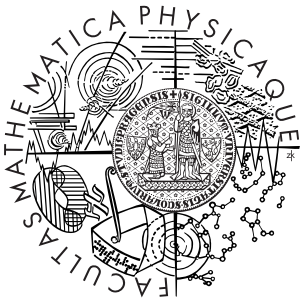
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jiří Vinárek

Nástroj pro analýzu XML dat

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Irena Mlýnková, Ph.D.
Studijní program: Informatika, programování

2009

Děkuji RNDr. Ireně Mlýnkové, Ph.D. za její rady, vstřícnost a odbornou pomoc při vedení práce. Své rodině děkuji za trpělivost, kterou se mnou měla během sepisování práce.

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

V Praze dne 2.8. 2009 Jiří Vinárek

Obsah

Název práce: Nástroj pro analýzu XML dat
Autor: Jiří Vinárek
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Irena Mlýnková, Ph.D.
e-mail vedoucího: irena.mlynkova@mff.cuni.cz

Abstrakt: Cílem práce je implementace nástroje umožňujícího provádět analýzu daných XML dokumentů, schémat DTD a XSD. Kromě analýzy dokumentů program umožňuje přehledné grafické zobrazení vstupních XML dokumentů, schémat DTD a XSD, tabulkové a grafové zobrazení výsledků, možnost srovnání výsledků odpovídajících si statistik pro XML dokumenty a jejich schémata. Výsledky je možné ukládat a později k nim přidávat nové dokumenty. Aplikace umožňuje efektivně zpracovávat i velké dokumenty. Je napsána s ohledem na snadnou rozšiřitelnost a maximální možnou parametrizovatelnost.

Klíčová slova: XML, DTD, XML Schema, Analýza

Title: A tool for analysis of XML data
Author: Jiří Vinárek
Department: Katedra softwarového inženýrství
Supervisor: RNDr. Irena Mlýnková, Ph.D.
Supervisor's e-mail address: irena.mlynkova@mff.cuni.cz

Abstract: The aim of the work is to implement a tool for analysis of XML data, DTDs and XSDs. In addition the tool is able to visualize input XML documents, DTD and XML schemes, show outputs as tables and plotted graphs. The application is capable of comparision between XML data and corresponding schemes results. Results can be saved and later merged with results of the new documents. The application can efficiently process large documents and was written with a respect to extensibility and maximal possible variability.

Keywords: XML, DTD, XML Schema, Analysis

Kapitola 1

Úvod

Počítačové programy dnešní doby často potřebují přijímat a produkovat data. Internetové prohlížeče zobrazují webové stránky, textové procesory ukládají a otevírají dokumenty, databázové systémy při výměně dat potřebují načítat data z externích zdrojů nebo data naopak exportovat. Sdílení těchto dat mezi různými aplikacemi vedlo k potřebě standardizovaného formátu. Takovým formátem je e**X**tensible **M**arkup **L**anguage neboli "rozšiřitelný značkovací jazyk", zkráceně XML. Jedná se o jednoduchý jazyk, který pomocí značek přidává k obsahu textu další informace. Množina značek přitom není jazykem XML určena, každý formát založený na XML si proto může zavést vlastní sadu značek, kterou bude využívat.

Na jazyku XML je založená celá řada formátů, které se využívají k různým účelům. S rostoucí oblibou jazyka XML vzniká potřeba zkoumat strukturu dat, která jsou v něm napsána. Zjištění struktury dat, která se běžně vyskytují v reálném světě, může pomoci při jejich efektivním prohledávání nebo volbě reprezentace během ukládání těchto dat.

Těžištěm práce je vytvoření aplikace XML Analyser, zaměřené na strukturální analýzu XML dat a jejich schémat.

Aplikace dokáže zjišťovat charakteristiky jednotlivých dokumentů (např. počty různých elementů v různých kontextech, jejich jména, hloubky dokumentů, ...) a z těchto charakteristik následně počítat souhrnné charakteristiky pro celé kolekce XML dokumentů (např. průměr z počtu různých elementů v celé kolekci, medián hloubek dokumentů v kolekci, apod.).

Ze srovnání souhrnných charakteristik XML dokumentů a jejich schémat lze poté vypozorovat, které vlastnosti schémat jsou ve velké míře v XML dokumentech využívány a které nikoli. Výsledky analýz je možné zobrazit v podobě grafů či tabulek.

Text je rozdělen do několika hlavních kapitol. Druhá kapitola obsahuje popis XML technologií použitých v práci, konkrétně formát XML, jazyky DTD a XSD pro popis schématu XML dokumentu a jazyky XPath a XQuery určené k dotazování nad XML dokumenty. Třetí kapitola obsahuje popis článku, zabývajícího se (mimo jiné) strukturální analýzou XML dat a jejich schémat a programu pro dávkovou analýzu dokumentů. Na konci třetí kapitoly je popsán postup, který je použit v aplikaci XML Analyser pro analýzu dokumentů a schémat. Čtvrtá kapitola obsahuje uživatelskou dokumentaci, popis instalace a konfigurace programu a ukázkovou analýzu. V páté kapitole je popsána programátorská dokumentace. Šestá kapitola je shrnutím práce, jsou v ní zhodnoceny klady a nedostatky programu a možnosti jeho dalšího rozšíření.

Kapitola 2

Popis použitých technologií

Tato kapitola obsahuje popis XML technologií použitých v práci, konkrétně popisuje formát XML, jazyky DTD a XML Schema pro popis schématu XML dokumentů a jazyky XPath a XQuery určené k dotazování nad XML dokumenty.

2.1 Formát XML a postupy jeho zpracování

Následující podkapitoly slouží jako stručný úvod k formátu XML a metodám jeho zpracování.

2.1.1 Formát XML

XML [5] dokument se skládá ze značek – elementů a vlastního textového obsahu. Elementy jsou uzavřeny ve špičatých závorkách (< a >). Příklad XML dokumentu ukazuje obr. 2.1. V něm se vyskytují elementy se jmény `report`, `analysis`, `levels` a `item`.

XML dokument musí obsahovat jeden nebo více elementů. Právě jeden element (tzv. kořenový element) "obklopuje" celý dokument, tzn. není součástí jiného elementu. Jméno ve značce koncového elementu musí být stejné jako v počáteční značce, jména rozlišují velikost písmen. Elementy musí být správně uzávorkované (tzv. *well-formed*) – pokud element začíná uvnitř jiného elementu, jeho koncová značka musí být opět obsažena uvnitř toho samého elementu.

```
<?xml version="1.0" encoding="utf-8"?>
<report>
  <analysis>
    <levels>
      <item value="1" type="average" depth="0" />
      <item value="2" type="average" depth="1" />
      <item value="5.01" type="average" depth="2" />
      <item value="30.42" type="average" depth="3" />
    </levels>
  </analysis>
</report>
```

Obrázek 2.1: Ukázkový XML dokument

Obsah elementu je text mezi počáteční a koncovou značkou elementu. Element bez obsahu je možné zapsat ve zkrácené formě, např. element `item` lze zapsat takto: `<item />`. Lomítko před `>` nahrazuje koncovou značku. Jména elementů mohou obsahovat písmena, číslice, pomlčky, podtržítka, tečky nebo dvojtečky. Dvojtečka má zvláštní význam, odděluje tzv. *prostor jmen* (*namespace*) od jména elementu. Jména elementů začínajících `xml` (bez ohledu na velikost písmen) jsou vyhrazena pro účely standardu XML.

Element může obsahovat libovolný počet *atributů*. Pro jména atributů platí stejná pravidla jako pro jména elementů. Jméno atributu je od své hodnoty odděleno rovnítkem (`=`). Hodnota atributu musí být uvedena uvnitř apostrofů (např. `'...'`) nebo uvozovek (`"..."`). Pokud je v hodnotě atributu použit apostrof, je potřeba hodnotu obklopit uvozovkami (a naopak). Ukázkový dokument obsahuje atributy se jmény `value`, `type` a `depth`.

Znaky `<` a `&` nemohou být použity v textu dokumentu přímo. Pokud se v textu vyskytnou, je potřeba je nahradit pomocí sekvencí `&lt;` a `&amp;`. Podobně znaky `>`, `&quot;` , and `&apos;` musí být nahrazeny sekvencemi `&gt;` , `&quot;` a `&apos;`.

Komentáře se mohou vyskytnout v dokumentu kdekoliv mezi značkami. Začínají posloupností znaků `<!--` a končí posloupností `-->`. Nesmí se v nich vyskytovat posloupnost "`--`".

Instrukce pro zpracování (processing instructions) umožňují vložit do dokumentu data, která slouží pro zpracování dalšími aplikacemi. Začínají sekvencí `<?` and končí sekvencí `?>`. Mezi instrukce pro zpracování patří i XML deklarace, která určuje verzi XML jazyku použitého v dokumentu a kódování dokumentu (případně ještě další informace).

Sekce *CDATA* slouží pro vložení bloku textu s obsahem, který by se jinak interpretoval jako část značek. Sekce *CDATA* začínají sekvencí znaků "`<![CDATA[`" a končí sekvencí "`]]>`". Uvnitř sekce se nesmí vyskytovat řetězec "`]]>`".

Více informací lze nalézt ve specifikaci jazyka XML 1.0 [5] či v řadě tutoriálů dostupných na webu. Části předchozího textu vznikly překladem [16].

2.1.2 XML DOM

XML DOM [1] (Document Object Model) je standard pro přístup k XML dokumentům a manipulaci s nimi. XML DOM nahlíží na XML dokument jako na *strom* (ve smyslu teorie grafů) [14]. V uzlech tohoto stromu jsou uchovávány informace o prvcích XML dokumentu (elementech, atributech, komentářích,...). Pro pohyb mezi těmito uzly a jejich manipulaci slouží sada funkcí (přechod na předka, potomka nebo obsah uzlu; přidání uzlu nebo jeho odebrání,...).

Nevýhodou tohoto přístupu je nutnost reprezentace celého dokumentu v paměti počítače. Typická implementace XML DOM projde celý dokument a vytvoří z něj struktury (nebo instance tříd), které uloží do operační paměti. Tento model je poté procházen a případně měněn. Vytvoření těchto struktur může u velkého dokumentu trvat dlouho a může mít velkou spotřebu paměti.

Mezi výhody lze uvést jednoduché procházení a změnu modelu, jakmile je dokument převeden na XML DOM strom.

2.1.3 Proudové zpracování dokumentu

SAX (Simple API for XML) [33] je rozhraní parseru pro proudové čtení XML dokumentu. Dokument je při zpracování pomocí SAXu procházen postupně od začátku do konce. Během tohoto čtení parser reaguje na *události* (např. nalezení počáteční nebo koncové značky elementu) a podle typu události volá metody, které do parseru přidal uživatel.

Výhodou při čtení dokumentu pomocí SAXu oproti DOMu je rychlost čtení a minimální zátěž paměti. Nevýhodou je obtížnější manipulace s dokumentem, protože při čtení se nelze vracet.

Podobný přístup je využit i u tzv. *readerů* [29]. Rozdíl oproti SAXu spočívá v tom, že čtení pomocí readeru je řízeno přímo uživatelem.

2.2 Schémata XML dokumentů

Následující podkapitoly obsahují stručný úvod k jazykům DTD a XML Schema, které se používají k vymezení struktury XML dokumentů.

2.2.1 DTD

Definice typu dokumentu (DTD) je jazyk pro popis schématu XML dokumentu. Schéma říká, které elementy (a atributy, entity a notace) se mohou v dokumentu vyskytnout, jaké jsou mezi nimi vztahy a jaké jsou jejich typy.

V XML dokumentu může být DTD obsaženo přímo (tzv. *inline deklarace*) nebo může být připojeno pomocí *externí reference*. Deklarace musí předcházet prvnímu elementu XML dokumentu.

Formát inline deklarace ukazuje obr. 2.2, formát externí deklarace obr. 2.3. U obou typů deklarací je potřeba určit kořenový element dokumentu.

```
<!DOCTYPE kořenový-element [obsah DTD]>
```

Obrázek 2.2: Formát inline deklarace DTD

Obsah DTD tvoří seznam deklarací (mezi nimi je i popis typu kořenového elementu).Na obr. 2.4 je ukázka DTD, které popisuje strukturu ukázkového XML dokumentu 2.1.

```
<!DOCTYPE kořenový-element SYSTEM "soubor-s-dtd">
```

Obrázek 2.3: Formát externí deklarace DTD

```
<!ELEMENT report (analysis)>
<!ELEMENT analysis (levels)>
<!ELEMENT levels (item*)>
<!ELEMENT item EMPTY>

<!ATTLIST item value CDATA #IMPLIED>
<!ATTLIST item type CDATA #IMPLIED>
<!ATTLIST item depth CDATA #IMPLIED>
```

Obrázek 2.4: Obsah DTD schématu pro ukázkový XML dokument

Podrobněji o DTD např. v článku [12] nebo ve specifikaci XML [5].

2.2.2 XML Schema

Jazyk XML Schema [7] vznikl jako náhrada DTD. Oproti DTD je možné podrobněji popsat schéma dokumentu (vestavěné typy), používat konstrukce známé z objektově orientovaného programování (dědičnost) a využívat *prostory jmen* (namespaces). Syntaxe XML Schema je na rozdíl od DTD založena na XML.

Oproti DTD se deklarace XML Schema připojuje přímo v elementu XML dokumentu pomocí atributů `noNamespaceSchemaLocation` nebo `schemaLocation`. Příklad připojení schématu XML Schema k dokumentu je uveden na obr. 2.5.

```
<?xml version="1.0" encoding="utf-8"?>
<root-element
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="soubor-s-xml-Schema">
</root-element>
```

Obrázek 2.5: Ukázka připojení XML Schema k dokumentu

Také schémata XML Schema mohou být definována přímo v XML dokumentu (inline), vždy ale alespoň jeden (kořenový) element není možné pomocí tohoto schématu ověřit (element nepatří do schématu).

V XML dokumentu lze použít i více než jen jedno schéma XML Schema, schémata je možné různě skládat a využívat již hotová schémata. Pro rozlišení mezi schématy jsou potom použity *prostory jmen*, více o nich např. v [19].

Výpis na obr. 2.6 ukazuje XML Schema, které stejně jako DTD 2.4 udává schéma ukázkového XML dokumentu 2.1. Při srovnání je vidět, že ačkoliv XML Schema umožňuje podrobnější popis schématu, jeho drobnou nevýhodou proti DTD je jistá "rozvláčnost" popisu.

2.3 Dotazovací jazyky nad XML daty

Následující podkapitoly obsahují popis jazyků XPath a XQuery, které slouží k dotazování nad XML daty.

2.3.1 Jazyk XPath

XPath [6] je dotazovací jazyk určený pro vybírání uzlů z XML dokumentů. Jeho syntaxe je podobná popisu cest v souborovém systému.

Například XPath výraz `//levels/item/@value` vyhodnocený na dokumentu 2.1 vybere hodnoty atributů `value`, které se nacházejí v elementech `item` a tyto elementy jsou přímými potomky elementů `levels`.

2.3.2 Jazyk XQuery

XQuery [8] je jazyk určený k vybírání a manipulaci uzlů XML dokumentu. Jazyky XQuery 1.0 a XPath 2.0 podporují stejné funkce a operátory. Svou syntaxí připomíná XQuery jazyk SQL [17] nad XML daty. Jazyk XPath 2.0 tvoří podmnožinu jazyku XQuery 1.0, XPath je v XQuery použit pro vybírání (a filtrování) uzlů XML dokumentu. Příklad dotazu v jazyku XQuery je uveden na obr. 2.7.

Pokud by ukázkový dokument 2.1 byl uložen v souboru `example.xml`, pomocí XQuery dotazu 2.7 bychom z dokumentu získali součet hodnot `value` z elementů `item`, v nichž je hodnota `depth` vyšší než `0`. Výstup je zachycen ve výpisu 2.8

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">

  <xs:element name="report" type="analysis"/>

  <xs:complexType name="analysis">
    <xs:sequence>
      <xs:element ref="analysis"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="analysis" type="levels"/>

  <xs:complexType name="levels">
    <xs:sequence>
      <xs:element ref="levels"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="levels">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="item"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="item">
    <xs:complexType>
      <xs:attribute name="value"/>
      <xs:attribute name="type"/>
      <xs:attribute name="depth"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Obrázek 2.6: XML Schema pro ukázkový XML dokument

```
<root>
{
let $depths :=
(
  for $x in doc("example.xml")/report/analysis/levels/item
  where $x/@depth > 0
  return $x/@value
)
return <depth sum="{fn:sum($depths)}" />
}
</root>
```

Obrázek 2.7: Ukázka XQuery dotazu

```
<root>
  <depth sum="37.43" />
</root>
```

Obrázek 2.8: Výstup ukázkového XQuery dotazu

Kapitola 3

Srovnání s podobnými pracemi

Aplikací s podobným zaměřením není mnoho. Kvůli získání dat do článků zabývajících se problematikou XML dat většinou vzniknou jednorázové aplikace, které ale nebývají uvolněny spolu s článkem.

3.1 Statistical Analysis of Real XML Data Collections

Statistical Analysis of Real XML Data Collections [15] je velmi podrobná technická zpráva zkoumající různé vlastnosti používaných XML dat. Jejím záměrem je analýza XML dat, jejich struktury a jejich celkové složitosti. Velikost analyzovaných kolekcí XML dat je větší než 20GB. Zpráva bere v úvahu existující články mající podobné zaměření, snaží se potvrdit nebo vyvrátit jejich závěry a přikládá vlastní zjištění. Zaměřuje se i na charakteristiky dokumentů, které se vyskytují v XML dokumentech relativně často (jako je např. smíšený obsah elementů či rekurze), ale ostatní zprávy se o nich zmiňují jen zřídka. Hodnoty těchto charakteristik jsou následně zveřejněny a porovnány jak pro instance XML dokumentů, tak pro jejich schémata.

Součástí zprávy jsou i definice pojmů, které nejsou běžnému čtenáři obvyklé - např. *rekurze elementů*, *fan-in* a *fan-out elementu* nebo *relační vzor*. Kvůli velkému počtu těchto definic jsou na tomto místě uvedeny jen některé z nich. Konstrukty, které mohou být zjišťovány pomocí aplikace XML Analyser jsou popsány v příloze B.1 a B.2, ostatní je možné dohledat přímo ve zprávě.

Kolekce dokumentů jsou rozděleny do šesti kategorií:

- Datatově zaměřené dokumenty (**dat**), např. exporty databází
- Dokumentová data (**doc**) – XHTML stránky, dokumenty ve formátu DocBook,...
- Dokumenty pro datovou výměnu (**ex**), např. lékařské informace o pacientech
- Přehledy obsahů dat (**rep**) – např. databází
- Dokumenty popisující speciální struktury (**res**) – např. DNA/RNA
- Dokumenty sémantického webu [3] (**sem**)

Už jen z rozdělení XML dokumentů do různých skupin a srovnání jejich velikostí je vidět rozdílná povaha dat. Zatímco průměrná velikost dokumentů z kategorie **doc** je malá (182 kB), dokumenty z kategorií **rep** a **sem** jsou poměrně velké (3903 kB **rep** a 5116 kB **sem**). Tuto skutečnost lze přičíst způsobu vzniku těchto dokumentů – zatímco dokumenty ze skupiny **doc** bývají často vytvářeny ručně nebo s pomocí editoru, dokumenty ze skupiny **rep** jsou výsledkem automatického exportu rozsáhlých databází. U skupiny **sem** situace není jednoznačná, tato kategorie obsahuje jak velké dokumenty (největší 1,97 GB), tak mnoho malých dokumentů – velikost je silně ovlivněna použitými konvencemi a nástroji. Drtivá většina dokumentů napříč kategoriemi odkazuje na své schéma. Výjimku tvoří dokumenty **sem**, které žádné schéma nemají.

Následuje strukturální analýza dokumentů a schémat. Na základě zkušeností s ruční úpravou dat bylo zavedeno několik nových pojmů popisujících strukturu dokumentu (*relační* a *DNA vzor*, dva typy *smíšeného obsahu*, čtyři typy *rekurze* a dva typy *fan-outu* elementu). Zpráva se zaměřuje hlavně na jednoduché charakteristiky, které mohou být reprezentovány relačními tabulkami, v případě rekurzivních statistik nebo statistik smíšeného obsahu na charakteristiky, které mohou být jednoduše zpracovány a uloženy bez dalšího zobecňování.

Strukturální statistiky jsou rozděleny do deseti kategorií – obecné statistiky, statistiky hloubky dokumentů, statistiky úrovně dokumentů, statistiky popisující fan-out a fan-in dokumentů, rekurzivní statistiky, statistiky smíšeného obsahu, DNA statistiky, relační statistiky a statistiky schémat.

Pro zkoumané vlastnosti bylo spočítáno velké množství statistických parametrů – počet výskytů či poměrné zastoupení, velikost, délka zkoumaných konstruktů, jejich minimum a maximum, průměr, modus, medián, kvantily apod. Ve zprávě jsou z výsledků uvedeny jen ty nejzajímavější.

Obecné statistiky jsou zaměřeny na obecné vlastnosti XML dat, jako je počet elementů různých typů (prázdných elementů, textových uzlů, elementů se smíšeným obsahem, rekurzivních elementů atd.), počty atributů, délku cest, hloubku dokumentů a podíl textu v dokumentech. V případě DTD/XSD jsou hloubky počítány pro každý globální element použitý v odpovídajícím XML dokumentu.

Z výsledků vyplývá, že většina XML dokumentů má jednoduchou strukturu a počet různých elementů a atributů použitých v instancích dokumentů nebývá vyšší než 150. Tomuto počtu odpovídá i průměrný počet různých cest a průměrná hloubka dokumentů.(13). Tyto závěry se nevztahují na schémata dokumentů, u nich jsou charakteristiky často mnohem vyšší.

Statistiky pro hloubky dokumentů obsahují podrobnější charakteristiky vztahující se k hloubce dokumentů. Výsledky pro schémata neodpovídají výsledkům pro XML dokumenty, protože schémata jsou příliš ovlivněna rekurzí.

Statistiky pro úrovně dokumentů jsou zaměřeny na distribuci elementů, atributů, textového a smíšeného obsahu na jednotlivých úrovních XML dokumentů. Z výsledků vyplývá, že nejvíce elementů se nachází na první úrovni dokumentu (jedná se o přímé potomky kořenového elementu) a počty elementů vzhledem k úrovním se velmi rychle snižují. Navzdory tomu zastoupení textových uzlů je rovnoměrné vzhledem ke všem úrovním dokumentu.

Fan-out statistiky popisují rozložení XML uzlů v dokumentu. Fan-out u elementu označuje počet elementů, které jsou jeho přímými potomky. Podobně jako u statistik pro úrovně XML dokumentů mají nejvyšší hodnoty fan-outu elementy na prvních úrovních, s rostoucí hloubkou hodnoty prudce klesají.

Fan-in statistiky elementů jsou počítány pouze pro schémata. Fan-in

elementu označuje počet uzlů, které mohou být rodiči daného elementu. Nejvyšší hodnoty se vyskytly v kategorii **doc** a **ex**, což je předpokládaný výsledek vzhledem ke složitosti schémat v těchto kategoriích. Vysoké hodnoty u těchto kategorií jsou způsobeny přítomností úplných podgrafů ve schématech (u XHTML např. u elementů `p`, `b`, `u` a `i`). Tyto podgrafy mohou zkomplikovat zpracování dokumentu navzdory jejich jednoduchému informačnímu obsahu.

Rekurzivní statistiky. XML element je *rekurzivní*, pokud obsahuje element se stejným jménem ve svém podstromu. Pro dokumenty jsou zjišťovány počty rekurzivních elementů, šířka větvení rekurzivních elementů a vzdálenosti nejbližších a nejvzdálenějších rekurzivních elementů se stejným jménem. Typy rekurze jsou podrobněji rozděleny do několika kategorií.

Rekurze se objevuje hlavně v kategoriích **doc** a **ex**, v ostatních kategoriích jen výjimečně. Z typů rekurze se objevuje nejvíce tzv. *lineární rekurze*, kdy rekurzivní element může obsahovat rekurzivní elementy pouze se stejným jménem (jako je jméno onoho elementu), navíc na stejné úrovni se může element vyskytnout pouze jednou. Toto pozorování je v rozporu s ostatními zprávami, které lineární rekurzi nepřikládají valnou důležitost.

Při porovnání dokumentů a schémat bylo zjištěno, že dokumenty (či celé kolekce) většinou nevyužívají všech možností, které jsou umožněny schématy. Proto je vždy nutné při analýze schémat brát v úvahu i data, která tato schémata využívají, samotná schémata nejsou spolehlivým zdrojem informací.

Statistiky smíšeného obsahu dále analyzují strukturu a složitost elementů se smíšeným obsahem. Zaměřují se na průměrnou a maximální hloubku smíšeného obsahu a procentuální zastoupení těchto elementů. Zpráva dochází ke zjištění, že struktura elementů se smíšeným obsahem není příliš složitá.

DNA statistiky zkoumají nově pojmenovaný konstrukt – tzv. *DNA vzor*. *DNA vzor* se skládá z libovolného počtu jednoduchých elementů (ty jsou prázdné nebo obsahují jen text) a právě jednoho složitého elementu. Konstrukt je pojmenován podle XML dokumentu, ve kterém byl poprvé zaznamenán – dokumentu popisujícím strukturu DNA.

Analýza se zaměřuje na výskyt těchto vzorů, jejich hloubku a šířku. Vzor se objevuje poměrně často, nejvíce u kategorií **doc**, **res** a **ex**. Průměrná

šířka je poměrně nízká – menší než 10, stejně tak průměrná hloubka nepřesahuje 3 elementy.

Relační statistiky zkoumají další z konstruktů nově zavedených v této zprávě – tzv.*relační vzor*. Element je označen za *relační vzor*, pokud není rekurzivní, ani nemá smíšený obsah a jeho obsahem jsou jednoduché elementy (prázdné nebo obsahují jen text). Na počtu atributů přitom nezáleží. Tento vzor se objevuje často v XML dokumentech, které vznikly exportem databází. Mohou být uloženy a zpracovány jako jednoduché tabulky. Ve statistikách jsou uvedeny dva typy relačních vzorů. Je zkoumán počet jejich výskytů v dokumentech, šířka a fan-out elementů (odpovídá počtu sloupců tabulky).

Vzor se vyskytuje v datech poměrně často, nejvíce v kategoriích **rep**, **sem** a **dat**.

Statistiky schémat se zaměřují na konstrukty využívané v schématech. Autoři zmiňují nízký podíl schémat XSD proti schématům DTD. Z konstruktů je nejvíce použita *libovolná posloupnost*, v jazyku XML Schema vyjádřená elementem `all`, v DTD vyjádřená pomocí neomezeného opakování výběru elementů. Další hojně využívanou vlastností schémat jsou *implicitní hodnoty*. Posledním rysem, který se ve schématech objevuje poměrně často, jsou konstrukty `ID` a `IDREF(S)`.

V závěru je zmíněn fakt, že v reálných XML datech lze nalézt častý výskyt jednoduchých vzorů, které se opakují. S ohledem na tuto skutečnost by mohly programy, které zpracovávají XML data, dále zefektivnit svoji činnost.

3.2 Analyzer

Analyzer [22] je aplikace pro dávkovou analýzu napsaná v jazyce Java [27] s využitím platformy NetBeans [31] . Byla vyvinuta pro analýzu XML dat, ale není omezena jen na analýzu XML dokumentů, může zpracovávat i jiná data. Aplikace vznikla jako softwarový projekt na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze.

Mezi výhody této aplikace patří rozšiřitelnost aplikace pomocí pluginů. Pluginy slouží k získávání a ukládání dat (např. z/do databáze), zajišťují stahování dat z internetu (tzv. *crawler*) i samotnou analýzu dat. Pomocí plu-

ginů je možné zpracovávat i různé typy souborů (nejen XML, ale i HTML [2]), data různě filtrovat (podle názvu, stáří dokumentu,...) a získávat statistické informace.

Tento projekt je stále ve vývoji a lze očekávat další rozšíření jeho funkcionality.

3.3 Zvolený přístup k analýze XML dokumentů a schémat

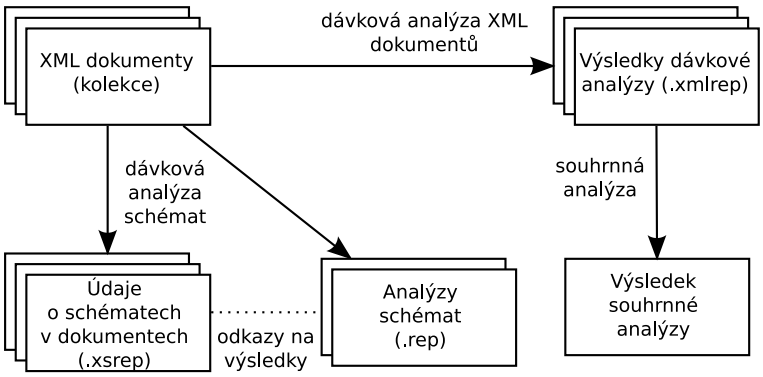
Jedním z požadavků kladených na aplikaci byla schopnost zpracovávat i velké XML dokumenty. Při přístupu k XML dokumentu pomocí DOM stromu (viz 2.1.2) je obtížné tento požadavek dodržet kvůli velké spotřebě paměti, kterou struktury DOM stromu vyžadují. Bylo by nutné nejprve zjistit strukturu dokumentu a vybrat z dokumentu podstrom, který se už vejde do paměti a z tohoto fragmentu následně vytvářet DOM strom. Navíc by bylo třeba ošetřit "napojování" těchto podstromů, aby hledané konstrukty ležící v různých podstromech byly nalezeny.

Kvůli těmto nedostatkům bylo zvoleno proudové procházení dokumentu (viz 2.1.3). V paměti je potřeba držet jen nepatrné množství informací oproti reprezentaci dokumentu pomocí DOM stromu a procházení je poměrně rychlé. Nevýhodou je obtížnější zjišťování charakteristik. Při čtení uzlu se nelze vracet a není možné "přeskakovat" na ostatní uzly.

Dalším požadavkem byla snadná rozšiřitelnost a velká parametrizovatelnost sad statistik, které jsou nad XML daty počítány. Kvůli tomuto požadavku není v aplikaci zabudovaná pevná sada analýz, je zde přítomná knihovna se "stavebními bloky" – tzv. *komponentami*, ze kterých se tyto analýzy sestavují (tzv. *řetízky*). Při potřebě rozšíření knihovny o další komponenty potom stačí změnit tuto knihovnu, nikoli samotnou aplikaci.

Statistiky nad kolekcemi dat jsou počítány ve dvou krocích. V prvním jsou spočítány charakteristiky XML dokumentů, pro každý vstupní dokument je vytvořen soubor s jeho výsledky (tzv. *dávková analýza*). Následně jsou počítány statistiky nad těmito výsledky (tzv. *souhrnná analýza*). Výhodou tohoto přístupu je, že při doplnění kolekce o nové dokumenty lze využít staré výsledky, *dávková analýza* se spočítá jen pro nové dokumenty a *souhrnná analýza* se znovu spočítá ze všech výsledků. Tento postup je

zachycen na obr. 3.1.



Obrázek 3.1: Přehled průběhu analýzy

Kapitola 4

Uživatelská dokumentace

V následující kapitole je probrána instalace a konfigurace aplikace XML Analyser a popis jejího uživatelského prostředí. V závěru kapitoly jsou předvedeny možnosti aplikace na ukázkové analýze.

4.1 Instalace a konfigurace

4.1.1 Instalace

Pro běh aplikace vyžaduje běhové prostředí .NET 3.5 [30]. Pokud není na cílovém počítači nainstalováno, lze jej nainstalovat z přiloženého CD. Instalace se spustí souborem `dotnetfx35.exe`.

V aplikaci je využíván externí balík graphviz [25], který je nutné nainstalovat pro vykreslování grafů. Instalační balík se nachází na CD v souboru `graphviz-2.22.2.msi`. Aplikace XML Analyser předpokládá umístění balíku do adresáře `C:\Program Files\Graphviz2.22`, při jiném umístění je potřeba tuto cestu změnit v souboru `DotPath.conf`, který se nachází v hlavním adresáři aplikace.

Vlastní aplikace se nainstaluje rozbalením samorozbalovacího archivu `XmlAnalyser-bin.exe` a spouští se souborem `XmlAnalyser.exe`.

4.1.2 Konfigurace

Aplikace využívá k editaci a zobrazování souborů (obrázků, vstupních XML dokumentů, ...) externí programy. Nastavení jednotlivých externích aplikací a jejich asociace se soubory se nachází v konfiguračním souboru `Extensions.conf` v hlavním adresáři aplikace. Konfigurační soubor použitý v aplikaci je zobrazen na obr. 4.1. Řádek, který obsahuje právě dva středníky, je považován za platný, ostatní jsou ignorovány. První sloupec oddělený středníkem označuje *klíč*. Druhý sloupec představuje cestu k externímu programu a třetí sloupec argument, se kterým se tento program bude volat. Například ve výpisu je obsažen řádek s klíčem `xml`, při otvírání XML dokumentu budou v aplikaci použity údaje z tohoto řádku. Pro otevírání XML souborů se tedy bude používat editor `notepad.exe`, který jako argument při spuštění dostane jméno otevíraného souboru. Obdobně pro otevírání obrázků ve formátu `png` bude použit prohlížeč, který je součástí systému Windows. Při vykreslování struktury dokumentu je možné ukládat výstup i do jiných formátů, než jsou uvedeny v ukázce (např. `svg`, `svgz`, `ps`), ale ve výchozí instalaci systému Windows nejsou nástroje, kterými je možné tyto formáty prohlížet, proto nejsou v konfiguračním souboru uvedeny. Externí aplikace je tedy potřeba touto cestou doplnit do konfiguračního souboru, je přitom nutné dodržet daný formát konf. souboru.

```
;;; format: "key ; application ; application-argument"

;;; text editor for viewing xml files
xml ; notepad.exe ; ${filename}

;;; text editor for viewing xquery files
xq ; notepad.exe ; ${filename}

;;; image viewers
png ; rundll32.exe ; C:\WINDOWS\System32\shimgvw.dll,ImageView_Fullscreen ${filename}
jpg ; rundll32.exe ; C:\WINDOWS\System32\shimgvw.dll,ImageView_Fullscreen ${filename}
```

Obrázek 4.1: Konfigurační soubor Extensions.conf

Pro účely logování je v aplikaci použita knihovna Apache log4net [28]. Výhodou této knihovny jsou široké možnosti nastavení pomocí konfiguračního souboru bez nutnosti překladu hotové aplikace. Je možné například měnit cíle logování – aplikace může psát do souboru, posílat logovací výpisy v e-mailu apod. Také je možné měnit úrovně výpisů, které budou

zaznamenány, např. je možné vypnout informační výpisy a nechat zapisovat pouze údaje o chybách.

Nastavení těchto vlastností je obsaženo v souboru `XmlAnalyzer.exe.config` v hlavním adresáři aplikace, ukázka tohoto souboru je na obr. 4.2.
`XmlAnalyzer.exe.config` je XML dokument, element `log4net` vymezuje nastavení logovací knihovny. Element `root` označuje tzv. *kořenový logger*, kterému jsou posílány logovací výpisy. Jeho element `level` označuje úroveň výpisů, které budou předávány tzv. *appenderům*. *Appendery* se starají o vlastní zápis logovacích zpráv - do souborů, na konzoli nebo na jiné umístění. V ukázce je apppender typu `FireEventAppender`, tento typ vyvolá *událost*, kterou je možné v kódu programu odchytávat a dále zpracovávat. Element `layout` uvnitř elementu `appender` slouží k určení formátu logovaného výpisu. Podrobněji o možnostech nastavení v dokumentaci knihovny log4net [28].

```
<log4net>
  <root>
    <level value="ALL" />
    <appender-ref ref="FireEventAppender" />
  </root>

  <appender name="FireEventAppender" type="SharedAssembly.FireEventAppender" >
    <layout type="log4net.Layout.PatternLayout">
      <param name="ConversionPattern" value="%d %-5p %c - %m%n" />
    </layout>
  </appender>
</log4net>
```

Obrázek 4.2: Fragment `log4net` z konfiguračního souboru `XmlAnalyser.exe.config`

Během dávkové analýzy aplikace je při analýze schémat a validující analýze potřeba pracovat se schématy DTD a XSD. Často jsou schémata vystavena na internetu. Aby nebylo potřeba opakovaně stahovat stejné soubory z internetu, uchovává si aplikace tyto soubory v tzv. *kešovacím adresáři*. Ve výchozím nastavení má tento adresář název `CacheDir` a nachází se v adresáři aplikace. Cestu a název *kešovacího adresáře* je možné změnit v souboru `XmlAnalyzer.exe.config`, výchozí nastavení je zobrazeno na obr. 4.3.

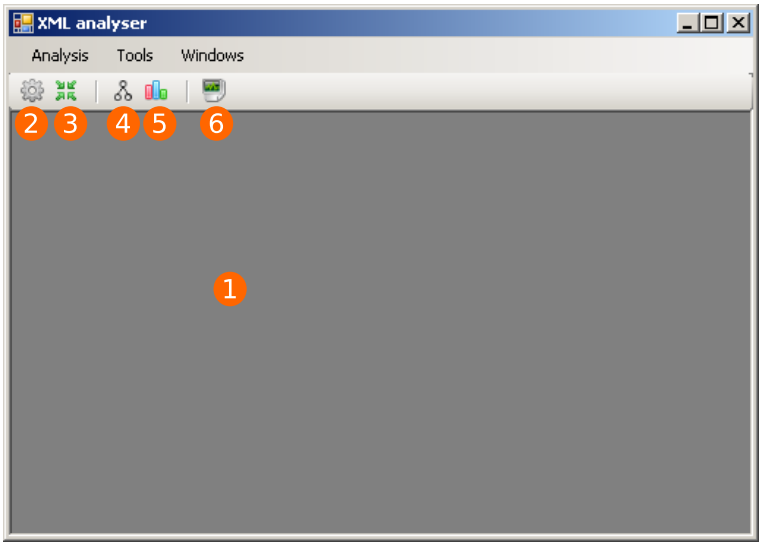
```
<userSettings>
  <XmlAnalyser.Properties.Settings>
    <setting name="CacheDir" serializeAs="String">
      <value>CacheDir</value>
    </setting>
  </XmlAnalyser.Properties.Settings>
</userSettings>
```

Obrázek 4.3: Fragment `userSettings` z konfiguračního souboru `XmlAnalyser.exe.config`

4.2 Popis uživatelského prostředí

V následujících podkapitolách jsou podrobně popsány části uživatelského prostředí.

4.2.1 Hlavní okno aplikace



Obrázek 4.4: Hlavní okno aplikace

Hlavní okno aplikace slouží jako tzv. *MDI okno* (z anglického názvu *multiple document interface*[36]). Zobrazeno je na obr. 4.4. Slouží jako rodičovské okno, do jehož oblasti (1) jsou umisťována další okna (grafy, tabulky, okno s logovacím výpisem). Jeho součástí je:

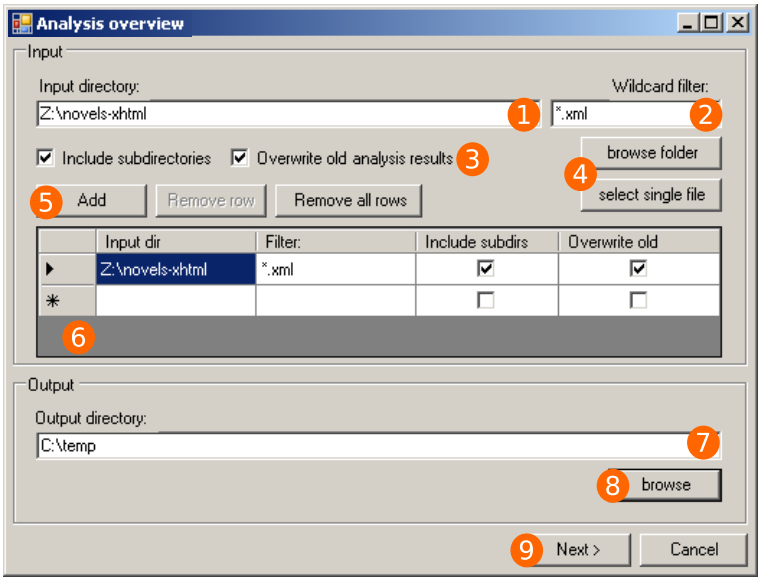
- nástroj pro dávkovou analýzu XML dokumentů (2)
- nástroj pro souhrnnou analýzu (3)
- nástroj pro grafové zobrazení vstupních dokumentů (dokumentů XML, DTD a XSD) (4)
- nástroj pro zobrazení výsledků ve formě grafů a tabulek (5).
- zobrazení logu (6).

4.2.2 Dávková analýza

Dialog dávkové analýzy se skládá ze dvou obrazovek. První obsahuje informace o vybíraných souborech, druhá o typu analýzy a použitých *"řetízcích"* – seznamech *komponent* určených k analýze.

Obrazovka pro výběr souborů (obr. 4.5) obsahuje:

1. Adresář se soubory k analýze
2. Filtr souborů (*wildcard* výraz[26])
3. Zaškrtávací pole *Include subdirs* pro rekurzivní procházení podadresářů vybraného adresáře a *Overwrite old analysis results* pro přepsání výsledků předchozí analýzy (pokud takové soubory s výsledky existují ve výstupním adresáři).
4. Tlačítko *browse* pro výběr celého adresáře a *select single file* pro výběr jediného souboru
5. Tlačítko *Add* pro přidání údajů o vstupním adresáři mezi seznam vstupních adresářů, tlačítko *Remove* pro odebrání označeného řádku z tabulkového přehledu a tlačítko *Remove all rows* pro odebrání všech řádků z tabulkového přehledu.
6. Tabulkový přehled vstupních adresářů a informací o nich.
7. Výstupní adresář
8. Tlačítko *browse* pro výběr výstupního adresáře
9. Tlačítko *Next* pro přechod na další obrazovku dialogu.



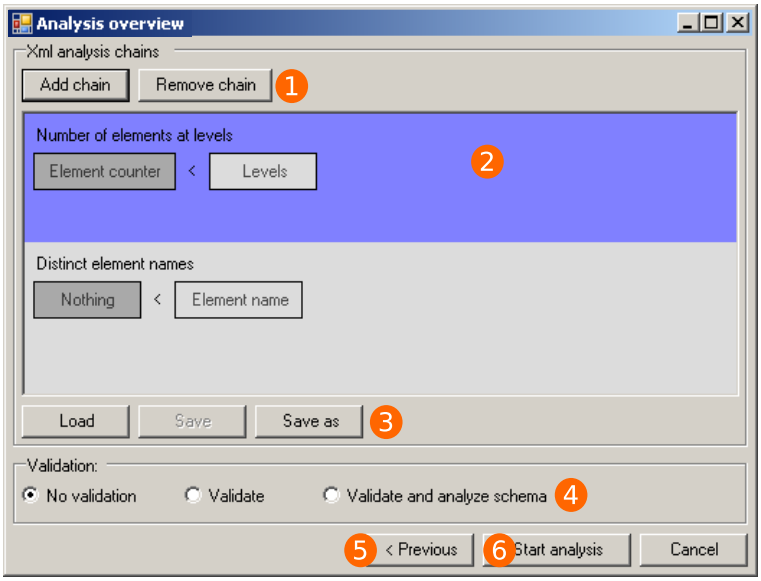
Obrázek 4.5: Dialog dávkové analýzy – výběr souborů

Tlačítko *Next* není aktivní do doby, než je vybrán výstupní adresář a alespoň jeden vstupní adresář.

V Obrazovce pro výběr "řetízků" (obr. 4.6) se nachazí:

1. Tlačítko *Add chain* pro přidání a *Remove chain* pro odebrání *"řetízku"* ze seznamu *"řetízků"* určených k analýze.
2. Seznam s *"řetízky"*. Vybraný *"řetízek"* je podbarven modrou barvou.
3. Tlačítka *Load*, *Save* a *Save as* pro uložení a načtení seznamu *"řetízků"*
4. Výběr typu analýzy, možnost přepnout mezi validující a nevalidující analýzou XML souborů a analýzou schémat.
5. Tlačítko *Previous* pro návrat do obrazovky s výběrem souborů
6. Tlačítko *Start analysis* pro spuštění analýzy.

U každého typu analýzy je ověřována správná struktura dokumentu, ale pouze při validující analýze nebo analýze schémat je u dokumentů



Obrázek 4.6: Dialog dávkové analýzy – výběr "řetízků"

ověřována validita. Nedodržení schématu je považováno za chybu, chyba zapsána do logu a soubor s výsledky analýzy pro daný soubor není vytvořen. Za chybu je považován i *nedeterminismus schématu*[24].

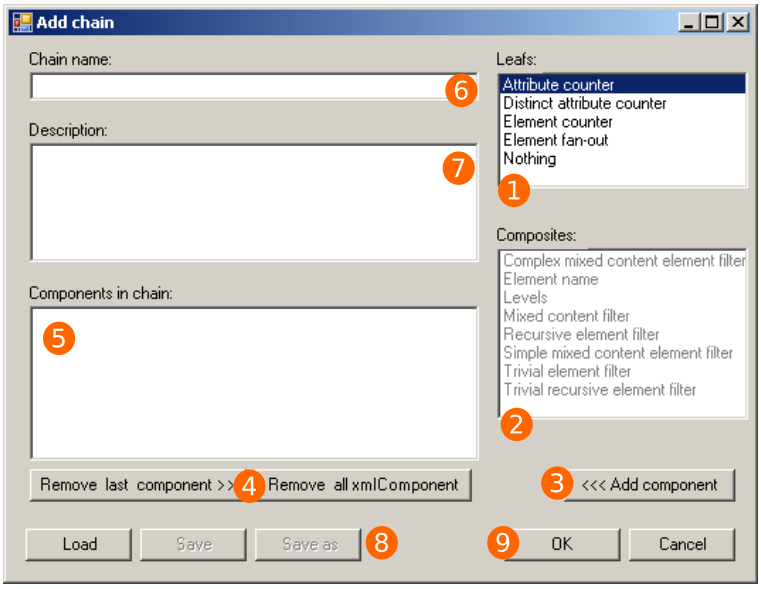
Pro konstrukci samotného "řetízku" slouží samostatný dialog (na obr. 4.7). V něm se nachází:

1. Seznam *listů* – ukončovačů "řetízku".
2. Seznam *kompozitů* – komponent, které mohou být tvořit "řetízek", ale nemohou jej ukončovat.
3. Tlačítko *Add component* pro přidání označené komponenty do vytvářeného řetízku.
4. Tlačítka *Remove last component* pro odebrání poslední přidané komponenty a *Remove all components* pro odebrání všech komponent.
5. Panel pro zobrazení vytvářeného *"řetízku"*.
6. Textové pole pro jméno *"řetízku"*.

7. Textové pole pro popis *"řetízku"*.

8. Tlačítka *Load*, *Save* a *Save as* pro načtení a uložení vytvářeného *"řetízku"*.

9. Tlačítka *OK* a *Cancel* pro opuštění dialogu.



Obrázek 4.7: Dialog pro konstrukci "řetízku"

Dokud neobsahuje *"řetízek"* alespoň jednu komponentu (list), je tlačítko *OK* neaktivní. Stejný princip funguje i u tlačítka *Save as*, tlačítko *Save* začne být aktivní po uložení "řetízku" pomocí tlačítka *Save as* a slouží k přepsání již uložené konfigurace.

4.2.3 Souhrnná analýza

V dialogu souhrnné analýzy se nachází tyto ovládací prvky:

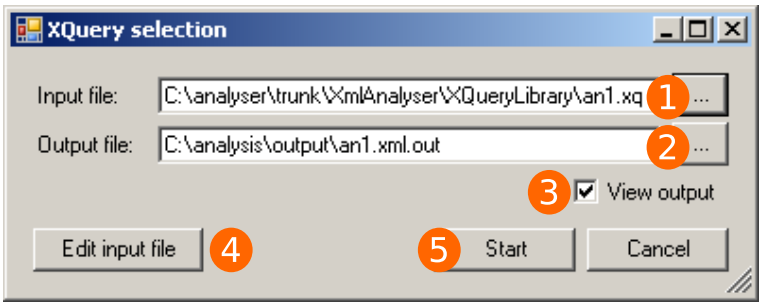
1. Textové pole s názvem vstupního souboru a tlačítko pro jeho výběr. Vstupní soubor obsahuje dotaz v jazyku XQuery.

2. Textové pole s názvem výstupního souboru a tlačítko pro jeho výběr.

3. Zaškrtávací pole pro zobrazení výsledného souboru. Editor, kterým bude výsledný soubor otevřen, je zvolen dle konfigurace, v konfiguračním souboru je editor vybrán podle klíče `xml` (viz sekce 4.1.2).

4. Tlačítko *Edit input file* pro úpravu vstupního souboru. K editaci je použit opět externí editor podle konfiguračního klíče `xq`.

5. Tlačítko *Start* pro spuštění analýzy a *Cancel* pro opuštění dialogu.



Obrázek 4.8: Dialog dávkové analýzy

4.2.4 Kreslení grafů

Při kreslení grafů je vybrán vstupní soubor (XML dokument nebo schéma) a výstupní soubor, do kterého bude uložen obrázek ve zvoleném formátu.

Hlavní dialog kreslení grafů (obr. 4.9) obsahuje následující prvky:

1. Textové pole se jménem vstupního souboru, tlačítko pro zvolení souboru.

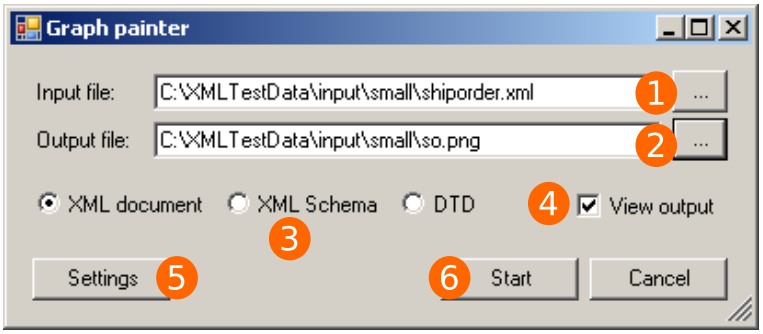
2. Výstupní soubor a tlačítko pro jeho zvolení.

3. Zvolení typu vstupního souboru

4. Zaškrtávací pole *View output* pro zobrazení výstupu

5. Tlačítko *Settings* pro volbu nastavení

6. Tlačítko *Start* pro spuštění vykreslování grafu.



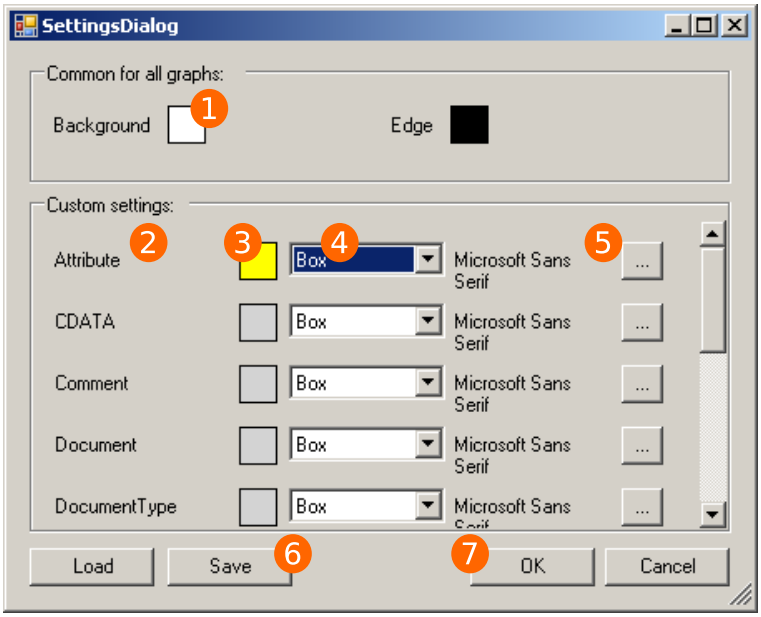
Obrázek 4.9: Dialog kreslení grafů

Tlačítko *Start* není aktivní do chvíle, kdy je vybrán soubor se vstupem a výstupem. U výstupního souboru navíc záleží na příponě, je potřeba zvolit jeden z formátů podporovaných aplikací (jejich přehled je k dispozici při výběru výstupního souboru pomocí tlačítka (2)). Pokud je navíc daná přípona asociována s nějakým prohlížečem obrázků v konfiguračním souboru (viz kapitola 4.1.2), zaškrtávací pole *View output* začne být aktivní a je možné nechat zobrazit výsledný soubor.

V dialogu pro nastavení vykreslování XML dokumentů (obr. 4.10) nalezneme:

1. Barvu pozadí a barvu šipek. Pro změnu barvy je potřeba kliknout na barevný čtverec (tlačítko)
2. Sloupec s typem uzlů
3. Barvy jednotlivých uzlů
4. Tvary uzlů
5. Písmo použité v textu uzlu
6. Tlačítka *Load* a *Save* pro načtení resp. uložení nastavení
7. Dvojice tlačítek *OK* a *Cancel* pro potvrzení nebo zrušení dialogu.

Nastavení pro schémata (obr. 4.11) navíc obsahuje následující dva sloupečky zaškrtávacích polí:



Obrázek 4.10: Nastavení vykreslování XML dokumentů

8. *Include in graph*. Pokud není zaškrnuto, uzly tohoto typu nebudou ve vykreslovaném grafu.

9. *Show type* pro zobrazení typu uzlu

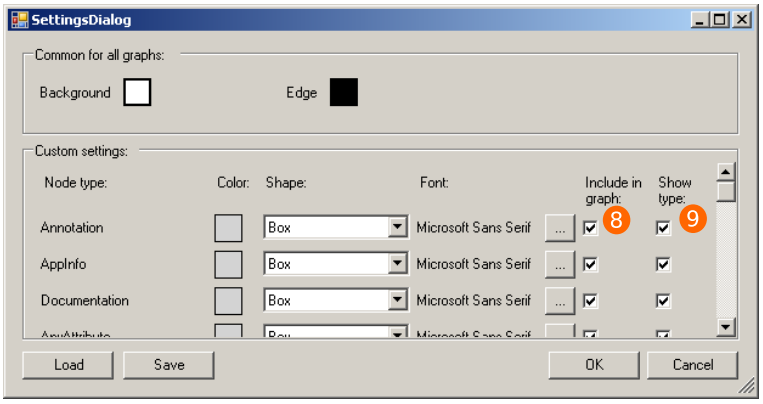
4.2.5 Zobrazení tabulek a diagramů

Dialog pro zobrazování tabulek a diagramů ukazuje obr. 4.12.

K získávání dat ze vstupních XML dokumentů slouží dotazy v jazyce XPath 1.0. Aplikace umožňuje zobrazení číselných dat s dvěma číselnými osami (typ dat *Numeric/Numeric*) nebo s jednou číselnou a jednou textovou osou (typ dat *Numeric/Textual*). Volba typu dat (2) ovlivní typ panelu (3), do kterého jsou zapisovány XPath výrazy (panel pro typ dat *Numeric/Textual* ukazuje obr. 4.13).

Popis dialogu hlavního dialogu (obr. 4.12):

1. Tlačítka *Show graph* pro zobrazení diagramu a *Show table* pro zobrazení tabulky.

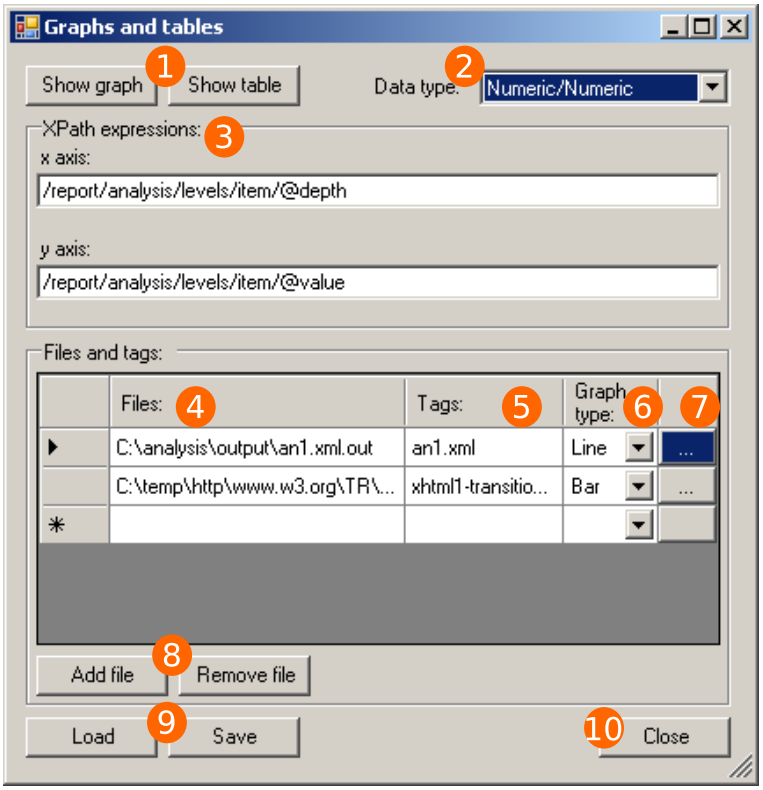


Obrázek 4.11: Nastavení vykreslování schémat

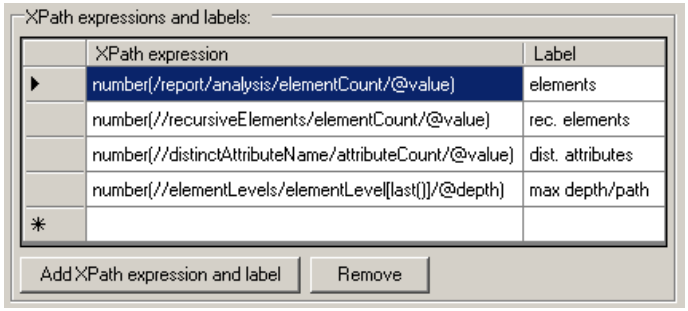
2. Přepínání typu dat.
3. Panel s dotazy. Mění se podle zvoleného typu dat.
4. Seznam souborů, nad kterými budou vyhodnocovány dotazy.
5. Zkrácené názvy pro soubory.
6. Typ grafu, který bude použit pro vykreslování dat zvoleného souboru.
7. Tlačítko sloužící k nastavení grafu (barvy, atd.). Dialog s nastavením se mění podle typu grafu (6).
8. Tlačítka *Add file* a *Remove file* pro přidání a odebrání souboru ze seznamu dotazovaných souborů (4).
9. Tlačítka *Load* a *Save* pro uložení/načtení nastavení.
10. Tlačítko *Close* pro opuštění dialogu.

Ke změně typu grafu pro daný soubor slouží sloupec tlačítek (6). Při stisknutí tlačítka ze sloupce (7) je zobrazen dialog pro nastavení zvoleného grafu. Typ dialogu závisí na typu grafu, pro typ *Numeric/Numeric* je dialog zobrazen na obr. 4.14. V něm lze nastavit:

1. Tvar, velikost a viditelnost značek použitých v grafu



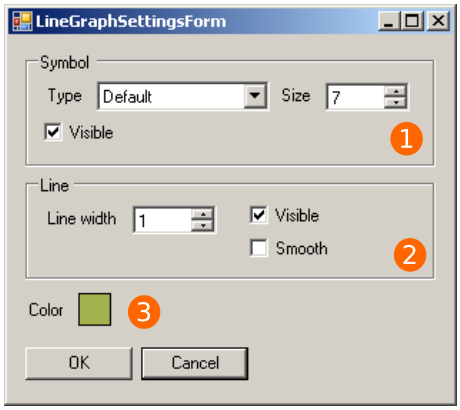
Obrázek 4.12: Dialog pro zobrazování tabulek a grafů



Obrázek 4.13: Detail panelu pro graf nebo tabulku s číselnou a textovou osou

2. Tloušťku, viditelnost a "zaoblenost" čáry grafu

3. Barvu čáry grafu.



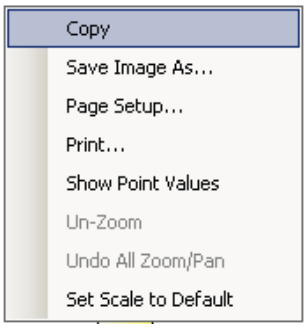
Obrázek 4.14: Nastavení grafu s číselnými osami

Při zobrazení jsou tabulky i grafy umístěny do hlavního okna aplikace. Po kliknutí na tabulku pravým tlačítkem myši se zobrazí *kontextová nabídka* s volbou *Export* pro uložení tabulky. Tabulku je možné uložit do XML, formát je podobný tabulkám používaným v XHTML [4].

Podobně i u grafů se při stisknutí pravého tlačítka myši zobrazí *kontextová nabídka* (viz obr. 4.15). Kromě uložení grafu do souboru (*Save Image As*) lze graf zkopírovat do schránky nebo vytisknout. Zobrazený graf je možné přibližovat a oddalovat pomocí pohybu kolečka myši, při "označení" oblasti pomocí levého tlačítka je daná oblast přiblížena. Po stisku kolečka myši lze měnit zobrazovanou oblast grafu.

4.2.6 Logování

Průběh dávkové analýzy je zaznamenáván do logu. Logované události mají přidělenu svou úroveň důležitosti – od informačních výpisů až po ohlášení chyby. Ve výchozím nastavení je zaznamenán začátek analýzy souboru (úroveň důležitosti `INFO`) a chyby, které se vyskytnou při zpracování dokumentu (např. nedodržení schématu při validující analýze – úroveň `ERROR`). Každý záznam logu obsahuje datum a čas, úroveň důležitosti a zprávu vztahující se k zaznamenané události. Výpis na obr. 4.16 ukazuje



Obrázek 4.15: Kontextová nabídka okna s grafem

část výpisu z logovacího souboru.

```
2009-07-17 16:32:59,737 INFO   -- Analysing file:///Z:/novels-xhtml/tdatr10.html
2009-07-17 16:32:59,837 INFO   -- Analysing file:///Z:/novels-xhtml/litlst10.html
2009-07-17 16:32:59,987 INFO   -- Analysing file:///Z:/novels-xhtml/alibit210.html
2009-07-17 16:33:00,057 INFO   -- Analysing file:///Z:/novels-xhtml/redcl10.html
```

Obrázek 4.16: Ukázka výpisu z logovacího souboru

Přímo v aplikaci lze prohlížet log pomocí okna s logovacími výpisy (je dostupné z hlavního menu *Tools – View log*). Logovací výpisy jsou také zaznamenávány do souboru, ve výchozím nastavení se soubor s výpisy jmenuje `log-file.txt` a nachází se v adresáři aplikace. Více o změně výchozího nastavení v kapitole 4.1.2.

4.3 Ukázková analýza

V následující analýze je předveden postup, kterým lze analyzovat kolekci XML dokumentů a jejich příslušné schéma. Obsah kolekce tvoří romány uložené ve formátu XHTML, celkový počet dokumentů je 992 a jejich velikost 469 MB. Tato testovací data lze nalézt na přiloženém CD spolu s výsledky jejich analýzy. Nastavení dialogů použitých v ukázkové analýze je možné nalézt v adresáři `SavedConfigs` v hlavním adresáři aplikace.

Pro každý XML soubor z kolekce bude spočten:

- počet elementů s různými jmény
- počet elementů v závislosti na hloubce dokumentu
- počet atributů s různými jmény
- počet rekurzivních elementů s různými jmény

Při analýze schématu bude spočten:

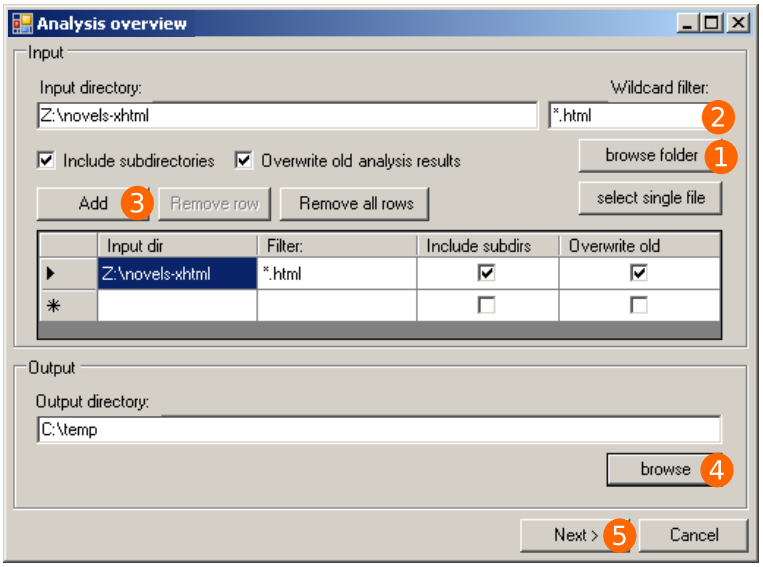
- počet elementů s různými jmény
- počet atributů s různými jmény
- počet rekurzivních elementů s různými jmény
- nejdelší cesta v schématu

Z jednotlivých výsledků analýzy XML souborů budou následně spočítány i charakteristiky celé kolekce. Budou jimi:

- průměrný počet elementů v závislosti na hloubce dokumentu
- průměrný počet elementů s různými jmény
- průměrný počet atributů s různými jmény
- medián z počtu rekurzivních elementů s různými jmény
- maximum z hloubek dokumentů

Nakonec budou srovnány výsledky pro kolekci a schéma.

V menu hlavního okna aplikace vybereme položku *Analysis* a následně *New analysis*. V dialogu pro výběr souborů (obr. 4.17) pomocí tlačítka *browse* (1) zvolíme adresář se vstupními XML soubory, nastavíme hodnotu textového pole *Wildcard filter* (2) na **.html* a přidáme mezi vstupní adresáře pomocí tlačítka *Add* (3). Tlačítkem *Browse* (4) vybereme adresář pro výsledky analýzy a stiskem tlačítka *Next* (5) pokračujeme na další obrazovku dialogu.

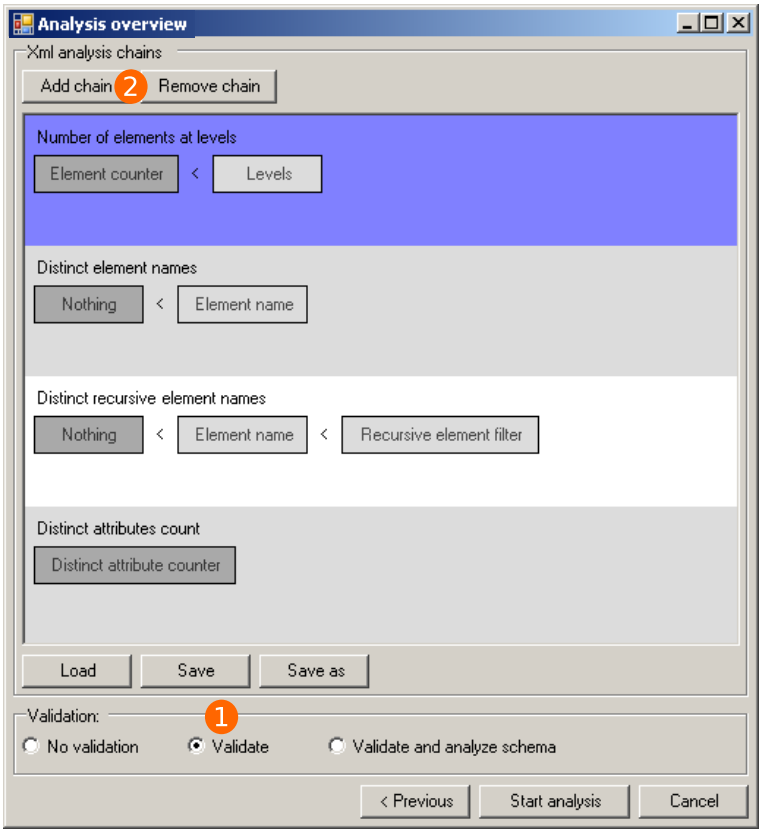


Obrázek 4.17: Obrazovka dialogu pro výběr souborů

V druhé obrazovce dialogu (obr. 4.18) vybereme "řetízky" k analýze dokumentu a schématu. Nejprve nastavíme typ analýzy (1) na *Validate* nebo *No validation*. Dále přidáme "řetízky" pro analýzu XML souborů pomocí tlačítka *Add* (2). Přepneme typ analýzy na *Validate and analyze schema* a přidáme "řetízky", které ale budou sloužit k analýze schématu. Vytvořenou konfiguraci uložíme pomocí tlačítka *Save as* (3), aby ji při příští budoucí analýze nebylo potřeba znovu ručně sestavovat. Přepneme typ analýzy na *Validate* a spustíme analýzu XML souborů pomocí tlačítka *Start analysis*.

Po dokončení analýzy XML souborů obsahuje log informace o průběhu analýzy. Log je možné prohlédnout buď v souboru (při výchozím nastavení má jméno `log-file.txt` a nachází se v hlavním adresáři aplikace) nebo přímo v aplikaci (položka v menu *Tools – Log window*). Protože byla vybrána validující analýza, byly výsledky zapsány pouze pro validní dokumenty. Jinými slovy – pokud se dokument odkazoval na nějaké schéma a nepodařilo se jej podle tohoto schéma ověřit (příp. neprošel validací), nebyly pro tento dokument počítány výsledky.

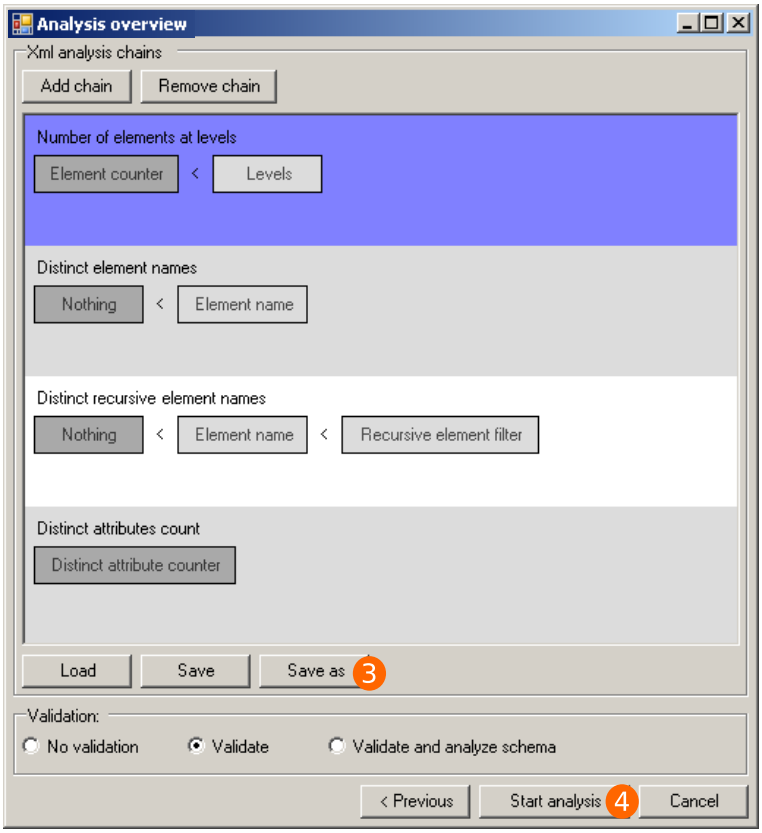
Pro analýzu schémat opět vybereme z menu *Analysis – New analysis*,



Obrázek 4.18: Obrazovka dialogu pro výběr "řetízků" pro analýzu XML souborů

hodnoty zachováme stejné jako při analýze XML souborů, pouze v druhé obrazovce přepneme typ analýzy na *Validate and analyze schema* a spustíme analýzu.

Soubory s výsledky se nacházejí ve výstupním adresáři. Jejich umístění kopíruje umístění analyzovaného souboru. Např. při zvoleném výstupním adresáři `C:\temp` pro analyzovaný soubor s cestou `Z:\novels-xhtml\1built10.html` bude soubor s obsahem analýzy umístěn do souboru `C:\temp\file\Z\novels-xhtml\1built10.html.xmlrep`. Soubory s analýzou XML mají příponu `xmlrep`, v souborech s příponou `xsrep` jsou uloženy informace o schématech užitých v daném XML souboru. Soubory s příponou `xsrep` obsahují odkazy na soubory s příponou `rep`, ve

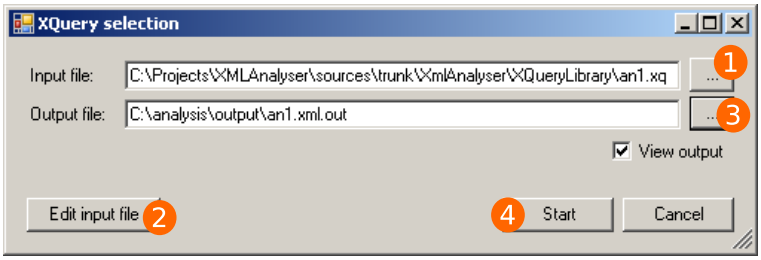


Obrázek 4.19: Obrazovka dialogu pro výběr "řetízků" pro analýzu schémat

kterých je uložena vlastní analýza schématu DTD nebo XSD. Důvodem k tomuto dvouúrovňovému odkazování na analýzu schématu je skutečnost, že dokument může odkazovat na více schémat, která má splňovat.

Nyní spočítáme souhrnné charakteristiky z výsledků pro jednotlivé soubory. V menu hlavního okna aplikace vybereme *Analysis – New XQuery*.

V dialogu pro souhrnnou (na obr. 4.20) analýzu vybereme umístění souboru se zdrojovým kódem analýzy v jazyce XQuery (1). Pro ukázkovou analýzu je zdrojový kód uložen v souboru `XQueryLibrary\an1.xq`. Soubor otevřeme pomocí tlačítka *Edit input file* (2) a upravíme cesty k souborům s analýzou. Vybereme soubor, do kterého bude uložen výsledek (3) a spustíme analýzu (4).

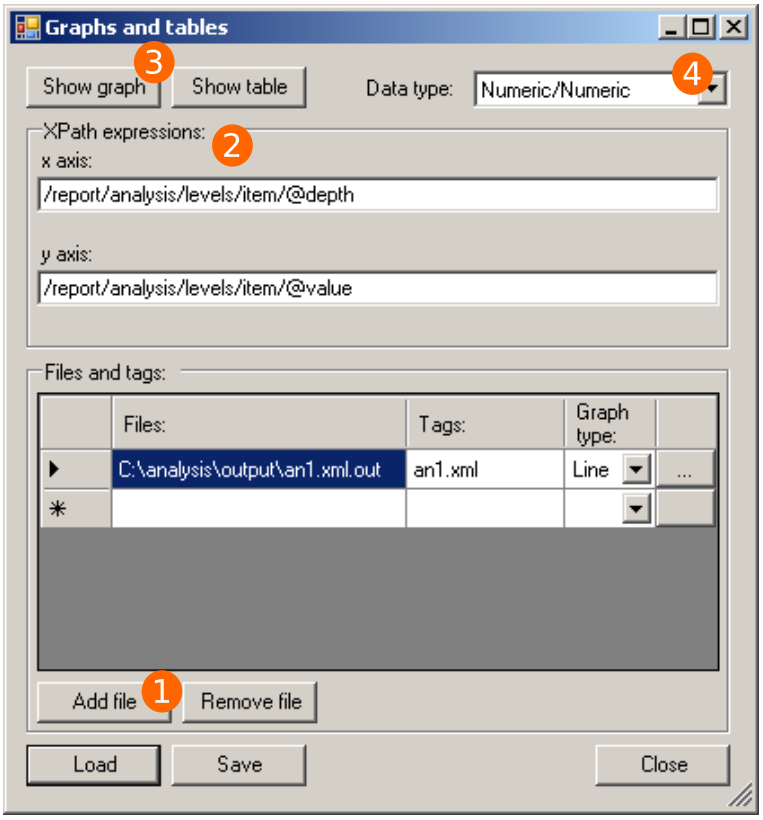


Obrázek 4.20: Dialog pro souhrnnou analýzu

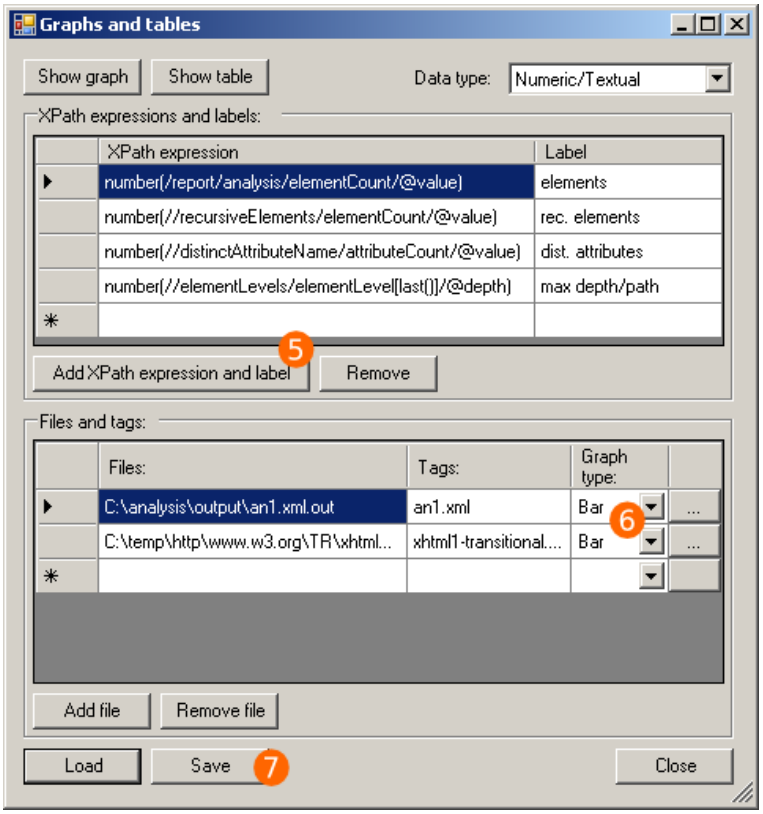
Pro grafické znázornění výsledků vybereme v menu hlavního okna aplikace *Tools – Charts and tables*, otevře se dialog z obr. 4.21. Jako první zobrazíme z výsledků kolekce průměrný počet elementů v závislosti na hloubce dokumentu. Po stisknutí tlačítka *Add file* (1) vybereme soubor s výsledkem souhrnné analýzy. Do textových polí *x axis* a *y axis* (2) zadáme XPath výrazy popisující cestu k datům ve zdrojovém souboru. Do *x axis* vložíme `/report/analysis/levels/item/@depth`, do *x axis* vložíme hodnotu `/report/analysis/levels/item/@value`. Výslednou tabulku nebo graf je možné zobrazit pomocí tlačítek *Show graph* a *Show table* (3). Grafy a tabulky se zobrazí do oblasti hlavního okna aplikace, které je přístupné po zavření dialogu (obr. 4.23).

Pro srovnání charakteristik kolekce XML dokumentů a jejich schématu změníme typ dat na *Numeric/Textual* (4) a přidáme soubor s výsledky analýzy schématu. Tlačítkem *Add XPath expression and label* zadáme XPath výrazy vedoucí k datům vybraných charakteristik. Změníme typ grafů na sloupcový tlačítky (6) (obr. 4.22) a opět necháme zobrazit tabulku a graf (obr. 4.24). Pro příští použití uložíme nastavení dialogu pomocí tlačítka *Save* (7).

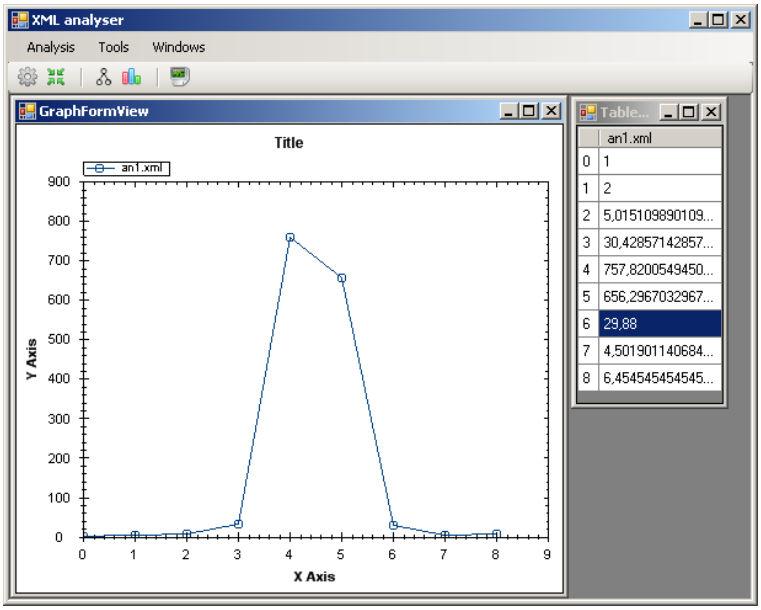
Výsledky odpovídají skupině analyzovaných souborů. Nejvíce elementů je umístěno na 4. a 5. hladině a dokumenty nejsou příliš hluboké (maximální hloubka je 8). Proti schématu používají dokumenty jen malé množství různých elemetů (prům. počet 14 proti možným 89). V dokumentech se téměř nevyskytují rekurzivní elementy (medián je 1) narozdíl od schématu, v němž je většina elementů rekurzivních (66 z 89). Rekurzivitou je ovlivněna délka nejdelší cesty schématu (schéma obsahuje cyklus), proto je délka cesty u schématu nižší než u dokumentů.



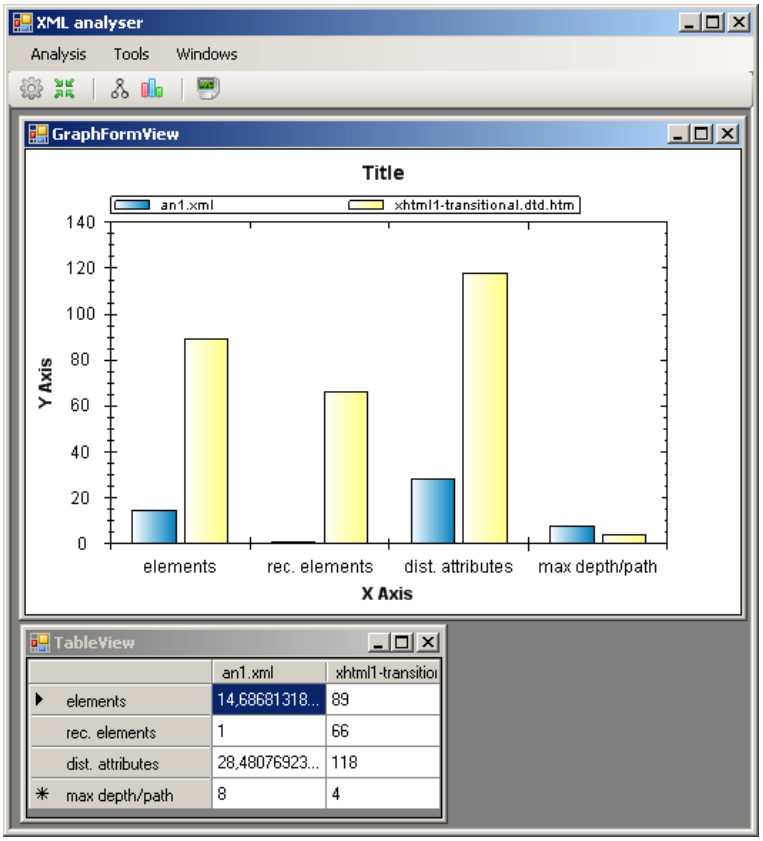
Obrázek 4.21: Dialog pro zobrazení tabulek a grafů



Obrázek 4.22: Dialog pro zobrazení tabulek a grafů



Obrázek 4.23: Hlavní okno s grafovým a tabulkovým zobrazením výsledků



Obrázek 4.24: Hlavní okno s grafovým a tabulkovým zobrazením výsledků

Kapitola 5

Programátorská dokumentace

V následujících podkapitolách jsou popsány použité nástroje a knihovny. Pro jednotlivé části aplikace jsou rozebrány použité programátorské přístupy a vztahy mezi třídami.

5.1 Programovací jazyk, knihovny, běhové prostředí

Aplikace XMLAnalyser je napsána v jazyce C# s využitím vývojového prostředí Microsoft Visual C# 2008 Express Edition. Je určena pro platformu .NET 3.5 a operační systém Microsoft Windows XP a vyšší. Pro dotazování a transformaci XML souborů v jazyce XQuery je použit procesor SAXON [34]. Procesor SAXON je primárně určen pro běhové prostředí Java, běh pod platformou .NET je zajištěn díky implementaci jazyka Java (a souvisejících knihoven) knihovnou IKVM.NET [23]. Převod DTD na XML Schema zajišťuje nástroj Trang [35]. Trang je také napsán v Javě, v prostředí .NET opět využívá IKVM.NET. Pro grafové znázornění struktury XML dokumentů a XML Schémat je využito knihovny QuickGraph [32] a utility dot z balíku GraphViz [25]. Vykreslování grafů zajišťuje knihovna ZedGraph [38]. Pro logování byl využit balík Apache log4net [28].

5.2 Grafické uživatelské prostředí

K ovládání aplikace slouží grafické prostředí využívající knihoven `System.Windows.Forms`. U složitějších dialogů je funkcionalita rozdělena mezi několik tříd podobně jako u vzoru *Passive View* [10]. Název třídy sloužící jako *view* končí slovem `View` a třída obsahuje zdrojový kód vygenerovaný pomocí návrháře dialogů Visual Studia. Třída končící slovem `Controller` zajišťuje ovládací logiku a obsahuje metody pro odezvu na akce uživatele. U tříd, v nichž je výhodné oddělit samotná data od ovládací logiky, je použita třída končící slovem `Content` pro sdružení těchto dat. Ovládací prvky (textová pole, zaškrtávací pole,...) bývají s odpovídajícími daty spojena pomocí *vázání dat* [18], pro komplexnější interakci mezi ovládacími prvky je u některých tříd (*kontrolerů*) přidán *Mediátor* [11] jako vnitřní třída.

5.3 Ukládání a načítání nastavení

Mnoho dialogů v aplikaci nabízí možnost ukládání/načítání konfigurace. Tato funkcionalita je implementována pomocí mechanismu *serializace* a *deserializace* objektů [21], konkrétně s využitím třídy `BinaryFormatter`.

5.4 Dávková analýza

Následující podkapitoly popisují třídy použité v dávkové analýze (jak schémat, tak dokumentů).

5.4.1 Příprava dat pro analýzu

K výběru vstupních dat slouží sada dialogů z prostoru jmen `XmlAnalyser.Analysis.Forms`. Pomocí těchto dialogů jsou vybrány vstupní XML dokumenty, na kterých bude pouštěna analýza, a vstupní "řetízky" určující analýzu.

Aby bylo možné ze vstupních adresářů vybírat pouze některé soubory (např. podle přípony), informace o vstupním adresáři je uložena do třídy `InputItemsWrapper`. V ní jsou kromě cesty uloženy informace o filtru, který musí jméno vstupního souboru v adresáři splňovat (tzv. *wildcards*), zda se má adresář procházet rekurzivně a zda se mají přepisovat výsledky

předchozí analýzy.

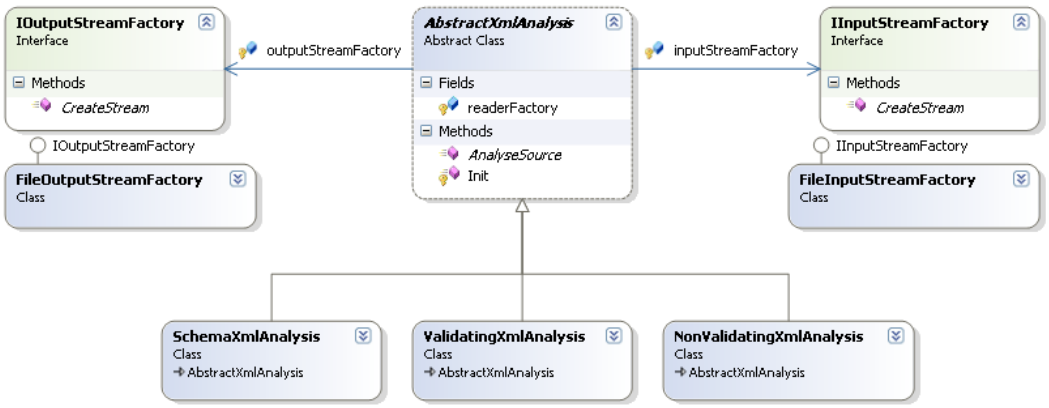
Vlastní "řetízek" reprezentuje třída `NamedChain`. Kromě názvu řetízku a jeho popisu je ve třídě přítomný seznam objektů splňujících rozhraní `IAbleToGetProductInfo`, toto rozhraní splňují třídy pro vytváření komponent pro analýzu (jak XML dokumentů, tak schémat).

Po stisknutí tlačítka *Start analysis* je vytvořen objekt pro příslušnou analýzu (podrobněji v 5.4.2) a spolu s informacemi o vstupních datech jsou tato data předána objektu třídy `FilesProcessingController`. Objekt této třídy nejdříve projde vstupní adresáře a vybere vstupní soubory a následně na nich pouští předanou analýzu. Příprava vstupních souborů a vlastní analýza jsou spouštěny ve zvláštním vlákně, aby se po čas výpočtu dalo s aplikací pracovat. Průběžný postup analýzy je zobrazován v okně třídy `FilesProcessingForm`.

5.4.2 Typy dávkové analýzy, vstup a výstup

Vlastní dávková analýza je reprezentována abstraktní třídou `AbstractXmlAnalysis`. Ta obsahuje abstraktní metodu `AnalyseSource`, která přebírá objekt s identifikátorem vstupu a výstupu, zajišťuje analýzu předaného vstupu a zapsání výsledků na výstup. Odvozené třídy dále určují typ analýzy. Třída `NonValidatingXmlAnalysis` slouží k nevalidujícímu typu analýzy, `ValidatingXmlAnalysis` navíc ověřuje schema dokumentu podle DTD nebo XML Schema, pokud na ně dokument odkazuje. Třída `SchemaXmlAnalysis` slouží k analýze schématu daného XML dokumentu.

Pro abstrakci nad vstupem a výstupem v prostředí .NET slouží *proud* – objekt třídy `System.IO.Stream`. V průběhu analýzy třídy analýzy nepotřebují znát původ dat – zda byla získána ze souboru, po síti nebo z paměti, potřebují pouze získat *proud dat* pro čtení obsahující XML data. To samé platí pro výstup. Kvůli této abstrakci nevytváří potomci třídy `AbstractXmlAnalysis` proud samy, ale využívají metod `CreateStream` rozhraní `IInputStreamFactory` a `IOutputStreamFactory` . Toto rozhraní implementují třídy `FileInputStreamFactory` a `FileOutputStreamFactory`. Třída `SchemaXmlAnalysis` navíc potřebuje pro svou činnost získávat *proud* se schématem dokumentu, k tomuto účelu slouží rozhraní `ISchemaStreamFactory`, které implementuje třída



Obrázek 5.1: Třída AbstractXmlAnalysis a její potomci

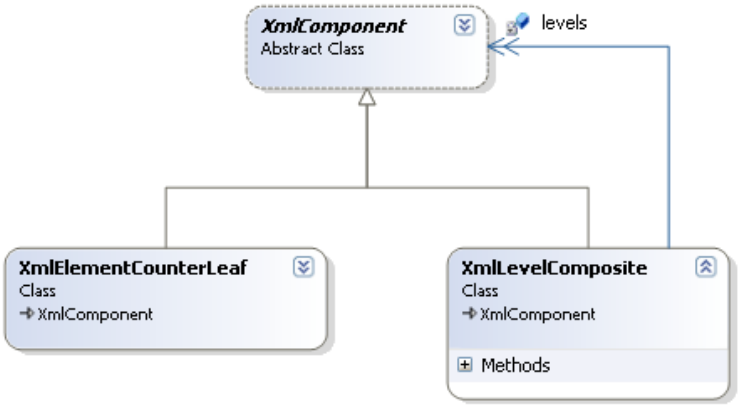
`FileSchemaStreamFactory`.

5.4.3 Struktura tříd určených k analýze XML dokumentů

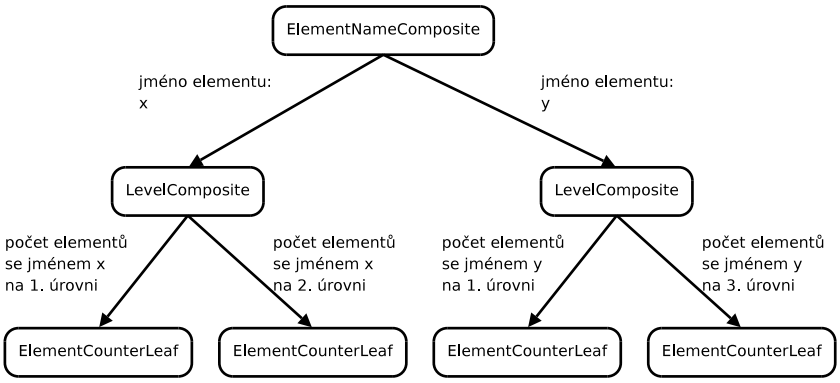
Konkrétní třídy určené k analýze XML dokumentů používají strukturu návrhového vzoru kompozit [11]. Všechny konkrétní třídy mají společného předka – abstraktní třídu `XmlComponent`. Třídy z ní odvozené se dělí z hlediska struktury na dvě skupiny. První skupinou jsou tzv. *kompozity*, tj. třídy, které mají reference na objekty typu `XmlComponent`. Druhou skupinou jsou tzv. *listy* – třídy, které už nemají další reference na objekty typu `XmlComponent`. Vztah tříd je ukazuje obr. 5.2.

Tento přístup má dvě výhody. Tou první je možnost libovolně skládat strukturu objektů ze zmiňovaných tříd. Jedinou podmínkou je nutnost dodržet stromovou strukturu objektové hierarchie.

Druhou výhodou je jednotný přístup ke všem třídám. Protože všechny třídy jsou odvozeny od společné abstraktní třídy, mají i společné rozhraní. Objekt, který funguje jako "kontejner", může zavolat některou z metod tohoto rozhraní na objektech, které shromažďuje, aniž by věděl, jakého jsou typu (ukázka struktury objektů na obr. 5.3).



Obrázek 5.2: Vztahy mezi třídou XmlComponent a jejími potomky



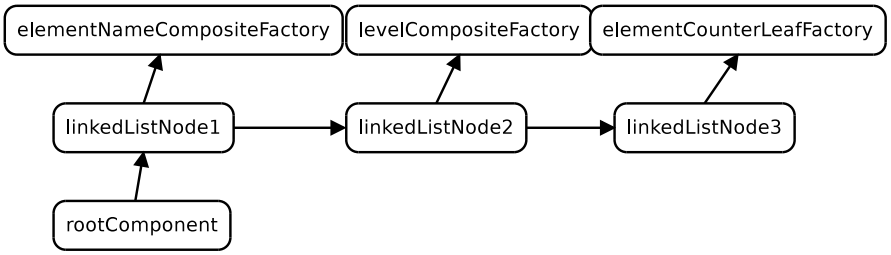
Obrázek 5.3: Příklad struktury objektů při běhu programu

5.4.4 Průběh analýzy XML souborů

Na počátku analýzy XML dokumentu je nad vstupním proudem dat vytvořen *reader dat* - objekt typu `XmlReader`. Typ *readeru* (validující či nevalidující) závisí na typu analýzy.

Při spuštění analýzy dostane třída analýzy seznam "řetízků" – spojových seznamů tzv. *továren* (viz návrhový vzor tovární metoda [11]). Továrny mají za úkol vytvářet instance příslušných typů (dále v textu jsou tyto instance nazývány *komponenty*). Následně jsou pomocí prvních

továren těchto "řetízků" vytvořeny tzv. *kořenové komponenty*, *fronty testů* a *zásobníky příkazů* (viz níže).

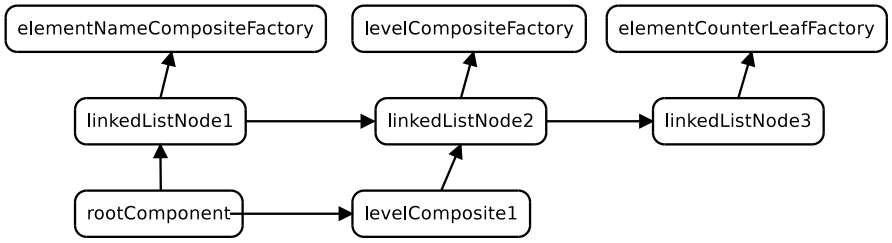


Obrázek 5.4: Situace objektů po inicializaci

Tím inicializace končí (stav zobrazen na obr. 5.4). Následuje sekvenční čtení vstupního dokumentu pomocí *readeru*. Načtený uzel dokumentu je postupně předán *kořenovým komponentám* Ty poté mohou předávat načtený uzel svým následníkům ve stromové struktuře komponent. Pokud v průběhu analýzy některá z komponent potřebuje vytvořit instanci typu, který shromažďuje, zavolá metodu `CreateChild()`. Tato metoda pomocí "řetízku" továren vyhledá továrnu příslušející následující komponentě a nechá si továrnou vytvořit požadovanou komponentu (viz obr. 5.5). Po ukončení čtení je na *kořenové komponentě* volána metoda `RemoveEmpty` , v níž jsou upraveny shromážděné výsledky. Nakonec je zavolána metoda `WriteResult` pro zápis výsledků na výstup. Tyto metody jsou opět propagovány od *kořenové komponenty* směrem k jejím potomkům. Více o konkrétních komponentách sloužících k analýze XML dokumentů v příloze B.1 na straně 70.

Všechny továrny musí splňovat rozhraní `IXmlComponentFactory`, aby bylo možné pomocí nich stejnou metodou vytvářet komponenty a aby se na ně dalo odkazovat z "řetízku".

Aby bylo možné sestavovat za běhu aplikace různé "řetízky", je nutné u každé třídy odvozené z `XmlComponent` vědět, která továrna slouží k vytváření instancí dané třídy. Tato metainformace, která se vztahuje k samotné třídě (typu), je ke kódu přidána pomocí tzv. *atributu* [13] a při běhu programu je získána pomocí mechanismu reflexe. Atribut `Factory` obsahuje typ továrny, která se stará o vytváření instancí dané



Obrázek 5.5: Situace objektů: komponenta `rootComponent` vytvořila komponentu `levelComposite1`

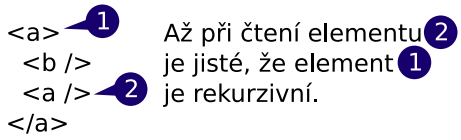
třídy. Aby nebylo nutné se starat o vytváření samotných továren, jsou všechny továrny vytvářeny staticky a přistupuje se k nim pomocí statické vlastnosti `Instance` (návrhový vzor singleton [11]). Protože by většina kódu u různých továren byla stejná, obsahuje knihovna generickou třídu `XmlGenericComponentFactory`, která po přidání typového parametru funguje jako továrna na výrobu instancí daného typu.

Dalším atributem, který je přiřazen ke každé třídě odvozené z třídy `XmlComponent`, je atribut `Name`. Atribut `Name` slouží k získání srozumitelného jména třídy.

Posledním atributem, který může být připojen k třídě, je atribut `Leaf`. Ten říká, že instance tohoto typu slouží jako listy v objektové hierarchii komponent.

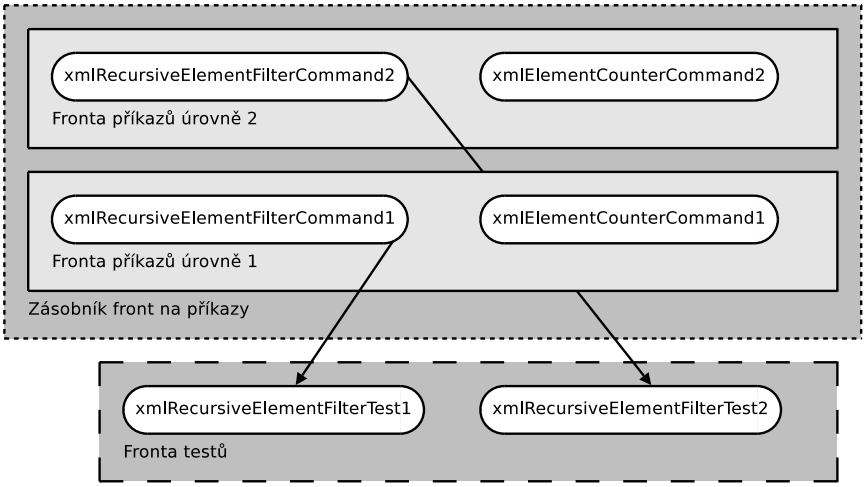
Specifickým druhem komponenty je tzv. "filtr". Filtr je v podstatě kontejner, který ale má pouze jednu referenci na následující komponentu. Příkladem takového filtru je třída `XmlRecursiveElementFilter`. Tato třída funguje tak, že následující komponentě předává pouze *rekurzivní* elementy. Element je *rekurzivní*, pokud se v jeho obsahu (nebo obsahu jeho potomků) vyskytuje element se stejným jménem. Protože při zvoleném sekvenčním přístupu k XML dokumentu není možné při načtení počátečního elementu s komplexním typem rozhodnout, jestli je element rekurzivní nebo ne, rozhodnutí je odloženo do chvíle, kdy je nalezena koncová značka tohoto elementu.

Pokud máme v analýze například řetěz továren



Obrázek 5.6: Problém rozhodování o elementu při sekvenčním přístupu

`XmlRecursiveElementFilter` – `XmlElementCounterLeaf`, požadavek zvýšení počtu elementů není vyřízen přímo při přečtení otevírací značky elementu. Místo toho příslušná komponenta typu `XmlElementCounterLeaf` vytvoří instanci speciální třídy – příkaz, v tomto případě typu `ElementCounterCommand`. Podobně komponenta typu `XmlRecursiveElementFilter` vytvoří svůj příkaz typu `RecursiveElementFilterCommand`. Obě tyto třídy jsou potomky abstraktní třídy `XmlCommand` a typ jejich chování je variací na návrhový vzor Command [11]. Oba příkazy jsou vloženy do fronty příkazů. Pro každou úroveň (hloubku) zanoření v XML dokumentu existuje jedna fronta, všechny fronty jsou potom uloženy v zásobníku. Každá fronta odpovídá právě jednomu komplexnímu elementu.



Obrázek 5.7: Objektová struktura zásobníku, front, příkazů a testů

Při vytváření objektu typu `XmlRecursiveElementFilterCommand` je vytvořen tzv. *test* (v tomto případě objekt třídy

`XmlRecursiveElementFilterTest`). Test je zařazen do fronty testů společné pro *řetízek*. Při načtení nového uzlu XML dokumentu je tento uzel postupně předán všem testům ve frontě testů. Námi zařazený test zkoumá, jestli se neobjeví v dokumentu XML element, který by měl stejné jméno jako dříve načtený komplexní element. Při dosažení koncové značky elementu je odebrána ze zásobníku vrchní fronta. Následně se postupně prochází příkazy ve frontě, a pokud test příslušný danému příkazu je označený jako splněný (metoda `PassedTest` příkazu vrátí `true`), je příslušný příkaz splněn. Při prvním nesplnění testu se zbytek příkazů ve frontě zahodí. Při procházení se testy zároveň vybírají z fronty testů. Některé třídy nepotřebují test (např. třída `XmlElementCounterCommand`) – v tom případě se při procházení neexistující test pokládá za splněný.

5.4.5 Struktura tříd určených k analýze schémat

Struktura tříd určených k analýze schémat je téměř identická jako u tříd určených k analýze XML dokumentů. Protože při analýze schemat je využíváno modelu SOM [37] (celé schéma je drženo v paměti), není potřeba *testů* a *příkazů* jako u analýzy XML dokumentů.

5.4.6 Průběh analýzy schémat

Prvním krokem při analýze schémat je vytvoření *readeru* nad vstupním XML dokumentem (objekt třídy `XmlReader`). Dokument je postupně procházen a při objevení odkazu na DTD nebo XML schéma (interní nebo externí) je spuštěna analýza tohoto schématu. Schéma je vždy analyzováno v kontextu XML elementu, pro který udává schéma platnost. U DTD je jím vždy kořenový element XML dokumentu, u schémat XML Schema se může jednat o jakýkoliv element v dokumentu. Výsledky pro daný element a dané schéma jsou zapisovány do souborů s příponou `.rep` ve výstupním adresáři, jméno výsledného souboru závisí na umístění zdrojového souboru, jménu analyzovaného elementu a typu schématu. Nakonec jsou informace o schématech použitých v XML souboru zapsány do souboru `.xsrep` ve výstupním adresáři.

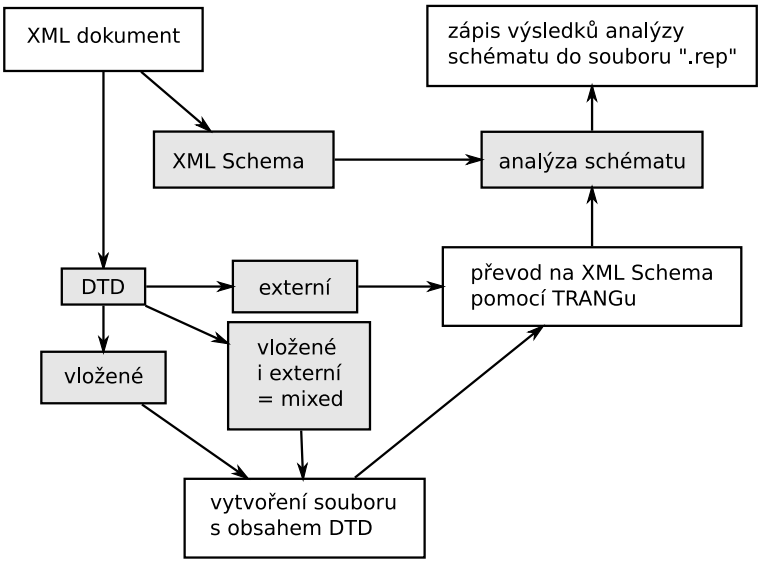
Při nalezení schématu XSD aplikace vyhledá model objektů SOM používaný *readerem* při čtení vstupního dokumentu. V modelu vyhledá element, pro který definuje schéma platnost, spustí z něj průchod do

Typ schématu	Jméno souboru s analýzou
externí DTD nebo XML Schema	`<soubor>.<element>.rep`
interní DTD	`<soubor>.<element>.dtd.rep`
interní XML Schema	`<soubor>.<element>.xs.rep`

Obrázek 5.8: Název souboru s výsledky analýzy schématu

hloubky [20] a sestaví seznam předchůdců, seznam následníků a pořadí průchodu uzlů. V tomto pořadí jsou potom procházeny uzly a analyzovány pomocí "řetízků". Více o konkrétních komponentách sloužících k analýze schémat v příloze B.2 na straně 72.

Bohužel pro práci s DTD není k dispozici podobné prostředí jako pro práci se schématy XML Schema, proto jsou schémata DTD převáděna pomocí nástroje Trang [35] na schémata XSD. Externí DTD jsou takto převedena přímo, pro DTD vložená do souboru je vytvořen soubor.



Obrázek 5.9: Postup analýzy jednoho schématu

Mnoho XML souborů při analýze odkazuje na schémata umístěná na internetu (např. XHTML [4] stránky). Aby nebylo potřeba při každé analýze znovu stahovat ten samý soubor se schématem, vytváří si aplikace *keš* těchto

vzdálených souborů. *Kešování* zajišťuje třída `XmlCachingUrlResolver` ze seskupení `SharedAssembly` Jako *keš* slouží adresář `CacheDir` v adresáři aplikace, ale lze jej změnit v konfiguračním souboru `app.config` (viz kapitolu 4.1.2) .

5.4.7 Vlastní ovládací prvky

Pro aplikaci bylo kvůli zobrazování *"řetízků"* vytvořeno několik vlastních ovládacích prvků. Ovládací prvek `ButtonCanvas` má na starosti zobrazování "řetízku". `ListPanel` a `ListPanelRow` potom zajišťují tabulkové zobrazení seznamu "řetízků". Všechny tyto ovládací prvky se nachází v knihovně `ControlsLibrary`.

5.5 Souhrnná analýza

Třídy pro souhrnnou analýzu se nachází v seskupení `XmlAnalyser` a prostoru jmen `XmlAnalyser.XQuery`.

Při souhrnné analýze je spuštěno vyhodnocení XQuery dotazu. Kvůli odezvě prostředí je spuštěno ve vlastním vlákně pomocí třídy `BackgroundWorker`. Po vytvoření *proudu* ze vstupního souboru a vytvoření *writeru* pro výstup je vyhodnocování XQuery dotazu přenecháno objektu třídy `XQueryEvaluator` z knihovny SAXON. Pokud si uživatel přeje zobrazit zdrojový XQuery dokument nebo výsledný XML dokument, je pomocí třídy `ProgramExecuter` ze seskupení `SharedAssembly` spuštěn externí proces. V něm je otevřen textový editor s výsledným souborem. Název aplikace a formát argumentů lze nastavit v konfiguračním souboru, viz kapitolu 4.1.2. Pro vstupní soubor je v konfiguračním souboru hledána aplikace s klíčem `xq`, pro výstupní dokument s klíčem `xml`.

Pro skripty v jazyce XQuery je v adresáři `XQueryLibrary` obsažena knihovna `mylib.xq`. V ní se nachází funkce pro výpočet modusu, mediánu a rozptylu.

5.6 Vykreslování XML dokumentů a schémat

Třídy pro vykreslování XML dokumentů a schémat se nachází v seskupení `GraphPainter`.

Vykreslování je podporováno pro XML soubory, schémata DTD a XSD. Při výběru vykreslování DTD je DTD je převedeno pomocí nástroje Trang na XSD. Po vybrání vstupních údajů (vstupního a výstupního souboru, typu dat a nastavení) je pomocí algoritmu průchodu do hloubky vytvořen graf reprezentující vstupní data. Vytváření grafu je zapouzdřeno ve třídách `XmlProcessing` a `XsdProcessing`, které jsou potomky abstraktní třídy `Processing`. Kvůli odezvě prostředí je tato činnost prováděna ve zvláštním vlákně. K vytvořenému grafu je následně přidáno formátování uzlů a pomocí knihovny QuickGraph je vygenerován dočasný soubor ve formátu `.dot`. Pro vyrenderování výsledného souboru je spuštěn externí proces s utilitou `dot.exe` z balíku GraphViz. Pokud uživatel zvolil, že si přeje prohlédnout výsledný soubor, je podle přípony souboru spuštěn proces s příslušným prohlížečem a výsledný soubor v něm zobrazen. Přípona, prohlížeč a formát argumentu musí být uveden v konfiguračním souboru, více v kapitole 4.1.2.

5.7 Grafové a tabulkové zobrazování výsledků

Třídy pro vykreslování XML dokumentů a schémat se nachází v seskupení `ChartsAndTables`.

Zobrazovací logika hlavního dialogu je soustředěna do třídy `SelectionFormController`. Ta shromažďuje soubory, z nichž se budou brát data, typ grafu, který bude pro daný soubor použit k vykreslení (sloupcový nebo spojový) a krátký název použitý pro soubor (*tag*). Vlastní vykreslování grafů a tabulek je přenecháno třídám implementujícím rozhraní `ISelectionFormSubController`. Pro grafy s dvěma číselnými osami je to `IntIntSelectionFormSubController`, pro grafy s pouze jednou číselnou osou `IntTextSelectionFormSubController`.

Data jsou získávána pomocí XPath výrazů. Dokument je v paměti

reprezentován pomocí třídy `XPathDocument`, hodnoty jsou vybírány pomocí instance třídy `XPathNavigator`. Vybrané hodnoty jsou při zobrazování tabulky vloženy do objektu třídy `DataGridView`, při zobrazování grafu jsou umístěny do ovládacího prvku `ZedGraphControl` z knihovny ZedGraph. Ukládání tabulky je realizováno pomocí instance třídy `XmlWriter`.

5.8 Možnosti rozšiřování programu

Pro přidání dalších komponent pro analýzu XML dokumentů či schémat je potřeba rozšířit knihovnu `AnalysisComponentLibrary`. Přidaná komponenta musí být potomkem abstraktní třídy `XmlComponent` (pro analýzu XML dokumentů) nebo `XsComponent` (pro analýzu schémat). Po rozšíření knihovny není potřeba upravovat vlastní aplikaci, stačí znovu zkompilovat pouze knihovnu. Aplikace zjistí přítomnost nových komponent po opětovném spuštění.

Pro přidání zcela jiného typu analýzy (např. analýzy XSLT skriptů) by bylo nutné vytvořit třídu, která by byla potomkem abstraktní třídy `AbstractXmlAnalysis`. Navzdory svému jménu třída může sloužit jako základ pro analýzu libovolných dat, nemusí to být nutně XML. Rozhraní třídy definuje pouze obecnou abstraktní metodu `AnalyseSource`, která slouží k analýze jednoho dokumentu. Způsob, jakým bude analýza prováděna potom závisí zcela na její implementaci.

Knihovna `mylib.xq` v adresáři `XQueryLibrary` je využívána při souhrnné analýze a obsahuje funkce v jazyku XQuery. Psaní složitějších funkci v jazyce XQuery je poměrně obtížné, proto nabízí procesor SAXON možnost rozšíření XQuery skriptů pomocí funkcí v jazyce Java. Rozsáhlejší funkce by tedy mohly být začleněny touto cestou, podrobnější informace lze nalézt v dokumentaci procesoru SAXON [34].

Kapitola 6

Závěr

Cílem práce byla implementace nástroje umožňujícího provádět analýzu XML dokumentů a schémat DTD a XSD. Navíc program měl umožnit přehledné grafické zobrazení vstupních XML dokumentů a schémat, tabulkové zobrazení výsledků a možnost srovnání statistik nad XML dokumenty a jejich schématy.

Vyvinutá aplikace XML Analyser se zaměřila na strukturální analýzu XML dokumentů a jejich schémat. Zkoumá tedy např. počty elementů v dokumentech (v různých kontextech).

Pro zobrazení vstupních dat byl v aplikaci vytvořen nástroj, který vykreslí strukturu vstupního dokumentu ve formě obrázku do souboru, který je možné pomocí dalších utilit zobrazovat.

Výsledky analýz je možné zobrazovat přímo v aplikaci ve formě tabulek a grafů, grafy i tabulky je možné ukládat pro další zpracování.

Zvláštní důraz byl kladen na možnost využití komponent analýzy v různých souvislostech. Výsledkem je knihovna `AnalysisComponentLibrary`. Třídy této knihovny slouží jako "stavební kameny" při analýze jednotlivých dokumentů. Následně jsou výsledky těchto analýz zpracovány pomocí XQuery skriptu, ten může kromě vestavěných funkcí jazyka XQuery využít i funkce z knihovny `XQueryLibrary`, která je součástí aplikace.

Protože podobná aplikace dosud neexistovala, bylo nutné vytvořit vlastní způsob postupu analýzy dokumentů.

Při práci jsem zjistil, že pro běžnou činnost s XML dokumenty poskytuje prostředí .NET poměrně slušné zázemí. Problémem je podpora pokročilejších nástrojů – chybí nativní nástroje pro vyhodnocování XQuery dotazů, jazyk XPath je zastoupen pouze ve starší verzi 1.0, není k dispozici nástroj pro práci s DTD. Tyto nedostatky byly překonány použitím knihovny IKVM.NET, která umožňuje běh aplikací určených pro platformu Java pod platformou .NET.

Další práce na aplikaci by mohla zahrnovat:

- rozšíření knihovny `AnalysisComponentLibrary`
- rozšíření knihovny `XQueryLibrary` sloužící k souhrnné analýze
- integraci pokročilejšího nástroje pro manipulaci s XQuery skripty při souhrnné analýze
- další typy grafů a širší možnosti exportu tabulek
- analýzu kódování či velikosti dokumentů.

Literatura

[1] *Document Object Model (DOM) Level 1 Specification*. W3C Recommendation, October 1998.
http://www.w3.org/TR/REC-DOM-Level-1/

[2] *HTML 4.01 Specification*. W3C Recommendation, December 1999.
http://www.w3.org/TR/html401/

[3] *The Semantic Web Homepage*. W3C.
http://www.w3.org/2001/sw/

[4] *The Extensible HyperText Markup Language (Second Edition)*. W3C Recommendation, August 2002.
http://www.w3.org/TR/xhtml1/

[5] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation, November 2008.
http://www.w3.org/TR/2008/REC-xml-20081126/

[6] *XML Path Language (XPath) 2.0*. W3C Recommendation, January 2007.
http://www.w3.org/TR/xpath20/

[7] *XML Schema Part 0: Primer Second Edition*. W3C Recommendation, October 2004 http://www.w3.org/TR/xmlschema-0/

[8] *XQuery 1.0: An XML Query Language*. W3C Recommendation, January 2007.
http://www.w3.org/TR/xquery/

[9] Čepek O.: *Přednáška Složitost 1*, prezentace
http://kti.mff.cuni.cz/~cepek/Slozitost1.ppt

[10] Fowler M.: *Passive View*, článek
`http://www.martinfowler.com/eaaDev/PassiveScreen.html`

[11] Gamma E., Helm R., Johnson R., Vlissides J.M.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994.

[12] Kosek J.: *DTD – Definice typu dokumentu pod lupou*, článek
`http://www.kosek.cz/clanky/xml/xml-01.html`

[13] Liberty J: *Programming C#: Attributes and Reflection*, ukázka z knihy,
`ondotnet.com/pub/a/dotnet/excerpt/prog_csharp_ch18/index.html`

[14] Matoušek J., Nešetřil J.: *Kapitoly z diskrétní matematiky*, 139–143, Nakladatelství Karolinum, Praha, 2003.

[15] Mlýnková I., Toman K., Pokorný P.: *Statistical Analysis of Real XML Data Collections (Technical Report)*, článek
`http://www.ksi.mff.cuni.cz/~mlynkova/doc/tr2006-5.pdf`

[16] Nic M.: *XML Tutorial*, článek
`http://www.zvon.org/xxl/XMLTutorial/General/book.html`

[17] Pokorný J., Halaška I.: *Databázové systémy*, str. 82–121, ČVUT, Praha, 2004

[18] Sells Ch.: *C# a Winforms, programování formulářů Windows*, str. 461–486, Zoner Press, Brno, 2005

[19] Srivastava R.: *XML Schema: Understanding Namespaces*, článek
`oracle.com/technology/pub/articles/srivastava_namespaces.html`

[20] Töpfer P.: *Algoritmy a programovací techniky*, 101–108, Prometheus, Praha, 1995.

[21] Virius M.: *Od C++ k C#*, str. 180–183, Kopp, České Budějovice, 2002.

[22] Analyzer 1.0
`http://urtax.ms.mff.cuni.cz/anaxml/`

[23] IKVM.NET, an implementation of Java for Mono and the Microsoft .NET Framework (verze 0.36)
`http://www.ikvm.NET`

[24] Deterministic and Non-Deterministic Schemas, MSDN Library
http://msdn.microsoft.com/en-us/library/9bf3997x(VS.71).aspx

[25] GraphViz – Graph Visualization Software (verze 2.22)
http://www.graphviz.org

[26] How to Use Wildcards, The Linux Information Project
http://www.linfo.org/wildcard.html

[27] Java.sun.com – The Source for Java Developers
http://java.sun.com

[28] Apache log4net (verze 1.2)
http://logging.apache.org/log4net/index.html

[29] XmlReader Class, MSDN Library
http://msdn.microsoft.com/en-us/library/system.xml.xmlreader.aspx

[30] Microsoft .NET Framework
http://www.microsoft.com/NET/

[31] NetBeans Platform
http://platform.netbeans.org/

[32] QuickGraph, Graph data structures and algorithms for .NET (verze 2.0)
http://www.codeplex.com/quickgraph

[33] SAX – Simple API for XML
http://www.saxproject.org/

[34] The SAXON XSLT and XQuery Processor (verze 9.1)
http://saxon.sourceforge.NET/

[35] Trang – Multi-format schema converter based on RELAX NG (verze 20030619)
http://www.thaiopensource.com/relaxng/trang.html

[36] Using MDI in C#
http://www.dreamincode.net/forums/showtopic57601.htm

[37] *XML Schema Object Model (SOM)*, MSDN Library
http://msdn.microsoft.com/en-us/library/bs8hh90b(VS.71).aspx

[38] ZedGraph (verze 5.1) http://www.zedgraph.org

Příloha A

Obsah přiloženého CD

K bakalářské práci je přiloženo CD s instalačními soubory, testovacími daty a elektronickou verzí tohoto textu.

Adresář `Installation` obsahuje:

- soubor `dotnetfx35.exe` pro instalaci běhového prostředí .NET 3.5
- balíček `graphviz-2.22.2.msi` pro instalaci balíku graphviz 2.22.2
- samorozbalovací archiv `XmlAnalyser-bin.exe` s aplikací XML Analyser

Adresář `Source` obsahuje soubor `XmlAnalyser-src.zip` se zdrojovými soubory aplikace.

V adresáři `TestData` jsou k dispozici testovací data použitá pro ukázkovou analýzu spolu s výstupy dávkové analýzy. Jsou zabalena v souboru `TestData.zip`

Soubor `XMLAnalyser.pdf` obsahuje text této práce.

Příloha B

Přehled komponent

V následující kapitole jsou popsány třídy, které jsou obsaženy v knihovně `AnalysisComponentLibrary` a slouží jako "stavební bloky" při dávkové analýze.

B.1 Komponenty pro analýzu XML dokumentů

B.1.1 Kompozity

XmlComplexMixedContentElementFilter

`XmlComplexMixedContentElementFilter` posílá další komponentě pouze elementy s *komplexním smíšeným obsahem*. Element má *komplexní smíšený obsah*, pokud obsahuje alespoň jeden podelement s *netriviálním* obsahem – tj. podelementem, který není ani prázdný, ani neobsahuje pouze text. Jiné uzly než elementy jsou ignorovány.

XmlElementNameComposite

`XmlElementNameComposite` rozděluje elementy podle jejich jména. Jiné uzly než elementy jsou ignorovány.

XmlLevelComposite

`XmlLevelComposite` rozděluje uzly podle hloubky, na které se nacházejí.

XmlMixedContentElementFilter

`XmlMixedContentElementFilter` předává další komponentě pouze elementy se *smíšeným obsahem*. Element má *smíšený obsah*, pokud obsahuje text a (případně) další elementy.

XmlRecursiveElementFilter

`XmlRecursiveElementFilter` předává následující komponentě pouze *rekurzivní* elementy, tj. elementy, které obsahují ve svém podstromě element se stejným jménem.

XmlSimpleMixedContentElementFilter

`XmlSimpleMixedContentElementFilter` posílá pouze elementy s *jednoduchým smíšeným obsahem*. Element má *jednoduchý smíšený obsah*, pokud každý z jeho elementů je prázdný nebo obsahuje pouze text.

XmlTrivialElementFilter

`XmlTrivialElementFilter` předává pouze elementy, které jsou prázdné nebo obsahují pouze text

XmlTrivialRecursiveElementFilter

`XmlTrivialRecursiveElementFilter` posílá jen elementy, které jsou *rekurzivní* a zároveň v jejich podstromu se nevyskytuje element s jiným jménem, než je to jejich.

B.1.2 Listy

XmlAttributeCounterLeaf

`XmlAttributeCounterLeaf` počítá celkový počet atributů u elementů, které mu jsou předány.

XmlDistinctAttributeCounterLeaf

`XmlDistinctAttributeCounterLeaf` počítá počet různých jmen atributů u elementů, které dostane.

XmlElementCounterLeaf

`XmlElementCounterLeaf` počítá celkový počet elementů, které mu jsou předány.

XmlElementFanOutCounterLeaf

`XmlElementFanOutCounterLeaf` zaznamenává počty elementů s daným *fan-outem* – tj. počtem podelementů, které jsou přímými potomky tohoto elementu.

XmlNothingLeaf

`XmlNothingLeaf` slouží jako jednoduchý ukončovač "řetízku". Vhodný ve chvíli, kdy není potřeba žádný konkrétní *list* a zajímá nás pouze hodnota předchozího kompozitu.

B.2 Komponenty pro analýzu schémat

B.2.1 Kompozity

XsAttributeNameComposite

`XsAttributeNameComposite` třídí atributy podle jména, jiné objekty než atributy ignoruje.

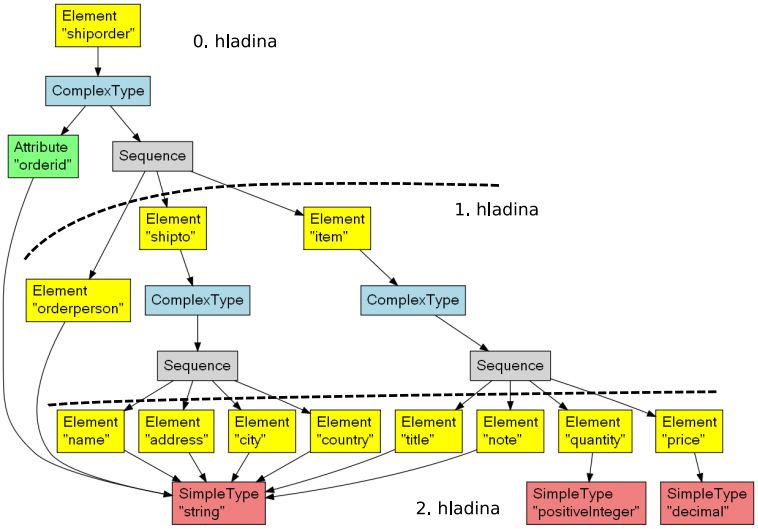
XsDistinctAttributeNameFilter

`XsDistinctAttributeNameFilter` pošle další komponentě vždy jen jeden atribut s daným jménem.

XsElementLevelsComposite

`XsElementLevelsComposite` třídí objekty podle jejich *elementové hladiny*. *Elementová hladina* pro element znamená počet jeho předchůdců – elementů ve stromě, který vznikl průchodem do hloubky z kořenového elementu. Jiné uzly, než jsou elementy, potom mají stejnou *elementovou hloubku* jako jejich nejbližší předchůdce – element.

Kvůli zefektivnění jsou *elementové hladiny* spočítány pro každé schéma jen jednou a následně referencovány pomocí statické proměnné. Proto není možné pouštět v jedné instanci aplikace dvě analýzy s touto komponentou paralelně.



Obrázek B.1: Elementové hladiny schématu

XsElementNameComposite

`XsElementNameComposite` rozřazuje elementy podle jména. Jiné objekty, než jsou elementy, ignoruje.

XsLongestElementPathsFilter

`XsLongestElementPathsFilter` vybírá pouze elementy, které mají ve schématu nejvyšší *elementovou hladinu* (viz výše) nebo elementy, které jsou jejich předchůdci ve stromě, který vznikl průchodem do hloubky z kořenového elementu schématu.

Kvůli zrychlení jsou opět použity statické proměnné, podobně jako u B.2.1, proto není možné pouštět dvě analýzy s touto komponentou v jedné aplikace paralelně.

XsRecursiveObjectFilter

`XsRecursiveObjectFilter` vybírá pouze objekty, které jsou *rekurzivní*, tzn. existuje z nich orientovaná cesta zpět do jejich uzlu. K nalezení rekurzivních objektů je použito algoritmu pro hledání *silně souvislých komponent grafu* [9].

Kvůli zrychlení jsou opět použity statické proměnné, podobně jako u B.2.1, proto není možné pouštět dvě analýzy s touto komponentou v jedné aplikace paralelně.

B.2.2 Listy

XsAttributeCounterLeaf

`XsAttributeCounterLeaf` počítá počet atributů, které mu jsou poslány.

XsElementCounterLeaf

`XsElementCounterLeaf` počítá celkový počet elementů, které jsou mu poslány předchozí komponentou.

XsElementFanInCounterLeaf

`XsElementFanInCounterLeaf` zaznamenává počty objektů, které jsou rodiči daného elementu. Kořenový objekt `schema` přitom není započítáván.

XsElementFanOutCounterLeaf

`XsElementFanOutCounterLeaf` zaznamenává počty objektů, které jsou přímými potomky typu předaného elementu. Např. pro element, jehož typ obsahuje dva různé atributy a textový obsah, si zapamatuje hodnotu *"3"*.

XsNothingLeaf

`XsNothingLeaf` slouží jako jednoduchý ukončovač "řetízku".