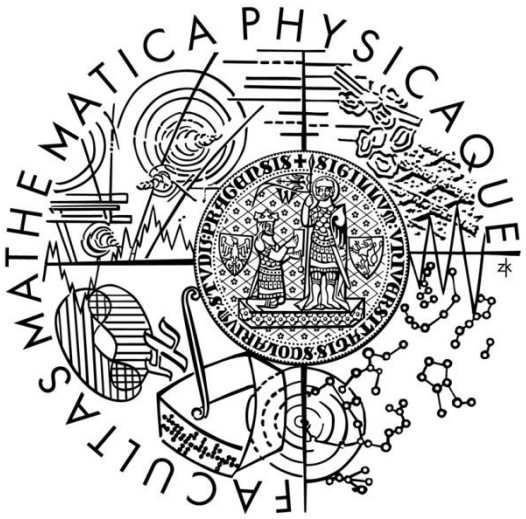
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jakub Stárka

Konvertor RDF formátů

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Jiří Dokulil

Studijní program: Informatika, programování

2007

Rád bych poděkoval vedoucímu této práce, panu magistru Jiřímu Dokulilovi, za všechny jeho podnětné rady, které mi velmi pomohly při jejím vytváření. Dále bych chtěl poděkovat své rodině a blízkým, kteří mi umožnili v klidu a pohodě tuto práci dokončit.

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

V Praze dne 9.8.2007 Jakub Stárka

Obsah

Název práce: Konvertor RDF formátů
Autor: Jakub Stárka

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Jiří Dokulil

e-mail vedoucího: jiri.dokulil@mff.cuni.cz

Abstrakt: RDF standard je jeden ze základních stavebních kamenů Sémantického webu. RDF/XML bylo navrženo jako primární způsob serializace RDF, avšak kvůli jeho složitosti a nízké čitelnosti zápisu pro člověka vzniklo několik alternativních, navzájem nekompatibilních, textových zápisů RDF dat.
Cílem práce je vytvořit nástroj pro konverzi mezi různými způsoby serializace RDF dat, například RDF/XML, Turtle, N3 nebo N-Triples. Nástroj musí být schopen zpracovat i všechny pokročilé způsoby zápisu RDF trojic definované v jednotlivých standardech. Možné rozšíření práce je využití těchto zkrácených forem i při generování cílové reprezentace dat.
Software musí být vytvořen s důrazem na maximální výkon a korektní podporu Unicode znaků.

Klíčová slova: RDF, N-Triples, Turtle, RDF/XML

Title: RDF format convertor

Author: Jakub Stárka

Department: Department of Software Engineering

Supervisor: Mgr. Jiří Dokulil

Supervisor's e-mail address: jiri.dokulil@mff.cuni.cz

Abstract: RDF is one of the basic building blocks of the Semantic web. RDF/XML was designed to be the primary form of serializing RDF. It has a complex and hard to read syntax which lead to creation of several alternatives, that are usually human readable but incompatible with the other alternatives.
The goal of the thesis is to create a conversion tool for various RDF serialization formats, e.g. RDF/XML, Turtle, N3, and N-Triples. The tool must correctly handle all of the advanced syntax of the languages on the input side. Support for these shortcuts in output is not required.
The implementation should be focused on high performance and correct Unicode handling.

Keywords: RDF, N-Triples, Turtle, RDF/XML

1 Úvod

Zatímco pro člověka není problém hledat a číst informace z webu, při strojovém zpracování dat je to mnohem složitější. Většinou nezbývá, než aby se aplikace přizpůsobila konkrétním datům. Z těchto důvodů vzniklo rozšíření WWW, nazvané sémantický web, který umožňuje nejen získávání a zpracovávání informací člověkem, ale i snadné a efektivní zpracování stroji.

Jedním z nejdůležitějších úkolů sémantického webu je popsat vztahy mezi jednotlivými daty (zdroji). Právě k tomu slouží jeden ze základních stavebních kamenů sémantického webu, kterým je RDF [1] (Resource Description Framework). V rámci RDF jsou tyto vztahy zapisované jako trojice, ze kterých se tvoří rozsáhlé grafy.

Základním prostředkem pro zápis RDF grafů se stal formát RDF/XML [6]. Díky tomu, že je založen na značkovacím jazyce XML, je snadno zpracovatelný dostupnými nástroji, ale výsledné soubory jsou relativně velké a navíc čitelnost tohoto formátu není pro člověka nejlepší. Z tohoto důvodu vzniklo několik dalších formátů, které se snaží tyto vlastnosti zlepšit, ale zároveň ponechat všechny výhody, které RDF přináší. K tomu tyto formáty využívají různé konstrukce, které jsou mezi sebou většinou zcela nekompatibilní.

Tato práce se zaměřuje na převod mezi formátem RDF/XML a některými dalšími formáty pro zápis RDF grafů. Jejím cílem není kontrolovat správnost vstupních dat, ale naopak co nejrychleji a s minimálními nároky na hardware převádět dokumenty mezi jednotlivými formáty.

V kapitole 2 je popsán krátký úvod do datového modelu RDF. Obsahuje jeho specifické vlastnosti a formáty, na které se tato práce zaměřuje. Hlavním cílem této kapitoly je dát čtenáři názornou představu o tom, jak RDF grafy vypadají a o tvaru jeho zápisu v jednotlivých formátech. Kapitola 3 je věnována programu a jeho základním vlastnostem. Popisuje jeho ovládání, věnuje se hlavním datovým strukturám a uvádí jeho základní algoritmy. Kapitoly 4 a 5 jsou zaměřeny na bližší prozkoumání jednotlivých formátů. Obsahují konstrukce, se kterými je možné se setkat při procházení vstupních souborů a naopak způsob, jakým jsou vytvářeny soubory výstupní. Zároveň ukazují na některé problémy, se kterými jsem se v průběhu práce setkal, a použitá řešení. Kapitola 6 se blíže věnuje jmenným

prostorům. Popisuje vlastnosti a problémy vycházející z jejich používání v Turtle, případně RDF/XML. V kapitole 7 jsou popsány používané způsoby ukládání dat a kódování. Kapitola 8 se zaměřuje na výsledky některých měření při zpracovávání objemnějších souborů. Zároveň poskytuje analýzu výsledných hodnot a možnosti, jakými by bylo možné dosáhnout zlepšení.

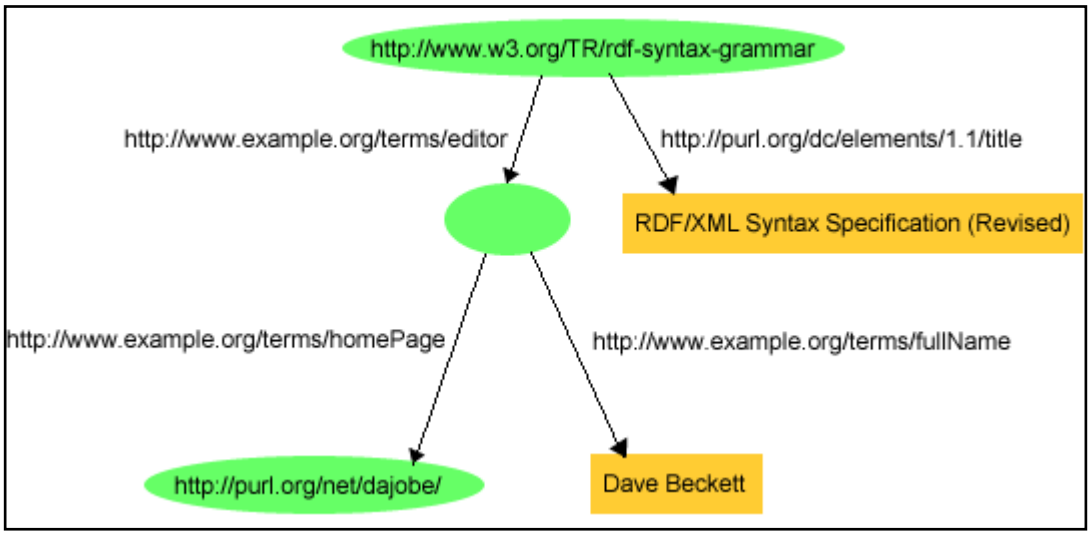
2 RDF

Specifikaci RDF jako datového modelu vydalo konsorcium W3 v roce 1999. Přestože byl původně navržen jako datový model pro reprezentaci metadat, používá se v současné době jako obecný prostředek pro modelování informací nebo zdrojů v mnoha různých syntaxích.

Základem tohoto modelu jsou trojice (triples). Každá trojice obsahuje subjekt (uzel, kterého se informace týká), predikát (vlastnost, kterou trojice popisuje) a objekt (hodnotu dané vlastnosti). Trojice se spojují do orientovaného RDF grafu. Vrcholy tohoto grafu jsou subjekty a objekty. Predikáty jsou reprezentovány pomocí hran, které vždy směřují od subjektu k objektu.

Následující graf (obr. 1) popisuje 4 trojice

- internetová stránka "http://www.w3.org/TR/rdf-syntax-grammar" (subjekt), jejíž název (predikát "http://purl.org/dc/elements/1.1/title") je "RDF/XML Syntax Specification (Revised) " (objekt)
- jejího editora (predikát) charakterizuje prázdný uzel (někdo = objekt)
- homePage někoho, kdo je editor, je "http://purl.org/net/dajobe/" a jeho jméno je "Dave Beckett"



Obrázek 1:RDF graf

Při zápisu RDF grafů v následujícím textu bude používáno pro přehlednost formátu Turtle. Každá trojice bude zaznamenána na nový řádek a nebude používáno žádných pokročilých metod pro zápis, kromě používání prefixů pro zkrácení dlouhých URI, které zlepší přehlednost textu. Tato metoda slouží k tomu, že část URI, která je často používaná, je nahrazena prefixem a za ním následuje dvojtečkou oddělená lokální část adresy.

V případě, že nebudou celé adresy potřeba k vysvětlení příkladu, budou klauzule *@prefix* vynechány. Používání prefixu *rdf* vždy značí jmenný prostor "http://www.w3.org/1999/02/22-rdf-syntax-ns#".

Například předchozí graf by byl v tomto formátu zapsán následovně.

```
@prefix dc: < http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://www.example.org/terms/> .
<http://www.w3.org/TR/rdf-syntax-grammar> ex:editor _:blank1 .
_:blank1 ex:homePage "http://purl.org/net/dajobe/" .
_:blank1 ex:fullName "Dave Beckett" .
```

2.1 Vlastnosti

2.1.1 Typy uzlů

Pro identifikaci zdrojů, bez ohledu na to, zda jsou umístěny jako subjekt, predikát nebo objekt, se používají URI (Uniform Resource Identifiers), což jsou jednoznačné identifikátory na webu.

Některé uzly není možné přímo jednoznačně pojmenovat. Podobné uzly jsou v RDF reprezentovány pomocí tzv. "blank node" (anonymní uzly). Tyto uzly nejsou v rámci RDF grafu pojmenované, ale jsou určeny svými vlastnostmi. Poskytují například prostředek k zápisu strukturovaných objektů. V Turtle jsou zapisované pomocí prefixu "_:".

Literály v RDF je možné vytvářet pouze jako objekty. RDF mohou být typované, s definicí jazyka nebo jako prosté. Slouží k ukládání hodnot, jako jsou čísla nebo řetězce.

2.1.2 Kontejnery v rámci RDF

RDF umožňuje pracovat s kontejnery, které umožňují pracovat s množinou prvků.

- *rdf:Alt* – Množina alternativních hodnot. Například popis alternativních jazyků, do kterých je přeložená kniha.
- *rdf:Bag* - Neuspořádaná množina hodnot. Může obsahovat některé hodnoty vícekrát.
- *rdf:Seq* - Uspořádaná množina hodnot. Může obsahovat některé hodnoty vícekrát.

Pro zápis jednotlivých prvků množin se používají buď predikát *rdf:li*, nebo pro jednotlivé prvky predikáty *rdf:_1, rdf:_2, ..., rdf:_n* .

Kontejnery jsou v RDF reprezentovány prázdným uzlem, ze kterého vedou predikáty k jednotlivým prvkům a predikát *rdf:type* k typu kontejneru.

Následující trojice, ve které je kontejner typu *rdf:Bag*, reprezentován anonymním uzlem *blank1*, reprezentují neuspořádanou množinu tří studentů ve třídě.

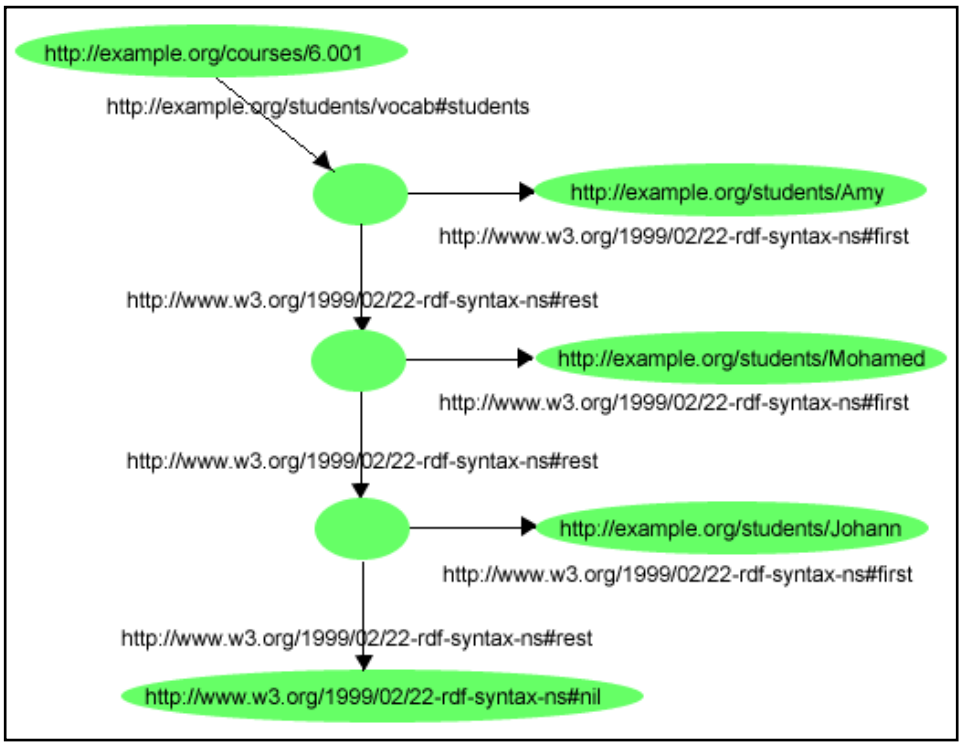
```
ex:trida ex:student _:blank1 .
_:blank1 rdf:type rdf:Bag .
_:blank1 rdf:_1 "Jan" .
_:blank1 rdf:_2 "Petr" .
_:blank1 rdf:_3 "Zuzana" .
```

Případně při použití *rdf:li*.

```
ex:trida ex:student _:blank1 .
_:blank1 rdf:type rdf:Bag .
_:blank1 rdf:li "Jan" .
_:blank1 rdf:li "Petr" .
_:blank1 rdf:li "Zuzana" .
```

2.1.3 Lists

Kontejnery nám říkají, jaké prvky patří do množiny, ale neříkají, jestli neexistují ještě nějaké jiné. Proto existují v RDF seznamy (collections), které slouží k ukládání všech prvků patřících do dané množiny.



Obrázek 2: Graf seznamu

Tyto kolekce jsou v RDF definovány jako spojové seznamy. Každý prvek v seznamu obsahuje jednak položku, na kterou ukazuje predicate *rdf:first* a dále predicate *rdf:rest*, který ukazuje na další prvek. Poslední prvek ukazuje na *rdf:nil* (Obrázek 2).

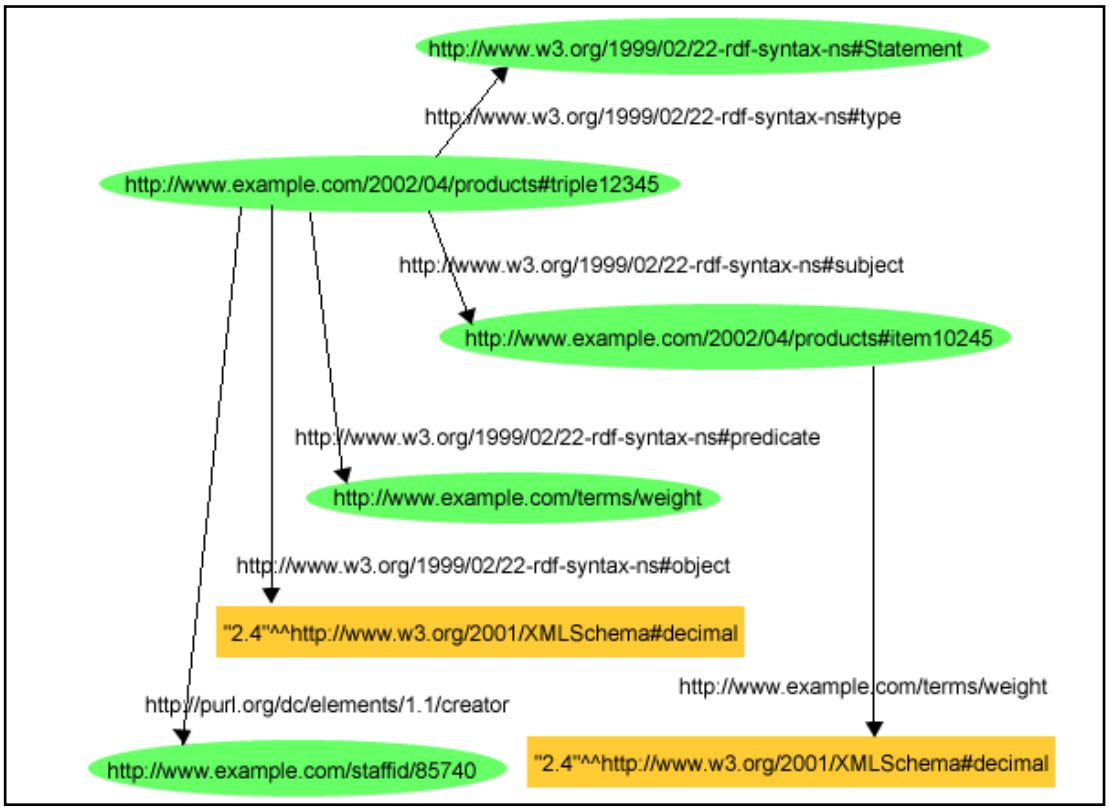
2.1.4 Reifikace

Někdy je potřeba popsat celou trojici pomocí dalších RDF trojic. Například chceme-li říct, kdo danou trojici vytvořil. Pro podobné účely slouží tzv. reifikace.

Mějme následující trojici, která popisuje váhu položky. Krom samotné hodnoty váhy je popsán i datový typ objektu "2.4", kterým je *xsd:decimal*.

```
exproducts:item10245    exterms:weight    "2.4"^^xsd:decimal .
```

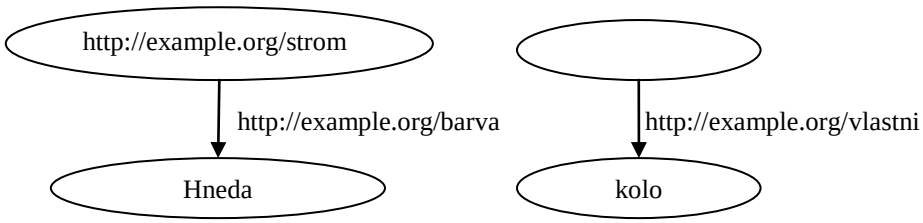
Vytvoří se nový anonymní uzel, který reprezentuje trojici. Od něj vedou predikáty *rdf:type*, *rdf:subject*, *rdf:predicate* a *rdf:objekt*, které mají jako objekty původní položky trojice (Obrázek 3). Krom toho obsahuje graf i původní trojici a navíc je možné popsat tvůrce celého tripletu použitím anonymního uzlu jako subjektu.



Obrázek 3 : Reifikace

2.2 Formáty

Pro ukázku jednotlivých formátů je použit následující graf skládající se ze dvou trojic (Obrázek 4). První je popis stromu (identifikovaným adresou "http://example.org/strom"), jehož vlastnost "http://example.org/barva" nabývá hodnoty "Hneda". Druhá trojice popisuje někoho (jako subjekt je v tomto případě použit anonymní uzel), kdo vlastní kolo.



Obrázek 4: RDF graf dvou trojic

Základním jazykem pro ukládání RDF grafů je **RDF/XML** [6]. Jazyk je založený na značkovacím jazyce XML, což usnadňuje jeho zpracovávání dostupnými prostředky pro práci s XML a přitom dostatečné možnosti pro složitější konstrukce. Bohužel kvůli vlastnostem XML je tento zápis celkově rozsáhlejší a navíc umožňuje zapsat stejnou informaci mnoha různými způsoby.

Typický dobře vytvořený RDF/XML dokument by měl, stejně jako každý XML dokument, začínat prologem. Ten obsahuje verzi XML a kódování, v jakém je soubor uložen. Za ním následuje právě jeden kořenový element. Tím je většinou element *rdf:RDF*, ale v některých případech to může být i jiný element (viz. Kapitola 4.4).

Následuje zápis předchozího grafu pomocí RDF/XML.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:ex="http://example.org/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:about="http://example.org/strom">
      <ex:barva>Hneda</ex:barva>
</rdf:Description>
<rdf:Description rdf:nodeID="x">
      <ex:vlastní>kolo</ex:vlastni>
</rdf:Description>
</rdf:RDF>
```

Pro kontrolu a jednoduchost vznikl formát **N-Triples** [2]. Ten se nezaobírá strukturou informací a obsahuje přímo trojice. Formát je celkem dobře čitelný pro člověka, ale neumožňuje snadný a efektivní způsob zápisu pokročilejších konstrukcí jazyka RDF. Formát umožňuje zápis zdrojů pomocí URI nebo prázdných uzlů a lze s ním zapisovat jednořádkové komentáře. Pracuje se základním sedmibitovým US–ASCII kódováním. Pro unicode znaky je nutné používat zástupné znaky. Každá trojice leží na novém řádku a je ukončena tečkou.

```
<http://example.org/strom> <http://example.org/barva> "Hneda" .
_:x http://example.org/vlastní "kolo" .
```

Dalším formátem významně rozšiřujícím N-Triples je **Turtle** [3]. Umožňuje použití konstrukcí, které zkracují celkový zápis, a poskytuje přirozené prostředky pro zápis prázdných uzlů a seznamů. Mezi takové prostředky patří například použití znaků "[" a "]" pro vyjádření anonymních uzlů nebo použití QName pro zkrácení dlouhých URI. Zápis předchozího grafu je možné napsat stejně jako v případě N-Triples. Zároveň je možné použít zkrácení a zapsat ho následovně.

```
ex:strom ex:barva "Hneda" .
[ ex:vlastni "kolo" ] .
```

Notation 3 [4] je formát pro zápis RDF trojic, který zároveň poskytuje prostředky pro zápis kvantifikátorů, formulí a klíčových slov. Všechny tyto prostředky není možné v rámci zápisu RDF trojic reprezentovat, a tak jsem se zaměřil jen na podmnožinu, která pokrývá RDF 1.0.

Notation 3 se snaží o lepší čitelnost pro člověka. Z tohoto důvodu bylo přidáno klíčové slovo "has", které má přiblížit RDF trojice běžným větám. Jelikož je Notation 3 stále ve stádiu vývoje, není mnoho věcí plně vyřešeno a v popisu gramatik jsou značné nejasnosti. Z tohoto důvodu je i implementace Notation 3 parseru spíše orientační. Notation 3 neposkytuje v porovnání s Turtle o mnoho více prostředků, které by se daly použít k serializaci. Proto je zbytečné pro tento formát vytvářet samostatný serializer.

Graf (Obrázek 4) je možné zapsat stejně jako v případě N-Triples nebo Turtle. Zároveň je možné při využití existenčního kvantifikátoru zapsat předchozí graf jako

```
ex:strom ex:barva "Hneda" .

@forSome <v> .

<v> ex:vlastni "kolo" .
```

3 Programová příloha

Tato kapitola popisuje použití programu a zároveň ukazuje jeho základní datové struktury a algoritmy.

3.1 Použití

Na přiloženém CD se nachází program pro konverzi mezi různými formáty RDF.

Pro napsání programu jsem zvolil jazyk C++. Hlavním důvodem je rychlost výsledné aplikace a dobrá přenositelnost kódu.

Jedná se o konzolovou aplikaci, která umí jako vstup zpracovat soubory ve formátu N-Triples, Turtle, RDF/XML a Notation 3. Jako výstup je možné nastavit N-Triples, Turtle nebo RDF/XML. Výstup ve formátu Notation 3 by byl totožný s výstupem ve formátu Turtle, proto nebyl implementován.

Při serializaci, kromě formátu N-Triples, dochází k částečnému zkracování výstupu tak, jak to formáty umožňují. V případě RDF/XML je možné nastavit, zda k tomuto zkracování bude docházet.

Program pracuje na příkazové řádce. Při spuštění je nutné prostřednictvím parametrů zadat vstupní a výstupní soubory a jejich formáty.

```
Pouziti:
--------
rdf -iX <input_filename> -oY <output_filename>

Parametry
---------
-iX <input_filename> - Vstupni format souboru nabyva hodnot :
        X - RDF-XML
        T - turtle
        N - n-triples
        3 - Notation 3
        input_filename - vstupni soubor
-oY <output_filename> - Vystupni format souboru nabyva hodnot :
```

3.2 Fungování

3.2.1 Použité knihovny

Bylo by zbytečné programovat vlastní rozhraní pro práci s XML soubory, když existuje mnoho knihoven, které to umožňují.

Stejně jako XML bylo potřeba získat podporu pro Unicode znaky, protože C++ pro ně nemá bohužel nativní podporu jako například Java.

V programu byly nakonec použity knihovny **Xerces** [8], které kromě zpracování XML souborů umožňují i snadnou práci se soubory. Navíc poskytují prostředky pro kódování a dekódování unicode znaků nejen do UTF-8 či ASCII. Další výhodou je přenositelnost mezi platformami.

3.2.2 Struktura programu

Základními součástmi programu jsou parsery a serializery. Parser postupně prochází soubor a vytváří subjekty, predikáty a objekty. Pro parsování formátu RDF/XML jsou použité funkce a třídy, které poskytuje Xerces pro práci s dokumenty XML. Patří mezi ně hlavně rozhraní pro práci s modelem SAX 2.0, které umožňuje proudové zpracování souboru.

V okamžiku, kdy je trojice kompletní, je předána serializeru, který se postará o její správné uložení do výsledného souboru. O přímé ukládání a načítání ze souborů se stará třída *RDF_FileStream*, která vytváří objektovou nadstavbu nad prostředky, které poskytuje Xerces.

Jelikož parser a serializer mohou mít odlišné požadavky na dobu, po kterou jsou prvky trojice dostupné, udržuje si každý prvek počet pointerů, které na něj směřují. V okamžiku, kdy na něj nesměřuje žádný, je smazán.

3.2.3 Detailnější popis hlavních tříd

Základní třídy programu jsou odvozené od abstraktní třídy *RDF_format*. Od ní jsou dále odvozeny další dvě abstraktní třídy. Jednak *RDF_Parser*, který definuje rozhraní pro všechny parsery a potom *RDF_Serializer*, který definuje rozhraní pro serializery.

Každá třída pro parsování dokumentu musí mít implementovanou funkci *proccess_file(char* file, RDF_Serializer * s)*, která jako argumenty požaduje soubor k parsování a serializer. Odvozenými třídami od *RDF_Parser* jsou třídy

- NTriplesParser
- Notation3Parser
- TurtleParser
- RDFXMLParser

Třídy pro serializaci musí implementovat funkci *put_triple(Node* subject, Predicat* predicate, Node* object)*. Pomocí této funkce posílá parser jednotlivé trojice ke zpracování. Zároveň jsou potřeba funkce *InitTriples()* a *TerminateTriples()*, které označují začátek a konec zápisu. V RDF/XML se pomocí nich zapisuje například prolog nebo ukončují otevřené značky. Poslední nutnou funkcí pro serializace je funkce *AddFeature()*. Pomocí této funkce předávají parsery, které používají jmenné prostory, jejich seznam serializeru. Jako její parametr je předávána mapa jmenných prostorů a jejich prefixů. Odvozenými třídami pro serializaci jsou třídy

- NTriplesSerializer
- TurtleSerializer
- SXMLSerializer
- SSXMLSerializer

Každá trojice je reprezentovaná třemi elementy. Pro každý element jsou použité třídy odvozené od třídy *RDF_Elem*. Třídy obsahují informaci o své hodnotě, případně prefixu a jeho délce. V případě třídy pro literály *LiteralNode* zahrnuje navíc svůj datový typ a jazyk. U predikátů jsou to například informace o kolekcích či o nucené reifikaci.

Vzhledem k tomu, že není možné při serializaci a parsování určit, kdy vytvořený uzel už nebude potřeba (například, když parser N-Triples načte trojici, už ji nepotřebuje, zatímco serializer formátu Turtle si ponechává uzel uložený pro případ, že by následující trojice měla stejný subjekt), existuje v programu speciální třída, která uzly "sbírá". Každý uzel si v sobě nese počet linků, které na něj ukazují. Parsery ani serializery nemažou přímo uzel, ale pošlou ho statické třídě *Collector*, která

zkontroluje, zda je počet linků nulový pomocí funkce *CheckLinks()* a až pak ho smaže.

Pro jakoukoliv práci se soubory existuje třída *RDF_FileStream*. Tato třída slouží pro zápis i čtení ze souboru. Při načítání i ukládání používá svého vnitřního bufferu, aby zbytečně nezpomalovala program častými zápisy na disk.

Při spuštění programu se inicializují predikáty a uzly, které jsou často využívány, případně zkracovány. Aby nebyly ničeny Collectorem, vrací jejich funkce *CheckLinks()* vždy false.

4 Zpracování souboru

Během zpracování vstupního souboru dochází k získávání jednotlivých subjektů, predikátů a objektů. Po jejich získání jsou celé trojice odeslány serializeru. Každý formát má své specifické konstrukce pro reprezentaci trojic, které jsou popsány v této části.

4.1 N-Triples

N-triples je jednoduchý řádkový formát navržený jako podmnožina specifikace Notation 3.

Každá trojice leží na novém řádku. N-Triples neumožňuje nijak zkrátit zapsané trojice, proto na každém řádku leží právě jeden subjekt, predikát a objekt vzájemně oddělené bílými znaky, které v tomto případě zahrnují jen mezery a tabelátory. N–Triples umožňuje zápis typovaných literálů a literálů s definovaným jazykem. Umožňuje zapisovat jednořádkové komentáře.

Pro zápis anonymních uzlů používá prefix "_:", URI jsou uzavřeny do "<" a ">".

```
#typicka trojice
_:blank <tady je uri> "Literal" .
```

Formát je definován jen nad sedmibitovým ASCII kódováním se zástupnými znaky (viz Kapitola 9).

4.2 Turtle

Kvůli těžkopádnosti formátu N-Triples vznikl formát Turtle, který ho rozšiřuje a zároveň přidává některé pokročilé vlastnosti z formátu Notation 3. Mezi tyto vlastnosti patří zápis prázdných uzlů pomocí "[" a "]",

```
[ ex:Jmeno “Jan” ] ex:vlastni “Kolo” .
```

jednoduchý zápis seznamů do kulatých závorek

```
ex:course ex:students ( Amy, Mohamed, Johann ) .
```

a především používání prefixů, které značně zkracují výsledný soubor oproti N-Triples. V okamžiku, kdy parser narazí na nový prefix, uloží ho do asociativního pole, které sdílí se serializerem. Ten ho může v případě potřeby použít pro zápis trojic.

Turtle navíc umožňuje zkracování zápisu při používání stejných subjektů a predikátů. Pomocí znaku ";" lze použít stejný subjekt pro více predikátů a pomocí "," lze použít stejný subjekt a predikát pro více objektů.

Následující graf popisuje osobu, její matku, otce a sourozence. Subjekt zůstává pro všechny trojice stejný a v případě sourozenců zůstává stejný i predikát.

```
ex:clovek   ex:matka "Ruzena";
            ex:otec "Josef";
            ex:sourozenci     "Frantisek",
                              "Zbynek" .
```

Mezi další prvky patří používání zkráceného zápisu URI "*http://www.w3.org/1999/02/22-rdf-syntax-ns#type*" pomocí "*a*" a používání číselných datových typů. Jejich používání vyvolává při prohlížení kódů nejasnosti. Vzhledem k tomu, že mezi koncem literálu a ukončovací tečkou nemusí být mezera nebo jiný bílý znak, může nastat následující případ, kdy není při čtení "e" jasné, jestli se bude jednat o QName, jako v následujícím případě

```
@prefix e1: <http://example.cz/> .
<a> <b> 1.e1:d <g> <f> .
```

nebo zda se bude jednat o číslo.

```
@prefix e1: <http://example.cz/> .
<a> <b> 1.e1 .
```

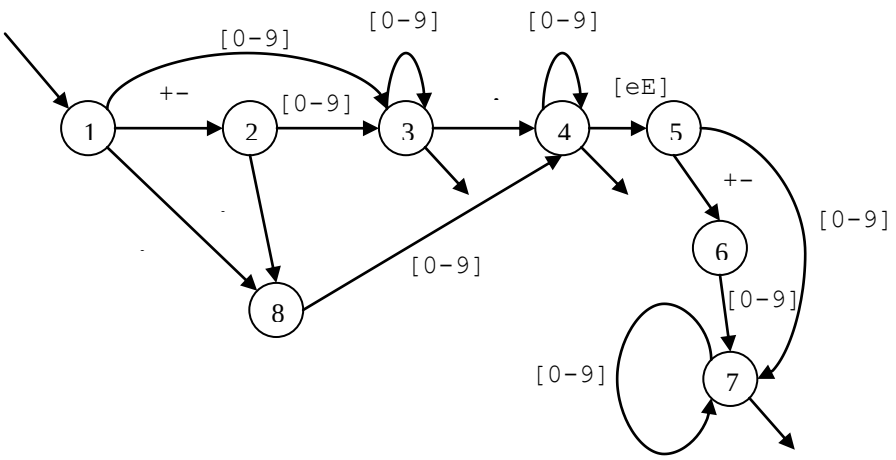
Podobný problém může nastat s tečkou, v tomto případě není jasné, jestli značí desetinou čárku

```
a b 1. . d e f.
```

nebo zda ukončuje aktuální trojici, jako v následujícím případě.

```
a b 1.   d e f.
```

Z tohoto důvodu je při zpracovávání čísel třeba speciálně ošetřit tyto případy. Vzhledem k sekvenčnímu zpracovávání souboru není možné se vracet, ale program si musí pamatovat kritické místo a pokračovat v procházení, dokud není možné jednoznačně určit význam trojice. Pro zpracovávání čísel je v programu použit jednoduchý stavový automat (Obrázek 5).



Obrázek 5 :Stavový automat pro zpracování čísel v Turtle

Vycházel jsem z toho, že číslo musí být správně zapsané. V tom případě se v automatu vyskytují tři přechody přes tečku.

- ze stavu 1 do stavu 8
- ze stavu 2 do stavu 8
- ze stavu 3 do stavu 4

V prvním ani v druhém případě není s tečkou žádný problém, protože stav číslo 8 není koncový, tudíž je význam jasný. Ve třetím případě dochází k tomu, že stav číslo 3 je koncový, tudíž tečka může, ale nemusí patřit k číslu. V tomto okamžiku program začne načítat další znaky. V případě, že prvním znakem za tečkou je číslo, znamená to, že tečka oddělovala desetinou část, jelikož Turtle neumožňuje použít jako první znak prefixu číslo. Jinak nezbývá nic jiného, než dohledat první bílý nebo

ukončovací znak. Při načítání jsou znaky ukládány do spojovaného seznamu bufferů (v případě, že by jeden nestačil). Pokud se ve výsledku ukáže, že šlo o číslo, smažou se uložené znaky a může se pokračovat dál. V opačném případě se nastaví pozice parseru na pozici tečky, automat vrátí číslo bez tečky.

Turtle je definovaná nad kódováním UTF-8 a umožňuje používání znaků unicode nebo stejné escape sekvence jako v N-Triples.

4.3 Notation 3

Notation 3 je formát, kterým lze zapsat RDF grafy. Zároveň ale umožňuje širší možnosti využití. Kromě prostředků, které sdílí Notation 3 s Turtle (těmi jsou zkracování stejných subjektů a predikátů pomocí "," a ";", používání "[" a "]" místo blank nodu), poskytuje Notation 3 prostředky pro práci s proměnnými, formulemi a kvantifikátory. Některé z těchto prostředků nejsou součástí modelu RDF, a tak jsem se při parsování Notation 3 zaměřil jen na vlastnosti pokrývající čisté RDF. Jelikož je tento formát stále ve stádiu vývoje, jsou i popsané specifikace občas zmatečné nebo nekompletní. Vycházel jsem proto ze specifikace N3 RDF a částečně ze specifikace Turtle.

Ve srovnání s formátem Turtle navíc Notation 3 používá zkratky pro snadnější čitelnost. První je zkratka *"has"*, která působí jen jako spojka pro snadnější čitelnost predikátu. Při parsování se zcela ignoruje a čte se pouze predikát, kterému zkratka předchází. Další možnou zkratkou je znak *"="*, který tu zastupuje predikát "*http://www.w3.org/2002/07/owl#sameAs"*.

Pro zápis prázdných uzlů nabízí oproti Turtle navíc paths. Paths zkracují hlavně rozsáhlé soustavy anonymních uzlů. Jedná se o zkrácení pomocí binárního operátoru "!", který vytvoří anonymní uzel v roli objektu, nebo použití znaku "^" jakožto zkrácení pro blank v roli subjektu. Následující výraz

```
:joe!fam:mother^fam:mother
```

je pak přeložen jako někdo, jehož matka je Joeova matka. Případně jde v Turtle napsat jako

```
:joe fam:mother _:blank1 .
_:blank2 fam:mother _:blank1 .
```

Dalším prvkem, který stále je součástí gramatiky, i když už patří spíše mezi pokročilé prostředky N3, je existenční kvantifikátor *@forSome*. Pomocí něj je možné definovat anonymní uzly. Ty jsou pak v následujících trojících značeny jako URI nebo QName. To vyvolává problémy, jelikož je při každém přečtení nutné ověřit, zda se nejedná o proměnnou, a v případě, že ano, nahradit tuto proměnnou za anonymní uzel.

Například

```
@forSome <a>, <b>.
<a> <c> <b> .
```

lze pomocí Turtle zapsat jako

```
_:genblank1 <c> _:genblank2 .
```

Další vlastnosti, jako například obecný kvantifikátor, formule nebo pravidla, již nejsou součástí RDF 1.0, ale spadají pod RDF Plus.

Notation 3 je stejně jako Turtle definována nad kódováním UTF-8 a umožňuje používání stejných escape sekvencí jako N-Triples.

4.4 Stream parsing XML

Pro práci s XML dokumentem, který musí být dobře vytvořen, je potřeba zvolit vhodný model pro jeho reprezentaci. Existují dva základní modely, prvním je stromová reprezentace. Během ní je dokument celý načten do paměti a jeho složky jsou snadno přístupné. Cenou za to je vysoká paměťová náročnost. Druhou možností je model řízený událostmi, konkrétně SAX. Parser postupně dokument čte a na základě příchozích dat vyvolává události, jakými jsou například začátek či konec uzlu nebo textová data. SAX nemá tak vysoké nároky na paměť jako stromové reprezentace a díky tomu umožňuje zpracovávat i velmi velké soubory, ale kvůli proudovému zpracování se neumožňuje v rámci dokumentu volně pohybovat.

Jelikož při překladu RDF z jednoho formátu do druhého není třeba se vracet či skákat k jiným uzlům, je SAX mnohem vhodnějším řešením. Pro práci s XML existuje mnoho vhodných nástrojů a bylo by proto zbytečné se zabývat kompletní implementací XML parseru. Proto jsou v programu použity multiplatformní

knihovny Xerces, který umožňují pohodlnou práci s XML dokumenty skrze rozhraní SAX2.

Při parsování si program musí pamatovat, v jakém stavu se parser nachází, jestli čte subjekt nebo predikát. Povaha RDF vybízí k použití tří stavů, ale element představující objekt zároveň může znamenat subjekt pro zanořené uzly. Proto je při načtení objektu vytvořena trojice a poté je objekt automaticky uložen do seznamu se subjekty a odebrán až při události konce elementu.

Během zpracovávání je potřeba odlišit kořenový element dokumentu, kterým by měl být element *rdf:RDF*. Je ale zároveň možné tento element vynechat a použít jako kořenový element přímo subjekt. V tomto případě musí subject splňovat všechny podmínky kořenového elementu. Na této úrovni tedy může být jediný element.

Při zpracování elementu, který reprezentuje subjekt nebo objekt, je potřeba nejprve zjistit, zda není v atributech obsažen jeho identifikátor. Ten obsahují atributy *rdf:id, rdf:nodeId* a *rdf:about*. Na jejich základě se určí, o jaký druh uzlu se jedná. Pokud uzel ani jeden z nich nemá, jedná se o anonymní uzel.

Navíc je možné vytvářet typované uzly. Elementy uzlů jsou většinou pojmenovány *rdf:Descritpion*. V případě, že chceme vytvořit typovaný uzel, máme dvě možnosti. Jednou z možností je použít predikát *rdf:type* jako atribut nebo zanořený element. Další možností je přímo pojmenovat element uzlu. Při jeho zpracování je potřeba vygenerovat trojici, která má na pozici predikátu *rdf:type* a jako objekt jméno elementu. Následující fragment XML

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/stuff/1.0/">
  <ex:Document rdf:about="http://example.org/thing">
    <ex:title>A marvelous thing</ex:title>
  </ex:Document>
</rdf:RDF>
```

generuje dvě trojice.

```
<http://example.org/thing> ex:title "A marvelous thing" .
<http://example.org/thing> rdf:Type ex:Document .
```

Následně dochází ke zpracování všech ostatních atributů elementu. Pro každý atribut je vytvořena trojice sestávající z identifikátoru daného uzlu, jména atributu a jeho hodnoty.

Při zpracovávání elementu, který reprezentuje predikát, je potřeba také správně zpracovat všechny atributy. Nejprve je potřeba zjistit, zda element neobsahuje atributy *rdf:nodeID* nebo *rdf:resource*. Pokud atribut najdeme, jedná se o prázdný element. RDF/XML umožňuje pomocí těchto atributů zkrátit zápis o zanořený uzel s identifikátorem uloženým v daném atributu. Pro každý atribut tohoto elementu je pak generována trojice se subjektem, kterým je identifikátor uzlu. Následující XML fragment

```
<rdf:Description rdf:about="http://www.example.cz/kniha">
  <ex:editor rdf:resource="Jan Novak">
  </ex:editor>
</rdf:Description>
```

generuje v Turtle.

```
<http://www.example.cz/kniha> ex:editor "Jan Novak" .
```

Elementy reprezentující predikát mohou obsahovat atribut *rdf:parseType*, který označuje změnu přístupu při parsování. Při jeho hodnotě *Collection* značí, že se jedná o seznam. Je potřeba vytvořit anonymní uzel pro reprezentaci seznamu a postupně navazovat prvky. Jelikož se každý prvek seznamu může dále větvit, musí si predikát obsahující kolekci pamatovat poslední prvek seznamu, aby na něj mohl navázat další, případně aby mohl ukončit seznam.

Pokud *rdf:parseType* nabývá hodnoty *Literal*, jedná se o XMLLiteral. RDF/XML tím umožňuje použít jako objekt přímo fragment XML. V tomto okamžiku by bylo potřeba, aby SAX parser přestal fungovat jako parser RDF a vše, co mu v tomto okamžiku přijde jako vstup, ukládal a nakonec vrátil jako literál. Proto pro parsování RDF/XML existují dva objekty, které jsou přepínány samotným objektem pro správu událostí. První objekt zpracovává XML dokument jako RDF/XML. V okamžiku, kdy narazí na XML Literál, přepne se zpracovávání příchozích událostí na druhý objekt a veškerý vstup je ukládán do paměti. Během tohoto ukládání dochází ke kontrole zanoření elementů, a jakmile by došlo k uzavření elementu predikátu, který vyvolal

přepnutí parseru, vrátí parser trojici s XML Literálem jako objektem a přepne se zpět na parsování RDF obsahu. V průběhu parsování XML Literálu je potřeba hlídat jmenné prostory, které literál používá. Ty jsou totiž definované v rámci RDF části a při vytvoření textového literálu by se informace o konkrétním jmenném prostoru ztratila. Proto jsou elementy XML Literálu doplněny o jejich definice. Pokud by měl mít literal definovaný jazyk, je třeba přidat atribut jazyka přímo do literalu.

Poslední možnou hodnotou atributu *rdf:parseType*, která má vliv na zpracování dokumentu, je *Resource*. Tento atribut umožňuje, aby byl vynechán uzel objektu. Tímto uzlem se pak stane anonymní uzel.

```
<rdf:Description about="http://ex/a">
  <ex:autor ex:parseType="Resource" />
      <ex:jmeno rdf:resource="Jan Novak" />
</rdf:Description>
```

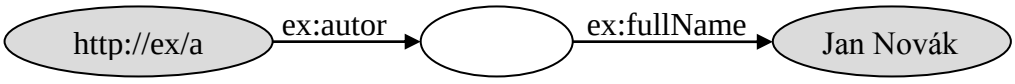
Vznikne například podobný zápis v N-Triples.

```
<http://ex/a> ex:autor _:blank1 .
```

Další možnost, jak zkrátit pomocí atributů zápis RDF/XML, je pomocí atributů umístěných přímo v elementu predikátů.

```
<rdf:Description about="http://ex/a">
  <ex:autor ex:fullName="Jan Novak" />
</rdf:Description>
```

Reprezentuje graf



Při zpracovávání XML atributů, které pracují s URI zdrojového dokumentu, jakými jsou *rdf:about, rdf:resource, rdf:ID* a *rdf:datatype*, je potřeba zpracovat hodnotu tohoto atributu dle zadané hodnoty base. V případě, že není hodnota base zadaná,

vztahují se atributy k parsovanému dokumentu (a tudíž není třeba hodnotu upravovat). Pokud je hodnota base zadaná (pomocí atributu *xml:base*), zpracování URI je popsáno v dokumentu **rfc 3986** [7].

5 Serializace

Při serializaci není třeba řešit velké problémy. Stačí jen uložit trojice do správného formátu. Některé formáty umožňují použití optimalizací výstupu, které vedou ke kratšímu, případně přehlednějšímu výstupnímu souboru. Naopak bez jakékoliv optimalizace by se výsledný soubor formátu Turtle mohl rovnat výslednému souboru u formátu N-Triples.

V případě XML nabízí program dvě varianty serializace. Jedna je zcela bez optimalizace, vždy uloží jednu trojici a uzavře koncové značky. Druhá se snaží hledat shodné prvky, aby zkrátila výsledný soubor. I přes to je tento zápis několikrát větší než ekvivalentní graf zapsaný pomocí Turtle.

5.1 N-Triples

Při serializaci N-Triples stačí brát přicházející trojice a postupně je zapisovat. Pokud byl zdroj definován jako QName, je potřeba najít správný jmenný prostor a spojit je. V případě URI N-Triples vypisuje vše. N-Triples neumožňuje žádný zápis pro seznamy a kvůli tomu se mohou některé krátké pasáže z Turtle značně roztáhnout.

Jelikož N-Triples jsou psané jen nad US-ASCII kódováním, zatímco RDF umožňuje použití libovolných unicode znaků, existují v N-Triples escape sekvence, které dávají prostředky pro zápis těchto znaků. Proto je potřeba všechny uzly, kde by se znaky mohly vyskytovat, překódovat.

5.2 Turtle

Při serializaci do formátu Turtle lze postupovat obdobně jako při zápisu N-Triples. Vznikl by dokument, který by až na použité kódování odpovídal výstupu při serializaci N-Triples. Proto jsem přistoupil k malé optimalizaci výstupu. Optimalizace se zaměřuje na prázdné uzly a opakování stejných subjektů a jejich vlastností.

V případě, že serializer dostane anonymní uzel, zkontroluje, zda měl tento uzel zadané jméno (například atributem *nodeId* v RDF/XML nebo zápisem *_:jmeno* v N–Triples a Turtle). Pokud jméno má, není možné ho zkrátit pomocí konstrukce "[" a "]". Pokud by bylo možné zaručit, že se tento uzel již nikde v dokumentu nevyskytuje, šel by nahradit i v tomto případě. Jelikož to při proudovém zpracování

nejde, je možné, že uzel stejného jména se již někde vyskytuje a jeho zkrácením by se ztratila informace o tom, že tyto uzly jsou stejné. Generované uzly naopak vznikly při parsování právě ze zkrácených uzlů a tudíž lze předpokládat, že všechny trojice související s tímto uzlem přijdou pohromadě. Proto je možné uzel zkrátit, ale zároveň je potřeba si ho zapamatovat a při příchodu dalších trojic kontrolovat, jestli už jeho predikát neskončil.

Program si pamatuje příchozí subjekty a predikáty nejen u anonymních uzlů, ale i u ostatních. Při zápisu další trojice jsou tyto uzly porovnány a na jejich základě je zapsána nová trojice, případně jen nový objekt nebo predikát za použití znaků "," a ";".

5.3 Stream serializing XML

Stejně jako při parsování XML dokumentu je třeba co nejméně zaplňovat paměť při serializaci do formátu XML. Běžně používaný model DOM by dobře fungoval pro malé dokumenty a umožňoval by použít i pokročilé způsoby zápisu jazyka RDF. Každý příchozí uzel by byl zařazen na své správné místo a mohly by vznikat rozsáhlé stromy. Zde to ale není možné použít kvůli jeho paměťové náročnosti. Z tohoto důvodu bylo třeba vytvořit jakousi obdobu SAXu pro zápis XML dokumentu. Jisté paměťové náročnosti se stejně neubráníme, protože XML používá koncové značky, takže při zanoření si musíme pamatovat, co potřebujeme uzavřít.

Jednou z možností obrany před paměťovou náročností je vytvářet každou trojici jako nový XML blok, pak není potřeba cokoliv v paměti uchovávat, protože každý otevřený element je okamžitě uzavřen. Značnou nevýhodou se stává enormní nárůst velikosti výsledného souboru. Ta se dá alespoň částečně snížit použitím podobné optimalizace jako v případě Turtle, tedy zanořování již použitých uzlů. To sice způsobí větší náročnost aplikace kvůli kontrole příchozích trojic, ale může výrazně snížit velikost výsledného souboru. Navíc vzhledem k tomu, že pevné disky patří k nejpomalejším částem počítače, může dojít ke snížení celkové doby, která je potřebná ke konverzi.

Při ukládání predikátu je potřeba dát si pozor na povolené znaky elementu. V případě predikátu, který je uložen jako URI, je potřeba oddělit lokální část a jmenný prostor elementu.

Podobně jako optimalizace zanořováním by bylo možné kontrolovat i seznamy. Množství kontrol a možností, jak a kde jsou v původním souboru seznamy zanořeny, by ale vedlo k neúměrnému zpomalení.

6 Jmenné prostory

Pro ulehčení a zkrácení zápisu lze v pokročilých formátech použít QName. Jedná se o zkrácení adresy (URI) zdroje. Adresa je zkrácena tak, že globální část se definuje kdesi v dokumentu (klíčovým slovem *@prefix* v Turtle a Notation3, případně jako klasický jmenný prostor v XML).

V rámci XML

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
…
</rdf:RDF>
```

případně v rámci Turtle nebo Notation 3

```
@prefix rdf: < http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
```

Při konverzi z formátů, které umožňují zápis zdroje jako URI (N-Triples, Turtle, Notation3), do XML, vzniká problém se zápisem URI predikát. Mějme například následující zápis trojice

```
@prefix ex: <http://example.org/>
ex:clovek <http://example.org/jmeno> "Frank" .
```

Při zápisu jako subjektu nebo objektu je vše v pořádku, jelikož se dané adresy uzlů vloží do atributu *about* elementu *Description*. Ale při podobné situaci u predikátu dochází k tomu, že jméno predikátu, který se v rámci RDF/XML zapisuje přímo jako element, není možné použít pro jméno elementu.

Většina URI nesplňuje znakové požadavky na jméno elementu jazyka XML. Je proto nutné převést URI na QName, respektive vytvořit lokální část URI, která je v pořádku. Jelikož nejsou všechny URI předem známé, nezbývá než definovat v kořenovém elementu známé jmenné prostory (tj. základní jmenné prostory pro RDF a OWL). Jakékoliv další je nutné definovat až v rámci elementu, který ho používá jako implicitní jmenný prostor.

Při zpracovávání souboru ve formátu Turtle (respektive Notation 3) sice vzniká seznam použitých prefixů, ale jelikož se deklarace prefixu může vyskytovat kdekoliv

v souboru, není možné znát všechny před začátkem serializace. Z tohoto důvodu není možné deklarovat všechny jmenné prostory v kořenovém elementu výstupního souboru.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:about="http://example.org/clovek">
      <jmeno xmlns="http://example.org/">Frank</jmeno>
</rdf:Description>
```

Tím se bohužel zvětšuje velikost výsledného souboru. Jinou možností by mohlo být prohlédnutí celého souboru a vyextrahování všech nutných jmenných prostorů, což by ale zvlášť u velkých zdrojových souboru značně prodloužilo dobu zpracování.

Při překladu URI do QName se vychází z tabulky povolených znaků. Jelikož URI se skládá ze standardních US-ASCII znaků, stačí velikost této tabulky jen 128 znaků.

7 Kódování

Při práci s RDF je kladen velký důraz na způsob, jakým jsou výsledná data ukládána. Bohužel zatím jazyk C++ neposkytuje snadnou přenositelnou možnost pro práci se znaky Unicode. Pro jejich zpracování je nutné vytvořit vlastní nástroje nebo použít nástroje stávající. V případě této práce byly použity knihovny Xerces, které, ač to není jejich primárním úkolem, poskytly dostatečné prostředky pro základní práci. Vzhledem k escape sekvencím, které umožňují formáty N-Triples, Turtle a Notation 3, bylo potřeba poupravit funkce pro načítání a ukládání v rámci těchto specifikací, aby byly tyto znaky korektně zpracovány.

7.1 Unicode

V současné době obsahuje unicode kolem 100 000 znaků. Pro jejich reprezentaci je možné použít 4-bytový typ, ale z hlediska paměťové úspory je lepší použít jen 2–bytový. Ten umožňuje uložit prvních 65536 znaků (tzv. BMP = Basic Multilingual Plane) beze změny. Zbylé znaky jsou zakódovány do dvou dvoubytových znaků. K tomuto kódování se používá tzv. surrogates pair. Jedná se o hodnoty znaků, které jsou v rámci unicode speciálně vyhrazeny pro podobné účely.

RDF povoluje práci se znakovou sadou Unicode. Specifikace Turtle a Notation 3 jsou definovány nad znakovou sadou UTF-8. Pro čtení RDF/XML jsou využité prostředky, které nabízí Xerces.

7.2 UTF-8

Specifikace Turtle a Notation 3 jsou definovány nad znakovou sadou unicode s kódováním UTF-8. Přesto nelze používat všechny znaky bez omezení. Při zápisu částí, jakými jsou například řetězce nebo adresy URI, je potřeba používat zástupné escape sekvence z N-Triples.

Formát UTF-8 umožňuje zápis Unicode znaků. Pro znaky US-ASCII nedochází k žádným změnám a znaky jsou ukládány pomocí jednoho bytu. Pro další znaky jsou využity byty 2 – 6 (Tabulka 1).

Kód znaku	Zakódované bity
U-00000000 – U-0000007F:	0*xxxxxxx*
U-00000080 – U-000007FF:	110*xxxxx* 10*xxxxxx*
U-00000800 – U-0000FFFF:	1110*xxxx* 10*xxxxxx* 10*xxxxxx*
U-00010000 – U-001FFFFF:	11110*xxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*
U-00200000 – U-03FFFFFF:	111110*xx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*
U-04000000 – U-7FFFFFFF:	1111110*x* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

Tabulka 1: Kódování UTF-8

7.3 ASCII

Pro zápis N-Triples je použito kódování 7-bitové ASCII. Pro větší znaky, případně pro řídící znaky, které nelze přímo vložit, používá N-Triples vlastní escape sekvence (Tabulka 2).

Tabulka 2: Escape sekvence pro N-Triples

(s kódem *u*)	N-Triples escape sekvence
[#x0-#x8]	\u*HHHH* 4 hexadecimální znaky *HHHH* kódují Unicode znak *u*
#x9	\t
#xA	\n
[#xB-#xC]	\u*HHHH* 4 hexadecimální znaky *HHHH* kódují Unicode znak *u*
#xD	\r
[#xE-#x1F]	\u*HHHH* 4 hexadecimální znaky *HHHH* kódují Unicode znak *u*
[#x20-#x21]	Znak *u*
#x22	\"
[#x23-#x5B]	Znak *u*
#x5C	\\
[#x5D-#x7E]	Znak *u*
[#x7F-#xFFFF]	\u*HHHH* 4 hexadecimální znaky *HHHH* kódují Unicode znak *u*
[#10000-#x10FFFF]	\U*HHHHHHHH* 8 hexadecimálních znaků *HHHH* kóduje Unicode znak *u*

8 Měření

Pro všechna následující měření byl použit počítač

- AMD Athlon XP 2000+ (1.67 GHz)
- 768 MB RAM
- Seagate Barracuda 7200.8 200826 a Seagate Barracuda 7200.7 80011

Testovací soubor ve formátu Turtle o velikosti 22 803 216 B obsahoval 1 503 200 trojic. Vznikl spojením testovacích souborů formátu Turtle [9] a jejich několikanásobným zkopírováním. Tato testovací data obsahují prakticky veškeré konstrukce pro tvorbu trojic, které formát Turtle nabízí a navíc zahrnují většinu vlastností, které se v RDF vyskytují.

Jelikož jsou ve vstupním souboru všude využívané jmenné prostory a jednotlivé prvky tak mají většinou jeden až dva znaky, je při překladu do ostatních formátů výrazně znát nárůst velikosti.

Nejprve se podíváme na překlad z Turtle do ostatních formátů. Při překladu do formátu N-Triples dochází ke kompletnímu rozvinutí zkrácených trojic. Tomu odpovídá velikost výsledného souboru. Při překladu do formátu RDF/XML bez optimalizace je tato velikost ještě navýšena. Tento nárůst je lehce zmírněn při zápisu optimalizovaného RDF/XML výstupu, který pak vychází o něco kratší než konverze do formátu N-Triples (Tabulka 3).

Výstupní formát	Čas překladu	Velikost výsledného souboru
RDF/XML optimalizovaný	62 s	189 642 011 B
RDF/XML neoptim.	104 s	334 757 468 B
N-Triples	61 s	194 569 102 B

Tabulka 3:Hodnoty překladu z Turtle

Následně byly testovány překlady z optimalizovaného formátu XML do Turtle a N–Triples. Jak je vidět, vzhledem k tomu, že N-Triples nepoužívá žádné zkracování, je výsledný soubor téměř stejně veliký a obsahově totožný s překladem z formátu

Turtle. Jediný rozdíl způsobují odlišně zapsané názvy prázdných uzlů a změna při zápisu bílých znaků.

Zápis Turtle se oproti originálu výrazně prodloužil, což je způsobeno hlavně absencí prefixů. Přesto došlo ke zkrácení oproti vstupnímu XML souboru (Tabulka 4).

Výstupní formát	Čas překladu	Velikost výsledného souboru
N-Triples	275 s	194 568 152 B
Turtle	246 s	112 474 780 B

Tabulka 4: Hodnoty překladu z RDF/XML

Při překladu z N-Triples byl opět použitý výstupní soubor z Turtle. Opět je viditelný nárůst velikosti výsledného souboru formátu Turtle kvůli absenci prefixů. Soubor je dokonce o něco větší než při serializaci z RDF/XML, což je způsobeno tím, že v rámci N-Triples jsou všechny anonymní uzly považované za generované, a tudíž nedochází ke zkracování pomocí "[" a "]". Kvůli stejnému problému dochází k zvětšení výstupních XML souborů (Tabulka 5).

Výstupní formát	Čas překladu	Velikost výsledného souboru
Turtle	109 s	112 483 637 B
RDF/XML optimalizovaný	153 s	256 197 539 B
RDF/XML neoptim.	174 s	345 286 662 B

Tabulka 5: Hodnoty překladu z N-Triples

Velikosti výsledných dat jsou oproti prvnímu vstupu enormní. V případě N-Triples se s tím nedá nic dělat, protože N-Triples neumožňují nijak zkrátit výsledné trojice. Ale v případě Turtle by bylo možné se více zaměřit na uchovávání jmenných prostorů a jejich lepší aplikaci. Sice by se tím zvětšila náročnost aplikace kvůli jejich neustálému ověřování, ale přinejmenším v tomto případě, kdy v původním souboru je zkracování pomocí prefixů velmi výrazné, by to přineslo mnohem menší výstupní soubory.

Stejně tak pokud by se v případě XML použily načtené jmenné prostory z Turtle, došlo by značnému snížení velikosti a nebylo by nutné rozkládat adresy predikátů na

lokální část a jmenný prostor, což by pravděpodobně přineslo i celkové zrychlení programu. Díky snížení velikosti souboru by bylo možné snížit výsledné časy překladu, které jsou z větší části způsobeny nutnou prací s pevným diskem.

Další možným zlepšením by bylo používání atributů při serializaci RDF trojic do formátu RDF/XML v případě literálů nebo používání atributu *rdf:parseType* v případě prázdných anonymních uzlů.

9 Závěr

Hlavním cílem této práce bylo vytvořit jednoduchý konvertor mezi některými formáty pro zápis RDF grafů. Nebylo cílem udělat program, který bude kontrolovat vstupní data. Podstatné bylo navrhnout a implementovat aplikaci, která by byla dostatečně rychlá a zároveň byla schopná zpracovávat všechny konstrukce daných formátů.

Výsledkem je program, který to umí. Umožňuje zpracování souborů ve formátu N-Triples, Turtle, Notation 3 nebo RDF/XML a jejich přeložení do formátů N-Triples, Turtle nebo RDF/XML. Navíc se v některých případech snaží výstupní soubor pomocí vlastností jednotlivých formátů zkrátit.

Zároveň jsme se v průběhu práce seznámili se samotným konceptem datového modelu pro ukládání informací RDF. Zaměřili jsme se na některé formáty pro ukládání RDF trojic a popsali jejich vlastnosti. Prošli jsme jednotlivé konstrukce a jejich zpracování při načítání souboru a zároveň jejich využití při vytváření výstupu. Také jsme zběžně prošli používaná kódování a znakové sady.

Přesto by asi bylo možné zkracování výstupu více rozvinout. Program v současnosti využívá jen ty nejzákladnější konstrukce pro zkracování. Je otázkou, zda by další rozšíření nevedlo k drastickému snížení rychlosti aplikace, ale bezesporu by bylo vhodným rozšířením zpracovávání jmenných prostorů a jejich plné používání při serializaci. Výrazně by to mohlo v některých situacích zmenšit velikost výstupního souboru při celkem slušném zachování rychlosti. Další možné rozšíření by mohlo být zahrnutí vytváření seznamů z načítaných dat. Během vytváření výstupu ve formátu RDF/XML by bylo možné více využít možností XML. Například širší využití atributů u XML elementů.

10 Zdroje

[1] RDF – Seznam dostupných dokumentů
http://www.w3.org/RDF/

[2] N-Triples specifikace
http://www.w3.org/TR/rdf-testcases/#ntriples

[3] Turtle specifikace
http://www.dajobe.org/2004/01/Turtle/

[4] Notation 3 specifikace
http://www.w3.org/DesignIssues/Notation3.html

[5] Notation 3 gramatika
http://www.w3.org/2000/10/swap/grammar/n3rdf-report.html

[6] RDF/XML specifikace
http://www.w3.org/TR/rdf-syntax-grammar/

[7] RDF 3986 STD 66 - URI: Generic Syntax
http://www.ietf.org/rfc/rfc3986.txt

[8] Knihovny XERCES
http://xml.apache.org/xerces-c/

[9] Testovací soubory formátu Turtle
http://www.dajobe.org/2004/01/turtle/tests/