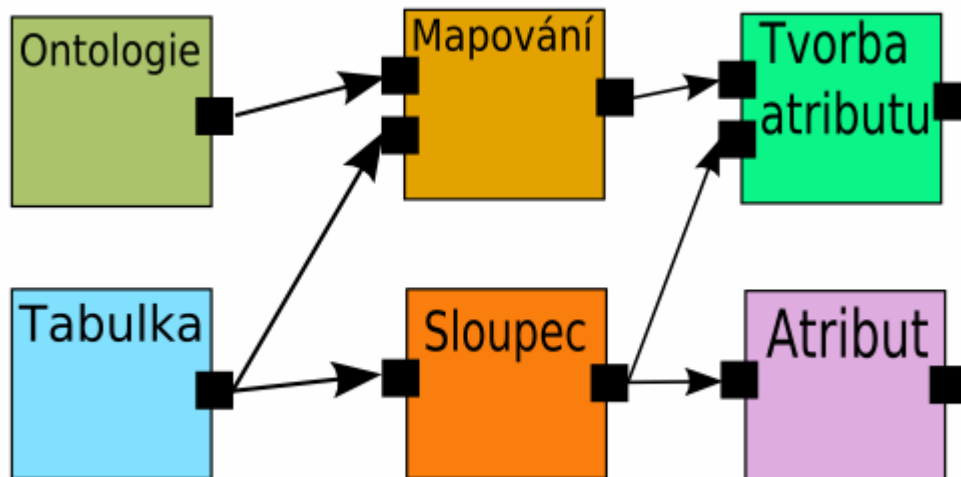
Nyní uvažujme, že by ve sloupci *KOURENI* byl uložen skutečný počet vykouřených cigaret (tzn. hodnota 15 by znamenala, že pacient vykouří 15 cigaret denně). Sémantika tohoto sloupce je zásadně odlišná od předchozí situace. Porovnání jednotlivých hodnot má reálný význam a také míru (rozdíl dvou hodnot) lze jednoduše interpretovat jako počet cigaret vykouřených denně navíc. Tento sloupec je *kardinální* a lze uvažovat o *ekvifrekvenční* či *ekvidistanční* kategorizaci atributu (pro oba typy kategorizace existují ve Ferdovi speciální krabičky: *ekvifrekvenční* a *ekvidistanční* atribut). Otázkou však je, která z těchto dvou krabiček je pro kategorizaci vhodnější a kolik kategorií se má vytvořit. Na tyto otázky však neexistuje jediná správná odpověď. Obvykle je právě zkoušení různých možností kategorizace hlavním úkolem datového analytika.

Příklad 5: Odlišná sémantika hodnot sloupců

Z výše uvedeného příkladu vyplývá, že sémantika hodnot sloupce udává v případě *kardinálních* a *ordinálních* sloupců pouze jakýsi rámec pro jejich kategorizaci.

Uvažujeme-li o automatizaci tvorby atributů, pak by výstupem tohoto procesu pro *kardinální* a *ordinální* sloupce měl být celý daný rámeček. Tzn. že by i atributy z těchto sloupců byly vytvořeny automaticky, ale analytik by měl možnost kategorizaci změnit pomocí parametrů kategorizace. Těmito parametry by byla jednak informace o tom, zda se intervaly budou vytvářet *ekvifrekvenčně*, nebo *ekvidistančně*, a jednak parametr určující počet vytvářených kategorií. Tyto parametry již oba ve Ferdovi fungují. Počet kategorií je parametrem *ekvifrekvenční* i *ekvidistanční* krabičky a způsob tvorby intervalů nastavuje analytik volbou příslušné krabičky.

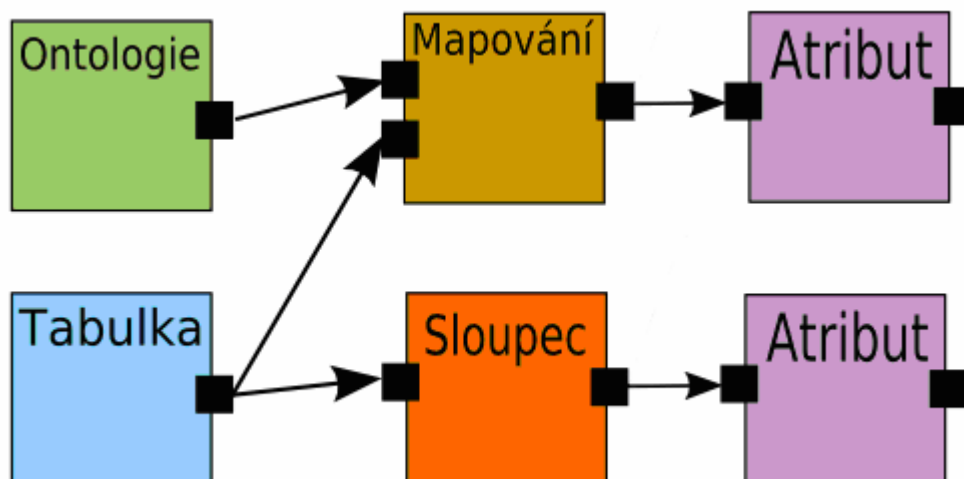
Krabičkový návrh tohoto PVO znázorňuje Obrázek 9, převzatý z [2].



Obrázek 9: Krabičkový návrh PVO Automatická tvorba atributu dle [2]

Cílem tohoto návrhu (i ostatních krabičkových návrhů v [2]) bylo nechat funkčnost, kterou zajišťují stávající krabičky ve Ferdovi, pouze na těchto krabičkách a do nově tvořených krabiček implementovat pouze novou, přidanou funkčnost. V průběhu konzultací jsme se však s vedoucím diplomové práce shodli, že bude pro uživatele přehlednější a uživatelský příjemnější, pokud se nebudeme tohoto paradigmatu příliš striktně držet. Pochopitelně zůstává snaha o omezení redundance zdrojového kódu, ale její zajištění nevyplývá z návrhu, ale je na programátorovi.

Výsledkem konzultací byl nový návrh, který umožňuje uživateli atribut přímo z mapování na jediné kliknutí. Upravený návrh znázorňuje Obrázek 10. Myšlenkou tohoto návrhu je fakt, že krabička *Mapování* zná prostřednictvím krabiček *Ontologie* a *Tabulky* všechny potřebné informace pro vytvoření *Atributu*. Ačkoliv krabičky *Sloupec* a *Atribut* (ve spodní řadě) do tohoto návrhu v podstatě nepatří, byly v Obrázku 10 ponechány úmyslně. Díky nim je pěkně vidět, že krabička *Mapování* v tomto novém návrhu supluje krabičku *Sloupec*, a tedy musí s ohledem na vytvořenou krabičku *Atribut* poskytovat stejné rozhraní.



Obrázek 10: Upravený krabičkový návrh PVO Automatická tvorba atributu

Kvůli rozhodnutí o změně napojení krabičky *Mapování* přímo za krabičku *Databáze* (viz kapitola 2.2.1) bylo nutné upravit také návrh tohoto PVO. Hlavní otázkou je, jak zajistit propojení krabičky atributu s krabičkou *Mapování*. Řešení se nabízela v zásadě tři, ale po dokončení implementace jsem začal uvažovat ještě o čtvrtém.

- 1) Napojit za krabičku *Mapování* krabičku *Tabulky* a za ní napojovat další stávající krabičky (*Sloupec* a krabičky *Atributů*). Poté vytvořit krabičku *Atribut odvozený z ontologie*, která by umožňovala využít informace z ontologie (přes řetězek předcházejících krabiček by komunikovala s krabičkou *Mapování* a získávala tak informace z ontologie). Tato možnost by byla implementačně nejjednodušší, ale obnášela by změnu v rozhraní krabičky *Tabulka*. Ta by musela mít novou zásuvku pro krabičku *Mapování*, což bylo v rozporu s předpokladem zachování rozhraní stávajících krabiček. Alternativní možností by bylo, aby krabička *Mapování* implementovala rozhraní krabičky *Databáze* a aby se pak krabička *Tabulka* napojovala buď do krabičky *Databáze*, nebo do krabičky *Mapování*. Tato možnost by ale nejspíše byla pro uživatele matoucí a tak bylo od této varianty upuštěno.
- 2) Druhým řešením by bylo kopírovat strukturu stávajících krabiček a pro každou stávající krabičku vytvořit obdobnou krabičku, která by podporovala využití ontologií. Toto řešení by obnášelo vytvořit krabičku *Tabulka podporující ontologie*, *Sloupeček podporující ontologie* a *Atribut odvozený z ontologie* s tím, že první dvě zmíněné krabičky by z hlediska funkčnosti kopírovaly krabičky *Tabulka* a *Sloupec*. Lišily by se od nich pouze v typu zásuvky, kde vstupem do stávající *Tabulky* je krabička *Databáze* a do *Tabulky podporující ontologie* by vstupem byla krabička *Mapování*. A obdobně u krabiček *Sloupece* a *Sloupece podporujícího ontologie*.

Výhodou této varianty je to, že se krabičky tváří a chovají jako původní krabičky a uživatel, který je zvyklý pracovat ve Ferdovi, si tak nemusí přivykat na nějaké výrazné novinky. Nevýhodou je zbytečný krok navíc. Tímto krokem je krabička *Tabulky podporující ontologie*. Základní myšlenka využití informací z ontologií ve Ferdovi spočívá v tom, že analytik už v krabičce *Mapování* pozná, které sloupečky

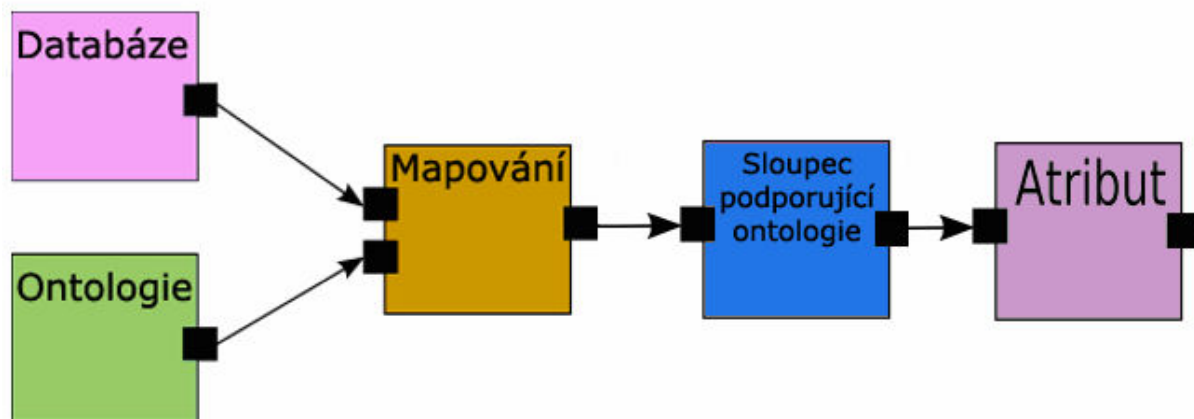
ho zajímají, a už v tomto momentě může tyto sloupečky vytvořit. A to bez ohledu na to, ve které konkrétní tabulce se daný sloupec nachází¹.

- 3) Třetí variantou je posloupnost krabiček *Mapování*, *Sloupec podporující ontologie* a *Atribut odvozený z ontologií*. Oproti předchozí variantě se liší právě ve vypuštěné krabičce *Tabulky podporující ontologie*. Jejím vypuštěním se ztrácejí následující možnosti:
 - a. Nastavení primárního klíče pro tabulku. Nezbytnou akcí, která se v krabičce *Tabulka* provádí je nastavení primárního klíče tabulky pro správné fungování navazujících modulů ve Ferdovi i samotný běh GUHA procedur. Tuto funkčnost je potřeba zachovat a v případě této varianty řešení, by ji musela převzít jiná krabička, v tomto případě nejspíše krabička *Mapování*.
 - b. Prohlížení struktury tabulky (*Explain Table Module*) a prohlížení obsahu tabulky (*Show Table Module*). Tyto dvě možnosti sice z řetízku (posloupnosti ontologických krabiček) ztratíme, ale bude-li mít uživatel potřebu prohlédnout si tabulku, je možné to vyřešit vytvořením krabičky *Tabulka* odpovídající tabulky z krabičky *Databáze* a prohlédnout si data tam.
- 4) Čtvrtá varianta vychází z úvahy, že na základě mapování znám nejen sloupce, které mě zajímají, ale již přímo atributy. Proč tedy nevypustit také krabičku *Sloupec podporující ontologie*? Zapojení krabiček by poté bylo maximálně jednoduché, na krabičku *Mapování* by rovnou navazovala krabička *Atributu odvozeného z ontologií*.

Důvodem proč původně nebyla uvažována varianta 4, byla představa o krabičce *Atributu odvozeného z ontologií*. Tato krabička měla kategorizovat atribut pouze na základě informací z ontologie. Cílem bylo ponechat uživateli možnost vytvořit ještě jinou kategorizaci daného atributu, tzn. vytvořit jiné krabičky kategorizující atribut (např. *Ekvidistanční atribut* a *Ekvifrekvenční atribut*), a proto byla krabička *Sloupece podporujícího ontologie* považována za nezbytnou.

Při implementaci se ovšem možnosti krabičky *Atributu odvozeného z ontologií* výrazně rozšířily. Bylo to dáno tím, že se z parametrů krabičky, které obsahovaly informace z ontologií a které byly původně určené pouze pro čtení, změnily na plnohodnotné uživatelem nastavitelné parametry. Tím pádem se stal z této krabičky velice silný prostředek pro kategorizaci atributu a vyústil v myšlenku univerzální krabičky pro kategorizaci. Blíže je tato myšlenka rozvedena v kapitole 6.2.1.

¹ Donedávna umožňovaly GUHA procedury implementované ve Ferdovi běh pouze nad jednou tabulkou z databáze. Hlavním přínosem krabičky *Tabulka* pro uživatele tedy bylo umožnění snadné vizuální kontroly, zda všechny atributy vstupující do dané GUHA procedury skutečně pochází ze stejné tabulky. Toto omezení částečně zrušil Alexander Kuzmin tím, že implementoval relační rozšíření vybraných GUHA procedur ve Ferdovi (4ft-Task a SD4ft-Task), čímž umožnil dobývání znalostí nad vícero tabulkami spojenými do schématu hvězdy. Toto rozšíření bylo hlavním předmětem diplomové práce [25].



Obrázek 11: Zapojení krabičky Atributu odvozeného z ontologií

Zvoleným řešením byla tedy varianta 3 (Obrázek 11), zejména z důvodu její uživatelské přívětivosti.

Klíčem k celému tomuto PVO jsou informace z ontologie, které následně krabička *Atribut odvozený z ontologie* využije ke kategorizaci hodnot. V [2] byl vytvořen seznam informací, které by se při kategorizaci atributu mohly nejvíce hodit. Byly to: Datový typ, Minimální a maximální hodnoty, Extrémní hodnoty, Významné hodnoty rozdělující doménu (pro ordinální hodnoty) a Typické hodnoty (pro nominální hodnoty). Jako příklad využití těchto informací autor uvádí pravidlo: „*Jestliže existují významné hodnoty rozdělující doménu, potom vytvoř kategorie atributu na základě těchto hodnot.*“ Autor však dodává, že podoba pravidel není rozmyšlena do té míry, aby mohl být PVO implementován. Rozmyšlení těchto pravidel bylo jedním z důležitých úkolů pro tuto diplomovou práci.

Kategorizace atributu znamená rozdělení hodnot domény do kategorií. Kategorie mohou být v zásadě pouze dvojího druhu.

- 1) Interval
- 2) Množina hodnot (speciálním a zároveň nejběžnějším případem jsou množiny s jedinou hodnotou)

Nyní jsem si vzal výše uvedený seznam a pokusil se sestavit pravidla pro jednotlivé typy informací. Na základě této snahy a konzultace s vedoucím diplomové práce vznikl následující seznam informací (vlastností), které mohou být užitečné pro kategorizaci hodnot atributů. Velice důležité při tvorbě tohoto seznamu bylo zohlednit dva odlišné účely, které tyto informace musí splnit. Na jedné straně musí tyto informace logicky spadat do ontologie, čili mají popisovat charakter příslušné entity ontologie (což je v podstatě nějaký abstraktní pojem). Na druhé straně mají být tyto informace použitelné ke kategorizaci hodnot konkrétní datové reprezentace příslušné entity. Jinak řečeno pokud uchováujeme nějaké informace o entitě v ontologii, měly by tyto informace být nezávislé na konkrétní formě hodnot reprezentujících tuto entitu v databázi. Vlastnosti navrhované pro uchování v ontologiích jsou tedy tyto:

- **Kardinalita (Cardinality)** – kardinalita, nebo též sémantika, určuje charakter škály hodnot příslušné entity ontologie. Možné hodnoty této vlastnosti jsou:
 - **Kardinální (Cardinal)** – kardinální škála je množina prvků s definovanou relací uspořádání. Kromě uspořádání prvků lze také určit míru rozdílu mezi dvěma prvky.

Např. systolický tlak tvoří *kardinální* škálu. Je možné nejen říci, kdo má vyšší a kdo nižší tlak, ale je také možné kvantifikovat rozdíl v tlaku dvou osob.

- **Ordinální (Ordinal)** – ordinální škála je množina prvků s definovanou relací uspořádání. Mezi každými dvěma prvky lze říci, který je menší, resp. větší, popř. mohou být oba prvky stejné. Nelze však kvantifikovat rozdíl mezi dvěma prvky.

Příkladem ordinálních škály může být množina prvků „špatný“, „dobrý“, „lepší“, „nejlepší“

- **Ordinální Cyklická (Ordinal Cyclic)** – ordinální cyklická škála sestává z množiny prvků, které jsou ordinální, mají definované uspořádání, ale mohou reprezentovat nějaké reálné objekty, na něž se toto uspořádání nevztahuje. Nejsnáze se to pochopí na příkladu.

Uvažujme hodnoty dnů v týdnu. Pro porovnání jednotlivých hodnot nám slouží jednoduchá tranzitivní relace ($PO < ÚT < ST < ČT < PÁ < SO < NE$), ale pokud chceme porovnat pátek 14. 12. a pondělí 17. 12., pak nám tato relace nijak nepomůže.

- **Nominální (Nominal)** – nominální škála je množina prvků, nad nimiž není definována žádná relace.

Příkladem nominálních hodnot může být například již jednou zmíněné poštovní směrovací číslo. Ačkoliv jednotlivé hodnoty *PSC* je možné jednoduše porovnávat, toto porovnání nemá žádný praktický význam.

- **Minimum** – určuje minimální hodnotu pro danou entitu (Minimum má smysl pouze na množinách hodnot, na nichž je definováno uspořádání, lze ho tedy uvažovat u kardinálních a ordinálních množin hodnot).
- **Maximum** – obdoba vlastnosti Minimum.
- **Hodnoty rozdělující doménu (Domain Dividing Values)** – hodnoty rozdělující doménu mají opět smysl pouze u kardinálních a ordinálních hodnot, jejich název je sebevysvětlující.
- **Významné hodnoty (Distinct Values)** – mezi významné hodnoty je možné zařadit různé extrémní, typické či jinak zajímavé hodnoty. Krabička *Atributu odvozeného z ontologie* ke všem těmto hodnotám přistupuje stejně, viz kapitola 4.4.

Pravidla (jejichž parametrem jsou výše uvedené informace), podle kterých krabička *Atributu odvozený z ontologie* vytváří jednotlivé kategorie, jsou popsána v kapitole 4.4, která se zabývá popisem implementace a popisem funkcí této krabičky.

Příklad 6 ukazuje příklad vyplnění hodnot vzorové entity.

Systolický tlak

- **Kardinalita** – *kardinální*
- **Minimum** – nevedeno
- **Maximum** – nevedeno
- **Hodnoty rozdělující doménu**¹ – 120, 130, 140, 160, 180
- **Významné hodnoty** – nevedeny

Příklad 6: Ukázka hodnoty vlastností v ontologii pro pojem *Systolický tlak*

Důležité je mít při určování hodnot daných vlastností neustále na zřeteli, že se jedná o tvorbu ontologie, a tedy příslušné hodnoty by měly odpovídat obecně uznávaným skutečnostem o konkrétních entitách ontologie. Pro ilustraci viz Příklad 7.

Zatímco u *Systolického tlaku* (Příklad 6) vycházelo určení hodnot vlastností z veřejně uznávané definice hypertenze, u jiných pojmů nemusí existovat takováto obecná shoda toho, jaké hodnoty by např. měly být v ontologii použity pro popis rozdělení domény.

Uvažujme jednoduchý příklad *věk člověka*. *Věk* bude nepochybně reprezentován *kardinální* škálou. Minimální věk je zřejmě 0, Maximální věk nelze jednoznačně určit, a tedy hodnota vlastnosti Maximum by měla zůstat nevyplněna.

Problém nastává při určování hodnot rozdělujících doménu. Vývojová psychologie² rozlišuje jednotlivá období lidského života, jako jsou kojenecké období, puberta, období adolescence, apod. Ovšem různé zdroje uvádějí různé věkové hranice jednotlivých období, mnohdy je uvádějí nejednoznačně (intervalem hodnot) a často používají zcela odlišné členění lidského života.

Tento problém je ovšem podobný jako jiné problémy při tvorbě ontologií, kdy je např. potřeba rozhodnout zda danou třídu dále rozdělit na podtřídy, či nikoliv. V obou případech je nutné najít shodu mezi jednotlivými subjekty, podílejícími se na tvorbě konkrétní ontologie.

Příklad 7: Ukázka potíží při určování hodnot vlastností pro entity ontologie

Práce [2] pojednává také o možnosti vytvoření expertního systému pro tvorbu atributů a atomů pomocí heuristik a odkazuje se přitom na [27], [28] a [29] (tyto práce však neuvažují o možnosti využití ontologií).

Práce [2] nastiňuje formu takového expertního systému pro kategorizaci atributu a tou jsou právě jednotlivá pravidla pro využití informací z ontologií. Tato pravidla byla v rámci této diplomové práce promyšlena a transformována do jednoduchého algoritmu pro kategorizaci atributu (tento algoritmus je podrobně popsán v kapitole 4.4).

Kromě kategorizace atributu je možné také uvažovat o vytvoření expertního systému pro tvorbu booleovských atributů jako vstupních parametrů pro ve Ferdovi implementované GUHA procedury. Tyto úvahy však tématicky spadají do následujícího PVO Vytvoření a využití skupin příbuzných atributů.

¹ Pro určení hodnot rozdělujících doménu byla použita definice hypertenze z článku *Léčba hypertenze v ordinaci praktického lékaře* (https://www.zdravcentra.cz/cps/rde/xchg/zc/xsl/81_1301.html), který tuto definici přebíral ze směrnice SZO (Světové zdravotnické organizace) a ISH (Mezinárodní společnosti pro hypertenzi) [26]

² Vysvětlení pojmu je např. na http://en.wikipedia.org/wiki/Developmental_psychology

2.2.4. Vytvoření a využití skupin příbuzných atributů

Tento PVO vychází z PVO, který byl v pracích [15] a [2] označován názvem Konstrukce dílčích cedentů za pomoci ontologie. V důsledku implementace nových GUHA procedur ve Ferdovi (viz kapitola 1.4.1), jejichž vstupem jsou místo dílčích cedentů booleovské atributy, ztrácí původní název PVO smysl, návrh se také výrazně mění, ale základní myšlenka přetrvává.

Smyslem tohoto PVO je identifikovat atributy, které jsou si významově blízké, a skupiny významově blízkých atributů pak využít při sestavování dataminingových úloh.

Myšlenka tohoto PVO vychází z předpokladu, že pokud analytika zajímá vliv jednoho atributu (*A1*) na jiný atribut (*A2*), bude ho zajímat také vliv příbuzných atributů atributu *A1* na atribut *A2*. Pro názornost je zde uveden Příklad 8.

Uvažujme opět náš vzorový příklad s analytickou otázkou STULONG 13.c.

Naším úkolem je prozkoumat, zda ve vstupních datech existuje nějaká významná závislost mezi *kouřením* a *výškou krevního tlaku* (pro jednoduchost budeme prozatím ignorovat fakt, že otázka 13.c. zkoumá závislost mezi *kouřením* a kombinací *krevního tlaku* a *indexem tělesné hmotnosti (BMI)*).

Pod pojmem *kouření* si představujeme počet vykouřených krabiček za den. Pro nás tedy výše uvedená otázka bude znít: „Existuje v datech závislost mezi počtem vykouřených krabiček za den a výškou krevního tlaku?“ a pro sestavení úlohy použijeme sloupec *KOURENI* z tabulky *Entry*.

Myšlenka tohoto PVO pak spočívá v tom, že prostřednictvím správně provedeného mapování sloupců databáze na entity ontologie, nás použitý nástroj, v tomto případě systém Ferda, upozorní na to, že se v našich datech vyskytují ještě další sloupce, které by nás mohly vzhledem k námi sestavené úloze zajímat.

V tomto příkladu by to byly sloupce *DOBAKOUR* a *BYVKURAK*.

Příklad 8: Ukázka skupiny booleovských atributů

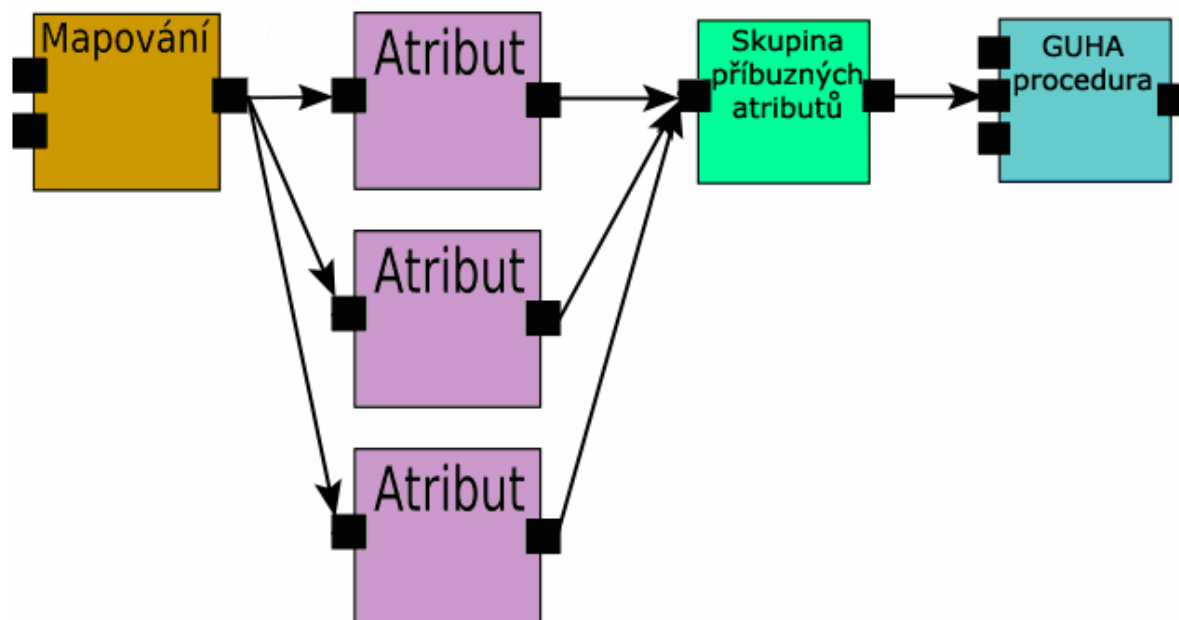
Otázkou je, jak poznat významově blízké atributy. Odpověď na tuto otázku byla popsána již v [2], já ji zde jen mírně rozvedu.

Významově blízké atributy jsou atributy odvozené z významově blízkých sloupců. To jsou sloupce, které jsou namapovány na entity ontologie, které mají společnou nadtřidu. Uvažujeme-li o tom, že bude možné mapovat sloupce také na instance v ontologiích, pak jsou příbuzné sloupce takové, které jsou namapovány na instance vytvořené ze stejného typu třídy.¹

Krabičkový návrh PVO Tvorba dílčích cedentů dle [2] předpokládá vznik nové krabičky *Tvorba dílčích cedentů*, do níž by se zapojily všechny literály, a tato krabička by je následně rozdělila do skupin podle příbuznosti (viz výše). Jednotlivé skupiny by pak reprezentovala krabička *Dílčích cedentů*.

¹ Práce [2] původně předpokládala mapování sloupců na sloty tříd ontologie, jelikož pouze u slotů se zdálo být možné uchovávat nějaké rozšiřující informace (např. datový typ). Toto řešení však bylo již v dané práci označeno za ne příliš vhodné a pro další zkoumání v oblasti využití ontologií bylo doporučováno zaměřit se na možnost uchování rozšiřujících informací u tříd a instancí tříd v ontologii, aby bylo možné mapovat sloupce na třídy a instance tříd v ontologiích. V rámci této diplomové práce byl nalezen vhodný způsob jak uchovávat rozšiřující informace o třídách a instancích (podrobnosti týkající se popisovaného způsobu reprezentace rozšiřujících informací jsou popsány v kapitole 3.5), a tedy již nadále není třeba uvažovat o mapování na sloty.

V důsledku nové implementace GUHA procedur ve Ferdovi (viz kapitola 1.4.1), je zmíněný návrh sice neaktuální, ale lze z něj dobře vyjít. Nový návrh zobrazuje Obrázek 12¹. Předpokládá vytvoření krabičky *Skupina příbuzných atributů*, do které budou zapojeny vytvořené atributy vzniklé z příbuzných sloupců. Lze uvažovat o automatické tvorbě krabičky *Skupina příbuzných atributů* přímo z krabičky *Mapování*.



Obrázek 12: Návrh PVO Vytvoření a využití skupin příbuzných atributů

Úlohou krabičky *Skupina příbuzných atributů* by bylo vytvářet různé kombinace svých vstupních booleovských atributů a předávat je jako vstupní booleovské atributy do GUHA procedury. Představu, jak by tento PVO mohl fungovat v praxi, ilustruje Příklad 9.

Uvažujme náš vzorový příklad a analytickou otázku 13.c. Omezme ji na podotázku: „*Jaký vliv má kouření na výšku tlaku?*“.

Z krabičky *Mapování* bychom na základě namapování sloupců na entity ontologie vygenerovali dvě skupiny příbuzných atributů (ty by byly reprezentovány krabičkami typu *Skupina příbuzných atributů*), z nichž jedna by odpovídala atributům odvozeným ze sloupců vyjadřujících parametry kouření (sloupce *KOURENI*, *DOBAKOUR* a *BYVKURAK*) a druhá by odpovídala atributům odvozeným ze sloupců vyjadřujících tlak. (sloupce *SYST1*, *SYST2*, *DIAS1*, *DIAS2*).

Následně by se vytvořila krabička GUHA procedury, do ní by se jako antecedent zapojila *Skupina příbuzných atributů kouření*, jako sukcedent by se zapojila *Skupina příbuzných atributů tlaku* a po spuštění procedury by se vygenerovalo velké množství jednodušších podúloh (v nichž by jednotlivými vstupy byly všemožné kombinace vstupních booleovských atributů), tyto úlohy by se spustily a dle zadaných parametrů GUHA procedury by byly vráceny dostatečně silné hypotézy.

Příklad 9: Představa fungování PVO Vytvoření a využití skupin booleovských atributů

¹ Obrázek z důvodu přehlednosti vynechává krabičky *Sloupce*, ve současné implementaci Ferdy je ještě mezi každou krabičkou *Atributu* a krabičkou *Mapování* krabička *Sloupce podporujícího ontologie*, nicméně na základě úvah o vytvoření univerzální krabičky pro kategorizaci v kapitole 2.2.3, lze uvažovat, že bude časem krabička *Sloupce podporujícího ontologie* vypuštěna

Nutno podotknout, že detaily naznačeného návrhu nejsou promyšleny do té míry, aby mohl být PVO implementován, avšak potenciál tohoto PVO v oblasti automatizace procesu DZD je značný. Promyšlení tohoto PVO je jedním z námětů navazujících prací (viz kapitola 6.2).

2.3. PVO, které byly v [2] shledány jako nevhodné pro implementaci

Cílem této kapitoly bylo zamyslet se krátce nad jednotlivými PVO, které byly v práci [2] označeny jako nevhodné pro implementaci v prostředí Ferda. U prvních dvou PVO bezvýhradně souhlasím s autorem a shledávám je taktéž nevhodnými či neužitečnými pro implementaci. U PVO Tvorba úloh pomocí ontologie a PVO Propagace zjištěných znalostí do ontologie bych však doporučoval ještě další průzkum v oblasti ontologií podporujících vyjádření neurčitosti. Výsledky takového průzkumu by totiž mohly poskytnout těmto PVO veliké možnosti a mohly by být obrovským skokem ve snaze o automatizaci procesu dobývání znalostí pomocí GUHA procedur¹. Více o těchto možnostech a ontologiích podporujících vyjádření neurčitosti je popsáno v podkapitole 2.3.3.

2.3.1. Využití ontologie pro identifikaci redundantních atributů (resp. sloupců)

Myšlenkou tohoto PVO je na základě mapování sloupců a pojmů v ontologii identifikovat redundantní sloupce v databázi. Autor v [2] správně poznamenává, že k takovéto redundanci bude docházet velmi zřídka. Obvykle bývá při návrhu databáze účasten aspoň jeden expert na oblast, kterou má databáze popisovat a lze tedy předpokládat, že v drtivé většině databází budou případné redundance odstraněny již v době návrhu.

2.3.2. Dekompozice 4FT úloh v závislosti na ontologii

Myšlenka tohoto PVO je diskutována v textech [15], [16] a [17] a vychází z představy běžného postupu dobývání znalostí pomocí GUHA procedur. Dle tohoto postupu je nejprve sestrojena co největší možná úloha, ta je poté spuštěna a nejsilnější hypotézy jsou interpretovány. Často je však interpretace takovýchto hypotéz kvůli velkému množství vstupních atributů velmi obtížná, a proto je nutné původní úlohu rozdělit na několik sémanticky správných podúloh. V tom nám dle uvedených textů mohou pomoci právě ontologie.

V práci [2] však autor názorně a zcela správně vysvětluje, že je tento PVO v zásadě redundantní, jelikož jeho funkčnost dokáže zajistit již PVO Vytvoření a využití skupin příbuzných atributů (viz kapitola 2.2.4).

2.3.3. Tvorba úloh pomocí ontologie

Práce [15] navrhuje řadu možností využití ontologií pro tvorbu úloh. Jednotlivé možnosti jsou ilustrovány názornými příklady. Všechny uvedené možnosti jsou velice slibné, ovšem vycházejí z chybného předpokladu. Tím je úvaha, že má smysl testovat platnost relací z ontologií v datech. Přesněji řečeno, tím nesprávným předpokladem je fakt, že jsou v ontologii uloženy informace typu „*Activity affects Disease or Syndrome*“ („Aktivita ovlivňuje nemoci nebo jejich syndromy“) nebo „*Smoking affects Cancer*“ („Kouření ovlivňuje rakovinu“). Problém nespočívá v technickém řešení uložení těchto informací do ontologií, to

¹ Snahy o automatizaci tohoto procesu jsou vyjádřeny prostřednictvím specifikací prozatím hypotetického systému Ever-Miner (více viz např. kapitola o Ever-Mineru v [2]).

je ve skutečnosti velmi snadné, uvedené vztahy se do ontologií uloží jako relace mezi třídami. Problém je v chybné představě o úloze ontologií.

Své pochybnosti ohledně toho, zda výše uvedená tvrzení patří do ontologií, nebo ne, jsem měl tu příležitost konzultovat s předními českými odborníky na problematiku ontologií¹. Na základě těchto konzultací jsem dospěl k názoru, že do ontologií výše zmíněná tvrzení nepatří a že by ontologie měly obsahovat pouze informace, které jsou obecně platné.

Zmíněnými odborníky jsem byl zároveň upozorněn na existenci výzkumu zabývajícího se možnostmi uchování neurčitých skutečností v ontologiích (doporučené zdroje pro další studium jsou [30], [31], [32]). Na základě úvodního náhledu do dané problematiky se mi zdá, že je daná oblast teprve na začátku bádání a že zatím nebyl určen žádný standard, ze kterého by bylo možné vyjít při implementaci *neurčitých („Fuzzy“) ontologií* ve Ferdovi². V rámci této diplomové práce však nebylo provedeno detailnější studium dané problematiky a vzhledem k potenciálnímu přínosu, který by tato oblast mohla pro DZD ve Ferdovi přinést, bylo zkoumání této oblasti doporučeno jako jeden z možných směrů při hledání dalších možností automatizace procesu DZD, viz kapitola 6.2.3 Náměty na další práci.

Pro ilustraci potenciálu *neurčitých ontologií* doporučuji prostudovat kapitolu Formulace analytické úlohy v práci [15] a uvážit, že všechny uvedené možnosti tvorby úlohy by mohly být na základě *neurčitých ontologií* proveditelné. V [15] navrženými postupy tvorby úlohy se zabývá také práce [2]. S předpokladem, že jsou neurčitá tvrzení uchovaná v ontologii, dochází ke třem hlavním otázkám, jejichž zodpovězení je nezbytné pro smysluplnou využitelnost navržených postupů. Tyto otázky zní:

- Která relace ontologie se má pro úlohu použít?
- Která procedura se má použít?
- Jak se má „sestavit“ úloha a který kvantifikátor se má použít?

Otázky volby procedury (a kvantifikátoru) a sestavení úlohy by bylo možné dle autora v [2] řešit pomocí nějakých heuristik. Pro potřeby prokázání užitečnosti daného PVO by však mohlo být postačujícím řešením také použití omezeného počtu nejčastěji používaných procedur a kvantifikátorů a standardní zapojení a nastavení krabiček. Po prokázání užitečnosti PVO by následně pro rozšíření možností mohl vzniknout nějaký, více či méně složitý, expertní systém.

Zásadním problémem je volba relace, která se má pro úlohu použít. Jsou-li všechny relace v ontologii stejného typu, pak je skutečně velmi obtížné určit, která skrývá vztah, který může být zajímavé ověřit v datech. A zde by nám mohla pomoci právě podpora *neurčitých ontologií*. Má představa *neurčitých ontologií* je taková, že by byly rozšířením běžných doménových ontologií. Tzn. byla by to v podstatě doménová ontologie, v níž by navíc bylo možné uchovávat neurčitá tvrzení. Tato speciální tvrzení by byla v rámci ontologie jasně odlišena od ostatních relací, čímž by se jednoduše identifikovala zajímavá tvrzení, jejichž platnost v datech by mohlo být zajímavé ověřit.

Pro lepší představu o možných přínosech tohoto PVO viz Příklad 10.

¹ Těmito odborníky byli Doc. Ing. Vojtěch Svátek, Dr. a Ing. Ondřej Šváb z Katedry informačního a znalostního inženýrství Vysoké školy ekonomické v Praze. Nutno podotknout, že se jedná o odborníky uznávané nejen v České republice, ale i celosvětově. Např. doc. Svátek je členem pracovní skupiny W3C, která se zabývá standardizací jazyka OWL určeného pro reprezentaci ontologií.

² Narozdíl od doménových ontologií, pro jejichž reprezentaci byla vytvořena řada standardních jazyků (viz kapitola 3).

Uvažujme následující situaci: máme k dispozici kvalitní databázi a chceme zjistit, zda mezi jednotlivými daty existují nějaké zajímavé závislosti. Pomocí podpory PVO Mapování (viz kapitola 2.2.1) namapujeme sloupce z databáze na entity z ontologie. V následném kroku necháme systém vygenerovat zajímavé úlohy na základě neurčitých tvrzení (neurčitých relací) z ontologie. A systém nám prostřednictvím nějakého vhodného výstupu oznámí, jaké zajímavé závislosti a s jakou mírou jsou platné v datech, příp. nás upozorní na rozpor našich dat oproti tvrzením z ontologie.

Příklad 10: Potenciální přínos neurčitých ontologií

2.3.4. Propagace zjištěných znalostí do ontologie

S úvahami o využití *neurčitých ontologií* k tvorbě úloh úzce souvisí úvahy o propagaci zjištěných znalostí do ontologie. Jestliže můžeme a chceme v ontologiích uchovávat neurčitá, ale přesto silná, tvrzení o vztazích mezi entitami, nabízí se otázka, odkud tato tvrzení čerpat. A jedním z nejvýznamnějších zdrojů může být právě proces DZD pomocí GUHA procedur. Analytik objeví nějakou silnou závislost ve zdrojových datech a kromě běžného reportu¹, v němž danou závislost interpretuje, navrhne také přidání dané informace do ontologie. Samotný proces přidání relace do ontologie pak již závisí na vlastníkově (správci) dané ontologie.

Jiným zajímavým zdrojem neurčitých tvrzení by mohly být znalosti doménových expertů, kteří by formulovali v oboru obecně uznávaná tvrzení. Na základě vhodných zdrojových dat by se platnost těchto tvrzení v datech dala jednoduše (a možná automatizovaně) ověřit. Uvedené znalosti doménových expertů jsou označovány jako *background knowledge* a podrobněji o nich pojednává [2].

¹ Podporou automatické tvorby reportů z běhů GUHA procedur se zabývá nástroj LISp-Miner Report Assistant [21], vyvinutý na MFF UK.

3. Volba jazyka pro reprezentaci ontologií

Základním problémem, který je potřeba vyřešit před tím, než bude implementována jakákoliv podpora ontologií ve Ferdovi, je volba vhodného jazyka pro reprezentaci používaných ontologií. Problém je diskutován již v [2]. V následujících odstavcích shrnu a rozvinu tuto diskusi.

Požadavky na jazyk pro reprezentaci ontologií jsou:

- **odpovídající vyjadřovací síla** – jazyk musí mít dostatečnou vyjadřovací sílu pro potřeby ontologií (popis tříd, vlastností těchto tříd, tzv. slotů a instancí tříd)
- **možnost uchovávat v ontologii rozšiřující informace o entitách ontologie** – tento požadavek vyplývá z návrhu krabičky pro automatickou tvorbu atributů, cílem této krabičky je určení (nebo aspoň nápověda) správného typu kategorizace atributu na základě rozšiřujících informací z ontologie (např.: jestliže budeme vědět, že hodnoty dané entity jsou *kardinální*, pak je zřejmé, že je možné z odpovídajícího sloupce vytvořit *ekvifrekvenční* nebo *ekvidistanční* atribut, budou-li však hodnoty např. *nominální*, pak víme, že smysl má pouze atribut typu *each value each category*)
- **snadná tvorba ontologií** – pro uživatele by neměl být použitý jazyk překážkou při tvorbě ontologie – existence nástrojů pro tvorbu ontologií v daném jazyce je velkou výhodou, stejně jako existence podpůrných nástrojů pro samotný jazyk (např. parser jazyka)

V [2] byly navrženy dvě možná řešení pro reprezentaci ontologií: OWL a *Prezentační ontologie*. Při studiu problematiky byly identifikovány další potenciálně zajímavé varianty a to tzv. *Topic Maps* a OKBC (Open Knowledge Base Connectivity).

Velice užitečným zdrojem informací byla [33], odkud také pochází Dodatek D této práce, který velmi pěkně znázorňuje vývoj jazyků pro reprezentaci ontologií.

3.1. Prezentační ontologie

Prezentační neboli *extrakční ontologie* jsou ontologie, které slouží k získávání a strukturování informací z obecně nestruturovaných textů a dokumentů. Typickým příkladem využití *prezentačních ontologií* je získávání informací z internetu – např.: vyhledání všech relevantních nabídek prodávaných cyklistických kol v České republice.

Základním úkolem, který *extrakčních ontologie* řeší, je identifikování jednotlivých atributů objektu v textu.

Uvažujme výše zmíněný příklad, který byl řešen v rámci projektu RAINBOW [34] – vyhledání internetových nabídek na prodej cyklistických kol. Cílem tohoto projektu bylo pomocí vytvořené aplikace vyhledat všechny relevantní informace. Hlavní problém spočíval v tom, že nebyly známy jednotlivé formáty, v nichž jsou nabídky jednotlivých prodejců publikovány. A právě získání těchto informací bez znalosti konkrétních formátů je hlavní úkol *extrakčních ontologií*. Ontologie dané aplikaci poskytne informace o tom, jak by taková data mohla vypadat a jak je v obecně nestruturovaném textu identifikovat (např. by taková ontologie mohla poskytovat informaci o tom, že číselný řetězec před výrazem „- Kč“ značí cenu).

Příklad 11: Ukázkový příklad využití *extrakčních ontologií*

Prezentační/extrakční ontologie jsou zatím ve fázi výzkumu a experimentování. Zabývá se jimi například [35], [36] či [34].

Důležité je zmínit fakt, že jazyk ani formát *prezentačních ontologií* není standardizován – ontologie bývá někdy reprezentována pomocí jazyku RDF, DAML, ale většinou je použitý jazyk výrazně upraven pro potřeby konkrétního projektu (někdy pouze XML s vlastním DTD).

Prezentační ontologie mají velký potenciál v oblasti sémantického webu. Pro další informace o *prezentačních ontologiích* doporučuji [37] a Přílohu 3¹.

3.2. OWL (*Web Ontology Language*)

Jazyk OWL (viz [38]) je standardem konsorcia *W3C* učeným pro reprezentaci ontologií. Důvodem pro vznik standardu pro reprezentaci ontologií byl rozvoj výzkumu v oblasti sémantického webu. Základní myšlenkou sémantického webu je změna prostředí *WWW (World Wide Web)* tak, aby byly internetové stránky čitelné nejen lidmi, ale aby byly informace z internetu také správně interpretovány počítači. Aby toho mohlo být dosaženo, je potřeba jednak popsat informace zveřejňované na internetových stránkách (dodat jim význam – sémantiku) a jednak vysvětlit počítačům, co jednotlivé informace znamenají (jaké jsou mezi jednotlivými informacemi vazby, apod.) a právě k tomu slouží ontologie a to je důvod, proč se konsorcium *W3C* rozhodlo pro vytvoření standardizovaného jazyka pro jejich záznam.

Prvním krokem *W3C* k vytvoření standardu pro reprezentaci vztahů mezi objekty byl jazyk **RDF** (*Resource Description Framework*) [39]. Vyjadřovacím prostředkem tohoto jazyka jsou tzv. *trojice* (angl. *triplets*), které vyjadřují vazbu mezi subjektem a objektem. Obecně lze tuto vazbu interpretovat následovně: „*Zdroj (subjekt) X nabyvá pro vlastnost (predikát) Y hodnoty (objekt) Z.*“ Názorně tuto vazbu ukazuje Příklad 12.

Následující RDF *trojice* říká, že alternativní název pro New York je NY
<urn:states:New%20York> <http://purl.org/dc/terms/alternative> "NY"
ve standardním RDF/XML² formátu se tato informace zapíše následovně:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:terms="http://purl.org/dc/terms/">
  <rdf:Description rdf:about="urn:states:New%20York">
    <terms:alternative>NY</terms:alternative>
  </rdf:Description>
</rdf:RDF>
```

Příklad 12: Ukázka trojice RDF

Nedostatkem *RDF* byla jeho nízká vyjadřovací schopnost a značná rozvláčnost. Výsledkem snah o vylepšení *RDF* byla nadstavba nad *RDF*, označovaná jako – **RDF Scheme**, tento jazyk umožňuje oproti *RDF* navíc specifikovat vztah třídy a podtřídy, vlastnosti a “podvlastnosti” a definiční obor a obor hodnot vlastnosti.

¹ Příloha 3 je přehled informačních zdrojů o prezentačních ontologiích a příbuzných tématech vytvořený Doc. Vojtěchem Svátkem z VŠE v Praze během příprav workshopu PriCKL 2007 <http://keg.vse.cz/prickl07/>.

² Alternativní formou k RDF/XML pro uchování RDF trojic je uchování těchto trojic v databázi. Potřebám systému Ferda je bližší RDF/XML.

- *subclass* (*Okres, Území*)
- *subproperty* (*sousedí, je_blízko*)
- *domain* (*lokalita*) = *Nemovitost*
- *range* (*lokalita*) = *Území*

Příklad 13: Nové možnosti RDF Scheme oproti RDF

Na *RDF* (resp. *RDF Scheme*) navázala svým výzkumem organizace *DARPA*. Cílem jejího projektu, *DAML* (*DARPA Agent Markup Language*) bylo vytvořit jazyk pro reprezentaci ontologií a pro značkování dat. Projekt do sebe záhy integroval obdobný projekt – *OIL* (*Ontology Inference Layer*) a nadále byl označován jako ***DAML+OIL***. Více viz [40].

V roce 2001 založila organizace *W3C* pracovní skupinu „*Web Ontology Working Group*“, která byla pověřena vědeckým přezkoumáním projektu *DAML+OIL* a vytvořením nového návrhu jazyka pro reprezentaci ontologií. V únoru 2004 bylo formálně schváleno a vydáno doporučení *W3C* s názvem **OWL** (*Web Ontology Language*).

Tabulka 2 strukturovaně uvádí hlavní charakteristiky jazyka *OWL*. Více o jazyku *OWL* viz [38].

- OWL je postaven nad RDF
 - OWL umožňuje lepší strojovou využitelnost, má větší slovník a silnější syntaxi než RDF
- OWL je psán v XML
- OWL slouží ke zpracování informací na webu a byl navržen tak, aby ho mohl snadno využít počítač, cílem nebyla čitelnost pro lidi
- OWL má tři podjazyky (někdy jsou označovány jako dialekty), které se liší vyjadřovací silou
- OWL Full
 - obsahuje všechny elementy a konstrukce jazyka OWL
 - umožňuje neomezené použití RDF konstrukcí
 - třídy mohou vystupovat zároveň jako instance jiných tříd
- OWL DL (Description Logic)
 - oproti OWL Full přináší jistá omezení, např: není možné používat všechny RDF konstrukce, třída ani vlastnost (slot) nesmí být zároveň instancí třídy, a další
 - odměnou za tato omezení je výpočetní úplnost a rozhodnutelnost jazyka OWL DL
- OWL Lite
 - nejnižší úroveň složitosti (menší počet povolených elementů oproti OWL DL a Full) představuje varianta OWL Lite, umožňuje definovat základní taxonomii a jednoduchá omezení
- OWL je standard *W3C*

Tabulka 2: Charakteristika jazyka OWL

3.3. Topic Maps

Topic Maps (tématické mapy) [41] jsou standardem ISO pro reprezentaci a výměnu znalostí (ISO/IEC 13250:2003).

Tématické mapy (Topic Maps) jsou jakousi obdobou ontologií. Stejně jako pro ontologie byl i pro *tématické mapy* hledán jazyk pro jejich reprezentaci. Tento jazyk je označován jako *Topic Maps Constraint Language (TMCL)*, ale jedná se pouze o hypotetický model, který zatím není v praxi realizovaný. Bylo vytvořeno několik adeptů na *TMCL*, např. *AsTMA!*, *OSL*, a *Xtche*, ale žádný z nich není standardem a žádný z nich zatím není nějak významně rozšířen.

3.4. Srovnání Prezentačních ontologií, Topic Maps a OWL

V době psaní [2] se *prezentační (extrakční) ontologie* jeví jako vhodná technologie, která by umožnila snadné zachycení doplňkových informací k entitám, na jejichž základě by bylo možné určit typ atributu (toto bylo evidentní slabé místo OWL). Ovšem po prostudování problematiky je zřejmé, že *prezentační ontologie* nejsou vhodnou volbou pro tento projekt. Hlavním důvodem je fakt, že struktura *prezentačních ontologií* je odlišná od ontologií doménových. Je to dáno především rozdílným účelem těchto ontologií. Cílem *prezentační*,

neboli *extrakční ontologie* je popsat způsob jak identifikovat v textu jednotlivé entity z ontologie. Oproti požadavkům doménové ontologie je tak v *prezentačních ontologiích* řada ve Ferdovi nevyužitelných informací. Lze také snadno odhadovat, že vytvoření *prezentační ontologie* bude mnohem náročnější než vytvoření ontologie doménové, čímž se ztrácí další z výhod OWL, kde se dá předpokládat řada ontologií, které vzniknou nezávisle na Ferdovi a přesto budou využitelné¹. Dalším velmi významným důvodem, proč nevyužívat prezentační ontologie je fakt, že tato oblast je teprve ve fázi výzkumu a vývoje a není v ní dosud žádný standardizovaný jazyk – některé projekty využívají jazyk *DAML*, některé např. pouhé *XML* a na míru vytvořené *DTD*.

Tématické mapy (Topic Maps) jsou obdobou ontologií. Ačkoliv byly *tématické mapy* prohlášeny v roce 2000 ISO standardem, nedokázaly se výrazněji prosadit. Důvodem je jednak neexistence standardního jazyka pro jejich reprezentaci a jednak, a to především, obrovská konkurence ve formě standardů *W3C (RDF Scheme a OWL)*. Konsorcium *W3* zkrátka v oblasti internetových standardů nemá konkurenci, a tedy dle mého názoru nemají *tématické mapy* budoucnost. Problematické uplatnění *tématických map* si uvědomili také jejich autoři a snažili se pro ně nalézt uplatnění v oblastech, kde by nebyly standardy *W3C* tak silné (viz např. článek [42]), ale identifikované slabiny souvisely spíše s nesprávným používáním *W3C* standardů, než s tím, že by neumožňovaly to, co *tématické mapy*.

V rámci studia problematiky mne ještě zaujalo tzv. *OKBC (Open Knowledge Base Connectivity)*. Nejedná se o jazyk pro reprezentaci ontologií, ale o softwarové API pro přístup k znalostním bázím. Myšlenka je obdobná jako u API *ODBC (Open Database Connectivity)*: zprostředkovat přístup k znalostem (v případě *ODBC* datům z databáze) nezávisle na programovacím jazyku, operačním systému, či formě, v jaké jsou znalosti uloženy. *OKBC* má slibný potenciál, ale prozatím není příliš rozšířeno. Vzhledem ke specifickým požadavkům této práce na jazyk pro reprezentaci ontologií (uchovávat u entit rozšiřující informace) a vzhledem k tomu, že bylo obtížné najít byť jediný jazyk, který by tyto požadavky dokázal splnit, bylo *OKBC* shledáno pro tuto práci jako neúčinné. Více o něm je možné se dozvědět např. na [43].

Zvoleným jazykem pro reprezentaci ontologií byl jazyk *OWL*, hlavními důvody byla skutečnost, že se jedná o standard *W3C*, díky čemuž je již dnes hojně rozšířený a používaný pro tvorbu velkého množství ontologií. Také existuje řada nástrojů, které tento jazyk podporují (např.: *Protégé, Jena, Sesame, RACER*). Přestože volba pro jazyk *OWL* byla vzhledem k ostatním možnostem zcela jednoznačná, nebylo v okamžiku volby zřejmé, jak budou požadavky dané touto prací pomocí tohoto jazyku implementovány. O řešení klíčových problémů pojednává následující kapitola.

3.5. Jak v ontologii uchovat rozšiřující informace o entitách

Pro porozumění úvah v této kapitole, je potřebné rozumět názvosloví oblasti ontologií (stručně vysvětleny jsou např. v Příloze 1). Zejména je důležité rozumět pojmu *slot* (což je vlastnost třídy ontologie) a pojmu *facet* (což je vlastnost slotu).

Jazyk *OWL*, jak již bylo napsáno v kapitole 3.2, má několik podjazyků (dialektů). Jsou to *OWL Full, OWL DL a OWL Lite*. Kromě těchto dialektů, bylo v roce 2005, na základě připomínek uživatelů jazyka *OWL*, vytvořeno rozšíření dialektu *OWL DL*, které je označováno jako *OWL 1.1* [44].

¹ *Extrakční ontologie* se od doménových ontologií zásadně odlišují kromě účelu také ve způsobu svého vzniku. Většina *extrakčních ontologií* vzniká prostřednictvím učení nějaké neuronové sítě. Doménové ontologie jsou oproti tomu vytvářeny přímo na základě konsensu představ jednotlivých tvůrců ontologie.

OWL 1.1 je nyní ve stádiu *W3C Member Submission* standardizačního procesu. Přínosem OWL 1.1 z pohledu této diplomové práce je zejména rozšíření množiny datových typů pro sloty (*data properties*) v ontologiích. V OWL 1.1 jsou jako datové typy slotů povoleny všechny konstrukce datových typů jako v jazyce *XML-Schema* (včetně možnosti definovat si vlastní uživatelský typ).

Klíčovým požadavkem této práce na jazyk pro reprezentaci ontologií, je možnost uchovat rozšiřující informace k jednotlivým entitám v ontologii. Přehled rozšiřujících informací, které chceme v ontologiích uchovávat, vzešel z návrhu PVO Automatická tvorba atributu (viz kapitola 2.2.3).

Nyní se dostáváme k otázce, jak reprezentovat uvedené rozšiřující informace v ontologii.

Práce [2] navrhovala poměrně nejasné řešení, které spočívalo v nutnosti použití tříd, jejichž instance mohou být zároveň třídy. To umožňuje pouze dialekt OWL Full, který však není výpočetně úplný a rozhodnutelný.

Snažil jsem se najít řešení, které by umožnilo uchovat informace i v OWL DL jazyce. Takové řešení jsem skutečně našel. Jeho principem je vytvoření slotů pro jednotlivé typy rozšiřujících informací. Doménou těchto slotů je třída *owl:Thing*. Tato konstrukce pak umožňuje u všech instancí libovolné třídy vyplnit hodnoty rozšiřujících informací. Velkou nevýhodou tohoto řešení je fakt, že umožňuje uchovat rozšiřující informace pouze u instancí tříd.

Díky dalším experimentům s nástrojem Protégé [24] však bylo nalezeno jiné, elegantnější a především jednodušší řešení. Toto řešení využívá faktu, že všechny třídy ontologie v jazyce OWL (*owl:Class*) jsou zároveň instancemi metatřídy *rdfs:Class*. Pro dosažení našeho cíle pak stačí jen upravit výše popsáný návrh s tím rozdílem, že doménou slotů nebude třída *owl:Thing*, ale třída *rdfs:Class*. Výhodou tohoto řešení je skutečnost, že je možné dokonce i v OWL Lite.

Aby bylo možné uchovávat rozšiřující informace i o třídách i instancích, bylo by potřeba obě výše uvedená řešení zkombinovat. Prozatím však byla ve Ferdovi implementována pouze podpora druhého řešení. Ferda tedy zatím neumožňuje uchovávat rozšiřující informace o instancích. Bez újmy na obecnosti lze však z každé instance vytvořit třídu a tak je možné se bez rozšiřujících informací u instancí obejít. Při tvorbě ontologie to pak stačí mít na zřeteli a entity, pro které chci uchovat rozšiřující informace, vytvářet jako třídy ontologie.

Pro tvorbu OWL ontologií existuje řada nástrojů (např. *OntoTrack* [45], *Jena* [46], nebo *SWOOP* [47]), ale v rámci této práce byl používán výborný nástroj Protégé [24], konkrétněji verze Protégé 3.3. V době psaní této diplomové práce existovala již alfa verze Protégé 4.0, tato verze přinášela nově podporu OWL 1.1, ale jelikož se jednalo teprve o alfa verzi, neměla plnou funkčnost a neumožňovala například zobrazení metatříd, což je pro tuto diplomovou práci nezbytné. Návod, jak v systému Protégé vytvořit ontologii, která v sobě umožní uchování Ferdou definovaných typů informací, je součástí této diplomové práce – Příloha 7.

4. Implementace modulů (krabiček)

V kapitole 2.2 jsou popsány postupy pro využití ontologií v prostředí Ferdy. Jádrem těchto postupů jsou návrhy krabiček, které by umožňovaly podporu příslušných PVO.

Pro implementační část této práce byl vybrán PVO Automatická tvorba atributu (viz kapitola 2.2.3). Nutným předpokladem pro tento PVO je PVO Mapování (kapitola 2.2.1). Na základě návrhů těchto PVO tak byly k implementaci určeny krabičky *Ontologie (Ontology)*, *Mapování ontologie (Ontology Mapping)*, *Sloupec podporující ontologie (Ontology Enabling Column)* a *Atribut odvozený z ontologie (Ontology Derived Attribute)*. V průběhu implementačních prací bylo ještě rozhodnuto o implementaci části funkčnosti z PVO Vytvoření a využití skupin příbuzných atributů (kapitola 2.2.4). Více o tomto doplňku pojednává kapitola 4.2, popisující implementaci krabičky *Mapování ontologie*, do níž byla tato funkčnost integrována.

V této části jsou v jednotlivých podkapitolách popsány implementované krabičky, jejich funkčnost, vlastnosti, akce apod. Zároveň jsou v nich popsány a diskutovány zajímavé problémy a otázky, na něž jsem v průběhu implementace daných krabiček narazil.

Pro implementaci krabiček bylo potřeba nastudovat programátorskou dokumentaci Ferdy (ta je součástí zdrojových kódů aplikace Ferda [23]). Neméně důležité bylo nastudování middlewaru *ICE (Internet Communications Engine [22])*, který je použit pro komunikaci mezi jednotlivými moduly systému Ferda. Každá krabička má určené rozhraní, jehož prostřednictvím s ní mohou ostatní krabičky komunikovat – to je tzv. *SLICE (ICE návrh)*. *SLICE* také definuje datové struktury, které mohou být použity jako parametry a návratové hodnoty funkcí.

Implementaci jednotlivých krabiček předcházelo vytvoření testovací ontologie, na níž bylo průběžně testováno fungování vytvářených krabiček. Postup pro vytvoření ontologie popisuje kapitola 5.1.

4.1. Krabička Ontologie

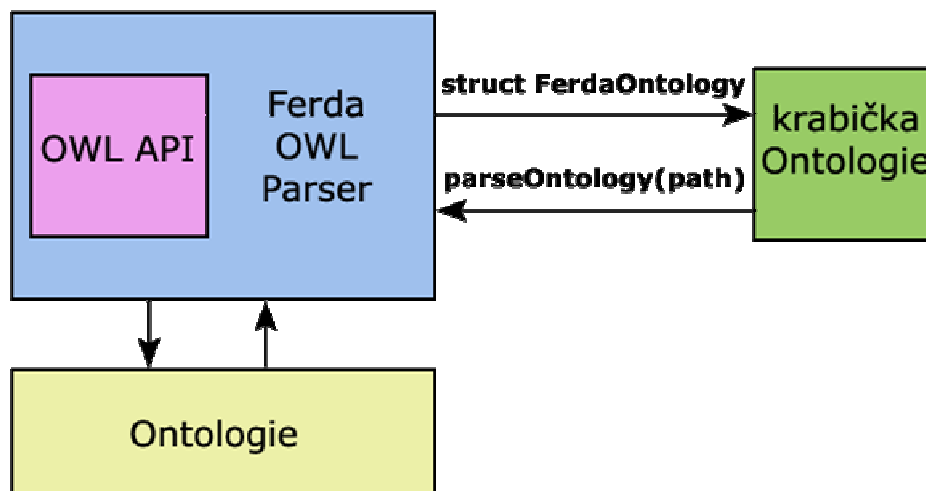
Úlohou krabičky *Ontologie* je připojení se k ontologii a poskytnutí informací o entitách z ontologie dalším krabičkám. V kapitole 3 byl zvolen jazyk pro reprezentaci ontologií, je jím OWL Full. Prvním úkolem při implementaci této krabičky tak bylo zajistit parsování informací uložených v souboru s ontologií.

Díky tomu, že je Ferda postaven nad middlewarem *ICE*, je programátorům poskytnuta částečná jazyková nezávislost. Dosud byl Ferda programován pouze v jazyce *C#*, ale v rámci této práce byla využita (a tím i ověřena) možnost použití jiného jazyka. Tímto jazykem byla Java.

Důvodem pro použití Javy byla skutečnost, že v ní byl implementován volně šiřitelný parser jazyka OWL¹. Obdobný parser nebyl implementován v *C#*, a tak zbývaly dvě možnosti. Buď použít javový parser a hledat způsob, jak ji propojit s krabičkami ve Ferdovi, nebo implementovat OWL parser v jazyce *C#*. Na základě konzultací s vedoucím diplomové práce byla vybrána první možnost. Hlavním důvodem byla očekávaná obtížnost programování parseru, vedlejším důvodem pak snaha ověřit možnost využití jiných než *C#* modulů v systému Ferda.

¹ Tento parser je součástí implementace API pro standard OWL, která je nazývána OWL API. Původní verze OWL API byla vytvořena v rámci projektu *WonderWeb* (<http://wonderweb.semanticweb.org/>). Novější verze OWL API, která podporuje také OWL 1.1 (viz kapitola 3.2) a která byla použita ve Ferdovi, vznikla v rámci projektu <http://www.co-ode.org/>. Domovská stránkou OWL API je <http://owlapi.sourceforge.net>.

Propojení javovské komponenty s krabičkami ve Ferdovi bylo pro mě nejobtížnější částí celé implementace a několikrát jsem musel konzultovat své potíže s tvůrci middlewaru *ICE*. Schématicky propojení javovské části se zbytkem Ferdy ilustruje Obrázek 13.



Obrázek 13: Integrace javovského parseru OWL ontologií do Ferdy

V javě byl napsán modul (*FerdaOWLParser*), který využíval OWL API pro načtení pro Ferdu důležitých informací z ontologie. Pro tento modul byl vytvořen ICE návrh, v němž bylo definováno rozhraní pro tento modul, které sestávalo z jediné funkce *parseOntology()* s parametrem typu *string* udávajícím cestu k ontologii. Prostřednictvím tohoto rozhraní pak s modulem *FerdaOWLParser* komunikuje již v C# programovaná krabička *Ontologie*, která volá zmíněnou funkci *parseOntology()*, jejímž výstupem je opět pomocí ICE návrhu definovaná struktura pro důležité informace o ontologii.

Tabulka 3 popisuje všechny důležité vlastnosti krabičky *Ontologie* z pohledu prostředí Ferda.

ikona krabičky	
název krabičky	<i>Ontologie (Ontology)</i>
vstupní zásuvky	---
parametry krabičky	<ul style="list-style-type: none"> • <i>cesta k ontologii (Path to Ontology)</i> • <i>počet tříd ontologie (Number of Classes)</i> • <i>URI ontologie (ontology URI)</i>
moduly pro nastavení	<ul style="list-style-type: none"> • <i>modul pro nastavení cesty k ontologii (Set Path to Ontology Module)</i>
akce krabičky	<ul style="list-style-type: none"> • <i>Znovunačtení ontologie (Reload Ontology)</i>
moduly pro interakci	---

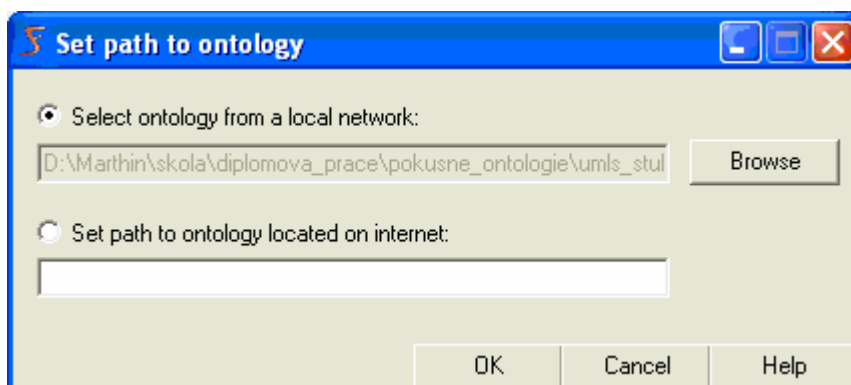
krabičky nabízené na vytvoření	<ul style="list-style-type: none"> • <i>Mapování ontologie (Ontology Mapping)</i>
--------------------------------	--

Tabulka 3: Vlastnosti krabičky Ontologie

Ikona krabičky znázorňuje sovu, podle překladu názvu zvoleného jazyka pro reprezentaci ontologií (OWL).

Krabička *Ontologie* neobsahuje žádné vstupní zásuvky ani moduly pro interakci.

Parametry krabičky *počet tříd ontologie* a *URI ontologie* jsou určeny pouze pro čtení a jsou nastavovány v okamžiku načtení ontologie krabičkou. Parametr *cesta k ontologii* slouží k určení, jakou ontologii má krabička *Ontologie* načíst. K nastavení této cesty slouží *modul pro nastavení cesty k ontologii*. Jedná se o jednoduchý dialog (viz Obrázek 14), v němž je možné nastavit cestu buď k ontologii na lokální síti, nebo zadat její umístění na internetu.



Obrázek 14: Modul pro nastavení cesty k ontologii

Krabička umožňuje akci *Znovunačtení ontologie*, která se pokusí znovu načíst zvolenou ontologii. Prvotní načtení ontologie se provádí vždy po zadání cesty k ontologii, tzn. po kliknutí na OK ve výše zobrazeném dialogu.

Krabička nabízí možnost automatického vytvoření krabičky *Mapování ontologie*.

4.2. Krabička *Mapování ontologie*

Krabička *Mapování ontologie* je klíčovou krabičkou celé této práce. Prostřednictvím mapování mezi sloupci datové tabulky a entitami ontologie zajišťuje propojení databáze a ontologie. Toto mapování je nezbytným předpokladem pro smysluplné využití ontologií při dobývání znalostí. Mapovanými entitami z ontologie mohou být jednak sloupce a jednak instance ontologií¹.

Implementace využívá toho, že v ontologii v jazyce OWL se každý název entity může vyskytovat pouze jednou, a tedy názvy entit mohou být použity jako jednoznačné identifikátory. Na druhou stranu sloupec z databáze je jednoznačně identifikován názvem sloupce a názvem tabulky, v níž je tento sloupec obsažen.

ikona krabičky	
název krabičky	<i>Mapování ontologie (Ontology Mapping)</i>

¹ Rozšiřující informace o entitách lze zatím uchovávat pouze u tříd, objasnění tohoto omezení je popsáno v kapitole 3.5.

vstupní zásuvky	<ul style="list-style-type: none"> • <i>Databáze (Database)</i> • <i>Ontologie (Ontology)</i>
parametry krabičky	<ul style="list-style-type: none"> • <i>mapování (mapping)</i> • <i>počet namapovaných dvojic (number of mapped pairs)</i> • <i>primární klíče tabulek (primary keys)</i>
moduly pro nastavení	<ul style="list-style-type: none"> • <i>nastavení mapování (Set Ontology Mapping)</i> • <i>nastavení primárních klíčů tabulek (Set Primary Keys)</i>
akce krabičky	---
moduly pro interakci	<i>Zobrazení mapování (Show mapping)</i>
krabičky nabízené na vytvoření	<i>Sloupec podporující ontologie (Ontology Enabling Column)</i>

Tabulka 4: Vlastnosti krabičky Mapování ontologie

V původním návrhu krabičky *Mapování ontologie* v [2] byla předpokládána možnost zapojení více krabiček typu *Ontologie* do zásuvky, ale v konečné implementaci je povolena pouze jedna krabička daného typu. Důvodem je právě možnost jednoznačné identifikace pomocí názvu entity ontologie, k níž se váže příslušný sloupec. Jedním z hlavních požadavků na vytvořené mapování byla jeho znovupoužitelnost. Tj. aby umožňovalo, po zapojení odpovídající ontologie a databáze a následném načtení mapování, propojit odpovídající sloupce tabulek s ontologickými entitami. Problém je v tom, že nelze ontologii jednoznačně identifikovat. Příklad 14 to názorně ilustruje.

Mějme krabičku *Mapování ontologie*, do ní zapojme krabičku *Databáze* a jednu krabičku *Ontologie*. Načteme do krabičky *Ontologie* nějakou ontologii a namapujeme sloupce z databáze na entity ontologie a uložíme mapování.

Nyní do krabičky *Mapování ontologie* zapojme druhou krabičku *Ontologie*, v níž načteme stejnou ontologii jako v první krabičce.

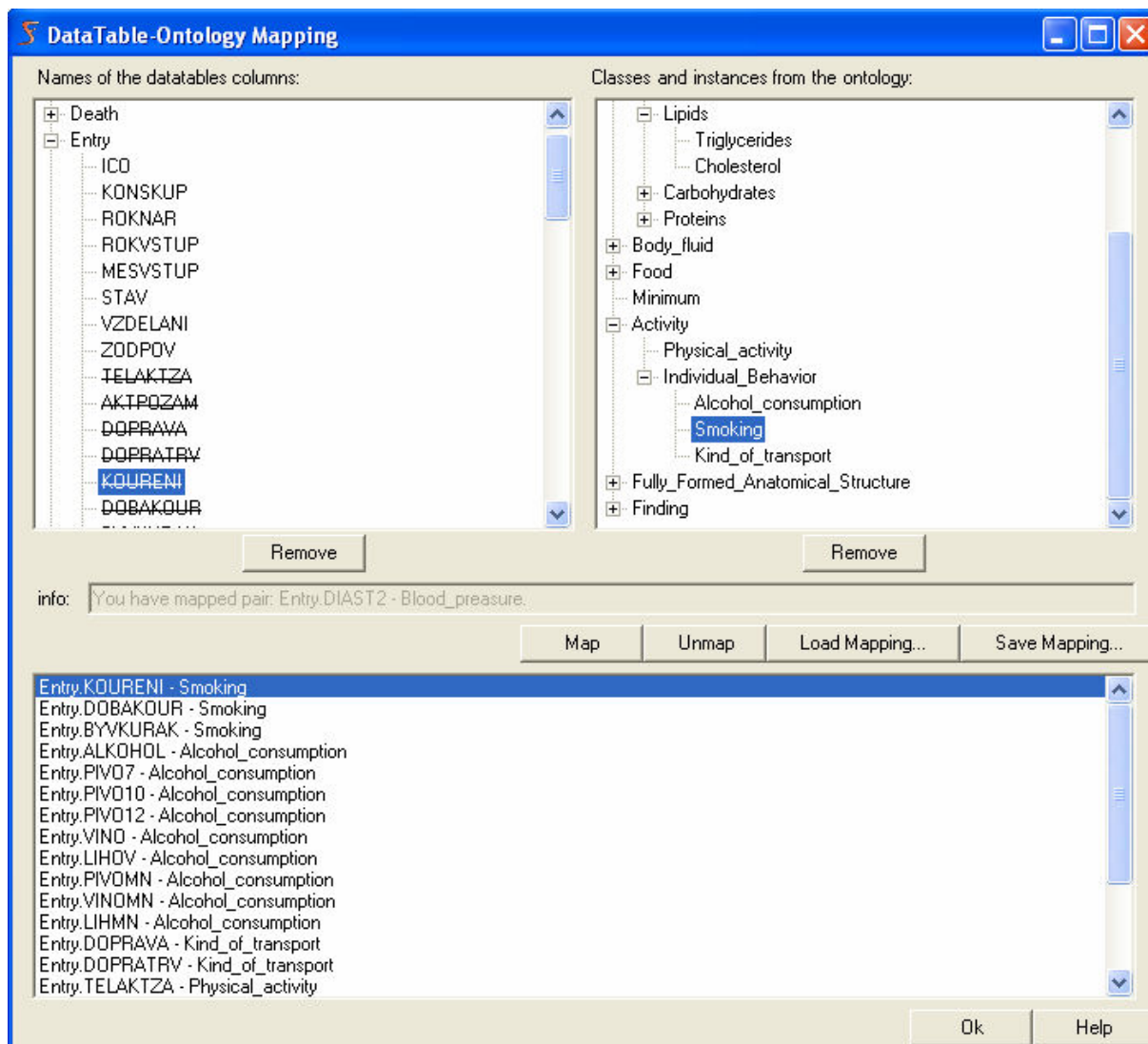
Nastává tak problém, že krabička *Mapování ontologie* nemůže jednoznačně rozhodnout, k jaké entitě ontologie se váže jaký sloupec. Dokonce ani žádný identifikátor ontologie (např. *URI*) nelze použít, jelikož jsou ontologie z tohoto pohledu taktéž shodné. Bylo by nutné ošetřit, aby krabička *Mapování ontologie* umožňovala zapojení pouze jedné krabičky se shodným *URI*, a navíc aby byly jednotlivé mapované entity ontologie identifikovány kromě jejich názvu také identifikátorem ontologie.

Tato možnost byla nakonec zavržena z důvodu poměru pracnosti implementace a přínosu, který by přinesla. Lze ji totiž snadno simulovat dvěma krabičkami *Mapování ontologie*, do kterých bude zapojena vždy jedna ontologie. Navíc lze předpokládat, že při dobývání znalostí bude analytik zkoumat data z jedné oblasti, která bude obvykle popsána jednou ontologií.

Příklad 14: Zapojení dvou krabiček *Ontologie* do jedné krabičky *Mapování ontologie*

Jak již bylo zmíněno, krabička *Mapování ontologie* zprostředkovává informace z ontologie pro další krabičky. První a prozatím jedinou implementovanou funkcí je vrácení rozšiřujících informací z ontologie, které se váží k příslušnému sloupci (jeho identifikátor, tzn. název tabulky a sloupce, je zadán jako vstupní parametr dané funkce). Krabička *Mapování ontologie* pomocí mapování zjistí, zda je daný sloupec namapovaný. Pokud ano, tak na jakou entitu. Z ontologie poté načte odpovídající hodnoty a pošle na výstup dotazující se krabičce. Pokud sloupec není namapovaný, je výstup stejný, jako by byl sloupec namapovaný na entitu, která nemá vyplněné vlastnosti (to jsou jednak třídy bez vyplněných vlastností a jednak jsou to prozatím všechny instance¹).

Obrázek 15 zobrazuje dialog, který slouží k vytvoření mapování. Podrobně je tento modul pro nastavení popsán v nápovědě systému Ferda. Zde zmíním jen funkce tlačítek *Save Mapping...* a *Load Mapping...* Tyto tlačítka umožňují znovuvyužití vytvořeného mapování. Mapování se pro jednoduchost ukládá v textovém formátu a přípona souborů s uloženým mapováním je *fom* (*Ferda Ontology Mapping*).



Obrázek 15: Modul pro nastavení mapování

¹ Rozšiřující informace o entitách lze zatím uchovávat pouze u tříd, objasnění tohoto omezení je popsáno v kapitole 3.5.

Krabička *Mapování ontologie* obsahuje jeden modul pro interakci, který slouží k prohlížení vytvořeného mapování. Kromě názvů namapovaných sloupců (a jejich tabulek) a entit ontologie jsou tam také rozšiřující informace o entitě ontologie a také anotace z ontologie k jednotlivým entitám. Ty umožňují uživateli porozumět obsahu jednotlivých sloupců tabulek databáze (viz kapitola 2.1.2).

Krabička *Mapování ontologie* obsahuje kromě *modulu pro nastavení mapování* ještě *modul pro nastavení primárních klíčů*. Nastavení primárních klíčů tabulek je potřebné pro správné fungování GUHA procedur a také pro řadu modulů pro interakci v různých krabičkách. V klasickém řetízku krabiček, sestaveném z původních krabiček Ferdy, je primární klíč nastavován krabičkou *Sloupec*, jelikož však obdobná krabička v řetízcích krabiček umožňujících využití ontologií není (viz kapitola 2.2.1), zajišťuje tuto funkčnost krabička *Mapování ontologie*. Tato krabička však není přiřazena k jednomu konkrétnímu sloupci, a tak umožňuje nastavení primárních klíčů všech tabulek databáze.

V průběhu implementace bylo rozhodnuto, že bude krabička *Mapování ontologie* zajišťovat také část funkčnosti, kterou požaduje PVO Vytvoření a využití skupin příbuzných atributů (kapitola 2.2.4). Smyslem tohoto PVO je seskupení atributů do skupin významově příbuzných atributů, a této příbuznosti dále využít při konstrukci úlohy DZD. Krabička *Mapování ontologie* zajišťuje první část popsané myšlenky: identifikování příbuzných atributů, resp. příbuzných sloupců, z nichž se následně příbuzné atributy vytvoří.

Příbuznost atributů je určována na základě ontologie a za příbuzné sloupce jsou označeny jednak sloupce, které jsou namapovány na stejnou entitu z ontologie a jednak takové sloupce, jímž odpovídající entity v ontologii mají společnou nadřídu.¹ Krabička poté nabízí na vytvoření jednotlivé skupiny příbuzných sloupců (pokud atribut nemá žádné příbuzné sloupce, pak se jedná o skupinu s jediným prvkem). Další využití příbuznosti sloupců, potažmo atributů zatím pouze na analytikovi. Možnosti využití této příbuznosti byly naznačeny v kapitole 2.2.4 a rozšíření podpory v systému Ferda bylo navrženo jako námět pro další práci v kapitole 6.2.2.


4.3. Krabička *Sloupec podporující ontologie*

Tato krabička zajišťuje stejnou funkčnost, jako původní krabička Ferdy *Sloupec (Column)*². Liší se pouze ve vstupních zásuvkách. Důsledkem toho, že do zásuvky této nové krabičky je zapojena krabička *Mapování ontologie*, která je zapojena přímo do krabičky *Databáze*, je to, že v krabičce *Sloupec podporující ontologie* je kromě sloupce potřebné nastavit také tabulku, z níž se daný sloupec vybírá³. Zjednodušeně by se dalo říci, že role této krabičky je pouze ve zprostředkování informací mezi krabičkami, které jsou zapojeny před ní a krabičkami, zapojenými za ní. Specifickou vlastností je to, že dokáže předat informace z ontologie, které určují charakter datového sloupce, do krabičky *Atributu odvozeného z ontologie*. Kromě této role poskytuje krabička *Sloupec podporujícího ontologie* uživateli možnost prohlédnout si data vybraného sloupce tabulky.

¹ Zatím je podporována pouze přímá nadřída entit, ale lze uvažovat také o možnosti příbuznosti sloupců, které jsou namapovány na libovolnou entitu z podstromu nějaké třídy – tyto úvahy jsou úzce spjaté s myšlenkami využití skupin příbuzných atributů pro automatizovanou konstrukci úlohy a jsou ponechány jako námět na další práci v kapitole 6.2.2.

² Pro pochopení úlohy původních krabiček ve Ferdovi doporučuji tutoriál ([ferdaTutorial.html](#)) k aplikaci Ferda, který je součástí uživatelské dokumentace. Krabička *Sloupec podporující ontologie* implementuje stejné rozhraní jako krabička tabulka *Sloupec*.

³ Při vytvoření krabičky z *Mapování ontologie* prostřednictvím krabiček nabízených na vytvoření se hodnoty tabulky i sloupce vyplní automaticky.

ikona krabičky	
název krabičky	<i>Sloupec podporující ontologie (Ontology Enabling Column)</i>
vstupní zásuvky	<ul style="list-style-type: none"> • <i>Mapování ontologie (Ontology Mapping)</i>
parametry krabičky	<ul style="list-style-type: none"> • <i>Vybraná Tabulka (DataTable)</i> • <i>Vybraný sloupec (SelectExpression)</i> • další parametry popisující data obsažená v sloupci tabulky
moduly pro nastavení	<ul style="list-style-type: none"> • <i>Výběr řetězce z pole řetězců (Multi Select Strings)</i>
akce krabičky	---
moduly pro interakci	<ul style="list-style-type: none"> • <i>Modul pro zobrazení frekvencí (Frequency displayer module)</i>
krabičky nabízené na vytvoření	<ul style="list-style-type: none"> • <i>Atribut odvozený z ontologie (Ontology Derived Attribute)</i> • další krabičky reprezentující atribut

Tabulka 5: Vlastnosti krabičky Sloupec podporující ontologie

Z vlastností krabičky jsou zajímavé zejména parametry *Vybraná Tabulka* a *vybraný sloupec*. Ty jsou nastavovány prostřednictvím modulu, který nabízí možné volby na základě databáze. První parametr nabízí na výběr všechny tabulky z databáze, druhý parametr pak sloupce vybrané tabulky. Tato krabička umožňuje, v rámci krabiček nabízených na vytvoření, generovat stejné krabičky, které nabízí na vytvoření krabička *Sloupec* (jedná se o krabičky pro kategorizaci atributů např. *Ekvifrekvenční atribut*, *Atribut Each value one category*, atd.), navíc však *Sloupec podporující ontologie* nabízí na vytvoření krabičku *Atributu odvozeného z ontologie*.

4.4. Krabička Atribut odvozený z ontologie

Úlohou této krabičky je kategorizace atributu na základě informací načerpaných z ontologie. Seznam typů informací, které mohou být uloženy v ontologii (tedy seznam možných charakteristik entit ontologie) byl promyšlen během návrhu PVO Automatická tvorba atributu a je popsán v kapitole 2.2.3. Jak tyto informace do ontologie vložit popisuje kapitola Příprava ontologie.

Již při návrhu uvedeného seznamu bylo přihlíženo k tomu, aby bylo možné na základě příslušných hodnot jednoduše a hlavně jednoznačně kategorizovat atribut. Tabulka 6 zjednodušeně popisuje v pseudojazyce algoritmus použitý pro uvedenou kategorizaci.

```

IF (kardinalita = nominální OR kardinalita = ordinálně cyklická) THEN {
    kategorizuj atribut stylem Each Value One Category;
}
ELSE {
    urči doménu hodnot;
    /* Najdi minimum domény neboli minimum z následující množiny:
    * [nejnižší hodnota uvedená v datech, parametr Minimum,
    * Hodnoty rozdělující doménu, Významné hodnoty]1.
    * Obdobně najdi maximum domény.
    */
    vytvoř z domény jeden velký interval;
    IF (existuje aspoň jedna Hodnota rozdělující doménu) THEN {
        rozděľ původní interval na intervaly podle Hodnot rozdělujících doménu
    }
    IF (existuje aspoň jedna významná hodnota) THEN {
        vytvoř pro každou Významnou hodnotu samostatnou kategorii2
    }
}

```

Tabulka 6: Algoritmus pro kategorizaci atributu odvozeného z ontologie


Velkou výhodou tohoto algoritmu je jeho jednoduchost. Analytik má díky tomu velmi přesnou představu o tom, jak se vytvoří kategorie atributu pro jednotlivé vstupní hodnoty. Tato jednoduchost také usnadňuje určování odpovídajících hodnot daných vlastností pro entity v ontologii.³

Původní návrh krabičky předpokládal, že budou parametry obsahující informace z ontologie určené pouze pro čtení. Pro snadnější testování algoritmu kategorizace byly parametry vytvořeny s možností zápisu. A po úvaze a konzultacích s vedoucím diplomové práce tak byly ponechány. Důsledkem tohoto rozhodnutí je velice zajímavé rozšíření využitelnosti této krabičky, které vedlo až k myšlence univerzální krabičky pro kategorizaci atributu, která by v sobě integrovala možnosti všech ostatních kategorizačních krabiček. Konkrétní řešení takové krabičky popisují v kapitole 6.2.1 která je podkapitolou námětů na další práci.

¹ Algoritmus je ve skutečnosti složitějších, například zahlásí chybu, pokud je některá z *významných hodnot* menší než parametr *Minimum* apod.

² Tato hodnota je pochopitelně vyjmuta z intervalu, kam původně patřila

³ Nedostatkem daného algoritmu je prozatím to, že z ordinálních hodnot vytváří automaticky intervaly – uvážíme-li např. ukázkový příklad ordinální škály z kapitoly 2.2.3, tedy škálu špatný-dobry-lepši-nejlepší, pak je zřejmé, že by bylo vhodnější použít kategorizaci, která pro každou hodnotu vytvoří samostatnou kategorii. Navíc nelze pomocí datového typu popsat uspořádání takových hodnot. Sofistikovanější expertní systém by mohl *ordinální* atribut kategorizovat na základě rozsahu hodnot v tomto sloupci. Při malém rozsahu hodnot by se poté kategorizovalo stylem *Each value one category*, v případě většího rozsahu by se pak s daným atributem nakládalo stejně jako s *kardinálním atributem*.

ikona krabičky	
název krabičky	<i>Atribut odvozený z ontologie (Ontology Derived Attribute)</i>
vstupní zásuvky	<ul style="list-style-type: none"> • <i>Sloupec podporující ontologie (Ontology Enabling Column)</i>
parametry krabičky	<ul style="list-style-type: none"> • <i>Kardinalita (Cardinality)</i> • <i>Minimum</i> • <i>Maximum</i> • <i>Hodnoty rozdělující doménu (Domain Dividing Values)</i> • <i>Významné hodnoty (Distinct Values)</i>¹ • další parametry obdobné jako u ostatních krabiček pro kategorizaci atributů
moduly pro nastavení	---
akce krabičky	<ul style="list-style-type: none"> • <i>Načti hodnoty parametrů z ontologie (Reload properties from ontology)</i>
moduly pro interakci	<ul style="list-style-type: none"> • <i>Modul pro zobrazení frekvencí (Frequency displayer module)</i>
krabičky nabízené na vytvoření	<ul style="list-style-type: none"> • <i>Pevný atom (Fixed Atom)</i> • <i>Nastavení atomu (Atom Setting)</i> • <i>Statický atribut (Static Attribute)</i>

Tabulka 7: Vlastnosti krabičky Atribut odvozený z ontologie

Z vlastností krabičky *Atributu odvozeného z ontologie* bych rád zmínil některé z dalších parametrů obdobných jako u ostatních krabiček pro kategorizaci. Je to především parametr *Uzavřený ze strany (Closed From)*, který určuje, z jaké strany budou uzavírané generované intervaly v kategoriích. Dále je to parametr *Počet kategorií (Number of categories)*, který zobrazuje počet vytvořených kategorií a je určen pouze pro čtení, na rozdíl od krabiček *Ekvidistančního* a *Ekvifrekvenčního atributu*, kde slouží jako vstupní parametr pro kategorizaci a určuje, kolik výsledných kategorií chceme vytvořit. Zároveň u krabičky *Atributu odvozeného z ontologie* nejsou parametry na omezení domény známé z ostatních krabiček. Zde je možné doménu omezit parametry *Minimum* a *Maximum*.

Z dalších vlastností krabičky stojí za zmínku akce načtení hodnot parametrů z ontologie. Tato akce nastaví parametry na původní nastavení, které odpovídá hodnotám

¹ Parametry *Hodnoty rozdělující doménu* a *Významné hodnoty* jsou zatím implementovány jako řetězce a jednotlivé hodnoty musí být odděleny oddělovačem čárka a mezera „,“. V budoucnu budu tyto hodnoty vyplňovány pomocí nějakého sofistikovanějšího modulu pro nastavení.

v ontologii. Jestliže nejsou hodnoty v ontologii vyplněny, nastaví se všechny parametry na prázdné hodnoty a výčtový parametr *Kardinalita* je nastavena na defaultní hodnotu *Nominal*.

V krabičce se nabízejí na vytvoření stejné krabičky jako v ostatních krabičkách pro kategorizaci atributů. Z nabízených krabiček je z pohledu kategorizace atributu nejzajímavější krabička *Statický atribut*, v níž je možné vytvořené kategorie v předchozí krabičce dále manuálně upravovat.

5. Experimentální část

Pro otestování uvedených implementovaných krabiček byla zvolena data projektu STULONG a analytická otázka 13.c (viz Příklad 1). Součástí Přílohy 1 této práce je uložený projekt ve Ferdovi, který obsahuje tuto DZD úlohu.

Experimentální část se dá rozdělit do čtyř částí. Příprava ontologie, mapování datových sloupců na entity ontologie, kategorizace atributů a sestavení úlohy.

Tato kapitola diplomové práce může sloužit také jako návod pro uživatele Ferdy, kteří chtějí využít výhod nově implementovaných krabiček.

5.1. Příprava ontologie

Pro reprezentaci ontologií byl zvolen jazyk OWL (viz kapitola 3). Používaným nástrojem pro tvorbu ontologií byl nástroj Protégé [24]. Důvody proč byl jako nástroj pro tvorbu a úpravu ontologií zvolen systém Protégé jsou zejména jeho uživatelská přívětivost a schopnost převádět ontologie mezi několika různými ontologickými jazyky. Navíc k této volbě přispěl také fakt, že Protégé bylo používáno v pracích, na něž tato diplomová práce navazuje, a některé výstupy těchto prací, které jsou v této práci využívány (např. ontologie *umls_stulong* z práce [15]), jsou v nativním formátu Protégé (soubory ontologie s koncovkou *.pont* a *.pins*).

Vytvořené krabičky umožňují využít stávajících ontologií v jazyce OWL. Aby však bylo možné využít všech možností, které tyto krabičky nabízejí, je nutné ontologie obohatit o rozšiřující informace charakterizující jednotlivé entity ontologie (o těchto rozšiřujících informacích podrobněji pojednávají kapitoly 2.2.3 a 4.4). Proto byl v rámci této diplomové práce vytvořen návod, jak tyto informace do ontologie přidat. Tento návod je také součástí Přílohy 1 této diplomové práce. Zároveň byla vytvořena šablona pro tvorbu nových ontologií, podporujících využití rozšiřujících informací ve Ferdovi. Jedná se o prázdnou ontologii s vytvořenými strukturami pro zachycení rozšiřujících informací. Tato šablona je Přílohou 4 této diplomové práce.

5.1.1. Ontologie UMLS_Ferda

Pro experimentální část této diplomové práce byla vytvořena ontologie UMLS_Ferda. Tato ontologie je odvozena z ontologie UMLS_2¹, vytvořené v rámci práce [15]. Oproti původní ontologii se nově vytvořená ontologie liší ve třech bodech.

- **formát reprezentace ontologie** – původní ontologie UMLS_2 byla uložena v nativním formátu Protégé. Z tohoto formátu byla ontologie exportována do jazyka do formátu RDF/XML (jako jazyk exportu byl zvolen OWL DL).

¹ V práci [15] je tato ontologie označována jako UMLS_2, ale název souboru s touto ontologií je *umls_stulong*. Tato ontologie je Přílohou 2 této diplomové práce.

- **rozšiřující informace o entitách ontologie** – ontologie UMLS_Ferda byla dále obohacena o definici rozšiřujících informací k entitám ontologie.
- **rozšíření třídy *blood_pressure*** – aby bylo možné ukázat všechny možnosti nově implementovaných krabiček, byla třída ontologie *blood_pressure* rozdělena na podtřídy *systolic_blood_pressure* a *diastolic_blood_pressure*, u kterých je možné stanovit *hodnoty rozdělující doménu*. K tomuto kroku se váží mj. Příklad 3 a Příklad 6.

U entit, které byly použity k experimentům popsaným v této kapitole, byly vyplněny relevantní hodnoty rozšiřujících informací. Tabulka 8 znázorňuje tyto hodnoty pro vybrané entity.

	<i>Cardinality</i>	<i>Minimum</i>	<i>Maximum</i>	<i>Domain Dividing Values¹</i>	<i>Distinct Values</i>
<i>Systolic_blood_pressure</i>	<i>Cardinal</i>			120, 130, 140, 160, 180	
<i>Diastolic_blood_pressure</i>	<i>Cardinal</i>			80, 85, 90, 100, 110	
<i>Smoking</i>	<i>Ordinal</i>				
<i>Kind_of_transport</i>	<i>Nominal</i>				
<i>BMI</i>	<i>Cardinal</i>			25	

Tabulka 8: Hodnoty rozšiřujících informací u vybraných entit ontologie UMLS_Ferda

5.2. Mapování sloupců datového zdroje na entity ontologie

V rámci této experimentální části bylo provedeno mapování sloupců, které se váží k jednotlivým pojmům z otázky STULONG 13.c. tedy ke kouření, tlaku a BMI. Postupoval jsem v podstatě obráceně, než by postupoval běžný uživatel. Já jsem ze znalosti dat v podstatě mapoval entity ontologie na sloupce. Bylo to tím, že jsem nemapoval celý datový zdroj, ale pouze jeho podmnožinu. Jinak bych nejspíše procházel sloupec po sloupci a snažil bych se ke každému jednotlivému sloupci nalézt odpovídající pojem v ontologii (což není tak obtížné, jelikož ontologie je obvykle dobře strukturována). Část vytvořeného mapování zobrazuje Tabulka 9. Soubor s obsáhlejším mapováním databáze STULONG na ontologii UMLS_Ferda je součástí Přílohy 1 této práce.

sloupec tabulky	entita ontologie
<i>KOURENI</i>	<i>Smoking</i>
<i>DOBAKOUR</i>	<i>Smoking</i>
<i>BYVKURAK</i>	<i>Smoking</i>
<i>SYST1</i>	<i>Systolic blood pressure</i>
<i>DIAST1</i>	<i>Diastolic blood pressure</i>
<i>SYST2</i>	<i>Systolic blood pressure</i>
<i>DIAST2</i>	<i>Diastolic blood pressure</i>
<i>VYSKA</i>	
<i>VAHA</i>	

Tabulka 9: Mapování vybraných sloupců tabulky *Entry* databáze STULONG na entity ontologie UMLS_Ferda

Při mapování sloupců datového zdroje na entity ontologie bude běžně docházet k situaci, kdy si sloupec a pojem z ontologie nebudou úplně odpovídat. Často bude jeden z mapovaných pojmů významově obecnější (viz např. entity *Smoking* a na ni mapovaných sloupců).

¹ Pro určení hodnot rozdělujících doménu pro systolický a diastolický tlak byla použita definice hypertenze z článku *Léčba hypertenze v ordinaci praktického lékaře* (https://www.zdravcentra.cz/cps/rde/xchg/zc/xsl/81_1301.html), který tuto definici přebíral ze směrnic SZO (Světové zdravotnické organizace) a ISH (Mezinárodní společnosti pro hypertenzi) [26].

Lze také uvažovat o mapování sloupců na nejbližší možnou entitu, pokud neexistuje žádná ani významově obecnější entita. Jako příklad poslouží sloupce *VYSKA* a *VAHA*. Ani pro jeden z nich neexistuje v ontologii odpovídající entita. Nabízí se možnost namapovat je na významově nejbližší entitu, kterou je v tomto případě *BMI*. Takové mapování by však bylo chybné. Mapování je potřeba chápat tak, že s každým jednotlivým sloupcem namapovaným na určitou entitu může být nakládáno jako s danou entitou. V případě sloupců *VYSKA* a *VAHA* a entity *BMI* to však tak není. Jednotlivé sloupce zdaleka nevyjadřují význam *BMI*, samy ne, pouze v kombinaci a k tomu mapování určeno není¹.

Namapování sloupce na obecnější entitu ontologie není na škodu, ale je zde ještě možnost rozšíření ontologie, která jak je ukázáno dále, může být v řadě případů velmi užitečnou (zde záleží především na tom, zda máme právo do ontologie zapisovat a zda má skutečně rozšíření pojmů v ontologii obecný význam v dané oblasti).

Toto rozšíření bylo pro ukázkou provedeno u entity *Blood_pressure*, aby bylo možné názorně předvést pozdější kategorizaci. Tato třída byla tedy rozdělena dále na *Systolic_blood_pressure* a *Diastolic_blood_pressure*. Kromě toho, že je možné atributy odvozené z těchto entit lépe kategorizovat, má toto přesnější mapování ještě jeden přínos.

Uvažujme situaci, kdy například jeden analytik ve spolupráci s doménovým expertem vytvoří mapování. Toto mapování uloží, aby ho mohli využít také další analytici, kteří se budou podílet na hledání znalostí. Tito analytici zatím neznají zkoumanou oblast ani datový zdroj. Vstupem pro ně bude pouze ontologie, datový zdroj a mapování. Pomocí krabiček ve Ferdovi si tyto vstupy správně propojí a pomocí modulu pro interakci *Zobrazení mapování* si prohlédnou mapování. Díky tomu mohou získat poměrně dobrou představu jak o zkoumané oblasti, tak o datovém zdroji. A pokud mají navíc k dispozici analytické otázky, které je potřeba zodpovědět, mohou kvalifikovaně provádět samotný datamining. A to až do okamžiku nalezení silných hypotéz. Při snaze interpretovat hypotézy, by analytici mohli dobře využít ontologií, ale jestliže jsou sloupce namapovány na obecnější pojmy nebo jsou si mapovaný sloupec a entita pouze příbuzné, nedokáží analytici konkrétní hypotézu přesně interpretovat. Tuto myšlenku ilustruje Příklad 15.

Představme si, že analytik objeví závislost v datech mezi hodnotami sloupce *DOBAKOUR* a *SYST1*. Pomocí ontologie a námi vytvořené ontologie a bez další znalosti datového zdroje dokáže takovýto vztah interpretovat jen takto: „*Jeden z parametrů kouření ovlivňuje systolický tlak.*“ Pro rozhodnutí, který z parametrů to je, potřebuje konzultovat výslednou hypotézu s někým, kdo zná blíže datový zdroj.

Pokud by však v ontologii existovala podtřída *Time Smoking* třídy *Smoking* a sloupec *DOBAKOUR* by na ni byl namapován, pak by analytik mohl přesně říci, že našel v datech závislost mezi dobou kouření a systolickým tlakem.

Příklad 15: Mapování sloupců na ne zcela odpovídající entity

5.3. Kategorizace atributů

Fáze kategorizace atributů je stejně jako interpretace hypotéz, jak ji popisuje Příklad 15, závislá na přesnosti mapování. Tuto skutečnost znázorňuje Příklad 16.

¹ Pokud bychom umožnili takovéto mapování, dostali bychom se do velkých problémů při kategorizaci atributu. V uvažovaném případě mapování sloupců *VYSKA* a *VAHA* na *BMI* by se odvozené atributy snažily kategorizovat jako *BMI*, což je pochopitelně zcela nepřesné.

Uvažujme kategorizaci atributu vzešlého ze sloupce *SYST1*, namapovaného na entitu systolický tlak (*Systolic_blood_pressure*). Díky přesnému mapování a vyplněným hodnotám rozdělujících doménu se nám podaří přesně rozdělit atribut na kategorie, které jsou v oboru (zdravotnictví) jednoznačně interpretovatelné.

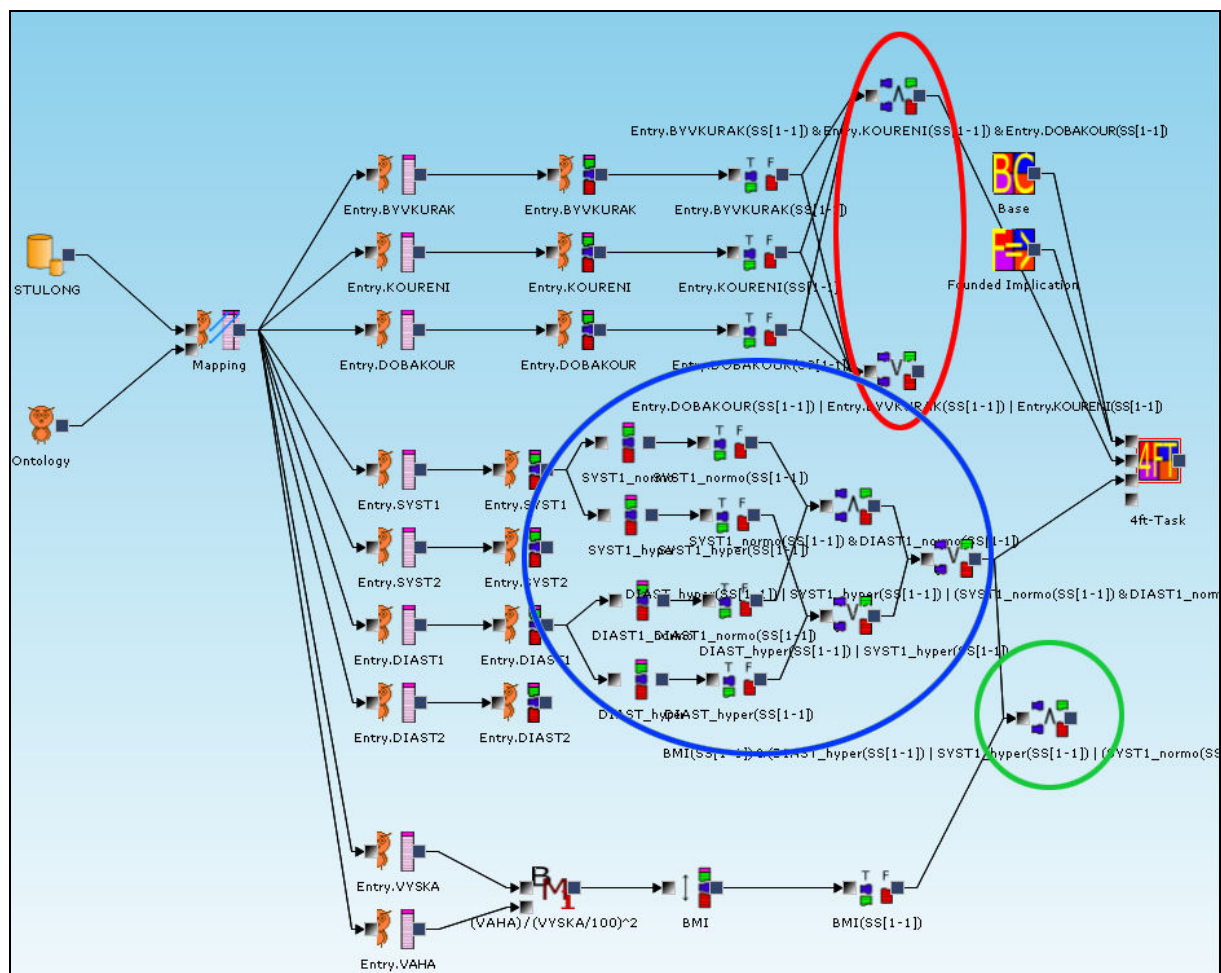
Pokud by však v ontologii neexistovala třída *Systolic_blood_pressure* a sloupec *SYST1* by byl namapován na obecnější entitu krevní tlak (*Blood_pressure*), pak takováto entita nemá jednoznačně dané hodnoty rozdělující doménu a informací, kterou by z ní atribut mohl získat, je očekávaná sémantika atributu. *Blood_pressure* říká, že sémantika je kardinální. Analytik tedy může uvažovat o *ekvidistanční* nebo *ekvifrekvenční* kategorizaci atributu, lepší členění nedokáže bez znalosti zkoumané oblasti vytvořit.

Systolický tlak je ideální případ atributu, který přímo v ontologii obsahuje popis kategorizace pomocí definovaných hodnot rozdělujících doménu. V případě sloupců týkajících se kouření už není přesně daná kategorizace. Entita z ontologie nám říká, že předpokládané hodnoty reprezentující danou třídu budou mít ordinální škálu. Na základě této informace lze uvažovat o *ekvidistančním* nebo *ekvifrekvenčním* rozdělení atributu. Bohužel vzhledem k návrhu databáze STULONG jsou obsažené hodnoty již kategorizovány – např. hodnota 4 ve sloupci kouření reprezentuje tvrzení, že pacient kouří 15–20 cigaret za den. Tady je tedy ze znalosti dat vhodnější změnit kardinalitu na nominální a vytvořit tak pro každou hodnotu samostatnou kategorii.

Příklad 16: Kategorizace atributů

5.4. Sestavení úlohy

Celou úlohu znázorňuje Obrázek 16. Úloha je ve formátu projektu Ferdy součástí Přílohy 1 této práce.



Obrázek 16: Úloha DZD - STULONG 13.c.

Modrý ovál zobrazuje vytvoření booleovského atributu, který určuje, zda je pacient hypertonik, či normotonik. Názorně je zde ukázána užitečnost krabiček disjunkce a konjunkce. Bez nich by nebylo možné vymodelovat skupinu hypertoniků.

Zelený ovál reprezentuje rozdělení pacientů na jednotlivé zkoumané skupiny, tj. normotonici s nadváhou, normotonici bez nadváhy, hypertonici s nadváhou a hypertonici bez nadváhy. Bohužel se ukázalo, že při zapojení krabičky *BMI*, do GUHA procedury nelze procházet výsledky běhů procedury¹, a tak jsem byl nucen se omezit na podotázku otázky STULONG 13.c.: „*Jaký je vliv kouření na hypertenzi?*“

Nejzajímavější částí úlohy z pohledu této diplomové práce je červený ovál, který obsahuje krabičky konjunkce a disjunkce, které využívají příbuznost atributů a vytvářejí z nich odvozené atributy. Tyto dvě krabičky (konjunkce a disjunkce) nejsou všemi možnostmi, jak lze příbuzné atributy propojit. V úvahu přichází ještě vytvoření různých kombinací, v nichž budou vystupovat negace původních booleovských atributů, ale z důvodu přehlednosti úlohy byly vynechány. Námětem na rozšíření automatizace tohoto zapojení je krabička *Skupina příbuzných atributů*, popsána v kapitole 2.2.4.

¹ Jedná se o technickou chybu, ale jelikož je BMI pouze ukázková krabička, napravení dané chyby nemá příliš velkou prioritu.

5.5. Zhodnocení experimentální části

Mapování bylo poměrně rychlé a jednoduché, možná je to dáno tím, že se již v datech dobře orientují, ovšem na stejné úrovni znalosti dat je obvykle také doménový expert, který by mohl mapování vytvořit.

Jednorázové vytvoření všech příbuzných sloupců z krabičky *Mapování ontologie* je velice příjemné. Co poněkud zdržovalo, bylo vytváření krabiček pro kategorizaci atributů. Bylo by lepší, aby se vytvořila krabička pro kategorizaci atributu rovnou z krabičky *Mapování*, ale jak bylo navrženo v kapitole 2.2.3 a podrobněji bude popsáno v kapitole 6.2.1, lze uvažovat o přetvoření atributu odvozeného z ontologií v univerzální kategorizační atribut. Poté by bylo možné krabičku *Sloupce podporujícího ontologie* zcela vynechat, čímž by byl zmíněný nedostatek vyřešen.

Po vytvoření krabiček *Atributů odvozených z ontologie* následoval proces kategorizace těchto atributů. U atributů, které obsahovaly popis kategorizace přímo v ontologii (systolický a diastolický tlak), proběhla kategorizace automaticky, což je výrazný přínos celé této diplomové práce. U ostatních atributů nám ontologie poskytla radu (v podobě typu kardinality), jak atributy kategorizovat, takže i zde bylo pro kategorizaci možné využít znalostí z ontologie.¹

Pro sestavení úlohy bylo využito příbuznosti atributů. Určení skupin příbuzných atributů (resp. sloupců) zajišťuje krabička *Mapování ontologie*. Uživatel pozná příbuzné sloupce podle toho, že je krabička *Mapování ontologie* nabízí na vytvoření v rámci jednoho příkazu. Prozatím si uživatel tyto skupiny po vytvoření krabiček *Sloupců podporujících ontologie* musí ohlídat sám a v rámci pracovní plochy si je například seskupit k sobě, ale do budoucna je uvažováno o dalším automatizovaném využití těchto skupin. Návrhy na další využití skupin příbuzných atributů jsou v této práci popsány v kapitole 2.2.4 a jsou zároveň jedním z námětů pro další rozvoj využití ontologií ve Ferdovi (6.2.2). Pro zvýšení přehlednosti by také pomohla existence „nadkrabiček“ (6.2.4).

Závěrem pro úplnost doplním výsledky dataminingové úlohy, jejímž cílem bylo zodpovědět otázku STULONG 13.c, resp. její podotázku (viz předchozí kapitola). Na základě vstupních dat se ukázalo, že vliv kouření na hypertenzi je zanedbatelný. Nejsilnější hypotéza (splňující podmínku *Base*) byla platná v datech s určitostí pouhých 61%. Jen pro zajímavost, touto hypotézou bylo tvrzení, že pacienti, kteří kouří déle než jedenáct a méně než dvacet let jsou normotonici (netrpí hypertenzí).²

5.6. ADAMEK – druhá úloha experimentální části

Vzhledem ke zmíněnému problému ohledně již kategorizovaných hodnot ve sloupcích týkajících se kouření, bylo využití ontologií otestováno ještě na jiném zdroji dat a to databázi ADAMEK (Aplikace DATového Modelu Euromise – Kardio [18]). Tato databáze obsahuje stejně jako databáze STULONG data z oblasti zdravotnictví. Díky tomu je možné použít stejnou ontologii UMLS_Ferda. Sloupce jsou v databázi ADAMEK pojmenovány jinak než

¹ Ve skutečnosti toto není úplně pravda, což ukazuje Příklad 16 a sloupce týkající se kouření, ale toto je dáno poněkud návrhem databáze STULONG (návrh vznikl v roce 1976, a tak příliš nepřekvapuje, že neodpovídá moderním požadavkům na datový návrh). U novějších databází by k takovýmto problémům docházet nemělo. To se také prokázalo při modelování nad databází ADAMEK, kde již mělo smysl sloupce týkající se kouření kategorizovat např. *ekvifrekvenčně*, jelikož počet vykouřených cigaret znamenal skutečný odhad vykouřených cigaret.

² Skutečnost, že právě tato hypotéza je nejsilnější (i když přesto slabá), by se dala zdůvodnit například tím, že lidé, kteří kouří po uvedenou dobu jsou obvykle mladí lidé (20–35 let) a vliv věku na hypertenzi bude pravděpodobně významnější než vliv kouření.

v databázi STULONG, je tedy nutné vytvořit nové mapování (pokud by aspoň nějaké sloupce odpovídaly, bylo by možné jejich mapování převzít s předešlých mapování).

Snahou bylo na datech databáze ADAMEK ověřit výstup z předchozí DZD úlohy, tedy ověřit, že kouření nemá vliv na hypertenzi.

Při sestavování úlohy jsme vycházeli přímo z předešlé úlohy, v krabici *Databáze* jsme nahradili databázi STULONG databází ADAMEK, vytvořili jsme mapování a následně vytvořili sloupce namapované na entitu *Smoking* a sloupce namapované na entitu *Blood pressure*. Krabice z předešlé úlohy jsme ani nemuseli mazat, smazali jsme pouze krabice *Sloupců podporujících ontologie*, abychom měli přehled o skupinách příbuzných atributů. Tyto nově vytvořené krabice pak už jen stačilo zapojit do krabiček *Atributů odvozených z ontologie*, které jsme si na ploše Ferdy nechali (ze sloupců týkající se kouření jsme vytvořili ekvifrekvenční atributy). Dále bylo potřeba znovu vytvořit krabice *Statických atributů*, ale jinak jsme celou strukturu mohli ponechat nezměněnu. Díky prostředí Ferda a využití ontologií trvala úprava úlohy pro nový datový zdroj necelé tři minuty.

Úloha zkoumající závislost kouření na hypertenzi nad databází ADAMEK je Přílohou 10 této práce. V datech bylo potvrzeno, že kouření nemá výrazný vliv na hypertenzi (nejsilnější hypotéza, v níž byl splněn antecedent, i sukcedent aspoň z 10%, byla platná v datech se spolehlivostí 55%).

6. Závěr

Cílem diplomové práce bylo rozvést návrhy postupů pro využití ontologií (PVO) při procesu DZD navržené v [2] a vybrané postupy implementovat. Snahou práce bylo prokázat, že ontologie mohou být pro dobývání znalostí velice užitečné, a připravit půdu pro další rozvoj využívání ontologií při dobývání znalostí (primárně v prostředí Ferda).

Vzhledem k rozsahu této práce bylo nutné vybrat pro implementaci z množiny navrhovaných PVO (viz kapitola 2) pouze její část. Při výběru PVO pro implementaci byl hlavním kritériem přínos konkrétního PVO pro uživatele (analytika), aby bylo možné názorně předvést využitelnost ontologií a motivovat tak k další aktivitě v oblasti podpory ontologií v prostředí Ferda.

PVO zvoleným pro implementaci a následně implementovaným byla Automatická tvorba atributů (viz kapitola 2.2.3). Nezbytným předpokladem implementaci tohoto PVO je existence PVO Mapování (2.2.1), který poskytuje jakési základní rozhraní pro jakékoliv využití ontologií. Implementace zvolených PVO sestávala z vytvoření čtyř nových krabiček, jejichž role jsou popsány v kapitole 4.

V kapitole 6.1 je shrnuta vykonaná práce, v kapitole 6.2 jsou poté vypsány klíčové náměty pro další práci

6.1. Shrnutí vykonané práce

Tato kapitola stručně shrnuje vykonanou aktivitu v rámci této diplomové práce.

Prvním krokem bylo seznámení se s uživatelským rozhraním systému Ferda, s procesem dobývání znalostí, s teorií vážící se k metodě GUHA a s teorií ontologií.

V dalším kroku byly důkladně prostudovány práce [2] a [15], na něž tato práce měla za úkol navázat. Byly detailně diskutovány jednotlivé návrhy pro využití ontologií z předchozích prací a byly identifikovány klíčové požadavky na jazyk pro reprezentaci ontologií.

Následovalo podrobné studium existujících jazyků pro reprezentaci ontologií a hledání možnosti, jak v některém z nich uchovat rozšiřující informace definované o entitách. Takovou možnost se nakonec podařilo najít v jazyce OWL.

Na řadu přišla implementace modulů (krabiček) v prostředí Ferda. Té předcházelo studium programátorské dokumentace systému Ferda. Při implementaci bylo nejnáročnější integrovat do prostředí Ferdy komponentu parseru jazyka OWL, která byla napsána třetí stranou v jazyce Java. Samotná implementace krabiček pak již nepřinesla zásadní problém, až na změnu návrhu zapojení krabičky *Mapování ontologie* v průběhu implementace.

Implementované krabičky byly následně otestovány na reálném příkladu, byl prokázán jejich přínos a byly navrženy možnosti, jak dále využít možností ontologií při procesu dobývání znalostí. V rámci této experimentální části práce vznikly dva užitečné výstupy: jednak návod na vytvoření ontologií, v nichž je možné uchovávat rozšiřující informace o entitách (Dodatek C), a jednak již hotová šablona pro vytvoření takovýchto ontologií (Příloha 4).

6.2. Náměty na další práci

Tato práce identifikuje čtyři významné směry, kudy by se mohl ubírat další vývoj v oblasti využití ontologií v procesu dobývání znalostí pomocí GUHA procedur a také další vývoj systému Ferda.

6.2.1. Univerzální krabička pro kategorizaci atributu

Prvním námětem pro další práci je vytvoření univerzální krabičky pro kategorizaci atributu. Tento námět vychází z implementace krabičky *Atributu odvozeného z ontologií* (viz kapitoly 2.2.3 a 4.4). Tím, že byla u této krabičky uživateli ponechána možnost nastavit jednotlivé parametry odvozené z ontologie, se možnosti této krabičky diametrálně rozšířily. Z této krabičky se stal velmi mocný nástroj pro kategorizaci atributu dle přání uživatele. Z možností, které nabízí ostatní krabičky zatím automatické vytvoření *ekvidistančního* a *ekvifrekvenčního* atributu. Pokud by byly tyto dvě možnosti doplněny a bylo by náležitě ošetřeno, aby tato krabička nevyžadovala jako předcházející krabičku *Ontologii*, mohla by krabička *Atribut odvozený z ontologie* (vhodněji přejmenovaná např. na *Atribut*) nahradit z hlediska funkčnosti všechny stávající krabičky pro kategorizaci atributu.

Řešení přidat možnost *ekvidistanční* a *ekvifrekvenční* kategorizace atributu by nejspíše byla realizována prostřednictvím přidání parametrů krabičky, ale možná by to vedlo ke snížení přehlednosti pro uživatele. V takovém případě by bylo vhodnější uvažovat o nějakém sofistikovanějším modulu pro nastavení kategorizace.

6.2.2. Automatická kombinace příbuzných booleovských atributů

Velice zajímavou oblastí pro další zkoumání možností využití ontologií při DZD je automatizace sestavení úlohy. Myšlenky pro využití ontologií v této fázi DZD jsou formulovány v kapitole 2.2.4.

6.2.3. „Fuzzy“ ontologie

Námět pro budoucí bádání v oblasti automatizace procesu DZD jsou tzv. „Fuzzy“ ontologie, tyto ontologie v sobě umožňují uchovávání neurčitých skutečností. Daná oblast nebyla v rámci této diplomové práce podrobněji zkoumána, tak nelze odpovědně říci, zda opravdu může znamenat nějaký přínos, či nikoliv.

V kapitole 2.3.3 a 2.3.4 jsou podrobněji rozepsány myšlenky, jak by se ontologií, do nichž by bylo možné zapsat neurčitá tvrzení, dalo využít pro automatizaci konstrukce úlohy DZD.

Doporučenými zdroji pro studium „Fuzzy“ ontologií jsou [30], [31] a [32].

6.2.4. „Nadkrabičky“

Poslední námět se netýká pouze využití ontologií, ale také prostředí Ferda jako takového. Myšlenkou „nadkrabiček“ je možnost sloučit větší množství jednodušších souvisejících krabiček do jednoho většího celku, který by byl na pracovní ploše v prostředí Ferda reprezentován jediným vizuálním prvkem, což výrazně zvýšilo přehlednost celého systému.

Reference

- [1] Kováč M., Kuchař T., Kuzmin. A, Ralbovský M.: Ferda, nové vizuální prostředí pro dobývání znalostí, Znalosti 2006, Sborník příspěvků 5. ročníku konference, Hradec Králové 2006, ISBN 80-248-1001-8
- [2] Ralbovský M.: Využití doménových znalostí při aplikacích GUHA procedur, Diplomová práce, Matematicko-fyzikální fakulta, Univerzita Karlova, Praha 2006
- [3] EUROMISE: Projekt STULONG, <http://euromise.vse.cz/stulong/index.php>
- [4] CRoss Industry Standard Process for Data Mining (CRISP-DM), <http://www.crisp-dm.org/>
- [5] Pracoviště VŠE, které je součástí Evropského centra pro medicínskou informatiku, statistiku a epidemiologii – Kardio, <http://euromise.vse.cz>
- [6] Hájek P., Havránek T.: Mechanising Hypothesis Formation – Mathematical Foundations for a General Theory, Springer, Berlin Heidelberg, New York 1978
- [7] Hájek P.: Metoda GUHA – současný stav, článek ve sborníku ROBUST 2002
- [8] Systém LISp-Miner: <http://lispminer.vse.cz>
- [9] Kováč M.: Uživatelsky orientovaný jazyk pro řešení úloh DZD, Diplomová práce, Matematicko-fyzikální fakulta, Univerzita Karlova, to appear
- [10] Kuchař T.: Experimentální GUHA procedury. Diplomová práce, Matematicko-fyzikální fakulta, Univerzita Karlova, Praha 2006
- [11] Gruber T.: It Is What It Does: The Pragmatics of Ontology as Language, Contract, and Content, <http://www.cs.man.ac.uk/~stevensr/workshop/gruber.zip>
- [12] Gruber T.: What is an ontology?, <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>
- [13] Gomez-Perez A., Fernandez-Lopez M., Corcho O.: Ontological Engineering: with the examples from the areas of Knowledge Management, e-Commerce and the Semantic Web, Springer-Verlag, London, Berlin, Heidelberg, ISBN 1-85233-551-3
- [14] Svátek, V.: Ontologie a WWW. Tutoriál ke konferenci Datakon 2002, <http://nb.vse.cz/~svatek/temata.htm>
- [15] Češpivová H.: Tvorba ontologií pro dobývání znalostí z databází, Diplomová práce, Fakulta informatiky a statistiky VŠE, Praha 2004
- [16] Češpivová H., Rauch J., Svátek V., Kejkula M., Tomečková M.: Roles of Medical Ontology in Association Mining CRISP-DM Cycle, ECML/PKDD04 Workshop on Knowledge Discovery and Ontologies (KDO'04), Pisa 2004
- [17] Svátek V., Rauch J., Ralbovský M.: Ontology-Enhanced Association Mining. In: Ackermann, Berendt (eds.). Semantics, Web and Mining, Springer-Verlag 2006
- [18] EUROMISE: Projekt ADAMEK, <http://www.euromise.cz/research/research.html>
- [19] Rauch J.: EverMiner – Architektura, interní dokument projektu LISp-Miner
- [20] Rauch J.: EverMiner – studie projektu, interní dokument projektu LISp-Miner
- [21] Projekt LM – Report Asistent, <http://reportasistent.berlios.de/>
- [22] The Internet Communications Engine (ICE), <http://www.zeroc.com/ice.html>

- [23] Stránka projektu Ferda, <http://ferda.sourceforge.net>
- [24] The Protégé Ontology Editor, <http://protege.stanford.edu/>
- [25] Kuzmin A.: Relační GUHA procedury. Diplomová práce, Matematicko-fyzikální fakulta, Univerzita Karlova, Praha 2007
- [26] 1999 World Health Organisation – International Society of Hypertension Guidelines for the Management of Hypertension. Guidelines Subcommittee. J Hypertension 1999, 17: 151–83
- [27] Kejkula M.: Self-Organized Data Mining – 20 Years after GUHA-80, prezentace na semináři KEG, <http://gama.vse.cz/keg/seminar/keg-sem.html>
- [28] Schutte S.: Automating the 4FT-Miner Set-Up: Two Example Approaches from the Toolbox of AI, prezentace na semináři KEG, <http://gama.vse.cz/keg/seminar/keg-sem.html>
- [29] Schutte S.: First considerations for applying AI techniques in LISp-Miner extension, Znalosti 2006, Poster na konferenci, Hradec Králové 2006
- [30] Smrž P., Vacura M., Šváb O.: Uncertainty ExtensionstoOntologiesasaToolforSemantic InterpretationinAudiovisualSystems, Proceedings of the 1st International Conference on Semantic and Digital Media Technologies, Poster and Demo, Athens, GR, 2006
- [31] Uncertainty Reasoning for the Semantic Web Workshop 2006, <http://www.iet.com/iswc/2006/ursw/index.html>
- [32] Uncertainty Reasoning for the Semantic Web Workshop 2007, <http://c4i.gmu.edu/ursw2007/index.html>
- [33] Cardoso, J, Semantic Web Services: Theory, Tools and Applications, <http://dme.uma.pt/jcardoso/Books/IDEA-SWTTA/>
- [34] The RAINBOW Project, <http://rainbow.vse.cz/>
- [35] Data Extraction Research Group, <http://www.deg.byu.edu/>
- [36] Project Ex information extraction system, <http://eso.vse.cz/~labsky/ex>
- [37] Labský, M., Nekvasil, M., Svátek, V.: Towards Web Information Extraction using Extraction Ontologies and (Indirectly) Domain Ontologies. Poster paper. In: Proc. 4th International Conference on Knowledge Capture, K-Cap 2007, Whistler, BC, Canada.
- [38] Web Ontology Language (OWL), <http://www.w3.org/2004/OWL/>
- [39] Resource Description Framework (RDF), W3C standard, <http://www.w3.org/RDF/>
- [40] The DARPA Agent Markup Language (DAML), <http://www.daml.org/>
- [41] TopicMaps.Org, <http://www.topicmaps.org>
- [42] Vatant B.: Ontology-driven topic maps, White Paper, Paris, France, 2003
- [43] Open Knowledge Base Connectivity (OKBC), <http://www-ksl.stanford.edu/software/OKBC/>
- [44] OWL 1.1 Web Ontology Language, <http://www.webont.org/owl/1.1/>
- [45] OntoTrack, <http://www.informatik.uni-ulm.de/ki/ontotrack/>
- [46] Jena – A Semantic Web Framework for Java, <http://jena.sourceforge.net/>
- [47] SWOOP, <http://code.google.com/p/swoop/>

Dodatek A: Slovníček pojmů

Tento dokument obsahuje přehled klíčových pojmů použitých v diplomové práci. V závorce je anglický ekvivalent českého názvu.

▪ Ferda

- **krabička (box)** – krabička je základní vizuální prvek systému Ferda. Krabičky se v závislosti na svém typu mohou zapojovat do řetízku. Prostřednictvím tohoto zapojení mohou spolu jednotlivé krabičky komunikovat. Propojením krabiček se sestavuje úloha DZD.
- **zásuvka krabičky (socket)** – zásuvka je vlastnost krabičky, která definuje typ krabiček, které se mohou do dané krabičky zapojit (které mohou předcházet dané krabičce). Krabička může mít 0 a více zásuvek a do každé zásuvky může být napojeno 1 či více krabiček určeného typu.
- **akce krabičky (action)** – akce krabičky zajišťuje nějakou funkčnost krabičky, kterou může uživatel explicitně vyvolat.
- **parametr krabičky (property)** – prostřednictvím parametrů krabičky uživatel upravuje chování dané krabičky, parametry určené pouze pro čtení informují uživatele o aktuálním stavu krabičky.
- **modul pro nastavení (setting module)** – modul pro nastavení slouží pro nastavení parametrů krabičky. Je používán především pro nastavení složitějších parametrů (např. pole řetězců, apod.)
- **modul pro interakci (module for interaction)** – modul pro interakci slouží k detailnějšímu zobrazení obsahu krabičky (např. u krabičky *Sloupec* existuje modul pro interakci, který zobrazí všechny hodnoty daného sloupce v datech a jejich frekvence).
- **krabičky nabízené na vytvoření (boxes asking for creation)** – pro usnadnění práce uživatele byl v prostředí Ferda implementován standardní postup, jak z jedné krabičky vytvořit krabičku jinou a spojit ji do řetízku s původní krabičkou. Tento postup se nazývá *krabičky na vytvoření*.

▪ Ontologie

- **ontologie** – ontologie je prostředek, do kterého se snažíme zachytit sdílené znalosti o určité oblasti tak, abychom mohli tyto znalosti (ve formě ontologie) dále rozvíjet, spravovat a používat.
- **OWL** – OWL (Web Ontology Language) je jazyk pro reprezentaci ontologií.
- **třída** – třída v ontologii reprezentuje nějaký pojem z reálného světa. Třídy jsou hierarchicky uspořádány podle obecnosti pojmu, který reprezentují. Podtřída vyjadřuje konkrétnější pojem než její nadtřída. Nejobecnější třídou ontologie je třída *Thing*.
- **slot (data property)** – slot je vlastnost třídy.
- **facet** – facet je vlastnost slotu.

- **instance třídy** – instance třídy je entita ontologie, která je odvozena z třídy. Instance třídy může nastavit hodnoty slotů (vlastností) definované její třídou.
- **relace (object property)** – relace reprezentují vztahy mezi jednotlivými třídami ontologie.

▪ **GUHA procedury**

- **4FT, antecedent, sukcedent, cedent** – 4FT (Four Fold Table) označuje implementaci nejpoužívanější GUHA procedury ASSOC. Cílem procedury 4FT je nalezení asociačních pravidel tvaru $A \sim S/P$, kde A (antecedent), S (sukcedent) i P (podmínka) jsou booleovské atributy a \sim je 4FT kvantifikátor. Procedura funguje tak, že sestrojí čtyřpolní tabulku obsahující v jednotlivých polích tabulky frekvence atributů splňujících antecedent a sukcedent (pole a), splňující antecedent, ale nesplňující sukcedent (pole b) atd. a tuto tabulku testuje vůči kvantifikátoru \sim za dané podmínky P.

	Sukcedent	\neg Sukcedent	
Antecedent	a	b	r
\neg Antecedent	c	d	s
	k	l	m

- **booleovský atribut** – booleovský atribut je formule splňující následující podmínky.
 - Pokud A je atribut a α je neprázdná podmnožina množiny $\{a_1, \dots, a_k\}$ všech možných hodnot atributu A, pak $A(\alpha)$ je *základní booleovský atribut*. α je koeficient základního booleovského atributu $A(\alpha)$. Každý základní booleovský atribut je booleovský atribut. Pokud φ a ψ jsou booleovské atributy, pak $\varphi \wedge \psi$, $\varphi \vee \psi$ a $\neg\varphi$ jsou odvozené booleovské atributy. Každý *odvozený booleovský atribut* je booleovský atribut.¹
- **KL, CF, SD4FT, SDCF a SDKL** – další GUHA procedury (potažmo jejich implementace)

¹ Zde uvedená definice booleovského atributu pochází z Rauch J.: Logic of Association Rules, Applied Intelligence, Vol. 22, Issue 1, pp. 9–28.

Dodatek B: Návod na vytvoření ontologií

Tento dokument je dodatkem k diplomové práci Využití ontologií pro GUHA procedury (Martin Zeman). Tento dokument poskytuje návod na obohacení ontologií v jazyce OWL o možnost uchovávat v nich rozšiřující informace definované ve výše zmíněné diplomové práci. Tento návod popisuje nastavení rozšiřujících informací v aplikaci Protégé-OWL verzi 3.3 (<http://protege.stanford.edu>). Lze předpokládat, že obdobně to bude fungovat i pro novější verze (ovšem finální, nikoliv alfa a beta verze).

Návod předpokládá dva způsoby vytvoření ontologie, která by umožňovala uchovat rozšiřující informace.

Je-li tvořena ontologie zcela nová, doporučuje se využít šablonu pro tvorbu nových ontologií, podporujících využití rozšiřujících informací v systému Ferda (je jednou z příloh diplomové práce). Pokud použijeme tuto šablonu, vytváříme ontologii běžným způsobem, ale kromě běžných možností máme také možnost u tříd nastavit zmíněné rozšiřující informace (Cardinality, Minimum, Maximum, Domain Dividing Values a Distinct Values). Pro pochopení významu těchto hodnot viz diplomová práce Využití ontologií pro GUHA procedury.

Druhou možností je obohatit již existující ontologii o rozšiřující informace. V takovém případě tento návod předpokládá, že je takováto ontologie ve formátu OWL (v libovolném dialektu Lite/DL/Full). Pokud ne, musí ji uživatel do takového formátu převést (např. aplikace Protégé-OWL umožňuje převod mezi různými formáty ontologií). Postup je následující:

1. Zobrazíme metatřídy

- a. postupně zvolíme záložky(menu) *OWL -> Preferences... -> Visibility*
- b. v oblasti *Metaclasses* zaškrtneme možnost *All*

2. Vytvoříme struktury pro rozšiřující informace

- a. zvolíme záložku *Properties*
- b. zvolíme záložku *Datatype*
- c. zvolíme možnost *Create Datatype property*, vytvoří se nový slot (*datatype property*)
- d. Nastavíme hodnotu *Domain* (volba *Specialise Domain*) na *rdfs:Class*
- e. Vyplníme další parametry slotu podle níže uvedené tabulky
!!!Pozor, case sensitive!!! Aby tyto vlastnosti správně fungovaly ve Ferdovi, musí názvy odpovídat, i co se velikosti písmen týká.
- f. nezapomeňte u každého slotu nastavit obor hodnot (*Domain*, viz bod d)

Name	Range	Functional	Allowed values
Cardinality	String	checked	Nominal, Ordinal, OrdinalCyclic, Cardinal
Minimum	Any	checked	
Maximum	Any	checked	
DomainDividingValues	Any		
DistinctValues	Any		

Tabulka: Vlastnosti vybrané pro uchování v ontologiích a hodnoty jejich nastavení

3. Upravíme si formuláře pro příjemnější nastavování rozšiřujících hodnot u konkrétních tříd

- a. zvolíme záložku *Forms*
- b. vlevo v oblasti *Asserted Hierarchy* vybereme metatřídou *owl:Class*
- c. zobrazí se nám rozvržení jejích slotů, kliknutím vybereme oblast slotu *Cardinality*
- d. vpravo nahoře vybereme v oblasti *Selected Widget Type* hodnotu *DataRangeFieldWidget*
- e. dle libosti si upravíme umístění dalších slotů, aby se nám snadno vyplňovaly při tvorbě nových tříd

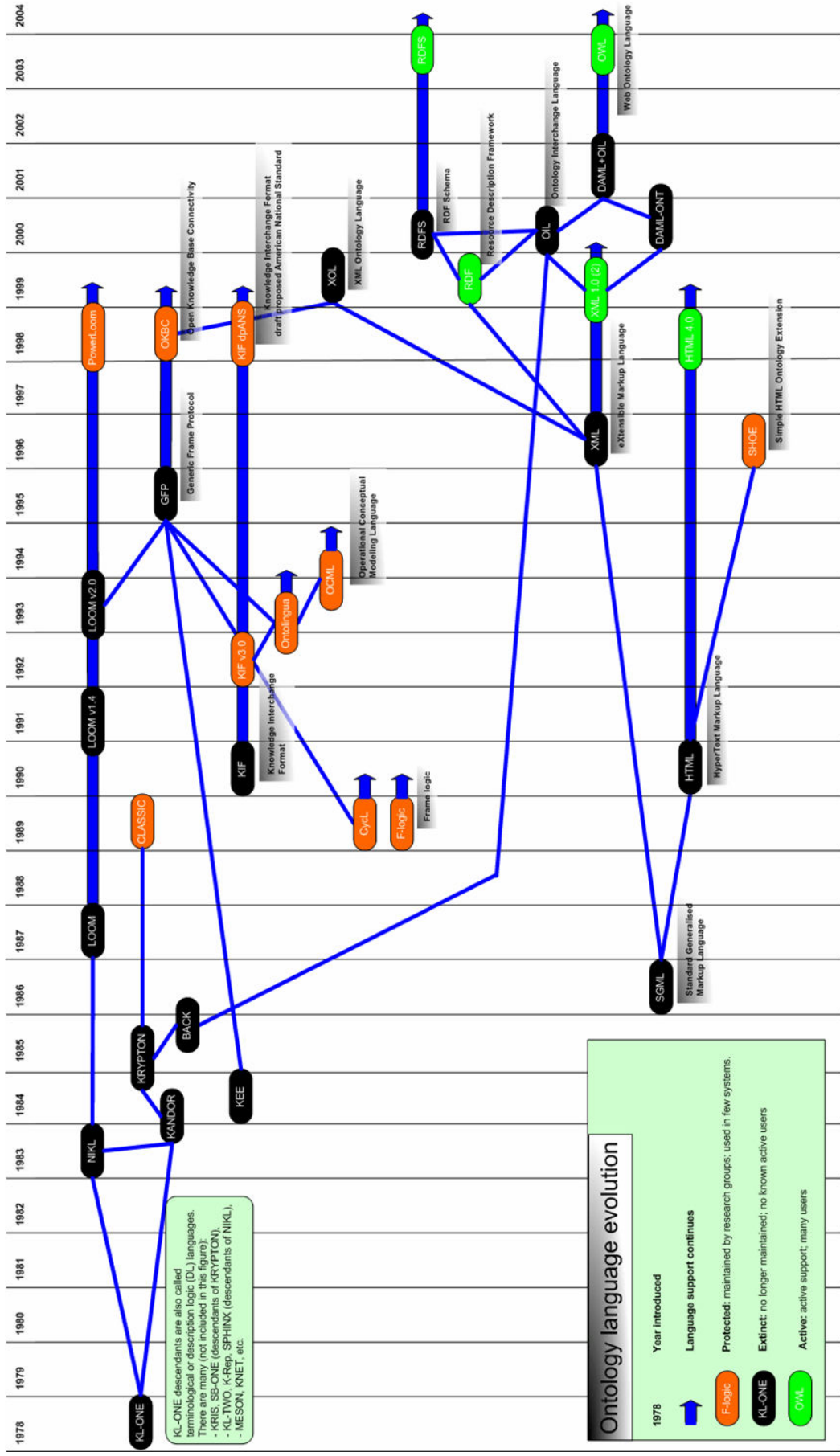
4. Vyplníme hodnoty pro třídy

- a. u jednotlivých tříd, příp. u nových tříd nyní máme možnost vybrat hodnotu vytvořených parametrů (slotů)

Dodatek C: Obsah přiloženého CD

- **Text** – text této diplomové práce i s dodatky
- **Prilohy**
 - **Priloha1_Ulohy_pro_DZD** – adresář obsahující projekty Ferdy, které byly v diplomové práci použity jako
 - **Priloha2_Ontologie_UMLS2_Cespivova** – ontologie vytvořená v rámci práce [15]
 - **Priloha3_Prezentacni_ontologie_biblio_Svatek** – soubor s odkazy na další zdroje informací o prezentačních ontologiích
 - **Priloha4_Sablona_pro_nove_ontologie** – prázdná šablona umožňující snadné vytváření nových ontologií, které budou moci využívat rozšíření Ferdy implementované v rámci této práce
 - **Priloha5_Ontologie_UMLS_Ferda** – ontologie, která byla použita pro experimentální část této diplomové práce, ontologie vznikla rozšířením ontologie UMLS2
- **Ferda** – zdrojové kódy prostředí Ferda
- **howToInstall.html** – pokyny pro instalaci prostředí Ferda
- **implementovano.doc** – přehled zdrojových adresářů/souborů, které byly vytvořeny v rámci této diplomové práce

Dodatek D: Vývoj jazyků pro reprezentaci ontologií



Obrázek 17: Vývoj jazyků pro reprezentaci ontologií (převzatý z [33])