VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

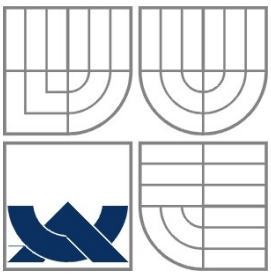
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FORMÁT XML PRO ZNAČKOVÁNÍ SLOVNÍKŮ

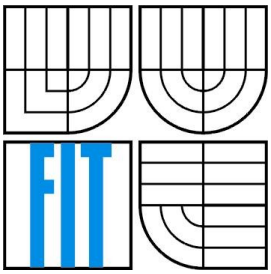
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JAN ŠUBRT
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FORMÁT XML PRO ZNAČKOVÁNÍ SLOVNÍKŮ

XML DICTIONARY TAGGING

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JAN ŠUBRT
AUTHOR

VEDOUCÍ PRÁCE — doc. RNDr. PAVEL SMRŽ, Ph.D.
SUPERVISOR

BRNO 2008

Abstrakt

Tato bakalářská práce popisuje systém správu slovníků ve formě XML. Zabývá se hlavně rodinou standardů XML (XSLT, XPath, XSD, ...) a poukazuje na kvalitní možnosti zpracování slovníkových dat uložené právě ve formě XML. V praktické části se zabývá převodem neanotovaných slovníkových dat do XML formy pomocí gramatik.

Klíčová slova

Slovník, XML, OLIF, MILE, DSL, ANLTR

Abstract

This Bachelor's thesis describes system for managing dictionaries in XML form. It also describes XML standard language family (XSLT, XPath, XSD, ...) and points out the qualities of computer processing of dictionaries stored in form of XML. The practical part shows translation of non-structured forms of dictionaries to XML using grammars.

Keywords

Dictionary, XML, OLIF, MILE, DSL, ANLTR

Citace

Jan Šubrt: Formát XML pro značkování slovníků, bakalářská práce, Brno, FIT VUT v Brně, 2008

Formát XML pro značkování slovníků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jméno Příjmení
Datum

Poděkování

Tímto bych rád poděkoval vedoucímu mé bakalářské práce doc. RNDr. Pavlovi Smržovi, Ph.D. za jeho ochotu, trpělivost a odbornou pomoc při vypracovávaní této práce.

Obsah

1 Úvod

Cílem této bakalářské práce je vytvořit systém, který je schopen převést různorodá slovníková data na jednotný formát založený na univerzálním značkovacím jazyce XML. Obzvláště se pak zabývá převodem klasických papírových slovníků v naskenované neanotované podobě.

První část této práce tvoří seznámení s použitými technologiemi. Nahlédneme zde do tajů světa XML a dozvíme se přednosti v počítačovém zpracování dat právě v tomto formátu. Je zde uveden přehled jazyka XML a souvisejících technologií, které jsou použity v navrhnutém systému.(XSD, XPath, XSLT, XML Binding...). Pro samotný proces převodu slovníkových dat, jenž tvoří jádro navrhnutého systému, jsou použity gramatiky napsané v jazyce ANTLR. Proto je v této části představen též nástroj ANTLR.

V další části této práce se seznámíme se slovníky a jejich reprezentacemi a to hlavně v podobě SGML a XML. Důraz je zde kladen na XML formát OLIF, který byl zvolen jako cílový formát.

V předposlední části je popsán realizovaný systém spolu s doporučeními na jeho použití a problémy s jeho implementací.

Závěr obsahuje zhodnocení celé práce, praktické výsledky a vlastní přínos.

2 Seznámení s užitými technologiemi

V této kapitole se seznámíme s hlavními technologiemi, na kterých staví navržený systém. Jedná se spíše o přehled, který by měl posloužit k lepšímu pochopení celého konceptu a objasnit proč byly použity právě tyto technologie.

2.1 Rodina standardů XML

Tato kapitola se klade za úkol vystihnout přednosti rodiny XML standardů, a to hlavně ve spojení s lehkostí počítačového zpracování dat uložených právě ve formátu XML a demonstrovat tak vhodnost použití XML pro zprávu slovníkových dat.

Svět XML je velmi bohatý a zahrnuje mnoho standardů pro definici obsahu XML, vyhledávání, transformaci, validaci apod.. V dnešní době navíc existuje celá řada technologií ba i celých frameworků, které s těmito standardy pracují a často poskytují kvalitní datový middleware.

2.1.1 Historie XML

XML [1] (eXtensible Markup Language*)* je univerzální značkovací jazyk od konsorcia **W3C** [0]. Snaha o vytvoření univerzálního značkovacího jazyka započala již v 70. letech minulého století. Na počátku 80. letech se objevil jazyk **GML** (Generalized Markup Language), z něhož v roce 1986 vychází **SGML** (Standart Generalized Markup Language*)*, ten umožňuje definovat další značkovací jazyky.

Z něho pak vychází i jazyk **HTML** (HyperText Markup Language*)*. Ten se v dnešní době využívá hlavně při tvorbě webových stránek. Ovšem jeho pevně dané značkovaní zabraňuje efektivnějšímu použití prohledávacích metod a slouží většinou čistě k prezentačním účelům. Text lze označit jako nadpis, odstavec, seznam nebo například obsah buňky. Další jeho nevýhodou oproti potřebám, které vedly k vzniku XML je skutečnost, že programy pro jeho zpracování musejí kontrolovat velké množství odchylek od standartu a jsou tím pádem velmi objemné. HTML totiž nevyžaduje striktní dodržování zápisu a umožňuje například křížení značek, nebo vynechání koncových značek. Proto časem vznikla snaha spojit kvalitu SGML s jednoduchostí HTML. Nový jazyk měl být podle specifikací strukturovaný, rozšiřitelný pomocí vlastních formátovacích značek, měl být přenositelný na různé platformy a ověřitelný k dané gramatice.

XML tedy spatřil světlo světa poprvé v roce 1998 jako důsledek snahy o stále se zvyšující nároky na kvalitu a kvantitu zpracovávaných a výměnu informací. XML nám vytváří jasně strukturovaný dokument, protože nemá žádné předdefinované značky a míra strukturování dokumentu je tedy neomezená. XML lze tedy považovat za jakýsi metajazyk, který používá značky k syntaktické konstrukci a popisuje tak jazyk jiný – spolu s definici struktury pak tvoří takzvaný XML standard. V dnešní době existuje mnoho organizací zabývajícími se vývojem XML standardů pro nejrůznější odvětví informačního světa. Mezi nejznámější patří **OASIS** [2] a **W3C** [0].

Aktuální verze XML je 1.1 a jedná se již o čtvrtou revizi této verze. Od 1.0 se liší v požadavcích na použité znaky v názvech elementů, atributů apod.. Verze 1.0 dovolovala pouze užívání znaků platných ve verzi Unicode 2.0, která neobsahuje všechny světové sady. Verze 1.1 tedy pouze upravuje standard aby byl politicky korektní, v praxi se ale prakticky nepoužívá.

2.1.2 Základy syntaxe XML

Jazyk XML je case-sensitive a jako jeden z prvních jazyků může obsahovat též Unicode znaky. Jeho základní struktura a pravidla jsou následující (celé znění standardu lze nalézt na [1])

Element:

Základním prvkem u XML je element. Každý element se skládá z hodnoty elementu a z tagu. Tag je počáteční a koncová značka ohraničující hodnotu elementu. Element může obsahovat atributy vyjadřující vlastnosti elementu. Hodnota atributu musí být uzavřená v uvozovkách.

```
<ElementTag atribut="hodnota atributu">hodnota elementu</ElementTag>
```

Pokud je element prázdný, lze jej zapsat zkrácenou notací

```
<ElementTag  />
```

Elementy do sebe lze nořit a tím vytvářet strukturu dokument.

```
<ElementTag>
        <VnořenýElement />
</Elementtag>
```

Elementy se nesmějí překrývat.

```
<ElementTag>
        <VnořenýElement></Elementtag>
<VnořenýElement />
```

Dokument musí obsahovat právě jeden kořenový element, který obsahuje všechny ostatní elementy.

Jmenné prostory:

Elementy a atributy lze kvalifikovat přidáním takzvaného jmenného prostou (z pravidla ve ve formě URI) a jednoznačně tak rozlišit konflikty v názvech těchto entit.

Prolog:

Na prvním místě v dokumentu musí být uveden takzvaný prolog, definující verzi a použité kódování.

```
<?xml version="1.0" encoding="utf-8"?>
```

Komentář:

V dokumentu lze použít komentáře pomocí následující syntaxe.

```
<!-- Komentář -->
<!--
        Více řádkový
        komentář
 -->
```

Escape sekvence:

Pro vyjádření speciálních znaků uvnitř dar lze následující escape sekvence.

&lt;	<	menší než
&gt;	>	vetší než
&amp;	&	ampresand
&apos;	'	apostrof
&quot;	“	uvozovka

Nebo uzavřít celý text do CDATA sekce.

```
<![CDATA["0 < 5"]]>
```

Well-formed dokument:

XML dokument je takzvaně well-formed (dobře formovaný), pokud splňuje všechny syntaktická pravidla XML. Tato vlastnost se nijak nevztahuje k samotnému obsahu, nebo struktuře dokumentu.

2.1.3 Definice a validace XML

Silnou stránkou jazyka XML je fakt, že obsahuje krom samotných dat i definici jejich struktury. Tím výrazně vylepšuje možnosti jeho počítačového zpracování. V praxi má tato vlastnost mnoho využití.

Tím základním je automatizovaný proces validace dokumentu oproti jeho definici (proces obstará XML parser, není nutné nic programovat). Dále pak například při výměně dat mezi systémy pomocí XML (popř. **WS/SOAP + XML Binding**), mohou tvořit tyto definice rozhraní mezi těmito systémy a pod..

Strukturu XML souboru lze definovat různými způsoby. Některé umožňují pouze samotné vyjádření struktury entit, jiné mohou obsahovat i jisté restrikce na jejich obsah. Některé definice umožňují pro návrh využít i objektovo-orientovaných technik (dědičnosti, zapouzdření, ...) a jmenných prostorů. Jednotlivé definice pak mohou obsahovat či vkládat definice jiné a používat i entity z jiných jmenných prostorů. Tímto se otevřel prostor pro vytváření takzvaných **XML standardů** a to především těch datových. Jak již bylo řečeno úvodem, v dnešní době existuje mnoho organizací zabývajícími se vývojem těchto standardů (**OASIS** [2], **W3C** [0], ...), které mají v praxi hojné využití.

2.1.3.1 DTD

DTD (Document Type Definition) je nejstarším jazykem pro definici struktury XML dokumentu. Pochází ještě z dob SGML a nevyužívá plně nových vlastností XML. Například v něm nelze používat jmenné prostory, nebo definovat restrikce obsahu entit apod. V dnešní době se prakticky používá již jen v legacy systémech.

Následující jednoduchá ukázka demonstruje definici DTD vloženou přímo do XML dokumentu. DTD lze též mít jako samostatný soubor a XML soubor s ním pouze asociovat.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE firma [
        <!-- Element "firma" je validním elementem
        a může obsahovat 0-N elementů "zamestnanec" -->
        <!ELEMENT firma (zamestnanec*)>
        <!-- Element "zamestnanec" je validním elementem a
```

```
        může obsahovat 1 element "jméno" a následně 0-1
        elementů "zamestnanec" -->
        <!ELEMENT zamestnanec (jmeno, osobni_cislo?)>
        <!-- Element "jmeno" je validním elementem
        a může obsahovat PCDATA (Parsed Charcter Data) -->
        <!ELEMENT jmeno       (#PCDATA)>
        <!-- Element "osobni_cislo" je validním elementem
        a může obsahovat PCDATA (Parsed Charcter Data) -->
        <!ELEMENT osobni_cislo  (#PCDATA)>
]>
<firma>
        <zamestnanec>
                <jmeno>Jan Novák</jmeno>
                <osobni_cislo>1</osobni_cislo>
        </zamestnanec>
        <zamestnanec>
                <jmeno>Jiří Šebesta</jmeno>
        </zamestnanec>
</firma>
```

Více informací o jazyce DTD lze nalést na [3].

2.1.3.2 XSD

XSD (Xml Schema Definition) je XML standard od konsorcia W3C pro definici struktury, obsahu a sémantiky XML dokumentů. Byl vyvinut jako odpověď na nedostatky jazyka DTD. Oproti DTD má následující výhody:

- Jedná se o XML standard, tím pádem je pro zpracováni XSD možno použit standardní XML nástroje
- Podporuje jmenné prostory
- Nabízí možnost definovat restrikce obsahu elementů a atributů
- Používá datové typy
- Používá principy OOP (dědičnost, zapouzdření, ...)
- Schémata lze do sebe vkládat
- **...**

Následující ukázka definuje obdobnou strukturu jako u příkladu na DTD, obsahuje však navíc definici datových typů některých elementů.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="firma"> <!-- element "firma" -->
   <xs:complexType> <!-- obsahuje -->
    <xs:sequence>
     <!-- 0 az n elementu "zamestanec"-->
     <xs:element name="zamestanec" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType> <!-- obsahuje -->
       <xs:sequence>
        <!-- 1 element "jmeno" typu string-->
        <xs:element name="jmeno" type="xs:string" />
        <!-- 0-1 element "osobni cislo typu int-->
        <xs:element name="osobni_cislo" type="xs:int" minOccurs="0" />
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

2.1.3.3 Relax NG

Zmínit si rozhodně zaslouží i jazyk Relax NG [5]. Ten je založeno na vzorech. Celé schéma je vzorem dokumentu. Vzor se přitom skládá ze vzorů pro elementy, atributy a textové uzly. Síla těchto vzorů je v tom, že je lze libovolně kombinovat do skupin, mohou být volitelné a s různým počtem opakování. Tato vlastnost umožňuje vypsat i složitě strukturovaný dokument velmi jednoduše.

V Relax NG je výbornou vlastností kompaktní syntaxe. Ta nám umožňuje zapsat schéma v textové syntaxi, která je mnohem úspornější a vhodnější pro ruční zpracování, než ta založená na XML. Kompaktní a XML syntax je parserem považována za ekvivalentní a je mezi sebou převoditelná.

2.1.3.4 Schematron

Schematron je validačním jazykem pro XML založeným na odlišné filozofii než ostatní jazyky jako jsou DTD či XSD. Schematron se stal součástí standardu ISO/IEC 19757 – **DSDL** (Document Schema Definition Languages) [6].

Schematron dokáže usuzovat na přítomnost jistých vzorů v XML stromu. Jedná se o jednoduchý, přesto velmi mocný strukturální schématický jazyk. Vzory jsou typicky popsány pomoci jazyka XPath [7].

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
        <pattern name="Osobní číslo musí být vetší než 1000">
                <rule context="zamestnanec">
                        <assert test="osobni_cislo &lt; 1000">
                        Osobní číslo musí být menší než 1000.
                        </assert>
                </rule>
        </pattern>
</schema>
```

2.1.4 Transformace a prezentace XML

XSL family (eXtensible Stylesheet Language family) [8] je rodina standardů od konsorcia W3C, pro definování XML transformací a prezentační vrstvy. Skládá se ze 3 částí:

- **XSLT** - XSL Transformací
- Jazyka **XPath**, který používa XSLT pro adresování částí dokumentu
- **XSL-FO –** XML slovníku pro specifikaci formátovací sémantiky

2.1.4.1 XSLT

XSLT (XSL Transformations) je jazyk sloužící k transformaci XML dokumentů [9]. Pomocí něj lze vytvářet styly, definující převod XML dokumentu do jiného XML s odlišnou strukturou, převod do HTML kódu, nebo do obyčejného textového souboru. Tento jazyk by navržen pro použití jako součást XSL spolu s formátovacími slovníky. Lze ho však použít nezávisle na této technologii pro libovolné transformace. Například pro vzájemný převod různých XML formátů, nebo pomocí něj lze generovat SQL příkazy, umožňující přidávající obsah XML do databáze apod..

2.1.4.2 XPath

XPath (XML Path Language) je jazyk, který slouží k adresování částí XML dokumentu [7]. Jeho nejzákladnějším úkolem je především vyjádřit relativní cestu od nějakého XML uzlu k jinému elementu nebo atributu. Pomocí tohoto jazyka lze z XML dokumentu vybírat právě tyto elementy a pracovat s jejich hodnotami a atributy.

XPath byl navržen, pro využití v XSLT a **XLink** (Xml LINKing language) [10]. Využívá se ale hojně všude tam, kde je potřeba vyhledávat v XML struktuře nějaká data. Chceme-li třeba vybrat určitou množinu dat, postačí nám k tomu jediný Xpath příkaz a (XML parser se již o zbývající vyhledávání a vyhodnocení podmínek postará sám).

2.1.5 XML API

2.1.5.1 Sax

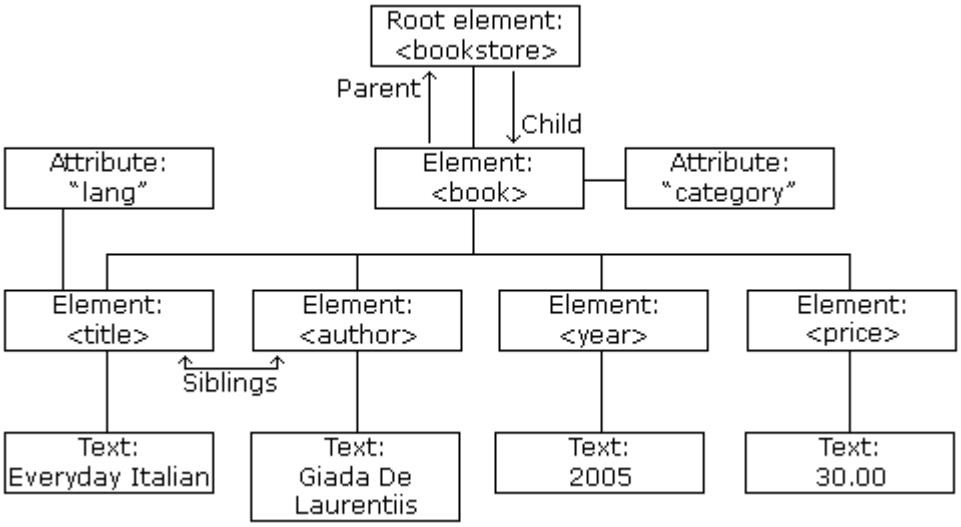
SAX (Simple Api for Xml) je jedním z nejznámějších API určené pro načítání XML dokumentů [11]. Původně se jednalo pouze o JAVA API, které se později rozšířilo i na jiné platformy a stalo se de fakto normou, kterou implementuje mnoho dnešních XML parserů.

Jedná se o stream-based rozhraní řízené událostmi. V praxi to znamená, ze uživatel si nadefinuje množinu call-backových metod, které budou vyvolány, dojde-li k určité události v průběhu procesu parsování. Mezi takové události může patřit načtení začátku/konec textového uzlu, elementu, komentáře apod.. Důležitý je též fakt, že proces stream-based parsingu je jednosměrný, dokument se tedy musí zpracovat v jednom průchodu zcela, nelze se vracet ani skákat dopředu. Díký této vlastnosti je SAX velmi nenáročný na pamět, avšak neposkytuje takový komfort jako konkurenční API jakým je například DOM či XML Binding.

2.1.5.2 DOM

DOM (Documnet Object Model) je standardem od konsorcia W3C [12]. DOM reprezentuje XML rozhraní nezávislé jak na platformě, tak na použitém programovacím jazyce. Toto rozhraní umožňuje dynamicky přistupovat k jednotlivým částem dokumentu.

DOM pracuje s XML dokumentem, jako se stromovou strukturou s elementy, atributy a obsahem jako uzly. Jeho největší výhodou oproti SAX je fakt, že se lze po dokumentu libovolně pohybovat – průchod stromem. Je však mnohem pomalejší a náročnější na paměť (dokument se z pravidla musí do paměti načíst celý).



Ilustrace 1: DOM - stromová struktura

2.1.5.3 XML Data Binding

XML Data Binding přístup k manipulaci s XML daty, kdy jsou tyto data v paměti reprezentován jako klasické objekty. S těmito objekty se potom manipuluje mnohem přirozenější cestou, než při použití DOM. Proces převodu z XML formy na objekty se nazývá unmarshaling. Serializace objektů zpět do XML formy pak marshaling.

Zpravidla se třídy pro binding generují ze XSD či DTD schémat, ale lze je vytvořit ručně za pomocí XPath jazyka. XML Data Binding se hojné využívá ve světě **SOA/J2EE** (Service Orientated Architecture), lze ho však využít jako samostatnou technologii. Některé XML Binding frameworky poskytují i některé pokročilé funkce, jako například perzistování objektů do relačních databází apod..

2.2 ANTLR

V této kapitole si představíme nástroj **ANTLR** (ANother Tool for Language Recognition) [13] , jakožto sofistikovaným frameworkem pro návrh, vývoj a testování překladačů a to hlavně pro takzvané doménově specifické jazyky - **DSLs** (Domain Specific Languages), mezi které řadíme i slovníkové datové formáty.

2.2.1 DSLs

DSLs jsou obecně vysokoúrovňové jazyky, které jsou šity na míru specifickým úkolům. Tyto jazyky jsou z pravidla navrženy tak, aby jejich užití bylo zvláště efektivní v určité doméně. Mezi DSLs řadíme široký okruh aplikací, které se na první pohled pro mnohé nemusí jevit jako jazyky. DSLs

zahrnuje například datové formáty, konfigurační soubory, sítové protokoly, proteinové vzory, genové sekvence, jazyky pro řízení vesmírných sond, regulární výrazy a všechny doménově specifické programovací jazyky.

DSLs mají své jisté své místo na slunci a jejich význam stále roste a to především v procesu softwarového vývoje. Umožňují totiž mnohem efektivněji a přirozeněji vyjádřit specifický problém, než při použití obecný jazyků, do kterých se později často překládají. Například NASA používá doménově specifický příkazový jazyk k dosažení lepší spolehlivosti, redukci rizika, nákladů a zvýšení rychlosti vývoje. Dokonce i první Apollo počítačový naváděcí systém z roku 1960 používal DSL, který podporoval vektorové výpočty [14].

2.2.2 ANTLR popis a historie

ANTLR je nástupcem nástroje PCCTS (Purdue Compiler Construction Tool Set) , který byl prvně vyvinut již v roce 1989. O jeho současný vývoj a údržbu se aktivně stará prof. Terrence Parr ze San Franciské Univezity. Nejnovější verze v3 je distribuována pod BSD licencí [15].

ANTLR je především parser generátorem, generujícím LL(*) třídu parserů. Na rozdíl od obdobných nástrojů, nepoužívá pro definici gramatiky regulární výrazy, ale jazyk podobný EBNF notaci obohacený o akce a predikáty zapsané v cílovém jazyce. ANTLR v současné době podporuje následující cílové jazyky: C, C++, JAVA, Python, Ruby, C#, a mnohé další [16].

ANTLR verze v3 je již celá napsána v předchozí verzi v2 a cílovém jazyce JAVA. Oproti starším verzím obsahuje vynikající podporu pro konstrukci stromů, jejich procházení, zotavení z chyb, chybová hlášení a ladění. Součástí ANTLR je i sofistikované vývojové prostředí **ANTLR Works** [17], jež umožňuje efektivní návrh, analýzu a vizuální ladění gramatik. K dispozici jsou i mnohé pluginy pro Eclipse a IntelliJ.

2.2.3 ANTLR Works

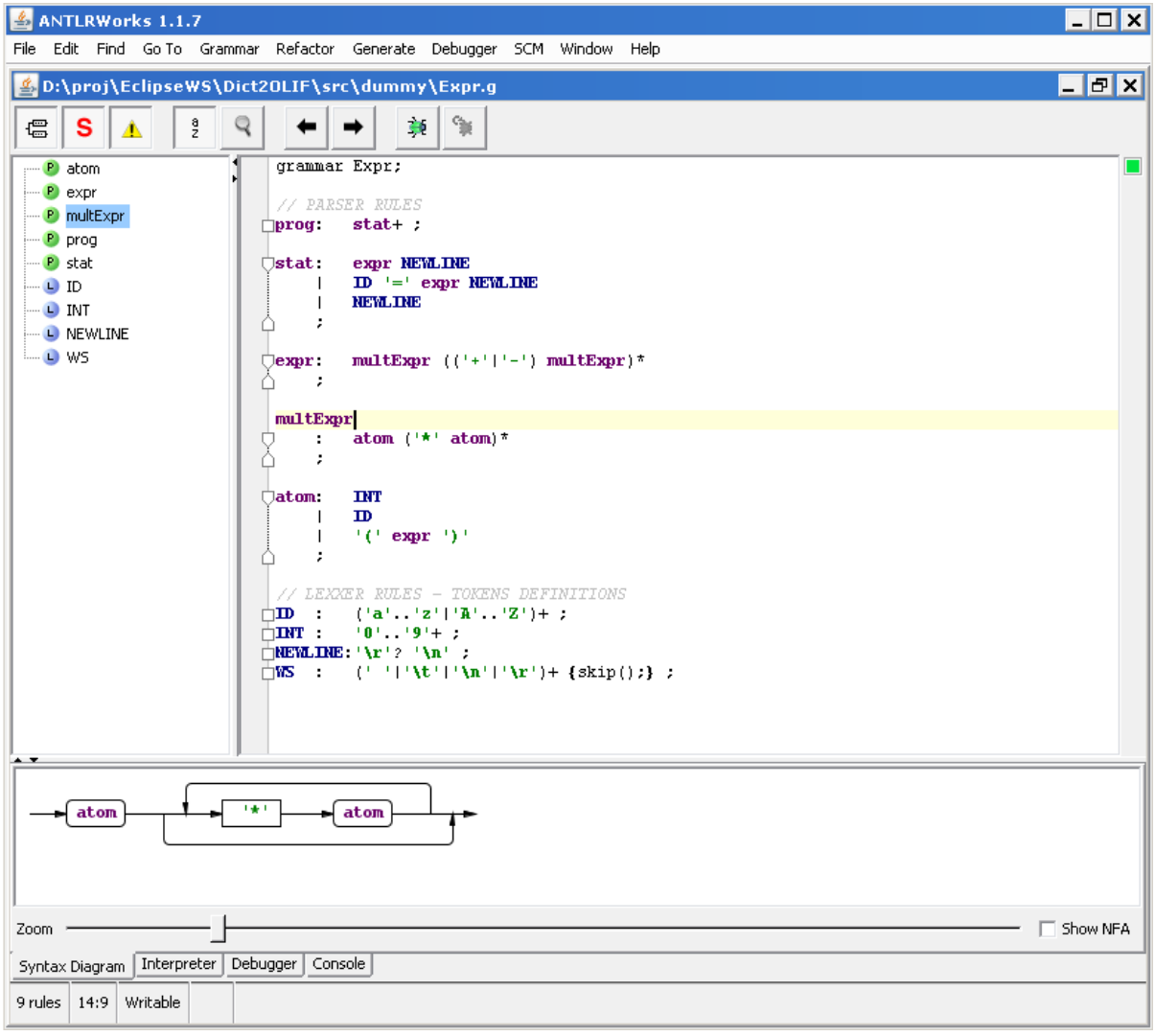
ANTLR Works je vývojové prostředí pro ANTLR v3 gramatiky. Je napsáno v programovacím jazyku JAVA a je distribuováno pod BSD licencí. Jeho autory jsou Jean Bovet Terence a Parr.

Toto prostředí kombinuje excelentní gramatiky-znalý editor spolu s interpretrem a debuggerem a umožňuje tak odhalit chyby, na které by se manuálně jen stěží přicházelo. Současná verze ANTLR Works 1.1.7 obsahuje následující hlavní funkce:

- Gramatiky-znalý editor
- Zobrazení syntaktického diagramu gramatiky
- Interpretr pro rychlé prototypováni gramatiky (pouze pro JAVA cílový jazyk)*
- Vizuální debugger pro izolování chyb v gramatice
- Zvýrazňovač nedeterministických cest v syntaktickém diagramu gramatiky

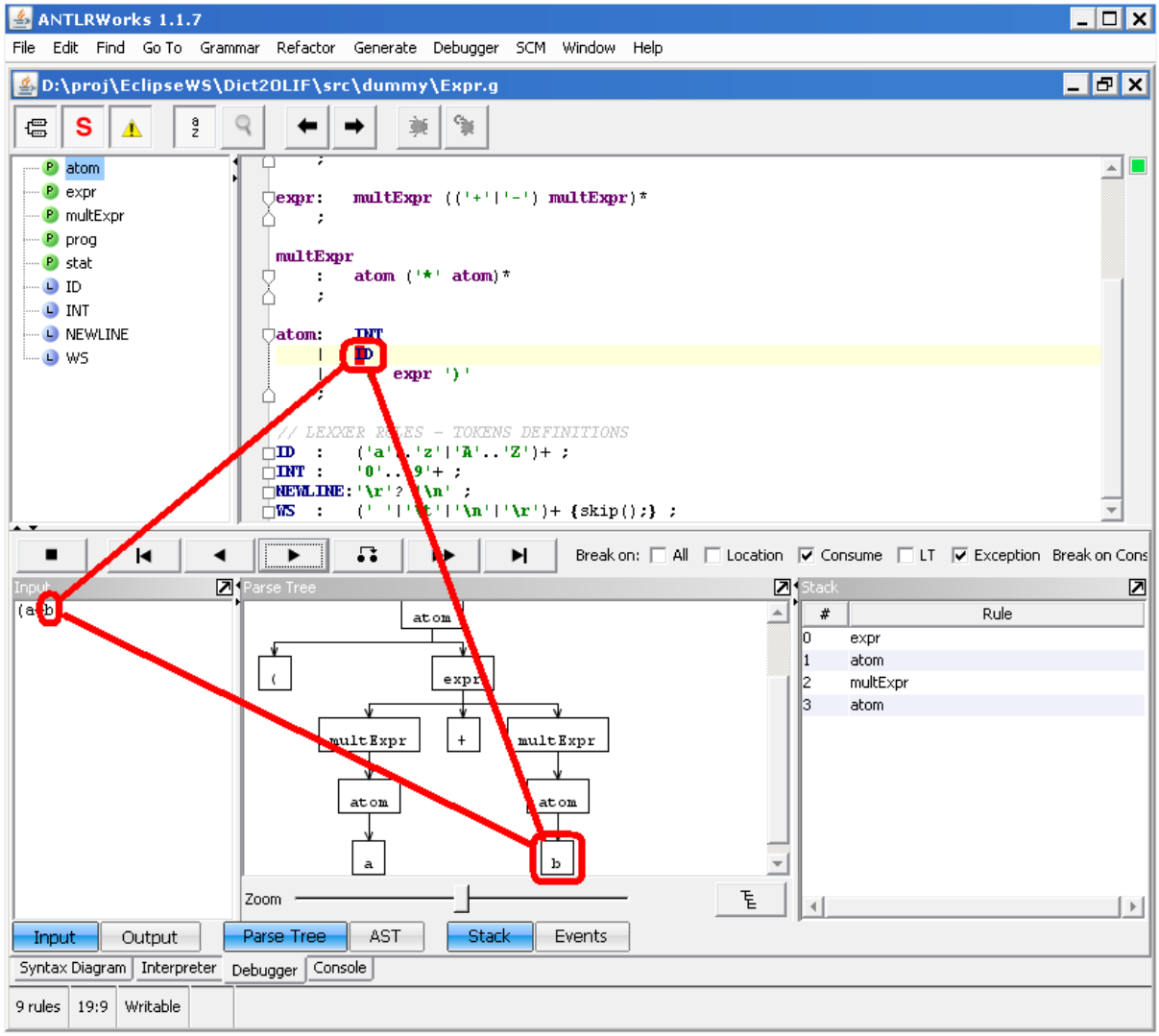
- DFA vizualizaci
- Refaktoring (na základě vzorů – např. odstranění levé rekurze)
- Dynamické zobrazení Parse tree
- Dynamické zobrazení AST

*Vzhledem k tomu že ANTLR works komunikují s bežícími parsery pomocí soketů, je možné možné vestavěný interpretr obejít a odladit pomocí debuggeru libovolný cílový jazyk (za předpokladu, runtime knihovna obsahuje potřebný podpůrný kód).



Ilustrace 2: ANTLR Works - vývojové prostředí

Na tomto obrázku je zobrazeno základní vývojové prostředí. Pro aktuální vybrané pravidlo, je zobrazen syntaktický diagram.



Ilustrace 3: ANTLR Works - ladění

Tento obrázek zobrazuje běh debuggeru. Všechny zobrazení jsou synchronizovány – aktuální token, gramatické pravidlo, Parse tree a Stack.

3 Reprezentace slovníkových dat

Existuje mnoho různých způsobů, jak mohou být textová a lexikální data anotována a strukturována v závislosti na teoretické složitosti a na souvisejících nástrojích. Nejčastěji používané formáty pro reprezentaci dat jsou založeny na SGML, XML a RDF standardech. Vetší pozornost bude věnována formátu OLIF, jenž byl zvolen cílový formát pro realizovaný systém.

3.1 Úvod do slovníků

Slovník je nejčastěji abecedně řazený seznam slovní zásoby, vysvětlující slova z různých hledisek. [18]. Ve většině slovníků jsou slova zachycena pouze ve svém základním tvaru, tzv. Lemmatu. Slovníky se vyskytují tradičně nejčastěji v knižní podobě. V poslední době se však objevují i digitální slovníky, dostupné na CD nebo na internetu.

3.1.1 Typy slovníků

Podle rozsahu se slovníky často dělí na:

- slovníky malé, kapesní (do 10 000 hesel)
- slovníky střední (50 000 až 60 000 hesel)
- slovníky velké (nad 60 000 hesel)

Podle typu se rozdělují na:

- slovníky výkladové (jednojazyčné) - Jsou napsány celé v jednom jazyce, u každého slova lze nalézt informace ve stejném jazyku, dále je lze rozdělit na:
  - slovníky současného jazyka (významové, pravopisné, frazeologické, slovníky synonym, slovníky cizích slov atd.)
  - slovníky jednotlivých historických období a slovníky etymologické
  - slovníky popisující slovní zásobu profesních skupin (např. filozofický slovník, biblický slovník apod.)
  - speciální speciální (retrográdní, frekvenční, valenční atd.)
- slovníky překladové (vícejazyčné) - Slouží pro překlad z jednoho jazyka do druhého, ke slovům jednoho jazyka obsahují jeho překlad v druhém jazyce, často i s výslovností, frázemi, nebo jinými doprovodnými informacemi. Některé větší překladové slovníky obsahují i druhou část, ve kterém jsou slova pro zpětný překlad z druhého jazyka do prvního. Tyto slovníky mohou být i specializované, například se omezovat jen na odborné termíny z některé oblasti.

3.2 SGML/XML

SGML a XML jsou široce používané standardy pro anotování textové struktury. XML předčil SGML v mnoha ohledech, přesto existuje stále velké množství zdrojů které používají stále SGML.

3.2.1 TEI

V roce 1994 vnikl formát **TEI** (Text Encoding Initiative) [19] jako množina detailních doporučení jak reprezentovat spoustu typů psaného a mluveného materiálu za použití rozšířitelného SGML frameworku. Tento formát měl též vliv na lexikální projekty jako PAROLE nebo EAGLES a příbuzné standardy. Následující příklad TEI ilustruje doporučení, jak reprezentovat záznam z klasického papírového slovníku. Záznam obsahuje různé informace o syntaxi, sémantice, fonologii apod.

```
<entry>
        <form>
                <orth>competitor</orth> <!-- ortografie (pravopis) -->
                <hyph>com|peti|tor</hyph> <!-- slabiky -->
                <pron>k@m"petit@(r)</pron> <!-- vyslovnost -->
        </form>
        <gramGrp>
                <pos>n</pos> <!-- Part Of Speach ~= slovní druh -->
        </gramGrp>
        <def>person who competes.</def> <!-- definice -->
</entry>
```

TEI specifikace byla adoptována a rozšířena projektem **CONCEDE** (Consortium for Central European Dictionary Encoding) [20].

3.2.2 OLIF

OLIF (Open Lexicon Interchange Format) je otevřený XML datový standard určen pro reprezentaci jak překladových, tak výkladových slovníkových dat. OLIF vytvořen konsorciem OLIF2, organizací

dodavatelů jazykových technologií, a výzkumných institucí. Je šířen zcela zdarma jako XSD (DTD) [21].

Jeho vývoj započal v roce 1990 jako jedna z alternativ pro výměnu dat mezi výpočetními lexikony a terminologickými databázemi. OLIF se vyvinul na standard, který nabízí široké spektrum podpory pro jazykovou výměnu dat a jejich reprezentaci a stal se velmi používaným výměnným formátem.

3.2.2.1 Struktura

Dokument ve formátu OLIF je dělen několika částí. Tou první je element **header** (hlavička), ve kterém jsou uloženy základní informace o dokumentu, jako například vlastník, datum vytvoření, kódování apod.. Header je povinný. Následující příklad ukazuje minimální možnou verzi headru.

```
<header CreaTool="Dict2OLIF" CreaDate="Thu May 15 15:25:25 CEST 2008" CreaId="default">
        <contentInfo>
                <quotMarkInfo QuotMarkRet="all" QuotMarkForm="unknown"/>
                <syllabificationMarkInfo>default</syllabificationMarkInfo>
                <langIdUse>default</langIdUse>
                <valueDefaults/>
        </contentInfo>
</header>
```

Druhou částí je element **body** (tělo), ten již obsahuje jednotlivá lexikální/terminologická data v podobě jednotlivých hesel a vtahů mezi nimi – elementy **entry**. Poslední nepovinnou část tvoří sdílené doplňkové zdroje (bibliografické informace, ...).

Element **entry** představuje 1 konkrétní záznam v danném jazyce. Tento záznam je v rámci dokumentu unikátní a je identifikován obsahem elementu **KeyDC.**

```
<keyDC>
        <canForm>břicho</canForm>          <!-- Kanonicka forma hesla -->
        <language>cz</language>            <!-- Jazyk -->
        <ptOfSpeech>other</ptOfSpeech>     <!-- Part Of Speach ~= slovni druh -->
        <subjField>general</subjField>     <!-- Znalostni domena ke ktere patri heslo -->
        <semReading>1</semReading>         <!-- Pro odliseni ruznych vyznamu stejneho h. -->
</keyDC>
```

Element KeyDC tedy slouží jako složený klíč pro identifikaci konkrétního hesla. Pro snadnější použití lze též vygenerovat identifikátor, který lze použít místo složeného klíče – atribut **MonoUserId.**

```
<keyDC MonoUserId="břicho@1@cz">
    ...
</keyDC>
```

Vztahy mezi jednotlivými hesly se vyjadřují buďto pomocí transferů (jedná-li se o překlad), nebo cross-refernecí (jedná-li se o vztah uvnitř stejného jazyka). Tyto linky jsou vždy pouze jednosměrné. Cíl se adresuje pomocí klíče KeyDC.

```
<entry>
    <mono MonoUserId="břicho@@cz">
        <keyDC>
            <canForm>břicho</canForm>
            <language>cz</language>
            <ptOfSpeech>other</ptOfSpeech>
            <subjField>general</subjField>
        </keyDC>
    </mono>
    <transfer TrTarget="abdomen@@en" TrDefault="yes"/>
</entry>
<entry>
    <mono MonoUserId="abdomen@@en">
        <keyDC>
            <canForm>abdomen</canForm>
            <language>en</language>
            <ptOfSpeech>other</ptOfSpeech>
            <subjField>general</subjField>
        </keyDC>
    </mono>
    <crossRefer CrTarget="distended abdomen@@en">
        <crLinkType>use</crLinkType>
```

```
        </crossRefer>
        <transfer TrTarget="břicho@@cz" TrDefault="yes"/>
</entry>
```

Součástí elementu **entry** mohou být další podelemnty:

- **monoDC** - Obsahuje další možné položky překládaného slova, například slovní druh, pád apod.. monoDC dále obsahuje:
    - **monoAdmin** – Obsahuje například vlastníka slova, oblast užití, původ apod..
    - **monoMorph** – Obsahuje mofrologické údaje o hesle
    - **monoSyn -** Slouží pro popis syntaxe
    - **monoSem** - Slouží pro popis Sémantiky
- **generalDC** - Obsahuje další položky spojené s daným jazykem, nebo poznámky k překládanému slovu. Tento element se může vyskytovat ve všech klíčových elementech.

3.3 ISLE/MILE

Projekt **ISLE** navazuje na dlouhodobé snažení iniciativy **EAGLES/PAROLE**. Je vyvíjen pod záštitou programu **HLT** (Human Language Technology) ve vzájemné spolupráci mezi americkými a evropskými skupinami v kontextu EU-US mezinárodní výzkumné spolupráce a podporovaný NSF a EC [22]. Jeho cílem je vývoj a šíření faktografických norem a směrnic HLT pro jazykové zdroje.

MILE (Multilingual Isle Lexical Entry) je standard od ISLE pro reprezentaci lexikálních dat založený na EAGLES/PAROLE. MILE reprezentuje modulární více-vrstevný model, který je nezávislí na formátu. Nejčastěji se však používá jeho RDF instanciace[23]. MILE se snaží vyjádřit flexibilně kompletní popis slova – mnohojazyčný popis, sémantická reprezentace, apod.. K tomuto účelu definuje definuje tyto moduly:

- **Jednojazyčná lingvistická reprezentace –** Tento modul obsahuje morfologické, syntaktické a sémantické informace charakterizující MILE v konkrétním jazyce. Obecně typologie informace zde obsažené odpovídá konvenčním slovníkům jako například PAROL-SIMPLE. Topologie informace obsažená v této úrovni je následující:
    - Fonetická vrstva
    - Morfologická vrstva
    - Syntaktická vrstva
    - Sémantická vrstva

- **Informace o uspořádání –** Tento modul obsahuje více méně fixní syntagmatické vzorce. Řadíme sem hlavně:
  - Typické uspořádání
  - Podpora pro tvoření sloves
  - Frazeologické nebo konstrukce, nebo konstrukce sousloví
  - Složeniny
- **Vícejazyčný aparát** – tento modul se snaží vytvořit obecný framework, jak vyjádřit vícejazyčné transfery. Skládá se z těchto částí:
  - Identifikace topologie nejběžnějších případů problematických transferů
  - Identifikace podmínek, které musí být splněny, aby se dosáhlo korektního multijazyčného mapování
  - Určení do kterých typů informací z předchozích modulů zasahují tyto podmínky
  - ...

4 Převod slovníkových dat na jednotný formát XML

V této kapitole bude popsán realizovaný systém na převod různorodých neanotovaných plain-textových slovníkových dat na jednotný formát OLIF. Součástí této práce byl i převod slovníku ve formě XML. Avšak implementace byla provedena pomocí jedné jednoduché XSLT šablony a tak v tomto případě není nic moc co zajímavého popisovat (naimplementováno za 1 hodinu).

4.1 Úvod do problematiky

Jak již bylo řečeno úvodem, cílem této práce je navrhnout a realizovat sytém, jenž by byl schopen převést různé reprezentace lexikálních dat na jednotný normalizovaný formát XML. Pro svou relativní jednoduchost a velkou vypovídací hodnotu byl zvolen formát OLIF. Složitost tohoto úkolu spočívá v kvalitě vstupních dat pro počítačové zpracování. Tyto data pocházejí z klasických papírových slovníkům které nebyly vůbec určeny k elektronickému zpracování, často obsahují chyby, objevuje se mnoho odchylek od standardní struktury apod.. Navíc se jedná o plain-textová data pořízená skenováním těchto slovníků. Tímto se již ztratilo mnoho informací. Například zvýraznění hesel odlišným stylem písma v bilinguálních slovnících. To vše vede k tomu, že proces převodu není triviální a často vyžaduje detailní znalost konkrétního jazyka. V bilinguálních slovnících pak znalost obou jazyků.

A [ei]: category "A" prisoners ['ei,prizene(r)z] brit. zvláště nebezpeční pachatelé trestn ých činů ve výkonu trestu, přibl. odsouzení ve třetí nápravně-výchovné skupině v náp zařízení se zvýšenou ostrahou; Table A brit. vzorová smlouva a stanovy pro společnost s r učením omezeným na akcie; "A" shares akcie s omezeným hlasovacím právem

Text 1: Příklad záznamu z bilinguálního slovníku ACLAW

4.2 Požadavky na systém a výběr technologií

- **Unicode** – vzhledem k tomu, že vstupní data jsou v různých kódováních a jazycích, měl by sytém převést vše na unicode a dále pracovat s daty v totmto kódování.

- **Knihovny pro analýzu přirozeného jazyka –** Je jasné že v některých případech se z pouhé struktury záznamu nedá vyčíst celé informace v něm obsažená. Je tedy nutné, aby systém umožňoval použití externích nástrojů pro analýzu textu v konkrétním jazyce.
- **Modularita** – Systém by měl obsahovat jasné definované části, které by měli být propojeny pomocí interfaců. Dosáhne se tak vetší míry znovupoužitelnosti a flexibility obecně.
- **Zavedení abstrakce –** Systém by měl zavést vhodně abstrakci tak, aby se optimalizoval vývojový cyklus.
- **Podpora pro ladění** – všechny použité technologie by měli poskytovat kvalitní možnosti ladění.
- **Rychlost** – pomocí navrženého systému by mělo jít co možná nejrychleji převést nový slovník.

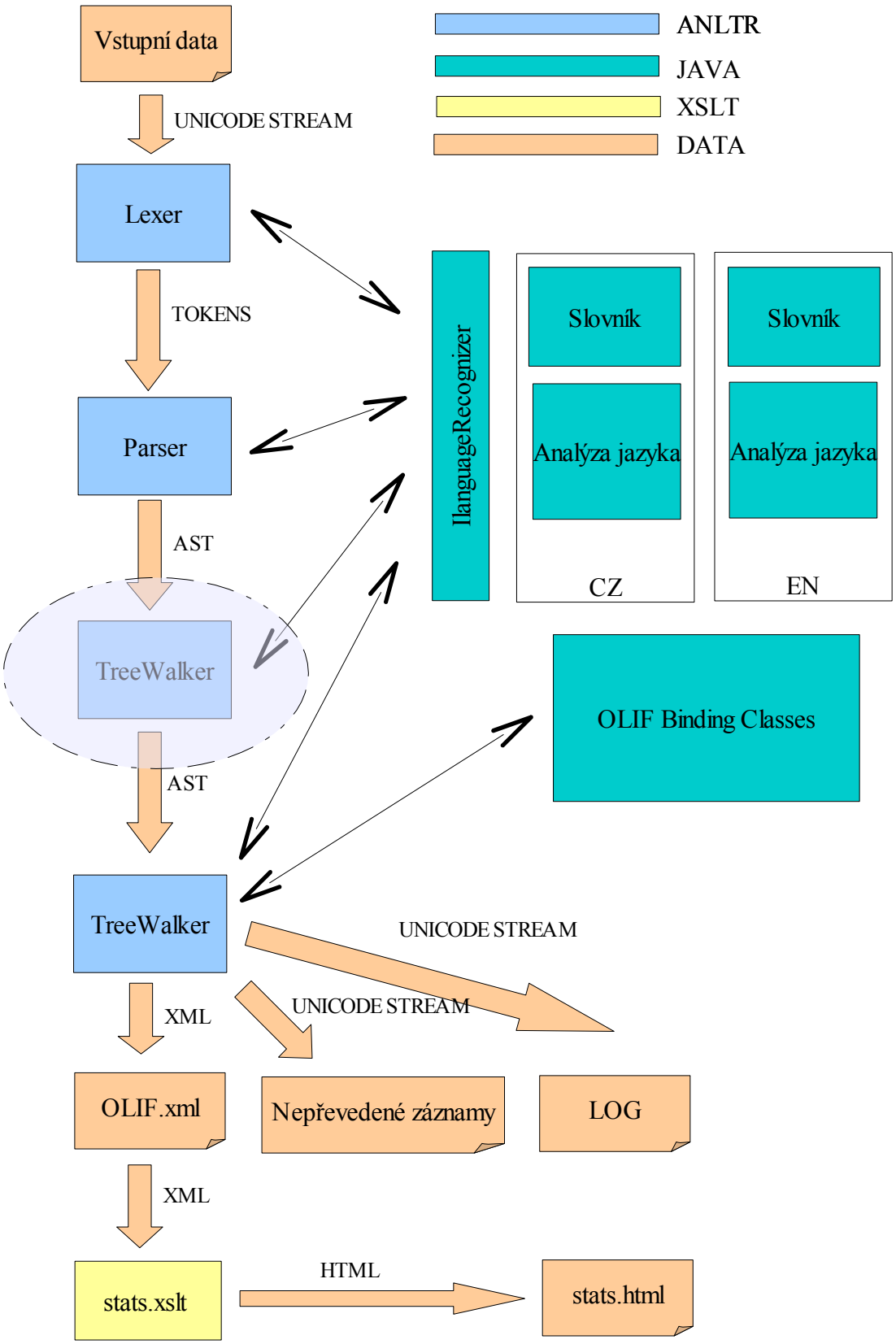
Všechny tyto požadavky mě vedly k použití gramatik pro popis struktury vstupních dat (abstrakce). Pro vývoj gramatik jsem zvolil nástroj **ANTLR** a to hned z několika důvodů:

- V jazyce ANTLR lze použít syntaktické a sémantické predikáty a tím do jisté míry vyjádřit kontextovost gramatiky.
- ANTLR podporuje stromové gramatiky
- Z definice gramatiky v ANTLR lze vygenerovat přímo lexer, parser i treewalker a to zcela (akce/predikáty jsou zapsány v cílovém jazyce). Není tedy nutné do výsledného kódu nijak zasahovat!
- ANTLR podporuje generování do mnoha cílových jazyků (JAVA, .NET, Python, ...)
- ANTLR má vynikající visuální debugger (i vzdáleně přes sokety).
- ANTLR je celosvětově hojně používán a aktivně vyvíjen

Jako cílový jazyk jsem zvolil platformu **JAVA** a to protože splňuje všechny požadavky kladené na systém. Z počátku jsem používal platformu .NET, bohužel jsem narazil na mnohé problémy. Tyto problémy přikládám tomu, že run-time knihovna ANTLR pro .NET nebyla v synchronizaci s ANTLR generátorem. Řešením by sice bylo zkompilovat si run-time knihovnu sám, ale vzhledem k tomu že jsem byl pouze v přípravné fázi, kdy jsem zkoumal vlastnosti ANTLR, tak jsem platformu .NET opustil.

Pro analýzu výsledných dat v OLIF jsem zvolil **XSLT**. A jako XML API jsem zvolil XML Binding (Castor framework).

4.3 Architektura systému



4.3.1 Vstupní data

Vstupní data je nejprve nutno převést na unicode. Dále pak analyzovat strukturu a pokusit se identifikovat problematické části na převod. Po té je třeba připravit si testovací vzorky dat a je i vhodné celý slovník rozdělit na menší části.

4.3.2 Lexer

Návrh lexeru byl většinou velmi jednoduchý, je však nutné mít na paměti očekávané unicode znaky a připravit lexer na jejich akceptaci. Pokud lexer narazí na znak který nezná, vyhodí vyjímku. Pokud je zaplá automatická obsluha vyjímek, dostane uživatel pouze výpis do logu a lexikální analýza bude pokračovat, jako by tam ten znak nebyl. Pokud ne, je výjimka propagována až do hlavní smyčky, kde se zaeviduje do logu a celý záznam se zapíše do nepřevedených záznamů.

4.3.3 Parser

Návrh parseru je už mnohem složitější. V této fázi je nutné vhodně reprezentovat strukturu slovníku gramatikou. U bilinguálních s nejasnou strukturou je nutné použití syntaktických predikátů pro rozlišení příslušnosti slova k danému jazyku.

Například máme-li takovýto záznam

hello ahoj zdravím

mohla by gramatika vypadat takto:

```
//PARSER
entry :     enExpr czExpr;
enExpr:     enWord+;
czExpr:     WORD+;
enWord:     {isWordInEnDict(token)} => WORD; //Pokud je toto slovo v anglickém slovníku, je to anglické slovo

// LEXER
WORD:       'a'..'z';
```

Je také nutné najít vhodné mapování z Parse Tree na AST (Abstract Syntax Tree) a dobře si rozmyslet jak se chovat k odlišnostem od standardní struktury záznamu. Gramatika jen stěží pokryje všechny případy a je tedy nutné postavit se nějak k těmto odlišnostem. Jedním přístupem je netolerovat žádné chyby a ihned vyhodit vyjímku. Ta se poté odchytí v hlavní smyčce, zaloguje a záznam se přidá do seznamu nepřeložených záznamů, stejně jako tomu bylo u Lexeru. Druhým přístupem je pokusit se doplnit (vymyslet si) potřebný token, a pokračovat dále. Tím ale riskujeme, že překlad bude obsahovat chyby. Oba tyto přístupy je tedy vhodné dobře zkombinovat tak, abychom dosáhli optimálního výsledku.

4.3.4 TreeWalker

TreeWalker je vlastně stromová gramtika použitá na průchod AST. TreeWalker může plnit rovnou funkci převodu do OLIF. Vzhledem k tomu, že jsem zvolil XML Binding jako XML API, dost se tím usnadňuje práce s generováním OLIFu. TreeWalker vlastně funguje jako interpretr, který přímo vytváří OLIF objekty a na závěr je provedena serializace do XML (marshaling).

Použití XML Bindingu má ale značnou nevýhodu a to v paměťové náročnosti. Při velký slovnících (>100 000) snadno přeteče paměť. Řešením je buďto provést převod po částech a na závěr provést merge pomocí jednoduché XSLT šablony (merge je nutný kvůli referencím a transferům). Nebo použít XML nativní databázi, popř. perzistovat OLIF objekty do klasické relační databáze (použitý Castor framework toto umožňuje).

TreeWalker může též AST pouze transformovat – dále ho zpracovat. Tento postup je vhodný u komplikovanějších slovníků, kdy by bylo složité celý slovník zpracovat v jednom průchodu. Bohužel verze ANTLR, která umožňuje transformaci stromů je aktuálně ve vývojové a značně nestabilní verzi. Tudíž transformace by tedy musela být napsána přímo v cílovém jazyce, čímž by se ztratila krása celého systému. A tak jsem i já při implementaci musel použit pouze jeden průchod.

4.3.5 Interface pro analýzu přirozeného jazyka – ILangugeRecognizer

Toto rozhraní definuje metody, které jsem použil při dodatečné analýze tehdy, když pro korektní rozpoznání záznamu potřeboval dodatečné informace závislé na konkrétním jazyku. Kvalita knihoven, které se skrývají za tímto interfacem výrazně ovlivňují kvalitu celého převodu. Při implementaci jsem například použil jako slovníkové API Google webovou službu a poté jednoduché slovníkové API BasicSuggester[26] a obě tyto varianty mi dávaly rozdílné výsledky.

4.3.6 Statistiky

Pro implementaci statistik byla použita jednoduchá XSLT šablona, která spočetla:

- počet hesel celkem
- počet hesel v daném jazyce
- počet transferů (překladů)
- počet transferů (překladů) / na heslo
- počet vztahů mezi hesly
- …

Statistiky mají též své místo při posuzování kvality převodu a při ladění. Pokud se hodnoty výrazně liší od očekávání, nebo existují podezřelé záznamy, je někde něco špatně.

5 Závěr

Navrhnutý systém předčil má očekávání. Pro odzkoušení jsem převedl dva bilinguální slovníky s odlišnou strukturou. Převod prvního slovníku zabral přibližně 3 dny a to především protože jsem se teprve seznamoval s možnostmi ANTLR. Převedl jsem více jak 99% záznamů, bohužel výsledek obsahuje pár chyb v souvislosti s nekvalitním API pro analýzu českého jazyka.

Převod druhého slovníku mi zabral už jen 3 hodiny a je převedeno více přibližně 95% záznamů. To přisuzuji chybám, které tento slovník obsahuje. Každopádně rychlost se kterou jsem byl schopen převést tento slovník mě až překvapila.

Za osobní přínos považuji odzkoušení si práce s gramatikami a obzvláště s nástrojem ANTLR, který považuji za absolutní špičku mezi nástroji pro tvorbu překladačů.

Literatura

[0] http://www.w3.org

[1] http://www.w3.org/XML

[2] http://www.oasis-open.org

[3] http://www.w3schools.com/DTD/default.asp

[4] http://www.w3.org/XML/Schema

[5] http://relaxng.org/

[6] http://www.schematron.com/

[7] http://www.w3.org/TR/xpath

[8] http://www.w3.org/Style/XSL/

[9] http://www.w3.org/TR/xslt

[10] http://www.w3.org/TR/xlink/

[11] http://www.saxproject.org/

[12] http://www.w3.org/DOM/

[13] http://www.antlr.org

[14] http://www.ibiblio.org/apollo/assembly_language_manual.html

[15] http://www.antlr.org/license.html

[16] http://www.antlr.org/wiki/display/ANTLR3/Code+Generation+Targets

[17] http://www.antlr.org/works/index.html

[18] http://cs.wikipedia.org/wiki/Slovn%C3%ADk

[19] http://www.tei-c.org/

[20] http://www.itri.bton.ac.uk/projects/concede/

[21] http://www.olif.net/

[22] http://citeseer.ist.psu.edu/681070.html

[23] http://citeseer.ist.psu.edu/662276.html

[24] http://www.wikipedia.org

[25] Kosek J. Různá školení XML 2005 http://www.kosek.cz/

[26] http://www.softcorporation.com/products/suggester/

Seznam příloh

Příloha 1. CD