Západočeská univerzita v Plzni

Fakulta Aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Webová aplikace pro zpracování a transformaci XML dokumentu

Plzeň, 2012 Jan Beneš

Prohlášení

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

V Plzni dne 6. května 2012

Jan Beneš

Abstract

This work is based on two original works from the previous academic year. It implements and describes the integration of two Java applications from the original works into a form of web application. The resulting web application allows to transform the input XML document into a logic program and it also allows to normalize this document. Firstly, the original works are analysed. Secondly, the most commonly used technologies which can be used to implement web application are introduced. Finally, one of these technologies is selected as the best for implementation. After that proposal and implementation of web application are described.

Poděkování

Rád bych tímto způsobem poděkoval Ing. Martinovi Zímovi, PhD za vedení mé bakalářské práce, věnovaný čas při konzultacích a za cenné rady, poskytnuté k realizaci tohoto dokumentu.

Obsah

1 Úvod

Využívat možností World Wide Webu[1] (WWW) mohou dnes téměř všichni uživatelé, kteří mají přístup k internetu. Jedná se o nejúčinnější způsob, jak téměř cokoliv zpřístupnit komukoliv na světě. Za webovou aplikaci považujeme program, umístěný na webovém serveru, čekající na požadavky klientů – uživatelů, které následně obslouží. Uživatel komunikuje s aplikací přes uživatelské rozhraní, které je tvořeno webovým prohlížečem. Běžný webový prohlížeč je ve většině případů jediné, co uživatel ke spuštění webové aplikace potřebuje. To lze označit za velkou výhodu webových aplikací. I díky tomu se tento typ aplikací těší rostoucí popularitě. S rostoucí popularitou webových aplikací přirozeně roste jejich počet a také vývoj technologií, kterými je možné je implementovat. Vývoj technologií následně umožňuje možnost jejich užití v různých oblastech.

Tato práce navazuje na dvě původní bakalářské práce. Práce Lukáše Gemely [1] a Františka Schneidera [2]. Cílem mé práce bude implementovat integraci obou prací (aplikace XMLogic [1], logicky spojující knihovnu XMLtoProlog [1] s knihovnou pro normalizaci XML[2] dokumentů [2]) právě do podoby webové aplikace. Pod pojmem normalizace si lze představit modifikaci XML dokumentu do podoby, odpovídající požadavkům vstupního XML dokumentu práce [1]. Pokud by dokument nebyl normalizovaný, nebylo by možné jej knihovnou XMLtoProlog transformovat. Procesem transformace bude dokument transformován do podoby faktů logického programovacího jazyka Prolog[3], dle zadaných parametrů.

Výstupem této práce bude webová aplikace, která umožní načíst vstupní XML dokument a také nabídne možnost jej editovat, před samotným procesem transformace. Výstupní množina predikátů bude zobrazena uživateli a zároveň uložena v souboru, který bude možné stáhnout pro další použití. Prolog, jako zástupce jazyků logického programování, umožňuje mimo jiné najít ve vstupním XML dokumentu informace, které nemusí být na první pohled patrné.

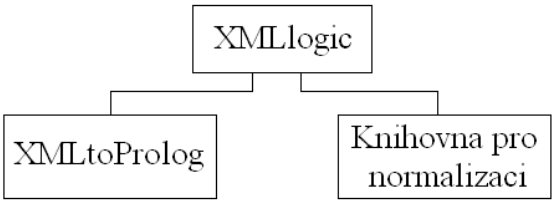
Čtenář bude v následující druhé kapitole nejprve seznámen s výchozím stavem obou původních prací. Ve třetí kapitole, budou analyzovány možnosti realizace webových aplikací. Bude vybrána nejvhodnější technologie pro implementaci, která bude následně popsána podstatně důkladněji. Ve čtvrté kapitole bude prezentován návrh a popsána implementace. Postupně se tak uvede vše podstatné, co obnáší transformace běžné Java aplikace do podoby webové aplikace. Funkčnost výsledného programu bude samozřejmě otestována.

[1] WWW – systém provázaných hypertextových dokumentů na Internetu
[2] XML – eXtensible Markup Language
[3] Prolog – logický programovací jazyk

2 Výchozí stav původních prací

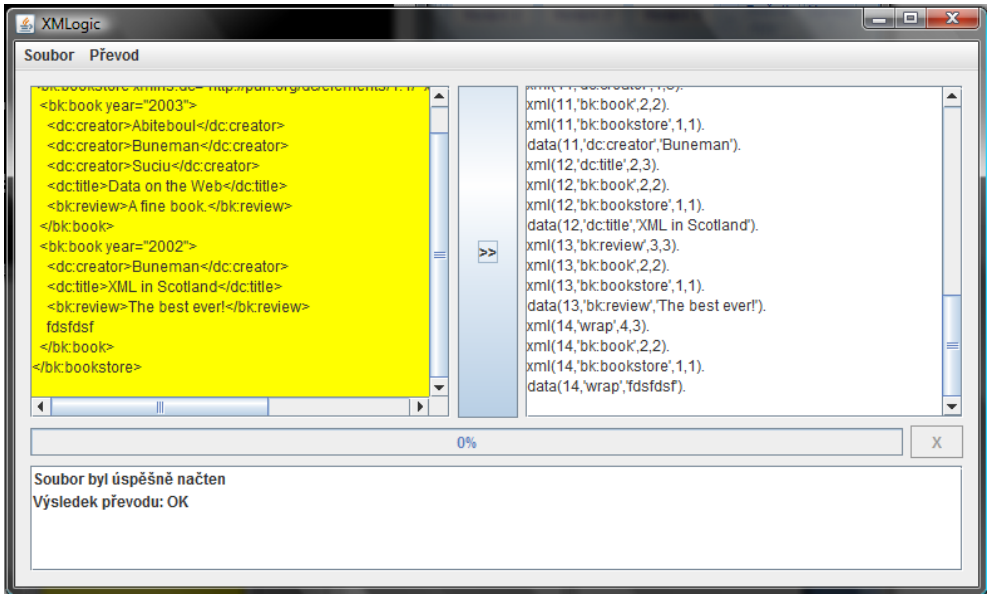
2.1 Transformace XML do podoby logického programu

Práce [1] je rozdělena do dvou implementačních částí. A to na část (knihovna XMLtoProlog), která má za úkol zrealizovat proces transformace na množinu predikátů pro Prolog. Druhá část - knihovna XMLlogic, je logická nástavba již zmíněné knihovny XMLtoProlog a knihovny pro normalizaci [2]. Knihovna XMLlogic řídí běh celé aplikace, dle zadaných parametrů a využívá při tom implementace obou knihoven. Logika celé aplikace je znázorněna na Obr. 2.1.



Obrázek 2.1: Hiearchie knihoven z výchozích prací.

Tato kapitola 2.1 je zaměřena především na popis důležité funkčnosti práce [1]. Nejprve se seznámíme s podstatnými informacemi, které jsou pro pochopení a navázaní mojí práce důležité. Aby byl sled prezentovaných znalostí chronologický, bude jako první přestaven formát XML. Jedná se o formát vstupních souborů, se kterými Java applet pracuje. Protože práce [1] již byla několikrát zmíněna, považuji za důležité čtenáři ukázat, jak její činnost vypadá (Obr. 2.2). Ještě před tím, než bude popsána. Porovnáním vstupu a výstupu aplikace, si lze představit účel aplikace.

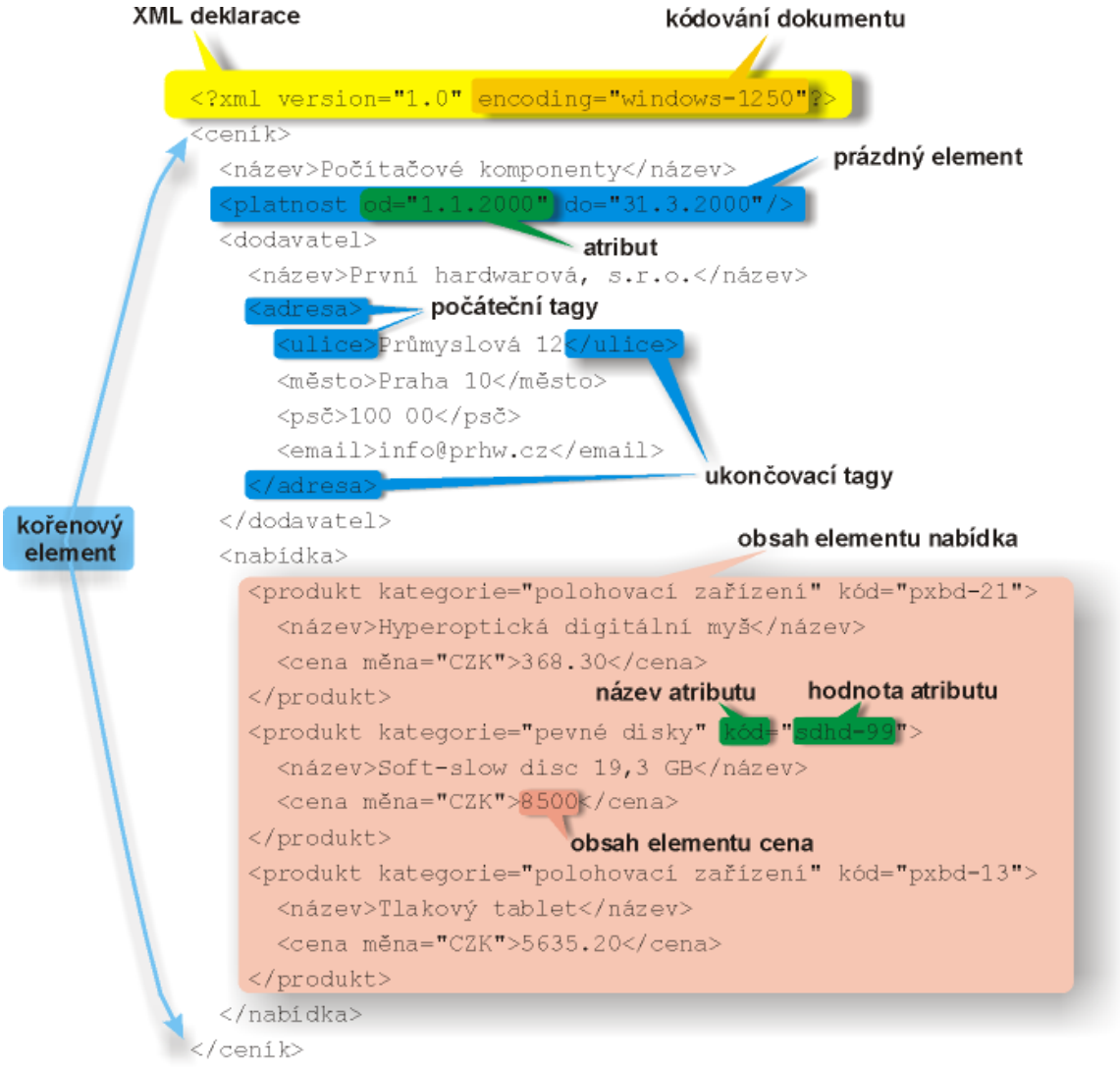


Obrázek 2.2: Výsledek činnosti práce [1].

2.1.1 Formát XML

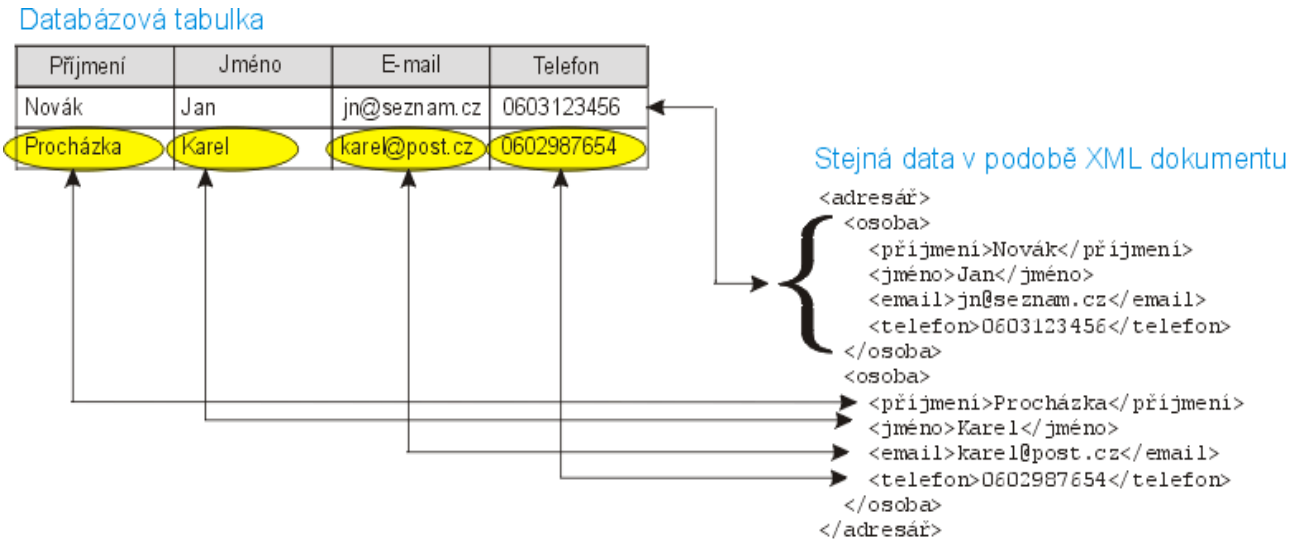
XML - eXtensible Markup Language. Jak už název v anglickém jazyce implikuje, XML patří mezi značkovací jazyky. V terminologii XML se jednotlivým částem dokumentu, označenými značkami říká *elementy*. Elementy se v textu vyznačují pomocí tzv. *tagů*. Mohou být navzájem vnořené, ale nesmí se překrývat. Tím mohou, dle potřeby, zachycovat požadovanou strukturu informací obsažených v dokumentu.

Kdybychom například chtěli do XML uložit obsah nabídky dané firmy, prodávající počítačové komponenty, skládal by se dokument z elementů *produkt*. Každý element produkt by pak obsahoval další elementy *název* a *cena*, které by popisovali jeho vlastnosti. Firma může poskytovat i více informací. Nejenom nabízené produkty, ale také podrobnější informace o firmě a platnosti nabízených produktů. Pomocí XML to není problém realizovat. Takto může vzniknout ceník, který pak může být součástí nějakého katalogu. Lze tak intuitivně abstrahovat k postupnému rozmělňování dokumentu na menší a menší části dle potřeby. Příklad takového XML s popisem jeho částí je uveden na Obr. 2.3 [3].



Obrázek 2.3: Příklad XML dokumentu.

XML svojí syntaxí připomíná jazyk HTML, disponující omezenou množinu značek, která je přesně definována - omezena. Pomocí XML jsou informace členěny do logického uspořádání, nikoliv do požadovaného vzhledu, jako je tomu u HTML. Takový přístup pak nabízí daleko větší možnosti při zpracování dat, než HTML a navíc dává možnost širšího uplatnění XML, jako formátu pro ukládání dat. Názorný příklad možnosti uložení záznamu z relační databáze do formátu XML je na Obr. 2.4 [3].



Obrázek 2.4: Záznam z databáze ve formátu XML.

Jak je na Obr. 2.4 vidět, logická struktura informace zůstala zachována. Následně bude uvedeno několik vlastností popisující využití formátu XML dokumentů.

- XML je nelicencované, platformě nezávislé a široce podporované.
- Díky možnosti definování vlastních tagů, lze využít i pro tvorbu webových stránek. Tím jsou obsaženy možnosti jazyka HTML. XML bývá někdy označováno za nástupce jazyka HTML.
- Je asi vůbec prvním formátem, který akceptuje i jiné jazyky než je angličtina. Jako znaková sada se používá ISO 10646, která mimo jiné zahrnuje i kódování UTF-8. ISO 10646 je 32bitová znaková sada, která dokáže pojmout všechny dnes používané znaky všech jazyků. Je zde i prostor kombinovat více jazyků dohromady.
- Jak lze tušit, XML dokumenty jsou informačně bohatší. To lze samozřejmě s výhodou využít v mnoha oblastech, například ve vyhledávání.

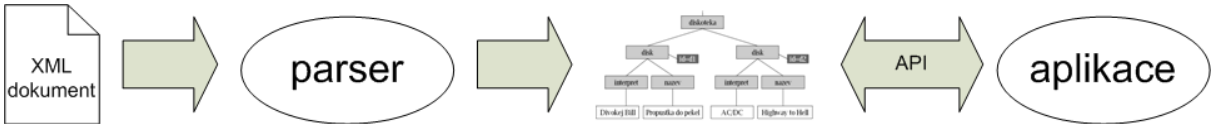
Pro tvorbu XML dokumentů samozřejmě existují pravidla, definující jejich správnou strukturu. Těmito obecnými pravidly jsou:

- každý XML dokument by měl začínat tzv. deklarací, ve které bude uvedena verze XML a kódování. Jedná se o speciální typ elementu ohraničený znakem “?“.
- element musí začínat uvozující značkou ve tvaru <jméno_značky>.
- konec elementu je vyznačen koncovou značkou </jméno_značky>, která je povinná.

- před ukončením elementu, musí být ukončeny všechny jeho vnitřní elementy – nesmí se překrývat.
- prázdný element vyznačíme <jméno_elementu/>
- uvozující značka může obsahovat atributy v podobě <jméno_značky atr1=„aaa“ atr2=„bbb“>
- znaky, které jsou součástí textu a mají speciální význam (např. <), musí být nahrazeny escape sekvencí.
- nejvyšší element v hierarchii elementů je nazýván „root (kořenový) element“ a smí být pouze jeden.

2.1.2 Problematika a možnosti zpracování XML dokumentů

Aby bylo možné vstupní dokument zpracovat (transformovat), je třeba nejprve zkontrolovat jeho strukturu dle uvedených pravidel. Program, který kontroluje syntaktickou správnost dokumentu, se nazývá *parser*. Parser za nás detekuje většinu možných chyb v datech a tím nám ušetří mnoho případných komplikací. Parserů existuje celá řada a každý se používá pro konkrétní úlohu. Známe základní dva typy parserů pro proudové čtení dat – Pull parsery a Push parsery. Jejich vlastnosti jsou detailně popsány v práci [2].



Obrázek 2.5: Význam parseru [2].

Mezi nejznámější dva způsoby jak zpracovávat (parsovat) XML dokument patří řízení událostmi a stromová reprezentace. Pro každý z těchto způsobů bylo vyvinuto jednotné API. Pro stromovou reprezentaci standart DOM (*Document Object model*) a pro zpracování řízené událostmi SAX (*Simple API for XML*). Následuje jejich stručný popis:

Rozhraní SAX (*Simple API for XML*) je založeno na řízení pomocí událostí (*event-driven*). Pomocí rozhraní vytvoříme vazbu mezi událostmi, které generuje parser, a naším kódem. V praxi to znamená, že si nadefinujeme funkce, které se zavolají v okamžiku, kdy parser narazí na začátek elementu, obsah, a konec elementu nebo na komentář a instrukce pro zpracování atd. Aplikaci jsou pak předány všechny potřebné parametry jako např. název elementu.

Rozhraní SAX dnes podporuje velké množství parserů, nejčastěji jím bývá označován hlavně XERCES.

Rozhraní DOM (*Document Object Model*) je postaveno na zcela odlišném principu než SAX. Dokument je reprezentován jako stromová hierarchická struktura, kde každému elementu odpovídá jeden uzel stromu. V tomto rozhraní je použito více než 10 typů uzlů. DOM obsahuje funkce, které nám umožní celou stromovou strukturu procházet, editovat jeho

jednotlivé uzly, mazat je a přidávat. Na rozdíl od SAXu, nemusíme dokument procházet sekvenčně, ale můžeme se v něm pohybovat dle naší libosti. Proto se rozhraní DOM uplatní v aplikacích, které provádějí náročnější operace s dokumentem – editory, prohlížeče, náhodný přístup.

Parsery SAX mají oproti DOM několik výhod. Tyto výhody jsou detailně popsány v práci [2]. Mezi hlavní důvod, proč se pro Java applety více hodí parser typu SAX patří především fakt, že velikost paměti, kterou SAX alokuje, je řádově menší než u DOM. Tudíž je i SAX rychlejší. Což je u webové aplikace přívětivé.

2.1.3 Prolog – zástupce logického programování

Prolog je deklarativní programovací jazyk. Charakteristikou deklarativního programování je popis cíle, nikoliv způsob výpočtu. Ten se nechává na programu a je vyhodnocován pomocí predikátové logiky. Nejprve zadáme vstupní fakta, a poté nad touto množinou faktů pomocí pravidel definujeme dotazy.

Nespornou výhodou jazyka Prolog je jeho jednoduchá syntaxe. Programování v Prologu se skládá ze tří základních kroků, kterými utvářím výstup:

1. Definice faktů, které jsou nám známy o objektech a vztahy mezi nimi.
2. Definice obecných pravidel, které platí mezi objekty a jejich vztahy.
3. Formulace dotazů.

Nejjednodušší příklad faktu “Lenka nejí maso“, by byl definován takto:

neji(lenka, maso).

S tímto typem predikátů si v této práci vystačíme, proto už programování v Prologu nebudu dále rozvádět. Celý seriál o možnostech programování v Prologu je uveden ve zdroji [4].

2.1.4 Proces transformace XML dokumentu na predikáty

Máme tedy k dispozici vstupní dokument, v němž nás zajímá jeho rozdělení na elementy, atributy a data. Zajímá nás tzv. Infoset XML dokumentu. Do Infosetu patří i komentáře. Protože ale v normalizovaných XML dokumentech, ani ve výstupních predikátech svůj význam nemají, nebude se s jejich zpracováním počítat. Ke struktuře vstupního XML dokumentu použijeme n-tici faktů, které budou popisovat minimální cestu mezi kořenovým a aktuálně rozpracovaným elementem. Pro každý z těchto elementů platí obecný tvar:

xml(cislo_radku, nazev_znacky, poradi_znacky, uroven_zanoreni).

Vysvětlení významu argumentů predikátu pro elementy XML dokumentu:

- **cislo_radku** – udává číselně pozici od začátku dokumentu, na kterém se značka elementu vyskytla
- **nazev_znacky** – obsahuje název tohoto elementu
- **poradi_znacky** – pořadí elementu v aktuální úrovni zanoření
- **uroven_zanoreni** – udává číselně aktuální úroveň zanoření

Pro svázání vnořených elementů s jejich nadřazenými předchůdci, je nutné pro každý vnořený element definovat nový predikát. A to pro každý jemu nadřazený element. Aby byla zajištěna potřená minimální cesta ke kořenovému elementu. Tvar těchto predikátů je:

xml(cislo_radku_vnorene znacky, nazev_nadrazene_znacky, poradi_nadrazene_znacky, uroven_zanoreni_ nadrazene znacky).

Výše definované predikáty ovšem definují pouze strukturu XML souboru. Jak už víme, můžou se v něm vyskytnout také atributy a data. Pro predikáty dat je obecný tvar:

data(cislo_radku, nazev_znacky, data).

Argument *data* obsahuje text, který se nachází mezi elementy. Pro predikáty atributů je obecný tvar:

atribut(cislo_radku, nazev_znacky, nazev_atributu, hodnota atributu).

Uvedené typy predikátů nám již postačí k transformaci. Nyní bude ukázána a okomentována transformace jednoduchého XML dokumentu. Tento příklad (Obr. 2.6) fragmentu vstupního dokumentu je přejat z práce [1].

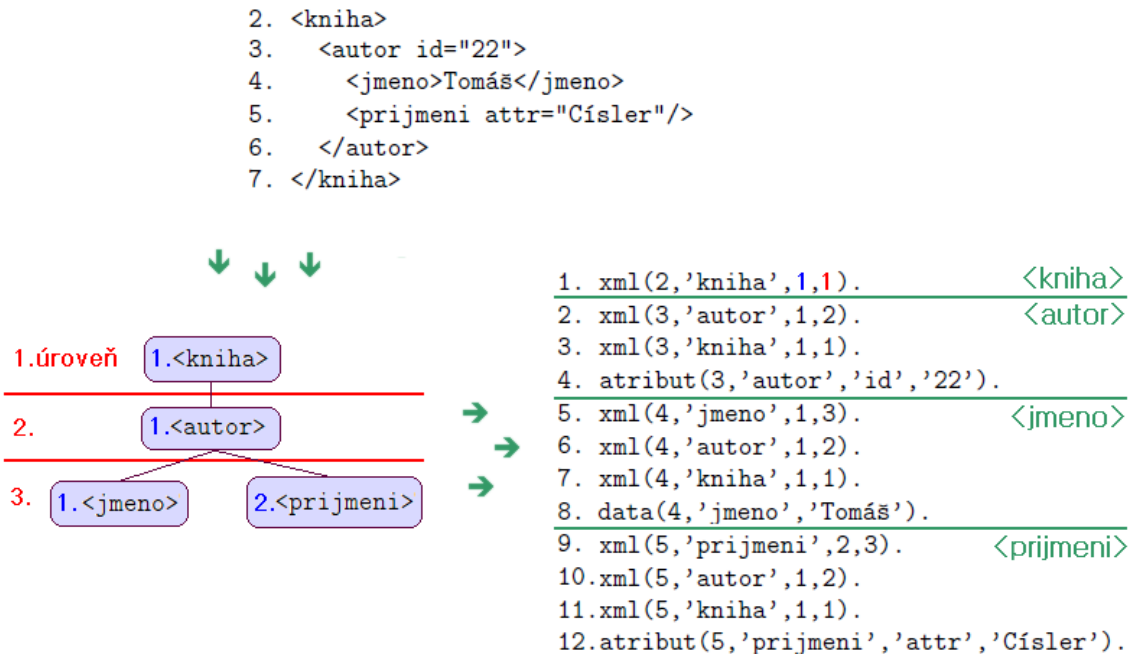
Nechť je dán vstupní XML soubor, jehož obsah bude následující:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <kniha>
3.   <autor id="22">
4.     <jmeno>Tomáš</jmeno>
5.     <prijmeni attr="Císler"/>
6.   </autor>
7. </kniha>
```

Obrázek 2.6: Obsah vstupního XML dokumentu.

Transformace se týká všech elementů vstupního souboru kromě deklarace XML. Pro každý element se vytvoří tolik predikátů, odpovídající úrovni zanoření (kořenový element je na úrovni zanoření 1). Za ně se případně přidají predikáty, vytvořené z atributů nebo dat.

Obr. 2.7 slouží pro lepší představu, jak se z XML struktury predikáty získají. Parser vytvoří stromovou strukturu elementů. Červenou barvou jsou zobrazeny úrovně zanoření. Modrou, pořadí elementů na stejné úrovni. V pravé části jsou zeleně odděleny predikáty pro každý element. Musí se uvést predikát nejenom aktuálně zpracovávaného elementu ale i všech na minimální cestě ke kořenovému elementu. Poté přijdou na řadu predikáty pro atributy a data.



Obrázek 2.7: Grafické zobrazení transformace XML dokumentu na predikáty.

Nyní jsme se dostali k výsledku transformace. Predikáty stačí už jen seřadit pro přehlednost. Způsob, jakým toho docílit, je uveden v uživatelské příručce.

2.2 Zpracování XML do potřeby sémantického webu

Obsahem práce [2] byla analýza možností zpracování XML dokumentů a jejím výstupem je knihovna pro normalizaci XML dokumentu. Tato knihovna umožňuje zpracovávat dokumenty pomocí čtyř nejzákladnějších API. Pro potřeby Java appletu, vzhledem k typu vstupních souborů, bohatě postačí pouze jedno API. I díky práci [2], za nejvíce univerzální považuji StAX, proto jsem ho také zvolil, jako API pro moji implementaci. Bude více popsáno v kapitole 2.2.2. Analýza všech ostatních API je podrobně rozepsána v práci [2]. V následující kapitole bude nastíněn proces normalizace, aby měl čtenář představu o tom, co vše proces normalizace obnáší.

2.2.1 Proces normalizace

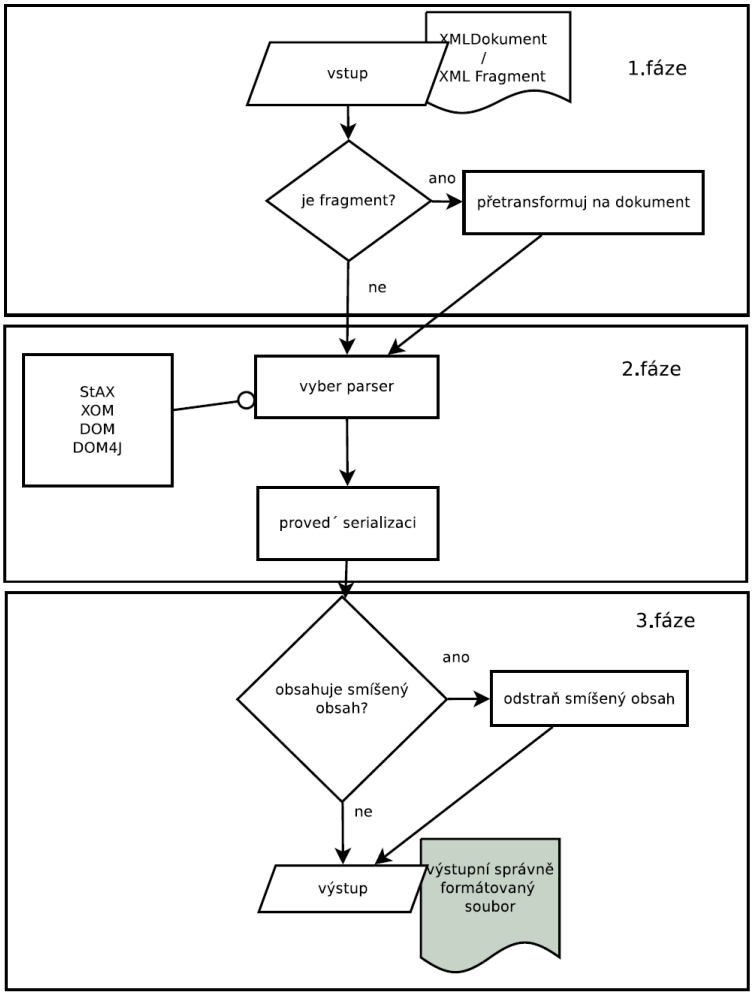
Cílem procesu normalizace je vstupní soubor rozparsovat a upravit jej tak, aby soubor odpovídal struktuře, splňující následující podmínky:

- Každý element XML dokumentu bude umístěn na své řádce. V případě, že bude element obsahovat jiné elementy, musí být uvedena jeho uvozující a ukončující značka na nové řádce.
- Dokument nesmí obsahovat smíšený obsah. Často se vyskytuje v HTML dokumentu, ale jeho výskyt je teoreticky možný i u XML dokumentů.

Případ smíšeného obsahu:

<tag1>Text mezi<tag2>další text</tag2>tagy</tag1>

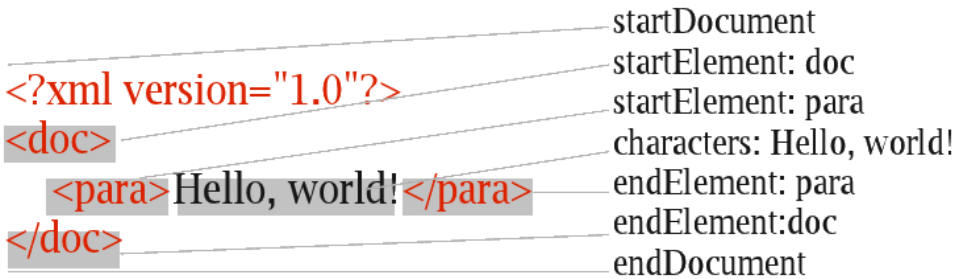
Normalizace je rozdělena do 3 fází, popsaných v práci [2]. Následující schéma alespoň naznačí, na jaké fáze je převod rozdělěn a co se v nich odehrává. Podrobněji jsou popsány v práci [2].



Obrázek 2.8: Schéma procesu normalizace XML dokumetu.

2.2.2 StAX - Streaming API for XML

Jedná se o pull-parser. Patří mezi událostně orientovaná API. Zapouzdřuje v sobě dvě různá API. První z nich - *Cursor API*, přes které přistupujeme k aktuální pozici v XML dokumentu (vždy pouze dopředu). A druhé z nich - *Event iterátor API*, přes které obdržíme událost, jako objekt. Příklad toho, jak se v závislosti na procházení dokumentu generují události je zobrazen na Obr. 2.9.

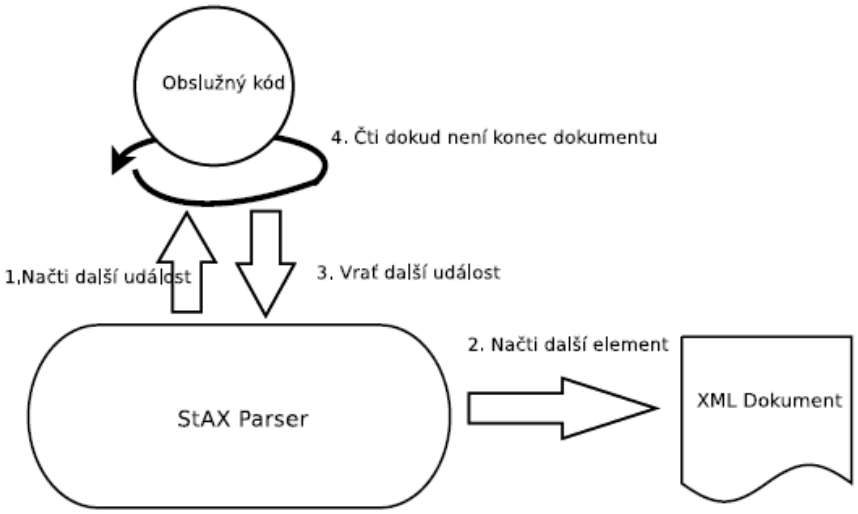


Obrázek 2.9: Příklad generování událostí.

Za největší výhodu StAXu považuji spojení výhod rozraní SAX a DOM. Těmi jsou: řízený posun v datech, zápis dat a poměrně malá spotřeba paměti. StAX se hodí i pro poměrně velké XML soubory. Dále budou uvedeny všechny výhody StAXu a zobrazen princip jeho činnosti.

Hlavní výyhody StAXu:

- Lze XML dokument jak vytvářet, tak i do něj zapisovat.
- Fakt, že pracuje jako stream, umožňuje zpracovat i velké soubory.
- Při čtení je zdrojový kód na jednom místě, u SAXu ve třech vzájemně propojených metodách.
- Textový obsah elementu je vracen najednou. Tím odpadá postupné ukládání do bufferu.
- Čtení probíhá na žádost.



Obrázek 2.10: Princip činnosti StAXu [2].

3 Způsoby realizace webových aplikací

3.1 Vlastnosti webových aplikací

Základem téměř každé webové aplikace je úvodní HTML stránka. V ní jsou definovány nebo rovnou použity technologie, kterých aplikace bude využívat. S webovou aplikací uživatel komunikuje pomocí tzv. *tenkého klienta*[1]. Webový server je umístěný na stroji připojenému k internetu, jehož úkolem je obsluhovat požadavky klientů (jejich prohlížečů), které se k němu připojí. Nejčastěji komunikace probíhá prostřednictvím *HTTP*[2] protokolu. Hlavními požadavky na webový server jsou: rychlost, bezpečnost, obsluha více uživatelů najednou a správná reakce na chybové stavy. Jedním z nejpoužívanějších webových serverů je *Apache*. Další informace o webovém serveru *Apache* jsou uvedeny v uživatelské příručce.

Hlavní výhodou webové aplikace je její užívání, bez dodatečného softwaru a rozsáhlejších instalací. Jediné, co je potřeba mít k dispozici je webový prohlížeč. Případně je nutné do prohlížeče doinstalovat chybějící zásuvné moduly (plug-iny), potřebné pro správnou funkčnost aplikace. Mezi další nesporné výhody patří: dostupnost webových aplikací odkudkoliv, kde je přístup na web; nezávislost fungování aplikace na operačním systému, instalovaný na stroji klienta. Protože některé typy webových aplikací mohou být na svých vlastnostech omezeny oproti desktopovým aplikacím, používá se k jejich částečnému nahrazení skriptovacích jazyků. Webový prohlížeč totiž zobrazuje webové stránky dynamicky generované webovou aplikací. Každá z těchto stránek je prohlížeči dodána jako statický dokument. Série takto dodaných stránek pak vyvolává pocit interaktivity. Ovšem pouze zdánlivě. Musí dojít například ke znovunačtení stránky, aby se požadavek od uživatele zpracoval (například zadání dat). V závislosti na dané události je pak vygenerován další sled událostí. Jedním ze způsobů, jak interaktivitu zajistit bez znovu načítání stránky, je využití technologie skriptovací jazyků.

Za první typ webových aplikací jsem zvolil nejčetnější skupinu, co se týče využití technologií. Bude popsána v následující kapitole. Dělení na různé způsoby realizace webových aplikací tedy bude probíhat podle nejčastěji využívaných technologií, které jsou vhodné pro implementaci interaktivní webové aplikace.

[1] Tenký klient – klient, který nezná logiku aplikace
[2] HTTP - HyperText Transfer Protocol

3.2 Kombinace HTML, CSS, PHP a JavaScriptu.

Jak bylo uvedeno výše - jedná se o nejvyužívanější skupinu technologií, za jejichž pomoci může být webová aplikace implementována. Níže budou popsány všechny uvedené technologie. Vyjma HTML.

CSS(*Cascading Style Sheets)* je jazyk pro popis způsobu zobrazení stránek ve formátu HTML, XHTML a XML. Jeho hlavním cílem je oddělení vzhledu dokumentu od jeho struktury a obsahu. Pomocí CSS je možné nadefinovat vzhled, umístění, chování a vlastnosti všech elementů, uvedených v HTML stránce.

PHP (*Hypertext Preprocessor, původně Personal Home Page*) je scriptovací programovací jazyk s volně dostupným zdrojovým kódem, který umožňuje programovat jak strukturovaně tak i objektově. PHP slouží především k programování dynamických interaktivních stránek na straně serveru. Poskytuje možnost předávat data mezi stránkami.

JavaScript je hlavní zástupce skupiny skriptovacích jazyků. Dnes už je podporován téměř všemi prohlížeči a na rozdíl od PHP pracuje na straně klienta. Z názvu implikuje možné spojení s Javou. Ovšem tento jazyk, až na podobnou syntaxi s Javou nemá nic společného. V názvu se Java objevila z komerčních důvodů. Mezi hlavní výhody JavaScriptu patří relativní jednoduchost a nepotřebnost kompilátoru. Účelem tohoto jazyka je možnost rozšířit možnosti statických HTML stránek především o interaktivní potenciál. Jedná se o jazyk, kterým je možné částečně nahradit PHP. Ovšem nevylučuje se i možnost využívat PHP společně s JavaScriptem.

Více informací o jednotlivých technologiích lze nalézt ve zdrojích [5] [6] [7].

Logiku webové aplikace, která je implementována těmito technologiemi, lze shrnout do jedné věty. HTML umožňuje zobrazit data, CSS dodá potřebný vzhled a PHP s JavaScriptem zajistí potřebnou interaktivitu a funkčnost. Problém této konstelace spočívá v potřebě znovunačtení stránky, při zpracování požadavků zadaných uživatelem přes uživatelské rozhraní. A to může narušovat pocit dynamičnosti aplikace.

3.3 AJAX

Aby webová aplikace byla opravdu dynamická, bez nutnosti znovunačtení stránky, je možné využít technologie AJAXu (*Asynchronous Javascript and XML*). Webové aplikace implementované touto technologií, využívají kombinaci HTML, JavaScriptu a rozhraní XMLHttpRequest. Toto rozhraní umožňuje načítat klientským skriptům informace ze serveru, aniž by bylo třeba obnovovat celou stránku. Další informace o této technologii lze nalézt ve zdroji [8].

Máme tedy k dispozici první technologii, kterou je možné implementovat opravdu dynamickou a interaktivní webovou aplikaci. Problém ale může nastat při jejich zobrazování

různými prohlížeči. Ne všechny typy webových prohlížečů totiž zobrazují uživatelské rozhraní webových aplikací stejným způsobem. Existují minimálně dvě možnosti, jak zajistit stejné zobrazení uživatelského rozhraní pro všechny existující typy webových prohlížečů. A to pomocí Java appletu nebo pomocí Adobe Flashů. Obě tyto možnosti realizace budou popsány v následujících dvou kapitolách.

3.4 Adobe Flash

Adobe Flash je určen spíše pro aplikace, náročnější na grafiku. Jako webová aplikace má také široké uplatnění v oblasti hraní her, animací, prezentací a vektorové grafiky. Tzv. Flashe se píší v objektovém programovacím jazyce ActionScript, nebo se vytváří v některém z grafických programů pro to určených. Už po tomto výčtu vlastností, je jasné, že tento typ webové aplikace využít pro účely mé práce nepůjde.

3.5 Java applet

Pojem *Java applet*, jako typ webové aplikace byl poprvé použit již v roce 1990. Jedná se o program napsaný v programovacím jazyce Java, který pro svoji činnost potřebuje Java-kompatibilní prohlížeč. Spustí se otevřením HTML[1] stránky, na niž je definován. Takto strukturovaná webová aplikace nabízí relativně velké možností využití, už jen tím, jak rozsáhlé jsou možnosti programovacího jazyka Java. Protože se applet spouští na klientské platformě, může svojí činností vyžadovat oprávnění, která jsou mimo rozsah stanovených práv appletů (např. načtení souboru). Pokud tento rozsah práv přesahuje, musí být digitálně podepsaný.

Protože obě původní práce jsou implementovány v Javě, bylo by velice náročné reimplementovat původní práce v jiném programovacím jazyce. Proto bude nejvhodnější zvolit cestu implementace pomocí Java appletu. Právě tato podoba webové aplikace bude ta, do které budou obě práce transformovány. Máme tedy zvolenou technologii, která bude použita při reimplementaci. Dále bude následovat popis vlastností Java appletů s využitím zdrojů [9] [10] [11] [12].

3.6 Vybraný způsob realizace – Java applet

Java applet představuje integraci JAR[2] archivu, do HTML stránky za pomoci speciálního párového elementu <APPLET>. JAR archiv, obsahující kód aplikace, je při každé návštěvě webové stránky natažen do webového prohlížeče uživatele a je spuštěn v prostředí Java Virtual Machine, které se na počítač instaluje spolu s prohlížečem.

[1] HTML – Hyper Text Markup Language
[2] JAR představuje Java Archiv. Jedná se o způsob distribuce souborů *.class, které jsou algoritmem ZIP zkomprimovány do jednoho souboru.

To znamená, že výpočetní výkon, který applet potřebuje, je závislý na parametrech stroje klienta.

Kvůli bezpečnostním rizikům existuje pro applety několik omezení, popsaných v kapitole 3.6.2 Po zavření HTML stránky, na niž je applet definován, se jeho činnost automaticky ukončí. Je také možné definovat, co se stane v případě nečekaného přerušení běhu appletu. Dále je potřeba zmínit, že například pro práci appletu se souborem, musí být applet digitálně podepsán, jak už bylo uvedeno výše. V rámci jednoho Java appletu je možné definovat více tříd, z nichž jedna bude spouštěcí a definovaná v párové značce <APPLET>. U appletu nemůže být definována metoda `main()`, jako je tomu u Java aplikace, kde je výskyt této metody povinný. Applet může mít zpřístupněné některé své vlastnosti a metody jako veřejné, což umožňuje externím programům manipulaci s nimi.

3.6.1 Obecná struktura Java appletu

Strukturu každého appletu tvoří třída *java.applet.Applet*. Tato třída definuje základní metody, které tvoří rozhraní mezi prohlížečem a appletem. Každý program, který má fungovat jako applet, musí být nutně potomkem této třídy. Pokud je applet načten do webové stránky s příponou *.html*, prohlížeč řídí zpracování appletu voláním určitých metod. Pro smysluplnou činnost je třeba v potomkovi překrýt alespoň jednu z nich. Jejich výčet a stručný popis je uveden níže.

- `public void init()` – tato metoda slouží k libovolné inicializaci, kterou applet vyžaduje.

- `public void start()` – tato metoda je automaticky volána po metodě `void init()` a to pokaždé, kdy se uživatel vrátí na stránku s appletem, po zobrazení jiných stránek.

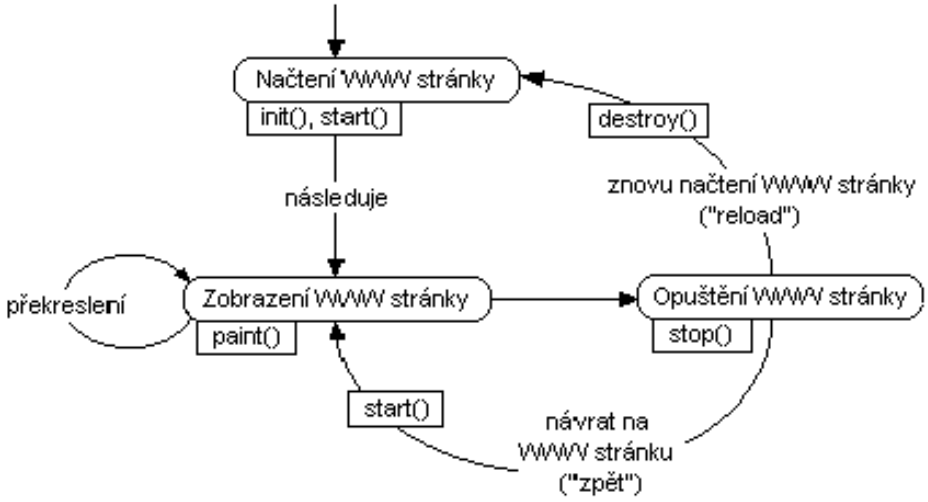
- `public void paint()` – tato metoda je volaná při překreslování stránky prohlížečem, konkrétně metodou `repaint()`.

- `public void stop()` – tato metoda je automaticky volána kdykoliv uživatel opustí stránku, na niž se applet nachází.

- `public void destroy()` – tato metoda je volána pouze při ukončení prohlížeče.

Mnoho dalších metod je uvedeno ve zdroji [9].

Znázornění toho, jak v závislosti na stavu prohlížeče, probíhá volání definovaných metod appletu, zobrazuje Obr. 3.1 [9].



Obrázek 3.1: Volání metod appletu v závislosti na stavu prohlížeče.

3.6.2 Omezení a rozšíření Java appletů

Z bezpečnostních důvodů platí pro applet některá omezení:

- Applet nemůže nahrávat knihovny ani definovat *nativní* metody.
- Applet nemůže navazovat síťové spojení na jiný než domovský server.
- Applet nemůže zapisovat do souborů na straně klienta (prohlížeče).
- Applet nemůže spouštět programy na domovském serveru.
- Applet nemůže číst některé systémové proměnné.

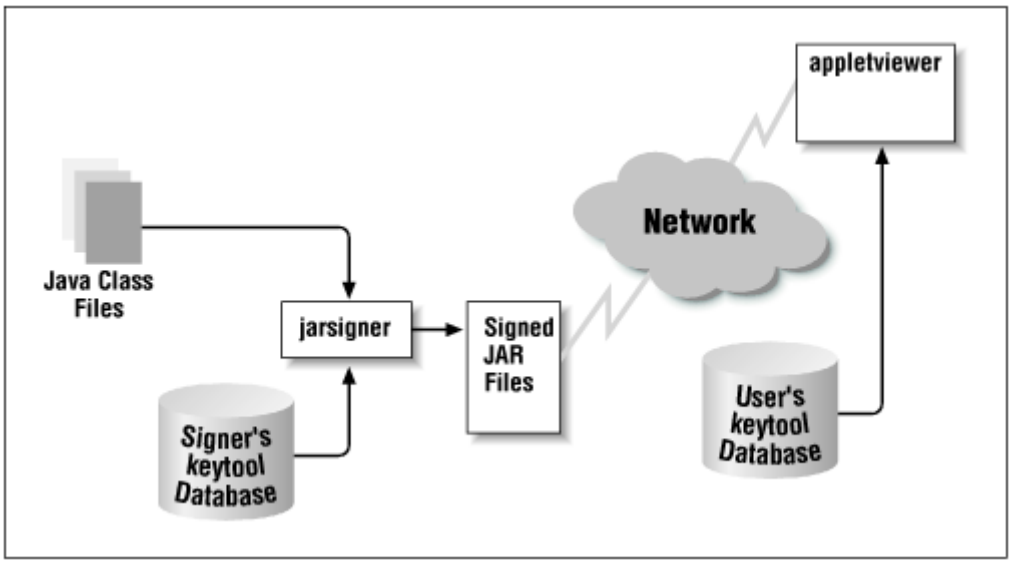
a naopak má applet některé funkce rozšířeny:

- Applet může přehrávat zvuky.
- Applet může požádat prohlížeč o zobrazení libovolné WWW stránky.
- Applet může volat veřejné metody jiných appletů umístěných na téže WWW stránce.

3.6.3 Digitální podpis

Podepsání JAR archivu je nutné vždy, vyžaduje-li applet práva, která jsou mimo rozsah stanovených práv appletů, nebo dojde-li k reimplementaci JAR archivu. Za prává přesahující stanovený rozsah se považuje například práce se souborem, umístěným na disku uživatele nebo například síťová komunikace. Tyto činnosti jsou považovány za potenciálně nebezpečné, neboť jich lze zneužít.

Princip celého procesu digitálního podepsání je zobrazen na Obr. 3.2 [13]. Za pomoci příkazu *keytool* se nejprve vygeneruje certifikační klíč. Na nově vygenerovaný klíč se vytvoří odkaz, jehož název lze nastavit. Dle tohoto odkazu se při procesu podepisování použije právě tento jeho název. Podepsání realizuje příkaz *jarsigner*. Podrobný postup je vylíčen v programové dokumentaci.



Obrázek 3.2: Princip digitálního podpisu.

3.6.4 Integrace appletu do HTML stránky

Vytvořený a digitálně podepsaný JAR archiv, který provádí požadovanou činnost, je třeba umístit na HTML stránku. Neexistuje totiž jiná možnost, jak ho umístit na web nějakým jiným způsobem. Integraci JAR archivu do HTML stránky umožňuje párová značka <APPLET>. Může obsahovat následující parametry:

- `CODE` – obsahuje název hlavní (spouštěcí) třídy appletu.

- `CODEBASE` – obsahuje cestu ke spouštěcí třídě. Používá se v případě, že je umístění spouštěcí třídy různé od umístění adresáře, ve kterém je HTML stránka umístěna.

- `WIDTH` – určuje šířku okna appletu. Je možné ji zadat jak v pixelech, tak i v procentech.

- `HEIGHT` – určuje výšku okna appletu. Je možné ji zadat jak v pixelech, tak i v procentech.

- `ARCHIVE` – specifikuje jeden, nebo více JAR archivů, ze kterých se Java applet skládá.

Následuje příklad, jak může taková integrace appletu do HTML stránky vypadat:

```
<APPLET CODE="AppletXML.class " ARCHIVE="AppletXML.jar"
WIDTH="100%" HEIGHT="100%"></APPLET>
```

Úplný kód HTML stránky pak může vypadat následovně:

```
<!DOCTYPE HTML SYSTEM>
<HTML>
<HEAD>
<TITLE>Webová aplikace pro zpracování a transformaci XML
dokumentu</TITLE>
</HEAD>
<BODY>
<APPLET CODE = "AppletXML" ARCHIVE = "AppletXML.jar"
WIDTH = "100%" HEIGHT = "100%">
</APPLET>
</BODY>
</HTML>
```

Prvním elementem je deklarace stránky. Má stejný význam jako u XML dokumentů. Výskyt párových tagů <HTML>,<BODY> je v kódu stránky povinný. Název celého appletu je uveden v titulku hlavičky HTML stránky. Hlavička stránky je část kódu umístěná mezi párovými tagy <HEAD>. Mohou v ní být umístěny různé tagy, definující vlastnosti stránky. Správnou strukturu HTML stránky můžeme verifikovat tzv. validátorem. Nejznámějším validátorem je http://validator.w3.org/. Stránka, zobrazená výše je samozřejmě validní.

Do této části dokumentu byly uvedeny všechny teoretické znalosti, potřebné k realizační části a vysvětlující postupy.

4 Programátorská dokumentace

4.1 Zadání

1. Seznamte se s možnostmi zpracování XML dokumentu a způsoby realizace webové aplikace.

2. Navrhněte webovou aplikaci, která transformuje vstupní XML dokument do podoby logického programu.

3. Navrženou webovou aplikaci implementujte nejlépe v programovacím jazyce Java.

4. Na testovací sadě dokumentů ověřte správnou funkčnost aplikace.

4.2 Analýza

Funkčnost obou prací odpovídá funkčnosti požadovaného appletu. Proto je nejprve třeba provést důkladnou analýzu, která je pro transformaci aplikace do podoby appletu stěžejní. Nejprve jsem si vyzkoušel programové vybavení obou původních prací. Vybíral jsem různé typy souborů a zkoušel možné kombinace nastavitelných parametrů v obou pracích. Výsledky byly porovnány s tím, co jsem nastudoval v dokumentacích. Tímto způsobem spolu s analýzou kódů jsem pochopil, jak proces transformace a normalizace probíhá a jaké jsou jejich možnosti nastavováním parametrů.

Dále jsem si vyzkoušel základní práci s Java applety. Jednalo se o různé typy appletů, zaměřující se vždy jen na určitou funkčnost (např. načtení souboru), které lze volně naleznout na webu. Zdroj [10] například prezentuje postup při načtení souboru Java appletem. Dalšími zdroji pro implementaci jednoduchých appletů jsou zdroje [11] [12]. Po úspěšné implementaci nejen těchto typů Java appletů, jsem mohl využít získaných znalostí i v mojí práci. Následně jsem dospěl k návrhu, který zde bude popsán v kap. 4.3. Nyní bude uvedeno několik podstatných informací pro návrh.

Protože třída *java.applet.Applet* definuje svoje vlastní metody pro aplikaci tak, aby mohla fungovat jako applet, bude nutné práce XMLogic a XMLtoProlog částečně reimplementovat. Reimplementace čeká také částečně knihovnu pro normalizaci XML. Z té bude vybrán jeden parser ze čtyř původních a ten bude používán na všechny typy vstupních XML dokumentů.

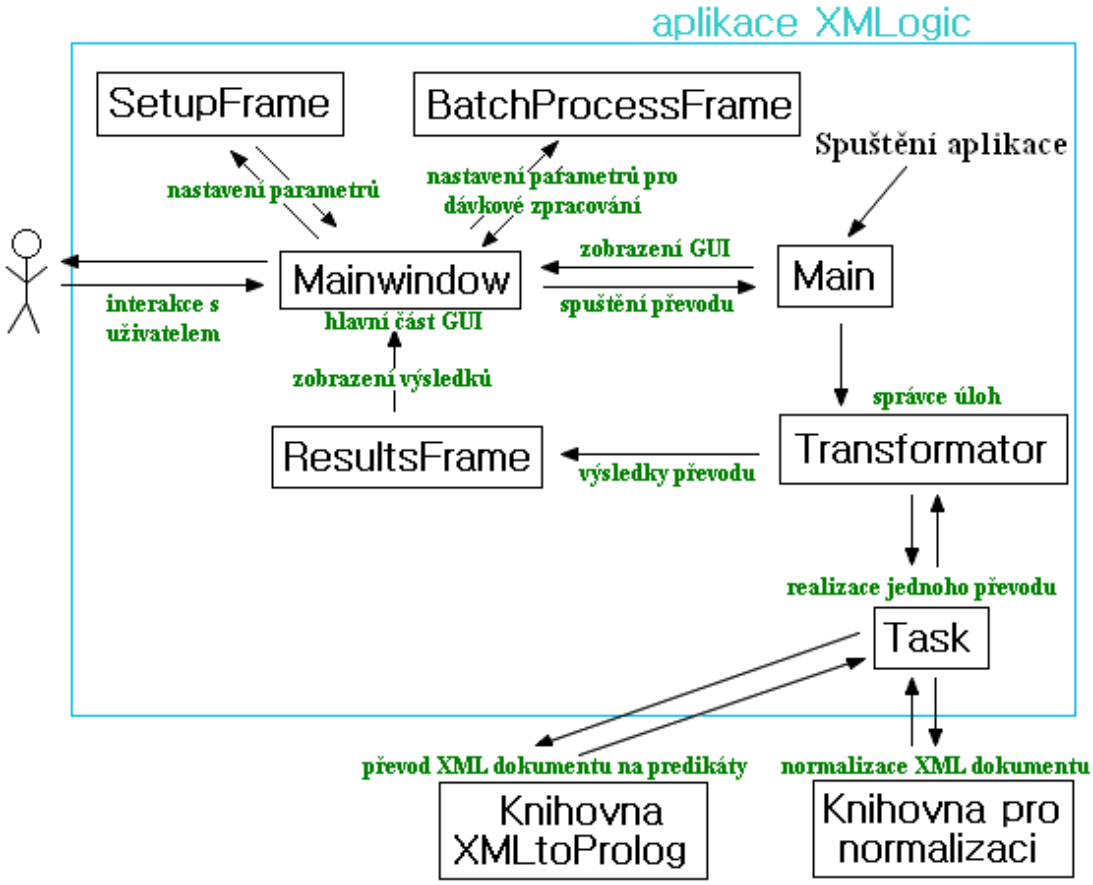
Uživatelské rozhraní aplikace XMLtoProlog se skládá z prvků knihovny Swing[1]. Doposud jsem s touto knihovnou neměl téměř žádné zkušenosti.

[1] Swing - knihovna pro tvorbu grafického uživatelského rozhraní v Jave

Proto jsem si od přátel vyžádal podklady z předmětu KIV/UUR a po určité době jsem získal potřebné dovednosti, použitelné k implementaci uživatelského rozhraní.

Převod XML do podoby logického jazyka Prolog (knihovna XMLtoProlog) zůstane stejný jako v práci [1]. Největší změny nastanou v aplikaci XMLogic. Dojde zde například k odstranění paralelismu. Nahrávat se vždy bude právě jeden soubor. Dávkové zpracování proto již není požadavkem, tudíž nebude potřeba paralelního zpracování (model Farmer-Worker). Modifikace se konkrétně dotkne tříd *Task* a *Transformator*.

Následuje schéma analyzované aplikace XMLogic, využívající knihoven XMLtoProlog a knihovnu pro normalizaci. Po tomto schématu bude následovat slovní popis pro podrobnější vysvětlení.



Obrázek 4.1: Schéma aplikace XMLogic.

Po spuštění aplikace se vytvoří instance třídy *MainWindow*, která tvoří hlavní část GUI[1]. Uživateli je umožněno nastavit přes horní menu, možnost načtení souboru. A nebo také parametry, pro nastavení dávkového převodu, stejně jako převodu pouze jednoho XML dokumentu.

[1] GUI - Grafické uživatelské rozhraní

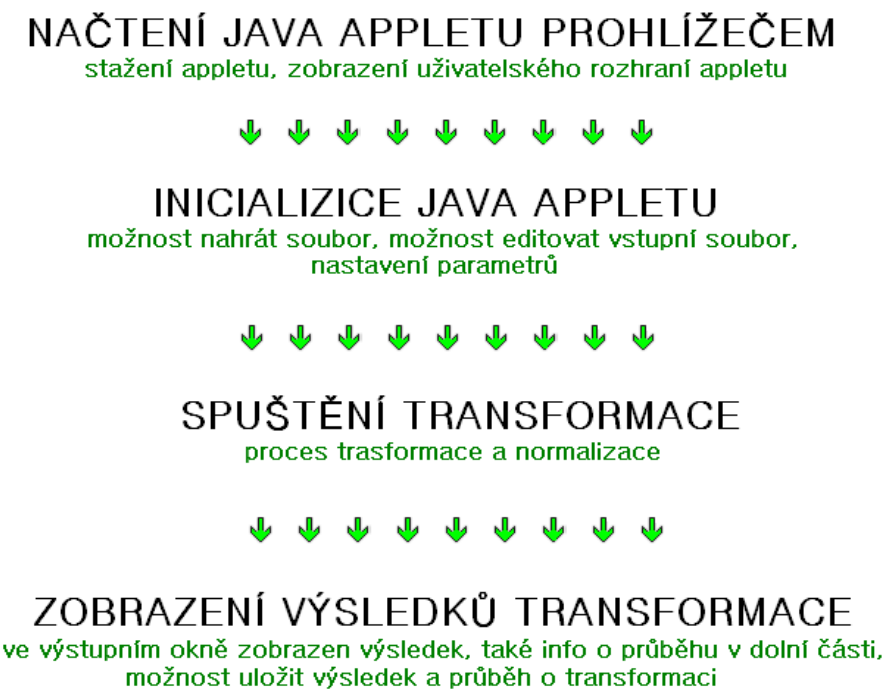
Mezi tyto parametry patří výběr metody pro předzpracování XML (StAX, XOM, DOM, DOM4J), možnost třídění výstupních predikátů, nastavení počtu vláken, kterých bude program využívat a pro dávkové zpracování – výběr souborů, které se budou transformovat. Parametry umožňují nastavit instance tříd *SeupFrame* a *BatchProcessFrame* (pro dávkové zpracování).

Po nastavení parametrů a zahájení převodu kliknutím na příslušné tlačítko, se vytvoří instance třídy *Transformator*. Účel implementace této třídy lze přirovnat ke správci úloh, kde jedna úloha realizuje převod jednoho dokumentu (instance třídy *Task*). V instanci třídy *Task* se dle zadaných parametrů, používají knihovny pro normalizaci a transformaci dokumentu na predikáty. Knihovnám je předán odkaz na soubor, který zpracovávají. Po dokončení procesu transformace a normalizace daná knihovna zašle informaci o průběhu, která se zobrazí v uživatelském rozhraní spolu s očekávaným výsledkem.

Způsoby, jakými pracují knihovny XMLtoProlog a knihovna pro normalizaci, jsou detailně analyzovány v pracích [1] a [2]

4.3 Návrh Java appletu

Při návrhu Java appletu jsem vycházel z podmětů získaných analýzou. V prvé řadě bude představena navrhovaná logika appletu (na Obr. 4.2). Tedy série důležitých činností appletu od načtení vstupního XML souboru, po zobrazení výsledků transformace.



Obrázek 4.2: Logický model navrženého Java appletu.

Už v prvním bodě logického modelu – načtení uživatelského rozhraní je zřejmé, že bude třeba

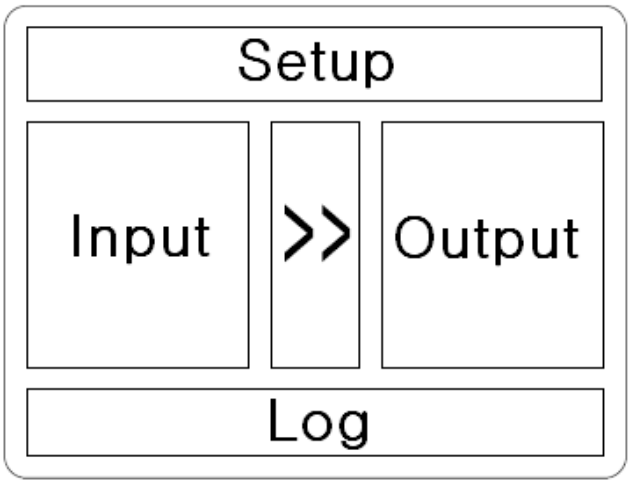
výrazně reimplementovat nejprve uživatelské rozhraní. K němu bude následně postupně implementována funkčnost tak, aby byl běh appletu korespondoval s navrhovanou logikou. Aby mohlo být uživatelské rozhraní zobrazeno ihned po načtení appletu, musí být definováno v kódu metody `init()`, třídy *Applet*. V aplikaci XMLogic bylo uživatelské rozraní tvořeno čtyřmi třídami - *MainWindow*, *BatchProcessFrame*, *SetupFrame*, *ResultFrame*. Existenci třídy *BatchProcessFrame* zcela vypustíme, neboť její instance umožňuje nastavení parametrů pro dávkové zpracování a možnost zobrazit indikátor stavu transformace. Po dohodě se zadavatelem se dospělo k závěru, že dávkové zpracování a indikátor stavu transformace nebudou potřeba. Z tříd *MainWindow* a *SetupFrame* bude vytvořena jedna metoda, která bude uvedena v metodě *`init()`* třídy *Applet*. V této metodě (`void userInterface()`) budou definovány veškeré prvky, se kterými uživatel v rámci appletu může přijít do kontaktu, tedy celé uživatelské rozhraní, umožňující inicializaci appletu. Návrh uživatelského rozhraní bude popsán níže.

Návrh uživatelského rozhraní

Původní GUI je implementováno za pomoci knihovny Swing. Možností této knihovny bude využíváno i nadále, v mé navazující práci. Nejdůležitější prvky uživatelského rozhraní budou mít stejnou funkčnost i podobné umístění, jako v práci [1]. Návrh je zobrazen na Obr. 4.3. Jedná se o prvky:

- levé editační okno (*Input*), do něhož je načten buďto obsah nahraného souboru, nebo je obsah vytvořen uživatelem. Přes toto okno je také možné nahraný soubor editovat.
- pravé editační okno (*Output*), zobrazující výstupní soubor
- tlačítko pro zahájení transformace vstupního XML souboru
- spodní informační panel (*Log*), informující uživatele o průběhu transformace.

Z původního GUI, bude zrušeno otevírací menu v horní liště, umožňující práci se souborem a nastavení parametrů transformace. Parametry transformace bude možné nastavit přímo. Budou přehledně umístěny v horní části appletu (oblast *Setup*). V této části se bude také nacházet tlačítko pro načtení souboru.



Obrázek 4.3: Návrh GUI Java appletu.

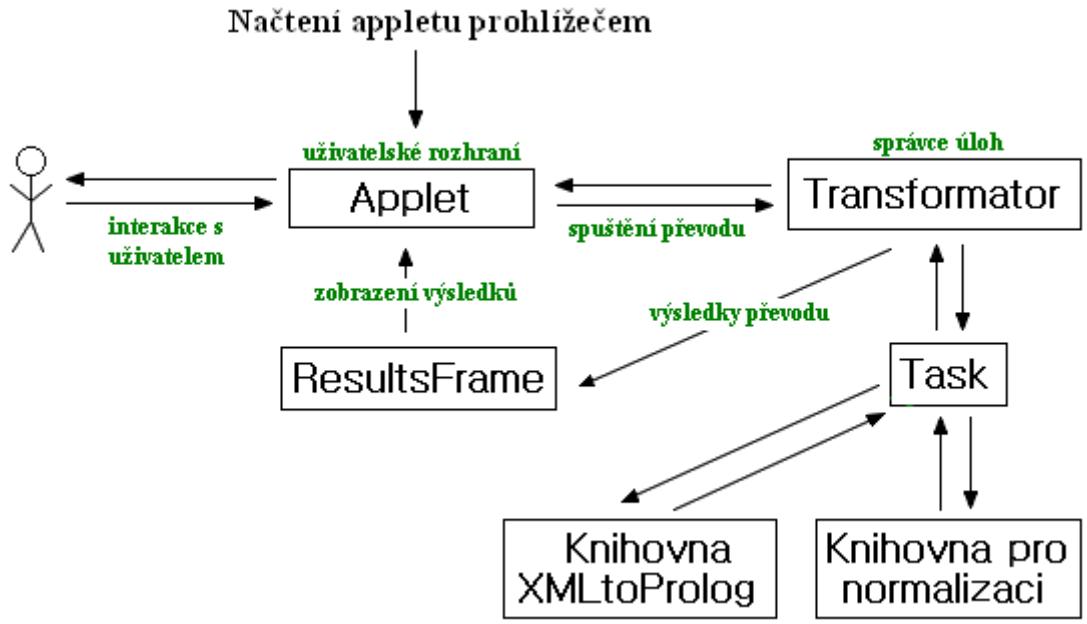
Editační a výstupní okno bude možné uložit a smazat, stejně jako dolní informační panel. A to přes samostatná tlačítka, která budou umístěna u příslušných komponent. Obsah výstupního okna bude existovat jako dočasný soubor, proto jej bude možné uložit.

Rozměry appletu budou závislé na velikosti okna prohlížeče. Při změně velikosti okna prohlížeče, se budou všechna tři okna (*Input, Output, Log*) uživatelského rozhraní měnit ve stejném poměru, jako byla jejich původní velikost. Rozměry ostatních komponent zůstanou zachovány. Při implementaci uživatelského rozhraní jsem využíval zdrojů [14] a [15].

Po implementaci nejdůležitějších komponent uživatelského rozhraní, bude následně postupně přidávána funkčnost původních prací. Nejprve bude zprovozněna část transformace predikátu do podoby Prologu a poté proces normalizace. Budou následovat návrhy modifikace obou prací.

Návrh modifikace práce [1]

Následující schéma na Obr. 4.4 zobrazuje navrhovanou modifikaci aplikace XMLogic do funkčnosti Java appletu. Nejvýraznější modifikaci lze zpozorovat u nově vytvořené třídy *Applet*, která převezme funkci uživatelského rozhraní a třídy *Main,* od tříd z původní práce.



Obrázek 4.4: Návrh logiky Java appletu.

Třída *Transformator* v návrhu zůstala i přes to, že se v analýze vyskytuje úvaha o odstranění paralelismu a tudíž není teoreticky potřeba. Pro převod jednoho souboru není správce úloh (tedy třída *Transformation*) potřeba. V návrhu třída zůstala pro její možné případné budoucí uplatnění. Na výkonnost ani paměť to vliv mít nebude, proto je nadbytečné tuto funkčnost odebírat.

Modifikace knihovny XMLtoProlog nebude nutná. Její implementaci nebude potřeba vůbec modifikovat.

Návrh modifikace práce [2]

Po předchozí subkapitole je k dispozici návrh appletu, který bude umět transformovat normalizovaný soubor na predikáty. Aby mohl být vstupem jakýkoliv XML soubor, je třeba implementovat i možnost normalizace vstupního XML souboru. Z jakéhokoliv vstupního XML souboru se pak stane soubor normalizovaný a ten bude možno předat knihovně XMLtoLogic.

Knihovnu pro normalizaci XML souborů, implementovanou pro potřeby práce [1], čeká odstranění některých tříd (např. XOMParser, DOMParser, DOM4JParser), z důvodu jejich nepotřebnosti. Modifikovány budou muset být třídy *StAX* a *ContentManagerm*, z důvodu korektního fungování odsazení predikátů, dle úrovně zanoření.

4.4 Implementace

Jak už bylo zmíněno, byl pro implementaci zvolen programovací jazyk Java. A to především proto, že obě původní práce jsou v jazyce Java také implementovány. Mezi hlavní výhody tohoto programovacího jazyka patří robustnost, objektová orientace a hlavně platformní nezávislost. Platformní nezávislost je u appletu, jako webové aplikace, která může běžet na všech možných platformách, velice vítaná vlastnost.

Co se týče architektury, nemá smysl, aby byla třívrstvá. Prezentační vrstvu by tvořila jediná třída (*Applet*). Vytvářet vrstvu, ve které by byla jediná třída (v datové by byly dvě) by přehlednému rozčlenění celé aplikace příliš nepomohlo.

V další části této kapitoly budou popsány třídy z prací [1] a [2], které byly modifikovány a třídy, které byly přidány. Jak už víme, došlo ke zrušení třídy *Main* z aplikace XMLtoLogic. Její úlohu přebrala nově vytvořená třída Applet, kterou popis implementace započne.

Applet.java

Hlavní, nově vytvořená třída Java appletu. V Metodě `void init()`, jejíž kód se vykoná po načtení Java appletu prohlížečem, se nachází metoda `void userInterface()`. V této metodě je definováno uživatelské rozhraní, tvořící základ appletu. Uživatelské rozhraní je tvořeno prvky grafické knihovny Swing, pomocí kterých uživatel nastavuje statické proměnné – parametry procesu normalizace a transformace.

Mezi zmíněné statické proměnné patří:

- `public static boolean transformToProlog` – umožní převod XML dokumentu na predikáty. Nastavuje se komponentou JCheckBox. Zaškrtnutím se nastaví logická hodnota TRUE a zobrazí se možnost řazení predikátů.

- `public static boolean sort` – umožní třídit výstupní predikáty. Nastavuje se také komponentou JCheckBox.

- `public static boolean normalizeXML` – umožní normalizovat vstupní soubor. Nastavuje se komponentou JCheckBox. Zaškrtnutím se zobrazí následující dvě komponenty, pro nastavení parametrů normalizace

- `public static String wrapTagName` – název obalovacího (wrap) elementu. Nastavuje se komponentou JTextField.

- `public static int indentSize` – číselná hodnota odsazení elementů XML dokumentu, dle úrovně zanoření. Nastavuje se komponentou JSlider.

Zahájením převodu se nastaví statickým proměnným hodnoty aktuálně zadaných parametrů. Ještě těsně před tím, dojde k ověření, zda je vyplněna hodnota obalovacího elementu. V případě že nikoliv, bude mu defaultně nastavena hodnota "wrap". Mezi další konstanty této třídy patří maximální velikost načteného souboru (*50kB*).

Vybráním souboru uživatelem, dojde ke kontrole toho, zda soubor nepřesahuje definovanou maximální hranici velikosti nahraného souboru. Pokud ano, soubor nebude načten a uživateli se v dolním informačním okně zobrazí příslušné upozornění. Všechna tlačítka pro uložení a načtení souboru si pamatují naposledy zvolený adresář. Je uložen ve statické proměnné `public static String lastPath`.

Z důvodu požadavku na dvě jazykové verze Java appletu (angličtina a čeština), jsou implementovány dvě metody, nastavující příslušným komponentům změnu jejich popisku.

Task.java

Instance této třídy reprezentuje převod jednoho XML dokumentu. Parametrem konstruktoru *File inputFile* nastavíme cestu ke vstupnímu souboru. Další parametry, nastavované uživatelem v uživatelském rozhraní, jako nastavení normalizace, transformace, odsazení, třídění predikátů, si instance třídy *Task* uloží do svých vnitřních proměnných ze statických proměnných třídy *Applet*.

Protože už byly zmíněny důvody k odstranění paralelismu, není již třeba, aby třída *Task* dědila od třídy *Thread*. V práci [1] to bylo z důvodu potřeby implementace dávkového převodu.

Proces transformace se zahájí voláním metody `run()`. V této metodě zůstane téměř vše stejné jako v práci [1]. Pouze u volby normalizace dojde k nastavení odsazení a názvu obalovacího elementu (wrap element), do vnitřních proměnných třídy *SettingsManager*. Instance této třídy reprezentuje nastavení a uchování parametrů pro normalizaci a je poté předána třídě *Job*.

Facade.java

Tato třída realizuje normalizaci XML dokumentu. Je pomyslným vstupním bodem pro aplikaci XMLogic. Proces normalizace se spustí voláním metody `void work(boolean temp)`. Parametrem *boolean temp* bylo možné v původní práci zapsat výstup normalizace do dočasného souboru, nebo výstup směrovat do předem nastavené složky. U Java appletu se počítá pouze s dočasným souborem, který si uživatel může stáhnout přes tlačítko Save.

Výraznější zásah v implementaci této třídy byl v odstranění použití instance třídy *Classloader*. Tato třída umožňuje dynamicky nahrávat třídy (v původní práci to byly parsery) za běhu aplikace, podle výběru v GUI. Protože v mé práci je použit parser pouze jeden (StAX), nebude již třída *ClassLoader* potřeba.

StaxParser.java

Instance této třídy realizuje parsování vstupního XML dokumentu. Důležité parametry, jako např. odsazení a název obalovacího elementu, jsou předány přes konstruktor instancí třídy *Job*. Parsování je zahájeno voláním metody `int parse()`. Dochází zde také k již zmiňovanému přidávání mezer před elementy, podle jejich aktuální úrovně zanoření. Instance této třídy je poté předána třídě *ContentManager*.

Content Manager.java

Content manager je třídou pro detekci a odstranění smíšeného obsahu. Byly upraveny metody `void fixMixedContent2(String line)` a `void fixMixedContent1(String line)` tak, aby bylo možné realizovat odsazení elementů dle úrovně odsazení. V instanci třídy *ContentManager* se metodou `String getIndentSpaces(int count)` zjistí počet mezer, vyjadřující odsazení před aktuálně zpracovávaným elementem. Tento počet mezer se znásobí hodnotou odsazení, nastavenou uživatelem. Výsledná hodnota odsazení určuje počet mezer, přidaných před aktuálně zpracovávaný element.

5 Testování funkčnosti sadou dokumentů

Tato kapitola popisuje testování funkčností implementovaného Java appletu. Testována byla hlavně doba, potřebná jak pro proces transformace, proces normalizace tak i obojí společně. Proces transformace na predikáty byl vždy spuštěn s volbou „třídění predikátů“.

Testy probíhaly na následujících platformách:

- Laptop Compaq Presario CQ60 s dvoujádrovým procesorem Intel Pentium Dual T3400, 2,16GHz a 2,17GHz, 3GB RAM, s pevným diskem o velikosti 250GB. Nainstalovaným operačním systémem na tomto stroji je Windows Vista Home Premium (32bit.). Stáří stroje 4 roky. Dále jen sestava č.1.

- Laptop HP Pavilion dv6 s dvoujádrovým procesorem Intel Pentium P6100, 2.00GHz a 2.00GHz, 4GB RAM, s pevným diskem o velikosti 300GB. Nainstalovaným operačním systémem na tomto stroji je Windows 7 Home Premium (64bit). Stáří stroje 5 měsíců. Dále jen sestava č.2.

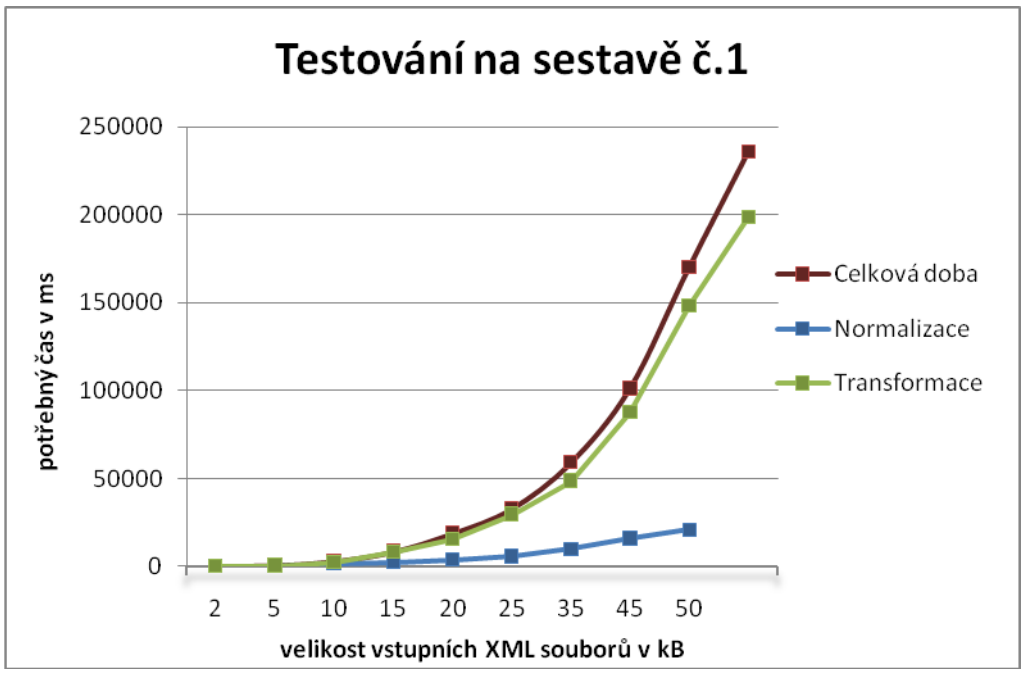
- Stolní PC Dell s dvoujádrovým procesorem Intel Pentium 6400, 2,13GHz a 2,13 GHz, 2GB RAM, s pevným diskem o velikosti 150GB. Nainstalovaným operačním systémem na tomto stroji je Windows 7 Enterprise (32bit). Stáří stroje 6 let. Dále jen sestava č.3.

Pro účely testování byla vytvořena následující sada dokumentů, s rozdílnou velikostí od 0,5 kB do 50 kB. Zde je jejich výčet:

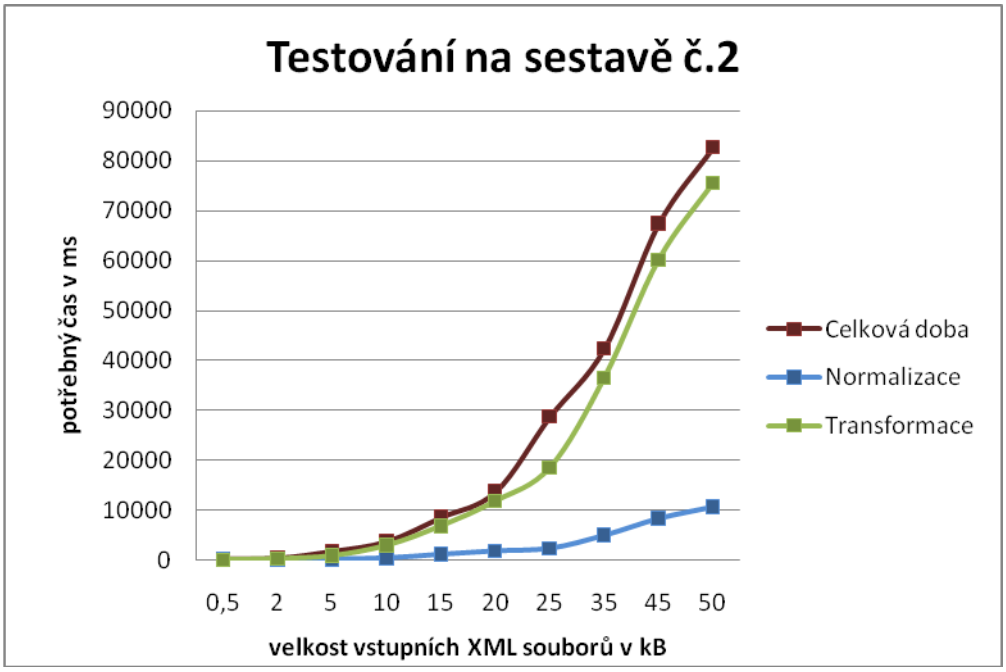
- knihy.xml (velikost ~ 0,5 kB)
- knihy02.xml (velikost ~ 2 kB)
- knihy05.xml (velikost ~5 kB)
- knihy10.xml (velikost ~10 kB)
- knihy15.xml (velikost ~15 kB)
- knihy20.xml (velikost ~ 20 kB)
- knihy25.xml (velikost ~ 25 kB)
- knihy35.xml (velikost ~ 35 kB)
- knihy45.xml (velikost ~ 45 kB)
- knihy50.xml (velikost ~ 50 kB)

Měřila se doba od zahájení převodu, po zobrazení výsledků. Pro každý vstupní soubor, byla doba zpracování naměřena celkem 5krát. Následně se hodnoty zprůměrovali. Nejprve byl měřen čas, potřebný pouze k normalizaci XML souboru. Poté pouze transformace normalizovaného souboru a na závěr byla měřena součinnost obou procesů. Takto testování probíhalo na všech třech testovacích sestavách.

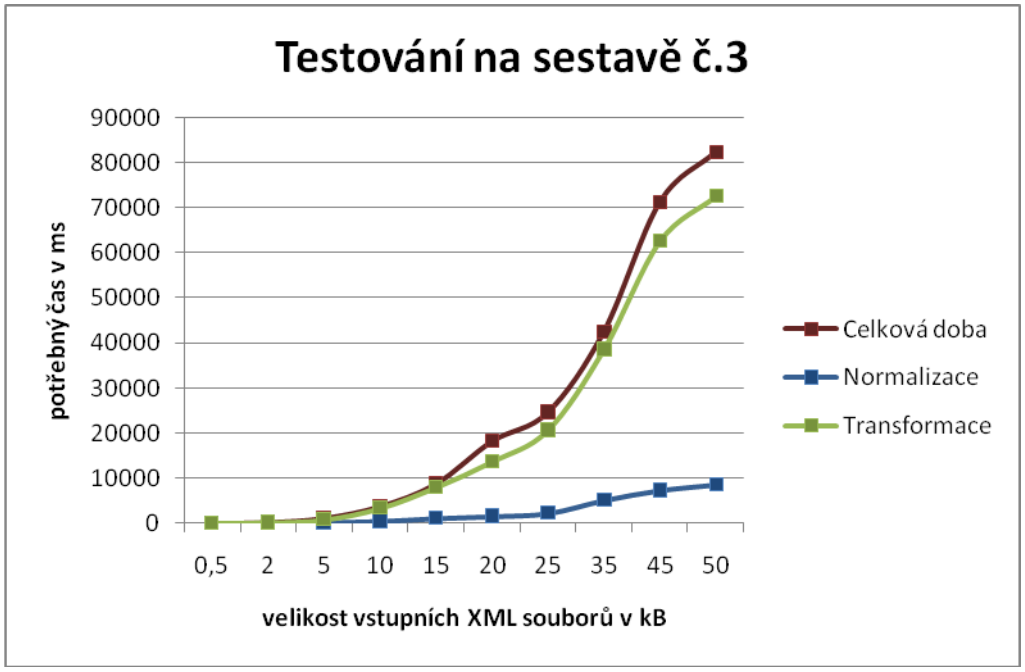
Následují data, naměřená ze všech tří sestav, zanesená do grafů.



Obrázek č. 5.1: Výsledky testování na sestavě č.1.



Obrázek 5.2: Výsledky testování na sestavě č.2.



Obrázek 5.3: Výsledky testování na sestavě č.3.

Z naměřených hodnot, zanesených do výše uvedených grafů je jasně vidět rozdílná doba, potřebná ke zpracování vstupního dokumentu. Nejvíce, mezi první sestavou a zbylými dvěma sestavami. Je také dobře vidět závislost doby zpracování na parametrech stoje, na kterém byl Java applet spuštěn.

Byl také proveden zátěžový test se souborem o velikosti 550 kB. Applet ani po 3 minutách nezobrazil výsledek. Poté došlo k "zamrznutí" prohlížeče z důvodu vypršení time-outu. Jinými slovy zobrazování výsledků trvalo příliš dlouho. Protože je velikost vstupních souborů limitována vlastnostmi prohlížeče, byla zvolena maximální velikost vstupního souboru na 50kB. A to právě proto, aby se mohl požadovaný výsledek zobrazit v rozumném čase.

6 Závěr

Účelem této práce bylo implementovat integraci původních prací do podoby webové aplikace. V první části tohoto dokumentu jsem se snažil o co nejbližší přiblížení stavu a významu podstatných částí původních prací. Po tomto seznámení následoval proces hledání pokud možno co nejvhodnější technologie pro implementaci webové aplikace. Ten byl také nalezen. I kdyby obě původní práce nebyly implementovány v programovacím jazyce Java, s největší pravděpodobností bych také zvolil Javu. Mám s tímto jazykem největší zkušenosti a myslím, že pro takto náročnou aplikaci se hodí nejvíce.

Po výběru technologie, následoval návrh budoucího Java appletu. Postupem času jsem zjistil, že nebyl tento návrh zcela korektní. Vyskytly se samozřejmě menší komplikace při implementaci, které jsem nemohl v rámci návrhu předpokládat. Proto byl návrh několikrát aktualizován tak, aby byl zcela korektní. Návrh byl také aktualizován postupnými připomínkami zadavatele, aby odpovídal jeho představám.

Po implementaci byl proveden zátěžový test větším dokumentem. Zjistilo se, že webová aplikace nemůže mít takový potenciál, co se týče velkých souborů, jako aplikace nativní, z práce [1]. Proces transformace a normalizace trval přibližně stejně jako u původních prací. Nejvíce času, více jak polovinu z celkového času, zabírá zobrazování výsledků. To je způsobeno omezeními webového prohlížeče, ve kterém je Java applet spuštěn. Omezení ve smyslu rychlosti zobrazování výsledků transformace či normalizace, které se výrazně projevuje u větších vstupních XML dokumentů.

Zadání této práce bylo splněno ve všech bodech zadání. V průběhu implementace byly přidány některé funkčnosti, které se v návrhu nevyskytovali, jako například druhá jazyková verze. Je možné zvolit za jazyk češtinu nebo angličtinu.

Díky této práci jsem získal další znalosti o způsobech realizace webových aplikací pomocí různých technologií. Vyzkoušel jsem si základní implementaci různých typů Java appletů a jejich možnosti použití, které mi pomohli při návrhu samotného Java appletu. Dozvěděl jsem se také z prací kolegů, něco dalšího o zpracování a transformaci XML souboru a naučil jsem se implementovat uživatelské rozhraní pomocí knihovny Swing.

Literatura

[1] GEMELA, Lukáš. *Bakalářská práce – Transformace XML dokumentu do podoby logického programu.* Západočeská univerzita v Plzni. Fakulta aplikovaných věd. Katedra informatiky a výpočetní techniky. Plzeň, 2011. [cit. 2011-04-25]

[2] SCHNEIDER, František. *Bakalářská práce – Zpracování XML dokumentů pro potřeby sémantického webu.* Západočeská univerzita v Plzni. Fakulta aplikovaných věd. Katedra informatiky a výpočetní techniky. Plzeň, 2011. [cit. 2011-04-25]

[3] Seriál o XML pro Softwarové noviny. KOSEK, Jiří. *Vše o WWW* [online]. 2000. [cit. 2012-04-25]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/index.html

[4] Programování v Prologu. RUBÁČEK, Filip. UNIVERZITA HRADCE KRÁLOVÉ, Fakulta informatiky a managementu. *Instinctively Rescued Information Server* [online]. Hradec Králové, 1999-2005 [cit. 2012-04-25].
Dostupné z: http://iris.uhk.cz/logpro/index.html

[5**]** JANOVSKÝ, Dušan. *Jak psát web* [online]. 2003 [cit. 2012-04-25].
Dostupné z: http://www.jakpsatweb.cz/

[6] FALTÝNEK, Jakub. Návrh a tvorba webových aplikací s využitím WebML a ASP.NET [online]. 2011 [cit. 2012-04-25]. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Jan Konečný. Dostupné z: <http://theses.cz/id/0i2y4c/>.

[7] ŠRUTKA, Petr. Tvorba dynamických www stránek s využitím PHP [online]. 2010. [cit. 2012-04-25]. Bakalářská práce. Masarykova univerzita, Fakulta sportovních studií. Vedoucí práce Martin Dosedla. Dostupné z: <http://is.muni.cz/th/213630/fsps_b/>.

[8] AJAX - návod pro začátečníky. ZRALÝ, Jiří. *Digitální citrón* [online]. 6. 11. 2005. [cit. 2012-04-25].
Dostupné z: http://citron.blueboard.cz/clanek-239-ajax-navod-pro zacatecniky.html

[9] KOTALA, Zdeněk; TOMAN, Petr. *Studentský informační bulletin* [online]. Plzeň, 2001. [editováno 2001-02-04; 19:33]. Západočeská univerzita v Plzni. Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. [cit. 2012-04-25].
Dostupné z: <http://v1.dione.zcu.cz/java/sbornik/17.html>

[10] OLEXIY, Alexander PROKHORENKO. Creating a Trusted Applet with Local File System Access Rights. *Http://www.developer.com* [online]. 26. 1. 2004. [cit. 2012-04-22].
Dostupné z: http://www.developer.com/java/other/article.php/3303561/Creating-a-Trusted-Applet-with-Local-File-System-Access-Rights.htm

[11] Java Applet krok za krokem. MORKES, David. ZONER SOFTWARE, a.s. *Http://interval.cz*: *Programování* [online]. 27. 09. 2002. 2002 [cit. 2012-04-25]. Dostupné z: http://interval.cz/clanky/java-applet-krok-za-krokem/

[12] *Applety*. Studijní materiál. Vysoká škola ekonomická v Praze. [online]. Dostupné z: <java.vse.cz/pdf/java-applety.pdf>. [cit. 2012-04-25]

[13] The KeyStore Class(Java Security). OAKS, Scott. O'REILLY & ASSOCIATES. *Java Security* [online]. 1998 [cit. 2012-04-25]. Dostupné z: http://docstore.mik.ua/orelly/java-ent/security/ch11_02.htm

[14] *Grafické uživatelské rozhraní* [online]. Studijní materiál. Vysoká škola ekonomická v Praze. [cit. 2012-04-25]. Dostupné z: http://java.vse.cz/pdf/java-graficke_uzivatelske_rozhrani.pdf

[15] The Java Tutorials: A GroupLayout Example. *Http://docs.oracle.com* [online]. 1995-2012 [cit. 2012-04-22]. Dostupné z: http://docs.oracle.com/javase/tutorial/uiswing/layout/groupExample.html

A Uživatelská příručka

Pro správné načtení HTML stránky, na které je Java applet definován, je třeba mít nainstalovanou Javu (konkrétně JDK 1.3 a vyšší). Nebo mít nainstalovaný Java plug-in v prohlížeči. HTML stránku (*applet.html*), ikony pro přepínání jazyků (*eng.jpg* a *cze.jpg*) a JAR archiv (*Applet.jar*) je třeba umístit do stejného adresáře. Tento adresář se musí nacházet v hierarchii webového serveru (např. nejznámější Apache) a jeho název je předem definovaný (nejčastěji www/). Pokud chceme applet spouštět lokálně, musí být na stoji uživatele webový server nainstalován. Návod na instalaci webového serveru Apache obsahuje následující odkaz:

http://www.geeksengine.com/article/install-apache2-on-windows.html

Pokud chceme, aby applet byl umístěn na webu a volně přístupný všem zájemcům, je třeba ho umístit na webhosting[1]. Má tu výhodu, že nemusíme nic dalšího instalovat. Stačí se zdarma zaregistrovat (např. www.ic.cz) a soubory nahrát. Nebo, pro lepší služby a více potřebného diskového prostoru, zajistit webhosting placený. V případě, že se bude JAR archiv modifikovat a bude znovu rekompilován, je nutným krokem k prvnímu spuštění appletu digitální podpis. Jeho realizaci popisuje následující subkapitola. Pokud uživatel nebude modifikovat JAR archiv, nemusí se následující subkapitolou pro podepsání appletu zabývat.

Podepsání appletu

V konzoli je nutné nejprve vygenerovat certifikační klíč příkazem:

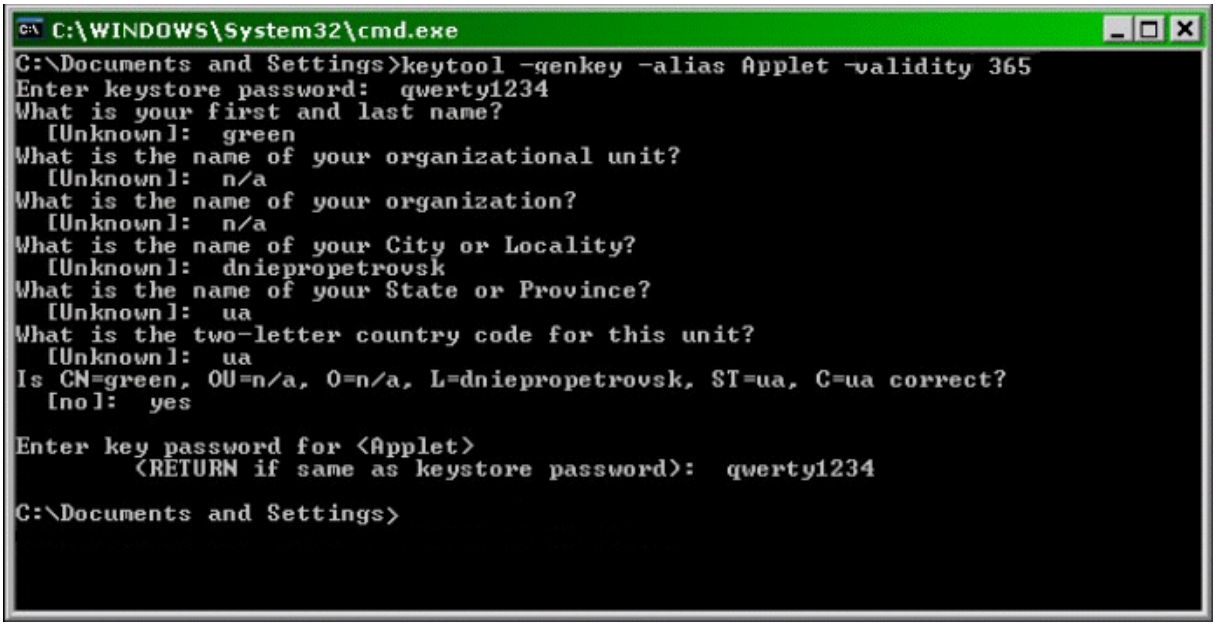
```
keytool -genkey -alias Applet -validity 365
```

Parametry tohoto příkazu jsou následující:

- `-genkey` – parametr pro vygenerování klíče.
- `-alias` – Název odkazu pro nově vygenerovaný klíč. Podle názvu se nalezne ten správný klíč, při podpisu v databázi certifikačních klíčů.
- `-validity xxx` – "xxx" je doba ve dnech, po kterou bude klíč aktivní. Defaultně je nastaveno dnů 180.

Po zadání tohoto příkazu, bude uživatel vyzván k vyplnění údajů zobrazených na Obr. A-1. Tyto údaje slouží k popisu certifikačního klíče. Heslo musí obsahovat více než 6 znaků. Jiné podmínky na vyplněné údaje nejsou.

[1] webhosting - pronájem prostoru pro webové stránky na cizím serveru



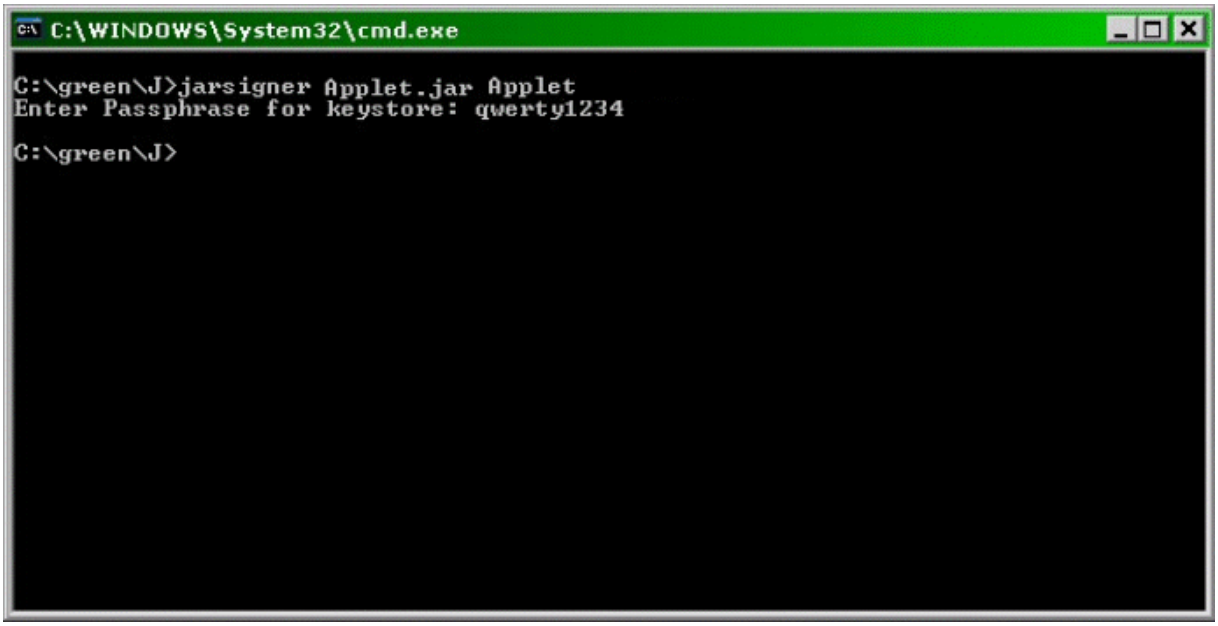
Obrázek A-1: Příklad vyplněných údajů k certifikačnímu klíči.

Po vygenerování certifikačního klíče, je možno provést proceduru podepsání appletu příkazem:

```
jarsigner Applet.jar Applet
```

Parametry tohoto příkazu jsou následující:
`Applet.jar` – název podepisovaného JAR archivu
`Applet` – název odkazu certifikačního klíče do databáze, kterým bude archiv podepsán

Dále (Obr. A-2) je uživatel vyzván, aby zadal heslo, stejné jako při generování certifikačního klíče. A tím je applet digitálně podepsaný a připraven ke spuštění.



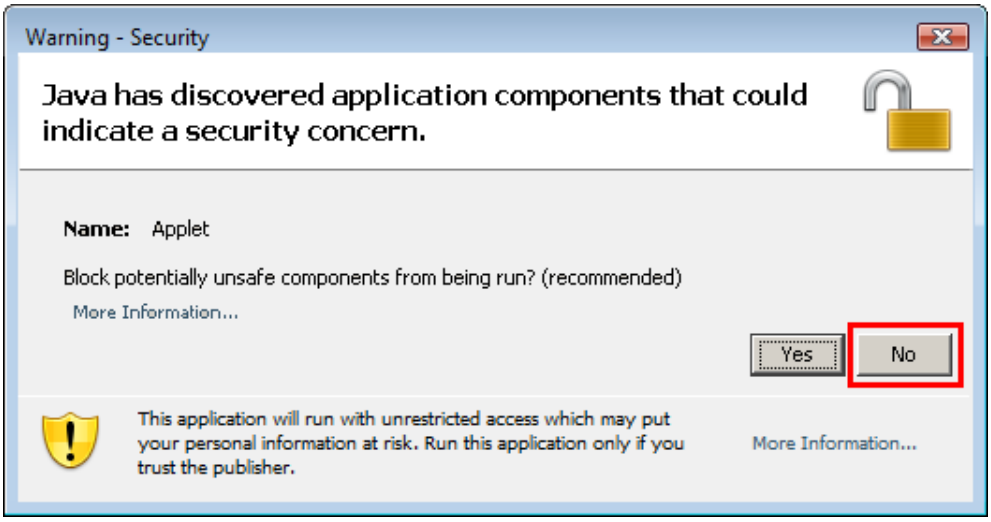
Obrázek A-2: Proces podepsání appletu.

Spuštění appletu

Provede se automaticky s načtením stránky appletu. Tedy zadáním odkazu umístění HTML stránky do webového prohlížeče. Při načítaní stránky se zobrazí postupně dialogová okna na Obr. A-3 a Obr. A-4, které je třeba správně potvrdit. Obě tyto okna se zobrazují z již zmíněných bezpečnostních důvodů. V prvním okně, potvrzením tlačítka "Run" uživatel souhlasí, že applet, který chce zobrazit, má zmíněná práva přesahující běžná práva appletů.

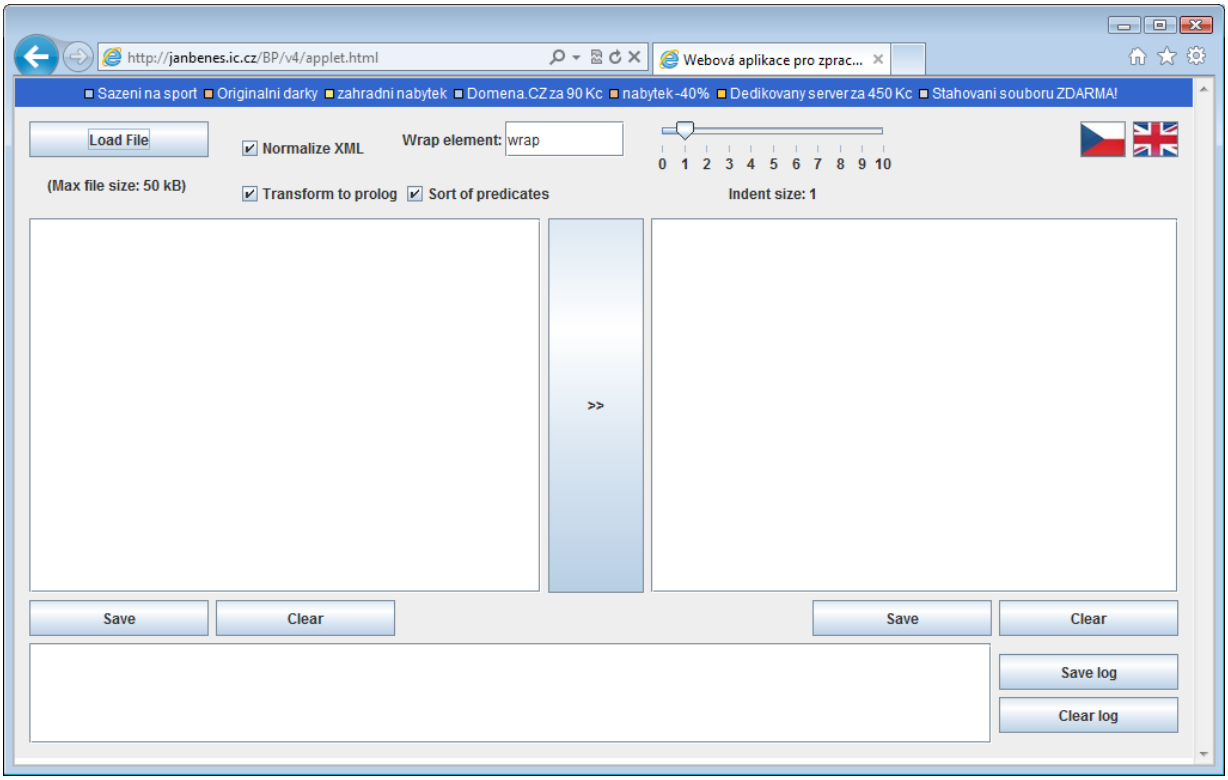
Obrázek A-3: Potvrzení důvěryhodnosti appletu.

Zaškrtnutím volby "Always trust content from this publisher" bude důvěryhodnost appletu potvrzena i pro další spuštění. Například při znovunačtení stránky. Druhé okno (Obr. A-4) nabízí možnost blokovat potenciálně nebezpečné komponenty, které chce aplikace použít. Jedná se ale pouze o ikony vlajek, umožňující přepínat na jinou jazykovou verzi. Proto pro blokovaní odmítneme tlačítkem "No".



Obrázek A-4: Potvrzení důvěryhodnosti appletu.

Applet při prvním spuštění je vidět na Obr. A-5.

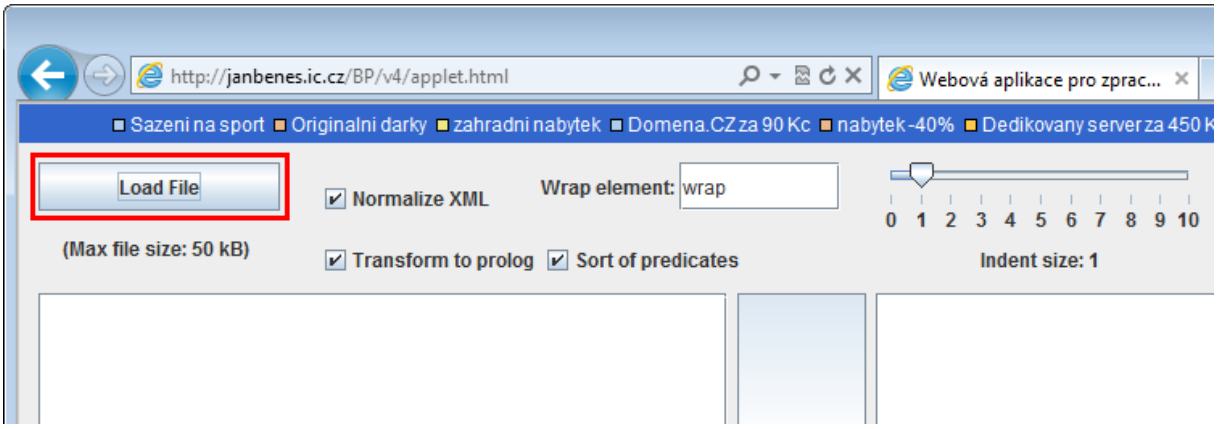


Obrázek A-5: Applet při prvním spuštění.

Ovladání appletu

Ovládáním appletu bude provázet následující série screenů, která bude zachycovat už jen důležité části appletu, jenž se budou zrovna popisovat. Nejprve bude ukázán proces normalizace a poté proces transformace, aby byla dostatečně demonstrována funkčnost obou částí.

Přes tlačítko “Load File” na Obr. A-6 je možné načíst vstupní XML soubor. Nebo je možné soubor zkopírovat přímo do levého editačního okna. Velikost nahraného souboru je omezena.

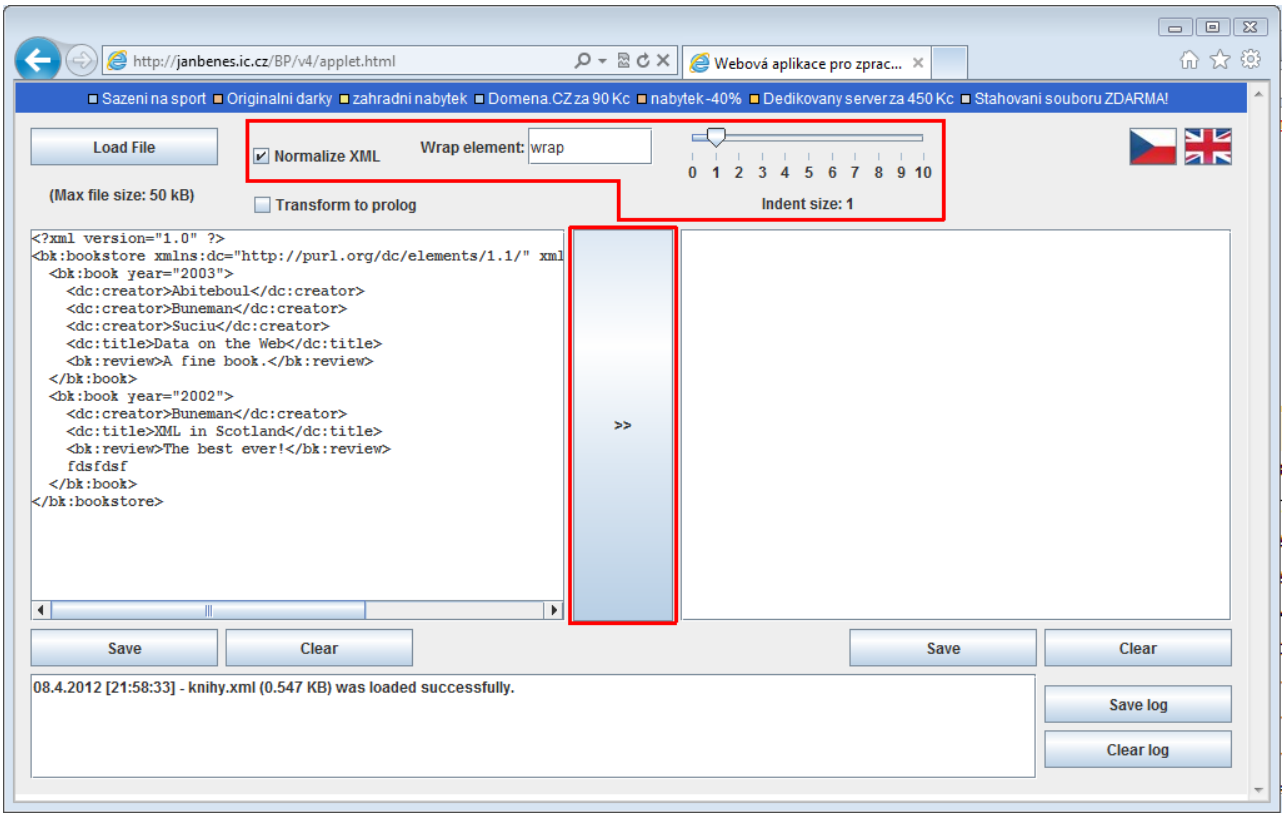


Obrázek A-6: Tlačitko “Load File“.

Na Obr. A-7 je situace úspěšného načtení vstupního souboru. V dolním okně, které má informační charakter, je zobrazen čas a velikost načteného vstupního souboru.

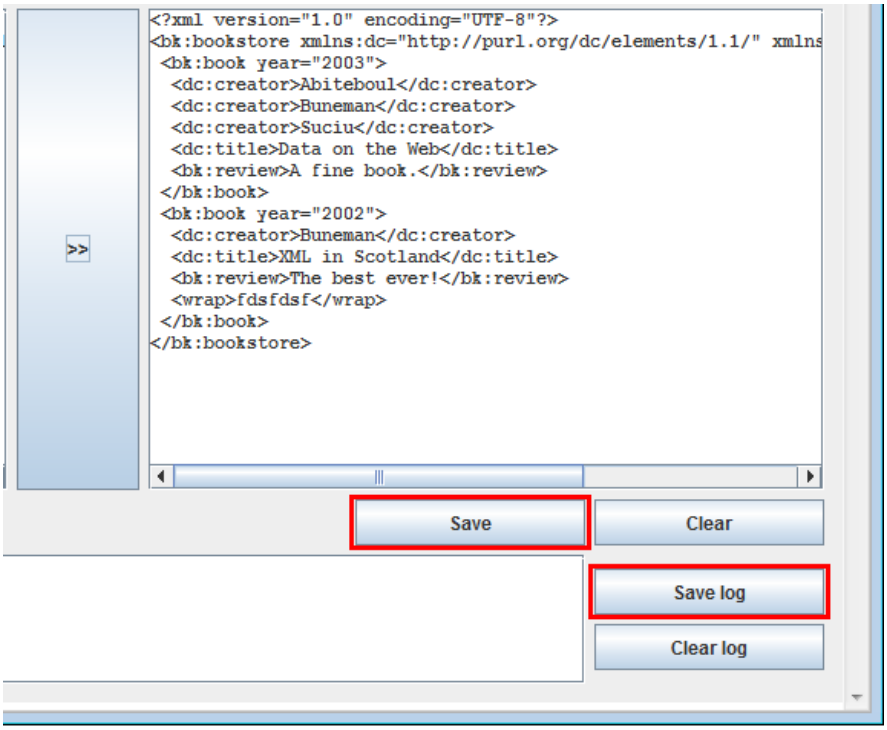
Dále se mohou nastavit vyznačené parametry normalizace. Zašktnutí checkboxu “Normalize XML“ umožňuje nastavení parametrů převodu. “Wrap element“ umožňuje nastavit název tagu, který obalí text, nenacházející se v žádném jiném tagu. “Indent size” umožnuje nastavit hodnotu od 1 do 10, která nastaví odsazení tagů od levého kraje, dle stupně zanoření.

Tlačítkem “>>“ se zahájí proces normalizace.



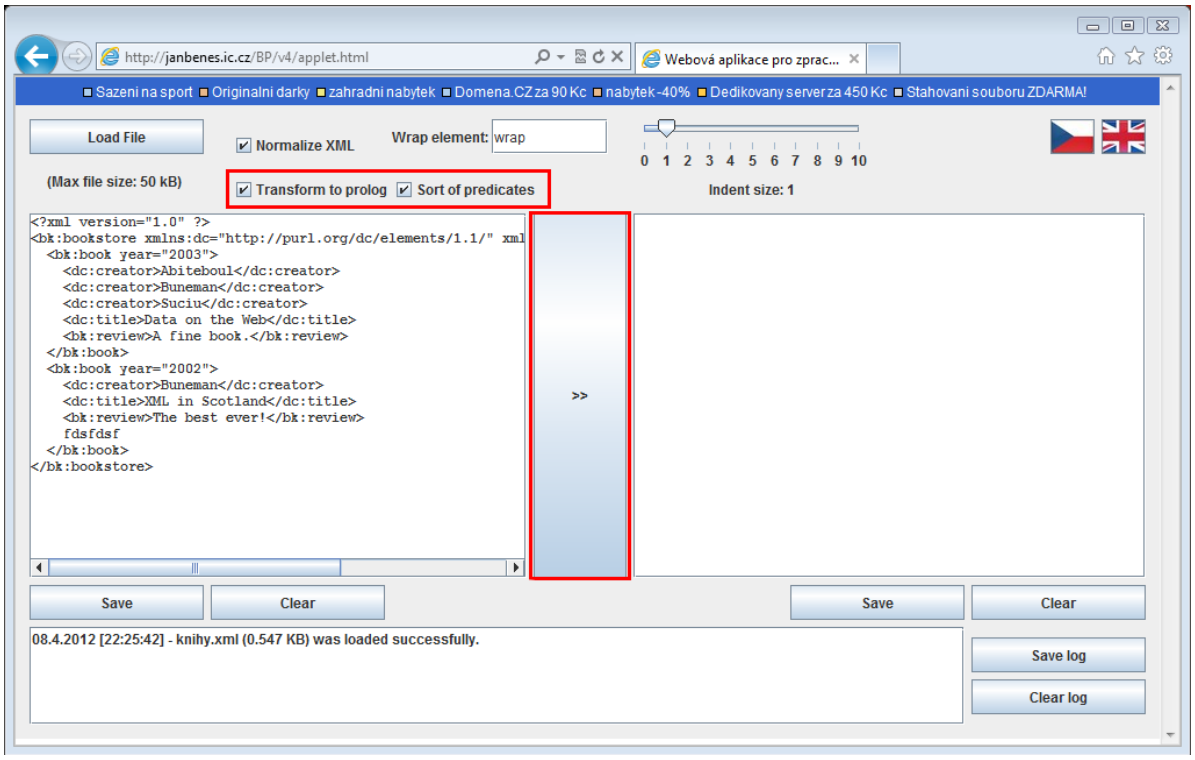
Obrázek A-7: Zobrazení načtení souboru a parametrů normalizace.

Na Obr. A-8 je v pravém okně vidět výsledek normalizace. Obsah tohoto okna lze uložit tlačítkem “Save” jako soubor. Stejně tak obsah dolního informačního okna přes tlačítko “Save log”.



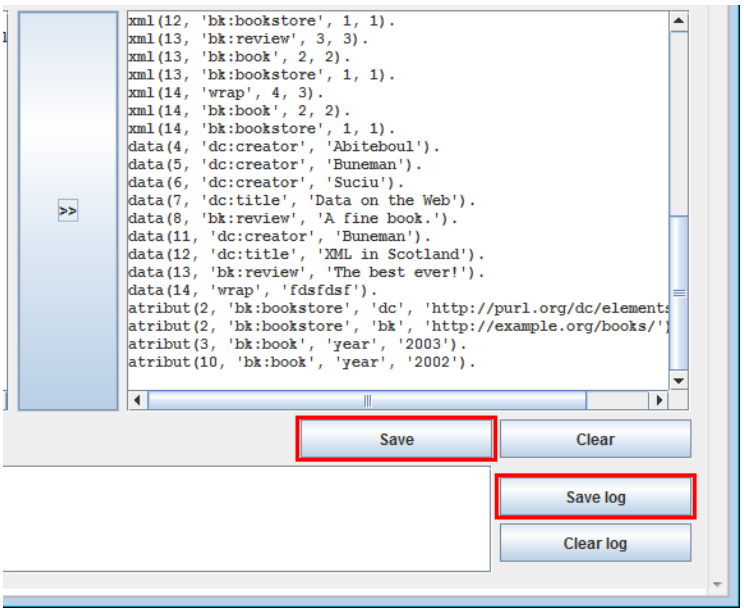
Obrázek A-8: Výsledek normalizace a tlačítka “Save”.

Následuje zobrazení procesu trasformace na Obr. A-9. Zašktnutí checkboxu “Trasform to prolog“, umožní kromě samotné transformace také možnost řazení predikátů jazyka prolog. A to přes zaškrtnutí checkboxu “Sort of predikates”. Řazení predikátů zajistí jejich lepší přehlednost.



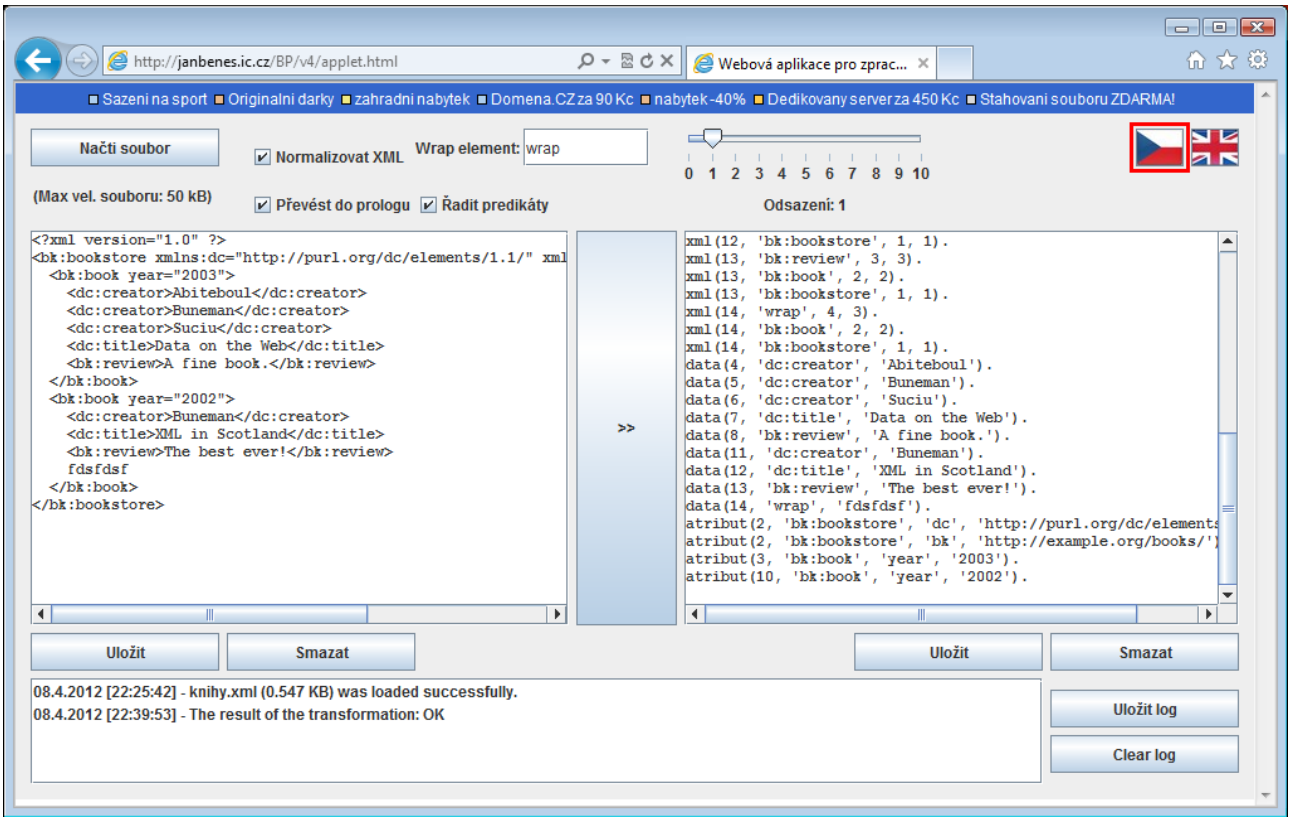
Obrázek A-9: Nastavení transformace.

Výsledek transformace spolu s obsahem informačního okna lze uložit přes tlačítka“Save”.



Obrázek A-10: Výsledek transformace.

Poslední srceen ukazuje možnost přepnutí na druhou jazykovou verzi, přes vyznačené tlačítko. Také je vidět, jak vypadá verze appletu v češtině.



Obrázek A-11: Ikona a zobrazení druhé jazykové verze (čeština).

B Obsah přiloženého média

Součástí této práce je také i paměťové médium (CD), obsahující tyto adresáře a soubory:

- `readme.txt` – textový soubor, obsahující popis a význam jednotlivých adresářů na médiu
- `www/` – obsahuje spustitelný soubor `Applet.jar`, HTML stránku `applet.html`, na které je definován a ikony, umožňující přepínání mezi jazykovými verzemi – `eng.jpg` a `cze.jpg`
- `doc/` – obsahuje tento dokument (`JanBenes_BP.pdf`)
- `src/` – obsahuje zdrojové soubory aplikace
- `javadoc/` – obsahuje *Java doc* dokumentaci k mojí práci
- `test/`– obsahuje soubory různých velikostí, na kterých byl výsledný Java applet otestován