VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

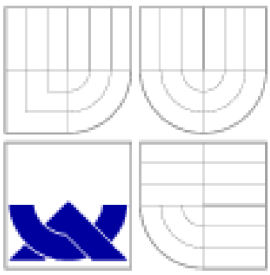
INFERENCE SCHÉMATU Z XML DOKUMENTU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

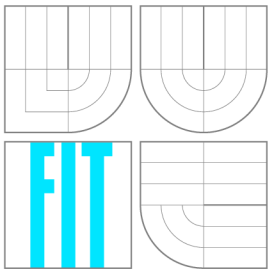
AUTOR PRÁCE
AUTHOR

NELA OLŠAROVÁ

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INFERENCE SCHÉMATU Z XML DOKUMENTU

XML DOCUMENT SCHEMA INFERENCE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

NELA OLŠAROVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZBYNĚK KŘIVKA, Ph.D.

BRNO 2009

Abstrakt

XML je v současné době oblíbeným formátem pro výměnu a uchovávání dat. Vnitřní struktura XML dokumentů je popisována schématem, které má důležitou úlohu při manipulaci s daty. V této práci se zabýváme návrhem metody, která umožní vytvořit k množině vstupních XML dokumentů odpovídající schéma. Prozkoumali jsme stávající publikované metody, které se pokusíme vylepšit o interakci s uživatelem, která se dosud neobjevila v žádném prozkoumaném algoritmu. Při odvozování schématu využíváme formálního základu metody odvozování gramatik.

Abstract

XML has become a popular format for data exchange and manipulation. The internal structure of XML documents is described by the schema that plays an important role in the data management. The main idea is to develop a method for the schema inference from the given set of XML documents. By observing published automatic methods, we discovered their disadvantages. Since there is no interactive implementation, we improve these methods with the user interaction. As a solid formal background, the principle of grammatical inference is extended in our approach.

Klíčová slova

inference XML schématu, XML, XML Schema, DTD, formální gramatiky, odvozování gramatiky, interakce s uživatelem, modulární implementace, XML editor

Keywords

XML schema inference, XML, XML Schema, DTD, formal grammars, grammatical inference, user interaction, modular implementation, XML editor

Citace

Nela Olšarová: Inference schématu z XML dokumentu, bakalářská práce, Brno, FIT VUT v Brně, 2009

Inference schématu z XML dokumentu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Zbyňka Křivky, PhD.

.....
Nela Olšarová
20. května 2009

Poděkování

Děkuji vedoucímu své bakalářské práce panu Ing. Zbyňku Křivkovi, PhD. za podnětné návrhy, vstřícnost a čas, který mi věnoval během konzultací.

Obsah

Kapitola 1

Úvod

V dnešní moderní době vyznačující se vzestupem informačních technologií, které umožňují mít na dosah velké množství různých informací, vznikla potřeba tyto informace také efektivně reprezentovat. K tomuto účelu vznikaly různé formáty pro ukládání a výměnu informací. Populárním standardem pro uchování strukturovaných dat je v současné době formát XML, který je vhodný k reprezentaci dat z různých aplikačních oborů. Oblíbenost tohoto v podstatě univerzálního formátu podporuje jeho textová reprezentace, která umožňuje snadné čtení a editaci, a jednoduchá syntaxe.

Ke zvýšení efektivity zpracování informací obsažených v XML dokumentu slouží jazyky pro popis schématu XML neboli struktury obsažených informací. Předem definovaná struktura XML dokumentu při jejím dodržení usnadní sémanticky správný zápis potřebných údajů na straně odesilatele a jejich náležitou interpretaci na straně příjemce. Obdobně užitečné je dodržení struktury i v případě automatického zpracování informací.

Znalost jazyků XML schémat (DTD, XML Schema a další) patří ale mezi netriviální záležitosti, protože jejich syntaxe je složitější, než syntaxe samotného XML. Velkou pomocí v této oblasti jsou proto nástroje, které uživateli umožní k existujícím XML dokumentům vytvořit odpovídající schéma.

V této práci se budeme zabývat tvorbou nástroje odvozujícího schéma z XML dokumentu a implementací XML editoru, který uživateli zpřístupní jeho použití.

V dalších kapitolách se nejprve seznámíme s technologiemi, které budeme používat, tedy s jazykem XML a jazyky pro popis schématu, z nichž budeme vybírat pro náš účel ten nejvhodnější. V další kapitole se budeme detailněji zabývat jednotlivými fázemi odvozování schématu, kdy zároveň probereme přístupy jednotlivých existujících metod, které budou pro náš algoritmus inspirací. V kapitole 4 navrhneme novou metodu postavenou na formálním základu problému. V další kapitole popíšeme aktuální implementaci metody a implementaci XML editoru, který tuto metodu zpřístupní koncovému uživateli. V kapitole o testování zhodnotíme úspěšnost implementace ve srovnání s otestovanými existujícími nástroji a v závěru nastíníme další možný vývoj metody.

Kapitola 2

Jazyk XML a jazyky pro popis schématu

V této kapitole se seznámíme s jazykem XML a dvěma nejrozšířenějšími jazyky pro popis schématu, kterými jsou DTD (Document Type Definition) a XML Schema. Na závěr provedeme srovnání těchto dvou jazyků pro popis schématu a zdůvodníme tak výběr jazyka pro vyvíjenou metodu, kterým bude XML Schema.

2.1 XML

Jak jsme zmínili již v úvodu, XML (eXtensible Markup Language) je oblíbeným jazykem pro ukládání a přenos informací. Jedná se o značkovací jazyk, který je podmnožinou jazyka SGML (Standard Generalized Markup Language).

Motivem vzniku značkovacích jazyků byla potřeba standardizace a uniformity, vznikající aplikace a platformy s sebou totiž většinou přinášely vlastní formát ukládání dat, který nebyl s ostatními kompatibilní. V roce 1986 vznikl jazyk SGML, který je velmi komplexní, jeho odvozením následně jazyk HTML (HyperText Markup Language), který se používá dodnes pro zobrazování informací na internetových stránkách. Jazyk HTML kromě definice logické struktury dokumentu umožňuje vyjádřit i způsob prezentace dokumentu, tedy vzhled. Jazyk XML narozdíl od HTML vyjadřuje pouze logickou strukturu dokumentu a má neomezenou sadu značek [45].

Seznamme se nyní s jeho základní syntaxí [35].

Následující kód je zapsán v jazyce XML:

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

<osoba id="15">
  <jmeno>Arnošt</jmeno>
  <prijmeni>Moravec</prijmeni>
  <!-- Komentář -->
  <adresa>Malá 3, 123 45 Habartov</adresa>
  <kontakt>arnost.moravec@habartov.cz</kontakt>
</osoba>
```

XML je reprezentováno textovým zápisem, to umožňuje jeho editaci v jakémkoli textovém editoru a zároveň jsou tato data přenositelná mezi různými platformami. Obsahuje

informace, které jsou vloženy mezi značky (tagy), tyto značky, které je možné definovat pro každý typ informace zvlášť (nejsou tedy předdefinovány standardem XML), udávají sémantiku.

Každý XML dokument má jeden kořenový element (zde je to element `<osoba>`), který obsahuje podelementy (`<jmeno>, <prijmeni>, <adresa>, <kontakt>`), ty mohou opět obsahovat další podelementy, tak vzniká stromová struktura dokumentu.

Každý element může mít textový obsah, atributy a podelementy, musí mít svůj počáteční a koncový tag a u správně strukturovaného XML dokumentu musí dvojice počátečních a koncových tagů tvořit správné uzávorkování.

Počáteční tag se zapisuje `<nazev-tagu>` a koncový pomocí znaku `/` takto: `</nazev-tagu>`. Pokud není potřeba zapsat mezi počáteční a koncovou značku obsah tagu, můžeme zápis zkrátit na `<nazev-tagu/>`.

Atributy jsou součástí počátečního tagu (v příkladu `id="15"`) a jejich ideální použití je reprezentace metadat, hodnota atributu je uzavřena do uvozovek (jendoduchých, nebo dvojitých).

Součástí XML dokumentů mohou být i komentáře (v příkladu `<!--Komentář-->`), komentáře jsou uvozeny sekvencí `<!--` a uzavřeny `-->`.

Pokud je XML dokument správně „uzávorkován“, říkáme, že je správně strukturovaný. Pokud jeho struktura navíc odpovídá určitému danému schématu, označujeme jej jako validní. Pro popis struktury XML dokumentů slouží jazyky XML schémat, kterými se budeme zabývat dále.

2.2 DTD

DTD definuje strukturu XML dokumentu pomocí seznamu popisů elementů a atributů [34]. Z hlediska DTD se všechny XML dokumenty skládají z elementů, atributů, entit, PCDATA a CDATA.

2.2.1 Deklarace DTD

Deklarace může být součástí XML dokumentu, nebo jako externí reference. Pokud je DTD součástí XML dokumentu, je syntaxe zápisu takováto:

```
<!DOCTYPE kořenový-element [deklarace-elementů]>
```

Následující příklad udává, že kořenovým elementem dokumentu je element `osoba`, který obsahuje 4 podelementy - `jmeno`, `prijmeni`, `adresa` a `kontakt`, které jsou typu `(#PCDATA)`.

```
<?xml version="1.0"?>
<!DOCTYPE osoba [
  <!ELEMENT osoba    (jmeno,prijmeni,adresa,kontakt)>
  <!ELEMENT jmeno    (#PCDATA)>
  <!ELEMENT prijmeni (#PCDATA)>
  <!ELEMENT adresa   (#PCDATA)>
  <!ELEMENT kontakt  (#PCDATA)>
]>
```

Při deklaraci v podobě externího souboru je zápis takovýto:

```
<!DOCTYPE kořenový-element SYSTEM "název-souboru">
```

V praxi potom:

```
<?xml version="1.0"?>
<!DOCTYPE osoba SYSTEM "osoba.dtd">

<osoba>
  <jmeno>Arnošt</jmeno>
  <prijmeni>Moravec</prijmeni>
  <adresa>Malá 3, 123 45 Habartov</adresa>
  <kontakt>arnost.moravec@habartov.cz</kontakt>
</osoba>
```

2.2.2 Entity

Entity jsou slova začínající na `&`, při parsování dokumentu je za ně dosazován jiný text. Umožňují tak efektivnější zápis opakovaného textu nebo zápis speciálních znaků.

2.2.3 PCDATA, CDATA

PCDATA označuje text, který má být parsován, CDATA text, který nemá být parsován (může obsahovat neescapované sekvence, aniž by měly význam, který mají v jazyce XML).

2.2.4 Deklarace elementů

V jazyce DTD jsou elementy deklarovány touto syntaxí:

```
<!ELEMENT název-elementu kategorie>
```

nebo

```
<!ELEMENT název-elementu (obsah-elementu)>
```

V případě, že zadáváme kategorii, použijeme některou z tabulky 2.1.

kategorie	význam
`EMPTY`	prázdný element
`(#PCDATA)`	obsah pro zpracování parserem
`ANY`	jakýkoli obsah

Tabulka 2.1: Kategorie elementů jazyka DTD

Definice potomků elementu se zadávají s použitím operátorů (viz tabulka 2.2).

2.2.5 Deklarace atributů

Deklarace atributů:

```
<!ATTLIST název-elementu název-atributu typ-atributu výchozí-hodnota>
```

operátor	význam	příklad
-nic-	právě jeden výskyt	`<!ELEMENT název-elementu (potomek)>`
,	sekvence	`<!ELEMENT název-elementu (pot1,pot2,...)>`
+	1-n	`<!ELEMENT název-elementu (potomek+)>`
*	0-n	`<!ELEMENT název-elementu (potomek*)>`
?	0-1	`<!ELEMENT název-elementu (potomek?)>`
|	výběr	`<!ELEMENT název-elementu (potomek1|potomek2)>`
()	skupina	umožňuje kombinace operátorů

Tabulka 2.2: Operátory jazyka DTD

Příklad:

```
<!ATTLIST img
  id      ID        #IMPLIED
  src     CDATA     #REQUIRED
>
```

Přehled typů atributů je uveden v tabulce 2.3. Výchozí hodnota může být buď určena přímým zápisem, nebo některým klíčovým slovem z tabulky 2.4.

typ	význam
`CDATA`	textový řetězec
`(en1|en2|en3)`	jeden prvek ze seznamu - enumerace
`NMTOKEN`	řetězec složený pouze z písmen, číslis a podtržítka
`NMTOKENS`	jako `NMTOKEN`, ale více oddělených mezerou
`ID`	jednoznačná hodnota v rámci dokumentu
`IDREF`	musí odkazovat na hodnotu ID v dokumentu
`IDREFS`	více odkazovaných ID oddělených mezerou

Tabulka 2.3: Typy atributů jazyka DTD

hodnota	význam
`#REQUIRED`	povinný atribut
`#IMPLIED`	nepovinný atribut
`#FIXED` *hodnota*	konstanta

Tabulka 2.4: Výchozí hodnoty atributů jazyka DTD

2.3 XML Schema

XML Schema bývá v literatuře označováno také jako XSD (XML Schema Definition) nebo WXS (W3C XML Schema), přičemž XSD je obvyklejší variantou používanou také jako přípona pro soubory s popisem schématu. Na rozdíl od jazyka DTD je XSD zapisováno

ve formátu XML, což je uznáváno jako jeho výhoda, protože je možné používat zažité prostředky pro práci s XML (editory, parsery, XSLT transformace, DOM).

XML Schema definuje elementy, atributy, podelementy a jejich počet a pořadí, datové typy elementů a atributů a jejich výchozí a konstantní hodnoty. Celé XML Schema je v podstatě založeno na definici datových typů, kdy jej každý element musí mít definovaný [35].

Definice XML schématu uvedená na začátku podkapitoly o DTD (podkapitola 2.2) by se v jazyce XML Schema zapsala takto:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="osoba">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="jmeno" type="xs:string"/>
        <xs:element name="prijmeni" type="xs:string"/>
        <xs:element name="adresa" type="xs:string"/>
        <xs:element name="kontakt" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

Element `osoba` je označen jako komplexní typ, protože obsahuje podelementy. Ty jsou naopak označeny jako jednoduchý typ.

Externí reference pro XML Schema se zapisuje takto:

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

<osoba
  xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
  xs:schemaLocation="osoba.xsd">

  <jmeno>Arnošt</jmeno>
  <prijmeni>Moravec</prijmeni>
  <adresa>Malá 3, 123 45 Habartov</adresa>
  <kontakt>arnost.moravec@habartov.cz</kontakt>
</osoba>
```

Každá definice v jazyce XML Schema obsahuje kořenový tag `schema`. Ten obsahuje atribut `xmlns:xs` pro označení jmenného prostoru jazyka XML Schema a `schemaLocation` pro adresaci schématu, který používáme s uvedeným jmenným prostorem.

2.3.1 Jednoduché datové typy

Jednoduchý datový typ elementu znamená, že element může obsahovat pouze text (libovolného typu definovaného v XML Schema, nebo uživatelsky), nemůže obsahovat jiné elementy nebo atributy. Definice jednoduchého datového typu vypadá takto:

```
<xs:element name="název-elementu" type="datovy-typ-elementu"/>
```

Vestavěných datových typů je spousta, nejvíce používané jsou: `xs:string`, `xs:decimal`, `xs:integer`, `xs:boolean`, `xs:date`, `xs:time`. Celé schéma hierarchie datových typů v jazyce XML Schema je uvedeno v příloze na obrázku A.1 [48].

Jednoduché typy můžou mít definovanou výchozí hodnotu (pomocí atributu `default`), nebo fixní (neměnnou) hodnotu (pomocí atributu `fixed`).

2.3.2 Atributy

Atributy se vyskytují pouze u komplexních typů elementů, jsou ovšem samy definovány jako jednoduché typy, deklarují se až za vnořenými elementy. Definice atributu se provádí takto:

```
<xs:attribute name="název-atributu" type="datový-typ-atributu"/>
```

Atributy mohou mít opět definovanou výchozí nebo fixní hodnotu. Mohou být volitelné, nebo povinné, toto se určuje atributem `use`.

2.3.3 Omezení hodnot elementů a atributů

U elementů a atributů mohou být definovány omezení (restrikce), příkladem takového omezení je určení datového typu, omezení můžeme také uživatelsky definovat. Následující ukázka definuje omezení hodnot od 0 do 120.

```
<xs:element name="vek">

<xs:simpleType>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="120"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

Omezení může být dáno také výčtem přípustných hodnot

```
<xs:element name="car">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

Pro omezení můžeme použít i regulární výraz

```
<xs:pattern value="([a-z][A-Z]){8}"/>
```

případně omezit délku

```
<xs:length value="8"/>
```

nebo přesněji

```
<xs:minLength value="5"/>
<xs:maxLength value="8"/>
```

Všechny možnosti omezení datových typů jsou v tabulce 2.5.

omezení	popis
enumeration	seznam povolených hodnot
fractionDigits	maximální počet desetinných míst
length	přesný počet znaků nebo položek seznamu
maxExclusive	omezení číselných hodnot shora
maxInclusive	omezení číselných hodnot shora včetně
maxLength	maximální počet znaků nebo položek seznamu
minExclusive	omezení číselných hodnot zdola
minInclusive	omezení číselných hodnot zdola včetně
minLength	minimální počet znaků nebo položek seznamu
pattern	řetězec specifikovaný regulárním výrazem
totalDigits	počet číslic
whiteSpace	určuje zacházení s bílými znaky

Tabulka 2.5: Tabulka omezení datových typů jazyka XML Schema

2.3.4 Komplexní datové typy

Komplexním typem označujeme element, u kterého chceme definovat podelementy nebo atributy. Rozlišujeme 4 druhy komplexních elementů (všechny tyto typy mohou obsahovat i atributy):

- Prázdné elementy
- Elementy, které obsahují pouze podelementy
- Elementy obsahující pouze text
- Elementy obsahující text i podelementy (smíšený obsah)

V jazyce XML Schema je také možné odvodit na základě definovaného typu typ nový.

Deklarace elementu, který má obsahovat pouze text probíhá pomocí

```
<xs:simpleContent>
```

Pokud má obsahovat text i elementy (smíšený obsah), potom pomocí

```
<xs:complexType mixed="true">
```

Deklarace prázdného elementu s atributy se provádí tak, že komplexnímu typu nepřiřadíme žádný výskyt podelementů, pouze zapíšeme atributy.

Způsob použití a výskyt podelementů se určuje pomocí identifikátorů (viz tabulka 2.6).

indikátor	význam
all	libovolné pořadí podelementů (můžou se ovšem vyskytovat pouze 0-1 krát)
choice	může se vyskytovat pouze jeden z uvedených podelementů
sequence	podelementy se musí vyskytovat v uvedeném pořadí
maxOccurs	maximální počet výskytů podelementu
minOccurs	minimální počet výskytů podelementu

Tabulka 2.6: Identifikátory komplexního typu v jazyce XML Schema

2.3.5 Přístupy k návrhu schématu

XML Schema umožňuje definovat datové typy buď jejich deklarací mimo element, odkud se na něj již jen odkážeme a umožníme zároveň použít tento typ více elementům, nebo přímým zápisem na místo deklarace elementu. Je potřeba také vybrat, kdy použít lokální a kdy globální elementy [22].

Matrjóška (Russian Doll Style) V tomto přístupu je jen jeden globální element a všechny ostatní jsou uvnitř něj definovány jako lokální. Dokument může začínat pouze tímto globálním elementem a jiný element nelze použít znovu v dalších schématech.

Salámová kolečka (Salami Slice Design) Zde jsou všechny elementy definovány jako globální a potom se na ně na místech jejich výskytu odkazuje pomocí odkazů `ref`. V tomto přístupu nelze pro jeden element definovat dva různé modely obsahu v závislosti na kontextu elementu.

Metoda slepého Benátčana (Venetian Blind Design) Za cenu větší pracnosti a složitosti tímto modelem získáme ve většině případů nejvhodnější řešení. Nejprve pro všechny elementy definujeme zvlášť datové typy, které potom přiřazujeme lokálně definovaným elementům atributem `type`. Tento model se jeví pro naši metodu jako nejvhodnější.

2.4 Srovnání a výběr cílového jazyka schématu

Pokud srovnáme vlastnosti XML Schema a DTD, vychází XML Schema jako pokrokovější a schopnější. Oproti DTD má některá vylepšení [23]:

- Zápis v XML
- Přesnější určování počtu opakování elementů
- Více vestavěných datových typů, ze kterých je zároveň možné odvozovat nové (omezením - `restriction`, rozšířením - `extension`)
- Určení datových typů i pro atributy
- Možnost rozdílných definic elementů v závislosti na jejich kontextu (předcích), v podstatě lze označit vazbu element-typ jako `n:n`
- Zavedení operátoru permutace (`all`), který zjednodušuje zápis

- Je možné využít objektově orientované rysy (dědičnost, abstraktní typy, substituce)
- Modularizace schématu (rozdělení definic do souboru - `include`, `import`, podpora jmenných prostorů)

Z hlediska výběru cílového jazyka pro odvozování se jeví XML Schema zajímavějším, protože nabízí více možností, kterými se lze zabývat v rámci návrhu odvozovacího algoritmu. Můžeme se zaměřit na prakticky kterékoli z výše uvedených vylepšení XML Schema oproti DTD.

Kapitola 3

Odvozování schématu

V této kapitole se seznámíme se základními principy odvozování schématu z XML dokumentů. Kapitola přináší ucelený přehled řešení objevených v prozkoumané literatuře a zobecňuje je do popisu tří specifických fází, kterými prochází metody odvozující schéma. Na závěr budou představeny konkrétní existující metody.

3.1 Úvod do odvozovacích algoritmů

Hledáme algoritmus, který na základě množiny vstupních XML dokumentů odvodí jejich vnitřní strukturu a vytvoří tak schéma, kterému vstupní dokumenty odpovídají. Průběh odvozování můžeme obecně shrnout těmito body:

1. Vytvoření prvotní reprezentace schématu ze vstupních dokumentů
2. Zjednodušování a zobecňování schématu
3. Transformace výsledných pravidel do cílového jazyka schématu

V dalším textu budeme postupně rozvíjet souvislosti týkající se jednotlivých bodů a přístupy k jejich řešení.

Při načítání vstupní množiny XML dokumentů je potřeba informace o jejich struktuře průběžně ukládat a v dalším bodě následně různě transformovat, a tak se v podkapitole 3.2 budeme zabývat způsoby reprezentace těchto údajů a souvisejícími pojmy z teorie formálních jazyků.

S druhým bodem souvisí hledání optima mezi přílišným zobecněním schématu a přesností popisu vstupních dokumentů, více o tomto problému včetně potřebných definic uvedeme v podkapitole 3.3.

Zadaný problém se dotýká teorie odvozování gramatik a někteří autoři volí formální řešení, více se s tímto problémem seznámíme v podkapitole 3.4.

Podmínkou správnosti výsledného schématu je determinismus schématu, jeho popis a způsoby, kterými jej řeší autoři ve zkoumané literatuře, si přiblížíme v podkapitole 3.5.

Kromě samotné struktury dokumentů, tedy popisu vnoření a uspořádání podelementů, jsou jednou ze součástí schématu i informace o atributech, datových typech, mezích opakování elementů a závislosti definic na předcích elementu, o řešení těchto detailů se více dozvíme v podkapitole 3.6.

3.2 Modely pro popis schématu

Některé metody využívají pro odvozování schématu teorii formálních jazyků (elementy zde představují vstupní abecedu a definice obsahu elementů potom gramatická pravidla). XML jazyk patří v Chomského hierarchii k jazykům bezkontextovým, ale strukturu můžeme také popsat pomocí více regulárních gramatik, kdy definujeme uspořádání přímých podelementů pro každý element zvlášť, podrobnější informace včetně důkazu lze nalézt v [43] a [3]. Souvisejícími modely jsou tedy gramatiky, konečné automaty (resp. prefixové stromy) a regulární výrazy. Základní definice jsou doplněny o speciální definice potřebné k vysvětlení algoritmů, které budou detailně popsány dále.

Jiné metody využívají pro grafické znázornění struktury celých dokumentů graf dokumentu.

3.2.1 Gramatiky

Gramatika definuje strukturu formálního jazyka, my si zde uvedeme 2 speciální případy používané při odvozování schématu, a to bezkontextové gramatiky a regulární gramatiky.

Bezkontextové gramatiky (BKG) jsou definovány takto [24]:

> **Definice 3.1** *Bezkontextová gramatika je čtveřice* $G = (N, T, P, S)$*, kde*
>
> - N *je abeceda neterminálů*
> - T *je abeceda terminálů, přičemž* $N \cap T = \varnothing$
> - P *je konečná množina pravidel tvaru* $A \to x$*, kde* $A \in N$*,* $x \in (N \cup T)^*$
> - $S \in N$ *je počáteční neterminál*
>
> *Rozšířené BKG* jsou bezkontextové gramatiky, které mají na pravé straně regulární výraz (viz definice 3.5).
>
> *Rozšířené BKG s rozšířeným regulárním výrazem* jsou rozšířené bezkontextové gramatiky, které mají na pravé straně rozšířený regulární výraz (viz definice 3.6).

Regulární gramatiky jsou definovány obdobně jako BKG, pravidla jsou ovšem buď ve tvaru $A \to xB$, nebo tvaru $A \to x$, kde $A, B \in N, x \in T^*$ [24].

3.2.2 Konečný automat

Konečný automat svou vyjadřovací schopností odpovídá regulárním gramatikám. Jak se dozvíme dále, používá se v různých obměnách při zobecňování schématu, je definován takto [24]:

Definice 3.2 *Konečný automat (KA) je pětice* $M = (Q, \Sigma, R, s, F)$*, kde* Q *je konečná množina stavů,* Σ *je vstupní abeceda,* R *je konečná množina pravidel tvaru* $pa \to q$*, kde* $p, q \in Q$*,* $a \in \Sigma \cup \{\epsilon\}$*.*

Single Occurence Automaton (SOA) je speciální KA, který je použit v algoritmu iXSD (viz sekce 3.7.8). Jeho definice je takováto:

Definice 3.3 *Nechť in a out jsou dva speciální symboly různé od názvů elementů, po řadě počáteční a koncový stav. SOA je graf $A = (V, E)$, kde všechny stavy ve $V - \{in, out\}$ jsou názvy elementů, a $E \subseteq (V - \{in\}) \times (V - \{out\})$ je relace hran.*

Pravděpodobnostní konečný automat (PFSA) je konečný automat doplněný o pravděpodobnosti přechodů mezi stavy.

3.2.3 Prefixový strom

Někteří autoři pracují z počátku při konstrukci konečného automatu s prefixovým stromem ([1], [46], [8]). Jedná se vlastně o deterministický konečný automat, následující definice pochází z [32].

Definice 3.4 *Mějme množinu řetězců $S \subseteq \Sigma^*$, kde Σ je abeceda. Nechť prefixový strom slouží k uložení této množiny řetězců, potom každá hrana vedoucí do některého z nekoncových stavů je označena symbolem abecedy Σ. Každá hrana vedoucí do jednoho z koncových stavů (listů) je označena \$ ($\$ \notin \Sigma$). Pro každý řetězec $s \in S$ existuje v prefixovém stromu cesta z vrcholu do listu, kde konkatenace označení hran je s\$. Pro každou cestu od kořene k listu získáme konkatenací označení hran některý řetězec množiny S.*

Příklad prefixového stromu pro $S := \{alpha, beta, bear, beast, beat\}$ viz obrázek 3.1 [32]

Pravděpodobnostní prefixový strom (PTA) vznikne doplněním hran a koncových stavů o počet průchodů při doplňování stromu o další vstupní řetězce. Takovýto prefixový strom může být použit jako *pravděpodobnostní konečný automat (PFSA)* (viz sekce 3.2.2) [46].

3.2.4 Regulární výrazy

Některé algoritmy pracují s generováním regulárních výrazů a následným zjednodušováním pomocí definovaných pravidel. Regulární výrazy jsou výrazově ekvivalentní regulárním gramatikám.

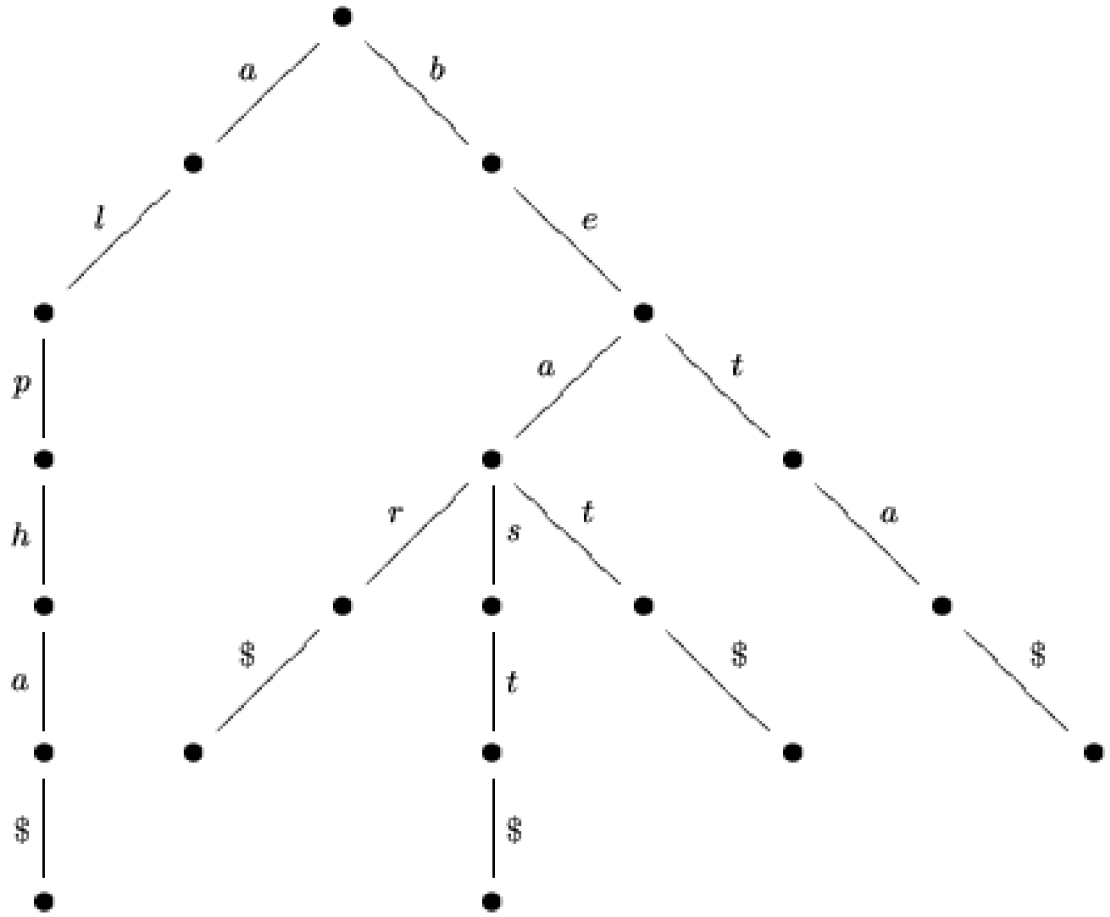
Uvedeme definici pro regulární výrazy [29] a rozšířené regulární výrazy, které se používají pro přesnější určení počtu opakování podelementů, které umožňuje jazyk XML Schema [9]:

Definice 3.5 *Nechť Σ je abeceda. Regulární výraz (RV) nad abecedou Σ je definován následovně:*

- *$\varnothing$ je RV*
- *ε je RV*
- *$\forall a \in \Sigma$, je RV*
- *Nechť r a s jsou RV, potom (rs) (konkatenace), $(r|s)$ (výběr), (r^*) (libovolné, i nulové, opakování předchozího výrazu), jsou RV.*

Jazyky pro popis schématu dále zavádějí další operátory, pro jazyk DTD jsou to: $(s|\varepsilon) = (s?)$ a $ss^* = (s^+)$. Jazyk XML Schema přichází ještě s operátorem permutace, který bývá vyjadřován jako &.

Definice 3.6 *Rozšířený regulární výraz je výraz nad abecedou Σ, kde každý element je definován s rozsahem $[l : r]$, který vyjadřuje počet možných výskytů, kde l a r jsou nezáporná celá čísla, $0 \leq l \leq r$, r může být i ∞.*



Obrázek 3.1: Příklad prefixového stromu

Bývá používán také *Single Occurence Regular Expression (SORE)*. Jedná se regulární výraz s unikátním výskytem symbolů.

3.2.5 Graf dokumentu

Graf dokumentu (v originální literatuře„spanning graf“) je uspořádaný acyklický graf reprezentující celou stromovou strukturu vstupních dokumentů. Každý uzel zde odpovídá jednomu elementu dokumentu a jeho potomci jsou podelementy. Graf je poté možno aplikací různých pravidel zobecňovat a následně se produkují regulární výrazy popisující stukturu podelementů. Tento přístup je využit pro generování DTD v [30].

3.3 Zobecňování schématu

Při odvozování schématu potřebujeme vytvořit schéma, které popisuje vstupní dokumenty, ale zároveň umožňuje i validaci dokumentů, které jsou jim podobné. Při vytvoření prvotní reprezentace schématu pomocí některého modelu popsaného v předchozí podkapitole (3.2)

dostáváme schéma, které přesně popisuje vstupní dokumenty. Potřebujeme jej tedy dále dostatečně zobecnit a potlačit detaily.

Postup zobecňování závisí na modelu, který byl vybrán pro reprezentaci prvotního schématu. Zobecňování se u metod postavených na teorii formálních jazyků provádí například slučováním stavů konečného automatu, nebo redukcí pravidel gramatiky.

Konečné automaty Zobecňování se provádí slučováním stavů konečného automatu. Po zhodnocení, že dva stavy automatu mohou být sloučeny, je vytvořen nový stav a všechny vstupní a výstupní hrany jsou přesměrovány na vstup respektive výstup nového stavu.

Gramatiky Zde probíhá zobecňování pomocí redukcí pravidel. Více se dozvíme v popisu algoritmu ECFG (viz sekce 3.7.9).

U metod, které jsou založené přímo na metodě odvozování gramatik (grammatical inference, GI), je zobecňování založeno na vlastnostech nadefinované podtřídy jazyka. Více o používaných podtřídách jazyka si uvedeme v podkapitole 3.4.

K rozhodnutí, které stavy sloučit, či která pravidla redukovat, je možné používat metriky a metody, které si uvedeme dále.

3.3.1 Určení podobnosti

Před sloučením stavů nebo sjednocením pravidel gramatiky potřebujeme určit, nakolik jsou pravidla či stavy podobné. U konečných automatů určujeme ekvivalenci stavů podle řetězců, které je následují. Používají se následující kritéria, jedná se o zjednodušení Nerodovy ekvivalence [43].

***k*-tails** Dva stavy jsou ekvivalentní, pokud jsou nerozlišitelné podle řetězců maximálně délky k, které je mohou následovat.

***k*-string** Liší se od k-tails v tom, že řetězec nemusí končit v koncovém stavu, ale pak má délku přesně k.

***sk*-string** Liší se od k-tails použitím pravděpodobnostního automatu (PTA), metoda byla prezentována v [46], bere v úvahu pouze s nejpravděpodobnějších řetězců odpovídajících kritériu k-string.

Pro slučování pravidel gramatiky je v [9] popsána metrika VSM, zmíníme i pojem editační vzdálenost.

VSM (Vector Space Model) Zavádíme vektorový prostorový model, na pravé strany pravidel nahlížíme jako na řetězce nad abecedou terminálů a měříme úhel jejich vzdálenosti. Ten potom porovnáváme s předem definovaným prahem, který určuje, zda jsou pravidla dostatečně podobná a mohou tedy být sloučena.

Editační vzdálenost Některé algoritmy provádějí výpočet editační vzdálenosti (tedy počet editačních kroků, které by bylo potřeba učinit, aby se 2 porovnávané výrazy zcela rovnaly), následně je obdobně jako u VSM porovnávají s definovaným prahem.

3.3.2 Hledání nejlepšího zobecnění

Některé metody zavádějí pro získání co nejlepšího výsledku metriky pro hodnocení generovaných schémat. Potom vybírají z možných zobecnění schématu to nejlépe ohodnocené. S těmito metrikami se nyní seznámíme, některé byly podrobněji popsány již v [43], proto nebudeme zabíhat příliš do detailů.

Princip MDL (Minimum Description Length) V [15] je využíván pro poměření ceny zakódování výsledného schématu a ceny zakódování všech možných vstupních dokumentů, které toto schéma popisuje. Snahou je tyto ceny minimalizovat.

MML (Minimum Message Length) Je to míra kvality schématu, cílem je tuto hodnotu minimalizovat, je založená na maximalizování podmíněné pravděpodobnosti abstrakce (schématu) vzhledem k daným příkladům (vstupní dokumenty). Metoda v [46] tento princip využívá za použití pravděpodobnostního prefixového stromu (PTA, Prefix Tree Automaton, viz sekce 3.2.3), jehož MML hodnotí.

ACO (Ant Colony Optimisation) Jedná se o pomocnou metodu (optimalizační techniku), která slouží k výběru nejlepší z variant výsledného schématu, které jsou ohodnocovány výše uvedenými metrikami. Mravenci prohledávají prostor řešení a hledají nejlepší zobecnění schématu. Použitím metody dochází k tomu, že není generováno nekonečné množství všech možných schémat, ale dále se rozvíjejí jen ty, u kterých to má největší smysl [43].

3.4 Odvozování gramatik

Jak již bylo podrobněji uvedeno v [43], problém může být řešen odvozováním gramatiky, neboli *GI (grammatical inference)*, z oboru strojového učení a umělé inteligence. Pro odvození gramatiky vytváříme odvozovací stroj, neboli *IM (inference machine)*, který na základě příkladů odvodí danou gramatiku.

V našem případě na vstupní dokumenty pohlížíme jako na pozitivní příklady a na výsledné schéma jako na odvozenou gramatiku. Takovéto metody fungují na principu slučování stavů, přičemž kritéria, podle kterých se slučují, závisí na vlastnostech gramatiky.

V [3] byla nadefinována XML gramatika. Různí autoři se následně zabývali problémem identifikovatelných podtříd jazyka, protože Gold v [16] dokázal, že jazyk není identifikovatelný pouze z pozitivních příkladů. V literatuře potom nacházíme přístupy, které jazyk nějakým způsobem omezují (definují jeho podtřídy) a činí jej tak identifikovatelným na základě pozitivních příkladů.

***k*-kontextové a (*k*,*h*)-kontextové jazyky** Metody pro odvozování zde předpokládají, že kontext elementů je omezený [1], 2 stavy potom mohou být sloučeny, pokud od nich vedou stejné cesty (posloupnost symbolů) délky k, případně sloučí i následujících h stavů.

f-rozlišitelné jazyky Jazyky rozlišitelné funkcí [12].

SORE, *k*-local V [6] bylo ukázáno, že SORE (viz sekce 3.2.4) a zároveň zavedení třídy k-local (definice prvku je závislá na k jeho předcích, viz sekce 3.6.4) je taktéž cestou k učení pouze z pozitivních příkladů.

3.5 Determinismus

Podle specifikace konsorcia W3C [47] musí být model obsahu elementu jednoznačný. To znamená, že při validaci dokumentu oproti schématu musí obsah elementu vstupního XML dokumentu odpovídat právě jednomu výskytu v popisu elementu. Musí být tedy zřejmé, oproti kterému symbolu v definici obsahu elementu je právě podelement validován. Toto potom vede na nižší nároky při implementaci validátoru, který nemusí kontrolovat všechny výskyty (tedy prozkoumávat celé schéma dopředu), které by odpovídaly právě kontrolované části XML dokumentu, ale jednoduše určí validnost XML dokumentu podle prvního výskytu. Ukažme si některé metody odstraňování nedeterminismu (nejednoznačnosti).

3.5.1 SORE, *k*-ORE

Metoda popsaná v [6] využívá SORE (viz sekce 3.2.4) jako podmínku postačující k zaručení determinismu. Pokud se v regulárním výrazu nebude opakovat jeden symbol vícekrát, potom je určitě deterministický. SORE generujeme z SOA (viz definice 3.3) speciálním algoritmem iDTD popsaném v [5], běžné metody převodu KA na regulární výraz totiž mohou vést k nepřehlednému a nedeterministickému výrazu.

Může ovšem nastat případ, že výraz je deterministický, i když neobsahuje pouze unikátní symboly (například výraz $b^*a(b^*a)^*$ deterministický je [4]). A tak jsou v [4] definovány ještě k-ORE, k zde znamená počet opakování symbolů ve výrazu, který tak umožňuje přesnější popis modelu obsahu elementu. Nemáme však prakticky použitelný algoritmus, který by odvodil deterministický k-ORE z pozitivních příkladů. Většina reálných dokumentů (99% [4]) ale odpovídá právě SORE, takže tento přístup je dostatečný.

3.5.2 Glushkovův automat

Další možností, jak získat deterministický regulární výraz, aniž bychom potřebovali SOA, je využití Glushkovova automatu [18]. Pokud bude výsledkem konstrukce Glushkovova automatu podle Glushkovova algoritmu deterministický automat, pak byl deterministický i regulární výraz. Tento postup lze obrátit a získat tak algoritmus pro vytvoření deterministického regulárního výrazu [4].

3.5.3 Hledání orbitů

Detailní definici nejednoznačnosti a algoritmus pro její odstranění při převádění KA na regulární výraz uvedla Helena Ahonen v [1]. Nejednoznačnost je zde určena pomocí orbitů, které nalézáme v nedeterministickém automatu. Orbity jsou silně souvislé komponenty konečného automatu, které vznikají při výskytu cyklů, a jsou dominantním původcem nejednoznačnosti. Více lze nalézt v [1]. Orbity řeší i algoritmus ECFG [9], který bude blíže popsán v sekci 3.7.9.

3.6 Odvozování dalších údajů

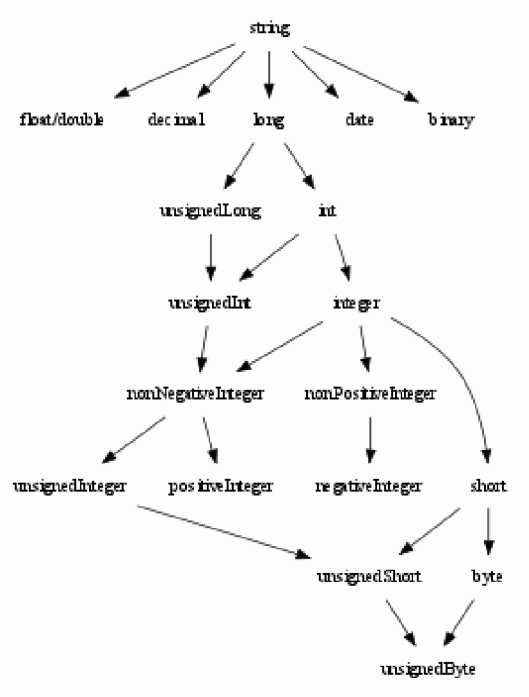
V této sekci popíšeme možnosti odvozování dalších částí schématu, které přímo nesouvisejí se stromovou strukturou popisovaných XML dokumentů. Jedná se o atributy, datové typy a uvedeme i řešení mezí opakování elementů a udržování kontextu předků.

3.6.1 Atributy

Práci s atributy implementuje například algoritmus XStruct (více viz sekce 3.7.7) a DTD-Miner (viz sekce 3.7.1). Atribut se může vyskytovat pouze jednou a být volitelný, nebo povinný. Pokud byl atribut přítomen u všech výskytů elementu, je označen jako povinný, pokud ne, jako volitelný. XStruct ještě implementuje atributy s konstantní hodnotou, odvození datových typů atributů je činěno stejně jako u elementů a je zde možnost detekovat výčtové typy.

3.6.2 Jednoduché datové typy

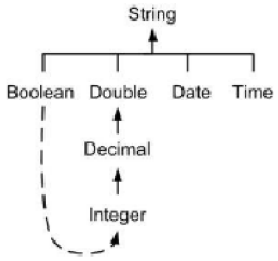
Odvození jednoduchých datových typů elementů implementují algoritmy ECFG [9] a XStruct [18]. Pro tento účel je zde definována tabulka hierarchie typů (pro algoritmus ECFG viz obrázek 3.2, pro XStruct viz obrázek 3.3) a následně jsou zkoumány jednotlivé výskyty hodnot a vybírá se nejmenší (podle uspořádání znázorněného na obrázcích) možný datový typ, který je ještě obsahuje. V algoritmu ECFG je pro odvození typů definována sada jednoduchých pravidel pro každý typ ve tvaru *(regulární výraz, rozsah)*.



Obrázek 3.2: Hierarchie datových typů algoritmu ECFG

3.6.3 Meze opakování elementů

Stejně jako datové typy, implementuje algoritmus ECFG [9] i meze opakování elementů. Počty opakování jsou odvozovány na základě speciálního konečného automatu, u kterého jsou na hranách znázorněny počty průchodů při postupném přijímání vstupních dokumentů.



Obrázek 3.3: Hierarchie datových typů algoritmu XStruct

3.6.4 Kontext elementů

Algoritmus iXSD [6] se při definici elementů ohlíží na jeho předky. Celá identifikace definovaného elementu závisí jednak na jeho názvu, ale také na k jeho předcích. Vychází ze studie reálných dokumentů, ve kterých kontext elementu závisí pouze na k předcích, nikoli na celé cestě od kořene.

3.7 Existující algoritmy

Po nadefinování všech potřebných pojmů a úvodu do problematiky si nyní můžeme představit jednotlivé algoritmy. Některé z dále uvedených algoritmů (DTD-Miner, Fred, Heuristika *sk*-Ant, XTRACT, Alergia a autorský SchemaMiner) byly již detailně pro české čtenáře popsány v [43], a tak se jimi zde již nebudeme příliš zabývat a nastíníme si pouze jejich základní principy.

3.7.1 DTD-Miner

Metoda popsaná v [30] využívá graf dokumentu, kterým popisuje struktury vstupních dokumentů, dokumenty do stromu rekurzivně slučuje na základě hledání nejdelších společných podstromů. Tento strom, ktery po vytvoření odpovídá přesně zadaným dokumentům, potom optimalizuje a zobecňuje a následně generuje DTD. Algoritmus pracuje i s atributy.

3.7.2 Fred

Tato metoda odvozuje schéma DTD pro jeden vstupní dokument, byla uvedena v [36]. Vnitřní strukturu dokumentu popisuje pomocí pravidel, kde na pravé straně je regulární výraz a na levé popisovaný element. Potom tyto pravidla redukuje a zobecňuje pomocí svých definovaných pravidel.

3.7.3 Heuristika sk-Ant

Metoda uvedená v [46] nejdříve konstruuje prefixový strom (automat) a následně slučuje jeho stavy. Slučování stavů provádí tak, že prochází prostorem řešení (možnosti sloučení stavů v prefixovém stromu) a ohodnocuje je, následně vybere to nejoptimálnější. K ohodnocení využívá ACO (Ant Colony Optimisation, viz sekce 3.3.2) v kombinaci s metrikou hodnocení schématu. Metrika poměřuje MML (viz sekce 3.3.2), tedy úspornost popisu jednotlivých řešení. Slučování stavů je podmíněno kritériem *sk*-string (viz sekce 3.3.1).

3.7.4 XTRACT

Metoda v [15] pracuje s regulárními výrazy, které poté zobecňuje. Zobecňováním generuje velké množství možných schémat, které vždy popisují část vstupních dokumentů. Tato dílčí schémata následně oceňuje použitím principu MDL (viz sekce 3.3.2). Na konci metody máme tedy na výstupu několik různých částečných řešení, z nichž se potom vybere nejoptimálnější kombinace popisující celé schéma, tato částečná řešení se konjunkcí spojí ve výsledný popis. Podle [18] tato metoda ale nemusí vést ke správnému výsledku, protože neošetřuje možný výskyt nejednoznačnosti (viz podkapitola 3.5).

3.7.5 ALERGIA

Tento algoritmus z [8] sestavuje prefixový strom a následně vytváří SRL (Stochastic Regular Language) jazyky, což jsou regulární jazyky doplněné o pravděpodobnosti výskytu. Pomocí slučování stavů potom odvozuje pravděpodobnostní konečný automat (viz sekce 3.2.2). Porovnává dvojice stavů podle pravděpodobnostního rozložení pro hrany, které z nich vedou, přičemž cílové uzly musí být také ekvivalentní.

3.7.6 SchemaMiner

Metoda, kterou přináší [43], odvozuje schémata XSD, využívá metody ACO, kritéria *sk*-string, podtřídy (*k,h*)-kontextových jazyků a metriku MDL. Zaměřuje se také na permutační operátory, možnost definovat více kořenových elementů dokumentu a elementy se stejným názvem, ale jinou strukturou a sémantikou.

3.7.7 XStruct

Tento algoritmus popsaný v [18] používá stejnou metodu slučování více definic (regulárního výrazu) jednoho elementu jako výše uvedený XTRACT (faktorizaci, více bylo uvedeno v [43]). Řeší i atributy a datové typy. Neumí zacházet s více kořenovými prvky dokumentu tak, jak je povoluje XML Schema, a také neumožňuje práci s více definicemi pro stejný název elementu ovšem s jiným kontextem [6]. Generuje tedy XSD, ale v omezeních pro DTD. Skládá se z pěti modulů:

1. Modul pro odvození modelu obsahu elementů
2. Modul pro práci s atributy
3. Modul pro rozpoznávání datových typů
4. Faktorizační modul
5. Modul pro tisk schématu

Vstup je parsován pomocí SAX, potom je předáván prvním třem zmíněným modulům. Z prvního modulu jde řízení programu do čtvrtého a do pátého potom vstupují data z modulu č. 2, 3 a 4. Informace o struktuře elementů jsou uchovávány v tomto tvaru:

$$E := (T_1 ... T_k)^{<min,max>}$$

kde T_n je *term*. *Term* je definován jako řetězec symbolů:

$$T_n := (s_{n1}^{opt} ... s_{nj}^{opt})^{<min,max>}$$

nebo výběr:

$$T_n := (s_{n1}^{opt}|...|s_{nj}^{opt})^{<min,max>}$$

kde $min \in \{0, 1\}, max \geq 1$ a $opt \in \{false, true\}$. Symbol s_{lm} je identifikátor XML elementu (název), *opt* vyjadřuje povinnost výskytu. Model obsahu elementu je tedy reprezentován rozšířeným regulárním výrazem. Algoritmus takto postupně nastřádá při načítání vstupních dokumentů definice pro každý element a následně faktorizuje definice stejných elementů dohromady.

Faktorizace definic vyjádřených regulárním výrazem probíhá tak, že se vyhledají společné prefixy a sufixy a zbylé střední části se spojí operátorem pro výběr.

Práce s atributy byla již blíze popsána v sekci 3.6.1 a odvozování datových typů v sekci 3.6.2.

3.7.8 iXSD

Algoritmus iXSD popsaný v [6] využívá SORE (viz sekce 3.2.4), zároveň zavádí k-local (viz sekce 3.6.4) jako vlastnost XSD, kdy všechny definice elementů závisí pouze na jejich k předcích). Tímto definuje identifikovatelnou podtřídu jazyka.

Algoritmus iXSD postupně používá tyto algoritmy:

- *iSOA* - algoritmus, který na základě vstupních řetězců odvodí SOA (viz definice 3.3)
- *ToSORE* - algoritmus pro odvození SORE ze SOA
- *iLocal* - vznikne složením iSOA a ToSORE
- *MINIMIZE* - k minimalizaci definic typů (sloučí typy se stejnou definicí, ale různým identifikátorem)
- *REDUCE* - využívá algoritmus MINIMIZE, slučuje dostatečně podobné definice typů (porovnává jejich editační vzdálenost se zadaným prahem)

Algoritmus iXSD je potom popsán takto:

$$iXSD(k, C) := REDUCE(iLocal(k, C))$$

kde k je počet předků, podle kterých je identifikován daný element, a C je vstupní množina XML dokumentů

Nejprve algoritmem *iLocal* odvodí z dostatečně velké množiny vstupních dokumentů strukturu budoucího XSD. Dostatečně velký vstupní vzorek (vzniklý SOA úplně popisuje hledaný jazyk) definují takto:

Definice 3.7 *Mějme SORE r a vstupní vzorek $S \subseteq L(r)$, který obsahuje všechny řetězce z $L(r)$ délky nejvíce $2n$, kde n je počet názvů elementů v r. Potom $L(iSOA(S)) = L(r)$.*

Poté algoritmus ToSORE odvodí SORE.

3.7.9 ECFG

Algoritmus uvedený v [9] se zaměřuje na některé možnosti, které má XML Schema oproti DTD. Odvozuje počty výskytů podelementů a číselné datové typy. Pracuje s rozšířenými bezkontextovými gramatikami (viz sekce 3.2.1), které dodefinovává takto:

Definice 3.8 *Rozšířená bezkontextová gramatika (ECFG) je definována pěticí* $G = (T, N, D, \sigma, S)$*, kde* T*,* N *a* D *jsou disjunktní abecedy terminálů, neterminálů a datových typů.* $T = T_1 \cup T_2$ *tak, že* $T_1 \cap T_2 = \emptyset$ *a mezi* T_1 *a* T_2 *existuje bijekce.* $S \in N$ *je počáteční neterminál,* σ *je konečná množina pravidel ve tvaru* $A \to \alpha$*,* $A \in N$ *a* α *je rozšířený regulární výraz složený z termů, kde jeden term* $t \in T_1(N \cup D)T_2$*, tento zápis zkracujeme na* $T : (N \cup D)$*.*

Souvislost ECFG a XML Schema včetně příkladů je znázorněna v tabulce 3.1.

XML Schema	**ECFG**	**Příklad**
název elementu (tag)	terminál	jmeno, kontakt
jednoduchý datový typ	datový typ	`String`
komplexní typ	neterminál	$\langle OsobaTyp \rangle$, $\langle KontaktTyp \rangle$
definice elementu	pravidlo	$\langle OsobaTyp \rangle \to$ jmeno:`String`
definice komplexního typu	jedno nebo více pravidel	(kontakt:$\langle KontaktTyp \rangle$)*

Tabulka 3.1: Tabulka souvislosti ECFG a XML Schema

Vstupní dokumenty jsou potom reprezentovány jako strukturované příklady pro odvození takovéto rozšířené bezkontextové gramatiky a algoritmus odvození probíhá takto:

1. Reprezentace vstupních XML dokumentů ve formě strukturovaných příkladů I
2. Odvození rozšířené bezkontextové gramatiky G z I
   (a) Vytvoření počáteční množiny neterminálů N
   (b) Sloučení neterminálů v N, které mají stejný nebo podobný obsah a kontext
   (c) Odvození nejužších datových typů (viz sekce 3.6.2)
   (d) Zobecnění obsahů v pravidlech do tvaru rozšířeného regulárního výrazu
3. Transformace výsledné G do definice XML Schema

Slučování neterminálů probíhá na základě podobnosti jejich obsahu a kontextu. Obecně se při odvozování bezkontextové gramatiky slučují neterminály, které mají identické pravé strany pravidel (stejný obsah) a při výskytu na pravých stranách pravidel mají stejný kontext. Tedy:

$$A \to \alpha, B \to \alpha \in \sigma \implies A = B$$

$$Aa \to \alpha B \beta, A \to \alpha C \beta \in \sigma \implies B = C, \alpha, \beta \in (N \cup D)^*$$

Kvůli zaručení determinismu je nutné nejprve odstranit výskyt nejednoznačných termů, tedy pokud se na pravé straně jednoho pravidla nachází například *term* $t : A$ a zároveň term $t : A'$, potom musí dojít ke sloučení neterminálů A a A' Toto odpovídá podmínce pro XML Schema, že jeden tag nesmí mít v jednom modelu obsahu definovány různé typy. Tento typ determinismu je označován v původním textu jako vertikální, druhým typem je determinismus horizontální. Zde se jedná o determinismus na úrovni regulárního výrazu na pravých stranách pravidel a odpovídá determinismu, který jsme zmínili již v podkapitole 3.5. Pro jeho odstranění je autory algoritmu ECFG využíváno hledání orbitů.

Po sloučení neterminálů a kroku pro odvození datových typů jsme získali sadu pravidel, kde na pravé straně jsou většinou disjunkce původních definic obsahů. Pro další zobecnění a přesnější popis pravidla se použije speciální konečný automat generovaný z dvojic vyskytujících se v řetězci (detaily zde nebudeme uvádět), celá akce probíhá takto:

1. Vygenerování konečného automatu pro pravou stranu
2. Převedení konečného automatu na deterministický (hlídání orbitů viz podkapitola 3.5)
3. Převedení automatu na ekvivalentní regulární výraz
4. Upřesnění výrazu pomocí nahrazení operátoru * přesným rozsahem počtu výskytů, v tomto kroku se používá opět konečný automat, do kterého dopisujeme počty průchodů jednotlivými hranami

3.7.10 Logická indukce

Jako poslední si uvedeme originální implementaci postavenou na predikátové logice z [11]. Pro odvozování schématu využívá logické programování a nástroj Progol. Postup algoritmu je takovýto:

1. Vygenerování predikátů na základě vstupních XML dokumentů
2. Odvození pravidel na základě predikátů
3. Odvození pravidel z predikátů pomocí ontologie
4. Na základě předchozích dvou bodů se odvodí XML Schema

Pomocí ontologického slovníku jsou vyhodnocovány i podobnosti datových typů.

Kapitola 4

Návrh inferenčního algoritmu

Po prozkoumání již dříve publikovaných metod z předchozí kapitoly se objevily některé náměty na vlastní návrh metody a implementaci, stávajícími metodami se inspirujeme a pokusíme se do této oblasti nově přinést interakci s uživatelem.

4.1 Inspirace stávajícími metodami

Výchozí inspirací pro nový algoritmus bude algoritmus ECFG (viz sekce 3.7.9). Tento algoritmus nejvíce využívá možností, které přináší XML Schema oproti DTD, a využití formálního základu problému umožní budoucí zkoumání a vylepšování metody i z formálního hlediska.

Pro výslednou organizaci implementace, tedy rozdělení řešení problému do modulů, se inspirujeme algoritmem XStruct (viz sekce 3.7.7), přičemž nová metoda bude také načítat vstupní dokumenty pomocí SAX, tím připravíme algoritmus i na zpracovávání větších dokumentů, protože SAX je méně paměťově náročný než DOM.

4.2 Interakce s uživatelem

Jako novinku v této oblasti zavedeme interakci s uživatelem, které se dosud neobjevila v žádném prozkoumaném algoritmu. Přínos zavedení interakce s uživatelem spočívá v nalezení ideální míry zobecnění schématu (viz podkapitola 3.3), v této fázi odvozování využívají prozkoumané metody předem definovaných metrik a heuristik, výsledek ale nemusí být z hlediska uživatele svým zobecněním optimální.

Z formálního hlediska očekáváme, že interakcí s uživatelem a jeho vhodným dotazováním získáme pro odvození gramatiky kromě vstupních pozitivních příkladů i příklady negativní, čímž můžeme reagovat na důkaz neidentifikovatelnosti jazyka pouze z pozitivních příkladů, který uvedl Gold v [16].

Pro budoucí vývoj metody bude zejména důležité řešit takovou koncepci pokládání dotazů, aby uživatel nebyl dotazován přespříliš a práce s nástrojem zůstávala uživatelsky příjemná. Nabízí se pomocné využití již používaných metrik, které jsme uvedli u přehledu algoritmů, případně zavedení veličin vztahujících se k popisu míry obecnosti schématu pro konkrétní sadu vstupních dokumentů, k jejichž hodnotám se budeme snažit v interakci s uživatelem dojít, a budeme tak schopni predikovat jeho rozhodování.

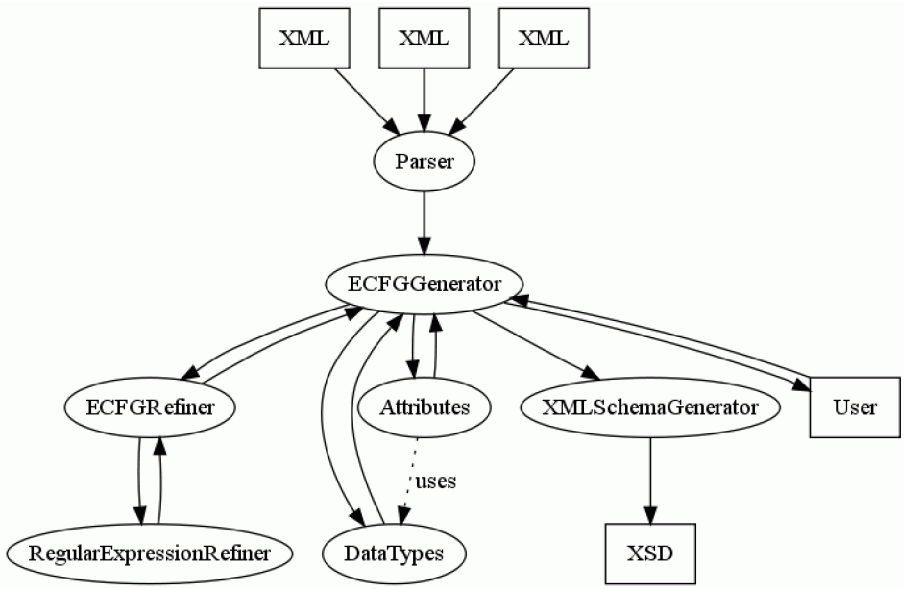
4.3 Modulární řešení

Řešení celého odvozovacího problému jsme rozdělili do modulů, na obrázku 4.1 je znázorněno výsledné schéma celého nástroje. Každý z modulů řeší specifickou část odvozování výsledného schématu.

Odvozování schématu začíná načtením vstupních XML dokumentů, toto provádí modul *Parser*. Následně je odvozena prvotní ECFG, kterou poté při interakci s uživatelem vylepšuje modul *ECFGRefiner* spolupracující s modulem pro odvozování regulárních výrazů na pravých stranách pravidel (*RegularExpressionRefiner*).

Součástí odvozování schématu je i upřesnění pravidel pro atributy elementů a odvození jednoduchých datových typů elementů a atributů, toto náleží modulům *Attributes* a *DataTypes*. Implementace těchto modulů je inspirována algoritmy zmíněnými v podkapitole 3.6, také zde můžeme využít interakci s uživatelem. V závěru je modulem *XMLSchemaGenerator* vygenerována konečná reprezentace XSD.

Schéma výsledného nástroje je znázorněno na obrázku 4.1.



Obrázek 4.1: Schéma nástroje

Popišme si nyní detailněji funkci jednotlivých modulů.

Parser Modul pro načtení vstupních XML dokumentů. Vstupní dokumenty načítá použitím API SAX a posílá jejich zjednodušenou reprezentaci modulu *ECFGGenerator*.

ECFGGenerator Modul pro vytvoření prvotního schématu zapsaného v ECFG. Na základě vstupu posílaného z modulu *Parser* vytváří prvotní reprezentaci ECFG. V tuto chvíli ještě ECFG nevyjadřuje správné schéma, toto schéma je nedeterministické, přesně popisuje vstupní dokumenty a musí být dále vylepšováno moduly *ECFGRefiner* a *RegularExpressionRefiner*.

ECFGRefiner Modul pro práci s ECFG. Tento modul slouží k vyhledávání podobnosti pravých stran pravidel a slučování neterminálů. Ve spolupráci s uživatelem bude nalézat ideální zobecnění gramatiky. Stará se o odstraňování vertikálního nedeterminismu.

RegularExpressionRefiner Modul pro odvození regulárního výrazu na pravých stranách pravidel. Jeho úlohou je odstranění horizontálního nedeterminismu a zobecňování regulární gramatiky, opět ve spolupráci s uživatelem, vyhledávání vzorů a určování počtu opakování skupin.

Attributes Modul pro odvození atributů. Určuje volitelnost/povinnost atributů náležících komplexnímu typu a ve spolupráci s modulem *DataTypes* odvozuje zároveň jejich jednoduché datové typy.

DataTypes Modul pro odvození jednoduchých datových typů. Přiřazuje vstupním hodnotám atributů a jednoduchých či smíšených elementů datový typ, detaily tohoto odvozování byly již zmíněny v podkapitole 3.6.

XMLSchemaGenerator Modul pro odvození schématu z ECFG. Z vygenerované finální ECFG a informací o atributech a datových typech vytiskne na výstup výsledné XML Schema, opět s použitím API SAX.

Kapitola 5

Implementace

V této kapitole popíšeme implementaci XML editoru a inferenčního nástroje, který využívá editor jako rozhraní při komunikaci s uživatelem.

Inferenční nástroj a editor byl implementován v jazyce Java za použití knihovny uživatelských prvků Swing, to dalo vzniknout multiplatformnímu produktu, který je uživatelům dostupný z různých operačních systémů.

5.1 XML editor

Podle zadání vznikl XML editor se zvýrazňováním XML syntaxe, který je navíc multiplatformní. Mimo běžných operací s textem má implementovány i další funkce pro usnadnění práce s XML dokumenty jako vestavěnou validaci dokumentu podle schématu, doplňování párového tagu, možnost automatického odsazování otevřeného dokumentu a v neposlední řadě umí odvozovat z právě otevřeného dokumentu XML schéma. V editoru je také umožněno uživatelské nastavení prostředí, které je ukládáno do připojeného XML souboru.

Pro implementaci zvýrazňování XML syntaxe byl využit existující volně dostupný projekt zpřístupňující komponentu `XmlTextPane`, která rozšiřuje standardní `JTextPane` o možnost zvýraznění XML syntaxe [37]. Tato komponenta byla dále přizpůsobena možnosti načítání barevného schématu z uživatelem definovaného nastavení.

Editor je také vybaven zobrazením nápovědy, pro implementaci prohlížeče HTML nápovědy posloužil projekt `HelpGUI` [42] distribuovaný pod licencí GPL.

Na uživatelské prostředí implementovaného XML editoru je možné nahlédnout v dodatku B.

5.2 Inferenční nástroj

Inferenční nástroj byl implementován podle modulárního návrhu z kapitoly 4, řeší odstraňování nedeterminismu podle požadavků W3C konzorcia a zavádí první interakci s uživatelem v oblasti odvozování atributů.

5.2.1 Implementované moduly

Implementace inferenčního algoritmu v tuto chvíli odpovídá zavedení funkčnosti v oblasti vstupně-výstupních modulů *Parser* a *XMLSchemaGenerator*, které pro načítání a zápis XML dokumentů využívají API SAX. Prvotní reprezentaci ECFG vytváří úspěšně modul

ECFGGenerator a pro odvozování pravidel pro výskyt atributů byl implementován modul *Attributes*.

Moduly *ECFGRefiner* a *RegularExpressionRefiner* pro vylepšování ECFG a pravých stran pravidel jsou nyní nahrazeny jednodušší implementací, kdy bylo ale již potřeba zavést prostředky pro hlídání horizontálního a vertikálního determinismu, jak jsme jej uvedli u popisu algoritmu ECFG (viz sekce 3.7.9).

Modul *Datatypes* není v současné době zaveden a všechny jednoduché datové typy jsou odvozovány jako typ `string`, který je v hierarchii datových typů, jak jsme si ji naznačili v podkapitole 3.6, nejvýše.

5.2.2 Odstraňování nedeterminismu

Vertikální nedeterminismus je odstraňován hlídáním výskytu nejednoznačných termů podle algoritmu ECFG a následným slučováním neterminálů. Během slučování pravých stran pravidel se ale může vyskytnout horizontální nedeterminismus, jak jsme jej popsali v podkapitole 3.5. Tento nedeterminismus je momentálně řešen převodem vzniklého regulárního výrazu na prefixový strom (viz sekce 3.2.3), který využíváme jako speciální případ deterministického konečného automatu.

Během vytváření prefixového stromu je ošetřován výskyt jednoduchých opakování za sebou následujících elementů a jsou upřesňovány počty výskytů.

5.2.3 Interakce s uživatelem

Implementace již zavádí prostředky pro interakci s uživatelem, kterou momentálně využívá modul *Attributes*, kdy má uživatel možnost rozhodnout o povinném či volitelném výskytu atributů. Uživateli je předložen dotaz s informací o výskytu upřesňovaného typu v původním XML souboru, tím uživateli předkládáme potřebné informace, na základě kterých může upřesnit definici podle svých priorit.

5.2.4 Propojení s XML editorem

Metoda je spustitelná z menu XML editoru, inference přijímá na vstup aktuálně otevřený XML dokument a generuje výstup do zvoleného výstupního souboru.

Inferenční metoda definuje rozhraní `UserInteractive`, které editor implementuje dialogem `InferenceDialog`, tímto způsobem spolu obě části komunikují a předávají si informace pro zobrazení uživateli a následně informace od uživatele získané. Inferenční dialog zobrazuje uživateli výskyty upřesňovaného typu přímo v prostředí editoru přehledným zvýrazněním relevantních řádků.

Ukázka interakce s uživatelem z prostředí XML editoru je znázorněna v dodatku C.

Detailnější pohled na celou implementaci nabízí programová dokumentace, která se nachází na přiloženém CD.

5.3 Složitost algoritmu

Časově nejnáročnější částí navrženého algoritmu je slučování neterminálů. Pokusíme se nyní odvodit asymptotickou časovou složitost pro nejhorší případ. Označme `n` jako počet značek ve vstupním dokumentu. Nejhorším případem na vstupu z hlediska časové náročnosti provedení, který může nastat, je XML dokument, kde jsou všechny tyto značky uvedeny

jako přímé podelementy kořenového elementu dokumentu, typy těchto `n` značek jsou spolu potom porovnávány a slučovány.

Provedli jsme experimenty s různým počtem `n` těchto značek a měřili jsme čas, po který trvá provedení odvozovacího algoritmu (pomocí příkazu `time`). Výsledky měření jsme zanesli do grafu, který jsme vykreslili nástrojem `gnuplot`, a získaná data jsme nechali proložit křivkou $f(x) = a * log^2(x) * x^3 + b$ (kde parametry a a b jsou po řadě počet značek a naměřený čas), která nejlépe odpovídala průběhu naměřených hodnot. Výsledný graf je k nahlédnutí v příloze na obrázku D.1.

Tímto jsme experimentálně odhadli třídu asymptotické časové složitosti pro nejhorší případ, kterou jsme stanovili na $O(log^2 n * n^3)$.

Paměťová složitost bude linární, tedy $O(n)$.

Kapitola 6

Testování

V této kapitole zhodnotíme reálný přínos a úspěch implementovaného XML editoru a inferenční metody. Zároveň budeme srovnávat nově implementovaný inferenční nástroj s existujícími nalezenými nástroji, jejichž kompletní stručný přehled včetně tabulky srovnávající jejich vlastnosti je k nahlédnutí v dodatku F.

Editor i metoda byly testovány na operačních systémech Windows XP SP2 a Kubuntu 8.04 s běhovým prostředím Sun Java JRE 6.

6.1 XML editor

Testování implementovaného XML editoru prokázalo, že je editor v rozsahu svých požadovaných schopností funkční, oproti jednoduchým textovým editorům zavádí několik vylepšení pro práci s XML dokumenty. Hlavní výhodou oproti stávajícím XML editorům je implementace interaktivního prostředí pro podporu prezentované odvozovací metody. Stávající specializované XML editory ale jinak nepřekonává.

6.2 Inferenční metoda

Implementovaná metoda byla otestována na volně dostupných XML souborech ze zdrojů [7] a [17], které je možno také nalézt na přiloženém CD. Ve všech případech metoda produkovala správné schéma, které následně úspěšně validovalo vstupní XML dokument. Problémy ale nastávají u zpracovávání souborů s větším počtem značek, kdy je jejich zpracování a odvození schématu hodně časově náročné. Toto bude potřeba do budoucna řešit rozdělením implementace do více vláken a návrhem efektivnějšího průběhu inferenčního algoritmu.

6.2.1 Výstup metody

Implementací odvozování podle algoritmu ECFG jsme získali metodu, která produkuje schéma odpovídající přístupu při vytváření schématu v sekci 2.3.5 uvedeném jako „Metoda slepého Benátčana“, který byl shledán jako nejvýhodnější z hlediska úspornosti a znovupoužitelnosti.

Příklad vstupu a odpovídajícího výstupu naší metody je uveden v dodatku E.

6.2.2 Vhodné použití metody

Současná implementace metody neobsahuje sofistikované vylepšování regulárních výrazů, počty opakování podelementů se určují pouze při výskytů opakování jednoho elementu, nikoliv skupin, a tak metoda není příliš vhodná pro vstupní dokumenty, které obsahují v podelementech takovéto složitější vzory.

Metoda je na druhou stranu velmi vhodná pro vstupní dokumenty s výskytem atributů, protože právě v tomto případě je momentálně možné vyzkoušet nově zavedenou interakci s uživatelem.

6.3 Srovnání s existujícími inferenčními nástroji

Vlastnosti implementovaného nástroje byly srovnávány s vlastnostmi nástrojů uvedených v dodatku F. Z nich byly vybrány volně dostupné programy generující schéma v jazyce XML Schema, které byly zároveň v období testování (květen 2009) dostupné ze svého uvedeného umístění. Multiplatformnost nebyla do podmínky zahrnuta, i když implementovaný nástroj tuto vlastnost splňuje.

Odvozovací dovednosti byly detailněji zkoumány u těchto nástrojů: Trang, XML Beans Tools, nástroje dostupné z Visual Studia a platformy .NET a online nástroje Allora XML Utilities a XML Schema Generator.

Sledovali jsme výstupní uspořádání XSD (viz přístupy pro vytváření schématu v sekci 2.3.5), schopnost odvozování datových typů, vyhledávání vzorů ve výskytech podelementů (tedy práce s regulárními výrazy), odvozování atributů a interakci s uživatelem.

6.3.1 Uspořádání výsledného XSD

Z prozkoumaných nástrojů nejlépe zachází s výstupním uspořádáním XSD nástroj XML Beans, kdy je možné tento model uživatelsky vybrat. Ostatní nástroje produkují vždy jeden výstupní model, stejně jako implementovaná metoda, která ovšem produkuje model, který je považován za nejvýhodnější. Umožňuje totiž znovupoužitelnost definovaných datových typů, na kterém je založeno zobecňování schématu podle navrženého algoritmu. Zároveň je schéma vytvořené podle tohoto přístupu dobře čitelné.

6.3.2 Vyhledávání vzorů

U zmíněných testovaných nástrojů nebyla nalezena schopnost odvození složitějších pravidel pro výskyt podelementů. Pro ilustraci například při opakovaném výskytu sekvence podelementů A, B a C, který by se regulárním výrazem dal vyjádřit jako $(ABC)^+$, zavádějí zápis typu $(A|B|C)^*$. Tedy zde dochází k přílišné generalizaci schématu.

Implementovaná metoda v tuto chvíli vytváří zápis ve tvaru $ABC(ABC(ABC)?)?$, což je dáno použitím prefixového stromu bez dalších úprav, a přesně tak popisuje vstupní dokument, ve kterém se skupina ABC za sebou objevila jednou, dvakrát a třikrát.

Tato funkčnost bude řešena modulem *RegularExpressionRefiner*, který bude odhalovat opakující se podsekvence a v interakci s uživatelem určovat přesnější pravidla pro počet opakování.

6.3.3 Odvozování datových typů

Některé z vyzkoušených nástrojů umožňují i odvození jednoduchých datových typů elementů a atributů. Tato funkčnost, která v návrhu metody odpovídá modulu *Datatypes*, není v současné době implementována.

6.3.4 Odvozování atributů

Odvozování atributů opět umožňují všechny prozkoumané nástroje, implementovaná metoda navíc umožňuje z uživatelského rozhraní pohodlně rozhodnout o povinnosti/volitelnosti výskytu atributů.

6.3.5 Interakce s uživatelem

Všechny prozkoumané nástroje generují schéma zcela automaticky bez dotazování uživatele. Jedinou možností, jak ovlivnit výsledek, je ruční editace výstupního souboru. Zde navržená metoda umožňuje uživateli jako jediná ovlivnit výsledek bez potřeby znalosti jazyka XML Schema.

6.3.6 Shrnutí

Ze srovnání s testovanými nástroji vyplývá, že implementovaná metoda v tuto chvíli jako jediná zavádí interakci s uživatelem, produkuje čitelnější výstup než většina prozkoumaných nástrojů (dáno již zmiňovaným modelem uspořádání XSD) a její budoucí vývoj bude směrovat k lepšímu vyhledávání vzorů v opakování podelementů a hledání optimální míry zobecnění schématu podle preferencí uživatele.

Kapitola 7

Závěr

Práce přináší ucelený přehled odvozovacích metod, které byly v rámci studia této problematiky objeveny v publikovaných odborných článcích a pracech. Těmito metodami jsme se inspirovali při návrhu nové inferenční metody, kdy využíváme formálního základu problému a zavádíme nově interakci s uživatelem, která se dosud neobjevila v žádném prozkoumaném algoritmu.

Byl implementován multiplatformní XML editor, který zpřístupňuje zatím zjednodušenou implementaci nově navržené metody, kde se již nově objevuje navržený koncept interakce s uživatelem v rámci odvozování atributů.

Navržený inferenční nástroj byl srovnán s existujícími otestovanými odvozovacími nástroji, toto srovnání mělo pro nástroj dobrý výsledek a s dalším pokračováním implementace a vývoji v současné době neimplementovaných modulů očekáváme, že nástroj bude vykazovat ještě lepší vlastnosti.

Další vývoj metody může vést směrem k implementaci modulů pro vylepšování použité rozšířené bezkontextové gramatiky, kdy zavedením interakce s uživatelem získáme optimální míru zobecnění generovaného schématu a budeme tak originálně řešit problém nalezení této míry. Formální základ umožní budoucí vylepšování a zkoumání metody i z formálního hlediska.

Literatura

[1] Ahonen, H.: *Generating Grammars for Structured Documents Using Grammatical Inference Methods*. Dizertační práce, Department of Computer Science, University of Helsinki, Finland, 1996.

[2] Altova: XML Schema and DTD Tools. [online], cit. 2009-01-10.
URL http://www.altova.com/features_dtdschema.html

[3] Berstel, J.; Boasson, L.: XML grammars. In *Mathematical Foundations of Computer Science 2000, Bratislava, Lect. Notes Comput. Sci. 1893*, Springer, 2000, s. 182–191.

[4] Bex, G. J.; Gelade, W.; Neven, F.; aj.: Learning deterministic regular expressions for the inference of schemas from XML data. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, New York, NY, USA: ACM, 2008, ISBN 978-1-60558-085-2, s. 825–834.

[5] Bex, G. J.; Neven, F.; Schwentick, T.; aj.: Inference of concise DTDs from XML data. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, VLDB Endowment, 2006, s. 115–126.

[6] Bex, G. J.; Neven, F.; Vansummeren, S.: Inferring XML schema definitions from XML data. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, VLDB Endowment, 2007, ISBN 978-1-59593-649-3, s. 998–1009.

[7] Bosak, J.: XML examples. [online], cit. 2009-05-12.
URL http://www.ibiblio.org/bosak/

[8] Carrasco, R. C.; Oncina, J.: Learning stochastic regular grammars by means of a state merging method. Springer-Verlag, 1994, s. 139–152.

[9] Chidlovskii, B.: Schema Extraction from XML Data: A Grammatical Approach. In *KRDB'01 Workshop (Knowledge Representation and Databases)*, 2000.

[10] Demo Source and Support: Generate XML Schema from XML document. [online], cit. 2009-01-10.
URL
http://www.java2s.com/Code/VB/XML/GenerateXMLSchemafromXMLdocument.htm

[11] Eki, M.; Ozono, T.; Shintani, T.: Extracting XML schema from multiple implicit XML documents based on inductive reasoning. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, New York, NY, USA: ACM, 2008, ISBN 978-1-60558-085-2, s. 1219–1220.

[12] Fernau, H.: Learning XML grammars. In *In Machine Learning and Data Mining in Pattern Recognition MLDM'01, number 2123 in LNCS*, Springer-Verlag, 2001, s. 73–87.

[13] Fitzgerald, M.; Tomoharu, A.: An Introduction to the Relaxer Schema Compiler. [online], cit. 2009-01-10.
URL http://www.xml.com/pub/a/2003/02/19/relaxer.html

[14] Foundation, T. A. S.: XMLBeans Tools. [online], cit. 2009-01-10.
URL http://xmlbeans.apache.org/docs/2.0.0/guide/tools.html#inst2xsd

[15] Gionis, A.; Rastogi, R.; Seshadri, S.; aj.: Xtract: a system for extracting document type descriptors from xml documents. In *In ACM SIGMOD*, ACM Press, 2000, s. 165–176.

[16] Gold, E. M.: Language identification in the limit. *Information and Control*, ročník 10, č. 5, 1967: s. 447–474.
URL http://www.isrl.uiuc.edu/~amag/langev/paper/gold67limit.html

[17] Group, U. D.: UW XML Repository. [online], cit. 2009-05-12.
URL http://www.cs.washington.edu/research/xmldatasets

[18] Hegewald, J.; Naumann, F.; Weis, M.: XStruct: Efficient Schema Extraction from Multiple and Large XML Documents. In *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops*, Washington, DC, USA: IEEE Computer Society, 2006, ISBN 0-7695-2571-7, str. 81.

[19] HiT Software, Inc: XML Tools : DTD, XML schema and XML document conversion software tool : XML Utilities. [online], cit. 2009-01-10.
URL http://www.hitsw.com/xml_utilites/

[20] JetBrains: XML/XSL editor. [online], cit. 2009-01-10.
URL http://www.jetbrains.com/idea/features/xml_editor.html#link2

[21] Kay, M. H.: DTDGenerator. [online], cit. 2009-01-10.
URL http://saxon.sourceforge.net/dtdgen.html

[22] Kosek, J.: Kapitola 3. XML schémata. [online], cit. 2009-02-06.
URL http://www.kosek.cz/xml/schema/wxs.html

[23] Lee, D.; Chu, W. W.: Comparative Analysis of Six XML Schema Languages. *ACM SIGMOD Record*, ročník 29, 2000: s. 76–87.

[24] Meduna, A.; Lukáš, R.: *Formální jazyky a překladače, studijní materiály.* FIT VUT v Brně, 2009.

[25] Microsoft Corporation: An Introduction to the XML Tools in Visual Studio 2005. [online], cit. 2009-01-10.
URL
http://msdn.microsoft.com/en-us/library/aa302298.aspx#xmltools_topic3

[26] Microsoft Corporation: DataSet.::.WriteXmlSchema Method. [online], cit. 2009-01-10.
URL http://msdn.microsoft.com/en-us/library/system.data.dataset.writexmlschema.aspx

[27] Microsoft Corporation: Using the XSD Inference Utility. [online], cit. 2009-01-10.
URL http://msdn.microsoft.com/en-us/library/aa302302.aspx

[28] Microsoft Corporation: XML Schema Definition Tool (Xsd.exe). [online], cit. 2009-01-10.
URL http://msdn.microsoft.com/en-us/library/x6c1kb0s(VS.71).aspx

[29] Mlýnková, I.: XML Schema Inference: A Study. Technická zpráva, Charles University, Prague, Czech Republic, 2008.

[30] Moh, C.-H.; Lim, E.-P.; Ng, W.-K.: Re-engineering structures from Web documents. In *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, New York, NY, USA: ACM, 2000, ISBN 1-58113-231-X, s. 67–76.

[31] Novell: XML Schema Inference. [online], cit. 2009-01-10.
URL http://www.mono-project.com/XML_Schema_Inference

[32] PlanetMath: Trie. [online], cit. 2009-01-17.
URL http://planetmath.org/encyclopedia/Trie.html

[33] Progress Software Corporation: Generate XML Schema. [online], cit. 2009-01-10.
URL http://www.stylusstudio.com/autogen_xsd.html

[34] Refsnes Data: DTD Tutorial. [online], cit. 2009-02-05.
URL http://www.w3schools.com/DTD/default.asp

[35] Refsnes Data: XML Schema Tutorial. [online], cit. 2009-04-28.
URL http://www.w3schools.com/xml/

[36] Shafer, K. E.: Creating DTDs via the GB-Engine and Fred. [online], cit. 2009-01-18.
URL http://xml.coverpages.org/shaferFred1995a.html

[37] software, B.: XML syntax highlighting in Swing JTextPane. [online], cit. 2009-05-11.
URL http://www.boplicity.net/confluence/display/Java/Xml+syntax+highlighting+in+Swing+JTextPane

[38] SyncRO Soft Ltd: XML Schema Editor. [online], cit. 2009-01-10.
URL http://www.oxygenxml.com/xml_schema_editor.html

[39] Tchistopolskii, P.: DTDGenerator Web Service. [online], cit. 2008-10-10.
URL http://www.pault.com/Xmltube/dtdgen.html

[40] Thai Open Source Software Center Ltd: Trang. [online], cit. 2009-01-10.
URL http://www.thaiopensource.com/relaxng/trang.html

[41] Thai Open Source Software Center Ltd: Trang Manual. [online], cit. 2009-01-10.
URL http://www.thaiopensource.com/relaxng/trang-manual.html

[42] Thomas, A.: HelpGUI, the Java Help Viewer. [online], cit. 2009-05-12.
URL http://helpgui.sourceforge.net/

[43] Vošta, O.: *Automatická konstrukce schématu pro množinu XML dokumentů.* Diplomová práce, Katedra softwarového inženýrství, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze, 2005.

[44] Wahlin, D.: XML Schema Generator. [online], cit. 2009-01-10.
URL http://www.xmlforasp.net/CodeBank/System_Xml_Schema/BuildSchema/BuildXMLSchema.aspx

[45] Webfaq.cz: XML a značkovací jazyky, historie a vznik. [online], cit. 2009-05-11.
URL
http://www.webfaq.cz/clanek/XML-a-znackovaci-jazyky-historie-a-vznik

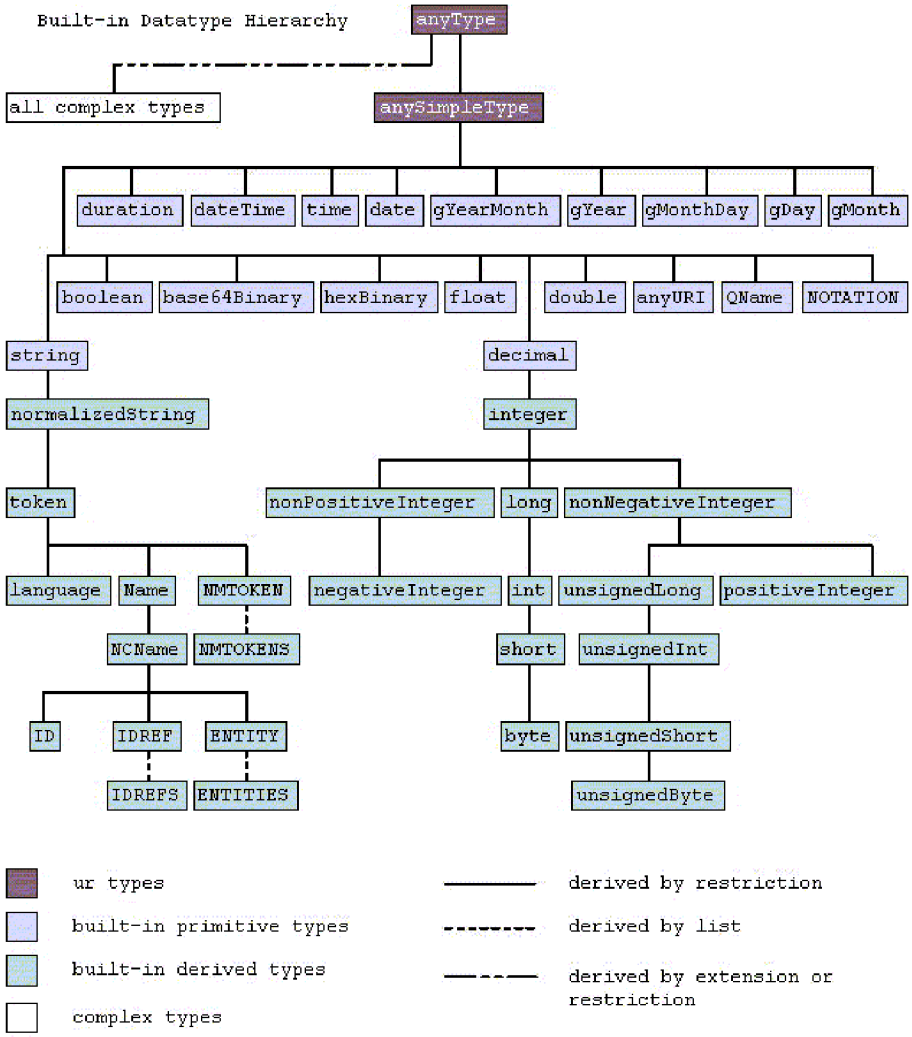
[46] Wong, R. K.; Sankey, J.: On Structural Inference for XML Data. Technická zpráva, School of Computer Science & Engineering, University of New South Wales, Sydney, Australia, 2003.

[47] World Wide Web Consortium: XML Schema Part 1: Structures. [online], cit. 2009-01-11.
URL http://www.w3.org/TR/1999/WD-xmlschema-1-19991217/

[48] World Wide Web Consortium: XML Schema Part 2: Datatypes Second Edition. [online], cit. 2009-02-06.
URL http://www.w3.org/TR/xmlschema-2/

Dodatek A

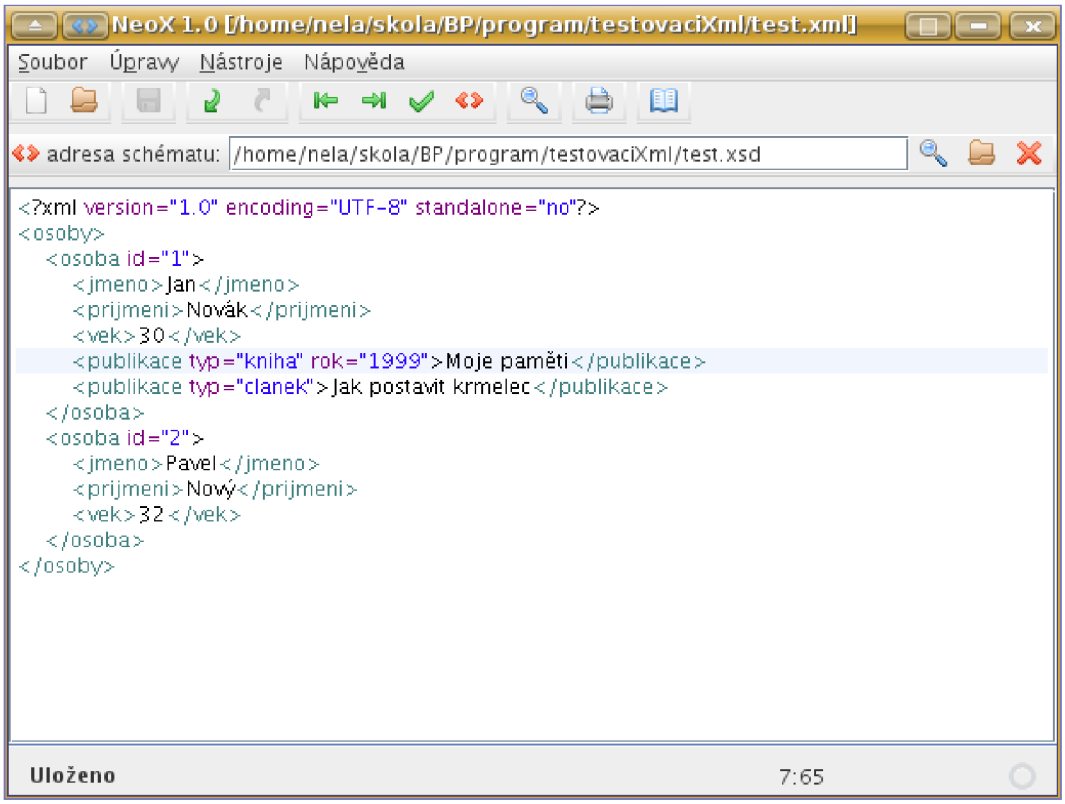
Hirearchie datových typů v jazyce XML Schema



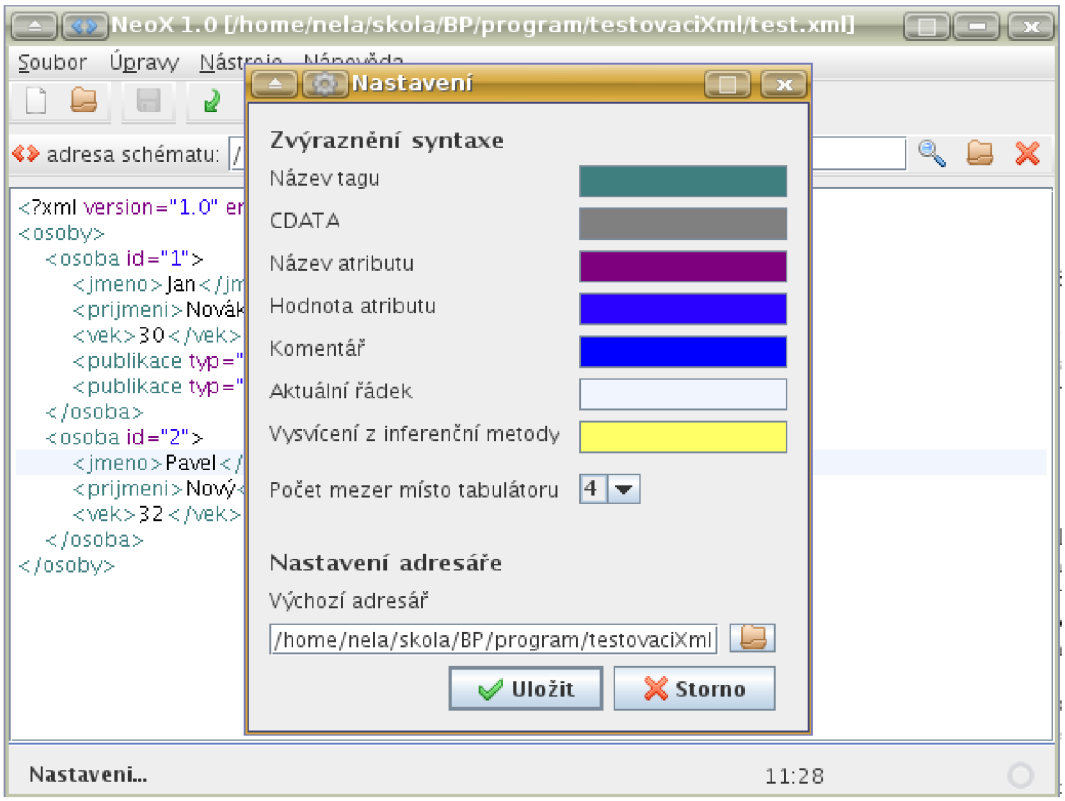
Obrázek A.1: Hierarchie datových typů jazyka XML Schema [48]

Dodatek B

Prostředí XML editoru



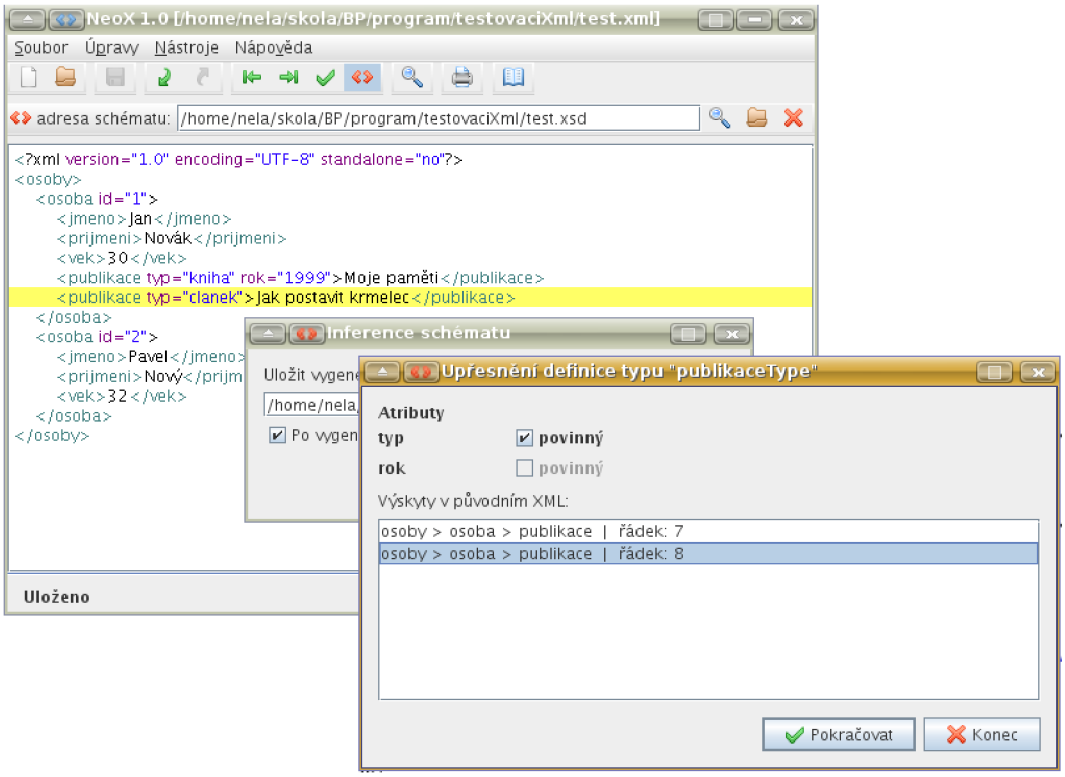
Obrázek B.1: Prostředí XML editoru



Obrázek B.2: Nastavení XML editoru

Dodatek C

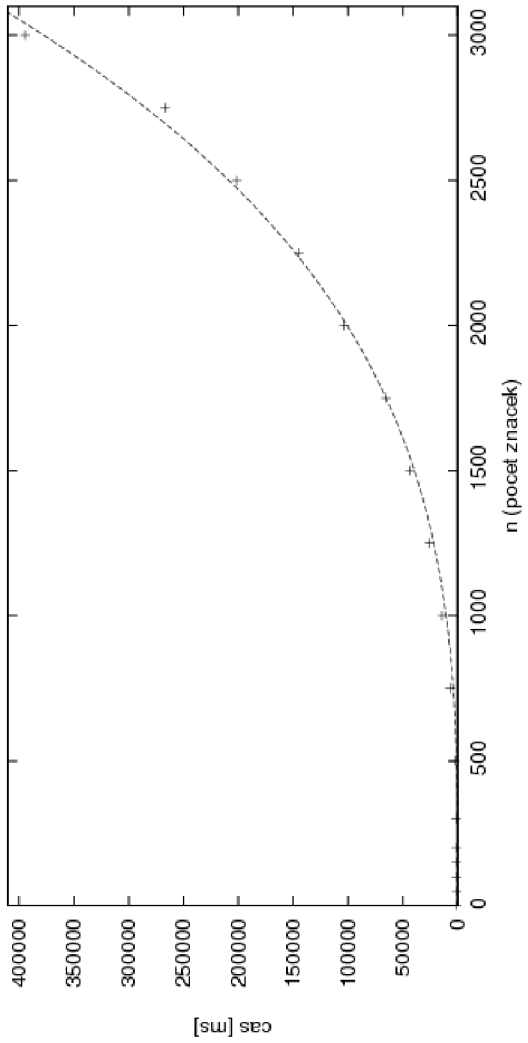
Interaktivní inference



Obrázek C.1: Interakce s uživatelem

Dodatek D

Odvození časové složitosti



Obrázek D.1: Odvození časové složitosti

Dodatek E

Vstup a výstup inferenční metody

Pro tento vstupní dokument:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<osoby>
    <osoba id="1">
        <jmeno>Jan</jmeno>
        <prijmeni>Novák</prijmeni>
        <vek>30</vek>
        <publikace typ="kniha" rok="1999">Moje paměti</publikace>
        <publikace typ="clanek">Jak postavit krmelec</publikace>
    </osoba>
    <osoba id="2">
        <jmeno>Pavel</jmeno>
        <prijmeni>Nový</prijmeni>
        <vek>32</vek>
    </osoba>
</osoby>
```

Dostáváme použitím naší metody tento výstup v jazyce XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="publikaceType" mixed="true">
        <xs:attribute name="typ" type="xs:string" use="required"/>
        <xs:attribute name="rok" type="xs:string" use="optional"/>
    </xs:complexType>
    <xs:complexType name="osobaType" mixed="true">
        <xs:sequence>
            <xs:element name="jmeno" type="xs:string"
                        minOccurs="1" maxOccurs="1"/>
            <xs:element name="prijmeni" type="xs:string"
                        minOccurs="1" maxOccurs="1"/>
            <xs:element name="vek" type="xs:string"
                        minOccurs="1" maxOccurs="1"/>
            <xs:choice>
```

```
                <xs:sequence/>
                <xs:choice>
                    <xs:sequence>
                        <xs:element name="publikace" type="publikaceType"
                                                    minOccurs="2" maxOccurs="2"/>
                    </xs:sequence>
                </xs:choice>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="osobyType" mixed="true">
        <xs:sequence>
            <xs:element name="osoba" type="osobaType"
                                                minOccurs="2" maxOccurs="2"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="osoby" type="osobyType"/>
</xs:schema>
```

Dodatek F

Existující inferenční nástroje

Programů pro inferenci schématu z XML dokumentů existuje celá řada, od jednodušších, volaných z příkazové řádky, až po rozsáhlé programy s uživatelsky příjemným ovládáním a vizualizací vytvářených schémat. Schémata je možné generovat i v rámci některých vývojových prostředí, kam se funkčnost přidává formou pluginu, nebo je již zaintegrována. Na generování schémat mysleli také vývojáři softwarové platformy .NET, i zde můžeme nalézt několik knihoven a nástrojů. Některé jednodušší nástroje jsou dostupné z webového rozhraní.

F.1 Nástroje příkazové řádky

Jedná se vesměs o multiplatformní programy vytvořené v jazyce Java, které jsou spustitelné z příkazové řádky, názvy vstupních souborů jsou předávány jako parametry, výsledné schéma je vypsáno na standartní výstup. Pro jejich používání je nutné mít nainstalováno běhové prostředí Javy (Java Runtime Environment).

F.1.1 Relaxer

Nástroj pro generování tříd v jazyce Java z XML dokumentů a schémat, třídy pak obsahují metody umožňující přístup k položkám dokumentu a jejich modifikaci. Umí i generovat schémata z více XML dokumentů najednou. Pro svou práci využívá DOM a ze zkoumaných nástrojů má nejširší nabídku výsledných typů schématu, v jeho možnostech je i vygenerování XSL šablony [13].

F.1.2 Trang (Translator for RELAX NG Schemas)

Rozšířený nástroj pro generování schémat, bývá využíván složitějšími programy zmíněnými dále (Oxygen, Stylus Studio XML IDE, IntelliJ IDEA).

Vnitřní reprezentace schématu je založena na RELAX NG, na toto schéma a zpět jsou ostatní schémata převáděna vstupními a výstupními moduly, které jsou součástí programu. Jak už z tohoto popisu skoro vyplývá, program umožňuje i vzájemnou konverzi jednotlivých schémat. Jediné omezení se týká XML Schema, které nelze použít jako vstupní [40].

Popis všech vstupních parametrů programu se nachází v manuálu [41].

F.1.3 DTDGenerator

Generuje pouze DTD schémata z jednoho dokumentu, využívá SAX. Je možné jej bez instalace vyzkoušet online z webového formuláře [39].

Další informace včetně popisu algoritmu v [21].

F.2 Programy s grafickým uživatelským rozhraním

Nástroje v této kategorii se vyznačují vysokou mírou pohodlnosti užívání, obsahují editory XML dokumentů i schémat se zvýrazňováním syntaxe, automatickým doplňováním tagů a atributů, vizualizací (WYSIWYG editor schémat), a je možné za jejich pomoci i generovat schémata z XML dokumentů.

Jde o placené robustní nástroje, u kterých je pro nás stěžejní funkcionalita (inference schématu) pouze jednou z mnoha funkcí, a tak jí není v dokumentaci věnována příliš velká pozornost. Zároveň tyto nástroje nebyly zahrnuty do testování a srovnávaní s vyvíjenou metodou. Jedná se o: Oxygen XML Editor [38], Altova XMLSpy [2] a Stylus Studio XML IDE [33].

F.3 Vývojová prostředí a pluginy

Kromě samostatných nástrojů je možné využívat pro generování schémat i některá vývojová prostředí, kam se funkčnost přidává formou pluginu (Eclipse, Visual Studio), nebo je v prostředí již od základní instalace (IntelliJ IDEA, XML Tools ve Visual Studiu 2005).

F.3.1 Eclipse

Eclipse je multiplatformní open source vývojová platforma pro mnoho programovacích jazyků, která v základní verzi obsahuje pouze prostředky pro vývoj v Javě, umožňuje tedy pohodlnou rozšiřitelnost pluginy. Kromě výše zmíněných programů (Altova XMLSpy a Oxygen XML Editor) s integrací do vývojových prostředí je jako další inferenční nástroj pro Eclipse k dispozici i XMLBeans s nástrojem inst2xsd.

Umožňuje zadat na vstup i více XML dokumentů [14].

F.3.2 Visual Studio

Visual Studio poskytuje vývojové nástroje pro tvorbu aplikací v prostředí Microsoft Windows. Je možné do něj volitelně přidávat rozšiřující pluginy, ty, které umožňují pro nás podstatnou inferenci schémat, jsou uvedeny níže.

- *XML to Schema Inference Wizard* – plugin pro Visual Studio 2008 pro projekty psané ve Visual Basicu.
- *XML Tools* – je součástí Visual Studia 2005, měl by kromě základní práce s XML soubory a schématy umět schéma také generovat a převádět, umí pracovat i s XSLT [25].
- Z dříve zmíněných programů připomeňme *Altova XMLSpy*.

F.3.3 IntelliJ IDEA

Samostatné vývojové prostředí pro Windows a Linux pro řadu programovacích jazyků, podle dokumentace umí inferenci schémat již od základní instalace, jako již některé další nástroje zmíněné výše využívá také Trang, lze do něj zaintegrovat XMLBeans a po přídání patřičného pluginu zvládá i Relax NG [20].

F.4 .NET třídy a nástroje

Prostředí .NET obsahuje následující třídy a nástroje pro inferenci schémat.

F.4.1 XSD Inference Tool

Tento nástroj umí odvozovat schéma z více dokumentů [27].
Používá se voláním metody
`Microsoft.XSDInference.Infer.InferSchema()`.
Funkce je implementována i v Monu jako
`System.Xml.Schema.XmlSchemaInference.InferSchema()` [31].

F.4.2 Metoda třídy DataSet

DataSet je třída v .NETu, která reprezentuje vnitřní podobu dat získaných z datového zdroje. Metoda pro inferenci schématu se volá jako `DataSet.WriteXmlSchema()` [26]. Příklad použití pro náš účel najdeme na [10].

Funkčnost je dostupná i v Monu.

F.4.3 XML Schema Definition Tool

Nástroj příkazové řádky se skrývá v Xsd.exe, umí navíc generovat třídy na základě XML schématu a naopak, produkovat třídy ve Visual Basicu (databinding) a vytvářet XML schéma pro zadané sestavení (assembly) [28].

F.5 Online nástroje

Pro pohodlné a rychlé vygenerování XML schématu z jednoho dokumentu slouží některé nástroje dostupné z webového rozhraní. Součástí je formulář pro nahrání lokálního souboru na server, nebo je možné zadat kód XML dokumentu zápisem do textového pole.

- *Allora XML Utilities* – umí generovat DTD a XML Schema a taky vzájemný převod obou schémat [19].
- *XML Schema Generator* – generuje pouze XML Schema [44].
- Své webové rozhraní mají i některé již výše zmíněné nástroje (*DTDGenerator*, *XML Beans Tools*, *XSD Inference Tools*)

F.6 Tabulkový přehled

Vlastnosti všech výše uvedených nástrojů jsou vypsány v tabulce F.1. Vyplněné políčko (X) znamená, že program či funkce má danou vlastnost, v opačném případě nebyla vlastnost zjištěna nebo není pro popis daného prostředku smysluplná.

F.6.1 Zkoumané vlastnosti

- *Platforma pro použití* – popisuje, jak je nástroj dostupný pro uživatele různých operačních systémů, některé programy mají i své webové rozhraní a jsou tak dostupné všem z internetového prohlížeče (podmínkou je dostupné internetového připojení)
  - *W* – spustitelné ve Wine
- *Licence* – některé nástroje jsou omezeny licencí
  - *G* – GNU General Public License (GNU GPL)
  - *A* – Apache License 2.0
  - *M* – Mozilla Public Licence Version 1.0
  - *F* – pro opensource projekty dostupné jako freeware
  - *S* – shareware
- *Běhové prostředí* – některé nástroje jsou spustitelné za podmínky instalace potřebného běhového prostředí, případně je jejich použití s některým kladeno do souvislosti (např. .NET třídy)
- *Ovládání* – vyjadřuje, jak se k funkci nástroje přistupuje - zda z příkazové řádky, nebo má nástroj uživatelské rozhraní, případně i pokročilé uživatelské rozhraní (WYSIWYG editory)
- *Práce se schématy a dokumenty* – dostupné formáty výsledného schématu, použitelnost nástroje pro převod mezi schématy a možnost inference na základě více vstupních XML dokumentů
- *Integrace do vývojového prostředí* – nástroj nabízí rozšiřitelnost vývojových prostředí o možnost inference schémat
- *Generování tříd* – některé nástroje umožňují navíc i generování tříd, jejichž metody je pak možno volat pro snadný přístup k dokumentu s určeným schématem (databinding)

		Relaxer	Trang	DTDGenerator	Oxygen XML Editor	Altova XMLSpy	Stylus Studio XML IDE	XML to Schema I. W.	XML Tools VS 2005	XML Beans Tools	IntelliJ IDEA	XSD Inference Tool	.NET DataSet	XML Schema D. T.	Allora XML Utilities	XML Schema Generator
		Nástroje příkazové řádky			Programy s GUI			Vývojová prostředí a pluginy				.NET třídy a nástroje			Online nástroje	
Platforma pro použití	Linux	X	X	X	X	W				X	X	X	X	X	X	X
	Windows	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Web			X	X	X	X	X	X	X	X	X	X	X	X	X
	Licence	G	A	M	S	S	S			A	F					
Běhové prostředí	Mono	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Java	X	X	X	X		X	X	X	X	X	X	X	X	X	X
	.NET							X	X			X	X	X	X	X
Ovládání	Příkazová řádka	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	GUI			X	X	X	X	X	X	X	X	X	X	X	X	X
	WYSIWYG			X	X	X	X	X	X	X	X	X	X	X	X	X
	Využívá Trang		X	X	X	X	X	X	X	X	X	X	X	X	X	X
Práce se schématy a dokumenty	Xml Schema	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	DTD	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Relax NG	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Relax Core	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Konverze	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Více dokumentů	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Integrace do vývojového prostředí	Eclipse	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	IntelliJ IDEA	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Visual Studio	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Generování tříd	Java	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	VB/C#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	C++	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Obrázek F.1: Tabulka srovnání nástrojů

Dodatek G

Obsah CD

Na přiloženém CD se nachází:

- Tato technická zpráva ve formátu `pdf` a zdrojové soubory zprávy pro sázecí systém LaTeX (adresář `/zprava`)
- Zdrojové kódy aplikace (adresář `/zdrojovy_kod`)
- Spustitelný program (adresář `/program`)
- Programová dokumentace vygenerovaná nástrojem `Doxygen` (adresář `/dokumentace`)
- Sada testovacích XML souborů (adresář `/xml`)

Podrobnější poznámky k aplikaci, návod na spuštění a překlad se nacházejí v souboru `readme.txt`, manuál k programu je vytvořen jako součást aplikace a je přístupný po spuštění aplikace z menu nebo klávesou `F1`.