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

Bakalářská práce

Nativní vícevláknová aplikace pro zpracování a transformaci XML dokumentů

Plzeň 2012

Kateřina Nová

Prohlášení

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

V Plzni dne 8. května 2012

Kateřina Nová

Abstract

The thesis, Native multithreading application for processing and transformation of XML documents, aims at developing a multithread application, which transforms data from an XML document into a Prolog document. Application normalizes the XML document and gives it a strictly defined structure. Subsequently, it can be transformed into the Prolog language. The resulting code and a set of universal rules make it possible to get answers to user enquiries by using a set of logical operations and obtain information which were not obvious before.

Obsah

1 Úvod **1**

2 XML **2**
2.1 Syntaxe XML 2
2.2 XML parsery 4
2.2.1 Událostmi řízené zpracování 4
2.2.2 Stromová reprezentace XML 5

3 Převod XML do Prologu **6**
3.1 Prolog 6
3.2 Algoritmus převodu 6

4 Vícevláknové aplikace **9**
4.1 Synchronizace vláken 9
4.2 Vlákna v Javě 10
4.3 Modely vícevláknových aplikací 11

5 Programátorská dokumentace **14**
5.1 Zadání 14
5.2 Popis existujících aplikací 14
5.3 Analýza a návrh 16
5.3.1 Grafické uživatelské rozhraní 16
5.3.2 Správce úloh 19
5.3.3 Zpracování a transformace 20
5.4 Implementace 24
5.4.1 Prezentační vrstva 24
5.4.2 Aplikační vrstva 27
5.4.3 Datová vrstva 28
5.5 Testování 29

6 Závěr **33**

1 Úvod

Jazyk XML[1] je jednoduchý formát pro ukládání textových dat, které obsahují strukturované informace. Původně byl navržený pro řešení problémů s publikováním velkých elektronických dokumentů, stále významnější roli hraje také při výměně nejrůznějších údajů na webu i jinde [1]. Výhodou jazyka XML je jeho přenositelnost (obsahuje pouze textová data v kódování definovaném v hlavičce) a také to, že značky definují význam uložených dat. Jazyk XML je jedním ze základních nástrojů sémantického webu. To je způsob uložení dat na internetu v takovém formátu, aby ke každé informaci byl jasně definován i její význam a bylo možné tyto informace strojově zpracovávat.

Pomocí jazyka XPath[2] nebo XQuery[3] je možné vyhodnocovat dotazy nad XML dokumentem. Pokud se data obsažená v XML dokumentu převedou do faktů jazyka Prolog, je možné pomocí logických pravidel vyvozovat i informace, které v původním XML dokumentu nebyly explicitně uvedeny[10].

Tento projekt navazuje na dvě bakalářské práce z minulých let. Bakalářská práce Františka Schneidera [6] se zabývá úpravou XML dokumentu do normalizovaného tvaru, aby bylo možné výsledný dokument převést do podoby logického jazyka. Bakalářská práce Lukáše Gemely [2] se zabývá převodem takto upraveného XML dokumentu do množiny faktů jazyka Prolog. Úkolem tohoto projektu je spojit funkcionalitu obou prací do jedné aplikace, vytvořit přívětivé grafické uživatelské rozhraní a vytvořit správce úloh, který bude spouštět transformaci dokumentů v zadaném počtu vláken.

[1] eXtensible Markup Language, česky rozšiřitelný značkovací jazyk

[2] XPath - XML Path language, tedy jazyk pracující s cestou ve stromové struktuře XML dokumentu

[3] XQuery - XML Query language, XML dotazovací jazyk, je rozšířením jazyka XPath[8]

2 XML

V této kapitole bude popsána syntaxe jazyka XML a možnosti zpracování XML dokumentů pomocí knihovních nástrojů.

2.1 Syntaxe XML

XML dokument se skládá z elementů. Element tvoří počáteční značka, obsah elementu a ukončovací značka. Obsah elementu může tvořit text nebo další elementy. Pokud obsahuje text i elementy, nazývá se tento obsah smíšený. Element také může mít atributy. Na pořadí, ve kterém jsou atributy uvedeny, nezáleží, ale nesmějí se opakovat jejich jména. Syntaxe elementu je zobrazena ve výpisu 2.1.

Výpis 2.1: Element

```
<elementName attributeName="attributeValue">
    content of the element
</elementName>
```

Ve výpisu 2.2 je vidět jednoduchý XML dokument. Na řádce 1 je XML deklarace neboli hlavička. V ní je definována verze XML a použité kódování. Každý XML dokument by měl hlavičku obsahovat. Pokud není uvedena, je kódování nastaveno na UTF-8 a verze na 1.0. Máme-li tedy XML dokument v jiné verzi nebo v jiném kódování, je nutné hlavičku uvést. Element `herbal` (řádek 2-7) tvoří kořenový element, všechny ostatní elementy jsou v něm vnořené. XML dokument musí obsahovat právě jeden kořenový element. Na řádkách 3 až 6 je element `herb`, který má atribut se jménem `id` a hodnotou `28`. Na řádce 5 je vidět zkrácený zápis prázdného elementu - místo `<medicinal use="tea"></medicinal>` je použita značka `<medicinal use="tea"/>`.

Výpis 2.2: Jednoduchý XML dokument

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <herbal>
3   <herb id="28">
4     <name>Thymus serpyllum</name>
5     <medicinal use="tea"/>
6   </herb>
```

```
7 </herbal>
```

XML dokument může obsahovat jmenné prostory. Nejprve je nutné deklarovat použitou sadu značek a jejich označení v dokumentu. Každá sada značek je jednoznačně definována URI[1] adresou. Při deklaraci se každé sadě určí prefix, kterým se bude označovat její použití[2].

Deklarace jmenného prostoru se provádí pomocí speciálního atributu `xmlns`. Prefix se poté používá před každým názvem značky. Příklad dokumentu, který jmenné prostory používá, je uveden ve výpisu 2.3. Na řádce 2 se definuje, že prefix `ns` bude označovat použití sady značek, která je identifikována URI adresou `http://myWeb.cz/xml/ns`. Jména elementů se poté zapisují ve tvaru `prefix:elementName`. Dokument může využívat více jmenných prostorů, jejich prefix musí být jedinečný.

Výpis 2.3: Jednoduchý XML dokument

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ns:herbal xmlns:ns="http://myWeb.cz/xml/ns">
3   <ns:herb id="28">
4     <ns:name>Thymus serpyllum</ns:name>
5     <ns:medicinal use="tea"/>
6   </ns:herb>
7 </ns:herbal>
```

Správná strukturovanost XML dokumentu

Dokument je správně strukturovaný (well-formed), pokud se neprohřešuje proti následujícím pravidlům:

- obsahuje pouze znaky, které spadají do kódování uvedeného v hlavičce dokumentů (pokud není uvedena tak do UTF-8)
- textový obsah elementů neobsahuje speciální znaky jako je ‘&’ a ‘<’

[1]URI - Uniform Resource Identifier, česky Jednotný identifikátor zdroje, jednoznačně identifikuje nějaký informační zdroj. Podmnožinou URI je URL - Uniform Resource Locator, česky Jednotný lokátor zdroje. URL jednoznačně určuje umístění zdroje v počítačové síti.

- značky počátečního, ukončovacího a prázdného elementu jsou správně vnořeny, žádná značka nechybí, nepřebývá ani se nekříží
- jméno počáteční a ukončovací značky elementu si přesně odpovídají (jazyk XML je Case-sensitive[2])
- existuje právě jeden kořenový element

2.2 XML parsery

Parser je knihovní program, který umí načítat XML dokumenty. Každý parser automaticky kontroluje, zda je dokument well-formed. Dále může kontrolovat validitu dokumentu oproti DTD[3] schématu nebo souboru XSD[4]. Názvy elementů, jejich hodnoty a atributy načítá a přes API[5] poskytuje jejich abstraktní model.

Existují dva základní přístupy zpracování XML dokumentu, událostmi řízené zpracování a práce se stromovou reprezentací dokumentu.

2.2.1 Událostmi řízené zpracování

Událostmi řízené zpracování se někdy také nazývá jako proudové čtení. XML dokument se čte sekvenčně a pro jednotlivé části dokumentu vyvolává události, na které může program reagovat. Proudové čtení je rychlé a nenáročné na paměť. Rozlišují se dva druhy parserů, push parsery a pull parsery. Push parsery vygenerují najednou všechny události, čtení nelze zastavit a nelze dynamicky měnit parametry zpracování. Příkladem je rozhraní SAX[6]. Pull

[2] case-sensitive - tento výraz by se dal přeložit jako „citlivý na velikost písmen“. Pokud jazyk je case-sensitive, jsou „ahoj“ a „aHoJ“ naprosto rozdílné řetězce, pokud není case-sensitive, program tyto řetězce vyhodnotí jako shodné.

[3] DTD - Document Type Definition, česky Definice typu dokumentu - schémový jazyk, který popisuje strukturu XML dokumentů a určuje, jaké značky se mohou v XML dokumentu použít.

[4] XSD - XML Schema Definition, česky Definice XML schématu - nástupce DTD, kromě struktury XML dokumentu definuje i hodnotu dat. Dnes se jeho použití preferuje před použitím DTD.

[5] API - Application Programming Interface, česky programátorské rozhraní

[6] SAX - Simple API for XML, česky Jednoduché rozhraní pro XML

parsery čekají na pokyn programu, že mají vygenerovat následující událost, jsou tedy flexibilnější. Příkladem je rozhraní StAX[7].

StAX

StAX je rozhraní pro proudové zpracování XML dokumentu. StAX parser je typu pull, čtení dokumentu tedy probíhá až na žádost programu. Poskytuje dva způsoby práce s XML dokumentem. První je událostní. Tento způsob je vysokoúrovňový, využívá rozhraní `XMLEventReader` a `XMLEventWriter`. Druhý způsob je kurzorový. Ten je rychlý, ale nízkoúrovňový. Je představován rozhraními `XMLStreamReader` a `XMLStreamWriter`[3]. Pro velké dokumenty je nejvhodnější použít právě kurzorové rozhraní StAX.

2.2.2 Stromová reprezentace XML

Parsery načtou celý dokument do paměti, vytvoří jeho stromovou reprezentaci. Programátor má přes API přístup k celému dokumentu najednou, může se v dokumentu vracet, číst vícekrát, měnit elementy a změněná data opět uložit do XML dokumentu. Nad dokumentem ve stromové reprezentaci je možné se dotazovat pomocí jazyka XPath. Nevýhodou je, že je tento přístup náročný na paměť, protože musí být celý dokument najednou načten do paměti. Pro XML dokumenty o velikosti několik stovek MB je tedy prakticky nepoužitelný. Příkladem rozhraní, které pracují se stromovou reprezentací dokumentu jsou DOM[8], XOM[9], DOM4J[10] a JAXB[11].

[7] StAX - Streaming API for XML, neboli rozhraní pro proudové zpracování XML

[8] DOM - Document Object Model, česky objektový model dokumentu

[9] XOM - XML Object Model, česky XML objektový model

[10] DOM4J - Document Object Model for JAVA, česky objektový model dokumentu pro Javu

[11] JAXB - Java Architecture for XML Binding

3 Převod XML do Prologu

Pro vyhledání požadované informace v XML dokumentu se používá jazyk XPath. Vyhodnotit dotaz nad XML dokumentem s využitím logických pravidel nám dovolí použití logického programovacího jazyka. Tato kapitola se zabývá algoritmem převodu XML dokumentu do faktů jazyka Prolog.

3.1 Prolog

Prolog je nejrozšířenější představitel logických programovacích jazyků. Program obsahuje klauzule - fakta a pravidla, která definují vztahy mezi objekty. Nad těmito klauzulemi pak vznášíme dotazy. Přesný algoritmus, kterým program dojde k výsledku, je dán konkrétní implementací a programátorovi zůstává skryt.

3.2 Algoritmus převodu

Algoritmem převodu XML dokumentu do fakt Prologu se zabývá Martin Zíma v [9] a [10]. Definuje množinu pravidel, která jsou platná pro každý XML dokument. Dále popisuje způsob transformace dat z XML dokumentu do množiny faktů Prologu, tento algoritmus zde bude popsán.

Předpokládejme, že vstupní dokument neobsahuje smíšený obsah a na každé řádce je pouze jedna značka nebo kombinace počáteční značka, textový obsah elementu a koncová značka. Postup bude ukázán na příkladě XML dokumentu zobrazeného ve výpisu 3.1.

Výpis 3.1: Jednoduchý XML dokument

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <herbal>
3   <herb id="28">
4     <name>Thymus serpyllum</name>
5   </herb>
6   <herb id="53">
7     <name>Salvia officinalis</name>
```

```
8     </herb>
9   </herbal>
```

Struktura XML dokumentu se definuje fakty, která mají obecnou strukturu:

```
xml(radek, nazev, poradi, zanoreni).
```

Pro každý element je vygenerován jeden fakt, který určuje jeho pozici v XML dokumentu. Argument `radek` určuje číslo řádku, na kterém se nachází začátek elementu, `nazev` určuje jméno elementu, `poradi` definuje pořadí elementu v aktuální úrovni zanoření a `zanoreni` určuje aktuální úroveň zanoření. Například pro element na 7. řádku XML dokumentu z výpisu 3.1 bude vygenerován fakt:

```
xml(7, ‘name’, 1, 3).
```

Pro každý vnořený element (tj. element, který není kořenový) jsou vygenerována další fakta, která definují cestu od daného elementu ke kořenovému elementu ve stromové struktuře XML dokumentu. Pro každý nadřazený element bude vygenerován jeden fakt podle obecného předpisu. Atribut `radek` bude označovat číslo řádku, na kterém začíná zpracovávaný element, atribut `nazev` určuje název nadřazeného elementu, `poradi` určuje pořadí nadřazeného elementu a `zanoreni` určuje úroveň zanoření nadřazeného elementu. Pro element na 7. řádku v ukázkovém příkladu budou vygenerována fakta:

```
xml(7, ‘herb’, 2, 2).
xml(7, ‘herbal’, 1, 1).
```

Tímto způsobem je popsána struktura dokumentu, ale nejsou zachycena samotná data, která XML dokument obsahuje. Pro každý textový obsah elementu je vygenerován fakt:

```
data(radek, nazev_elementu, obsah_elementu).
```

V našem případě bude pro data na 7. řádku vygenerovaný fakt:

```
data(7, ‘name’, ‘Salvia officinalis’).
```

Pro každý atribut elementu je vytvořen fakt:

```
atribut(radek, nazev_elementu, nazev_atributu, hodnota_atributu).
```

Pro atribut `id` na 3. řádce ukázkového XML dokumentu bude vygenerovaný fakt:

```
atribut(3, ‘herb’, ‘id’, ‘28’).
```

Celý XML dokument uvedený ve výpisu 3.1 bude po převodu do Prologu obsahovat tato fakta:

```
xml(2, ‘herbal’, 1, 1).
xml(3, ‘herb’, 1, 2).
xml(3, ‘herbal’, 1, 1).
atribut(3, ‘herb’, ‘id’, ‘28’).
xml(4, ‘name’, 1, 3).
xml(4, ‘herb’, 1, 2).
xml(4, ‘herbal’, 1, 1).
data(4, ‘name’, ‘Thymus serpyllum’).
xml(6, ‘herb’, 2, 2).
xml(6, ‘herbal’, 1, 1).
atribut(6, ‘herb’, ‘id’, ‘53’).
xml(7, ‘name’, 1, 3).
xml(7, ‘herb’, 2, 2).
xml(7, ‘herbal’, 1, 1).
data(7, ‘name’, ‘Salvia officinalis’).
```

4 Vícevláknové aplikace

Tato kapitola se věnuje vláknům a jejich synchronizaci, především v programovacím jazyku Java. Dále zde jsou popsány modely vícevláknových aplikací.

Vlákno je posloupnost instrukcí, kterou lze spustit souběžně s ostatními vlákny. Vlákna je možné pustit na více procesorech (popř. více jádrech jednoho procesoru). Pokud má aplikace k dispozici pouze jeden procesor, zabezpečí se souběžné zpracování pomocí dělení času (tzv. pseudoparalelní běh). V rámci jednoho procesu může běžet více vláken.

Zpracování ve více vláknech se hodí použít, pokud aplikace vykonává nějakou výpočetně náročnou operaci, která není závislá na ostatní činnosti programu. Další využití je v programech, kde očekáváme nějakou asynchronní událost (například vstup od uživatele), je vhodné jedno vlákno vyhradit na obsluhu této události (komunikaci s uživatelem).

4.1 Synchronizace vláken

Vlákna v rámci jednoho procesu sdílí adresní prostor. Je nutné ošetřit souběžný přístup k těmto datům a to ať už vlákna běží paralelně nebo pseudoparalelně. Část programu, ve které je prováděna manipulace se společnými daty, se nazývá kritická sekce. Úkolem synchronizace je zajistit, aby se v kritické sekci nacházelo vždy pouze jedno vlákno. Dále také synchronizace může sloužit ke komunikaci mezi vlákny.

Existují tři základní synchronizační primitiva: zámek, semafor a monitor. Jsou vzájemně ekvivalentní, pomocí každého z nich lze implementovat ostatní.

Zámek

Zámek je reprezentován booleovskou proměnnou. Pokud je zámek k dispozici (má hodnotu true), je možné ho získat (nastavit na false) a vstoupit do kritické sekce. Po jejím ukončení vlákno zámek uvolní (nastaví na true). Pokud chce vlákno získat zámek, který je držen jiným vláknem (má hodnotu

false), vzdá se času procesoru a zkusí zámek získat jindy[1]. Uvolněním zdrojů umožní vláknu, který zámek drží, dokončit kritickou sekci.

Semafor

Semafor obsahuje celočíselný čítač a frontu vláken, která jsou nad semaforem uspána. Na začátku se nastaví hodnota čítače, která určuje, kolik vláken může vykonávat nějakou instrukci (obvykle označuje počet sdílených zdrojů). Nad semaforem jsou definované dvě operace: `wait()` a `signal()`. Operace `wait()` zkontroluje, zda nemá čítač hodnotu nula. Pokud ano, uspí se, pokud má čítač jinou hodnotu, sníží ji o jedna a vstoupí do semaforu. Operace `signal()` zvýší hodnotu čítače o jedna a pokud má hodnotu jedna (tj. před zvýšením měl hodnotu nula), probudí vlákno z fronty uspaných vláken.

Monitor

Monitor je synchronizační primitivum, které poskytuje větší míru abstrakce než předchozí primitiva. Lze si ho představit jako objekt, který má definovaná data a metody, které s nimi pracují. Vlákna mohou volat metody monitoru, ale nemohou přímo přistupovat k datům. Uvnitř monitoru může být aktivní pouze jedno vlákno, to může libovolně provádět metody. Monitor může mít definované podmínky. Pokud není podmínka splněná, vlákno se pozastaví, a reaktivuje se, až když jiné vlákno způsobí splnění této podmínky. Výhodou monitoru je, že automaticky řeší vzájemné vyloučení a je odolnější proti chybám programátora.

4.2 Vlákna v Javě

V programovacím jazyce Java jsou vlákna reprezentována objekty třídy *Thread*. Vytvoření vlákna, které vykonává námi požadované instrukce, je možné dvěma

[1] Některé implementace zámků se času procesoru nevzdávají a v cyklu zkoušejí znovu zámek získat. To je vhodné, pouze pokud se na zámek čeká jen krátkou dobu, například v jádře operačního systému. Použití v uživatelských aplikacích je nevhodné. Jiné zámky jsou implementovány jako binární semafor. Pokud tedy zámek není k dispozici, vlákno se uspí a vzbudí až po uvolnění zámku.

způsoby. První možnost je vytvořit třídu jako potomka třídy *Thread*, zavoláním konstruktoru této třídy je pak vytvořeno nové vlákno. Druhá možnost je vytvořit třídu, která implementuje rozhraní *Runnable*. Instanci této třídy pak předáme v konstruktoru třídy *Thread* jako parametr. V obou těchto případech je nutné implementovat metodu `run()` a do ní napsat kód, který má vlákno vykonávat. Spuštění vlákna se provádí zavoláním metody `start()`[2] nad objektem typu *Thread*.

Synchronizace

Nejjednodušší způsob, jak v Javě zajistit synchronizaci vláken, je použití synchronizovaných metod. Stačí před definici metod, které pracují se sdílenými daty, napsat klíčové slovo `synchronized`. Každý objekt v Javě má se sebou svázaný monitor, vstupem do synchronizované metody získá vlákno zámek monitoru. Tím se zajistí exklusivní přístup k datům, každé vlákno, které se pokusí zámek získat (tedy vstoupit do monitoru), bude zablokováno. Když vlákno opustí synchronizovanou metodu, uvolní zámek a probudí některé z vláken, které je nad zámkem uspané.

Tento postup je jednoduchý, pro programátora příjemný, ale nedokáže vyřešit některé složitější úlohy a pokud vlákna často přistupují ke sdíleným datům, není příliš výkonný (zamkne se celý objekt). Všechny tyto problémy řeší knihovna `java.util.concurrent`, která je součástí Javy od verze 1.5. Obsahuje implementaci zámků, semaforů, monitorů, atomických proměnných i kolekcí uzpůsobených pro práci ve více vláknech.

4.3 Modely vícevláknových aplikací

Modely vícevláknových aplikací popisují strategie rozdělování práce jednotlivým vláknům. Zde jsou uvedeny nejčastější z nich[4]:

[2]Metoda `start()` zahájí vykonávání metody `run()` v novém vlákně.

Delegation

Model delegation (česky pověření) se někdy také označuje termíny farmer-workers (farmář-dělníci) nebo boss-worker (šéf-dělník). Jedno vlákno je řídící (farmář, šéf) a přiděluje práci ostatním vláknům (dělníkům). Poté čeká, až práci některé z vláken dokončí a přidělí mu další úlohu. Pokud už byly zpracovány všechny úlohy, vlákno dělníka se ukončí a po dokončení všech vláken dělníků se ukončí i řídící vlákno. Vlákna dělníků vykonávají přidělenou práci a po jejím dokončení odevzdají řídícímu vláknu výsledky a řeknou si o další práci.

Peer-to-Peer

V modelu peer-to-peer (rovný-s-rovným) vykonávají všechna vlákna činnost pracovníků. Oproti modelu delagation zde neexistuje žádné řídící vlákno. Jedno vlákno sice vytvoří a spustí ostatní vlákna, ale dále nijak nezasahuje do jejich činnosti a samo je považováno za obyčejné vlákno, které zpracovává úkoly jako každé jiné. Vlákna mohou zpracovávat požadavky z jednoho proudu nebo může mít každé vlákno svůj proud vstupních dat, za které je zodpovědné. Vstupní data také mohou být uložena v databázi, souboru nebo frontě. Vlákna mohou navzájem komunikovat a sdílet zdroje.

Pipeline

Model pipeline (potrubí) si lze představit jako montážní linku. Každá vstupní jednotka je zpracovávaná v několika fázích, zpracování je kompletní až po absolvování všech fází. Každou fázi zpracování představuje jedno vlákno. To je zodpovědné za průběžné výsledky a jejich předání další fázi zpracování. Pokud některá fáze trvá déle než ostatní, jsou průběžné výsledky předchozí fáze zpracování ukládány do vyrovnávací paměti. Není ale vhodné, aby se na nějakém místě práce hromadila. Rychlost zpracování vstupů je ovlivněna nejpomalejší fází. Pokud tedy některá etapa zpracování trvá příliš dlouho, je vhodné ji rozdělit do více fází nebo dané fázi zpracování přidělit více vláken.

Producer-consumer

V modelu producer-consumer (producent-konzument) je vlákno, které data produkuje (vytváří) a vlákno, které data konzumuje (zpracovává). Vyprodukovaná data jsou uložena ve vyrovnávací paměti (bufferu), ke které má přístup vlákno producenta i vlákno konzumenta. Nejprve musí producent data vytvořit a potom je může konzument zpracovávat. Pokud producent produkuje data rychleji, než je konzument stíhá zpracovávat, musí po naplnění paměti producent počkat, až konzument data zpracuje. Pokud konzument zpracovává data rychleji, než je producent dokáže produkovat, musí po vyprázdnění vyrovnávací paměti konzument počkat, až budou k dispozici nějaká data ke zpracování. Model se někdy nazývá také client-server (klient-server). Lze ho zobecnit i pro více producentů i konzumentů.

5 Programátorská dokumentace

Programátorská dokumentace obsahuje analýzu, návrh a popis implementace aplikace pro převod XML dokumentů do faktů jazyka Prolog, na konci kapitoly jsou uvedeny výsledky testování. Výslednou aplikaci je možno volně používat, šířit a modifikovat.

5.1 Zadání

Úkolem praktické části této bakalářské práce je vytvořit uživatelsky přívětivý program, který zpracuje vstupní XML dokumenty a převede je do podoby logického programu. Aplikace má dva režimy, interaktivní a dávkový. Při volbě interaktivního režimu se výsledný dokument ihned zobrazí v GUI[1]. Pokud je vybrán dávkový režim, zpracování vstupních dokumentů proběhne v zadaném počtu vláken. Výsledné soubory budou uložené na disku, zobrazí se pouze výsledky převodu.

5.2 Popis existujících aplikací

Úprava vstupního XML dokumentu se řeší v knihovně pro normalizaci XML, která je popsána v [6], jeho následnou transformací do Prologu se zabývá [2]. Pro účely tohoto projektu byly obě práce poskytnuty včetně všech zdrojových kódů a testovacích dat. Zde bude stručně popsána jejich funkcionalita.

Knihovna pro normalizaci XML

Knihovna pro normalizaci XML dokumentů je součástí projektu [6] a zpracovává XML dokument dle následujících požadavků:

- na jednom řádku bude pouze jedna značka nebo kombinace počáteční značka - data - uzavírací značka

[1] GUI - Graphical User Interface, česky grafické uživatelské rozhraní.

- XML dokument nebude obsahovat smíšený obsah - každý element tedy obsahuje buď pouze další elementy nebo pouze text
- bude se jednat o syntakticky správný XML dokument (v sadě testovacích souborů jsou také fragmenty XML, k těm bude přidán kořenový element)

V aplikaci jsou implementovány čtyři parsery: DOM, XOM, DOM4J a StAX. K nim je vytvořeno jednotné rozhraní *IParser*, které umožňuje přistupovat ke všem implementovaným metodám stejně. Při úpravě XML dokumentu se vždy používá jeden konkrétní parser, který si zvolí uživatel. Pro velmi rozsáhlé XML dokumenty (v řádu stovek MB) je použitelný pouze parser StAX, který data zpracovává proudově. Dokáže tedy zpracovat libovolně velké soubory.

XMLtoProlog

Knihovna XMLtoProlog je součástí projektu [2]. Obsahuje potřebné třídy a metody pro převod XML dokumentu do Prologu. Vstupem je normalizovaný XML dokument, který má pevně danou strukturu a lze ho tedy načítat po řádcích a zpracovávat pomocí regulárních výrazů. Algoritmem převodu XML dokumentu do Prologu se zabývá práce [9]. Výstupem je textový soubor s množinou faktů jazyka Prolog, které popisují jak strukturu XML, tak i obsažená data.

XMLogic

Aplikace XMLogic je také součástí [2]. Logicky spojuje obě výše uvedené knihovny a obsahuje také správce úloh, který zajišťuje vykonávání požadovaných transformací XML dokumentů v definovaném počtu vláken v dávkovém režimu. Dále také obsahuje grafické uživatelské rozhraní, které umožňuje zpracování vstupních souborů ve dvou režimech - interaktivním a dávkovém.

5.3 Analýza a návrh

Analýza částečně vychází z hotových aplikací popsaných v předchozí kapitole programů. Výsledný program obsahuje kromě funkcionality těchto prací také správce úloh, který řídí zpracování vstupních dokumentů v dávkovém režimu ve vláknech a příjemné uživatelské rozhraní. Navíc sjednocuje přístup ke zpracování XML dokumentů, který je v těchto aplikacích rozdílný.

5.3.1 Grafické uživatelské rozhraní

Hlavní okno GUI bude mít čtyři části: menu, panel nastavení, panel transformace a log.

V menu budou dostupné volby pro nastavení konfigurace programu, jazyku aplikace, režimu zpracování a pro práci s levým oknem panelu transformace a s panelem logu.

Panel nastavení bude obsahovat základní volby pro nastavení převodu: zda se má dokument normalizovat a zda se má převést do Prologu. Pokud bude vybrána normalizace XML dokumentu, zobrazí se její podrobnější nastavení: velikost odsazení a jméno obalového elementu. Pokud bude vybrána transformace do Prologu, objeví se ještě volba řadit predikáty. Dále bude panel obsahovat přepínač režimu zpracování.

V panelu log se budou objevovat informace o výsledcích zpracování. Tyto informace bude možné pomocí tlačítek smazat nebo uložit.

Panel transformace bude obsahovat dvě textová okna, uprostřed bude tlačítko pro spuštění transformace. Pod textovými okny se budou nacházet tlačítka pro práci s obsahem těchto oken. Dále bude k dispozici tlačítko pro zrušení převodu a indikátor průběhu převodu.

Chování panelu transformace se bude lišit podle aktuálního režimu zpracování. Během interaktivního režimu bude možné napsat dokument určený k transformaci přímo do levého textového okna. Pokud už dokument máme připravený, je možné ho načíst ze souboru. Tento dokument můžeme dále upravovat a můžeme jeho obsah uložit do souboru. Po stisku prostředního tlačítka se provede transformace dle nastavení a výsledný dokument se zobrazí v pravém textovém okně. Obsah tohoto okna lze také uložit. Indikátor

průběhu převodu bude v interaktivním režimu skryt. Během dávkového režimu bude povoleno vybírat k transformaci více souborů. V levém okně se zobrazí pouze jména těchto souborů včetně absolutní cesty. Po stisknutí prostředního tlačítka se spustí transformace a indikátor průběhu bude ukazovat, kolik souborů je již zpracováno a jaký je celkový počet souborů určených ke zpracování. V pravém okně se budou objevovat jména souborů (včetně absolutní cesty), které byly transformací vytvořeny. Po dobu zpracování budou nedostupné komponenty, jejichž použití by mohlo převod souborů narušit.

Okno konfigurace bude obsahovat podrobné nastavení normalizace a transformace XML dokumentů a dále nastavení pro dávkový režim: počet vláken, ve kterých bude převod probíhat a adresář, do kterého se budou ukládat výsledky transformace. Po stisku tlačítka OK bude toto nastavení uloženo jako výchozí.

Uložení konfigurace

Aby mohl uživatel nastavit výchozí konfiguraci programu (v tomto případě nastavení transformace a dávkového režimu), je nutné toto nastavení uložit.

Jednou z možností je uložit konfiguraci do textového (případně XML) souboru. Výhodou je, že tento konfigurační soubor je pro člověka srozumitelný a po otevření se tedy může dozvědět, jaké je výchozí nastavení, aniž by program spustil. Nevýhodou tohoto přístupu je, že ukládání a hlavně zpětné načítání souboru musí zajistit programátor a může být i relativně komplikované.

Další možnost je využít nástroj programovacího jazyka pro serializaci[2] objektů. Všechno nastavení se tedy uloží do zvláštního objektu, provede se jeho serializace a uložení. Tímto postupem vznikne binární soubor, který má menší velikost než textový soubor, ale není pro člověka srozumitelný. Výhoda tohoto přístupu spočívá především ve snadné implementaci. Programátor pouze využije nástrojů jazyka pro serializaci i deserializaci objektu.

Úspora paměti je v tomto případě zanedbatelná. Jelikož v tomto případě není důvod zjišťovat konfiguraci programu bez jeho spuštění, využijeme druhý způsob uložení, tedy nástroje pro serializaci objektu.

[2]serializace = ukládání objektů do formátu, který může být uložen nebo poslán po síti, deserializace = vytvoření objektu ze serializovaných dat.[4]

Vybrání souborů ve složce

V dávkovém režimu bude umožněno vybrat ke zpracování celou složku a program vybere všechny soubory uvnitř této složky. Otázkou je, zda má vybrat i všechny soubory z případných podsložek. Z hlediska uživatele to je žádoucí. Může mít připravenou složku k dávkovému zpracování, která je dále strukturovaná například podle typu souborů či podle data vytvoření. Pokud uživatel vybere tuto složku ke zpracování, bylo by vhodné, aby výstupní soubory byly umístěny ve složce, která odpovídá původní struktuře.

Pokud by vstupní složka se svými podsložkami tvořila strom, nebyl by problém rekurzivně projít všechny složky, vybrat všechny soubory uvnitř a pamatovat si relativní cestu ke každému ze souborů. Problém však nastává, pokud ve složce existuje cyklus, například některá ze složek obsahuje symbolický odkaz[3] na sobě nadřazenou složku. V tomto případě se totiž načítání souborů zacyklí.

Pokud si budeme pamatovat, které složky jsme již procházeli, bude možné zacyklení zabránit. Nestačí ale kontrolovat, zda absolutní cesta složky, kterou chceme prohledat, již není v seznamu prošlých složek. Pokud jsou použity symbolické odkazy, jeden soubor má více absolutních cest. K řešení tohoto problému využijeme kanonický tvar absolutní cesty[4]. Při pokusu prohledat podruhé složku s danou kanonickou cestou rekurzi zastavíme.

Kromě symbolických odkazů existují také pevné odkazy[5]. Pevné odkazy lze vytvořit pouze na soubor, ne na adresář. Proto nemůže vzniknout problém zacyklení adresářové struktury.

Pokud je v adresářové struktuře cyklus vytvořen pomocí zástupců[6] složek, nemůže se načítání zacyklit, protože zástupce je obyčejný binární soubor s příponou `.lnk`. Pokud bychom ale chtěli vybrat i soubory ze složky, na kte-

[3]Symbolický odkaz na složku - virtuální složka, po vstupu do složky můžeme dále procházet adresářovou strukturu. Jeden soubor tedy může mít více absolutních cest. Používá se v Unixových systémech a v operačním systému Windows od Windows Vista

[4]Kanonický tvar absolutní cesty - absolutní cesta, která jednoznačně definuje cestu k danému souboru.

[5]Pevný odkaz - vytváří druhé plnohodnotné jméno souboru. Změna původního souboru či v jeho pevného odkazu budou vždy viditelné z obou umístění. Pevné odkazy lze stejně jako symbolické odkazy vytvořit v Unixových systémech a v systému Windows od verze Windows Vista

[6]Zástupce souboru - soubor s informacemi odkazujícími na jiný soubor. Používá se v operačním systému Windows.

rou zástupce odkazuje, nastane problém. Cestu k odkazovanému adresáři lze zjistit z binárního souboru. Jeho obsah se ale může lišit s každou verzí systému Windows. Aby nebyla ohrožena přenositelnost aplikace, raději nebudeme brát zástupce v potaz. Je velmi nepravděpodobné, že by uživatel připravoval data k dávkovému zpracování vytvářením zástupců souborů.

5.3.2 Správce úloh

Jak již bylo řečeno, transformace se bude moci spustit ve dvou režimech, v interaktivním a dávkovém. V obou režimech bude správce vytvářet a spouštět úlohy. Úlohy budou dvojího typu, jedny budou mít na starost normalizaci dokumentu a druhé převod XML do Prologu.

Interaktivní režim

Pokud spustíme transformaci v interaktivním režimu, uloží se na disk dočasný soubor, do kterého se uloží obsah levého editačního okna. Poté se dle nastavení vytvoří úloha normalizace nebo transformace (nebo obě dvě) a ty se spustí. Výsledný dočasný soubor se přečte a jeho obsah zobrazí v pravém editačním okně. Zpracování bude probíhat pouze v jednom vlákně.

Dávkový režim

Dávkové zpracování souborů je vhodné spustit ve více vláknech[7]. Správce úloh bude mít na starosti vytvoření zadaného počtu vláken a vytváření a přidělování úloh jednotlivým vláknům.

Správce úloh bude vytvořen podle modelu Farmer-Worker. Správce (farmář) bude jednotlivým vláknům (dělníkům) přidělovat úkoly. Vlákna po dokončení úlohy vrátí správci úloh výsledky a požádají o další práci.

Správce bude udržovat seznam úloh a dvě množiny souborů - soubory určené ke zpracování a již zpracované soubory. Na začátku vytvoří zadaný počet vláken. Pro každý soubor určený ke zpracování se vytvoří jedna úloha

[7]Výsledky výkonových testů aplikace XMLogic ukázaly, že spuštění dávkového zpracování ve více vláknech vede ke značnému vylepšení času běhu programu. Přitom nelepší efekt má použití tolika vláken, kolik jader má procesor.

a vloží se do seznamu úloh. Každému vláknu se přidělí jedna úloha ke zpracování. Když nějaké vlákno úlohu dokončí, zpracuje výsledek převodu - pokud se jednalo o poslední úlohu vztahující se k jednomu souboru, přidá soubor do seznamu již zpracovaných souborů, pokud byla dokončena normalizační úloha a má se provést ještě transformace, vytvoří tuto úlohu a vloží ji do seznamu. Pokud není seznam úloh prázdný, přidělí vláknu další úlohu.

Pro seznam úloh využijeme datovou strukturu zásobník. Pokud je nastavena buď pouze normalizace dokumentů nebo pouze jejich transformace do Prologu, nezáleží na tom, v jakém pořadí se úlohy vykonají. Pokud je ale nastavena normalizace i transformace dokumentů, je žádoucí, aby program co nejdříve dokončil zpracování celého souboru a potom teprve začal zpracovávat další soubory. Kdybychom úlohy vkládali do fronty, nejprve by se provedla normalizace všech dokumentů a potom teprve jejich transformace (transformační úlohy by se vkládaly na konec fronty). Při velké dávce souborů by pak aplikace zdánlivě nevykazovala žádnou činnost. Možným řešením je vytvořit dvě fronty úloh, pro každý typ úlohy jednu. Správce by vláknům zadával normalizační úlohy, pouze pokud by fronta transformačních úloh byla prázdná. Stejnou funkci bude mít použití datové struktury zásobník - normalizační úlohy se vytváří na začátku, budou tedy u dna zásobníku, nově vytvořené transformační úlohy budou vloženy na vrchol zásobníku a tedy se také dříve zpracují. Výhoda zásobníku oproti dvěma frontám je ta, že není nutné zkoumat, jakého typu úloha je, ke všem lze přistupovat stejně.

5.3.3 Zpracování a transformace

Datová vrstva se bude starat o načítání a zpracování XML dokumentů.

Přístup ke zpracování XML dokumentu se v knihovně pro normalizaci XML a v knihovně XMLtoProlog zásadně liší. Knihovna pro normalizaci XML využívá knihovní parsery pro zpracování XML (celkem je na výběr ze čtyř implementovaných). Nepracuje tedy s XML dokumentem jako s obyčejným textovým souborem, ale využívá toho, že knihovní parsery se při čtení starají o syntaktickou analýzu XML dokumentu a programátor pak už pracuje s jeho abstraktním modelem.

Knihovna XMLtoProlog využívá toho, že na vstupu je již předzpracovaný XML dokument, který má pevně definovanou strukturu a je možné ho tedy načítat po řádcích a zpracovávat ho syntaktickou rekurzivní analýzou pomocí regulárních výrazů.

Otázkou tedy je, jak ke zpracování XML souborů přistoupit. Normalizace XML dokumentů bude určitě i nadále využívat knihovní parsery. Zpracovávat obecný XML dokument pomocí regulárních výrazů by bylo nejenom pracné, ale hlavně málo efektivní. Jsou tedy dvě možnosti: buď ponecháme stávající rozdílné přístupy ke zpracování XML dokumentů nebo přístup sjednotíme a změníme tedy od základu knihovnu pro převod XML do Prologu, místo regulárních výrazů zde taktéž využijeme knihovní parsery.

Jako lepší varianta se jeví sjednocení přístupu, využijeme tedy knihovní parsery i pro transformaci XML dokumentu do Prologu. Tento přístup má několik výhod. Výsledná aplikace bude působit ucelenějším dojmem, implementace bude snazší a hlavně zpracování by mělo být efektivnější[8].

Normalizace XML dokumentů

Normalizace XML dokumentů bude vycházet z [1], podrobné informace lze tedy nalézt tam. Protože ale bude nutné normalizaci lehce upravit (především přidat odsazování elementů podle úrovně zanoření), bude zde popsáno zpracování dokumentů parserem StAX, který bude v aplikaci využit.

StAX parser načítá XML dokument sekvenčně a nemá ho tedy celý najednou uložený v paměti. To znamená, že jím lze zpracovat libovolně velké XML dokumenty, ale také to, že se v dokumentu nelze vracet a všechny informace, které budeme pro transformaci potřebovat si musíme za běhu uložit. Aby byla paměťová náročnost programu co nejnižší, budou potřebné informace ukládány do souboru a poté z něj opět sekvenčně načítány.

Normalizace probíhá ve třech fázích. První fázi se provádí pouze u dokumentů s příponou `.dump`. Tyto dokumenty jsou pouze XML fragmenty, neobsahují kořenový element. V této fázi se z nich vytvoří plnohodnotný XML dokument, který má hlavičku a kořenový element.

Ve druhé fázi se dokument transformuje tak, aby na každé řádce byla pouze počáteční značka nebo pouze ukončovací značka nebo kombinace počáteční značka, textový obsah elementu a uzavírací značka. Pokud dokument neobsahoval smíšený obsah, je již na konci této fáze správně formátovaný. Po-

[8] Z knihy Java a XML ([3], str. 117): „Nejhorší (tzn. nejméně efektivní), co můžeme s XML dokumentem při zpracování provádět, je považovat ho za textový soubor a napsat vlastní program, který jednotlivé elementy atributy a jejich hodnoty postupně získává analýzou načtených textových řádek."

kud se v dokumentu vyskytoval smíšený obsah, nepodařilo se všechen textový obsah uzavřít mezi počáteční a ukončovací značku na jedné řádce.

Ve třetí fázi se provádí detekce a odstranění smíšeného obsahu. Detekce smíšeného obsahu je snadná, smíšený obsah je na řádce, která nezačíná znakem ‘<’ nebo nekončí znakem ‘>’. K souboru tedy v této fázi stačí přistupovat jako k obyčejnému textovému dokumentu a načítat ho po řádkách. Smíšený obsah uzavřeme do obalovacího elementu. Na konci této fáze je dokument normalizovaný.

Takto je tedy v knihovně pro normalizaci XML vyřešena normalizace. Aby byl normalizovaný dokument lépe čitelný, bylo by vhodné realizovat odsazování XML elementů podle hloubky zanoření. Pokud dokument neobsahuje smíšený obsah, je realizace velmi snadná. Pouze vytvoříme čítač, který se inkrementuje pokaždé, když na vstup přijde počáteční značka a dekrementuje, když přijde ukončovací značka. Na začátek každé řádky se vytiskne tolik mezer, jaká je aktuální hodnota čítače. Problém nastává, když element obsahuje smíšený obsah. Pokud tento postup například aplikujeme na XML dokument zobrazený ve výpisu 5.1, bude po druhé fázi transformace vytvořen soubor, jehož obsah je zobrazen ve výpisu 5.2.

Výpis 5.1: XML dokument se smíšeným obsahem

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <mixedContent>
3   <first>this is <b>mixed</b> content</first>
4   <second><b>this content</b> is <b>mixed</b></second>
5 </mixedContent>
```

Výpis 5.2: XML dokument se smíšeným obsahem po druhé fázi normalizace

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <mixedContent>
3   <first>this is
4    <b>mixed</b>
5  content</first>
6   <second>
7    <b>this content</b>
8  is
9    <b>mixed</b>
10  </second>
11 </mixedContent>
```

Je vidět, že jsou celkem tři možnosti, jak se smíšený obsah projeví. Ve všech případech se volný text uzavře do obalovacího elementu a odsadí se dle následujících pravidel:

- řádek nekončí znakem '>' (viz řádek č. 3): obalovací element se přesune na další řádek a odsadí se o jednu mezeru víc, než byl odsazen původní řádek.
- řádek nezačíná znakem '<' (viz řádek č. 5): obalovací element se odsadí se o stejný počet mezer jako je odsazena předchozí značka, zbylá značka se přesune na další řádek a odsadí se o jednu mezeru méně než předchozí.
- řádek nezačíná znakem '<' a nekončí znakem '>' (viz řádek č. 8): obalovací element se odsadí o stejně mezer, jako předchozí značka.

Obsah výsledného souboru vytvořeného ve třetí fázi normalizace je vidět ve výpisu 5.3.

Výpis 5.3: XML dokument se smíšeným obsahem po třetí fázi transformace

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <mixedContent>
3   <first>
4    <wrap>this is</wrap>
5    <b>mixed</b>
6    <wrap>content</wrap>
7   </first>
8   <second>
9    <b>this content</b>
10   <wrap>is</wrap>
11   <b>mixed</b>
12  </second>
13 </mixedContent>
```

Transformace do Prologu

Jak již bylo řečeno, transformace do Prologu se bude provádět pomocí knihovních parserů, aby se sjednotil přístup ke zpracování XML dokumentů. V knihovně pro normalizaci XML jsou implementovány čtyři parsery (DOM,

XOM, DOM4J a StAX). Vždy se ale vybere pouze jeden, který se pro transformaci použije, ostatní jsou implementovány pouze pro srovnání. Stačí tedy vybrat jeden z nich, který bude použit i pro transformaci XML dokumentu do Prologu. StAX parser je jako jediný z knihovních parserů univerzální a lze ho použít pro libovolně velké XML soubory, proto bude implementován.

Samotný algoritmus převodu XML do Prologu bude vycházet z aplikace XMLtoProlog[6], jiným způsobem se budou rozpoznávat XML značky. Převezme se především zásobník, který zajišťuje uchovávání struktury XML dokumentů.

5.4 Implementace

Aplikace je implementována v jazyce Java[9] podle pravidel třívrstvé architektury. Prezentační vrstva je tvořena balíkem `presentation`, aplikační vrstva balíkem `taskManager` a datová vrstva balíky `inputOutput` a `data`. Balík `exceptions` obsahuje vlastní výjimky.

Aplikace se spouští z třídy *Main*, která pouze vytvoří novou instanci třídy *MainWindow* z balíku `presentation`.

5.4.1 Prezentační vrstva

Prezentační vrstvu tvoří balík `presentation`, který obsahuje třídy pro zobrazení grafického uživatelského rozhraní a komunikaci s uživatelem. V uživatelském rozhraní jsou použity grafické komponenty z knihovny Swing[10].

Třída MainWindow reprezentuje hlavní okno aplikace. Uživatel zde může nastavovat parametry zpracování XML dokumentu, volit mezi interaktivním a dávkovým režimem, vkládat soubory ke zpracování a spouštět transformaci. Aktuální nastavení transformace se nikam explicitně neukládá, pamatují si ho grafické prvky. Až po stisku tlačítka pro zobrazení okna konfigurace nebo po stisku tlačítka pro zahájení převodu se nastavení uloží do objektu

[9]Jazyk Java byl vybrán především proto, že v něm byli implementovány aplikace, na které se v tomto projektu navazuje.

[10]Swing - knihovna pro tvorbu grafického uživatelského rozhraní, je součástí standardních knihoven Javy od verze 1.2.

SettingsManager. Ten slouží ke vzájemné komunikaci mezi hlavním oknem a oknem konfigurace a při spuštění tlačítka pro převod dokumentů se předává k dalšímu zpracování.

Třída *SettingsWindow* reprezentuje okno konfigurace. V tomto okně může uživatel nastavit výchozí hodnoty parametrů zpracování a také parametry dávkového režimu. Toto nastavení se po stisku tlačítka OK opět uloží do objektu třídy *SettingsManager* a předá se zpět hlavnímu oknu, kde se nové nastavení parametrů ihned projeví. Navíc se objekt serializuje a uloží. Při dalším spuštění aplikace se tento objekt znovu načte a výchozí parametry transformace se podle něj nastaví.

Při stisku tlačítka pro spuštění převodu se aktualizuje objekt *SettingsManager*, který uchovává parametry transformace. V dávkovém režimu se vytvoří instance třídy *TaskManager* z balíku `taskManager` a v konstruktoru se mu předá objekt uchovávající nastavení spolu se seznamem souborů určených ke zpracování. V interaktivním režimu se vytvoří nový objekt třídy *InteractiveTransformation* také z balíku `taskManager`, kterému se v konstruktoru předá spolu s textovým obsahem levého editačního okna.

Dostupnost GUI během zpracování

Aby během zpracování souborů (ať už v interaktivním nebo dávkovém režimu) zůstalo GUI aktivní, je nutné ho spustit v novém vlákně. Třída *TaskManager* je zděděna od třídy *Thread*, třída *InteractiveTransformation* implementuje rozhraní *Runnable*. Obě tedy mají definovanou metodu `run()`, která se vykonává v novém vlákně.

Uživatelské rozhraní je tedy během zpracování dostupné, uživatel může například tisknout tlačítka nebo se posuvníkem pohybovat v editačních oknech. Je nutné ale omezit dostupnost některých funkcí, jejichž použití je v dané chvíli nežádoucí a mohlo by narušit průběh zpracování. Po ukončení zpracování jsou všechny funkce opět zpřístupněny.

Zobrazování údajů během zpracování

Během zpracovávání souborů se mění okno s výpisy o průběhu zpracování (logy), pravé editační okno a v dávkovém režimu také indikátor průběhu zpracování (progress bar). Aby se tyto změny v daný čas projevily, je použit

návrhový vzor Pozorovatel[11]. Logy se ukládají do objektu třidy *MyLogger*, údaje o zpracovaných souborech při dávkovém zpracování se ukládají do instance třídy *FinishedFiles*. Při každé změně těchto objektů se zavolají metody `setChanged()` a `notifyObservers()`, které upozorní příslušné komponenty GUI na změnu.

Změna jazyku aplikace

V hlavním okně v menu Nastavení - Jazyk lze přepnout jazyk aplikace. Na disku jsou uložené soubory ve tvaru `název_jazyk_země.properties`, ve kterých jsou uloženy veškeré textové informace, které se v GUI zobrazují. Třída *ResourceBundle* ze standardní knihovny Javy s těmito soubory umí pracovat a snadno se tak načtou dané řetězce podle aktuálně nastaveného národního prostředí. Při stisku tlačítka pro změnu jazyka se tedy pouze změní národní prostředí reprezentované třídou *Locale* ze standardních knihoven a poté se zavolá metoda pro aktualizaci názvů tlačítek a ostatních textů. Čtení proběhne ze správného souboru. V aplikaci je na výběr pouze čeština a angličtina, ale díky této konstrukci je možné velice snadno přidat i další jazyky.

Načítání souborů v dávkovém režimu

V kapitole 5.3.1 byl popsán způsob načítání všech souborů ve složce i jejích podsložkách. Tento způsob je implementován, pro zjištění absolutní cesty souboru v kanonickém tvaru se používá funkce `getCanonicalPath()`, která se volá nad objektem třídy *File*. Celý proces načítání jmen souborů zajišťuje třída *DirLoader* z balíku `data`. Soubor a jeho relativní cesta z vybrané složky se ukládá do objektu třídy *FileToProcess*. Díky tomu se po zpracování souborů vytvoří ve výstupní složce stejná adresářová struktura, jaká byla ve vstupní složce.

[11]Pozorovatel, anglicky Observer - návrhový vzor, zavádí vztah mezi objekty(pozorovateli) reagujícími na změnu stavu (pozorovaného) objektu. Pozorované objekty své pozorovatele upozorňují na změny[5]. V jazyce Java se při implementaci používá třída *Observable* a rozhraní *Observer*.

5.4.2 Aplikační vrstva

Do aplikační vrstvy patří balík `taskManager`, který reprezentuje správce úloh.

Správce úloh byl implementován dle návrhu v kapitole 5.3.2. Úlohu normalizace XML dokumentu představuje třída *NormalizeTask*, úlohu převodu dokumentu do Prologu představuje třída *TransformTask*. Obě tyto třídy jsou zděděné od abstraktní třídy *Task*. Objekty reprezentující úlohy mají za úkol spuštění zpracování souboru a případné zachycení výjimek. Vrací číslo, které reprezentuje výsledek zpracování, příslušné konstanty jsou definované ve třídě *Results*.

V interaktivním režimu se používá třída *InteractiveTransformation*. Ta ze vstupního řetězce vytvoří dočasný soubor, vytvoří příslušné úlohy a spustí je. Zpracování probíhá v jednom vlákně.

V dávkovém režimu se používá objekt třídy *TaskManager* a objekty třídy *Worker*. TaskManager je vždy jen jeden. Vytvoří tolik instancí třídy *Worker*, v kolika vláknech bude zpracování probíhat. Potom vytvoří úlohy, pro každý soubor ke zpracování jednu a každému vláknu přidělí jednu úlohu. Metodou `run()` spustí v každém objektu *Worker* zpracování v novém vlákně. Pak už pouze čeká na dokončení vláken. Objekt Worker na pokyn spustí zpracování zadané úlohy. Když se zpracování dokončí, vrátí výsledek zavoláním synchronizované metody `reportResults(FileToProcess ftp, int result)`, kde parametr `ftp` určuje soubor, který byl zpracováván a parametr `result` výsledek zpracování. TaskManager tento výsledek zaznamená do objektu *MyLogger* z balíku tools a soubor přesune do seznamu zpracovaných souborů nebo pro něj vytvoří další úlohu. Poté *Worker* požádá o další úlohu zavoláním synchronizované metody `getNextTask()`. Pokud se mu vrátí hodnota *null*, *Worker* se ukončí, jinak pokračuje ve zpracování. Poté, co se dokončí všechny vlákna, zaznamená se celkový výsledek do objektu *MyLogger* a vlákno *TaskManager* se ukončí.

Během dávkového zpracování je možné v GUI stisknout tlačítko pro zrušení transformace. Pro implementaci této funkce bylo nutné přidat synchronizovanou metodu `cancel()` do třídy *TaskManager* i dalších tříd, které jsou v této třídě vytvářeny a probíhá v nich zpracování souborů. V každém cyklu je pak testováno, zda není převod zrušen.

5.4.3 Datová vrstva

Datová vrstva obsahuje balík `inputOutput`, který zajišťuje čtení a zápis do souborů a balík `data`, který obsahuje třídy pro normalizaci XML dokumentů a jejich transformaci do Prologu. Jak vyplývá z analýzy, pro načítání XML dokumentu se používá parser StAX.

Normalizace XML dokumentu

Při normalizaci se nejprve zkontroluje, zda jméno dokumentu nemá příponu `.dump`. Pokud ano, jedná se pravděpodobně o fragment XML dokumentu a tomuto dokumentu bude přidána hlavička a kořenový element, aby se jednalo o regulérní XML dokument. Tato oprava se provádí ve třídě *DumpFixer*.

Dále se soubor zpracovává ve třídě *StAXParser*. XML dokument se načítá parserem StAX a provádí se druhá fáze transformace popsaná v kapitole 5.2. XML se zapisuje v takovém formátu, aby na jednom řádku byla buď pouze počáteční nebo pouze ukončovací značka nebo kombinace počáteční značka - textový obsah elementu - ukončovací značka. Zároveň se řádky odsazují podle úrovně zanoření.

Při třetí fázi transformace se dokument načítá jako obyčejný soubor a provádí se uzavření textové části smíšeného obsahu do obalovacího elementu. To zajišťuje třída *ContentManager*. Zároveň se zde opět řeší odsazování (algoritmus odsazení obalovacích elementů je popsán v analýze). Protože uživatel může nastavit, o kolik mezer budou řádky odsazeny, a ve druhé fázi se řádky odsadily jen o jednu mezeru, v této fázi se spočte počet mezer na začátku řádky a tento počet se vynásobí velikostí odsazení zadanou uživatelem. Výsledkem je počet mezer, které se vytisknou na začátek řádky při výstupu.

Transformace XML do Prologu

Transformace do Prologu probíhá ve více úrovních. Třída *Preprocessor* rozpoznává XML elementy, předává je třídě *Parser*, která uchovává strukturu elementů a vytváří predikáty Prologu a ty předává třídě *PostProcessor*, která se stará o jejich zápis do souboru.

Preprocessor načítá vstupní XML dokument parserem StAX. Pokud na-

razí na počáteční značku, vytvoří objekt třídy *ElementStart*, kterému přiřadí seznam objektů třídy *Attribute*, které představují atributy elementu, a předá ho objektu *Parser*. Pokud narazí na textová data, vytvoří objekt *ElementData* s informacemi o obsahu elementu. Při nalezení ukončovací značky se nevytváří žádný objekt, parseru se pouze předá jméno ukončovací značky. Třídy *ElementStart* a *ElementData* jsou potomky třídy *Element*.

Třída *Parser* zpracovává nalezené elementy. Počáteční element vloží do speciálně upraveného zásobníku *XMLStack* (podrobný popis viz [6]). V zásobníku nalezne údaje o úrovni zanoření elementu, o pořadí v aktuální úrovni a o jménech nadřazených elementů. Poté vygeneruje predikáty `xml`, které popisují strukturu XML dokumentu. Pokud má počáteční značka uvedené atributy, vygenerují se ještě predikáty `atribut`.

Z datového elementu se vygeneruje predikát `data`, který kromě textového obsahu zahrnuje také jméno elementu, ke kterému se data vztahují.

Když *Parser* přijme jméno ukončovací značky, vyjme z vrcholu zásobníku počáteční značku a zkontroluje, jestli jejich jména odpovídají. Pokud ne, obsahoval vstupní dokument chyby, nastane výjimka a zpracování dokumentu se přeruší.

Predikáty vytvořené třídou *Parser* se nezapisují rovnou do souboru, ale předávají se třídě *PostProcessor*. Tato třída řeší řazení a zápis predikátů. Pokud se predikáty mají řadit, vytvoří kromě výstupního souboru ještě dva dočasné soubory a každý typ predikátu (`xml`, `data`, `atribut`) ukládá do jednoho souboru. Nakonec obsah dvou dočasných souborů připíše do výstupního souboru a predikáty jsou tak seřazeny.

5.5 Testování

Výsledky testování aplikace byly srovnávány s výstupy aplikací, ze kterých tato práce vycházela (podrobný popis viz kapitola 5.2). Použití aplikace vytvořené v [2] k transformaci XML dokumentů do Prologu bude dále značeno jako {T}. Použití této aplikace k normalizaci XML dokumentů pomocí knihovny vytvořené v [6] bude dále značeno jako {N}. Výsledky zpracování dokumentů aplikací vyvinutou v rámci této bakalářské práce budou dále značeny {P}.

Testy funkčnosti

Správná funkčnost aplikace byla testována na sadě krátkých dokumentů, každý z nich obsahoval určité specifikum, které by při transformaci mohlo činit problémy. Výsledné soubory bylo možné zkontrolovat „ručně“. Výsledky zpracování velkých souborů byly s {T} a {N} srovnávány pomocí programu *KDiff3*[12], který porovnává dva textové dokumenty a vyznačuje rozdíly mezi nimi.

Výsledky normalizace odpovídaly výsledkům dosažených v {N}, navíc bylo implementováno odsazování elementů dle úrovně zanoření. Výsledky transformace do Prologu odpovídají výsledkům {T}, jediným rozdílem je převod elementu, který obsahuje definici jmenného prostoru. Výsledný dokument {T} v názvu atributu neobsahuje prefix `xmlns`, který označuje, že bude definován jmenný prostor. Při definici implicitního jmenného prostoru[13] byl název atributu 'null'. Tento nedostatek byl odstraněn.

Výkonnostní testy

Na sadě osmi XML fragmentů (soubory s příponou `.dump`) byla testována i výkonnost aplikace. Testování probíhalo na dvou různých sestavách:

A) Stolní počítač s procesorem Intel Core i7 s frekvencí 3,07 GHz, 4 jádra, 8 vláken, operační paměť 12 GB. Operační systém Windows 7.

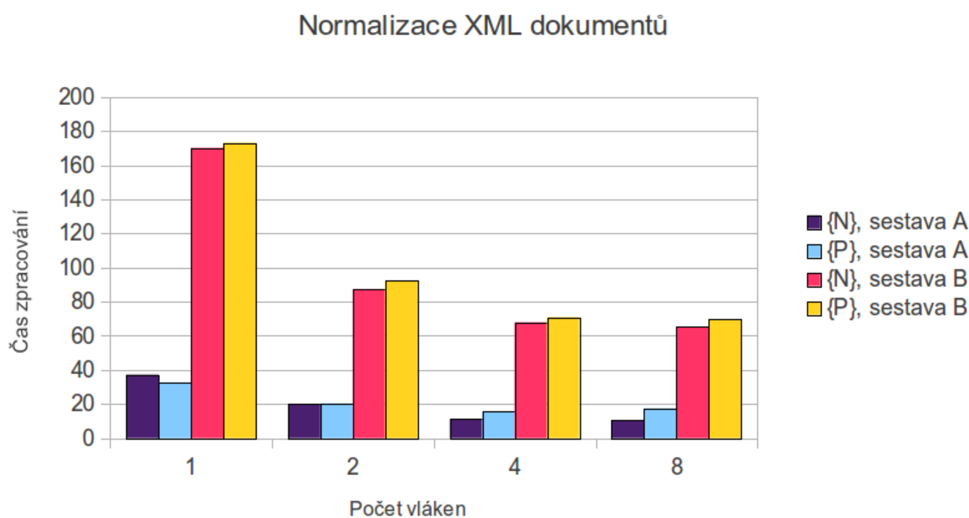
B) Notebook HP Mini, procesor Intel Atom N550 s frekvencí 1,5 GHz, 2 jádra, 4 vlákna, operační paměť 2 GB. Operační systém Ubuntu 11.04.

Normalizace

Normalizace XML dokumentů s využitím parseru StAX byla převzata z [6]. Přidána byla implementace odsazení dokumentů. Jak je vidět na grafu 5.1, výsledky {P} a {N} se od sebe příliš neliší (rozdíly jsou v řádu jednotek procent). Horší výsledek {P} je způsoben právě implementací odsazení elementů.

[12] http://kdiff3.sourceforge.net

[13] Implicitní jmenný prostor - jmenný prostor, který se neoznačuje žádným prefixem. Platí tedy pro všechna jména elementů, která nemají žádný prefix uvedený, nevztahuje na atributy.



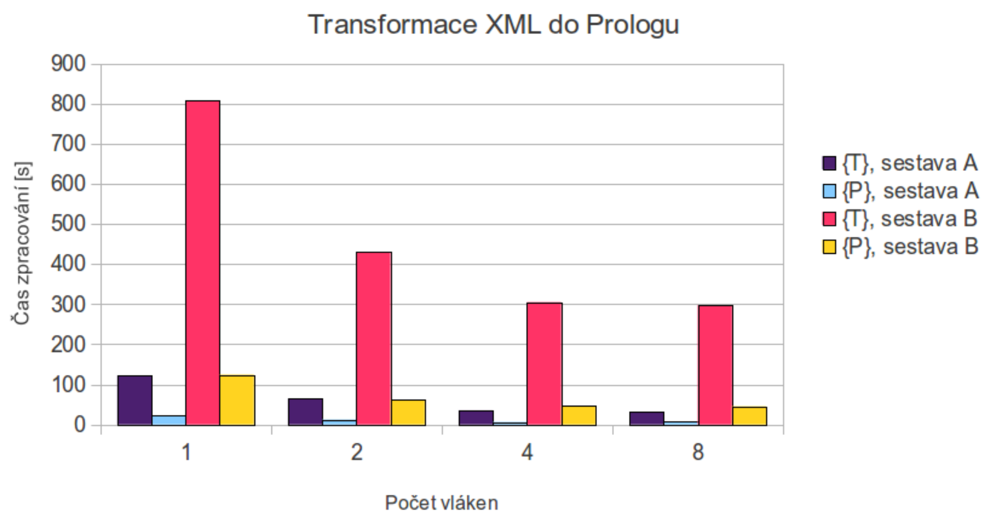
Obrázek 5.1: Výsledky měření rychlosti normalizace

Transformace do Prologu

Transformace byla řešena s využitím parseru StAX, v [2] bylo rozpoznávání elementů implementováno pomocí regulárních výrazů. Předpoklad byl, že by se využitím knihovního parseru měla transformace do Prologu zefektivnit. Jak je vidět na grafu 5.2, byl tento předpoklad správný, transformace probíhá asi 6 krát rychleji.

Vliv paralelismu

Na obrázcích 5.1 a 5.2 je vidět, že zpracování souborů ve více vláknech má velký vliv na rychlost zpracování. Při použití dvou vláken místo jednoho se ve všech případech měření snížila doba zpracování téměř na polovinu. Při použití čtyřech vláken se zpracování opět zrychlilo. U sestavy A, která má procesor se čtyřmi jádry, bylo zlepšení znovu téměř o polovinu. U sestavy B nebylo zrychlení tak výrazné. Při spuštění převodu v osmi vláknech doba zpracování zůstávala téměř stejná, v některých případech lehce klesla, v některých se lehce zvýšila. Ideální počet spuštěných vláken je rovný počtu jader procesoru.



Obrázek 5.2: Výsledky měření rychlosti transformace

Při použití více vláken již nedojde k lepšímu využití procesoru a začne se zvyšovat režie operačního systému.

6 Závěr

Tato bakalářská práce se zabývá vývojem vícevláknové aplikace pro zpracování XML dokumentů a jejich převod do Prologu. V teoretické části jsou objasněny postupy zpracování XML dokumentů pomocí knihovních nástrojů dostupných v jazyce Java, je zde popsán algoritmus transformace XML dokumentů do faktů jazyka Prolog a různé možnosti koncepce a implementace vícevláknové aplikace.

V praktické části byla aplikace implementována. Dokáže odstranit ze vstupního dokumentu smíšený obsah a formátovat dokument do speciálního normalizovaného tvaru. Z něj poté dokáže vygenerovat fakta jazyka Prolog, která popisují strukturu i data obsažená v XML dokumentu. Spolu s množinou univerzálních pravidel umožňují vyhodnocovat dotazy s využitím prostředků, které logické programování poskytuje. V dokumentaci byla podrobně popsána analýza i implementace programu a jsou zde rozebrány i zajímavé problémy, které se během vývoje aplikace vyskytly.

Všechny body zadání byly splněny, práci považuji za úspěšnou. Podařilo se vytvořit sofistikovanou správu vláken a intuitivní uživatelsky přívětivé ovládání interaktivního i dávkového režimu aplikace. V dávkovém režimu je možné zpracovávat i extrémně velké XML dokumenty (největší testovaný dokument měl velikost cca 700 MB). Navíc se podařilo výrazně zefektivnit transformaci XML dokumentů do Prologu. Aplikace je přínosem pro výzkum sémantického webu.

Přehled zkratek

API – Application Programming Interface
DOM – Document Object Model
DOM4J – Document Object Model for JAVA
DTD – Document Type Definition
GUI – Graphical User Interface
JAXB – Java Architecture for XML Binding
SAX – Simple API for XML
StAX – Streaming API for XML
URI – Uniform Resource Identifier
URL – Uniform Resource Locator
XML – eXtensible Markup Language
XOM – XML Object Model
XPath – XML Path language
XQuery – XML Query language
XSD – XML Schema Definition

Literatura

[1] *Extensible Markup Language* [online]. April 2011. Dostupné z: `http://www.w3.org/XML/`.

[2] GEMELA, L. *Transformace XML dokumentu do podoby logického programu.* Bakalářská práce, Západočeská univerzita, 2011.

[3] HEROUT, P. *Java a XML.* České Budějovice : KOOP, 2007. ISBN 978-80-7232-307-4.

[4] HUGHES, C. *Parallel and Distributed Programming Using C++.* Boston, : Addison-Wesley, 2004. ISBN 0-13-101376-9.

[5] PECINOVSKÝ, R. *Java - Programujeme profesionálně.* Brno : Computer Press, 2007. ISBN 978-80-251-1582-4.

[6] SCHNEIDER, F. *Zpracování XML dokumentů pro potřeby sémantického webu.* Bakalářská práce, Západočeská univerzita, 2010.

[7] SPELL, B. *Java – Programujeme profesionálně.* Brno : Computer Press, 2002. ISBN 80-7226-667-5.

[8] *XQuery 1.0: An XML Query Language* [online]. January 2011. Dostupné z: `http://www.w3.org/TR/xquery/`.

[9] ZÍMA, M. *Transformace XML dokumentu do podoby logického programu.* Technical report, Katedra informatiky a výpočetní techniky, Fakulta aplikovaných věd, Západočeská univerzita. DCSE/TR-2009-13.

[10] ZÍMA, M., JEŽEK, K. *Translation of XML Documents into Logic Programs.* Proceedings of the 14th International Conference on Electronic Publishing, June 2010, pp. 351-362, Helsinki, Finland, ISBN 978-952-232-086-5

Uživatelská příručka

Pro spuštění aplikace je nutné mít nainstalovanou Javu verze 1.6 nebo vyšší. Program se spouští poklepáním na soubor `XMLtoProlog.jar` nebo z konzole příkazem:

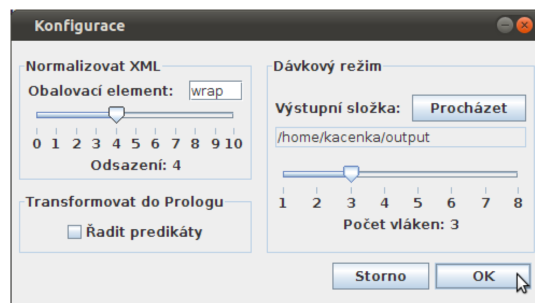
```
java -jar XMLtoProlog.jar
```

Po spuštění se zobrazí hlavní okno grafického uživatelského rozhraní (viz obrázek 2).

Nastavení zpracování

V horní části okna se nastavují parametry transformace. Je zde možné vybrat, jestli má proběhnout pouze normalizace XML dokumentů, pouze jejich transformace do Prologu nebo obojí. Když se některá z těchto voleb zaškrtne, zobrazí se její podrobnější nastavení. U normalizace lze nastavit název obalovacího elementu a počet mezer, o které se budou odsazovat vnořené elementy. U transformace do Prologu lze zaškrtnout, zda se mají vygenerovaná fakta řadit podle typu nebo ne. V pravém horním rohu lze přepínat mezi interaktivním a dávkovým režimem.

Další nastavení lze provádět v menu *Nastavení*. Zde lze opět přepínat mezi interaktivním a dávkovým režimem (*Nastavení* → *Režim*), měnit jazyk aplikace (Nastavení → Jazyk). Po stisku volby *Nastavení* → *Konfigurace* se otevře okno konfigurace (viz obrázek 1). V tomto okně lze nastavovat volby normalizace a transformace jako v hlavním okně aplikace. Dále jsou zde volby pro dávkový režim zpracování. Lze zde vybrat adresář, do kterého se budou ukládat výstupní soubory a počet vláken, ve kterých bude dávkové zpracování probíhat. Stiskem tlačítka OK se toto nastavení uloží jako výchozí.



Obrázek 1: Okno konfigurace

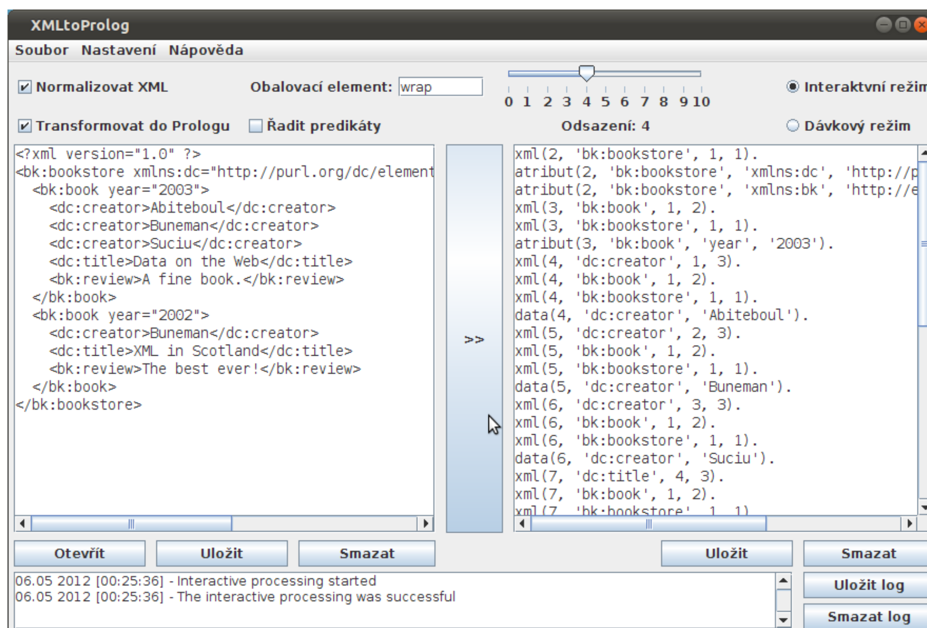
Logy

Ve spodní části hlavního okna je panel, kde se zobrazují logy neboli výpisy o průběhu převodu dokumentů. Tyto výpisy lze uložit stiskem tlačítka *Uložit log* nebo volbou *Soubor* → *Uložit log*. Smazání logů se provádí stiskem tlačítka *Uložit log* nebo volbou v menu *Soubor* → *Smazat log*.

Interaktivní režim

Na obrázku 2 je vidět hlavní okno aplikace v interaktivním režimu. V levém okně je vidět XML dokument. Ten lze do okna přímo vepsat nebo ho lze načíst z disku. Po stisku tlačítka *Otevřít* nebo volby *Soubor* → *Otevřít soubor* se objeví dialog pro výběr souboru, který se má načíst (maximální velikost XML dokumentu, který lze načít v interaktivním režimu je 150 kB). Tento dokument lze dále editovat a lze uložit na disk stisknutím tlačítka *Uložit* nebo volby *Soubor* → *Uložit*. Stiskem tlačítka *Smazat* se vymaže obsah tohoto okna.

Stisknutím prostředního tlačítka se symbolem '»' nebo volbou *Soubor* → *Zahájit zpracování* se spustí transformace dokumentu dle nastavených parametrů. Po skončení transformace se v okně logu se objeví informace o tom, zda transformace proběhla úspěšně. Pokud transformace byla neúspěšná, zůstane pravé okno prázdné. Pokud byla úspěšná, zobrazí se výsledný dokument v pravém okně. Obsah pravého okna lze uložit nebo smazat pomocí tlačítek umístěných pod ním.

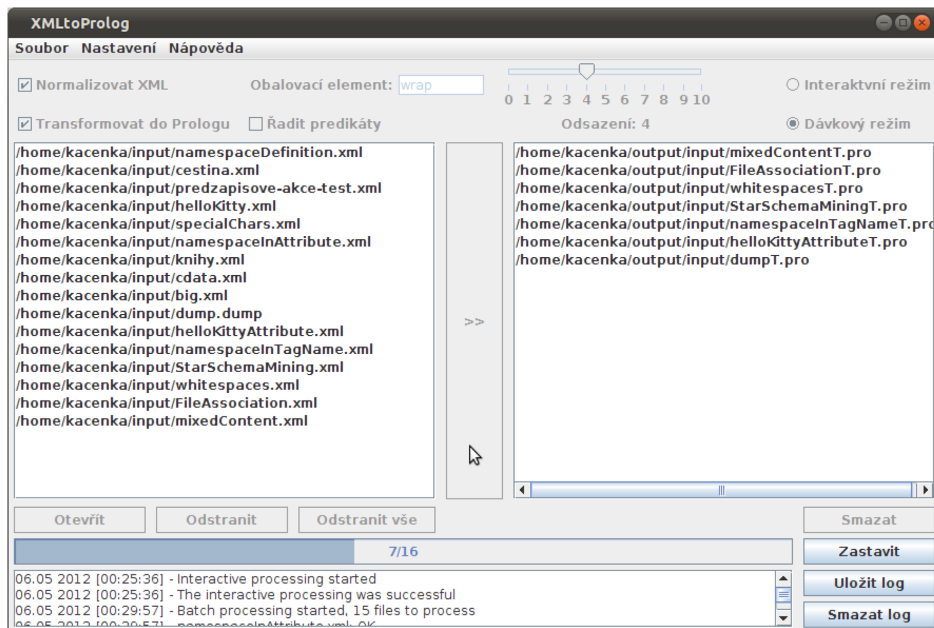


Obrázek 2: Hlavní okno aplikace v interaktivním režimu.

Dávkový režim

V dávkovém režimu se provádí převod více dokumentů najednou. Hlavní okno aplikace v dávkovém režimu viz obrázek 3. Stiskem tlačítka *Otevřít* se zobrazí okno pro výběr souborů. Se stiskem tlačítka `ctrl` je možné vybrat i více souborů a složek. K otevření je možné použít i volby v menu *Soubor* → *Otevřít soubor* (v tomto případě bude povoleno vybírat pouze soubory) a *Soubor* → *Otevřít složku* (bude povoleno vybírat pouze adresáře). V levém okně se zobrazí seznam vybraných souborů. Pokud byl vybrán celý adresář, zobrazí se seznam všech souborů uložených v tomto adresáři i ve všech jeho podadresářích. Kliknutím na název souboru se vybere příslušná položka a stiskem tlačítka *Odstranit* je možné soubor odstranit ze seznamu souborů určených ke zpracování. Stiskem tlačítka *Odstranit vše* se odstraní celá dávka souborů.

Po stisku tlačítka se symbolem '»' nebo volbou *Soubor* → *Zahájit zpracování* se spustí dávkové zpracování všech souborů vypsaných v levém okně podle nastavených parametrů. Výsledky zpracování souborů se vypisují v okně s logy. Pokud transformace proběhla úspěšně a byl tedy vytvořen výstupní soubor, je jméno tohoto souboru zobrazeno v pravém okně. Nad oknem s logy se nachází indikátor průběhu zpracování, který ukazuje, kolik souborů již bylo zpracováno a jaký je celkový počet souborů určených ke zpracování. Během



Obrázek 3: Hlavní okno aplikace v dávkovém režimu.

zpracovávání souborů jsou některé funkce nedostupné. Transformaci lze zastavit stiskem tlačítka *Zastavit*. Po dokončení dávkového zpracování je v okně s logy vypsán celkový výsledek dávkového zpracování.

Ukončení aplikace

Aplikace se ukončuje volbou *Soubor* → *Konec* nebo stiskem tlačítka pro zavření hlavního okna aplikace (zpravidla křížek v pravém horním rohu).