

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Diplomant: Bc. Marek Nekvasil
Vedoucí diplomové práce: Ing. Martin Labský
Recenzent: Ing. Radek Burget, Ph.D.

VYUŽITÍ ONTOLOGIÍ PŘI INDUKCI WRAPPERŮ

školní rok 2005/2006

PROHLÁŠENÍ

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne

.....

podpis

PODĚKOVÁNÍ

Na tomto místě bych chtěl předně poděkovat vedoucímu mé diplomové práce Ing. Martinu Labskému za výjimečný zájem, konstruktivní připomínky a množství času, které mi při tvorbě této práce věnoval.

Také bych chtěl směřovat poděkování všem mým blízkým, za jejich trpělivost a podporu.

ABSTRAKT

Cílem této práce je navržení způsobu rozšíření pokročilých znalostních modelů, ontologií, za účelem jejich využití při automatizované extrakci informací z webových dokumentů.

První část práce je věnována popisu současných přístupů k získávání informací z webových dokumentů za pomoci takzvaných wrapperů a také metodám, jakými je možno wrappery vytvářet s co nejvyšším stupněm automatizace. Tyto přístupy a metodiky jsou kriticky zhodnoceny zejména s ohledem k záměru využívat k automatické tvorbě wrapperů rozšířené ontologie.

Druhá část práce je pak věnována popisu standardů pro zápis ontologií a návrhu rozšíření jednoho z ontologických jazyků, jazyka OWL, o možnost zápisu šablon pro identifikaci typických hodnot vlastností extrahované třídy. Tyto šablony jsou navrženy tak, aby je bylo možno tvořit pomocí hierarchicky uspořádaných dílčích vzorů. Několik konkrétních vzorů je rovněž navrženo.

Většina této části je pak věnována návrhu a odvození inferenčního modelu, založeném na principech fuzzy logiky, pro vyhodnocování shody vzorů a jejich skládání do šablon. Tento model lze využít pro automatické označení vzorových hodnot vlastností extrahované třídy v polostrukturovaném dokumentu, tedy automatizovanou anotaci tohoto dokumentu.

V poslední části práce je pak navržen jednoduchý typ wrapperu a způsob jeho učení s využitím navržené metodiky automatizované anotace pomocí rozšířené ontologie.

ABSTRACT

The purpose of this work is to bring in an extension of advanced knowledge models, known as ontologies, so that they can be utilized in the process of automated information extraction from the web documents.

In the first part of this paper we describe current approaches to information acquisition from web document by using so called wrappers and we also describe various methods to create such wrappers with as high degree of automation as possible. These methods are commented with respect to our intention to utilize the extended ontologies in the process of automated creation of wrappers.

The second part is then devoted to the description of the ontology notation standards and to the proposed extension of OWL, one of the ontology languages. This extension is meant to bring in the possibility to include templates for the common values of properties of the extracted class in the ontology. These templates are designed to enable their composition of hierarchically ordered partial patterns. A few such patterns are also proposed.

However most of this section is devoted to a proposition and derivation of an inference model, based on principles of fuzzy logic, for evaluation of the pattern matches and their combination into a template. This model can be used to automatically annotate the examples of properties of the extracted class in the document.

Finally there is proposed a simple type of wrapper and a way it can be learned automatically using the proposed method of automated annotation with an extended ontology.

OBSAH

PROHLÁŠENÍ	I
PODĚKOVÁNÍ	II
ABSTRAKT	III
ABSTRACT	IV
OBSAH	V
KAPITOLA 1 ÚVOD A VYMEZENÍ	1
1.1 DNEŠNÍ WEB	1
1.1.1 <i>Informační zdroje a jejich uživatelé</i>	1
1.1.2 <i>Softwarové agenty a sémantický web</i>	2
1.2 CÍLE A OMEZENÍ PRÁCE	3
1.3 STRUKTURA PRÁCE	5
KAPITOLA 2 ZÍSKÁVÁNÍ INFORMACÍ ZA POUŽITÍ WRAPPERŮ	7
2.1 ZAMĚŘENÍ KAPITOLY	7
2.2 ŘETĚZCOVÉ WRAPPERY	8
2.2.1 <i>LR</i>	8
2.2.2 <i>HLRT</i>	10
2.2.3 <i>OCLR</i>	12
2.2.4 <i>HOCLRT</i>	13
2.2.5 <i>N-LR a N-HLRT</i>	14
2.3 STROMOVÉ WRAPPERY	14
2.3.1 <i>Elog</i>	15
2.3.2 <i>Konečné automaty</i>	17
2.3.3 <i>XPath</i>	17
2.4 SROVNÁNÍ JEDNOTLIVÝCH ZPŮSOBŮ EXTRAKCE	19
KAPITOLA 3 ZPŮSOBY INDUKCE WRAPPERŮ	21
3.1 ZAMĚŘENÍ KAPITOLY	21
3.2 ZNÁMÉ METODY TVORBY WRAPPERŮ	21
3.2.1 <i>Věštec a PAC analýza</i>	21
3.2.2 <i>Interaktivní vizuální tvorba vzorů</i>	24
3.2.3 <i>Učení XPath výrazů dotazy na fragmenty stromů</i>	27
3.2.4 <i>Učení XPath výrazů zobecněním vzorů pro průchod stromem</i>	29
3.3 CELKOVÉ ZHODNOCENÍ METOD	30
KAPITOLA 4 NÁVRH ROZŠÍŘENÍ ZÁPISU ONTOLOGIÍ	31
4.1 ZAMĚŘENÍ KAPITOLY	31
4.2 ONTOLOGIE	31
4.2.1 <i>Původní pojem ontologie</i>	31
4.2.2 <i>Ontologie v sémantickém webu</i>	32
4.3 JAZYKY PRO ZÁPIS ONTOLOGIÍ	33
4.3.1 <i>RDF</i>	34
4.3.2 <i>RDF Schémata</i>	38
4.3.3 <i>OWL</i>	40
4.4 ROZŠÍŘENÍ JAZYKA OWL	45

KAPITOLA 5 NÁVRH VYUŽITÍ ROZŠÍŘENÉ ONTOLOGIE PŘI INDUKCI WRAPPERU.....	49
5.1 ZAMĚŘENÍ KAPITOLY	49
5.2 ŠABLONY A VZORY	50
5.3 JEDNODUCHÉ VZORY.....	51
5.3.1 Shoda vzoru a evidence vzoru	52
5.3.2 Odvození výpočtu míry evidence vzoru	53
5.3.3 Zahrnutí přesnosti vzoru do výpočtu	54
5.3.4 Úplnost vzoru	56
5.3.5 Kombinovaný výpočet s užitím přesnosti i úplnosti.....	58
5.4 SLOŽENÉ VZORY	60
5.4.1 Druhy složených vzorů	60
5.4.2 Výpočet míry shody šablony.....	61
5.4.3 Příklad vyhodnocení míry evidence šablony	63
5.4.4 Další efekt parametrů přesnosti a úplnosti.....	65
5.5 NÁVRH ZÁPISU VZORŮ	66
5.5.1 Šablona <ot:Template>	66
5.5.2 Řetězec <ot:String>	67
5.5.3 Seznam řetězců <ot:Stringlist>.....	68
5.5.4 Sřetěžení <ot:Concatenation>	69
5.5.5 Kontext <ot:Context>	70
5.5.6 Číslo <ot:Number>.....	71
5.5.7 Rozdělení <ot:Distribution>	72
5.5.8 Regulární výraz <ot:Regex>.....	73
5.5.9 Další vzory.....	73
KAPITOLA 6 IMPLEMENTACE A OVĚŘENÍ NAVRŽENÝCH METOD	75
6.1 ZAMĚŘENÍ KAPITOLY	75
6.2 ANOTACE DOKUMENTU.....	75
6.2.1 Ontologie.....	75
6.2.2 Automatická anotace dokumentu.....	77
6.3 INDUKCE WRAPPERU	80
6.3.1 Očištění evidencí	80
6.3.2 Zobecnění XPath cesty	82
6.3.3 Hodnocení chyby zobecnění cesty	83
6.3.4 Korespondence hodnot různých vlastností	84
KAPITOLA 7 ZÁVĚRY.....	86
TERMINOLOGICKÝ SLOVNÍK.....	88
Termíny přeжатé z terminologického slovníku katedry informačních technologií.....	88
Další termíny.....	89
ZDROJE	93

Kapitola 1

ÚVOD A VYMEZENÍ

1.1 Dnešní web

1.1.1 Informační zdroje a jejich uživatelé

Informační věk s sebou přinesl ohromné množství informačních zdrojů. Všeobecné přijetí standardu HTTP a víceméně standardního formátu HTML a jejich implementace do levných webových prohlížečů umožnily zpřístupnit tyto informační zdroje on-line komukoliv. S rostoucí dostupností připojení k internetu tak v posledních deseti letech rapidně narostl i počet uživatelů těchto webových informačních zdrojů. Dnes je prakticky běžné, že si člověk může na webu vybrat z ceníku a objednat libovolné zboží, zobrazit jeho vlastnosti a přečíst komentáře, jaké k němu měli ostatní uživatelé. Informační zdroje dostupné na webu jsou nejrůznějšího charakteru, ať už se jedná o zmíněné katalogy zboží, předpovědi počasí, telefonní seznamy, firemní stránky, knihovnické záznamy, diskusní fóra, stále populárnější blogy či cokoliv jiného, co si jen můžeme představit.

Naprostá většina na internetu dostupných informačních zdrojů je navržena k přímému zpracování lidmi. Používané formáty pro zobrazování (např. HTML) a zadávání (např. HTML formuláře) dat tomuto účelu plně vyhovují. Ne vždy je však vhodné zpracovávat informace z dostupných zdrojů ručně.

Alternativou k takovému ručnímu zpracování je automatické zpracování, čímž se rozumí užití počítačového programu namísto člověka k interakci s informačními zdroji. Obyčejný uživatel, hledající konkrétní informace, většinou nemá potřebu svoji interakci s informačním zdrojem nějak

automatizovat. Oproti tomu však některé firmy provádějící průzkum trhu, statistické úřady, vědecká pracoviště pracující na vývoji sémantického webu nebo jakékoliv jiné společnosti provozující sběr dat ve větším měřítku budou nejspíše tuto automatizaci považovat za nutnou součást své činnosti, nebo přinejmenším součást velmi vítanou. Ze zřejmých důvodů tak existuje poměrně silná poptávka po systémech, které automatizují proces vyhledávání, sběru, řízení, třídění a redistribuce informací z rozličných zdrojů.

1.1.2 *Softwarové agenty a sémantický web*

Program nebo část programu automatizující některé z výše uvedených úloh můžeme podle názvosloví oblasti sémantického webu nazvat softwarovým *agentem*¹. Tento pojem je v [Antoniou, Harmelen, 04, str. 14] definován takto:

„Agenty jsou části softwaru, které pracují autonomně a proaktivně. Koncept softwarových agentů se vyvinul z konceptů objektově orientovaného programování a komponentově založeného vývoje softwaru.

Osobní agent na sémantickém webu bude dostávat úkoly/dotazy a preference od uživatele, vyhledávat informace z webových zdrojů, komunikovat s jinými agenty, porovnávat informace o uživatelských požadavcích a preferencích, činit jistá rozhodnutí a poskytovat odpovědi uživateli.“

Zde je namístě zmínit, že *sémantický web* je pojem, který sice vznikl již dříve, nicméně jej od roku 2001 prosazuje a zaštiťuje konsorcium W3C (blíže [W3C, 01]). Jedná se o vizi budoucnosti webu, jako informačního prostoru, kde budou dokumenty a data v nich obsažená publikovány spolu s jejich

¹ V některých publikacích (např.: <http://www.akvs.cz/akp-2005/09-pokorny.pdf>) se můžeme setkat s životným skloňováním sousloví „softwarový agent“, což je vysvětlováno autonomií a proaktivitou takové entity. My se však přidržíme konzervativnějšího neživotného skloňování.

sémantikou a sice v takové formě, která bude strojově zpracovatelná s přijatelnou přesností.

Je otázkou, zda se vize sémantického webu prosadí ve větším měřítku. Hlavní problém spočívá v tom, že dnešní web je silně heterogenní prostředí, kde může jakýkoliv jedinec či organizace publikovat data v libovolném formátu a libovolného obsahu. Z tohoto důvodu vydává W3C svoje standardy formou nezávazných doporučení, neexistuje tedy žádná vynutitelnost jejich dodržení. Navíc samozřejmě připojení metadat obsahujících sémantiku příslušného dokumentu přidává na pracnosti při jeho tvorbě a rostou náklady na vytvoření takového dokumentu, časové i finanční, bez jakéhokoliv přínosu pro zpracování člověkem. To je také důvod, proč naprostá většina ekonomických subjektů publikujících na webu dosud k sémantickému značkování svých dokumentů nepřestoupila.

Pokud tedy budeme chtít vyvinout softwarové agenty takové, které budou alespoň částečně funkční v prostředí dnešního webu, budou to muset být agenty pracující s rozhraními informačních zdrojů určenými pro lidskou interakci. Postihnout takto celou oblast softwarových agentů tak, jak bylo popsáno výše, by bylo přespříliš složité. Existují však poměrně dobře identifikovatelné dílčí úlohy, mezi něž patří i extrakce informací z polostrukturovaných dokumentů, která je hlavním předmětem této práce. Dalšími v současnosti postižitelnými dílčími úlohami jsou například vyhledávání relevantních dokumentů, klasifikace jejich zdrojů, vyhodnocování dotazu a preferencí uživatele či integrace těchto činností, což však již rámec této práce přesahuje.

1.2 Cíle a omezení práce

Pro program extrahující informace z polostrukturovaných dokumentů (jako např. z webových stránek) se v literatuře (např. [Papakonstantinou et al.,95]) používá označení *wrapper*, což je samo o sobě širší pojem, pokud však

nebude uvedeno jinak, budeme jej v této práci chápat pod tímto významem. Také se omezíme na extrakci informací z dokumentů dnes na webu nejběžnějších a sice webových stránek ve formátu HTML, popřípadě XHTML. Wrappery lze tvořit dvěma způsoby.

Prvním způsobem je ruční tvorba speciálního programu pro konkrétní sadu webových stránek s neměnnou strukturou. Jak lze snadno uhadnout, má tento způsob řadu nedostatků, počínaje omezeností na danou doménu a netolerantností vůči změnám struktury dokumentů v čase konče. Obrovskou výhodou této ruční tvorby wrapperu je však jeho vysoká úspěšnost, která je stejná jako úspěšnost jeho tvůrce.

Druhým způsobem je využití postupů strojového učení ([Mitchel, 97]) k automatizování tvorby wrapperu. Protože takovýto úkol je úlohou induktivního učení, označujeme jej tedy *indukce wrapperu* ([Kushmerick, 97, str. 10]). Práce ubírající se tímto směrem se obvykle zabývají způsobem, jak nejlépe generalizovat extrakční algoritmus, přičemž si za vstup berou vzorové dokumenty s anotovanými výskyty hodnot určených k extrakci, nebo naopak množinu podobných dokumentů, ve kterých se objevují společné vzory a analyzují se mezi nimi rozdíly. S tímto přístupem se můžeme setkat poměrně často (např.: [Kushmerick, 97], [Muslea et al., 99], [Chang et al., 03] nebo [Zhao et al., 05]).

Zřídka je však při indukci wrapperu brán zřetel na sémantiku extrahovaných informací, ale samozřejmě existují i snahy o její zapojení do problému. S příkladem takového přístupu se setkáme například v [Hogue, Karger, 04], kde při poloautomatické indukci wrapperu (blíže viz 3. kapitolu) je uživateli dána možnost, aby při označování příkladů extrahovaných hodnot určoval také jejich sémantiku, a sice ve stylu jazyka RDF.

V této práci se pokusíme navrhnout metodiku využití sémantiky dané oblasti neboli doménových znalostí v samotném procesu indukce wrapperu. Specifikace konceptuálních doménových znalostí se nazývá *ontologie* a lze ji

zapisovat v různých formátech. Ontologii vytvořenou za účelem využití v extrakční úloze budeme nazývat *extrakční ontologie*.

Tato práci si klade za cíl navrhnout vhodné rozšíření jazyka OWL, který se dnes k zápisu ontologií používá nejčastěji (jedná se totiž o doporučený standard W3C, viz [W3C, 04, 3]), tak, aby jej bylo možno použít k automatizované sémantické anotaci dokumentů. Dalším cílem je navržení jednoduchého algoritmu indukce wrapperu, který bude zmíněný způsob anotace s pomocí ontologie využívat a ověřit funkčnost a omezení navrhované metodiky. Zde se omezíme na indukci wrapperů určených pro dokumenty ve formátu HTML, konkrétně pak ještě na dokumenty s tabulární strukturou dat.

1.3 Struktura práce

V následující kapitole se budeme zabývat druhy wrapperů (někdy se hovoří o třídách wrapperů), o kterých jsou dostupné informace, a posoudíme vhodnost jednotlivých druhů wrapperů pro různé typy extrakčních úloh a pro různé formáty dokumentů.

Kapitola 3 bude věnována metodikám indukce wrapperu zejména z hlediska oblasti jejich určení. Posoudíme zde využitelnost jednotlivých metodik pro náš záměr provádět automatizované anotování na půdě polostrukturovaných dokumentů a navrheme vhodný způsob indukce wrapperu pro jeho ověření.

Návrhu rozšíření zápisu ontologie v jazyce OWL bude věnována kapitola 4. Stručně popíšeme jazyk OWL, způsob jeho vzniku a navrheme rozšíření tohoto jazyka o zápis hierarchických vzorů pro typické hodnoty atributů doménových tříd v extrakční ontologii.

V 5. kapitole se budeme zabývat metodikou využití navržených vzorů k sémantické anotaci dokumentů. Rozebereme zde jednotlivé navržené vzory, možnosti parametrizace těchto vzorů a samozřejmě také jejich alternativy.

V neposlední řadě pak navrhne také inferenční mechanismus pro skládání různých zde navržených vzorů.

Ověření navržených metod pomocí jejich implementace se budeme věnovat v kapitole 6. Navrhne zde konkrétní algoritmus pro automatickou anotaci dokumentu a algoritmus indukce jednoduchého wrapperu z takto anotovaného dokumentu.

V poslední části rozebereme možná omezení navrženého modelu a vhodnost jeho použití. Zde také vyvodíme závěry z dosažených zjištění a navrhne možnosti dalšího rozšíření našich metodik.

Kapitola 2

ZÍSKÁVÁNÍ INFORMACÍ ZA POUŽITÍ WRAPPERŮ

2.1 *Zaměření kapitoly*

K přístupu k datům obsaženým na webových stránkách se dnes běžně používá technika známá jako extrakce informací z webu nebo také HTML wrapping. Účelem této techniky je převést užitečná data z webové stránky, kde se nalézají v polostrukturované podobě, do formátu, který je lépe strojově zpracovatelný, ať už se jedná o uložení do XML, do obsahu relační databáze anebo naplnění datového modelu integrované aplikace.

Na HTML wrapping můžeme pohlížet jako na úlohu, jejímž cílem je identifikace relevantních částí dokumentu. Protože relevantní data mají vždy nějakou strukturu, díváme se na extrahované části jako na *instance* určitých tříd, což je blíže vysvětleno ve čtvrté kapitole. Formálně lze wrapper definovat jako zobrazení z množiny obsahu stránky do množiny instancí, které tato stránka obsahuje.

V extrakci informací samozřejmě neexistuje jediný správný nebo nejlepší způsob jak z webových stránek získávat relevantní data. Ba právě naopak, technik HTML wrappingu existuje mnoho, a cílem této kapitoly je popsat nejdůležitější z nich.

V zásadě lze rozlišit dva druhy HTML wrapperů, podle způsobu, jakým je u nich chápána struktura webové stránky. Prvním druhem jsou wrappery, které interpretují stránku jako řetězec textu. U toho druhu wrapperů úloha identifikace relevantních částí odpovídá odlišení důležitých částí od nepodstatných podle jisté struktury. Toto je historicky straší přístup, má však i své výhody. Řetězcovým (jinak také lineárním) wrapperům je věnována první část této kapitoly.

Druhým, novějším, přístupem k HTML wrappingu jsou wrappery, které stránku interpretují pomocí tzv. DOM stromů, což je stromová struktura, která se opírá o hierarchickou strukturu elementů HTML dokumentu. Identifikování relevantních částí dokumentu se pak rovná identifikování podstatných uzlů v takovémto stromě. Tomuto druhu wrapperů se budeme věnovat v druhé části kapitoly.

2.2 *Řetězcové wrappery*

Jak již bylo zmíněno, podstatou tohoto typu wrapperů je reprezentace webové stránky formou textového řetězce. Při identifikaci relevantních částí dokumentu se zde spoléhá na jistou strukturu dokumentu, což je opodstatněné zaměřením na dokumenty ve formátu HTML. Princip identifikace tedy spočívá v tom, že relevantní části řetězce jsou uvozeny, ohraničeny či jinak vymezeny jinými pevně danými částmi řetězce, konkrétně se pak ve většině případů jedná o HTML tagy.

Extrakční úlohu lze pomocí řetězcového wrapperu provádět několika způsoby, podle struktury a složitosti extrahovaných dat. Vyčerpávající seznam tříd řetězcových wrapperů obsáhl ve své disertační práci Nicolas Kushmerick [Kushmerick, 97, str. 76 - 104], zde si představíme nejdůležitější z nich, přičemž zachováme Kushmerickovo označení těchto tříd.

2.2.1 *LR*

Třída wrapperů LR je nejjednodušší z tříd řetězcových wrapperů. Označení LR je zkratkou z „Left-Right“, což znamená, že relevantní části dokumentu jsou vymezeny danými řetězci zleva a zprava.

Instance datové třídy určené k extrakci se v tomto případě uvažuje ve formě uspořádané n-tice hodnot. Každý prvek této n-tice může být v dokumentu vymezen jinými řetězci L a R. Pro extrakci z dokumentu tedy



Obrázek 2.1 - dokument pro LR extrakci

```
<HTML>
<TITLE>Ceny pobytů</TITLE>
<BODY>
<B>Řecko - Lefkada</B> <I>16 299 Kč</I><BR>
<B>Mallorca - Santa Ponsa</B> <I>21 100 Kč</I><BR>
<B>Egypt - Sharm El Sheikh</B> <I>18 500 Kč</I><BR>
<B>Egypt - Ghiza</B> <I>19 049 Kč</I><BR>
</BODY>
</HTML>
```

Obrázek 2.2 - kód dokumentu

třída wrapperů LR má $2n$ parametrů – pro každý prvek n -tice je třeba určit L a R zvlášť.

Algoritmus LR wrapperu je následující: Projdeme dokument od začátku až do prvního výskytu prvního parametru L. Vše, co následuje až do výskytu prvního parametru R, je načteno jako hodnota prvního parametru extrahované n -tice. Ve čtení dokumentu se pokračuje obdobně přes další parametry L_i a R_i , dokud není načtena celá n -tice. Poté se pokračuje načítáním dalších n -tic nebo dokud se nenarazí na konec dokumentu.

příklad: Budeme uvažovat velmi jednoduchý dokument, jehož vizuální reprezentace je na obrázku 2.1 a jeho zápis na obrázku 2.2. Jedná se o zjednodušený seznam pobytových zájezdů, z nějž budeme chtít extrahovat informace o místě a ceně pobytu. Datová struktura extrahované třídy „pobyt“ tedy bude tvořena dvojicí hodnot „místo pobytu“ a „cena pobytu“.

Jistě si všimnete, že místo pobytu je uvedeno tučným písmem a cena vždy kurzívou, čemuž v HTML kódu odpovídají tagy a <I></I>. Můžeme tedy parametry pro LR wrapper stanovit právě jako „“, „“, „<I>“, „</I>“, což bude mít za následek správné extrahování dvojic hodnot místo-cena podle výše uvedeného algoritmu.

Třída LR je ovšem velmi nepřesná, a proto je na reálných dokumentech z webu téměř nepoužitelná. Lze však s ní dobře provádět extrakci dat tabulární struktury. Ostatní třídy řetězcových wrapperů z třídy LR vycházejí a jistým způsobem ji rozšiřují.

2.2.2 HLRT

Další třídou řetězcových wrapperů je třída HLRT. Tato třída se snaží vyřešit některé problémy spojené s užitím třídy LR, které mohou nastat zejména pokud se hodnoty parametrů L a R vykytují v dokumentu i jinde než tam, kde se vyskytují data určená k extrakci.

HLRT je zkratkou z „Head-Left-Right-Tail“. Tato třída wrapperů identifikuje pomocí dodatečných parametrů H a T část dokumentu, na které bude provádět extrakci obdobně jako LR. Parametr H zde označuje hlavičku dokumentu (respektive konec hlavičky dokumentu) a T konec dokumentu, s tím, že relevantní data pro extrakci se vyskytují pouze mezi hlavičkou a koncem dokumentu. Hlavička a konec jsou tedy v tomto pojetí chápány jako řetězce, vymezující jednoznačně relevantní část dokumentu shora a zdola. Wrapper HLRT pro extrakci n-tic z dokumentu vyžaduje $2n+2$ parametrů (2 pro hlavičku/konec a 2 pro každý prvek n-tice).

Algoritmus extrakce u třídy HLRT bude tedy následující: Nejprve se v dokumentu přeskočí za první výskyt parametru H. Potom se pro každou dvojici parametrů L_i , R_i pokračuje ve čtení dokumentu až za výskyt L_i . Text, mezi L_i a dalším výskytem R_i je extrahován jako i-tá část n-tice. Obdobně se



Obrázek 2.3 – dokument pro HLRT extrakci

```
<HTML>
<TITLE>Ceny pobytů</TITLE>
<BODY>
<B>Ceny pobytů</B><P>
<B>Řecko - Lefkada</B> <I>16 299 Kč</I><BR>
<B>Mallorca - Santa Ponsa</B> <I>21 100 Kč</I><BR>
<B>Egypt - Sharm El Sheikh</B> <I>18 500 Kč</I><BR>
<B>Egypt - Ghiza</B> <I>19 049 Kč</I><BR>
</P><HR>
<I>Další informace ...</I>
</BODY>
</HTML>
```

Obrázek 2.4 - kód dokumentu

pokračuje v extrakci dalších n-tic, dokud se v dokumentu nenarazí na výskyt parametru T.

příklad: Oproti třídě LR můžeme pomocí třídy HLRT extrahovat z o něco složitějších dokumentů. Příklad takového dokumentu je na obrázku 2.3 a jeho zápis v HTML na obrázku 2.4. Úloha zůstává stejná, jako v předchozím příkladu. Jak je vidět, použitím wrapperu LR se stejnými parametry („“, „“, „<I>“, „</I>“) by mělo za následek neúspěch, protože nadpis „Ceny pobytů“ je označen stejně jako první členy extrahované dvojice. Při takovémto použití bychom tedy přeskočili místo prvního pobytu.

Již v takto jednoduchém dokumentu můžeme spatřit hlavičku, tj. uvození relevantní části dokumentu, a konec. Můžeme tedy parametry H a T zvolit kupříkladu jako „ $\langle B \rangle \langle P \rangle$ “ a „ $\langle P \rangle \langle HR \rangle$ “, přičemž množinu parametrů L a R ponecháme nezměněnou. Takováto sada parametrů tedy opět povede k požadovanému výsledku.

Pohledem na dokument ve formě hlavička-tělo-konec získává třída HLRT oproti třídě LR zvýšenou odolnost proti chybným identifikacím parametrů. Stejně jako LR je i HLRT určeno k extrakci z dokumentů tabulkové struktury.

2.2.3 OCLR

Oproti pohledu HLRT na dokument jako na hlavičku, tělo a konec, reprezentuje třída OCLR dokument jako množinu n -tic navzájem oddělených nepodstatným textem.

Stejně tak, jako HLRT zabraňovalo chybné identifikaci v hlavičce a konci dokumentu, je OCLR navrženo, aby zabránilo chybné identifikaci v prostoru mezi n -ticemi. OCLR je zkratkou z „Opening-Closing-Left-Right“ a zavádí další parametry O a C , které označují začátek a konec každé n -tice, obdobně jako u HLRT označují parametry H a T začátek a konec dokumentu. Wrapper třídy OCLR má tedy také $2n+2$ parametrů (2 pro počáteční a koncové návěští n -tice a 2 pro každý prvek n -tice).

Algoritmus extrakce bude nyní probíhat tak, že se načte dokument až za první výskyt parametru O , poté se pro každou dvojici parametrů L_i a R_i extrahují dané části n -tice a pokračuje se ve čtení dokumentu, dokud se nenarazí na výskyt parametru C . Od dalšího výskytu parametru O se tento postup opakuje, dokud se nenarazí na konec dokumentu.

příklad: Pokud budeme nyní uvažovat dokument, který a na obrázku 2.5 a kód k němu na obrázku 2.6, zjistíme, že v takovémto případě by třída HLRT na úspěšnou extrakci nestačila. Je tomu tak proto, že mezi relevantními daty



Obrázek 2.5 - dokument pro OCLR extrakci

```
<HTML>
<TITLE>Ceny pobytů</TITLE>
<BODY>
  <B>1. </B><B>Řecko - Lefkada</B> <I>16 299 Kč</I><BR>
  <B>2. </B><B>Mallorca - Santa Ponsa</B> <I>21 100 Kč</I><BR>
  <B>3. </B><B>Egypt - Sharm El Sheikh</B> <I>18 500 Kč</I><BR>
  <B>4. </B><B>Egypt - Ghiza</B> <I>19 049 Kč</I><BR>
</BODY>
</HTML>
```

Obrázek 2.6 - kód dokumentu

(místa a ceny pobytů) se zde vyskytují ještě nepodstatná data, a sice pořadová čísla pobytů, která jsou označena stejnými tagy jako místo zájezdu („“, „“).

*Pokud stanovíme parametry O a C jako „“ a „
“, docílíme tím kýženého výsledku.*

OCLR je také určeno k extrakci z tabulkové struktury dat. Stejně jako HLRT přináší i OCLR oproti LR zvýšenou odolnost proti chybným identifikacím v nepodstatných částech dokumentu.

2.2.4 HOCLRT

Jak již název napovídá, je HOCLRT třída wrapperů, která spojuje vlastnosti tříd HLRT a OCLR. Lze dokázat ([Kushmerick, 97, str. 81]), že HOCLRT má větší popisnou schopnost než HLRT, zda má ale větší popisnou

schopnost než OCLR dokázat nelze. Každopádně se jedná o nejkompexnější třídu lineárních wrapperů určených pro dokumenty ve tvaru tabulky.

Za povšimnutí stojí také fakt, že třída HOCLRT vyžaduje pro extrakci n -tice z dokumentu $2n+4$ parametrů (2 pro hlavičku/konec dokumentu, 2 pro počáteční a koncové návěští n -tice a 2 pro každý prvek n -tice), což má také za následek větší složitost při jejím učení, o tom ale více až v následující kapitole.

2.2.5 *N-LR a N-HLRT*

N-LR a N-HLRT jsou další třídy řetězcových wrapperů, tyto však nejsou určeny pro data v tabulární formě. N v jejich názvu je zkratkou anglického „Nested“ a jsou určeny pro extrakci dat hierarchické stromové struktury jako je například obsah knihy nebo organizační struktura podniku.

Tyto třídy jsou rozšířením tříd LR a HLRT a sice o možnost rekurzivního zanořování atributů extrahované třídy. Tyto třídy však uvádíme jen pro úplnost, jejich algoritmus je příliš složitý a překračuje rámeček tohoto textu. Za povšimnutí však stojí fakt, že toto rozšíření pro hierarchickou strukturu dat je ve skutečnosti zobečněním, a že tabulární struktura dat je jen speciálním případem hierarchicky uspořádaných dat.

2.3 *Stromové wrappery*

Narozdíl od řetězcových wrapperů mohou stromové wrappery operovat namísto zdrojového HTML kódu dokumentu nad DOM stromy z toho kódu vygenerovanými. Pokud se pohybujeme nad DOM stromem, spočívá extrakční úloha wrapperu v nalezení podstatných částí tohoto stromu. Obvykle extrahovanými typy dat pak jsou například podstromy (množiny uzlů původního stromu), textový obsah uzlu, jeho část nebo hodnota atributu nějakého elementu.

Pohybujeme-li se tedy na DOM stromovou reprezentací dokumentu, můžeme použít několik různých postupů k nalezení relevantních instancí extrahované třídy. Těmi jsou programy podobné logickému programování vyhodnocovanému nad stromovou doménou ([Baumgartner et al., 01]), konečné stavové (stromové) automaty ([Carme et al., 04]) a výběr uzlů založený na dotazovacím jazyce XPath popřípadě XQuery ([Ceresna, Gottlob, 05]). Na tyto jednotlivé způsoby se nyní zaměříme blíže.

2.3.1 Elog

Elog ([Baumgartner et al., 01]) je neprocedurální logický jazyk, ne nepodobný jazyku Prolog, navržený speciálně pro hierarchickou extrakci dat. Namísto postupu extrakce se v tomto jazyce zapisují extrakční pravidla a to formou logických inferencí. Jednotlivé atributy extrahované třídy zde znázorňují pojmenované predikáty. Ukázka zápisu pravidel extrakce v jazyce Elog je na obrázku 2.7 (přejato z [Baumgartner et al., 01, str. 7]).

Predikáty se zde dělí na *atomy* a *filtry*, přičemž první jsou ty, které instance generují z nezávislých vstupů (např. *document(„www.ebay.com/“, S)* vytváří S jako instanci dokumentu) a druhé ty, které je definují z jiných instancí. Za povšimnutí zde stojí fakt, že se lze odkazovat na anonymní instance (v příkladu označeny znakem „_“), což jsou v podstatě všechny instance, které nejsou předmětem extrakce.

```

tablesq(S, X) ← document(“www.ebay.com/“, S), subseq(S, (.body, []), (.table, []), (.table, []), X),
before(S, X, (.table, [(elementtext, item, substr)], 0, 0, _, _), after(S, X, .hr, 0, 0, _, _))
record(S, X) ← tablesq(_, S), subelem(S, .table, X)
itemnum(S, X) ← record(_, S), subelem(S, *.td, X), notbefore(S, X, .td, 100)
itemdes(S, X) ← record(_, S), subelem(S, (*.td.*.content, [(a, substr)], X)
price(S, X) ← record(_, S), subelem(S, (*.td, [(elementtext, \var[Y].*, regvar)]), X), isCurrency(Y)
bids(S, X) ← record(_, S), subelem(S, *.td, X), before(S, X, .td, 0, 30, Y, _), price(_, Y)
currency(S, X) ← price(_, S), subtext(S, \var[Y], X), isCurrency(Y)
pricewc(S, X) ← price(_, S), subtext(S, [0 - 9]+ \. [0 - 9]+, X)

```

Obrázek 2.2.7 - Zápis extrakčního programu v jazyce Elog

Další skutečností hodnou naší pozornosti je způsob adresování množin vnořených elementů v dokumentu. Vnořené elementy se zde adresují názvem HTML element, přičemž je zde povolen náhražkový znak „*“ pro libovolný element, přes tečkovou syntaxi je dovoleno adresování elementů hierarchicky dále vnořených. Můžeme se tak setkat se zápisem „*.td.*.content“, což již na první pohled znázorňuje jistou cestu dokumentem, existuje zde tedy jistá podobnost s jazykem Xpath. Narozdíl od Xpath jsou zde tyto adresy parametry predikátů a jsou spojovány s dalšími podmínkami, jako například regulárními výrazy či dalšími predikáty.

Sada takovýchto pravidel, jako na obrázku 2.7, se pak nazývá *vzor*. Pro každé pravidlo vzoru jsou vytvořeny instance z predikátů všude tam, kde jsou splněny premisy pravidla. Výstupem extrakce jsou pak pouze minimální instance, což jsou instance takové, které v sobě neobsahují další instance, namísto toho jsou instance na výstupu uspořádány hierarchicky.

Extrakční program je zde tvořen sadou extrakčních vzorů. V tomto ohledu má jazyk Elog poměrně vysokou vyjadřovací schopnost a dovoluje dokonce vzájemné rekurzivní zanořování vzorů. Díky této vlastnosti lze správnou volbou pravidel například instanciovat další dokumenty pomocí atributů, potažmo odkazů, obsažených v původním dokumentu, což nám dovoluje procházet i odkazované dokumenty.

Tento způsob extrakce je používán uvnitř extrakčního nástroje Lixto, o kterém ještě bude řeč v následující kapitole v souvislosti s učením extrakčních vzorů. Extrakční pravidla lze psát i v podobě nezávislé na stromové struktuře dokumentu, takže se pak extrakční program chová jako řetězový wrapper. Do této části však extrakční jazyk Elog zařazen proto, že využití stromové struktury dokumentu je jeho autory preferováno.

2.3.2 Konečné automaty

V [Carme et al., 04] se můžeme setkat z dalším možným přístupem, jak extrahovat informace ze stromové reprezentace dokumentu. Tento přístup zakládá na vyhledávání ve struktuře stromu dokumentu pomocí konečného stavového automatu. Jednotlivé elementy dokumentu jsou reprezentovány jako stavy z konečné množiny všech stavů (zde se předpokládá konečná množina možných elementů). Extrakčním vzorem je zde stromová struktura stavů, jejíž výskyty je cílem identifikovat v původním stromu dokumentu a hodnoty konkrétních elementů tak extrahovat.

Tento přístup je poměrně efektivní, co se týče výpočetní náročnosti, protože využívá speciálních vlastností konečných automatů po transformaci do binárních stromů, avšak v současné Carmeho implementaci ignoruje hodnoty atributů elementů a dokonce i samotné obsahy elementů.

Konečné automaty jsou jinak poměrně hojně využívány v extrakčních úlohách, jedná se však většinou o extrakce ze struktury lineárního typu. Tento přístup však ukazuje, že je možné je využít i pro extrakci s využitím stromové struktury dokumentu.

2.3.3 XPath

XPath je doporučením konsorcia W3 ([W3C, 99]), jedná se o jazyk určený k adresování částí dokumentu ve formátu XML. Formálně nelze dokument ve formátu HTML ztotožnit s formátem XHTML, který je rozšířením XML, lze jej však do tohoto formátu s jistou tolerancí transformovat. Pak již nic nebrání využití tohoto hojně využívaného adresovacího jazyka k extrakci relevantních částí dokumentu.

K extrakčním účelům se podle potřeby ([Ceresna, Gottlob, 05], [Anton, 05]) používá někdy jen základní část jazyka XPath, která neobsahuje funkce a

predikáty, to však záleží na požadavcích na vyjadřovací schopnosti XPath výrazů.

Výraz XPath definuje cestu napříč DOM stromem dokumentu. Tato cesta je sekvencí kroků, oddělených lomítky, kde každý krok se skládá ze tří částí:

- směr prohledávání, označovaný také jako *osa*, následovaný znaky „::“
- filtry uzlů (název nebo typ adresovaných uzlů)
- posloupnost predikátů v hranatých závorkách (může být prázdná)

Osa prohledává určuje směr procházení, tedy zda se pokračuje na předchozí, následující, rodičovské nebo vnořené elementy. Filtry omezují množinu uzlů podél osy prohledávání a konečně predikáty jsou výrazy XPath (uvedeny v hranatých závorkách), které dále omezují množinu výsledků

Každý krok je vyhodnocen nad množinou uzlů DOM stromu a vrací také množinu uzlů DOM stromu. Výraz je pak vyhodnocen tak, že výstup každého kroku je vstupem pro následující krok, přičemž vstupem prvního kroku je vstup výrazu a výstup posledního kroku je výstupem výrazu.

XPath výrazy mohou vypadat například takto:

- `/descendant-or-self::H1`
- `/child::BODY/child::H1[position()=1]`
- `/child::BODY/child::H1[following-sibling::P[child::HR]]`
- `/descendant-or-self::H1[preceding-sibling::IMG[@src='filename.gif']]/child::text()`

Způsob zápisu extrakčních pravidel formou výrazu XPath má o něco menší vyjadřovací schopnost než logická neprocedurální formulace u jazyka Elog, má však také velmi dobrou úspěšnost, která závisí pouze na kvalitách tvůrce těchto pravidel.

2.4 Srovnání jednotlivých způsobů extrakce

Pokud se pokusíme srovnat jednotlivé třídy wrapperů, musíme vzít v úvahu hledisko typu extrakční úlohy, potažmo typu a struktury dokumentu, ze kterého má být extrahováno a samozřejmě také struktury extrahovaných dat.

Na počátku této práce jsme jako jedno z omezení určili skutečnost, že se budeme pohybovat výhradně na poli polostrukturovaných dokumentů ve formátu HTML. Ani zde však nelze jednoznačně říci, že by jeden přístup byl ve všech ohledech lepší, než ostatní. Pokud budeme uvažovat pevnou tabulární strukturu dat, tak zde mají navrch stromové wrappery obecně, neboť jsou daleko více odolné vůči nepravidelnostem ve struktuře dokumentu. Budeme-li uvažovat obecně libovolnou strukturu dat, je situace poněkud jiná. Konečně automaty zde vynecháme, protože jsou častěji využívány pro vyhledávání v sekvencích. U použití výrazů jazyka XPath není extrakce libovolné datové struktury vyloučena, záleželo by na konkrétní formulaci těchto pravidel, v literatuře se však, pokud je nám známo, o tomto zmínky nevyskytují. Jedině extrakce s použitím jazyka Elog a dále Kushmerickovy třídy wrapperů N-LR a N-HLRT jsou pro obecně hierarchickou datovou strukturu navrženy.

Po uvážení těchto faktů, tedy extrakce s pomocí logického jazyka Elog vychází jako nejuniverzálnější řešení, které je ovšem také velmi složité a náročné na tvorbu. Extrakční pravidla Elog lze sice vytvářet pomocí vizuálních vzorů v nástroji Lixto, k tomu ale více až v následující kapitole.

Jako poněkud vhodnější řešení, i když méně univerzální, se jeví použití výrazů jazyka XPath. Důvodů k tomu je hned několik. Jednak je tento jazyk

určen výhradně pro stromovou strukturu dokumentu (konkrétně XML), snaží se být maximálně simplicistický a jednak se jedná o doporučený standard W3C, není tudíž zapotřebí vytvářet nový umělý jazyk určený výhradně pro extrakční úlohy.

Kapitola 3

ZPŮSOBY INDUKCE WRAPPERŮ

3.1 *Zaměření kapitoly*

V minulé kapitole jsme se zabývali jednotlivými způsoby extrakce informací z webových dokumentů. Všechny způsoby měly společné to, že vždy na základě nějakých extrakčních pravidel identifikovaly v dokumentu instance relevantních extrahovaných tříd dat. Opomíjeli jsme však způsoby tvorby těchto extrakčních pravidel a mlčky dosud předpokládali jejich ruční tvorbu.

Tuto kapitolu věnujeme stručnému popisu jednotlivých metodik pokud možno co nejvíce automatizované indukce wrapperu. Také se pokusíme tyto metodiky zhodnotit zejména ve vztahu k našemu záměru provádět automatické anotování dokumentů s pomocí extrakční ontologie a na závěr kapitoly navrhneme způsob indukce wrapperu vhodný pro jeho ověření.

3.2 *Známé metody tvorby wrapperů*

3.2.1 *Věštec a PAC analýza*

První metodika indukce wrapperu, které se zde budeme věnovat, je striktně založena na způsobech induktivního strojového učení ([Mitchel, 97]) a ve své práci ji poprvé uvádí Nicolas Kushmerick ([Kushmerick, 97, str. 29]).

Na nejobecnější úrovni je induktivní úlohou vypočtení generalizace z množiny příkladů nějakého neznámého cílového konceptu takové, která (v nějakém smyslu specifickém pro danou doménu) vysvětluje všechna pozorování. Idea je taková, že tato generalizace je považována za dobrou

v případě, že nejen vysvětluje pozorované příklady, ale navíc také vytváří přesné předpovědi.

Na této nejobecnější úrovni může být za úlohu induktivního učení považována většina způsobů tvorby wrapperu, což je také důvod, proč se o nich velmi často hovoří jako o indukci wrapperu.

Pro tvorbu dostatečně kvalitních generalizací tedy Kushmerickův přístup

- a) potřebuje vstup ve formě správně označených příkladů instancí určených k extrakci a
- b) potřebuje těchto příkladů dostatek.

Vstupem tohoto algoritmu je funkce, kterou Kushmerick označuje jako *Oracle* (česky doslovně „věštec“, jinak se překládá jako „učení s učitelem“), a která bez jakýchkoliv vstupních parametrů vrací vždy jeden označený příklad instance určené k extrakci. Z pohledu induktivního algoritmu je funkce Oracle černou skříňkou, v praxi se předpokládá, že její funkci bude plnit uživatel, který instance označí ručně.

Prvním krokem algoritmu tedy je nasbírání dostatečného počtu označených instancí. Na to, co znamená slovo „dostatečný“, používá Kushmerick PAC analýzu, čímž je stanovena přerušovací podmínka pro ukončení sběru.

PAC analýza (písmena PAC jsou zkratkou z „Probably Approximately Correct“, tedy „pravděpodobně přibližně správně“) je podrobně popsána například v [Haussler, 90] nebo [Kearns, Vazirani, 94].

V této metodice jsou pro učení určeny dva parametry, požadovaná přesnost ϵ a požadovaná spolehlivost d , přičemž oba jsou číselné hodnoty v rozsahu $\langle 0,1 \rangle$.

Význam parametrů ϵ a d je ten, že požadujeme, aby chybnost výstupní generalizované hypotézy byla menší než ϵ s pravděpodobností alespoň $1 - d$. Tato podmínka je splněna v případě, že počet označených instancí N splňuje následující nerovnost (což je odvozeno za značného množství předpokladů a omezení v [Kushmerick, 97, str. 217 - 225]):

$$N > \frac{2}{\epsilon} \log \frac{\Psi(R)}{d}, \quad (3.2.1)$$

kde $\Psi(R)$ je funkce délky dokumentu, která závisí na volbě konkrétní třídy wrapperu.

Po nasbírání dostatečného počtu označených instancí se spustí generalizační funkce, kterou v zápětí popíšeme, její úspěšnost však závisí na správné volbě parametrů přerušovací podmínky PAC analýzy, což je dokázáno v [Kushmerick, 97, str. 45 – 54].

Samotná generalizace wrapperu pak probíhá tak, že se, zjednodušeně řečeno, pro všechny označené instance testují inkrementálně všechny možné parametry wrapperu (např. HLRT), dokud se nenarazí na takovou sadu parametrů, která by byla s příklady konzistentní. Konzistencí se zde myslí, že wrapper extrahuje všechny pozitivně označené příklady a žádný příklad označený negativně. Množina wrapperů, které je takto možné testovat, než je nalezena konzistentní kombinace, je sice velká, nicméně konečná.

Jak je asi zřejmé, stupeň automatizace tohoto způsobu tvorby wrapperu závisí na volbě funkce Oracle, neboť ostatní kroky jsou automatizovány plně.

Pokud chceme posoudit vhodnost tohoto typu indukce wrapperu při použití automatizované anotace příkladů, zjistíme, že PAC analýza není k takovému typu úlohy určena. Můžeme ale při nahrazení funkce Oracle množinou automaticky anotovaných příkladů přepsat přerušovací podmínku PAC (3.2.1) do inverzní podoby a určit tak maximální možnou chybu generalizovaného wrapperu na požadované hladině pravděpodobnosti, což také

může mít svůj význam. Toto opačné použití přerušovací podmínky nejspíše ale nebude příliš důvěryhodné, protože se nemůžeme spolehnout, že automaticky anotované instance budou odpovídat skutečnému rozložení instancí v dokumentu.

3.2.2 Interaktivní vizuální tvorba vzorů

Další způsob tvorby wrapperu je využíván v programovém nástroji *Lixto* a jsou jím vytvářeny wrappery v jazyce *Elog* (viz. [Baumgartner et al., 01]). Tvorba wrapperu tímto způsobem probíhá tak, že se interaktivně vytváří vzory v hierarchickém pořadí. Lze tak například definovat vzor <výrobek> a poté definovat podvzor <cena> (vzory jsou zde zapisovány jako elementy XML, proto tento způsob zápisu). Vztah těchto vzorů je potom takový, že se každá extrahovaná instance vzoru <cena> musí vyskytovat uvnitř instance vzoru <výrobek>. V tomto směru tedy vzory zastupují extrakční třídy a zároveň jejich atributy (tyto pojmy budou vysvětleny v následující kapitole).

Každý takovýto vzor je definován jedním nebo více filtry, v tom významu, jak bylo uvedeno v části 2.3.1. Tvorba těchto filtrů probíhá v nástroji *Lixto* následovně:

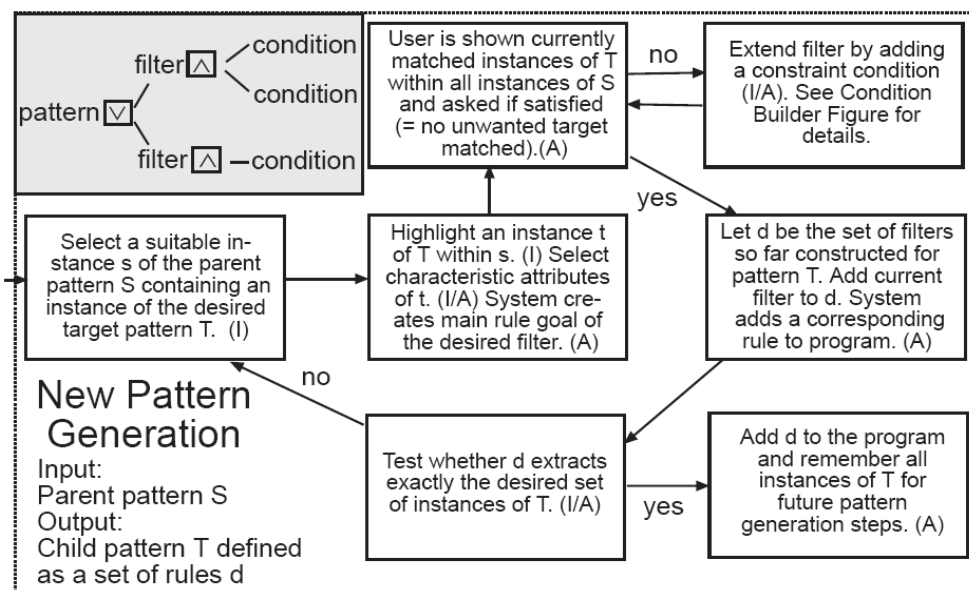
Nejprve uživatel přímo na vzorové stránce označí reprezentativní výskyty instancí zvolených vzorů. Systém pak zobecní cestu ve stromové reprezentaci dokumentu k těmto vybraným instancím (bohužel se však nepodařilo zjistit konkrétní algoritmus, jakým nástroj *Lixto* toto zobecnění provádí, v [Baumgartner et al., 01, s. 3] je pouze zmíněno, že v případě, že se zobecněnou cestu nepodaří nalézt, je tato nahrazena množinou možných elementů) a identifikaci podobných instancí na základě této zobecněné cesty vyjádří jako pravidlo v jazyce *Elog* ve formě filtru.

Dále pak uživatel může k tomuto filtru přidávat další omezující podmínky, což je systémem reprezentováno jako dodatečné cíle uvnitř pravidel tento filtr popisujících. Omezujícími podmínkami mohou být například

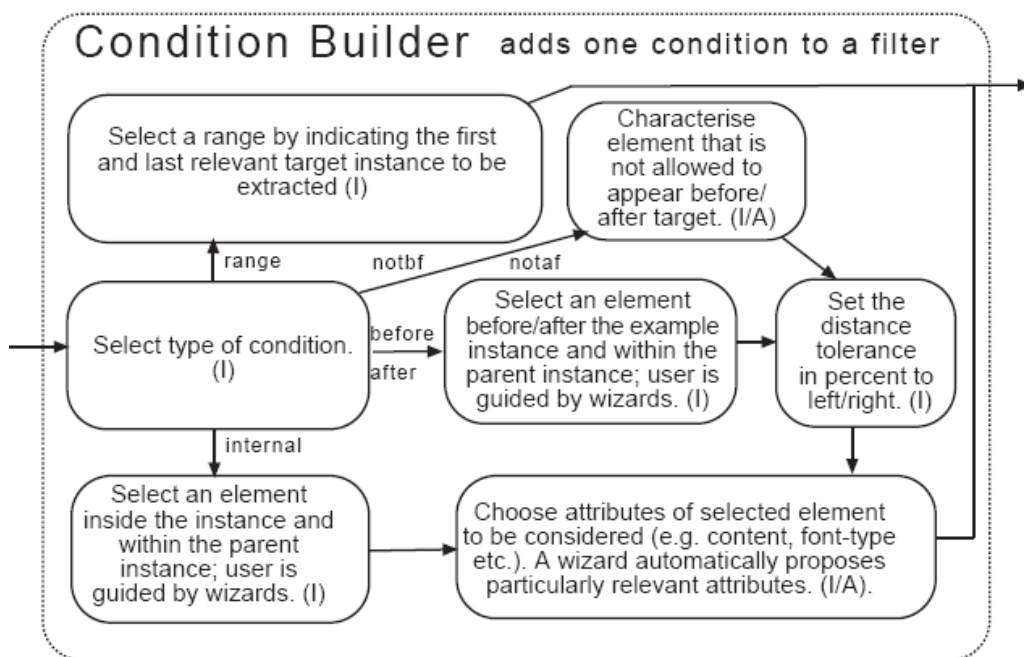
- podmínka **před / po** – tato podmínka vyjadřuje, že se instance daného vzoru může vyskytovat pouze před nebo po nějakém určitém elementu dokumentu.
- podmínka **ne před / ne po** – tato podmínka vyjadřuje, že se instance daného vzoru nesmí vyskytovat poblíž určitého elementu dokumentu.
- podmínka **internosti** – tato podmínka vyjadřuje, že se nějaký konkrétní element musí (nebo nesmí) vyskytovat uvnitř extrahované instance.
- podmínka **rozsahu** – v případě, kdy vzoru odpovídá více instancí, omezuje tato podmínka jejich množinu na podinterval.

Všimněme si tedy, že filtry množinu vybraných instancí rozšiřují, zatímco podmínky tuto množinu omezují, protože extrahovány jsou ty instance, které splňují všechny podmínky alespoň jednoho filtru.

Konkrétní algoritmus interaktivní tvorby jednotlivých vzorů je pak



Obrázek 3.1 - algoritmus tvorby vzoru v nástroji Lixto [Baumgartner et al., 01, s.3]



Obrázek 3.2 - tvorba podmínek v nástroji Lixto [Baumgartner et al., 01, s.3]

znázorněn na obrázku 3.1 a algoritmus tvorby podmínek na obrázku 3.2, s tím, že jsou zde odlišeny kroky automatizované (A) a interaktivní (I). Na počátku existuje implicitně jeden vzor <dokument>, který reprezentuje konkrétní načtený HTML dokument, a na něm uživatel vytváří jednotlivé podvzory, z nichž každý může být definován jedním nebo více filtry. Pro každý filtr pak může uživatel definovat libovolný počet omezujících podmínek.

Pokud máme posoudit tento způsob tvorby wrapperu, tak oproti Kushmerickovu způsobu zde není snaha automaticky hodnotit kvalitu a dostatečnost vstupu od uživatele, nýbrž se z toho, co uživatel poskytne vychází bezpodmínečně. Povšimneme-li si jednotlivých omezujících podmínek, zjistíme, že jsou schopny plně nahradit Kushmerickovy parametry wrapperů (např. podmínka před / po je v podstatě rovnocenná z parametry HT), bohužel se však zde tvorba těchto podmínek ponechává na uživateli, automatizováno je pouze z malé části nastavení parametrů těchto podmínek.

Pokud zvážíme možnosti toho způsobu tvorby ve vztahu k využití automatické anotace dokumentu pomocí extrakční ontologie, nalezneme zde

několik míst, které šlo takto vylepšit. Zaprvé by jistě bylo možné vytvořit strukturu vzorů extrahovaných instancí na základě tříd extrakční ontologie, neboť si tyto v zásadě odpovídají. A zadruhé by také bylo jistě možné nahradit vybírání vzorových instancí uživatelem vzorkem reprezentativních výsledků automatické anotace dokumentu. Po realizaci takového rozšíření by na uživateli zůstal jen výběr omezujících podmínek a ověřování jednotlivých filtrů. Toto rozšíření však překračuje rámec této práce, nicméně by mohlo být vhodným námětem na její navázání.

3.2.3 *Učení XPath výrazů dotazy na fragmenty stromů*

Pokud chceme automaticky vytvářet wrapper založený na extrakci pomocí XPath výrazů, musíme mít způsob, jakým lze tyto XPath výrazy vytvářet. V literatuře lze v zásadě narazit na dva odlišné přístupy (a možná ještě nějaké další), zde probereme první z nich. Předem je však nutno podotknout, že vždy existuje nekonečné množství XPath výrazů popisujících danou množinu elementů, proto také neexistuje jediný správný způsob, jak XPath výraz vytvořit.

Prvním přístupem k automatizované tvorbě XPath výrazů je učení XPath výrazů dotazy na fragmenty stromů, publikované například v [Ceresna, Gottlob, 05].

Předně v tomto přístupu autoři definují dva druhy dotazů na XPath výrazy, jejichž předmětem je vždy dvojice výrazů. Prvním je dotaz na rovnost dvou výrazů, jehož výsledkem je buď odpověď „ano“, pokud jsou množiny uzlů dokumentu nalezené těmito výrazy shodné, nebo pokud tyto množiny shodné nejsou je odpovědí „ne“ spolu s označeným protipříkladem, kdy tomu tak není.

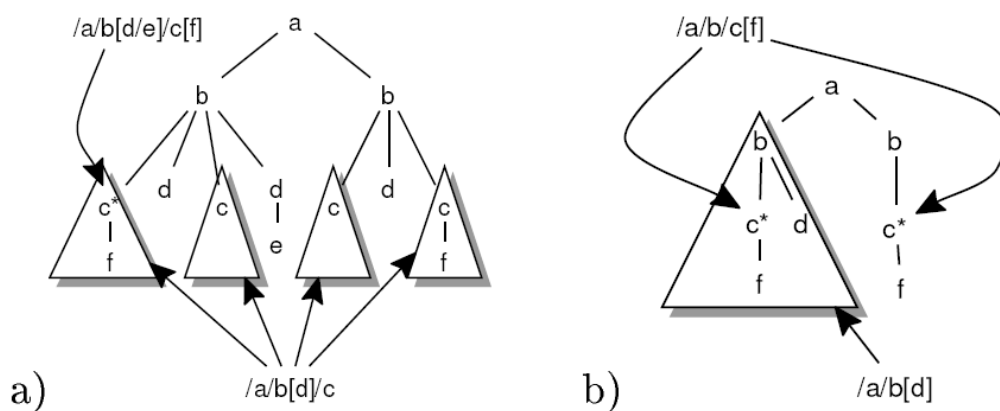
Druhým typem dotazu je dotaz na stromový prefix (v originále „tree-prefix query“). Odpověď na tento dotaz je kladná v případě, že všechny uzly označované prvním výrazem se nalézají v podstromech uzlů označených

druhým výrazem – v tomto smyslu tedy druhý výraz tvoří stromový prefix prvního výrazu. Pokud tato podmínka splněna není, je vrácena záporná odpověď, opět spolu s protipříkladem.

Příklad dotazu na stromový prefix je uveden na obrázku 3.3. Nalevo je varianta, kdy výraz `/a/a[d]/c` tvoří stromový prefix výrazu `/a/b[d/e]/c[f]` a odpověď na tento dotaz je kladná, a napravo varianta, kdy naopak výraz `/a/b[d]` netvoří stromový prefix výrazu `/a/b/c[f]` a odpověď na tento dotaz je záporná spolu s uvedením protipříkladu, zde `c*` označené šipkou napravo.

Samotný algoritmus učení XPath výrazu je následující. V prvním kroku se procházejí části absolutní cesty prvního pozitivního příkladu vybraných uzlů (potažmo extrahované instance) ke kořeni dokumentu sérií dotazů na stromové prefixy, dokud se nenarazí na takový výraz, který by byl stromovým prefixem všem pozitivním příkladům. Tento výraz se vždy hledá ve tvaru `/a1/a2/.../an`. Druhým krokem je pak tvorba omezujících podmínek, která probíhá tak, že se systematicky zkouší přidávat podmínky na výskyt konkrétních elementů, dokud není splněna podmínka ekvivalence s množinou označených vzorových příkladů. Za poznámku stojí fakt, že osa prohledávání dokumentu se zde omezuje na přímého předka.

V tomto modelu není proces označování pozitivních příkladů



Obrázek 3.3 - příklad dotazu na stromový prefix, [Ceresna, Gottlob, 05, s. 3]

vybíraných elementů nijak blíže specifikován, mohl by proto být velmi dobře slučitelný s automatickou anotací dokumentu. Tento model je však určen zejména pro tvorbu extrakčních pravidel pro mnoho dokumentů se stejnou strukturou, přičemž se spoléhá na kompletní anotaci vzorového dokumentu. Protože od automatické anotace dokumentu nelze očekávat, že by byla kompletní, bylo by v tomto kroku nutno počítat s účastí uživatele na opravě a doplnění anotace tohoto vzorového dokumentu.

3.2.4 Učení XPath výrazů zobecněním vzorů pro průchod stromem

Druhým možným přístupem pro učení XPath výrazů je zobecnění vzorů pro průchod stromem dokumentu, což je prozkoumáno v [Anton, 05]. Jak již bylo zmíněno v oddíle 2.3.3, XPath výraz určuje cestu napříč stromem dokumentu. Tato cesta začíná na množině uzlů a končí na jiné množině uzlů, přičemž je tvořena jednotlivými kroky, oddělenými lomítkem. Každý takovýto krok má specifikovaný směr (jinak také osu) prohledávání dokumentu, které určuje, zda se bude pokračovat na předky, potomky nebo sourozence zvažovaných elementů.

Narozdíl od předchozího přístupu, tato metodika bere v úvahu všechny možné směry pohybu napříč dokumentem. Dokument je znázorněn formou stromového grafu a XPath výraz pak znázorňuje cestu napříč grafem. Pro nalezení daného XPath výrazu jsou pak využity metodiky teorie grafů z oblasti lineárního programování pro nalezení nejkratší cesty grafem, konkrétní metodiku zde popisovat nebudeme, neboť je příliš komplexní, více viz [Anton, 05].

Tato metodika pro využití automatického anotování není příliš vhodná, protože příliš spoléhá na označení negativních příkladů, čehož automaticky dosáhneme jen těžko. Naopak oproti předchozí metodice jsou takto vytvořené XPath výrazy jednodušší, obecnější a přesnější.

3.3 Celkové zhodnocení metod

Z výše uvedených metodik tvorby wrapperu se jako nejvhodnější pro použití s automatizovanou anotací dokumentu jeví metodika učení XPath výrazů dotazy na fragmenty stromů, zejména proto, že není závislá na označování negativních příkladů, ale také pro svůj vysoký stupeň automatizace.

Ke zhodnocení navrženého způsobu anotace dokumentu využijeme zjednodušenou analogii tohoto způsobu učení XPath výrazů, přičemž omezíme závěrečnou podmínku ekvivalence. Více se tomuto tématu budeme věnovat v šesté kapitole.

Velmi přínosné by také mohlo být zakomponování automatické anotace dokumentu a širšího využití extrakčních ontologií do metodiky indukce wrapperu využívané v nástroji *Lixto*, neboť by tak zde bylo možné značně snížit stupeň interakce uživatele. Kromě samotné anotace dokumentu by bylo možné využít extrakční ontologii k návrhu struktury vzorů v extrakčním jazyce *Elog* a pravděpodobně také k tvorbě některých filtrů těchto vzorů.

Kapitola 4

NÁVRH ROZŠÍŘENÍ ZÁPISU ONTOLOGIÍ

4.1 *Zaměření kapitoly*

V této kapitole se nyní oprostíme od samotné indukce wrapperů a budeme se blíže věnovat ontologiím.

Nejprve vysvětlíme samotný pojem ontologie, jeho vznik a jeho význam, jak jej chápeme v této práci.

Poté se zaměříme na způsoby zápisu ontologií. Stručně zde probereme různé jazyky pro zápis doménových ontologií, konkrétně to budou, v pořadí, v jakém se vyvíjely, jazyk RDF, RDF schémata a jazyk OWL.

Následně zhodnotíme možnosti jazyka OWL, jakožto v současnosti nejpokročilejšího z ontologických jazyků, pro zápis extrakčních ontologií a navrhneme rozšíření tohoto jazyka o zápis šablon hodnot datotypových vlastností tříd ontologie.

4.2 *Ontologie*

4.2.1 *Původní pojem ontologie*

Termín *ontologie*, vzniklý literárním překladem řeckého slova *Οντολογία* (složenina z „ón“ - jsoucí a „logos“ – řeč), původně pochází z oblasti filosofie. V tomto kontextu je používán pro označení oblasti filosofie zkoumající podstatu bytí, odvětví metafyziky zabývající se identifikací druhů věcí, které skutečně existují, a způsoby, jak je popsat. Například pozorování toho, že svět je složen ze specifických objektů, které mohou být roztříděny do

určitých abstraktních tříd založených na společných vlastnostech těchto objektů, je typickým pohledem oboru ontologie.

4.2.2 *Ontologie v sémantickém webu*

Dnes se však již s pojmem *ontologie* setkáváme i jiném kontextu. Použijeme-li definici z [Antoniou, Harmelen, 04, str. 10] (definice T. R. Grubera upravená R. Studerem), tak můžeme říci, že ontologie v tom významu, v jakém jej chápeme v této práci, je definována takto:

„Ontologie je explicitní a formální specifikace konceptualizace.“

Obecně lze říci, že v tomto kontextu je ontologie formálním popisem nějaké domény, tedy oblasti zájmu. Typicky pak se ontologie skládá z konečného počtu termínů a vztahů mezi těmito termíny. Tyto termíny označují důležité koncepty (třídy objektů) z dané domény.

Vztahy mezi termíny obvykle zahrnují hierarchii tříd. Hierarchie tříd specifikuje fakt, že nějaká třída C je podtřídou jiné třídy C' , pokud každý objekt třídy C je zároveň objektem třídy C' . Tento vztah bývá nazýván vztahem generalizace / specializace.

Kromě hierarchického vztahu tříd mohou ontologie obsahovat další informace jako:

- vlastnosti tříd (např. vztahy tříd jiné, než generalizace / specializace, nebo datotypové vlastnosti)
- omezení hodnot vztahů mezi objekty
- disjunktnost tříd
- omezení kardinality vztahů mezi objekty

Z pohledu sémantického webu takovéto ontologie poskytují sdílení porozumění dané oblasti zájmu, tedy doménových znalostí a jejich konceptů.

4.3 Jazyky pro zápis ontologií

V oblasti umělé inteligence existuje dlouhá tradice vývoje a užívání jazyků pro zápis ontologií. Jsou to základy, na kterých stojí výzkum sémantického webu.

V současnosti jsou v oblasti sémantického webu nejdůležitějšími tyto jazyky (viz [Antoniou, Harmelen, 04]):

- XML – poskytuje syntax pro zápis strukturovaných dokumentů, ale neumožňuje zápis jakýchkoliv sémantických omezení významu těchto dokumentů
- XML Schémata – jazyk pro omezení struktury dokumentů v jazyce XML
- RDF – datový model pro objekty („zdroje“) a vztahy mezi nimi, který poskytuje možnosti jednoduchého vyjádření sémantiky; bývá reprezentován v XML syntaxi
- RDF Schémata – jazyk určený pro popis vlastností a tříd RDF zdrojů s možností pro zápis sémantiky generalizace hierarchie těchto vlastností a tříd
- OWL – bohatší jazyk pro popis vlastností a tříd; oproti RDF schématům umožňuje například zápis vztahů jako je disjunktnost tříd, kardinalita vztahů a rovnost; také obsahuje větší možnosti zápisu vlastností a jejich charakteristik.

Jazyky XML a XML Schémata zde rozebírat nebudeme ze dvou důvodů, zaprvé neobsahují žádné možnosti zápisu sémantiky zaznamenaných dat a zadruhé jsou v praxi většinou považovány spíše za formát zápisu a jeho specifikaci než za samostatné jazyky.

Další tři jazyky rozeberme podrobněji.

4.3.1 RDF

Ačkoliv je RDF (zkratka z „Resource Definition Framework“) často nazýváno jazykem (a i v této práci tak činíme), jedná se ve skutečnosti o datový model, jehož specifikaci nalezneme v [W3C, 04, 1]. RDF je tedy dalším doporučením konsorcia W3. Základním stavebním kamenem tohoto modelu jsou trojice zdroj-vlastnost-hodnota vlastnosti, kterým se v oficiální terminologii říká *tvrzení*.

Na první člen trojice RDF tvrzení, zdroj, můžeme nahlížet jako na objekt nebo „věc“ (i když z filosofického hlediska by jistě bylo správnější nazývat jej *subjektem*), o které chceme sdělovat nějaké znalosti. Každý zdroj má svůj jednoznačný identifikátor, podle terminologie webových jazyků označený jako URI (Universal Resource Identifier), který může být ve formě webové adresy URL nebo nějakého jiného unikátního identifikátoru. Existence URI identifikátoru však ještě nezaručuje přístup k tomuto zdroji, neboť koncept těchto identifikátorů nezahrnuje pouze umístění na webu, ale také tak rozdílné oblasti jako telefonní čísla, ISBN označení publikací, geografická umístění a další, nebudeme však zde zabíhat do detailů.

Vlastnosti (jinak též atributy) jsou v RDF chápány jako speciální druh zdroje a jako takové jsou také označeny unikátním identifikátorem URI. Koncept využití URI identifikace pro všechny zdroje je celkem podstatný, protože tím získáváme pro každý zdroj unikátní označení, v ideálním případě celosvětové, což podstatně redukuje problém vzniku homonym. Vlastnosti popisují druh vztahu mezi jinými zdroji.

Jak již bylo řečeno, tvrzení v RDF je tvořeno trojicí zdroj-vlastnost-hodnota vlastnosti. Význam tohoto tvrzení je ten, že objekt označený jako „zdroj“ má vlastnost označenou „vlastnost“ a její hodnotou je „hodnota vlastnosti“. Hodnotou vlastnosti může být buďto jiný zdroj, pak hovoříme o objektové vlastnosti, nebo může být hodnotou vlastnosti *literál*, což je

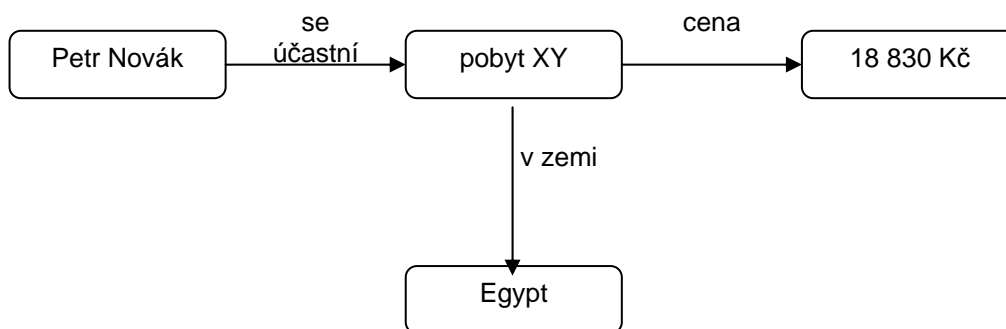
primitivní hodnota (v definici chápaná jako řetězec znaků), pak hovoříme o datotypové vlastnosti.

Existují tři možné způsoby zápisu tvrzení v RDF, přičemž má každé své výhody a samozřejmě také nevýhody.

První možností, jak zapsat tvrzení v podobě RDF trojice, spočívá ve využití logických vlastností takového tvrzení. Označíme-li si tvrzení jako trojici (x, P, y) , můžeme jej zapsat také formou logické formule $P(x, y)$, kde P je binární predikát určující vztah objektu x k objektu y . V RDF však takto zapsané mají smysl pouze binární predikáty.

Druhou možností je využití grafické interpretace. Výsledkem tohoto způsobu zobrazení je graf, ve kterém uzly znázorňují jednotlivé zdroje a orientované spojnice mezi nimi pak znázorňují vztahy mezi zdroji, tedy vlastnosti objektu, ve kterém spojnice začíná, s hodnotou objektu, ve kterém spojnice končí. Příklad takového grafu je na obrázku 4.1. V oblasti umělé inteligence tento druh zobrazení označujeme výrazem *sémantická síť*. Výhodou tohoto grafického zobrazení je velmi dobrá čitelnost pro člověka.

V praxi se však dnes nejvíce využívá třetí způsob zápisu a to zejména kvůli své velmi dobré strojové čitelnosti. Jedná se o zápis RDF tvrzení pomocí značkovacího jazyka XML. V souladu s tímto, je RDF dokument reprezentován jako XML element označený tagem `<rdf:RDF>`. Obsahem



tohoto elementu je množství popisů, označených elementy `<rdf:Description>`. Každý z těchto popisů označuje jedno tvrzení týkající se zdroje, který může být identifikován třemi způsoby:

- atributem *about* odkazujícím na existující zdroj
- atributem *ID* vytvářejícím nový zdroj
- bez označení, což vytvoří anonymní zdroj

Pokud tedy zapíšeme v XML syntaxi stejný RDF dokument, jaký byl na obrázku 4.2, bude vypadat takto:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/TR/rdf-syntax-grammar"
  xmlns:trip="http://www.nej.akadomena.cz/zajezdy-ns">
  <rdf:Description rdf:ID="Petr Novák">
    <trip:ucastni zajezdu>
      <rdf:Description>
        <trip:cenazajezdu>
          18 830 Kč
        </trip:cenazajezdu>
        <trip:cilovazeme rdf:resource="Egypt"/>
      </rdf:Description>
    </trip:ucastni zajezdu>
  </rdf:Description>
</rdf:RDF>
```

Ačkoliv je tedy tento zápis v XML dobře čitelný strojově, zdaleka již není tak přehledný pro člověka jako bylo grafické znázornění. Stručně se pokusíme tento zápis vysvětlit.

První řádka určuje, že používáme XML. Atributy elementu `<rdf:RDF>` určují jmenné prostory, které se uvnitř něj budou používat. Jmenný prostor *rdf:* odkazuje na definici jazyka RDF, jmenný prostor *trip:* zde odkazuje na smyšlenou definici domény pobytových zájezdů.

První element `<rdf:Description>` označuje tvrzení o novém zdroji „Petr Novák“. Tomuto zdroji je přiřazena vlastnost, konkrétně „ucastnizajezdu“.

Hodnota této vlastnosti, tedy konkrétní zájezd, kterého se Petr Novák účastní, je definována pomocí dalšího elementu `<rdf:Description>`. Tento element je uveden bez atributů, vytváří se tedy „zájezd“ jako anonymní zdroj. Dále jsou tomuto anonymnímu zdroji určeny další vlastnosti: „cenazajezdu“, jejíž hodnotou je literál „18 830 Kč“ a „cilovazeme“, jejíž hodnotou je zdroj s identifikátorem „Egypt“.

Za povšimnutí stojí druhý element `<rdf:Description>`, který kromě toho, že definuje anonymní zdroj, je také vnořený. V této syntaxi zápisu RDF lze jednotlivá tvrzení vnořovat tímto způsobem. Samozřejmě by bylo možné určit tomuto anonymnímu zdroji nějaký identifikátor a vyjádřit jej samostatným elementem `<rdf:Description>`, tedy nikoliv vnořeným jinému popisu. Z toho je vidět, že pro identický obsah RDF dokumentu mohou existovat různé zápisy, což dále zesložituje čitelnost tohoto formátu člověkem.

Kromě čitelnosti má však RDF další řadu nevýhod a nedostatků. Například již bylo zmíněno, že RDF umožňuje pouze tvorbu binárních predikátů, v mnoha případech by však bylo užítí predikátu s více argumenty přirozenější. Predikáty s více argumenty lze převést na binární predikáty zavedením abstraktních zdrojů, ale tím vzrůstá nejen celková složitost modelu, ale také jeho nepřirozenost.

Dalším velkým problémem RDF jsou datové typy literálů. RDF samo o sobě neumožňuje definici datových typů, proto bývá k tomuto účelu využíváno možností XML Schémat. Pro plnohodnotný jazyk toto ale není ideální řešení.

V RDF lze vytvářet tvrzení zcela jiného charakteru, než bylo doposud diskutováno. Jednak, protože je vlastnost považována za speciální typ zdroje, je možné vytvářet tvrzení o vlastnostech a jednak je možné vytvářet tvrzení o samotných tvrzeních. Tento mechanismus se nazývá *reifikace*.

Velkou výhodou RDF je, že má poměrně velkou vyjadřovací schopnost, na které lze dále stavět a také fakt, že jej lze jednoduše převádět na sémantické

mapy. RDF je dnes přijatým standardem a výhody ukládání sémantických modelů ve formátu RDF lze přirovnat k výhodám ukládání jiných dat ve formátu XML.

4.3.2 *RDF Schémata*

RDF, tak jak bylo nastíněno, je univerzálním jazykem pro zápis datových modelů, který dovoluje uživateli popsat zdroje a jejich vztahy s použitím vlastního slovníku. RDF samotné netvoří žádné předpoklady o konkrétní doméně a dokonce ani nedefinuje žádnou sémantiku dané domény. Uživateli je ponechána možnost, aby tak učinil s použitím RDF schémat. RDF schémata jsou tedy navržena k omezení tvrzení v dané doméně a jejich název byl zvolen podle toho, že k RDF mají podobný vztah, jaké mají XML schémata k XML. Definici tohoto jazyka je možno nalézt v [W3C, 04, 2], jedná se tedy opět o doporučení konsorcia W3.

Jak tedy popíšeme konkrétní doménu? V příkladu v minulé části zabývající se jazykem RDF jsme byli schopni popsat skutečnost, že „Petr Novák se účastní pobytového zájezdu v Egyptě za 18 830 Kč“. Představme si ale, že kromě Petra Nováka a zájezdu do Egypta bychom se chtěli bavit o vlastnostech zájezdů a účastníků zájezdu obecně. Rozdíl je v tom, že se zde narodí od konkrétních individuálních objektů bavíme o *třídách*, které definují typy objektů. Konkrétní existující objekty jsou pak považovány za instance těchto tříd.

Důležitým přínosem zavedení konceptu tříd je možnost omezení použitím schématu toho, co může být obsahem tvrzení v RDF dokumentu. V programovacích jazycích bývá k omezení možností výrazů obvykle použito typování, čímž jsou určeny typy proměnných, které lze v daném výrazu použít. V zápisu doménových znalostí je potřeba podobná, a tak RDFS (zkratka pro RDF schémata) umožňuje omezit třídy objektů, kterých se daná vlastnost týká a to jak třídy, které mohou vlastností disponovat (zde hovoříme o omezení domény vlastnosti), tak i třídy, které mohou tvořit hodnoty této vlastnosti.

Pokud se vrátíme k příkladu, můžeme sestavit omezení že vlastnost „účastní se zájezdu“ může mít jen objekt třídy „člověk“, ale také omezení, že hodnotou vlastnosti „cílová země“ může být jen instance třídy „země“.

Při zavedení pojmu tříd zavádí RDFS také hierarchický vztah mezi třídami, a sice dědičnost. Formálně má v RDFS dědičnost stejný význam jako v běžných objektových modelech, přičemž vícenásobná dědičnost je zde povolena (což má za následek možnost vzniku nekonzistencí v modelu).

Zavedení dědičnosti tříd v tomto pojetí datového modelu má však jeden zajímavý důsledek. Protože v RDF jsou vlastnosti považovány za speciální druh zdroje a pro zdroje zde zavádíme třídy, zavádíme současně také dědičnost vlastností. Kromě toho, že tedy nějaká třída může být podtřídou jiné třídy, může být také nějaká vlastnost podvlastností jiné vlastnosti, což může mít svůj význam. V příkladu s pobytovými zájezdy můžeme zavést například vlastnost „být delegátem na zájezdu“ jako podvlastnost vlastnosti „účastnit se zájezdu“. Budeme-li pak mít tvrzení, že „Pan XY je delegátem na zájezdu“, můžeme zároveň prohlásit, že je také účastníkem zájezdu. Obecně můžeme říci, že vlastnost P je podvlastností vlastnosti Q , pokud platí $P(x,y) \Rightarrow Q(x,y)$.

Výše uvedené vlastnosti RDFS lze zapsat v RDF, k čemuž se používá XML syntaxe (někdy se tak hovoří o „vrstvách“ sémantických jazyků). Z hlediska RDF tak jmenný prostor RDFS definuje celou řadu tříd a vlastností, o které pak lze zápis samotného RDF dokumentu doplnit. Kromě toho jsou definovány základní třídy (např. třída tříd, třída vlastností) a základní vlastnosti (např. vlastnost „být podtřídou“). Těchto a podobných prvků je ve jmenném prostoru definována celá řada, pro úplnější seznam můžeme odkázat na [W3C, 04, 2] nebo na [Antoniou, Harmelen, 04].

Nejzásadnějším přínosem jazyka RDFS však je to, že přináší do dokumentu již jistou, byť omezenou, možnost zápisu sémantiky. Oproti RDF, které tedy „pouze“ datovým modelem, můžeme již RDFS považovat za jistou formu ontologického jazyka.

4.3.3 OWL

Vyjadřovací schopnosti jazyků RDF a RDFS, které jsme probrali v předchozích částech, jsou poněkud omezené. RDF je omezeno na binární predikáty a RDFS je omezeno na třídy, vlastnosti, jejich hierarchii a omezení.

Web Ontology Working Group, skupina, která je součástí aktivit konsorcia W3 spojených se sémantickým webem identifikovala řadu charakteristických situací, kdy by vhodný ontologický jazyk pro použití v oblasti sémantického webu vyžadoval mnohem větší vyjadřovací schopnost než RDF a RDFS, například ve [W3C, 01]. Tato skupina také formulovala požadavky na tento vhodný ontologický jazyk v několika bodech, ty jsou následující:

- dobře definovaná syntaxe
- formální sémantika
- vhodnost výrazů
- efektivní podpora usuzování
- dostatečná vyjadřovací síla

Důležitost dobře definované syntaxe je zřejmá a známá z jiných programovacích a značkovacích jazyků, je totiž nutnou podmínkou pro strojovou zpracovatelnost zapsaných informací. RDF i RDFS ji v tomto smyslu dobře definovanou mají. Samozřejmě zůstává otázkou, jestli syntaxe RDF založená na XML je dostatečně uživatelsky přívětivá. Tato nevýhoda však není tak značná, protože časem jistě uživatelé budou ontologie vytvářet s použitím nějakého intuitivního nástroje nejspíše graficky namísto přímého zapisování kódu.

Požadavek na formální sémantiku je požadavkem, aby bylo možno popsat význam zapsaných znalostí přesně. Slovo přesně zde označuje fakt, že

sémantika nesmí odkazovat na subjektivní intuici ani nesmí být možné ji interpretovat různými způsoby.

Existence formální sémantiky je předpokladem pro podporu usuzování nad ontologickými znalostmi. Automaticky lze usuzovat například na příslušnost instance k třídě, rovnost tříd nebo konzistenci zkoumané ontologie. podpora automatického usuzování je důležitá v případě, že budeme brát v úvahu možnost spojování ontologií z více různých zdrojů nebo od různých autorů při tvorbě rozsáhlejších ontologií.

RDF a RDFS umožňují modelovat jen některé z ontologických znalostí. Existují případy, na které užití těchto jazyků nedostačuje, například:

- lokální omezení hodnot vlastností – v RDFS můžeme omezit hodnoty, jakých může daná vlastnost nabývat napříč celou doménou, nemůžeme v něm ale omezit hodnoty této vlastnosti pro subjekty z konkrétní třídy
- disjunktnost tříd – v RDFS nelze vyjádřit situaci, kdy jsou třídy neslučitelné, tedy disjunktní
- booleovské kombinace tříd – RDFS neumožňuje vyjádřit třídu jako sjednocení, průnik nebo doplněk dvou jiných tříd
- kardinalitní omezení – omezení počtu hodnot, kterých může nějaká vlastnost nabývat nelze také s použitím RDFS vyjádřit
- speciální charakteristiky vlastností – RDFS neobsahuje možnosti zápisu některých speciálních charakteristik vlastností, jako je tranzitivnost, unikátnost nebo skutečnost, že jsou vůči sobě dvě vlastnosti inverzní

Tyto požadavky a nedostatky jazyka RDFS vedly k potřebě vzniku bohatšího ontologického jazyka. Postupně se tak nezávisle na sobě vyvinuly jazyky známé jako OIL a DAML-ONT a jejich spojením pak dále jazyk

DAML-OIL. Jazyk DAML-OIL pak vzala za základ Web Ontology Working Group konsorcia W3 při snaze vytvořit jednotný ontologický jazyk sémantického webu, který dostal označení OWL, jako přesmyčka zkratky z „Web Ontology Language“.

V ideálním případě by měl být jazyk OWL rozšířením jazyka RDFS a to takovým, které by mělo takovou vyjadřovací schopnost, že by podporovalo všechny výše uvedené požadavky. Ontologické jazyky bohužel však obecně mají tu nepříjemnou vlastnost, že čím větší je jejich vyjadřovací schopnost, tím menší jsou možnosti usuzování nad doménami v nich zapsanými.

Proto byl také OWL navrhnut jako tři dílčí verze jazyka, každá s určitým cílem.

Kompletní verze jazyka je nazývána OWL Full a umožňuje použití všech prvků jazyka spolu s kombinací těchto prvků s prvky RDF a RDF schémat. Výhodou této verze jazyka je její plná kompatibilita, jak sémantická, tak i syntaktická, s RDF – každý RDF dokument je současně i OWL Full dokumentem a každý závěr plynoucí z RDFS schématu je současně platným závěrem OWL Full dokumentu. OWL Full je však natolik obecný a rozsáhlý, že prakticky znemožňuje jakékoliv usuzování, natož usuzování efektivní.

Za účelem dosažení výpočetní efektivnosti byla navržena verze jazyka s označením OWL DL (DL je zkratkou pro „Description Logic“), která spočívá v podstatě jen v omezení možností použití konstruktů jazyka OWL Full. V první řadě je zde zakázáno použití prvků jazyka na prvky jazyka samé, čímž je zajištěno, že jazyk odpovídá zákonům popisné logiky. Omezení je samozřejmě více, jejich efektem je, že tato verze jazyka dovoluje logické usuzování. Těmito omezeními však jazyk ztrácí kompatibilitu s RDF.

Poslední verzí jazyka je OWL Lite, což je další omezení verze OWL DL. Například zde není umožněno zápisu disjunktnosti tříd ani kardinality hodnot vlastností. Výhodou této verze je její jednoduchost, nevýhodou samozřejmě její omezenost.

Protože ve své podstatě jazyk OWL alespoň v plné verzi vychází z jazyka RDF, využívá se pro jeho zápis většinou RDF syntaxe založená na XML.

Kořenovým elementem dokumentů v jazyce OWL, které někdy označujeme prostě jako OWL ontologie, je element `<rdf:RDF>`, který definuje jmenné prostory `rdf`, `rdfs`, `xsd` a `owl`. Poté může následovat element `<owl:Ontology>`, uvnitř kterého lze zapsat metadata týkající se ontologie, přičemž zde také lze specifikovat import jiných ontologií. Importem se myslí prosté zahrnutí importovaných ontologií do stávající. Hlavička dokumentu tedy může vypadat například takto:

```
<rdf: RDF
  xml ns: rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xml ns: xsd="http://www.w3.org/2001/XMLSchema#"
  xml ns: rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml ns: owl="http://www.w3.org/2002/07/owl#" >

  <owl: Ontology rdf: about="" >
    <owl: imports rdf: resource="http://www.nej akademena.cz/x" />
  </owl: Ontology>

  ...

</rdf: RDF>
```

Další podstatnou konstrukcí jazyka OWL je element `<owl:Class>`, kterým lze zapsat vytvoření nové třídy nebo odkaz na existující třídu stejně jako se v RDF odkazovalo na zdroje, použitím unikátního identifikátoru. Uvnitř tohoto konstruktu pak lze uvést speciální vlastnosti této třídy, jako například disjunktnost s jinou třídou nebo dědění od jiné třídy. Zápis pak může vypadat takto:

```
<owl: Class rdf: ID=" tri da_A" >
  <owl: disjoi ntWi th rdf: resource=" #tri da_B" />
</owl: Class>

<owl: Class rdf: ID=" tri da_C" >
  <owl: subCl assOf rdf: resource=" #tri da_A" />
</owl: Class>
```

V jazyce OWL jsou dvě předdefinované třídy, jmenovitě *owl:Thing* a *owl:Nothing*. První je nejobecnější třídou, která zahrnuje vše, druhá je prázdnou třídou. Obecně pak lze říci, že všechny třídy jsou podtřídou třídy *owl:Thing* a třída *owl:Nothing* je podtřídou všem třídám.

Co se týče zápisu vlastností tříd, jsou v jazyce OWL rozlišeny dva druhy.

- objektové vlastnosti, které jsou vztahem objektu k jiným objektům
- datotypové vlastnosti, které jsou vztahem objektu k datotypovým hodnotám

Datotypové hodnoty jsou zde obdobou literálů v RDF, jedná se tedy o primitivní hodnoty. OWL nemá předdefinované žádné datové typy ani neobsahuje možnosti jejich definice, nicméně zde umožňuje použití definice datového typu odkazem na XML schéma. Takto lze však odkázat pouze na základní datové typy, jako řetězec, datum, číslo nebo logickou hodnotu, složitější konstrukty XML schémat zde nejsou povoleny.

U každé vlastnosti je pomocí elementu `<rdf:domain>` definována doména, které se tato vlastnost týká (jinými slovy třída, která touto vlastností disponuje), a pomocí elementu `<rdf:range>` její rozsah, tedy hodnoty, kterých může vlastnost nabývat. Takto může být uvedeno i více domén a rozsahů, v tom případě se bere v úvahu jejich průnik.

Datotypovou vlastnost v jazyce OWL zapíšeme takto. Jak je vidět, informace o rozsahu můžeme vynechat:

```
<owl:DatatypeProperty rdf:ID="vlastnost_A">  
  <rdfs:domain rdf:resource="#trida_A"/>  
</owl:DatatypeProperty>
```

Zápis objektové vlastnosti je následující. Lze využít i speciálních konstruktů pro zápis vlastností této vlastnosti:

```
<owl:ObjectProperty rdf:ID="vlastnost_B">
  <rdfs:domain rdf:resource="#trida_A"/>
  <rdfs:range rdf:resource="#trida_B"/>
  <owl:subPropertyOf rdf:resource="#vlastnost_C"/>
</owl:ObjectProperty>
```

Jazyk OWL obsahuje celou řadu dalších možností, počínaje omezením hodnot vlastností a dalšími speciálními vlastnostmi tříd a vlastností, přes zápis booleovských kombinací tříd a vyjmenovaných tříd až po možnosti uložení metadat a informací o verzi ontologie. Tyto možnosti však nejsou v tomto bodě podstatné, neboť je při tvorbě extrakčních ontologií prozatím nevyužijeme.

4.4 Rozšíření jazyka OWL

Zde je namístě zopakovat a zpřesnit, co myslíme pojmem *extrakční ontologie* a jaké pohnutky nás vedou k uvažování o rozšíření jazyka OWL.

Jak již bylo výše naznačeno, můžeme z formálního hlediska považovat ontologie za jistý druh konceptuálního datového modelu, který však kromě definice tříd objektů, jejich vztahů a jejich omezení obsahuje také informace o tom, kde a za jakých podmínek se mohou tyto objekty vyskytovat a jejich vlastnosti nabývat hodnot. Je jistě zřejmé, že pokud by se podařilo tuto dodatečnou vlastnost dostatečně zpřesnit, bylo by možné vhodně formulovanou ontologii využít nejen k vysvětlení významu dat, ale také k jejich identifikaci. V ideálním případě by mělo být možno pomocí takovéto ontologie identifikovat instance konkrétních tříd v nestruturovaných a polostruturovaných dokumentech, jakými jsou například webové stránky, což je typická úloha extrakce informací.

Extrakční ontologii tedy nazveme takovou ontologií, která je určena k použití v procesu extrakce informací.

Nyní se podíváme na podmínky, za jakých můžeme ontologii v jazyce OWL považovat za extrakční ontologii, nejlépe to půjde na příkladu. Navážeme na příklad pobytových zájezdů a zapíšeme definici třídy pobytového zájezdu:

```
<?xml version="1.0"?>

<rdf: RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#" >

  <owl:Ontology rdf:about="" />

  <owl:Class rdf:ID="Zajezd" />

  <owl:DatatypeProperty rdf:ID="cena">
    <rdfs:domain rdf:resource="#Zajezd" />
  </owl:DatatypeProperty>

  <owl:ObjectProperty rdf:ID="cilovazeme">
    <rdfs:domain rdf:resource="#Zajezd" />
    <rdfs:range rdf:resource="#Zeme" />
  </owl:ObjectProperty>

</rdf:RDF>
```

Tímto zápisem jsme definovali v ontologii třídu pro zájezd, která má definovanou jednu datotypovou vlastnost pro cenu a jednu objektovou vlastnost pro cílovou zemi zájezdu, přičemž pro hodnoty vlastnosti cílová země je zde uvedena (smyšlená) třída „Zeme“.

Podíváme-li se na tuto ontologii z hlediska úlohy extrakce informací, zjistíme, že se nedopustíme tak velkého prohřešku, pokud nahradíme objektovou vlastnost pro zemi určení vlastností datotypovou. Tato záměna sice není zcela korektní z hlediska sémantiky, ale přípustná, pokud uvážíme, že hodnotou této vlastnosti má být část dat extrahovaná z webové stránky.

Extrakční úlohu tedy můžeme ztotožnit s identifikací hodnot datotypových vlastností tříd uvnitř dokumentu, potažmo na webové stránce.

Ontologie zapsaná v prostém jazyce OWL, ať již v kterékoliv jeho verzi, má však pouze omezené možnosti popisu hodnot těchto vlastností a neobsahuje tudíž informace nutné k identifikaci těchto hodnot uvnitř dokumentu.

Abychom odstranili tento nedostatek, zavedeme rozšíření ontologie o možnost zápisu *šablon* pro hodnoty datotypových atributů. Abychom se vyhnuli ambiguitě v zápisu ontologie, zavedeme pro šablony nový jmenný prostor, který označíme *ot* (jako zkratku pro „Ontology Template“) a definujeme například jako

```
xmlns:ot="http://st.vse.cz/~XNEKM06/ontologytemplates#".
```

Šablony samotné pak budeme zapisovat v elementu vnořeném elementu příslušné datotypové vlastnosti. Pro takový element šablony budeme používat označení `<ot:Template>` a umístění v kontextu ontologie pak bude následující:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:ot="http://st.vse.cz/~XNEKM06/ontologytemplates#">
  <owl:Ontology rdf:about="" />
  <owl:Class rdf:ID="Zajezd" />

  <owl:DatatypeProperty rdf:ID="cena">
    <rdfs:domain rdf:resource="#Zajezd" />
    <ot:Template ...>
    ...
  </ot:Template>
</owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="cilovazeme">
    <rdfs:domain rdf:resource="#Zajezd" />
    <ot:Template ...>
    ...
  </ot:Template>
</owl:DatatypeProperty>
</rdf:RDF>
```

Tímto zápisem jsme tedy rozšířili již výše uvedenou jednoduchou ontologii pro pobytové zájezdy. Vlastnost „cilovazeme“ jsme nahradili

datotypovou vlastností a pro obě vlastnosti třídy „Zájezd“, tj. cílovou zemi a cenu, jsme zavedli šablony.

V tomto příkladu jsou prozatím atributy a obsahy elementů pro tyto šablony vynechány. Možnostem náplně těchto šablon a jejich využití k identifikaci relevantních hodnot v dokumentu se budeme věnovat v následující kapitole. Zde však zavedeme jedno důležité omezení, jehož důvod ozřejmíme později, a sice že každé datotypové vlastnosti bude takto možné přiřadit nejvýše jedinou šablonu.

Kapitola 5

NÁVRH VYUŽITÍ ROZŠÍŘENÉ ONTOLOGIE PŘI

INDUKCI WRAPPERU

5.1 *Zaměření kapitoly*

V minulé kapitole jsme se věnovali zápisu ontologií a navrhli jsme rozšíření jazyka OWL o možnost zápisu šablon pro popis hodnot datotypových vlastností objektových tříd a v této kapitole se budeme věnovat možnosti využití těchto šablon při indukci wrapperu.

Jak bylo nastíněno v kapitole 3, existuje několik přístupů k automatizované tvorbě wrapperů. Některé z těchto metod spoléhají na vstup uživatele méně, některé více, všechny z nich však do jisté míry fungují na principu zobecnování příkladů výskytu instancí extrahovaných tříd, přičemž poskytování těchto příkladů je ponecháno na uživateli. Příklady instancí se v těchto indukčních metodikách získávají buďto přímým dotazem na uživatele nebo s použitím anotovaného dokumentu, ve kterém jsou již vzorové instance vyznačeny.

Naším cílem je navrhnout takovou strukturu šablon datotypových hodnot, aby je bylo možné využít k automatické anotaci dokumentů, tedy k identifikaci instancí dané třídy uvnitř dokumentu. Tyto automaticky anotované dokumenty by bylo možno poskytnout jako vstup indukčním algoritmům, čímž by se snížil stupeň nutné interakce uživatele při tvorbě wrapperu.

V této kapitole tedy nejprve rozebereme navrhovanou strukturu šablon a možnosti jejich parametrizace. Poté navrhne inferenční mechanismus pro identifikaci podezřelých hodnot jednotlivých vlastností a také pro skládání

složených vzorů. Na závěr kapitoly rozebereme možné nevýhody a omezení tohoto přístupu.

5.2 Šablony a vzory

Nejprve si zavedeme pojmy *šablona* a *vzor* ve vztahu k hodnotám datotypových atributů.

Vzorem v tomto smyslu nazveme obecné pravidlo, pro které lze pro každou spojitou část dokumentu (ve smyslu řetězce znaků nebo slov) určit, zda jej splňuje či nikoliv, popřípadě s jakou mírou. *Šablonou* pak nazveme takovou kombinaci vzorů, pro níž také lze pro každou část dokumentu určit, zda jí vyhovuje či nikoliv, popřípadě s jakou mírou. Šablonu tedy můžeme považovat za speciální případ vzoru.

Jak je z výše uvedené definice zřejmé, může některým ze vzorů, z nichž se šablona skládá, být další šablona. Šablony lze tedy tímto způsobem skládat a rekurzivně zanořovat, což může mít svůj význam.

Každou shodu vzoru s částí dokumentu, tedy každé místo v dokumentu, kterému bude nějaký vzor vyhovovat, budeme považovat za podezřelého dílčího kandidáta na výskyt hodnoty datotypové vlastnosti, jíž je vzor přiřazen.

Jak bylo uvedeno ve vymezení pojmů, šablona je tvořena kombinací vzorů. Aby bylo možno určit v jaké míře dané místo v dokumentu vyhovuje této šabloně, bude nutné skládat dílčí kandidáty vzniklé shodou jednotlivých dílčích vzorů, jimiž je šablona tvořena. Těchto způsobů skládání existuje několik, k čemuž se dostaneme později, a žádný z nich nelze preferovat. Způsob skládání dílčích kandidátů určíme pomocí parametrizace šablony.

Nyní se dostáváme k důvodu onoho omezení, a sice možnosti přiřadit každé datotypové vlastnosti pouze jedinou šablonu. Pokud bychom datotypové vlastnosti přiřadili více šablon, museli bychom nějakým způsobem skládat jimi

označené kandidáty na výskyt hodnot vlastností a to bez možnosti výběru způsobu jejich skládání. V případě potřeby přiřazení více šablon jedné vlastnosti je tedy budeme muset zahrnout uvnitř jedné nadřazené šablony, u níž tento vybraný způsob skládání určíme.

Nyní se blíže zaměříme na obecné způsoby vyhodnocování shody vzorů a jejich skládání. Poté navrhne několik konkrétních vzorů a na závěr kapitoly o vzory rozšíříme příklad ontologie zájezdů.

5.3 *Jednoduché vzory*

Při vyhodnocování shody jednotlivých vzorů se potýkáme s úlohou odvození míry jistoty označení dílčího kandidáta na výskyt hodnoty vlastnosti z jiných dílčích vstupů. Vzor v tomto pojetí chápeme jako algoritmus, který dokáže podle vstupních parametrů určit pro každé místo v dokumentu míru shody vzoru, s jakou těmto parametrům odpovídá.

Zde se tedy střetáváme se dvěma různými termíny. Prvním je *míra shody vzoru*, která reprezentuje jistotu, s jakou vzor dané místo v dokumentu označí. Druhým je pak *míra jistoty označení dílčího kandidáta na výskyt hodnoty vlastnosti*, která reprezentuje jistotu, s jakou je dané místo v dokumentu skutečně výskytem hodnoty vlastnosti objektu extrahované třídy, dáno pouze izolovaným pozorováním výskytu daného vzoru nezávisle na ostatních vzorech. V tomto smyslu označení dílčího kandidáta na výskyt hodnoty vlastnosti nazveme výrazem *evidence vzoru* a druhý z uvedených termínů je tedy synonymem pro *míru jistoty evidence vzoru*.

příklad: Budeme uvažovat jednoduchý vzor, například v podobě výskytu řetězce „cena:“. Pokud se v dokumentu skutečně narazí na výskyt řetězce „cena:“, míra shody vzoru potom bude vysoká (přesně bude rovna 1). Pokud si budeme v tomto případě jisti platností vzoru (tedy že vzor je přesný), bude míra jistoty evidence vzoru také vysoká. U tohoto jednoduchého vzoru lze

předpokládat pouze výsledek shoda / neshoda, později ale ukážeme vzory, které mohou nabývat různých měř shody.

5.3.1 Shoda vzoru a evidence vzoru

Intuitivně by tedy *míra shody vzoru* a *míra jistoty evidence vzoru* měly korespondovat, a to podle pravidla, které lze slovy vyjádřit takto:

„Pokud vzor dané místo v dokumentu označí, je toto místo skutečně důležitým kandidátem na výskyt hodnoty vlastnosti odpovídající tomuto vzoru.“

Toto pravidlo můžeme jednoduše formalizovat. Přiřadíme míře shody vzoru označení A a míře jistoty evidence vzoru označení E . Výše uvedené pravidlo pak můžeme přepsat jako

$$A \rightarrow E. \quad (5.3.1)$$

Takto zapsané pravidlo lze již považovat za formu usuzování, neboli inference, konkrétně usuzujeme z hodnoty A usuzujeme na hodnotu E . Existuje několik dobře propracovaných inferenčních modelů, v podstatě máme v této oblasti na výběr mezi Bayesovskou pravděpodobnostní inferencí, frekvenční statistickou inferencí, fuzzy inferencí a několika dalšími.

Pro zbytek této kapitoly si zvolíme model fuzzy inference ([Hájek, 98]) pro její relativní jednoduchost ve srovnání s ostatními. Podstatou tohoto typu inference je využití vlastností fuzzy logiky, což je rozšíření klasické booleovské logiky na spojitý interval pravdivostních hodnot $\langle 0, 1 \rangle$.

Při využívání fuzzy logiky tedy můžeme A a E definovat jako logické výroky:

- A – „Vzor dané místo v dokumentu označil.“
- E – „Označené místo je skutečně evidencí vzoru.“

Pravdivostní hodnota těchto výroků pak bude odpovídat míře, s jakou jsou splněny a bude se pohybovat v intervalu $\langle 0, 1 \rangle$, přičemž čím blíže k 1, tím větší míra shody a míra jistoty evidence bude.

5.3.2 Odvození výpočtu míry evidence vzoru

K usuzování se ve fuzzy logice používá dedukční pravidlo *modus ponens*, které bychom v tomto případě mohli formulovat jako

$$(A \ \& \ (A \rightarrow E)) \Rightarrow E, \quad (5.3.2)$$

kde $\&$ je logická spojka silné konjunkce, definovaná funkcí *t-normy* (viz např. [Hájek, 98, str. 28 a 36]), a \Rightarrow je logická spojka implikace, odvozená jako reziduum *t-normy* za účelem platnosti tohoto dedukčního pravidla.

Neformálně toto pravidlo říká, že pokud platí výrok *A* a také platí, že z platnosti výroku *A* plyne platnost výroku *E*, potom platí výrok *E*. Nic se zde však již neříká o platnosti výroku *E* v případě, že není splněna premisa, sice když neplatí *A* nebo neplatí, že z platnosti *A* plyne platnost *E*.

Ve fuzzy logice je průběh pravdivostních hodnot implikace (označen *i*) definován jako maximální reziduum *t-normy*, která je funkcí průběhu pravdivostních hodnot silné konjunkce (označené *t*):

$$i(x,y) = \max \{z \in \langle 0,1 \rangle \mid t(x, z) \leq y\}, \quad (5.3.3)$$

kde *x* je pravdivostní hodnota premisy implikace a *y* pravdivostní hodnota jejího závěru. Z platnosti pravidla *modus ponens* (5.3.2) a definice implikace (5.3.3), plyne

$$\text{val}(E) \geq \text{val}(A \ \& \ (A \rightarrow E)), \quad (5.3.4)$$

kde *val()* je formální funkce přiřazující výroku jeho pravdivostní hodnotu.

Pokud shoda vzoru jednoznačně označuje dílčího kandidáta na výskyt hodnoty vlastnosti, musí platit

$$\text{val}(A \rightarrow E) = 1, \quad (5.3.5)$$

a proto také po dosazení (5.3.5) do (5.3.4) podle vlastností silné konjunkce ([Hájek, 98, str. 28]) musí platit

$$\text{val}(E) \geq \text{val}(A). \quad (5.3.6)$$

Pokud tedy vzor označí nějaké místo dokumentu s nějakou mírou shody $\text{val}(A)$, musíme toto místo prohlásit za evidenci vzoru s jistotou větší nebo rovnou míře shody vzoru.

Abychom se však vyhnuli nekonzistencím v našem inferenčním modelu, nechceme míru jistoty evidence vzoru $\text{val}(E)$ přeceňovat, a proto použijeme její minimální možnou hodnotu z (5.3.6). Dostaneme tedy

$$\text{val}(E) = \text{val}(A), \quad (5.3.7)$$

čímž se potvrzuje náš intuitivní předpoklad korespondence shody vzoru a evidence vzoru, uvedení na začátku sekce 5.3.1.

5.3.3 Zahrnutí přesnosti vzoru do výpočtu

Tento přístup k výpočtu míry jistoty evidence však nabízí první možnost obecné parametrizace vzoru a sice možnost určení přesnosti vzoru.

Nemusíme tedy přijímat pravidlo zvažovaného vzoru za stoprocentně platné, namísto toho můžeme zavést parametr p , který nazveme *přesnost* vzoru, a definujeme jej jako

$$\text{val}(A \rightarrow E) = p. \quad (5.3.8)$$

Tento parametr tedy označuje jistotu, s jakou ze shody vzoru plyne evidence tohoto vzoru. Parametr p zadává tvůrce vzoru při tvorbě extrakční ontologie.

Obdobně jako jsme (5.3.6) zjednodušili na (5.3.7), můžeme i (5.3.4) zjednodušit na

$$val(E) = val(A \& (A \rightarrow E)). \quad (5.3.9)$$

Abychom mohli tuto rovnici řešit numericky, musíme si zvolit konkrétní fuzzy logiku danou funkčním předpisem t -normy. Pro naše potřeby použijeme například Łukasiewiczovu logiku, protože v ní platí některé speciální zákony (například zákon dvojité negace), které využijeme později a které v ostatních logikách dokázat nelze.

Łukasiewiczova logika je definována Łukasiewiczovou t -normou, kterou označujeme t_L a je definována předpisem

$$t_L(x, y) = \max(0, x + y - 1), \quad (5.3.10)$$

což zároveň definuje funkční předpis silné konjunkce. Pro úplnost ještě uvedeme funkční předpis Łukasiewiczovy implikace

$$i_L(x, y) = \min(1, 1 - x + y). \quad (5.3.11)$$

Nyní již máme vše, co potřebujeme k vyjádření funkčního předpisu pro výpočet míry jistoty evidence vzoru na výstupu. Pro snadnější značení ještě zavedeme proměnnou a , jejíž hodnotou bude hodnota výstupu algoritmu vzoru a bude tedy odpovídat hodnotě $val(A)$, a proměnnou e , jejíž hodnotou bude hodnota míry jistoty evidence vzoru a bude tedy odpovídat hodnotě $val(E)$.

Výpočtem vztahu (5.3.9) pomocí předpisu (5.3.10) můžeme potom míru jistoty evidence vzoru určit za použití parametru p jako

$$e = \max(0, a + p - 1). \quad (5.3.12)$$

příklad: Budeme mít definován vzor s parametrem přesnosti $p = 0,8$. Vzor označí nějaké místo v dokumentu s mírou shody $a = 0,9$. Toto místo v dokumentu nazveme evidencí vzoru s jistotou

$$e = \max(0, 0,9 + 0,8 - 1) = 0,7.$$

Jiné místo v dokumentu označí vzor s mírou shody $a = 0,2$, evidencí vzoru jej tedy nazveme s jistotou

$$e = \max(0, 0,2 + 0,8 - 1) = 0,$$

tedy stejně, jako kdybychom žádný vzor definovaný neměli.

Nyní jsme tedy zavedli jednoduchý inferenční mechanismus pro výpočet jistoty evidencí jednoduchých vzorů. Výsledek bude záležet na výsledcích shody konkrétního vzoru a také na jeho parametru přesnosti p , který jsme tímto zavedli pro všechny vzory. Parametr p jsem tudíž zavedli i pro šablony, protože je považujeme za speciální případ vzoru.

5.3.4 Úplnost vzoru

Doposud jsme uvažovali v inferenci pouze jeden druh nejistoty, a sice že část dokumentu identifikovaná vzorem je skutečně evidencí vzoru. Usuzovací pravidlo jsme používali ve tvaru (5.3.2).

V úloze tohoto typu však existuje ještě jeden druh nejistoty a sice možnost, že předpokládanou evidenci vzoru algoritmus neoznačil. Pokusíme se odvodit usuzovací pravidlo pro tento případ.

Bezesporu můžeme napsat

$$(E \wedge \neg A) \Rightarrow E, \tag{5.3.13}$$

protože jak je dokázáno v například v [Nekvasil, 04, str. 22 – 23], je tento výrok 1-tautologií. Pomocí de Morganových zákonů jej můžeme přepsat jako

$$\neg(A \vee \neg E) \Rightarrow E. \quad (5.3.14)$$

Protože Łukasiewiczova logika obsahuje speciální spojku silné disjunkce, označenou \vee_S , pro kterou platí

$$(A \vee \neg E) \Rightarrow (A \vee_S \neg E) \Rightarrow (E \Rightarrow A), \quad (5.3.15)$$

můžeme tedy říci (ale pouze v Łukasiewiczově logice), že

$$\text{val}(E \Rightarrow A) \geq \text{val}(A \vee \neg E), \quad (5.3.16)$$

a proto také díky vlastnostem negace v této logice platí i

$$\text{val}(\neg(E \Rightarrow A)) \leq \text{val}(\neg(A \vee \neg E)). \quad (5.3.17)$$

S využitím definice implikace (5.3.11) můžeme z (5.3.17) dokázat

$$\neg(E \Rightarrow A) \Rightarrow \neg(A \vee \neg E). \quad (5.3.18)$$

Dosazením (5.3.18) do (5.3.14), může dokázat, že v Łukasiewiczově logice platí také vztah

$$\neg(E \Rightarrow A) \Rightarrow E. \quad (5.3.19)$$

Formálně pak můžeme výraz v premise tohoto výroku nahradit definitivní relací ($E \rightarrow A$). Tuto relaci nazveme *úplností* vzoru, neboť vyjadřuje míru jistoty, že předpokládanou evidenci vzoru tento vzor označí. Pro možnost nastavení úplnosti vzoru zavedeme parametr *úplnost*, označíme jej c a definujeme

$$\text{val}(E \rightarrow A) = c. \quad (5.3.20)$$

Stejně jako přesnost, bude i parametr úplnosti vzoru zadávat tvůrce extrakční ontologie.

Nově odvozené pravidlo tedy zapíšeme jako

$$\neg(E \rightarrow A) \Rightarrow E \quad (5.3.21)$$

a spolu s (5.3.2) toto pravidlo definuje míru jistoty evidence vzoru.

Pro připomenutí ještě jednou zdůrazníme, že toto odvození je možné pouze v Łukasiewiczově fuzzy logice a v jiných logikách není možné.

5.3.5 Kombinovaný výpočet s užitím přesnosti i úplnosti

Mají-li pravidla (5.3.2) a (5.3.21) platit současně, bude výrok

$$((A \& (A \rightarrow E)) \Rightarrow E) \wedge (\neg(E \rightarrow A) \Rightarrow E) \quad (5.3.22)$$

nutně 1-tautologií. V souladu s důkazem v [Hájek, 98, str. 40] můžeme tento výrok přepsat do podoby

$$((A \& (A \rightarrow E)) \vee \neg(E \rightarrow A)) \Rightarrow E, \quad (5.3.23)$$

čímž získáme opět jednoduché inferenční pravidlo. Vyjádříme-li výpočet míry jistoty evidence se stejným předpokladem jako při odvození (5.3.7) a využitím definice implikace (5.3.10), (za použití standardní max-disjunkce, Łukasiewiczovy negace a parametrů p a c) dostaneme tento předpis:

$$e = \max(\max(0, a + p - 1), 1 - c) \quad (5.3.24)$$

Protože však platí $c \in \langle 0, 1 \rangle$, můžeme jej zjednodušit jako

$$e = \max(a + p - 1, 1 - c). \quad (5.3.25)$$

Při pohledu na tento předpis zjistíme, že hodnota parametru c určuje minimální míru jistoty evidence, kterou může vzor přiřadit libovolnému místu v dokumentu. Nemá tedy příliš smysl určovat hodnoty c velmi nízké, protože v tom případě ztrácí vzor smysl. Stanovíme-li například $c = 0$, bude každé místo v dokumentu označeno za evidenci vzoru s jistotou 1 , což v praxi nemá význam, je to však logickým důsledkem tvrzení, že vzor nikdy neoznačí předpokládanou evidenci, čemuž hodnota $c = 0$ odpovídá.

Aby výsledné míry jistoty evidence daného vzoru měly smysl, musí být splněna nerovnost

$$c > 1 - p. \quad (5.3.26)$$

Pokud tato nerovnost splněna není, neovlivní sebelepší shoda vzoru výslednou míru jistoty evidence. Ze vztahu (5.3.25) totiž vyplývá

$$e \in \langle 1-c, p \rangle, \text{ pro } 1-c < p, \text{ jinak } e = 1-c. \quad (5.3.27)$$

příklad: Budeme mít stejně jako v minulém příkladu definován vzor s parametrem přesnosti $p = 0,8$ a navíc doplníme parametr úplnosti $c=0,7$. Stejně jako předtím tento vzor označí nějaké místo v dokumentu mírou shody $a = 0,9$. Toto místo v dokumentu nazveme evidencí vzoru s mírou jistoty

$$e = \max(0,9 + 0,8 - 1, 1 - 0,7) = 0,7.$$

Protože je parametr c dostatečně veliký, vidíme, že jeho zavedení zde nemá vliv. Jiné místo v dokumentu označí vzor s mírou shody $a = 0,2$, evidencí vzoru jej tedy nazveme s mírou jistotou

$$e = \max(0,2 + 0,8 - 1, 1 - 0,7) = 0,3,$$

ke slovu zde tedy přichází neúplnost tohoto vzoru a evidence je označena s minimální jistotou 0,3. V tomto případě by výsledek byl stejný, kdykoliv by algoritmus označil místo s mírou shody $a \leq 0,5$.

Parametr c má tři možná využití. Jelikož je jí stanovena minimální hodnota míry jistoty evidence vzoru, můžeme tak evidence, které by bez jeho použití byly označeny s jistotou $e = 1 - c$ nebo menší ignorovat, čímž efektivně snížíme absolutní počet evidencí v dokumentu, což bude mít kladný vliv výpočetní náročnosti této úlohy.

Druhé využití tohoto parametru uvidíme při skládání dílčích vzorů a bude jej také možno využít při automatizovaném hodnocení důvěryhodnosti indukovaného wrapperu.

Třetím výhodou použití tohoto parametru bude využití vlastností jeho vlivu při skládání dílčích vzorů, k čemuž bude řečeno více na konci oddílu 5.4.2 a 5.4.3.

Parametry p a c jsme zavedli pro všechny vzory, tedy i pro šablony. Pro zjednodušení zápisu vzorů do ontologií by mohlo být vhodné nevyžadovat jejich jmenovité uvedení u každého vzoru a v případě neuvedení za ně dosazovat výchozí hodnoty $p = 1$ a $c = 1$.

5.4 Složené vzory

5.4.1 Druhy složených vzorů

V teorii fuzzy množin lze inferenční pravidla považovat za formu zobrazení do fuzzy množiny. V tomto úhlu pohledu můžeme tedy míru jistoty označení části dokumentu za evidenci vzoru považovat za míru příslušnosti této části dokumentu do fuzzy množiny všech výskytů hodnot konkrétní vlastnosti.

Pokud budeme chtít skládat evidence více vzorů dohromady, bude to v podstatě úloha ve formě jisté množinové operace. K tomuto účelu lze v zásadě využít množinové operace dvě a sice sjednocení a průnik, které odpovídají použití logických operací \vee (disjunkce – *nebo*) potažmo \wedge (konjunkce – *a*) na hodnoty příslušnosti jednotlivých prvků těmto množinám.

K vyjádření kombinace vzorů, jak bylo uvedeno výše, budeme používat šablony. Budeme tedy muset zavést parametr šablony, kterým určíme způsob skládání dílčích evidencí vzorů šabloně vnořených, čímž dostaneme dva druhy šablon – šablony konjunktivní a šablony disjunktivní, podle toho, kterou logickou operaci pro skládání evidencí použijeme.

Samotné stanovení míry jistoty evidence podle šablony je pak triviální záležitostí. Podle terminologie, kterou jsme zavedli v minulém oddíle

stanovíme míru shody šablony (ekvivalent míry shody vzoru) prostým složením měr jistoty evidencí jednotlivých dílčích vzorů šablony pomocí příslušné logické operace. V případě konjunktivní šablony tedy určíme míru shody šablony jako

$$A \Leftrightarrow (E_1 \wedge E_2 \wedge \dots \wedge E_n), \quad (5.4.1)$$

kde E_i pro $i = 1..n$ je míra jistoty evidence i -tého dílčího vzoru šablony. Pro disjunktivní šablony vyjádříme míru jistoty evidence na vstupu obdobně

$$A \Leftrightarrow (E_1 \vee E_2 \vee \dots \vee E_n). \quad (5.4.2)$$

Z této míry shody šablony pak určíme výslednou míru jistoty evidence šablony naprosto stejným způsobem, jako v případě jednoduchých vzorů.

5.4.2 Výpočet míry shody šablony

Budeme-li chtít určit konkrétní funkční předpis pro výpočet míry shody šablony podle vztahů (5.4.1), popřípadě (5.4.2), jedině, co k tomu budeme potřebovat, bude zvolit funkční předpisy spojek konjunkce a disjunkce. Protože se stále pohybujeme v Łukasiewiczově logice, máme na výběr v podstatě ze dvou možností, buďto zvolíme Łukasiewiczovu silnou konjunkci a disjunkci ($\&$ a \vee_s) nebo obecnou slabou min-konjunkci a max-disjunkci (\wedge a \vee).

Považujeme zde za vhodnější zvolit klasické spojky min-konjunkce a max-disjunkce a máme k tomu hned dva důvody. První důvod je formální a je jím skutečnost, že použití těchto logických spojek jsme odvodili z množinových operací. Teorie fuzzy množin se neopírá o žádnou konkrétní t -normu, a proto použití obecných spojek je v tomto ohledu správnější. Druhý důvod je čistě pragmatický. Vzhledem k průběhu pravdivostních funkcí lze obecné spojky považovat za opatrnější. Použitím Łukasiewiczových silných spojek by tak hrozila možnost podceňování míry shody šablony, v případě konjunkce, nebo naopak její přeceňování v případě disjunkce. Pokud bychom však vyžadovali vzájemné posilování pravidel v disjunkci a oslabování

pravidel v konjunkci, připadalo by využití Łukasiewiczových silných spojek v úvahu.

Funkční předpis pro výpočet této míry shody šablony tedy můžeme pro konjunktivní šablony vyjádřit jako

$$a = \min (e_1, e_2, \dots, e_n) \quad (5.4.3)$$

a pro disjunktivní šablony jako

$$a = \max (e_1, e_2, \dots, e_n), \quad (5.4.4)$$

kde e_i pro $i = 1..n$ je míra jistoty evidence i -tého dílčího vzoru šablony.

Interpretace těchto typů skládání vzorů je také zřejmá. Pro konjunktivní skládání určíme shodu šablony, pokud jsou jisté všechny evidence všech dílčích vzorů šablony. Pro disjunktivní skládání určíme naopak shodu šablony v případě, že je jistá evidence alespoň jednoho z dílčích vzorů.

Zajímavé je však diskutovat význam parametrů p a c jednotlivých dílčích vzorů šablony při různých typech jejich skládání. Výpočet míry jistoty evidencí dílčích vzorů jsme stanovili jako vztah (5.3.25), zde jej doplníme pro úplnost o indexy označující pořadí vzoru:

$$e_i = \max (a_i + p_i - 1, 1 - c_i). \quad (5.4.5)$$

Pokud je parametr p_i menší než 1, snižuje se o tento rozdíl maximální hodnota, jaké může míra jistoty evidence i -tého vzoru dosáhnout. V případě disjunktivního skládání vzorů tato hodnota určuje maximální možný příspěvek tohoto vzoru k celkové míře shody šablony. Pokud tedy uvažujeme disjunktivní skládání vzorů, znamená vysoká hodnota parametru p_i , že podmínka splnění i -tého vzoru je pro celkovou shodu šablony podmínkou postačující.

Hodnota c_i zde naopak určuje nejnižší možnou hodnotu, jaké může dosáhnout celková míra jistoty evidence šablony na vstupu. Zde je třeba se

vyvarovat příliš nízkých hodnot c_i , protože jakýkoliv vzor s hodnotou p_i nižší než je nejvyšší ze všech dílčích hodnot ($1 - c_i$) by poté ztrácel smysl, protože jeho přínos by zanikl v neúplnosti jiného postačujícího vzoru.

Oproti tomu parametr c_i určuje nejmenší možnou hodnotu, jaké může míra jistoty evidence i -tého vzoru nabýt, její minimum je totiž tímto parametrem stanoveno jako $1 - c_i$. Uvažujeme-li konjunktivní skládání vzorů, určuje hodnota c_i maximální možné omezení celkové míry shody šablony tímto vzorem. V případě použití konjunktivního skládání vzorů tak vysoká hodnota c_i znamená, že podmínka splnění i -tého vzoru je pro celkovou shodu šablony podmínkou nutnou. Parametr p_i zde určuje maximální možnou hodnotu, jaké může dosáhnout celková míra shody šablony. Nízká hodnota kteréhokoliv parametru p zde tak může vyrušit přínos ostatních dílčích vzorů.

5.4.3 Příklad vyhodnocení míry evidence šablony

příklad: Budeme uvažovat šablonu složenou ze dvou vzorů. Šablona sama bude mít nastaveny hodnoty parametrů

$$p = 0,95 \quad c = 0,8$$

a její dílčí vzory

$$p_1 = 0,5 \quad c_1 = 0,95$$

$$p_2 = 0,95 \quad c_2 = 0,3$$

Nyní porovnáme, jak bude vypadat výsledná míra jistoty evidence za předpokladu, že oba dílčí vzory dané místo v dokumentu označí jako shodu zcela jistě (tedy $a_1 = a_2 = 1$), pokud zvolíme rozdílné způsoby skládání těchto dílčích vzorů.

V obou případech nejprve určíme míry jistoty evidence obou dílčích vzorů:

$$e_1 = \max(1 + 0,5 - 1; 1 - 0,95) = 0,5$$

$$e_2 = \max(1 + 0,95 - 1; 1 - 0,3) = 0,95$$

V případě konjunktní šablony bude výsledná míra shody šablony stanovena jako

$$a = \min(0,5; 0,95) = 0,5$$

a z toho pak míra jistoty evidence šablony jako

$$e = \max(0,5 + 0,95 - 1; 1 - 0,8) = 0,45.$$

Zde je tedy poměrně nízká výsledná míra jistoty evidence šablony dána nízkou přesností prvního vzoru, která tak zcela vyruší úspěch vzoru druhého.

V případě disjunktivní šablony určíme obdobně

$$a = \max(0,5; 0,95) = 0,95$$

a

$$e = \max(0,95 + 0,95 - 1; 1 - 0,8) = 0,9.$$

Vysoká hodnota výsledku je zde dána úspěchem druhého vzoru a jeho vysokou přesností, která zaručuje, že jeho evidence je do velké míry podmínkou postačující.

Nyní změníme parametry dílčích vzorů a také míry jistoty jejich evidencí na vstupu a budeme pozorovat vliv na výslednou míru jistoty evidence šablony. Stanovíme

$$p_1 = 0,95 \quad c_1 = 0,95 \quad a_1 = 0,5$$

$$p_2 = 0,7 \quad c_2 = 0,4 \quad a_2 = 0,5.$$

Opět nejprve určíme míry jistoty evidence obou dílčích vzorů:

$$e_1 = \max(0,5 + 0,95 - 1; 1 - 0,95) = 0,45$$

$$e_2 = \max(0,5 + 0,7 - 1; 1 - 0,4) = 0,6.$$

V případě užití konjunktivní šablony vypočteme

$$a = \min(0,45; 0,6) = 0,45$$

$$e = \max(0,45 + 0,95 - 1; 1 - 0,8) = 0,4.$$

Nízká přesnost druhého vzoru v tomto případě neovlivní výsledek negativně, protože má tento vzor rovněž stanovenou nízkou úplnost, díky níž není v tomto případě podmínkou nutnou (pouze však v rozsahu hodnot pod 0,6).

Za použití disjunktivní šablony vypočteme obdobně

$$a = \max(0,45; 0,6) = 0,6$$

$$e = \max(0,6 + 0,95 - 1; 1 - 0,8) = 0,55.$$

Zde je výsledná hodnota dána nízkou úplností druhého vzoru. Tato hodnota však nemá žádný prakticky použitelný význam, neboť je nejnížší možnou hodnotou, jakou bylo možno míře jistoty evidence šablony za daných podmínek přiřadit, můžeme ji tedy ignorovat.

5.4.4 Další efekt parametrů přesnosti a úplnosti

V příkladu jsme demonstrovali efekt různých hodnot parametrů p a c dílčích vzorů šablony v případě různých typů skládání.

Zajímavá je skutečnost, že v případě konjunktivního skládání vzorů se navzájem posilují parametry přesnosti, takže efektivní maximum, jakého může míra shody šablony nabýt je určeno nejnížším ze všech parametrů p_i . V případě disjunktivního skládání vzorů je tomu přesně naopak a navzájem se posilují

parametry úplnosti, přičemž nejnižší ze všech parametrů c_i určuje efektivní minimum, jakého může míra shody šablony nabýt.

Při tvorbě šablon je tedy potřeba pečlivě zvážit nastavované hodnoty parametrů dílčích vzorů, přičemž je nutné vzít v úvahu jednak typ skládání těchto vzorů a jednak celkový význam šablony.

5.5 *Návrh zápisu vzorů*

V minulé části jsme navrhli inferenční mechanismus pro vyhodnocování míry jistoty evidence vzoru a také dva způsoby skládání vzorů. Zavedli jsme také dva parametry společné všem vzorům, sice přesnost a úplnost, s označeními p a c .

Vzory budeme v ontologii zapisovat tak, jak bylo navrženo na konci čtvrté kapitoly, sice formou XML elementů ze jmenného prostoru *ot:*, který můžeme pracovním způsobem definovat jako

```
xmlns:ot="http://st.vse.cz/~XNEKM06/ontologytemplates#"
```

Typ vzoru bude dán názvem XML elementu a jeho parametry budeme uvádět formou atributů tohoto elementu, popřípadě formou vnořených elementů.

Parametry p a c však budeme vždy uvádět formou atributů příslušného elementu a pokud tyto atributy nebudou přítomny, budeme za ně dosazovat výchozí hodnoty $p = 1$ a $c = 1$.

Nyní tedy můžeme postoupit k návrhu několika konkrétních vzorů.

5.5.1 *Šablona <ot:Template>*

Prvním konkrétním vzorem, o kterém zde již do jisté míry byla řeč, je šablona. Šablonu označíme XML elementem *<ot:Template>* a jí vnořené

vzory, které bude šablona skládat, budou reprezentovány formou vnořených XML elementů elementu šablony.

V návaznosti na oddíl 5.4.1 definujeme elementu šablony atribut *ot:type*, který bude vyznačovat způsob skládání vnořených dílčích vzorů. Tento atribut bude moci nabývat dvou hodnot a sice „*disjoint*“ a „*conjoint*“, přičemž při neuvedení atributu zvolíme za výchozí hodnotu „*disjoint*“.

Zápis šablony pak bude vypadat například takto:

```
<ot:Template ot:p="0.95" ot:c="0.8" ot:type="disjoint">
  ...
</ot:Template>
```

Šablona je oproti ostatním vzorům výjimečná v tom, že ji lze jakou jedinou uvádět přímo vnořenou elementu definice datotypové vlastnosti v jazyce OWL, samozřejmě ji však lze používat všude tam, kde ostatní vzory.

Míra jistoty evidence vzoru šablona bude dána složením evidencí dílčích vzorů tak, jak bylo odvozeno v oddíle 5.4.2.

5.5.2 Řetězec <ot:String>

Tento vzor bude reprezentovat výskyt prosté textové hodnoty. Parametrem tohoto vzoru tedy bude textová hodnota a shoda tohoto vzoru nastane na každé části dokumentu, která se s touto hodnotou shoduje.

Parametr textové hodnoty budeme uvádět pomocí textového obsahu elementu. Zápis tohoto vzoru může tedy například vypadat takto:

```
<ot:String ot:p="0.7">Egypt</ot:String>
```

Tento konkrétní vzor se vyhodnotí tak, že pokud se někde v dokumentu vyskytuje řetězec „*Egypt*“, bude v tomto místě míra shody vzoru $a = 1$ a výsledná míra jistoty pak $e = 0.7$ (použitím výše uváděných vztahů pro inferenci a hodnot $p = 0.7$ a $c = 1$).

Pro tento vzor zavedeme ještě další atribut pro rozlišení malých a velkých písmen *ot:case*. Jeho možné hodnoty pak budou

- „*any*“ – text může být v libovolné kombinaci malých a velkých písmen
- „*same*“ – text musí být ve stejné kombinaci malých a velkých písmen jako parametr textové hodnoty
- „*lower*“ / „*upper*“ – text musí být uveden malými / velkými písmeny
- „*upperfirst*“ – první písmeno musí být velké, ostatní malá

5.5.3 Seznam řetězců <*ot:Stringlist*>

Seznam řetězců je vzor určený pro identifikaci známých typických hodnot uvnitř dokumentu podle jejich seznamu. Vyhodnocení tohoto vzoru bude probíhat stejně, jako kdyby se jednalo o disjunktivní šablonu s mnoha vnořenými vzory typu řetězec.

Poskytnutí seznamu známých typických hodnot tomuto vzoru pro identifikaci bude probíhat formou načtení z externího textového souboru. Pro možnost zadání umístění tohoto textového souboru zavedeme atribut *ot:source*.

Konkrétní použití tohoto vzoru může být například takovéto:

```
<ot:Stringlist ot:source="c:\temp\zeme.txt" ot:c="0.62" />
```

Můžeme také povolit zahrnutí dalších vnořených vzorů uvnitř elementu <*ot:Stringlist*>, přičemž toto budeme interpretovat jako prosté přidání těchto vzorů zkonstruované disjunktivní šabloně. Vzor typu seznam řetězců bude tedy možné použít i namísto disjunktivní šablony tam, kde k tomu důvody budou čistě estetické, například u výčtu hodnot. Jiné použití tohoto vzoru tedy může být toto:

```
<ot: Stringlist>
  <ot: String>Egypt</ot: String>
  <ot: String>Itálie</ot: String>
  <ot: String>Řecko</ot: String>
</ot: Stringlist>
```

Evidence v dokumentu se však v tomto případě určí stejně, jako bychom namísto *<ot:Stringlist>* použili *<ot:Template ot:type="disjoint">*.

5.5.4 Sřetěžení *<ot:Concatenation>*

Vzor typu sřetěžení bude skládat evidence jemu vnořených vzorů konjunktivním způsobem, tentokrát však nikoliv evidence ve stejném místě dokumentu, nýbrž po sobě v textu dokumentu bezprostředně následující. Použitím tohoto vzoru bude možné definovat vzory vzniklé složením jiných vzorů za sebe, tedy jejich sřetěžením.

Vypočet míry shody bude probíhat tak, že pro všechny evidence prvního vzoru se vyhledají evidence druhého vzoru bezprostředně po nich následujících, po druhém vzoru se vyhledají bezprostředně následující evidence vzoru třetího a obdobně pak dalších. Míry jistoty jednotlivých evidencí se pak složí, jako kdyby se jednalo o konjunktivní skládání evidencí, s tím rozdílem, že v tomto případě se nejedná o evidence ve stejném místě dokumentu.

Příklad použití tohoto vzoru může být takovýto:

```
<ot: Concatenation>
  <ot: Template ot:type="disjoint">
    <ot: String>sportovní</ot: String>
    <ot: String>poznávací</ot: String>
    <ot: String>jazykový</ot: String>
  </ot: Template>
  <ot: Template ot:type="disjoint">
    <ot: String>pobyt</ot: String>
    <ot: String>zájezd</ot: String>
  </ot: Template>
</ot: Concatenation>
```

Jako shodu tento již poměrně komplikovaný vzor označí všechny výskyty kombinací textu jako „jazykový zájezd“, „poznávací zájezd“ nebo „sportovní pobyt“. Míry jistoty označení těchto jednotlivých kombinací by bylo samozřejmě možné upřesnit vhodným použitím parametrů p a c na všech úrovních vzorů.

5.5.5 *Kontext* <ot:Context>

Kontextem se v tomto případě myslí fyzická blízkost evidence vnořeného vzoru v těle dokumentu. Vzorek typu kontext tedy bude mít právě jeden vnořený vzor.

Určujeme-li fyzickou blízkost, musíme určit, na kterou stranu od referované evidence vzdálenost měříme. Zavedeme tedy parametr tohoto vzoru *ot:side*, který bude určovat stranu, na které se má kontext od výsledného místa shody vyskytovat (s možností nabytí hodnot „*left*“ nebo „*right*“).

Druhým parametrem omezíme maximální vzdálenost, měřenou ve slovech, do jaké se evidence vnořeného vzoru bere za relevantní. Tento parametr budeme uvádět v atributu *ot:maxdistance*.

Za shodu vzoru typu kontext, tedy označíme takové místo v dokumentu, jemuž předchází (nebo následuje, to je dáno parametrem *ot:side*) evidence vzoru kontextovému vzoru vnořeného, a to maximálně do vzdálenosti, dané hodnotou parametru *ot:maxdistance*.

Z výše uvedeného je tedy jasné, že pokud zadáme hodnotu parametru *ot:maxdistance* větší než 1, bude pro každou evidenci vnořeného vzoru kontextový vzor generovat více míst shody. Z toho plyne, že kontextový vzor je vhodnější použít jako podmínku nutnou v konjunktivní šabloně s jinými vzory, než jako podmínku postačující.

Jako příklad můžeme použít vzor pro pravidlo, že výskyt ceny a textu dokumentu bývají často uvozeny zleva textem „cena“ (zde ještě ve variantě s dvojtečkou):

```
<ot:Context ot:side="left" ot:maxdistance="1" ot:c="0.5">
  <ot:Template>
    <ot:String ot:case="any">cena</ot:string>
    <ot:String ot:case="any">cena: </ot:string>
  </ot:Template>
</ot:Context>
```

Zde jen připomeneme, že při neuvedení atributu *ot:type* u elementu vzoru typu šablona volíme jeho výchozí hodnotu a sice „*disjoint*“.

5.5.6 Číslo <ot:Number>

Název tohoto vzoru nejspíše hovoří za vše. Tento vzor v dokumentu označí jako shodu všechny části textu, které jsou číselného typu. Pro tento vzor zavedeme parametry:

- *ot:type* – určuje číselný typ, tedy zda se jedná o číslo celé nebo reálné s příslušnými hodnotami „*integer*“ a „*real*“
- *ot:min* – minimální hodnota čísla. Stanovením *ot:min* = “1” a *ot:type* = “*integer*“ lze například určit požadavek na číslo přirozené.
- *ot:max* – maximální hodnota čísla
- *ot:mod* – dělitel čísla. Za shodu budou označeny pouze takové výskyty čísel, po jejichž vydělení hodnotou tohoto parametru bude zbytek dělení nulový. Stanovením *ot:type* = “*real*“ a *ot:mod* = “0.01” lze tak například určit požadavek na číslo uvedené maximálně na dvě desetinná místa..

Parametrů tohoto typu vzoru by jistě mohlo být více, tyto však pro začátek plně postačují. Využití vzoru typu číslo je jistě zřejmé.

5.5.7 Rozdělení <ot:Distribution>

Vzor tohoto typu popisuje číselnou náhodnou veličinu se specifickým rozdělením. Abychom vůbec mohli nějakou hodnotu označit za shodu vzoru rozdělení, musí se jednat o veličinu číselného typu, který můžeme blíže upřesnit souběžným použitím vzoru typu číslo.

Pro určení konkrétního předpisu rozdělení náhodné veličiny použijeme parametr *ot:type*, přičemž zatím budeme uvažovat pouze hodnotu „*gauss*“ pro Gaussovo normální rozdělení. Dále zavedeme parametry *ot:mean* a *ot:variance* pro stanovení parametrů tohoto rozdělení.

Míru shody vzoru na vstupu určíme jako míru, s jakou splňuje zvažovaná veličina parametry daného rozdělení. Pokud tedy označíme hodnotu zvažované číselné veličiny (potenciální evidence) jako x a parametry rozdělení standardně μ a σ , můžeme tuto míru shody vyjádřit jako

$$a = 1 - P(\mu - \text{abs}(x - \mu) < X < \mu + \text{abs}(x - \mu)),$$

což lze přepsat pro normální rozdělení jako

$$a = 2 \cdot \Phi_{\mu, \sigma}(\min(x, \mu - (x - \mu))),$$

kde $\Phi_{\mu, \sigma}(x)$ je distribuční funkce rozdělení.

Pokud bychom chtěli zavést jiná rozdělení, bude postup identický.

Zapojení vzoru typu distribuce do již celkem komplexní šablony pro cenu by mohlo být následující:

```
<ot: Template ot: type="distribution" ot: c="0.9">
  <ot: Concatenation>
    <ot: Distribution ot: type="gauss" ot: mean="10900"
      ot: variance="9200000"/>
    <ot: Stringlist>
      <ot: String ot: case="any">kc</ot: string>
      <ot: String ot: case="any">kč</ot: string>
      <ot: String ot: case="same">,<-</ot: string>
    </ot: Stringlist>
```

```

</ot: Concatenation>
<ot: Context ot: side="left" ot: maxdistance="2" ot: p="0.6">
  <ot: Template>
    <ot: String ot: case="any">cena</ot: string>
    <ot: String ot: case="any">cena: </ot: string>
  </ot: Template>
</ot: Context>
</ot: Template>

```

V kontextu této šablony udává vzor typu rozdělení typické hodnoty, v jakých se tyto ceny pohybují.

5.5.8 Regulární výraz <ot:Regex>

Vzor typu regulární výraz použijeme v případě, že máme potřebu použít pokročilejší formu definice řetězce. Samotný regulární výraz bude vzoru zadán pomocí textové hodnoty elementu vzoru. Shoda tohoto vzoru pak bude označena, pokud daná část dokumentu vyhovuje zadanému regulárnímu výrazu, například v tom smyslu, jak je doporučeno vyhodnocování regulárních výrazů ve standardech XML schémat konsorcia W3.

5.5.9 Další vzory

V principu jsme naznačili systém, jakým vytváříme jednotlivé druhy vzorů. Obdobným postupem by šlo samozřejmě vytvořit nespočet dalších vzorů pro potřeby různých datotypových atributů. Rozsah vzorů může být skutečně rozmanitý od vzoru pro rozeznávání času, přes identifikaci pojmenovaných entit až po vzory vyhodnocující kontext ve formátu dokumentu (například barvu pozadí textu).

Při návrhu nového typu vzoru je však vždy třeba mít na paměti možnosti jeho kombinace s ostatními vzory. Nezbytnou součástí je pak specifikace způsobu určení míry shody tohoto vzoru.

Nám bude prozatím stačit několik již výše navržených vzorů, abychom mohli tuto metodiku ověřit a zhodnotit její využitelnost při indukci wrapperu, čemuž se budeme věnovat v následující kapitole.

Kapitola 6

IMPLEMENTACE A OVĚŘENÍ NAVRŽENÝCH METOD

6.1 *Zaměření kapitoly*

Jak vyplývá z názvu, budeme se v této kapitole věnovat ověření navržených metodik na jednoduchém nicméně skutečném příkladě. Tento úkol si rozdělíme do dvou částí.

Nejprve zformulujeme ontologii pro oblast cestovních zájezdů včetně šablon pro identifikaci datotypových hodnot a posléze ji k identifikaci kandidátů na výskyt těchto hodnot, tedy k automatické anotaci dokumentu, na stránkách jedné velké cestovní kanceláře využijeme.

Poté navrhne jednoduchý indukční algoritmus pro tvorbu XPath wrapperu pro data v tabulární struktuře na webové stránce ve formátu HTML. Tento indukční algoritmus také ihned vyzkoušíme pomocí získaných kandidátů na výskyt jednotlivých datotypových hodnot.²

6.2 *Anotace dokumentu*

6.2.1 *Ontologie*

I v této kapitole budeme pokračovat v příkladu ontologie cestovních zájezdů. Pro připomenutí v této ontologii definujeme třídu „Zajezd“, která má dva datotypové atributy, cenu a cílovou zemi.

² Reálná implementace zatím zahrnuje načítání ontologií včetně vzorů, přičemž podporuje vzory navržené v oddíle 5.5., a identifikaci evidencí s pomocí těchto vzorů. Výsledky dosavadních experimentů odpovídají předpokládaným výsledkům a tedy závisí na vhodné formulaci vzorů. Algoritmy pro tvorbu XPath wrapperu jsou ve stádiu návrhu.

Tuto ontologii jsme zde doplnili o šablony pro možnost jejich automatické identifikace v rámci dokumentu. Výsledný zápis ontologie bude následující.

```
<?xml version="1.0"?>

<rdf: RDF
  xml ns: rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xml ns: xsd="http://www.w3.org/2001/XMLSchema#"
  xml ns: rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml ns: owl="http://www.w3.org/2002/07/owl#"
  xml ns: ot="http://st.vse.cz/~XNEKM06/ontologytemplates#">

  <owl: Ontology rdf: about="" />

  <owl: Class rdf: ID="Zajezd" />

  <owl: DatatypeProperty rdf: ID="cena">
    <rdfs: domain rdf: resource="#Zajezd" />

    <ot: Template ot: type="disjoint" ot: c="0.9">
      <ot: Concatenation>
        <ot: Distribution ot: type="gauss" ot: mean="10900"
          ot: variance="9200000" />
        <ot: Stringlist>
          <ot: String ot: case="any">kc</ot: string>
          <ot: String ot: case="any">kč</ot: string>
          <ot: String ot: case="same">,<-</ot: string>
        </ot: Stringlist>
      </ot: Concatenation>
      <ot: Context ot: side="left" ot: maxdistance="2" ot: p="0.6">
        <ot: Template>
          <ot: String ot: case="any">cena</ot: string>
          <ot: String ot: case="any">cena:</ot: string>
        </ot: Template>
      </ot: Context>
    </ot: Template>

  </owl: DatatypeProperty>

  <owl: DatatypeProperty rdf: ID="cilovazeme">
    <rdfs: domain rdf: resource="#Zajezd" />
    <ot: Template ot: c="0.8" ot: p="0.6">
      <ot: Stringlist ot: source="c:\temp\zeme.txt">
    </ot: Template>
  </owl: DatatypeProperty>
</rdf: RDF>
```

Znalostní model a pravidla tedy máme tímto určená. Nyní se podíváme na způsob, jakým je využijeme k automatické anotaci dokumentu.

6.2.2 Automatická anotace dokumentu

Anotací dokumentu, jak jsme již několikrát dříve zmínili, myslíme označení vzorových hodnot vlastností třídy určené k extrakci. Zde se podíváme na možnost, jak můžeme tento proces automatizovat.

Máme tedy extrakční ontologii včetně šablon datotypových hodnot extrahované třídy a chceme vyznačit instance této třídy v dokumentu ve formátu HTML. Pro příklad použijeme skutečný dokument, a sice nabídku last-

The screenshot shows the website of Čedok, a travel agency. The main content area is titled 'Zahraníční zájezdy Čedoku - Last minute'. It features a search bar with options to sort by date, country, departure point, or price. Below this, there are several travel packages listed in a table format, categorized by destination: Egypt, Chorvatsko - střední Dalmácie, Chorvatsko/Bosna-Hercegovina, and Chorvatsko/Černá Hora. Each package includes details such as duration, departure point, hotel, and price, along with an 'OBJEDNAT' button.

Destination	Duration	Departure	Hotel	Price (Kč)	Action
Egypt	3. 5.	10 dnů	OVA pobyt - Hotel Coral Beach 4	10 990 Kč	OBJEDNAT
	3. 5.	10 dnů	OVA pobyt - Hotel Sea Star 4	10 490 Kč	OBJEDNAT
	3. 5.	10 dnů	OVA pobyt - Hotel Triton Empire 3	7 490 Kč	OBJEDNAT
	3. 5.	10 dnů	OVA pobyt - Hotel Aqua Blu 4	9 990 Kč	OBJEDNAT
	3. 5.	10 dnů	OVA pobyt - Hotel Sea Gull 4	13 990 Kč	OBJEDNAT
Chorvatsko - střední Dalmácie	3. 6.	8 dnů	PRG pobyt - Pavilony Mediteran 2	8 990 Kč	OBJEDNAT
	3. 6.	8 dnů	PRG pobyt - Depandance Laguna 2	8 990 Kč	OBJEDNAT
	3. 6.	8 dnů	PRG pobyt - Pavilony Riviera I 2	9 490 Kč	OBJEDNAT
	3. 6.	8 dnů	PRG pobyt - Pavilony Zora 2	9 490 Kč	OBJEDNAT
	3. 6.	8 dnů	PRG pobyt - Pavilony Riviera II 2	8 990 Kč	OBJEDNAT
	3. 6.	8 dnů	PRG pobyt - Pension Vila Luka 3	9 990 Kč	OBJEDNAT
	3. 6.	8 dnů	PRG pozn - Příroda a památky Dalmácie letadlem s výletem do Bosny Hercegoviny	11 990 Kč	OBJEDNAT
Chorvatsko/Bosna-Hercegovina	3. 6.	8 dnů	PRG pozn - Příroda a památky Dalmácie letadlem s výletem do Bosny Hercegoviny	11 990 Kč	OBJEDNAT
Chorvatsko/Černá Hora	26. 5.	9 dnů	pozn - Příroda a památky Dalmácie a	2	

Obrázek 6.1 - náhled dokumentu pro extrakci

minute zájezdů jedné velké cestovní kanceláře, náhled tohoto dokumentu je na obrázku 6.1, zdrojový kód neuvádíme, protože má velikost 57 KB.

První krok automatické anotace dokumentu bude vyhodnocení evidencí šablon jednotlivých datotypových vlastností ve smyslu podle definice z oddílu 5.4. Inferenci na evidence jednoduchých vzorů a skládání vzorů jsme probrali v páté kapitole. Záměrně jsme však opomíjeli jeden fakt a sice neurčitost pojmu „místo v dokumentu“, na který jsme neustále odkazovali.

Abychom mohli provádět označování evidencí jednotlivých vzorů, rozdělíme si nejprve dokument po slovech a to tak, že vynecháme formátovací značky (HTML tagy) a budeme brát v úvahu pouze prostý text dokumentu. Hovoříme zde o slovech, v praxi však půjde o libovolné shluky znaků, oddělené mezerami. Každé takto získané slovo si ovšem bude „pamatovat“ svoje původní umístění v dokumentu, tedy adresu elementu v němž bylo uvedeno a svoji pozici v něm. Tato slova uložíme v uspořádaném jednorozměrném poli ve stejném pořadí, v jakém se vyskytovala v dokumentu.

V této datové struktuře již můžeme ve většině případů označit evidence jednotlivých vzorů, přičemž evidencí vzoru myslíme trojici hodnot: počátek evidence, délka evidence, míra jistoty evidence. Počátek a délka evidence označují index a délku v rámci pole slov.

Pouze ty evidence vzorů, které mají stejný počátek a délku, mohou být předmětem skládání. Protože však při skládání evidencí dílčích vzorů nebereme v úvahu typ vzoru, který je generoval, nastane problém u speciálních vzorů, jako je vzor typu kontext, u kterých nemůžeme určit délku evidence, ale pouze její počátek. Délku takovýchto speciálních evidencí označíme nějakou speciální hodnotou (např. *-1*) a budeme tyto evidence skládat s jinými evidencemi, bude-li splněna alespoň podmínka shodného počátku, přičemž délku výsledné evidence určíme z délky evidencí, se kterými jsme tyto speciální evidence skládali.

V tomto okamžiku již tedy nic nebrání označení evidencí každé šablony na námi vybrané vzorové stránce. Umístění evidencí převedeme do formy absolutní indexované cesty XPath.

Pro šablonu vlastnosti „cilovazeme“ dostaneme množinu dvaceti šesti evidencí. Uvedeme reprezentativní tři vzorky, ve formě míra jistoty evidence – hodnota evidence – cesta v dokumentu

- 0,6 – Egypt -
/html/body/form/center/div/div/div[3]/div[2]/div/table[1]/tbody/tr/td[6]/table/tbody/tr/td[1]
- 0,6 – Chorvatsko -
/html/body/form/center/div/div/div[3]/div[2]/div/table[8]/tbody/tr/td[6]/table/tbody/tr/td[1]
- 0,6 – Itálie -
/html/body/form/center/div/div/div[3]/div[2]/div/table[17]/tbody/tr/td[6]/table/tbody/tr/td[1]

Zde máme pouze evidence s mírou jistoty 0,6, protože jsme k jejich označení použili seznam řetězců v šabloně s přesností 0,6, čímž jsme omezili maximální možnou hodnotu míry jistoty.

Zde stojí za povšimnutí fakt, že všechny XPath cesty jsou vesměs shodné až na index jedné tabulky. To ukazuje na dobrou strukturovanost dokumentu.

Pro šablonu ceny dostaneme také množinu dvaceti evidencí. Tentokrát uvedeme tři evidence s největší jistotou.

- 0,976 – 10 990 Kč -
/html/body/form/center/div/div/div[3]/div[2]/div/table[1]/tbody/tr/td[7]/b

- 0,892 – 10 490 Kč -
/html/body/form/center/div/div/div[3]/div[2]/div/table[2]/tbody/
tr/td[7]/b
- 0,764 – 9 990 Kč -
/html/body/form/center/div/div/div[3]/div[2]/div/table[11]/tbody/
tr/td[7]/b

Situace s indexy v cestě je zde obdobná jako u minulé množiny.

Takovéto množiny tedy budeme považovat za výstup automatické anotace dokumentu. Ke každé množině ještě přidáme parametry p a c šablony, která ji generovala.

6.3 *Indukce wrapperu*

6.3.1 *Očištění evidencí*

Prvním krokem našeho jednoduchého algoritmu indukce wrapperu bude očištění množin evidencí o zavádějící prvky. K tomu použijeme parametr p u příslušné množiny, protože jeho významem je přesnost šablony generující množinu.

Pokud tedy v případě první množiny byl parametr $p = 0.6$, můžeme si dovolit malou formální nepřesnost v interpretaci tohoto čísla, když prohlásíme, že tedy průměrně 60% všech označených hodnot šablonou je skutečně výskytem dané vlastnosti. Pokud toto vysvětlení připustíme, plyne z toho, že tedy až 40% všech poskytnutých evidencí může být falešných.

Protože spoléháme na tabulární strukturu dat v dokumentu, můžeme se pokusit rozdělit jednotlivé evidence do několika segmentů, podle počátku XPath cesty. Zjistíme, že až do tabulky, u níž sledujeme proměnný index jsou

```

Segmentuj Evidenci (pole evidencí vlastnosti E)
S = pole segmentů
pro každou evidenci e1 ∈ E
    pro každou evidenci e2 ∈ E, e2 ≠ e1
        pokud se XPath e1 a e2 liší právě jedním indexem
            pokud neexistuje segment s ∈ S pro cestu e1, e2
                vytvoř segment s pro cestu k e1, e2
                přiřaď e1 a e2 do s
                přiřaď s do S
            jinak přiřaď e1 a e2 do s, pokud již nejsou zahrnuty
vrať pole S

```

Obrázek 6.2 – pseudokód pro segmentaci evidencí šablony

cesty všech dvaceti šesti evidencí shodné. Můžeme z toho usoudit, že různé instance jsou tedy v různých tabulkách této úrovně cesty.

Postup při segmentaci evidencí jedné vlastnosti můžeme vyjádřit pomocí pseudokódu na obrázku 6.2.

Vidíme, že jednotlivé segmenty evidencí se mohou i překrývat. V takovém případě může být jedna evidence zahrnuta ve více segmentech.

Pokusíme-li se v našem příkladě segmentovat evidence podle zbytku cesty, objevíme dvě skupiny cest. Jedna obsahuje dvacet členů, druhá šest. Menšinová skupina tak tvoří $6/26 = 23\%$ všech evidencí, a většinová 77% . Evidence z menšinového segmentu jsou tedy pravděpodobně zavádějící a můžeme je vyřadit.

Toto pravidlo můžeme zobecnit tak, že seřadíme-li segmenty podle jejich podílu na celku, můžeme je postupně vyřazovat jako zavádějící, dokud celkový poměr vyřazených nedosáhne hodnoty $(1 - p)$ parametru generující šablony. Poslední segment, jehož vyřazením bychom tuto hodnotu přesáhli nevyřazujeme (mohl by totiž být zároveň prvním nebo posledním ze všech). Alternativně můžeme segmenty vyřazovat podle poměru součtu měr jistoty jejich evidencí vůči celkovému součtu měr jistoty všech evidencí dané šablony, což zřejmě bude spolehlivější způsob, protože bude brát v úvahu přesnost určení konkrétních evidencí.

Očisti Evidenci (pole evidencí vlastnosti E, přesnost šablony p)

```

S = Segmentuj Evidenci(E)
P = vypočti součet měr jistoty evidencí e ∈ E
pro každý segment s ∈ S
    s.P = součet měr jistoty evidencí e ∈ s
procházej s ∈ S podle velikost s.P od nejmenšího
dokud (s.P + X) / P ≤ p
    X = X + s.P
    odstraň s z S
    pokračuj na další segment
vrať pole S

```

Obrázek 6.3 - Pseudokód pro očištění evidencí

Po očištění nám tak zbude v obou množinách dvacet prvků.

Pomocí pseudokódu můžeme tento obecný postup očištění o zavádějící segmenty zapsat způsobem, který je na obrázku 6.3.

6.3.2 Zobecnění XPath cesty

Druhým krokem na naší cestě k indukci wrapperu bude zobecnění XPath cesty, nejprve pro každou datotypovou vlastnost zvlášť a poté pro instanci extrahované třídy jako celek.

Pokud jsou data uložena v tabulární struktuře, jsou zpravidla relevantní části textu „obaleny“ neměnnými elementy, přičemž se tato struktura opakuje. V XPath výrazu se tento jev projeví pouze jako změna indexu jednoho elementu a můžeme ve většině případů spoléhat na to, že jednotlivé instance jsou odděleny právě tímto elementem.

Při segmentaci za účelem očištění množin evidencí jsme již na proměnný index elementu narazili. V našem přístupu nám nezbyvá, než se spolehnout, že u tabulární struktury dat bude v XPath cestě každého segmentu právě jeden tento proměnný index. Pokud jich bude více, nebudeme schopni úspěšně pokračovat ve zobecňování cesty na tomto segmentu evidencí.

V rámci jednoho segmentu můžeme tedy cestu zobecnit jednoduše tak, že proměnný index z XPath výrazu vypustíme (prozatím), čímž po vyhodnocení výsledného XPath výrazu dostaneme množinu elementů, které by v ideálním případě měly všechny obsahovat hodnotu datotypové vlastnosti.

6.3.3 Hodnocení chyby zobecnění cesty

Zobecněním XPath výrazu segmentu evidencí generovaných danou šablonou jsme získali množinu elementů této zobecněné cesty vyhovujících. Vzpomeneme-li si na parametr úplnosti šablony c , byl definován jako jistota, že tato šablona bude identifikovat všechny výskyty příslušné vlastnosti. Dopustíme-li se opět drobné formální nepřesnosti, můžeme tento parametr považovat za podíl hodnot extrahované vlastnosti, které šablona skutečně označí vůči všem hodnotám této vlastnosti.

Víme tedy, že pokud je například parametr $c = 0,8$, měla by šablona označit přibližně 80% všech výskytů hodnot dané vlastnosti. Zobecněním XPath cesty jsme dostali množinu předpokládaných hodnot extrahované vlastnosti, přičemž by poměr počtu evidencí v daném segmentu před zobecněním cesty vůči počtu prvků této množiny měl činit přibližně c , tedy 80%. Pokud je tento poměr vyšší, nedošlo k nepředpokládanému zobecnění zahrnutí nových hodnot. Pokud je ale tento poměr nižší, znamená to, že zobecněním XPath cesty jsme se dopustili větší nepřesnosti, než se při tvorbě šablony předpokládalo. Chybu zobecnění cesty můžeme tedy vyjádřit jako absolutní odchylku tohoto poměru od parametru c šablony v rámci jednoho segmentu.

V našem příkladu jsme se žádné chyby nedopustili, protože množina, která je výsledkem vyhodnocení zobecněného XPath výrazu, obsahuje stejně prvků jako segment, z něž jsme XPath výraz odvodili.

6.3.4 *Korespondence hodnot různých vlastností*

Nyní máme tedy pro každý segment každé množiny evidencí generovaných šablonami hodnot datotypových vlastností určenou zobecněnou cestu k předpokládaným hodnotám dané vlastnosti. Poslední co tedy zbývá, je přiřadit k sobě odpovídající hodnoty datotypových vlastností a určit tak konkrétní instance extrahované třídy.

Obecně můžeme předpokládat, že máme pro každou šablonu různý počet segmentů evidencí, kde každý segment má různý počet prvků. Můžeme si ale dovolit předpokládat, že v tabulárně strukturovaném dokumentu, kde jsou instance extrahované třídy zapsané formou n-tic hodnot, by měl být počet segmentů pro každou šablonu přibližně stejný a počet prvků v odpovídajících segmentech hodnot by měl také přibližně odpovídat.

Prioritně tedy budeme segmenty přiřazovat na základě počtu jejich prvků. Pokud existuje od každé šablony právě jeden segment se stejným počtem prvků jako je v některém segmentu jiné šablony, můžeme říci, že tyto dva segmenty si odpovídají.

Pokud nejsme schopni přiřadit všechny segmenty na základě počtu prvků, v případě že existuje více řešení, přiřadíme segmenty na základě podobnosti zobecněné XPath cesty. Pokud neexistuje žádné řešení přiřazení segmentů na základě počtu jejich prvků, nejsme bohužel schopni jednoznačně přiřadit k sobě navzájem si odpovídající hodnoty vlastností jedné instance.

Výsledkem tohoto vzájemného přiřazení segmentů by měla být množina n-tic zobecněných XPath cest, kde n je počet datotypových vlastností extrahované třídy. Dosazením konkrétní hodnoty za původní hodnotu proměnného indexu ve zobecněných XPath výrazech dostaneme n-tici hodnot, která odpovídá n-tici vlastnostem extrahované instance. Extrakce pak bude probíhat opakováním tohoto kroku pro všechny možné hodnoty proměnných indexů a pro všechny skupiny segmentů.

Při řadě segmentů (pole segmentů evidencí šablony $PS = \{S_1 \dots S_n\}$)
 $V =$ pole skupin segmentů
 pro všechny kombinace segmentů $s_1 \in S_1, s_2 \in S_2 \dots s_n \in S_n$
 pokud poč. prvků $s_1 =$ poč. prvků $s_2 = \dots =$ poč. prvků s_n
 vytvoř skupinu segmentů v pro $s_1, s_2 \dots s_n$
 přiřaď v do V
 pro všechny skupiny segmentů $v_1 \in V$
 pro všechny skupiny segmentů $v_2 \in V$
 pokud počet prvků segmentů $v_1 =$ počet prvků segmentů v_2
 odstraň z V skupinu segmentů s většími rozdíly mezi
 zobecněnými cestami jednotlivých segmentů
 vrať V

Obrázek 6.4 - Pseudokód pro přiřazení segmentů

Konkrétní algoritmus pro přiřazování segmentů je naznačen v pseudokódu na obrázku 6.4. Tento přístup je však pouze jednoduchým návrhem a má řadu omezení. Kromě toho, že jím lze extrahovat pouze datotypové vlastnosti s kardinalitou 1 (tedy jedna hodnota na jednu instanci), je také dosti omezený, co se týče tolerance vůči chybám v pravidelnosti struktury dokumentu. Spoléhá se totiž na pravidelnou tabulární strukturu bez sebemenších odchylek. Naopak vůči nepravidelnostem v samotných hodnotách vlastností je tento přístup poměrně odolný.

Konečně celkovou chybu zobecnění cest při indukci wrapperu můžeme vyjádřit jako průměr chyb zobecnění cest v jednotlivých segmentech vážený jejich velikostmi. Tuto chybu můžeme také určit pro každou skupinu segmentů zvlášť a získat tak předpokládanou chybu v extrakci instancí.

Kapitola 7

ZÁVĚRY

V práci se podařilo navrhnout rozšíření jazyka OWL pro zápis šablon hodnot datotypových vlastností extrahované třídy. K tomuto rozšíření jsme navrhli také příslušný inferenční mechanismus na základě Łukasiewiczovy fuzzy logiky, díky kterému je možné usuzovat na míru označení konkrétních míst v dokumentu danou šablonou. Možným tématem budoucí práce v této oblasti by mohlo být podrobné prozkoumání různých inferenčních modelů a jejich srovnání ve vztahu k automatizované anotaci dokumentu.

Navržený způsob zápisu šablon umožňuje hierarchické skládání dílčích vzorů pro hodnoty vlastností a je otevřený možnosti návrhu dalších vzorů. Způsob vyhodnocování šablon je tedy nezávislý na struktuře a formátu dokumentu.

Tento mechanismus, označovaný jako automatizovaná anotace dokumentu, je navržen s ohledem k použití s různými již existujícími metodikami automatické indukce wrapperu. Možnosti doplnění stávajících indukčních mechanismů jsou diskutovány ve třetí kapitole a vesměs spočívají v poskytování příkladů chování wrapperu. Například v případě Kushmerickova modelu indukce můžeme použít množinu automaticky indukovaných příkladů k určení maximální možné chyby wrapperu na požadované hladině pravděpodobnosti.

Jako nejuniverzálnější v přístupu k extrakci se nám jeví metodologie používaná extrakčním jazykem Elog prostřednictvím nástroje Lixto. Možnosti extrakčních pravidel jazyka Elog jsou ohromné. Dle našeho názoru by mohlo být možné některá tato extrakční pravidla odvozovat ze speciálních vlastností ontologie, jako jsou kardinalitní omezení a objektové vlastnosti tříd. V tomto bodě vidíme další možný námět pro budoucí řešení v této oblasti.

Nevýhodou naší automatické anotace v tomto ohledu je skutečnost, že na rozdíl od anotujícího člověka nedokážeme automaticky vyznačit v dokumentu negativní příklady chování wrapperu, které většina moderních systémů indukce wrapperu dokáže využít.

V práci jsme také navrhli jednoduchý způsob indukce wrapperu, založený na zobecnování cesty XPath. Omezením tohoto modelu je, že se opírá o tabulární strukturu dokumentu a možnost konstrukce DOM stromu z jeho elementů, dokument tedy musí být ve formátu HTML nebo ještě lépe XHTML. Tento způsob využívá námi navržené automatické anotace dokumentu pomocí rozšířené ontologie a je použitelný i na skutečné dokumenty, které mají data uvedena v tabulce. Výhodou tohoto přístupu je, že je zcela automatizovaný.

Při využití automatické anotace v některém z konvenčních principů indukce wrapperu by se zřejmě jednalo jen o předběžné vyznačení podezřelých hodnot v dokumentu a finálním sestavitelem anotace dokumentu by nejspíše zůstal člověk.

Zajímavým námětem pro budoucí práci by také mohlo navržení způsobu automatického učení ontologií, nebo alespoň šablon jednotlivých datotypových hodnot a jejich parametrů, aby jejich tvůrcem člověk zůstat nemusel.

TERMINOLOGICKÝ SLOVNÍK

Termíny přejaté z terminologického slovníku katedry informačních technologií

DOM Aplikační programové rozhraní (API) pro práci s XML dokumentem, prochází jeho stromovou strukturu.

HTML Jazyk, který vychází z normy ISO8879 (Standard Generalized Markup Language). HTML vznikl v souvislosti s rozvojem služby -> WWW. Je založen na principu označování (mark-up) částí textu pomocí předem známé množiny značek popsané v Document Type Definition (DTD).

Značky specifikují význam textu (např. určitá část textu je nadpisem) nebo umožňují vkládat do textu odkazy na jiné objekty. Takové odkazované objekty mohou být buďto následně součástí zobrazovaného textu (obrázky) nebo se jedná o hypertextové odkazy. Hypertextovým odkazem autor dokumentu vytváří vazbu na soubor obsahující informace, které rozšiřují informační hodnotu textu takto označeného. Vazba je vytvořena pomocí standardizovaného zápisu adresy objektu, který se nazývá URI (Uniform Resource Identifier). Např. adresa http://www.cssi.cz/trh_main.asp specifikuje, že pro přístup k objektu bude použit protokol http (tedy služba WWW), že objekt je umístěn na uzlu se jménem www.cssi.cz a že objekt nese jméno [trh_main.asp](http://www.cssi.cz/trh_main.asp). Pro pokročilé formátování vzhledu dokumentu se užívá kaskádových stylů (Cascading StyleSheet Language). Vývoj HTML byl ukončen verzí 4. Pokračování je založeno na jazyce -> XML, kde je definována jako jedna z množin značek (DTD) i množina značek odpovídající poslední verzi HTML 4. Tento jazyk se nazývá XHTML (Extensible HTML).

HTTP Protokol pro přenos dokumentů v Internetu (používá se nejčastěji na WWW).

metadata Metadata jsou informace o datech v produkčních databázích tj. názvy tabulek, názvy atributů, datové typy, primární klíče, vazby, komentáře, atd.

web (WWW) Název populární služby -> Internetu, která uživatelům prostřednictvím -> WWW prohlížeče zpřístupňuje informace službou spravované. Informace jsou uloženy na serverech a mohou mít různé formáty (text, binární data – grafika, zvuk, ...). O zpřístupnění se stará WWW prohlížeč, který komunikuje se serverem prostřednictvím protokolu http (HyperText Transfer Protocol).

XHTML viz. -> **HTML**

XML schémata Jazyk navržený pro omezení struktury a obsahu XML dokumentů. Překonávají hlavní problémy DTD. Používají se v architektuře webových služeb. Jejich nevýhodou je snad až velká složitost a poněkud nesystémové řešení datových typů.

XML (eXtensible Markup Language) je značkovací jazyk obsahující příkazy definující syntax (strukturu) dokumentu, definovaný doporučením W3C (WWW Consortium). Navazuje na něj řada dalších standardů/technologií např. DTD (Document Type Definition), XSTL (XSL Transformations), SOAP (Simple Object Access Protocol).

XPath Určuje jednotlivé části dokumentu XML. Používá kompaktní syntaxi, odlišnou od XML, která umožňuje užití jazyka XPath v adresách URI a v hodnotách atributů XML. Operuje s abstraktní logickou strukturou dokumentu XML, nikoliv s jeho povrchovou syntaxí.

Další termíny

extrakce informací Typ vyhledávání informací, jehož cílem je automaticky vypsát strukturované nebo polostrukturované informace z nestrukturovaných strojově čitelných dokumentů.

fuzzy logika Obor logiky, zobecňující klasickou booleovskou logiku na spojitý interval pravdivostních hodnot výroků. Konkrétní vlastnosti jsou zde jednoznačně určeny volbou -> **t-normy**, funkčního předpisu pro průběh hodnot spojky silné konjunkce.

indukce wrapperu Využití metod strojového -> **induktivního učení** při automatické tvorbě -> **wrapperu**.

induktivní učení Učení se z příkladů. Cílem induktivního učení je nalezení hypotézy, která by popisovala všechny vzorové příklady a vytvářela další předpovědi.

inference Proces odvození závěrů za použití známých předpokladů. Tento proces studuje logika. Rozlišujeme inferenci deduktivní, induktivní, abduktivní a retroduktivní, podle použitého způsobu odvozování. K výpočetnímu usuzování na nejistoty se užívá bayesovská pravděpodobnostní logika, frekvenční statistika, -> **fuzzy logika**, nebo nemonotónní logika.

literál Reprezentace posloupnosti znaků nebo symbolů. V oblasti moderních -> **ontologií** může literál tvořit primitivní hodnotu datotypové vlastnosti.

Łukasiewiczova logika Konkrétní reprezentace -> **fuzzy logiky**, daná volbou funkce $t(x,y) = \max(x + y - 1; 0)$ za -> **t-normu**. Oproti obecné -> **fuzzy logice** zde platí navíc některé speciální zákony.

ontologie Filozofická disciplína, která se zabývá jsouncem, bytím jako takovým a jeho základními pojmy. Ontologie představuje v klasické filozofické systematice, součást metafyziky, konkrétně všeobecnou metafyziku. V moderní informatické terminologii představuje ontologie explicitní a formální specifikaci konceptualizace. V tomto významu je tedy ontologie považována za formalizovaný znalostní model nějaké konkrétní domény.

OWL (Web Ontology Language) Značkovací jazyk navržený pro tvorbu ontologií. Rozšiřuje slovník -> **RDF** a -> **RDF**

Schémat o další elementy vztahující se k třídám a vlastnostem. Předchůdcem jazyka OWL byl jazyk DAML+OIL, který vznikl doplněním jazyka DAML (DARPA Agent Markup Language), vyvinutého americkou armádní organizací DARPA, o některé konstrukce evropského projektu OIL (Ontology Inference Layer).

RDF Schémata Jazyk navržený pro omezení možnosti zápisu tvrzení v jazyce -> **RDF**. Pomocí RDF Schémat lze formulovat jednoduchou sémantiku -> **RDF** tvrzení.

RDF (Resource Definition Framework) Základ pro zpracování metadat. Napomáhá spolupráci mezi webovými aplikacemi, které si vyměňují strojově čitelná data. RDF je v podstatě kombinace slovníku a tezauru pro značky XML. Kóduje význam pomocí jednoduchých tvrzení, jimž -> **softwarové agenty** dokáží porozumět a zpracovat je. Jedná se o systém trojic, přičemž každá trojice obsahuje subjekt, predikát a objekt nějakého tvrzení.

reifikace V oblasti metafyziky je reifikace vytvořením předmětu z něčeho, co předmětem není. V rámci -> **RDF** se za reifikaci považuje použití konstruktů jazyka na konstrukty jazyka samé. V -> **RDF** se nejedná o efektivní mechanismus, neboť jeho prostřednictvím mohou vznikat nekonzistence a prázdná místa v modelu.

sémantický web Pojem, který od roku 2001 prosazuje a zaštiťuje konsorcium W3C. Jedná se o vizi budoucnosti webu, jako informačního prostoru, kde budou dokumenty a data v nich obsažená publikovány spolu s jejich sémantikou a sice v takové formě, která je strojově zpracovatelná s přijatelnou přesností.

softwarový agent Část softwaru, která pracuje autonomně a proaktivně. Koncept softwarových agentů se vyvinul z konceptů objektově orientovaného programování a komponentově založeného vývoje softwaru. Osobní agent na -> **sémantickém webu** bude dostávat úkoly/dotazy a preference od uživatele, vyhledávat informace z webových zdrojů,

komunikovat s jinými agenty, porovnávat informace o uživatelských požadavcích a preferencích, činit jistá rozhodnutí a poskytovat odpovědi uživateli.

***t*-norma** Pravdivostní funkce logické spojky silné konjunkce. Tato funkce je rozšíření booleovské konjunkce na spojitý interval hodnot a jejím zvolením jsou definovány vlastnosti ostatních logických spojek -> **fuzzy logiky** a tedy určuje její konkrétní reprezentaci.

wrapper V kontextu -> **sémantického webu** se tímto pojmem označuje program extrahující informace z polostrukturovaných dokumentů.

ZDROJE

všechny uvedené URL adresy jsou platné k 30.4.2006.

- [Anton, 05] Anton T.: XPath-Wrapper Induction by generalizing tree traversal patterns, in: Wissensentdeckung und Adaptivität (LWA), 2005, www.ke.informatik.tu-armstadt.de/lehre/diplomandenseminar/folien/anton-lwa05-revised-2006-02-15.pdf
- [Antoniou, Harmelen, 04] Antoniou, G., van Harmelen, F.: A Semantic Web Primer, Cambridge MA.: MIT Press, 2004, 272 s., ISBN 0-262-01210-3
- [Baumgartner et al., 01] Baumgartner R., Flesca S., Gottlob G.: Visual Web information extraction with Lixto, in: The VLDB Journal, strany 119–128, 2001, <http://www.dbai.tuwien.ac.at/staff/baumgart/lixto/vldb.pdf>
- [Carme et al., 04] Carme J., Lemay A., Niehren J.: Learning node selecting tree transducer from completely annotated examples, in: ICGI, strany 91–102, 2004, <http://www.grappa.univ-lille3.fr/~carme/publi/nst.pdf>
- [Ceresna, Gottlob, 05] Ceresna M., Gottlob G.: Query based learning of XPath fragments, in: Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web, 2005, <http://www.dbai.tuwien.ac.at/staff/ceresna/wrap/Query%20Based%20Learning%20of%20XPath%20Fragments.pdf>
- [Hájek, 98] Hájek P.: Metamathematics of fuzzy logic, Dordrecht: Kluwer, 1998, 312 s., ISBN: 0-792-35238-6
- [Haussler, 90] Haussler D.: Probably Approximately Correct Learning, University of Carolina, Santa Cruz, 1990, <http://citeseer.ist.psu.edu/haussler90probably.html>

- [Hogue, Karger, 04] Hogue, A., Karger, D.: Wrapper Induction for End-User Semantic Content Development, in: Interaction Design and the Semantic Web Conference, 2004, <http://interaction.ecs.soton.ac.uk/idsw04/>
- [Chang et al., 03] Chang, Ch. H., Hsu, Ch. N., Lui, S. Ch.: Automatic information extraction from semi-structured Web pages by pattern discovery, in: Decision Support Systems, Amsterdam: Elsevier Science Publishers B. V., 2003, strany 129-147, ISSN:0167-9236
- [Kearns, Vazirani, 94] Kearns M., Vazirani U.: An introduction to computational learning theory, Cambridge MA.: MIT Press, 1994, 221 s., ISBN:0-262-11193-4
- [Kushmerick, 97] Kushmerick, N.: Wrapper induction for information extraction, PhD thesis, University of Washington, 1997, Department of Computer Science and Engineering
- [Mitchel, 97] Mitchell, T.: Machine Learning, 2. vyd., New York: McGraw-Hill, 1997, 414 s., ISBN 00-7042-807-7
- [Muslea et al., 99] Muslea, I., Minton, S., Knoblock, C.: A Hierarchical Approach to Wrapper Induction, in: Proc. 3rd Conference on Autonomous Agents, 1999, <http://www.isi.edu/~muslea/papers.html>
- [Nekvasil, 04] Nekvasil M.: Porovnání tautologií výrokové fuzzy logiky s dvouhodnotou, bakalářská práce, VŠE v Praze, 2004
- [Papakonstantinou et al.,95] Papakonstantinou, Y., Garcia-Monlina, H., Widom, J.: Object exchange across heterogenous sources, in: Proc. 11th Conference on Data Engineereng, 1995, <http://citeseer.ist.psu.edu/papakonstantinou95object.html>
- [W3C, 01] World Wide Web Consortium: Semantic Web Activity, 2001, <http://www.w3.org/2001/sw/>

- [W3C, 04, 1] World Wide Web Consortium: RDF/XML Syntax Specification (Revised), 2004, <http://www.w3.org/TR/rdf-syntax-grammar/>
- [W3C, 04, 2] World Wide Web Consortium: RDF Vocabulary Description Language 1.0: RDF Schema, 2004, <http://www.w3.org/TR/rdf-schema/>
- [W3C, 04, 3] World Wide Web Consortium: OWL Web Ontology Language Guide, 2004, <http://www.w3.org/TR/owl-guide/>
- [W3C, 99] World Wide Web Consortium: XML Path Language (XPath), 1999, <http://www.w3.org/TR/xpath>
- [Zhao et al., 05] Zhao, H., Meng, W., Wu, Z., Raghavan, V., Yu, C.: Fully automatic wrapper generation for search engines, New York: ACM Press, 2005, 75 s., ISBN:1-59593-046-9