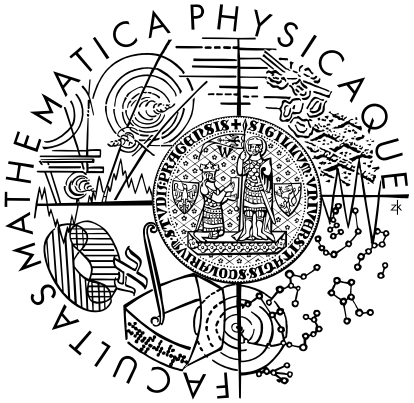
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jakub Michalko

Generátor testovacích XML dat

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Irena Mlýnková, Ph.D.

Studijní program: Informatika, programování

2011

Chcel by som veľmi poďakovať vedúcej mojej práce RNDr. Irene Mlýnkovej za jej pomoc, čas a cenné rady ktoré mi dávala pri písaní tejto práce. Svojím blízkym a kolegom ďakujem za zhovievavosť a psychickú podporu.

Prehlasujem, že som svoju bakalársku prácu napísap samostatne a výhradne s použitím citovaných prameňou. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe dňa 23. 5. 2011 Jakub Michalko

Obsah

Název práce: Generátor testovacích XML dat
Autor: Jakub Michalko
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Irena Mlýnková, Ph.D.
e-mail vedoucího: irena.mlynkova@mff.cuni.cz

Abstrakt: Cieľom práce je implementácia nástroja, ktorý generuje testovacie XML dáta na základe užívateľom definovaných parametrov. Tie bude môcť užívateľ použiť pre otestovanie aplikácie, ktorá pracuje s XML dátami. Aplikácia poskytuje užívateľovi GUI rozhranie pre editáciu parametrov vygenerovaných dokumentov, ktorých veľkosť môže presahovať veľkosť operačnej pamäte. Medzi hlavné parametre vygenerovaného dokumentu patria: počet vygenerovaných dokumentov, definovanie atribútov, počet elementov, vetvenie (fanout), frekvencia zastúpenia elementov/textu, generovanie zmiešaného dokumentu. Aplikácia ďalej umožňuje vygenerovať DTD alebo XSD schému pre vygenerovaný dokument, spustiť testovanú aplikáciu nad vygenerovaným dokumentom/dokumentami alebo odoslať dokument službe, ktorú testovaná aplikácia poskytuje.

Klíčová slova: XML, XSD, generátor dát, šablóna, testovacie dáta

Title: Generator of Testing XML Data
Author: Jakub Michalko
Department: Department of Software Engineering
Supervisor: RNDr. Irena Mlýnková, Ph.D.
Supervisor's e-mail address: irena.mlynkova@mff.cuni.cz

Abstract: The aim of the thesis is to implement a tool which generates testing XML data according to user defined attributes. The user can use this data for testing applications which work with XML data. The application provides the user with GUI for editing attributes of generated documents whose size can be greater than main storage. Main attributes of generated documents are number of generated documents, fanout, frequency of element/text, generating mixed content, etc. Application can also generate a DTD or XSD document for all generated documents, execute the tested application over generated documents or send generated data to service of tested application.

Keywords: XML, XSD, data generator, template, testing data

1. Úvod

Pri vývoji aplikácie stúpa jej zložitosť, čím sa môže aplikácia buď spomaliť, alebo sa v nej môžu vyskytnúť chyby. Týmto problémom sa predchádza pomocou testovania aplikácie ako na rýchlosť, tak aj na správnosť výsledku. Testovanie „ručne“ je ale časovo, a teda aj finančne, náročné a preto sa snažíme v praxi tento proces automatizovať čo najviac. V súčasnej dobe pracuje mnoho aplikácií s XML dátami [1]. Môže ich používať buď ako komunikačný prostriedok s inými vrstvami aplikácie, inými aplikáciami či systémami. Môže ich používať ako vstupné dáta alebo dokonca databázu. Takéto XML dáta však musia dodržiavať istú štruktúru, ktorá býva popísaná v DTD (Document Type Definition) [1] alebo XSD (XML Schema Definition) dokumente [2], aby aplikácia danému dokumentu rozumela a akceptovala ho. Manuálne vytváranie takých dokumentov je časovo veľmi náročné, alebo pri veľkých dokumentoch nedostatočne rôznorodé a nemusí pokrývať všetky prípady, ktoré sa môžu stať pre aplikáciu kritickými.

V tejto práci sa budeme zaoberať hlavne tým, ako vytvoriť dostatočne veľké, zložité a čo najrozličnejšie XML dokumenty tak, aby zachovali štruktúru definovanú užívateľom. Medzí základné nároky, okrem štruktúry, ktoré sú kladené na generovanie XML dát patrí napríklad počet a frekvencia opakovania sa elementov, dátové typy atribútov a textových elementov, alebo počet detských elementov. Okrem týchto nárokov sa snažíme, aby vygenerovaný dokument bol nie len dostatočne veľký, tj. väčší ako operačná pamäť, ale aby aj rýchlosť generovania bola čo najlepšia, čo v kombinácií niektorých požiadavkov môže spôsobovať problémy, ktoré riešime, alebo sa im snažím predísť. Ďalšou témou, ktorú rozoberieme, je definovanie vlastných dátových typov a ich použitie.

V rámci práce bola vypracovaná aplikácia XMLGenerator, zameraná na generovanie XML dokumentov. Aplikácia dokáže na základe vyššie spomenutých atribútov generovať veľké XML dokumenty, pričom dokáže skombinovať jednotlivé atribúty. Pomocou modulov dokáže vygenerovať širokú paletu dát alebo ďalej spracovať vygenerovaný dokument.

Práca je rozdelená do niekoľkých kapitol. Druhá kapitola obsahuje úvod do problematiky, ako aj stručný popis XML dát, dátových typov, DTD dokumentu, XSD dokumentu, XPath a iných technológií, ktoré boli použité pri riešení problematiky a implementovaní aplikácie. Tretia kapitola porovnáva a diskutuje nad rozdielmi už existujúcich aplikácií, ktoré sa danú problematiku snažia riešiť, alebo ich je možné použiť pri riešení problematiky. Štvrtá kapitola sa zaoberá používaním aplikácie. Obsahuje návod na inštaláciu, popis GUI rozhrania a tiež spôsob, akým XML dokument vygenerovať. Piata kapitola obsahuje popis implementácie. Nájdeme v nej použité algoritmy, popis dát, s ktorými pracuje, rozhranie s modulmi a návod, ako si vytvoriť vlastný modul. Šiesta kapitola zhrňuje celú prácu. Obsahuje subjektívny popis kladných vlastností, poukazuje na niektoré nedostatky a naznačuje ďalšie možné rozšírenia.

2. Popis použitých technológií

V tejto kapitole v stručnosti popíšeme technológie, ktoré boli použité pri implementácií a úvod do problematiky generovania XML dokumentov.

2.1. XML

XML (eXtensible Markup Language) [3] je v doslovnom preklad „rozšíriteľný značkovací jazyk“. Jeho podstatou je dokázať rozoznať dáta v texte. Na to slúži XML formát, ktorý definuje oddeľovacie znaky < a > a elementy, ktoré sú týmito značkami oddelené. Elementy môžu obsahovať text, môžu byť prázdne, môžu obsahovať iné elementy, alebo kombináciu textu a elementov. Elementy naviac môžu obsahovať atribúty, kde každý má v rámci elementu unikátne meno. Krátky príklad, ako vyzerajú XML dáta je v Ukážke 1.

```
<hlavnyelement atribut1=”hodnota”  atribut2=”nejaka hodnota”>
      <prazdnyelement />
      <textovyelement > text elementu </textovyelement>
</hlavnyelement>
```

Ukážka 1: príklad xml dát

XML formát obsahuje mnoho ďalších definícií, ktoré ale v projekte nepoužívame.

XML dokument je textový súbor, ktorý je v XML formáte. Ten obsahuje len jeden hlavný element, ktorý sa označuje ako koreňový element. Ak by sme Ukážku 1 uložili ako súbor, element *hlavnyelement* by predstavoval koreňový element.

2.2. Well formed XML a Valid XML

Všetky XML dokumenty musia byť správne štrukturované (*well formed*) [4]. To znamená, že musia byť syntakticky správne, podľa pipisu v kapitole 2.1, a XML dokument musí obsahovať práve jeden koreňový element. Ku všetkým počiatočným značkám musia existovať koncové (môžu sa vnárať, ale nie prekrývať), atribúty elementov musia mať rôzne mená.

Dokument je validný (*valid*) [3], ak je k nemu pridružený dokument definujúci jeho štruktúru (DTD a XSD, viď kapitola 2.4.1 a 2.4.2), a zároveň ju spĺňa.

2.3. Zmiešaný obsah elementu

XML Generátor dokáže generovať zmiešaný obsah (*mixed content*). Myslí sa tým štruktúra spĺňajúca *mixed content model* [4], ktorá povoľuje zmiešať text a elementy v ľubovolnom poradí.

2.4. Úvod do XSD a DTD

Aby bolo možné s XML dokumentom lepšie pracovať, mali by sme poznať jeho štruktúru. Definícia štruktúry sa môže nachádzať priamo v dokumente, alebo v inom súbore (na lokálnom počítači, alebo na nejakom serveri) a v hlavičke dokumentu je len zaznačený odkaz na tento súbor. V našom projekte budeme využívať pre vygenerované XML dokumenty tento spôsob.

Pre definovanie schémy sú najčastejšie používané XSD a DTD dokumenty.

2.4.1. DTD

DTD (Document Type Declaration) [1] patrí medzi jednoduchšie deklarácie štruktúry XML dokumentov. Definuje vzťah elementov, textu ich poradím a opakovaním k elementu, ktoré pripomína regulárny výraz. V ukážke 2 je príklad DTD dokumentu, ktorý popisuje XML dokument z ukážky 1.

```
<!ELEMENT hlavnyelement (prazdnyelement,textovyelement)>
<!ATTLIST hlavnyelement atribut1 CDATA #IMPLIED>
<!ATTLIST hlavnyelement atribut2 CDATA #IMPLIED>

<!ELEMENT prazdnyelement EMPTY>
<!ELEMENT textovyelement (#PCDATA)>
```

Ukážka 2: príklad DTD dokumentu

2.4.2. XSD

XSD (XML Schema Definition) [2] definuje štruktúru XML dokumentov oveľa podrobnejšie ako DTD naviac s podporou dedenia. XSD súbor je sám XML dokumentom. Priamo v XML dokumente môžeme k elementu pripojiť jeho definíciu z XSD. Užívateľ si sám môže definovať vlastný dátový typ, ako napríklad *e-mail*, ktorý nepatrí k základným dátovým typom, ale dá sa definovať pomocou regulárneho výrazu. V ukážke 3 je uvedený príklad XSD súboru popisujúceho XML dokument z ukážky 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="hlavnyelement">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="prazdnyelement"/>
        <xs:element name="textovyelement" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="atribut1" type="xs:string"/>
      <xs:attribute name="atribut2" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Ukážka 3: príklad XSD dokumentu

2.4.3. Dátové typy

Dátové typy tvoria základ všetkých dát v XML Schema. XML Schema poskytuje sadu vstavaných primitívnych dátových typov [5], ako napríklad string, integer, number, alebo date, ktoré sú hierarchicky usporiadané, kde koreňom je typ

any. Vo všeobecnosti môžeme dátové typy rozdeliť na jednoduché a komplexné. My sa budeme zaoberať len jednoduchými dátovými typmi, ich dedením a obmedzeniami (*restricitons*).

2.5. DOM

DOM (Document Object Model) [6] je objektový model popisujúci XML dokument. Slúži pre čítanie a manipuláciu dát v dokumente. Rovnako ako XML dokument má stromovú štruktúru. Jeho základom je XML node, predstavujúci uzol v strome, od ktorého sú zdedené objekty reprezentujúce element, atribút alebo celý dokument. Tie poskytujú funkcie a informácie špecifické pre daný uzol, ako napríklad zoznam atribútov, detských alebo nadradených elementov. V súčasnosti je DOM implementovaný do všetkých programovacích jazykov. Jeho nevýhodou je nutnosť celý XML dokument držať v operačnej pamäti. V našej aplikácii je použitý pre editovanie XTL súborov, čo sú XML dokumenty.

2.6. SOAP

SOAP (Simple Object Access Protocol) [7] je protokolom popisujúcim spôsob výmeny správ používajúcim sa hlavne v http serveroch. Správy sú tvorené XML dátami, čím poskytuje prostredie pre zložitejšiu komunikáciu. SOAP správa pozostáva z dvoch hlavných častí (podobne ako HTML stránka): HEADER (hlava) a BODY (telo). Header nie je vyžadovaný a zvyčajne obsahuje informácie o prenose, či autentifikáciu. BODY obsahuje prenášané dáta.

2.7. XPath

Jazyk XPath (XML Path Language) [8] je dotazovacím jazykom, ktorý listuje uzly v XML dokumente. Definícia XPath pripomína cestu k súboru v adresári.

V následujúcom príklade vyberáme mená tých, ktorý bývajú v domoch s číslom 21:

```
//ulica/dom[@cp=21]/osoba/@meno
```

Ukážka 4: príklad XPath

2.8. Regular expression

Regexp (regular expression) [9] je regulárny výraz, ktorý reprezentujeme celú množinu reťazcov, ktoré pomocou iterácie môžu byť z neho vytvorené. Jeho prednosťou je, že môže byť reprezentovaný nedeterministickým konečným automatom [10], ktorý pracuje veľmi rýchlo. Využívaný je hlavne pri vyhľadávaní v textoch. Je implementovaný v mnohých aplikáciách (ako napríklad grep) a využívaný v mnohých jazykoch. Príkladom je aj XPath. Väčšina implementácií je orientovaná v z mysle zistenia, či text splňuje regulárny výraz. V našom prípade

sme ale potrebovali opačnú implementáciu, a to síce na základe regexpu vygenerovať taký text.

Základné značky/symboly, ktoré Regexp používa :

- `.` - bodka. V texte nahradzuje akýkoľvek znak.
- `*` - hviezdička. Znak uvedený pred hviezdičkou sa opakuje 0 až nekonečne krát. V kombinácií s bodkou `.*` dostávame výraz, ktorý spĺňa každý text
- `[ ]` - hranaté zátvorky. V ich nutrí sa definuje zoznam znakov, ktoré nahradzujú jeden znak. Napríklad výrazu `[ab]*` odpovedá text `babbaa`. V prípade, aby sme nemuseli vypisovať všetky znaky abecedy, alebo čísel, môžeme do zátvoriek pomocou znaku `-` definovať intervaly znakov. Napríklad `a-z` predstavuje všetky malé písmená, a `[a-zA-Z0-9]` predstavuje akékoľvek číslo alebo malé/veľké písmeno anglickej abecedy.
- `{x,y}` – počet výskytu, kde x predstavuje najmenší počet a y najväčší možný počet. Napríklad regulárny výraz `[0-9]{1,3}` spĺňajú všetky čísla menšie ako 1000, napr. `4`, `584`, alebo `054`.
- `\` – v prípade, že potrebujeme v texte kontrolovať nejaký znak, pričom v Regexpe patrí medzi základné značky/symboly, používame znak `\` , ktorý vyruší jeho význam v Regexpe. Napríklad výrazu `a.b` vyhovuje text `axb`, ale výrazu `a\.b` vyhovuje jedine text `a.b` .

2.8.1. Regular Expression Data Generator

Pri implementácií bola použitá knižnica *rxdg* (*Regular Expression data*) [11] pre generovanie reťazcov z regulárneho výrazu. Jeho zdrojové kódy sú voľne dostupné na stránke http://code.google.com/p/rxrdg/.

Pre použitie ale museli byť opravené jeho nedostatky:

- pri zadaní množiny znakov `[a-z]`, negeneroval posledný znak. (súbor *Visitor.DataGenerator.cs*). Príčinou bolo znamienko ostrej nerovnosti <, vo *for* cykle, kde ale malo byť znamienko <= (vo funkcii *Visit(BracketNode node)*)
- Generovanie trvalo veľmi zbytočne dlho:
  - generátor vytváral inštancie všetkých objektov, z ktorých ale bol náhodne vybraný len jeden (v súbore *Visitor.DataGenerator.cs*, funkcia *Visit(BracketNode node)*). Oprava spočívala vo vygenerovaní len jednej inštancie objektu.
  - pri každom získavaní náhodného čísla, vytváral inštanciu objektu *Random* (súbor *Visitor.DataGenerator.cs*). V opravenej verzii v implementácii sa vatvára len jeden statický objekt.

2.9. SQL

SQL (Structured Query Language) [12] je jazyk určený pre listovanie a manipuláciu s dátami uložených v RDBMS (relational database management system). RDBMS je systém určujúci vzťahy medzí jednotlivými dátami. Tie sú väčšinou implementované pomocou tabuliek (so stĺpcami), medzi ktorými sú definované väzby a nad ktorými sa prevádzajú jednotlivé dotazy. Základnými príkazmi SQL sú:

- listovanie: `SELECT` stĺpce `FROM` tabuľky `WHERE` podmienka
- vloženie dát: `INSERT INTO` tabuľka `(`stĺpce`)` `VALUES (`hodnoty`)`
- zmeny dát: `UPDATE` tabuľka `SET` stĺpec=hodnota `WHERE` podmienka
- zmazanie dát: `DELETE FROM` tabuľka `WHERE` podmienka

2.10. PostgreSQL

PotgreSQL [13] je objektovo-relačný databázový systém. Radí sa medzi najrozšírenejšie databázové systémy. Okrem základných príkazov SELECT, UDATE, INSERT a DELETE podporuje písanie funkcií, indexáciu , triggre, definovanie objektov,alebo dedičnosť.

2.11. MySQL

MySQL [14] je ďalší, vo svete obľúbený relačný databázový systém. Od počiatku jeho vývoja bol kladený dôraz na jeho rýchlosť, čo malo za následok postrádania niektorých funkcií. Okrem základných príkazov, ako SELECT, UPDATE, INSERT a DELETE, podobne podporuje tirggery, indexáciu, poddotazy, R-stromy a v posledných verziách už aj partitioning.

2.12. CSV

CSV z Comma-separated Value, je textový súbor predstavujúci tabuľku, kde riadok v súbore predstavuje riadok tabuľky a dáta v jednotlivých stĺpcoch sú oddelené vo väčšine prípadov bodkočiarkou (pôvodne boli oddelené čiarkou, preto Comma-separated)

3. Analýza existujúcich aplikácií

V súčasnej dobe existuje nepreberné množstvo aplikácií, ktoré dokážu generovať dáta. Bohužiaľ ale veľa z nich sú len jednoduché benchmarky, ktoré si musí užívateľ naimplementovať, aby dáta vygeneroval. Potom existujú aplikácie riadené skriptami, v ktorých si užívateľ pomocou cyklov generuje požadované dáta. Ich nevýhodou je, že sa s nimi pracuje a udržiava ich dosť nepohodlne a pridaná hodnota oproti skriptovacím jazykom nie je až tak veľká. Medzi najlepšie aplikácie, ktoré generujú XML dáta sa radia tie, ktoré generujú dáta podľa šablóny.

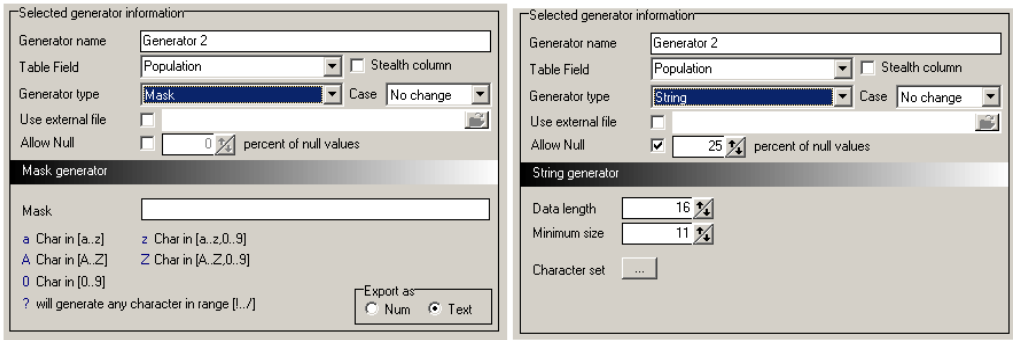
V ďalšom texte v krátkosti popíšeme výber z aplikácií, ktoré dokážu generovať XML dáta.

3.1. DataGen

DataGen, celým názvom E-NAXOS Test Data Generátor je aplikácia určená pre generovanie testovacích dát. Celá aplikácia je obsiahnutá v jednom okne, rozdelenom do niekoľkých záložiek. Jej spôsob generovania dát sa dá popísať ako generovanie tabuľky s formátovaním výstupu, kde si užívateľ môže zvoliť, ako ho chce naformátovať:

- SQL Script
- CSV
- XML
- OleDb

Pre generovanie XML dokumentov sa síce nehodí, zato pomocou XSLT sa do istej mieri dá s touto aplikáciu nahradiť XML generátor. Jej veľkou prednosťou je rôznorodosť generovaných dát. Pre generovanie dát používa generátory dát. Každý modul má špecifické atribúty, ktoré sú editovateľné v dynamickom GUI, ktoré sa mení podľa aktuálne zvoleného generátora.

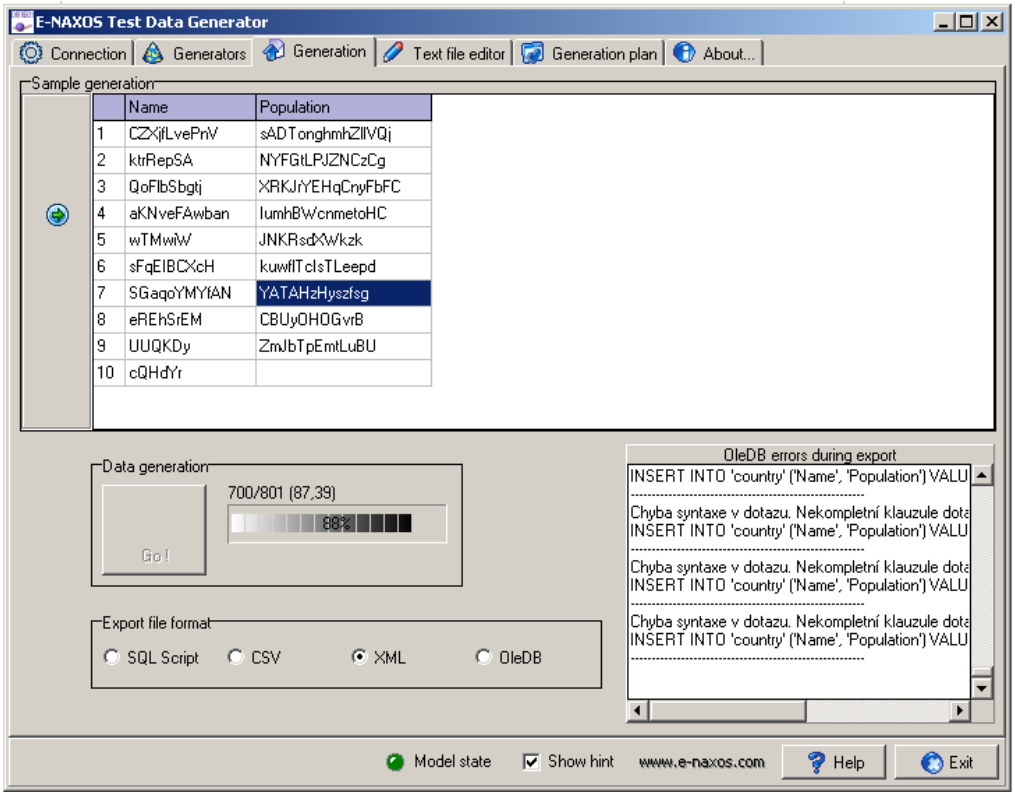


Obrázok 1: Dynamická znema GUI podľa zvoleného generátora

Každý je pomenovaný podľa typu generovaných dát :

- Address - generuje adresy

- Bit - 0/1
- City - mená miest
- Const - generuje jedinú hodnotu
- Date - dátumy
- Email - emaily
- First & Last name – mená a priezviská
- Mask – generuje stringy podľa masky
- a mnoho ďalších



Obrázok 2: generovanie dokumentu v DataGen

Na obrázku 2 je zobrazené okno, ktoré generuje výstup, pomocou ktorého sme si vygenerovali XML dokument.

Ako už bolo spomenuté, táto aplikácia generuje jednoduché výstupy. V príklade bolo zadané vygenerovať Name a Population. Výsledkom bol súbor:

```
<?xml version="1.0" encoding="windows-1250" standalone="yes"?>
<!-- Generated by E-NAXOS Data Test Generator - Standard XML
Compatibility  -->
<RECORDS>
<METADATA>
 <FIELDS>
 <FIELD attrname="Name" fieldtype="string" WIDTH="11"/>
 <FIELD attrname="Population" fieldtype="string" WIDTH="16"/>
```

```
 </FIELDS>
</METADATA>
<RECORD><ROW Name="UNlAxnWCDU" Population=""/></RECORD>
<RECORD><ROW Name="UavrbiAvB" Population="BOmhOafLSD"/></RECORD>
<RECORD><ROW Name="zsmumTMoopu" Population=""/></RECORD>
<RECORD><ROW Name="hOaBtQNWh"
Population="ZYWMFWCBTzuJVmF"/></RECORD>
<RECORD><ROW Name="NVAZoNqlRTQ"
Population="ljEUEcSzbkXu"/></RECORD>
<RECORD><ROW Name="NNpHRYUnZZ"
Population="FDSXIdRQFpSBRoHe"/></RECORD>
<RECORD><ROW Name="pGrKTHptK"
Population="rwTlEnNKVSbeDysq"/></RECORD>
<RECORD><ROW Name="vGjipIRJGCG" Population=""/></RECORD>
<RECORD><ROW Name="DpQSZeDBYj" Population=""/></RECORD>
<RECORD><ROW Name="yxWJiA" Population=""/></RECORD>
 ...
</RECORDS>
```

Ukážka 5: príklad výstupu z DataGen

3.2. Stylus Studio

Stylus Studio [15] je aplikácia primárne určená pre prácu s XML dátami. Pracuje na platforme Windows a je to platená a aplikácia. Aplikácia poskytuje grafické aj textové rozhranie. Zahŕňa v sebe veľké množstvo nástrojov, ktoré pracujú s takmer všetkým okolo XML dát. Pomocou nich dokáže :

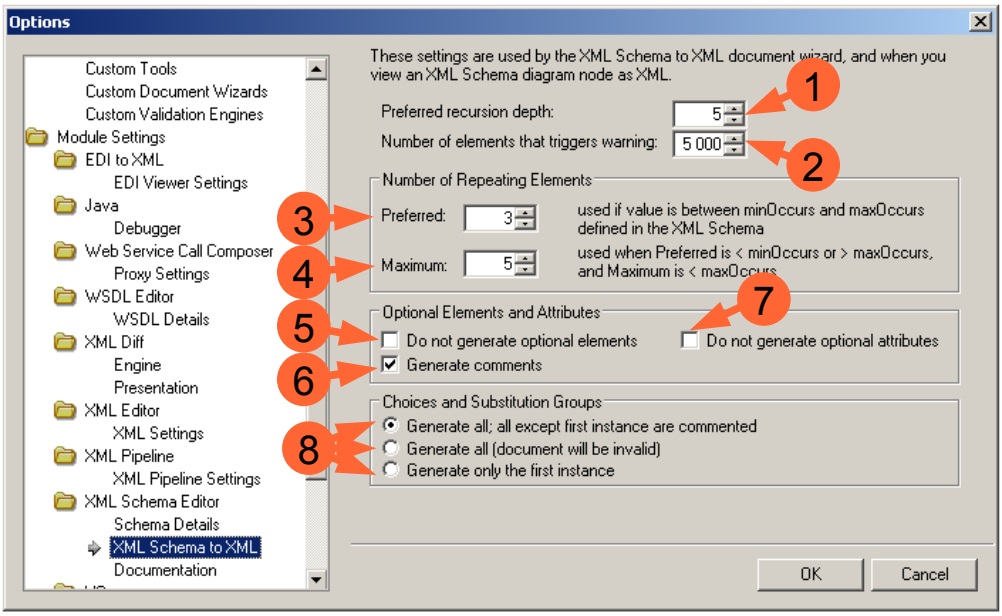
- kontrolovať, či dokument je well-formed (správne uzatvorený) a validný
- debuggovať XQuery
- pracovať s XSLT
- generovať z XML dokumentu DTD aj XSD dokument
- pracovať s webovými službami a WSDL dokumentami
- prepojiť sa s databázou
- na základe DTD alebo XSD dokáže vygenerovať XML dokument
- generovať zdrojový kód pre platformu Java, z XQuery a XSLT
- premapovať XML dáta
- a mnoho ďalšieho

Navzdory jeho prepracovanosti, je generovanie XML dokumentu z XSD alebo DTD dokumentu dosť jednoduché [16].

Z obrázka 3 je vidieť, že je málo spôsobov, ako ovplyvniť vygenerovaný dokument. Podľa očíslovania z obrázku je možné nastaviť:

1. Hĺbku rekurzie, ktorej minimálna hodnota je 1 a maximálna 5.

2. Minimálny počet elementov, ktorý vyvoláva upozornenie je 1000. V prípade, že počet elementov je viac ako tento počet, aplikácia zobrazí upozornenie s voľbou, či sa má tento dokument vygenerovať.
3. Počet elementov daného typu (v prípade, že je medzi minOccurs a maxOccurs)
4. Maximálny počet elementov daného typu (ignoruje minOccurs v prípade, že minOccurs je viac ako maximálny počet)
5. Negenerovať nepovinné elementy
6. Negenerovať nepovinné atribúty
7. Generovať komentáre
8. Spôsob generovania Substiution groups alebo Choices : všetky možnosti alebo len prvú možnosť



Obrázok 3: Nastavenie XML generátora Stylus Studio

Z preskúmania vygenerovaných dokumentov môžeme povedať, že generátor vyrobí minimum, ktoré mu DTD, XSD a nastavenie dovolí. Týka sa to štruktúry aj dát. Príklad vygenerovaných dát:

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-01-21"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///d:/Program%20Files/Stylus
%20Studio%202010%20XML%20Enterprise
%20Suite/examples/simpleMappings/purchaseOrder.xsd">
    <shipTo country="USA">
        <name>string</name>
        <name>string</name>
        <name>string</name>
```

```
        <street>string</street>
        <city>string</city>
        <state>string</state>
        <zip>1.23</zip>
    </shipTo>
    <comment>string</comment>
    <optionalAddress orderDate="1999-01-21">
        <shipTo country="USA">
            <name>string</name>
            <name>string</name>
            <name>string</name>
            <street>string</street>
            <city>string</city>
            <state>string</state>
            <zip>1.23</zip>
        </shipTo>
        <comment>string</comment>
        <optionalAddress orderDate="1999-01-21">
            <shipTo country="USA">
                <name>string</name>
                <name>string</name>
                <name>string</name>
                <street>string</street>
                <city>string</city>
                <state>string</state>
                <zip>1.23</zip>
            </shipTo>
            <comment>string</comment>
            <optionalAddress orderDate="1999-01-21">
                <shipTo country="USA">
                    <name>string</name>
                    <name>string</name>
                    <name>string</name>
                    <street>string</street>
                    <city>string</city>
                    <state>string</state>
                    <zip>1.23</zip>
                </shipTo>
                <comment>string</comment>
                <optionalAddress orderDate="1999-01-21"/>
            </optionalAddress>
        </optionalAddress>
    </optionalAddress>
</purchaseOrder>
```

Ukážka 6: príklad výstupu z Stylus studio

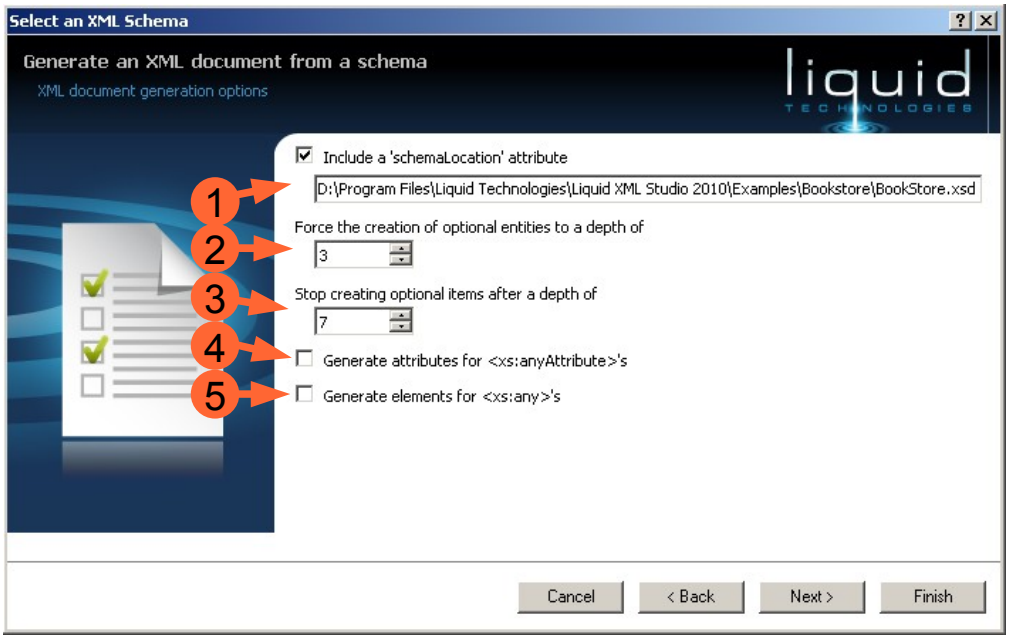
3.3. Liquid XML Studio

Liquid XML Studio [17] je rovnako ako Stylus studio aplikácia zameraná na prácu s XML dátami. Pracuje na platforme Windows a je to platená aplikácia. Tiež obsahuje mnoho nástrojov, ktoré poskytujú širokú škálu úprav XML dokumentu a dokumentov s ním spojených. Pre meditáciu dokumentov poskytuje textové aj grafické rozhranie. Nástroje Liquid XML Studio poskytujú editovanie a debuggovanie

- XSLT dokumentov
- XML dokumentov
- XQuery
- XPath
- WSDL

Jeho nástroje samozrejme toho poskytujú oveľa viac, ale nás predovšetkým zaujíma jeho XML generátor. Ten je opäť jednoduchý. Pri generovaní sa zobrazí okno s ponukou, ktorá je na obrázku 4. Na ňom je možné nastaviť (podľa číslovania na obrázku):

1. Do vygenerovaného dokumentu vložiť odkaz na XSD dokument
2. Maximálna hĺbka nepovinných elementov
3. Nevytvárať nepovinné elementy s hĺbkou väčšou ako...
4. Generovanie atribútov s rôznym obsahom
5. Generovať elementy s rôznym obsahom



Obrázok 4: Liquid XML Studio - XML Sample generator

Príklad vygenerovaných dát:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid XML Studio Developer Edition (Trial)
8.1.7.2743 (http://www.liquid-technologies.com) -->
<bs:bookstore xmlns:bs="http://www.liquid-
technologies.com/sample/bookstore"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.liquid-
```

```
technologies.com/sample/bookstore file:///D:/Program
%20Files/Liquid%20Technologies/Liquid%20XML%20Studio
%202010/Examples/Bookstore/BookStore.xsd">
    <bs:book price="-9880.3250941961" publicationdate="1983-02-
22" ISBN="string">
        <bs:title>string</bs:title>
        <bs:author>
            <bs:first-name>string</bs:first-name>
            <bs:last-name>string</bs:last-name>
        </bs:author>
        <bs:next price="-4291.2650941961" ISBN="string">
            <bs:title>string</bs:title>
            <bs:author>
                <bs:first-name>string</bs:first-name>
                <bs:last-name>string</bs:last-name>
            </bs:author>
        </bs:next>
        <bs:next price="-3860.1650941961" publicationdate="1980-
07-12">
            <bs:title>string</bs:title>
            <bs:author>
                <bs:first-name>string</bs:first-name>
                <bs:last-name>string</bs:last-name>
            </bs:author>
        </bs:next>
    </bs:book>
    <bs:book price="-9538.1250941961" publicationdate="1994-07-
25" ISBN="string">
        ...
    </bs:book>
</bs:bookstore>
```

Ukážka 7: príklad výstupu z Liquid XML Studio

3.4. ToXgene

ToXgene [18] je asi najrozšírenejší generátor syntetických XML dát. Pracuje na platforme Java. Dáta generuje na základe šablón. Podporuje zdieľanie elementov, generuje dáta na základe obmedzení zadané užívateľom, dovoľuje používať existujúce dáta pri generovaní dokumentov. Pre definovanie šablón pre generovanie XML dát používa rozšírenie XML Schema. Program sa ovláda z príkazovej riadky.

XML šablóny, ktoré majú príponu *tsl*, sa editujú pomocou ľubovoľného textového editora. ToXgene dokáže pracovať s elementmi aj atribútmi XML dát.

Celý dokument je v elemente *tox-document*. Jeho hlavná časť (element) sa nazýva *tox-document*. V ňom je definovaná štruktúra šablóny. *tox-document* predchádzajú definície, ako sa majú generovať obsahy atribútov, alebo elementov. Na tieto hodnoty sa odkazujú položky z *tox-documentu*.

Okrem definovania si vlastných typov elementov a dátových typov, má množstvo funkcionalít:

- *for* cykly,

- rekurzívne generovanie,
- distribúciu detských elementov – akým spôsobom sa má generovať ich zastúpenie v rodičovskom elemente
- inkludovanie iných súborov
- načítanie iných súborov ako zdroj dát
- podpora XPath, vlastné definovanie zoznamov
- a mnoho ďalších

Príklad vstupu:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE tox-template SYSTEM 'ToXgene2.dtd'>
<tox-template>
<tox-document name="moj">
  <element name="root" minOccurs="1000" maxOccurs="1000">
      <complexType>
            <element name="x" minOccurs="100"
                  maxOccurs="100" type="long"/>
            <element name="y" minOccurs="100"
                  maxOccurs="100" type="long"/>
            <element name="z" minOccurs="100"
                  maxOccurs="100" type="long"/>
      </complexType>
      </element>
</tox-document>
</tox-template>
```

Ukážka 8: príklad vstupu pre ToXgene

Vygenerovaný súbor:

```
<root><x>3185538623097274368</x>
<x>290146275163111424</x>
<x>-2215310321294245888</x>
…
<y>290146275163111424</y>
<y>290146275163111424</y>
…
<z>-3894721423989538816</z>
<z>1509389771401592832</z>
</root>
```

Ukážka 9: príklad vygenerovaného výstupu pomocou ToXgene

Jeho nevýhodou je, že sa ťažko dá rozšíriť o vlastné funkcie. Chýba mu grafické rozhranie. I keď je stavaný na generovanie veľkých súborov, ich generovanie je veľmi pomalé, pri ktorom nezobrazuje aktuálny stav generovania.

3.5. Požiadavky kladené na generátor

3.5.1. Vlastnosti generátora

Základom každého generátora je generovať dostatočne rozličné dáta nejakého typu, ako napríklad mená, alebo e-maily. Teda mal by mať vlastný zdroj dát, alebo zdroj dát definovaný užívateľom. V našej aplikácií sa tento bod stal základom. Je preto riešená modulárne, kde užívateľ si môže zvoliť, ktorý modul použije pre generovanie dát (modul pre získavanie dát priamo z databáze, alebo zo súbora), pričom užívateľ si sám môže naprogramovať vlastný modul, ktorý mu bude dáta generovať.

Ďalším požiadavkom pre XML generátor je zachovávať štruktúru vygenerovaných dokumentov podľa užívateľom definovanej štruktúry. S tým sa spája nie len definícia štruktúry elementov, ale aj definícia vlastných dátových typov.

Pri generovaní elementov sa nám naskytujú ďalšie nároky kladené na generátor, ako je:

- počet vygenerovaných elementov
- fanout
- počet jednotlivých elementov
- percentuálne zastúpenie elementu
- hĺbka elementu

O čosi väčšie nároky sú:

- GUI rozhranie
- rozšíriteľnosť
- generovanie veľkých dokumentov
- možnosť vygenerovať DTD alebo XSD k dokumentu

3.5.2. Naše riešenie

Najčastejšie sa opakujúce chyby generátorov sú malá (až skoro žiadna) variabilita vygenerovaných XML dokumentov, generujú buď malé dokumenty, alebo ich generujú veľmi pomaly. Veľmi malé percento spĺňa aspoň niektoré väčšie nároky spomenuté vyššie.

Naša aplikácia je jedinečná v tom, že dokáže generovať dostatočne rozličné XML dokumenty, čo do dát, tak do štruktúr elementov pričom generuje súbory väčšie ako operačná pamäť s čo najväčšou rýchlosťou. Ďalšou prednosťou aplikácie je poskytnutie GUI rozhrania a informovania užívateľa o aktuálnom stave generovania (užívateľ je informovaný, ako dlho sa dokumenty budú ešte generovať) . V neposlednom rade je jej výhodou možnosť rozšíriť aplikáciu o

moduly, ktoré sú buď zdrojom dát, alebo dokážu spracovať výstup, z čoho sa stáva veľmi užitočná pomôcka pre priame testovanie aplikácií. Naša aplikácia je tiež unikátna v tom, že dokáže spojiť podmienky ako fanout, percentuálne zastúpenie + počet jednotlivých elementov v kombinácií s definovaním celkového počtu vygenerovaných elementov (ten je potrebný vedieť pre zobrazovanie stavu generovania).

4. Užívateľská dokumentácia

4.1. Požiadavky na prostredie

XML Generator pracuje na platforme Windows XP a novších verziách, ktoré majú nainštalovaná .NET Framework 3.0. Ten je možné zadarmo stiahnuť a nainštalovať z internetu.

4.2. Inštalácia

Aplikácia je zabalená v súbore XMLGener.zip, ktorú nájdete na priloženom disku k tejto práci. V prípade, že disk nie je k dispozícii, je možné si ho stiahnuť z tohto odkazu http://code.google.com/p/xmlgener/downloads/list, kde okrem iného nájdete aj zdrojové kódy aplikácie. V súbore sa nachádza zabalený adresár XMLGener, ktorý si rozbalíte na vami požadované miesto.

4.2.1. Popis adresárov a súborov aplikácie

V adresári XMLGener je 5 adresárov:

- data - tento adresár obsahuje súbory, ktoré sa používajú ako zdroj dát pre generovanie. Z inštalácie obsahuje CSV súbory, ktoré obsahujú mená a priezviská, mestá, mená zemí a dátumy. Sami si môžete do tohto adresára nakopírovať súbory, ktoré chcete používať ako zdroj dát.
- modules - v tomto adresári sa nachádzajú knižnice, obsahujúce triedy (moduly) ktoré generujú dáta pre atribúty, alebo textové elementy alebo triedy, ktoré spracúvajú vygenerovaný súbor. Aplikácia tento adresár prechádza a z každej nájdenej knižnice sa pokúsi vytvoriť dva zoznamy modulov. Prvý obsahuje moduly generujúce dáta a druhý obsahuje moduly, ktorá spracúvajú vygenerovaný súbor.
- out – tento adresár slúži ako východzí adresár pre vygenerované dokumenty.
- templates – je východzí adresár pre šablóny, ktoré slúžia ako predloha, podľa ktorej sa vygeneruje XML súbor.
- tools – v ňom sa nachádza adresár LTFViewr, ktorý obsahuj aplikáciu LTF (Large Text File Viewer), ktorý používa aplikácia pre zobrazovanie vygenerovaných súborov. Do tohto adresáre môžu byť nahrávané vlastné aplikácie a nástroje, by boli spustitelné z príkazovej riadky a dokázali by spracovať alebo zobraziť vygenerovaný XML dokument.

V adresári sa nachádzajú knižnice, ktoré sú používané modulmi, setting.xml, ktorý obsahuje typy dát a XMLGener.exe, ktorým sa púšťa aplikácia.

4.3. Predstavenie aplikácie

Aplikácia generuje XML dokumenty podľa istej predlohy. Túto predlohu nazývame šablóna, ktorá určuje štruktúru výstupného XML dokumentu.

Šablóna je XML dokument, s príponou *xtl* (*XML Template*). Jej štruktúra vychádza zo štruktúry XML Schema [19], v ktorej sú popisované štruktúry elementov, ich atribúty, a pre dokument určuje hlavný (koreňový) element (resp. ktorý z elementov môže byť koreňovým elementom).

4.3.1. Podobnosti s XSD

Koreňovým elementom *XML Template* súbore je *xdoc*, ktorého obsah definuje šablónu (podobne ako v XSD je element *schema*). Ten obsahuje element *mainelment*, ktorý definuje koreňový element vygenerovaného XML dokumentu (podobne ako v XSD *element* s tým rozdielom, že v *XML Template* byť len jeden). V ňom sa môže nachádzať:

- element - rovnako ako je definovaný v XSD
- atribút – rovnako ako v XSD
- elements - čo je podobné s *groups* v XSD
- text node - čo vo vygenerovanom dokumente predstavuje text

4.3.2. Odlišnosti oproti XSD

Ako prvý podelement elementu *xdoc* je element *settings*, ktorý obsahuje nastavenia týkajúce sa výstupného súboru (meno, modul spracúvajúci výstupný súbor a pod.)

Ďalší v poradí je hlavný element *mainelement* popísaný v kapitole4.3.1.

Za ním následuje element *datatypes*, obsahujúci elementy *datatype*, ktoré definujú vlastné dátové typy, ktoré dedia z defaultných dátových typov. Tie sú uložené v súbore *settings.xml* a ich hierarchia kopíruje hierarchiu dátových typov v XML Schema [5] .

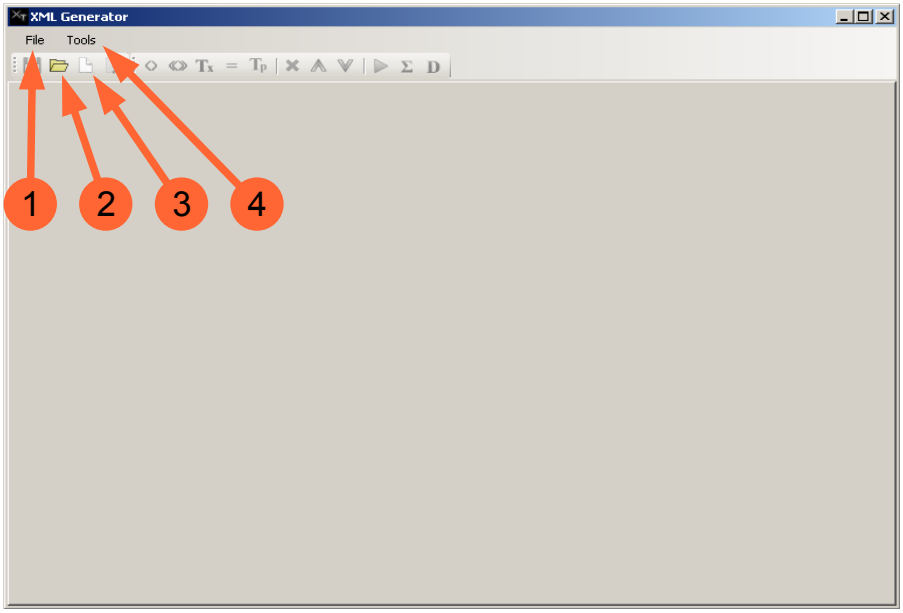
4.4. Spustenie aplikácie

Adresár, ktorý je popísaný v kapitole 4.2.1, obsahuje súbor *XMLGener.exe*, ktorým sa spúšťa aplikácia. Po spustení sa zobrazí okno, ktoré je na obrázku 5.

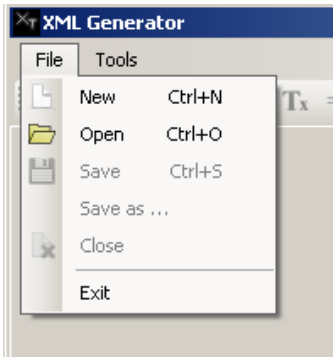
Všetky tlačítka, okrem tých, na ktoré ukazujú šípky, sú neaktívne. V tejto situácii nám aplikácia umožní len:

- otvoriť súbor: Pomocou menu tlačítka *Open* (na obrázku 6), umiestnenom v menu „File“ (na obrázku 5 šípka 1), alebo pomocou ikony na paneli nástrojov (na obrázku 5 šípka 2), alebo pomocou klávesovej skratky Ctrl+O.
- vytvoriť nový súbor: Pomocou menu tlačítka *New* (na obrázku 6), umiestnenom v menu *File* (na obrázku 5 šípka 1) alebo pomocou ikony na

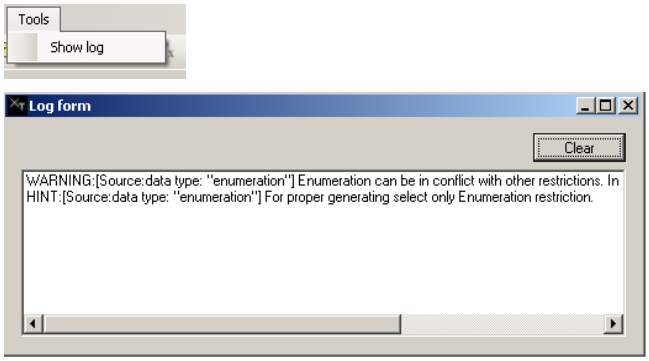
paneli nástrojov (na obrázku 5 šípka 3), alebo pomocou klávesovej skratky Ctrl+N.



Obrázok 5: Spustenie aplikácie



Obrázok 6: Otvorené menu File

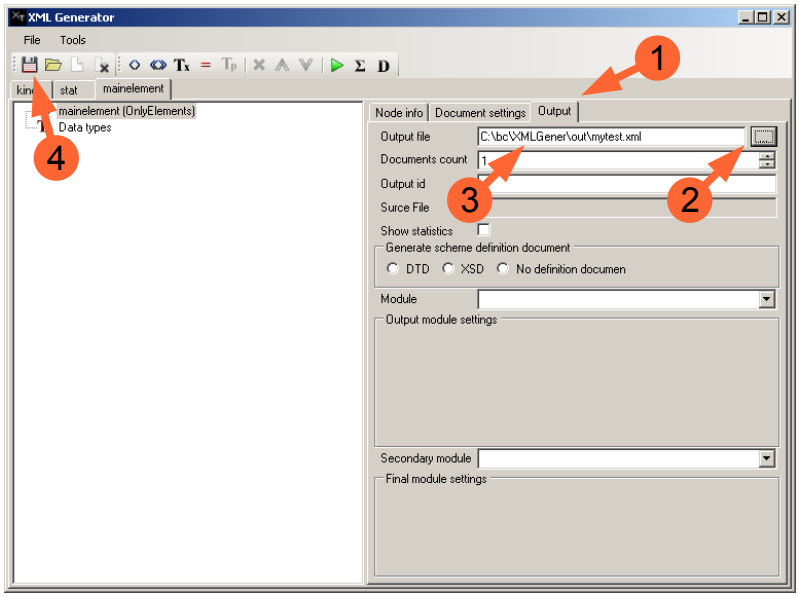


Obrázok 7: Otvorené menu Tools a Log form

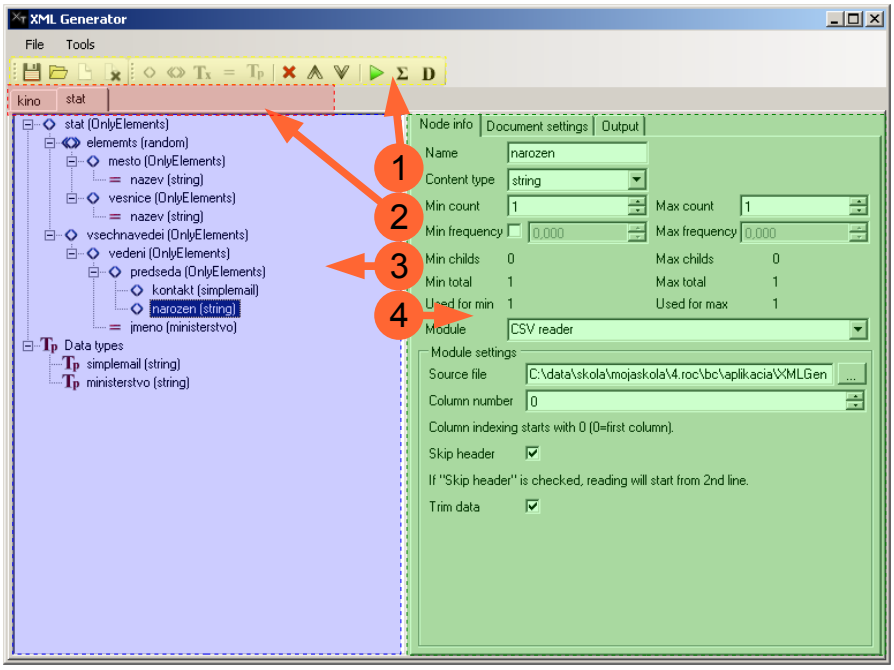
Aplikácia ešte umožňuje zobraziť okno s výpismi správ (Obrázok 7), pomocou menu tlačítka *Show log* (Obrázok 7) umiestnenom v menu *Tools* (na obrázku 5 šípka 4), ktoré je ale po inicializácií prázdne v prípade bezchybného spustenia [1]. Toto okno je využité pre zobrazovanie všetkých správ, ktoré aplikácia oznamuje užívateľovi.

1 Na obrázku 7 je pre predstavivosť schválne vyvolaná chybová hláška, ktorá sa zobrazuje pri zlej definícií dátového typu.

4.4.1. *Popis okna*



Obrázok 8: nová šablóna



Obrázok 9: rozdelenie okna

Aplikácia je rozdelená do 4 častí, ako je znázornené na obrázku 9:

1. Panel nástrojov. Pomocou ktorého editujeme stromovú štruktúrou alebo vytvárame nové šablóny.

2. Otvorené šablóny - otvorené šablóny sa zobrazujú to záložiek. V nich je napísané meno *mainelementu*.

3. V tejto časti záložky sa nachádza stromová štruktúra šablóny. Tá určuje štruktúru vygenerovaného dokumentu. Každý uzol v tejto stromovej štruktúre má zobrazené svoje meno. Až na zopár vinimiek, každá uzol má svoj charakteristický typ (napríklad dátový typ atribútu), ktorý je zobrazený v zátvorkách následujúcich za menom.

4. Vpravo je zobrazený detail šablóny, ktorý slúži pre editovanie jednotlivých uzlov v strome a pre editovanie šablóny.

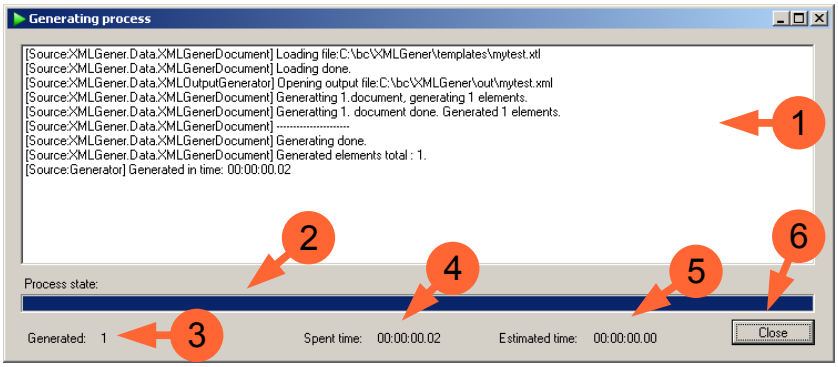
4.5. Vytvorenie novej šablóny a vygenerovanie dokumentu

Novú šablónu môžeme vytvoriť pomocou klávesovej skratky Ctrl+N, alebo iným spôsobom popísaným v kapitole 4.4. Obrázok 8 zobrazuje novo vytvorenú šablónu.

Prvé, čo si musíme nastaviť je výstupný súbor. V pravej časti okna kliknutím na záložku *Output* (šípka 1 na obrázku 8) sa nám zobrazia parametre špecifikujúce výstup. Kliknutím na tlačítko … pre výber výstupného súboru (šípka 2 na obrázku 8), sa nám zobrazí dialógové okno pre uloženie[2] súboru. Cesta súboru je zobrazená priamo v text boxe (šípka 3 na obrázku 8). Na obrázku vidíme, že výstupný súbor sme už vybrali.

Aby generátor mohol pracovať paralelne, musí byť šablóna uložená v súbore. Preto môžeme súbor uložiť pomocou tlačítka v paneli nástrojov *Save* (šípka 4 na obrázku 8), pomocou klávesovej skratky Ctrl+S, alebo pomocou menu tlačítka *Save* v menu *File*.

Výstup sa generuje pomocou tlačítka *Generate output* v paneli nástrojov. V prípade, že šablóna nie je uložená, generátor nás automaticky vyzve k uloženiu šablóny.



Obrázok 10: okno procesu generovania XML dokumentu

Pri generovaní sa zobrazí okno *Generating process* (obrázok 10), ktoré skončí generovanie veľmi rýchlo. Okno obsahuje (podľa šípok na obrázku 10):

2 Uloženie, pretože vygenerovaný súbor v podstate ukladáme

1. výpis generovania, v ktorom sa zobrazujú všetky správy, ktoré nastanú pri generovaní, tj. takzvané logy, upozornenia, chyby a nápovedy pre ich odstránenie. Pretože aplikácia umožňuje paralelné generovanie XML dokumentov, z každého výpisu musí byť jasné, z ktorej šablóny generuje, a čo je výstupom. V hranatých zátvorkách je uvedený zdroj správy (ten je pri chybových správach dôležitý). Na obrázku 10 vidíme:

    - šablónou je mytest.xtl a natiahnutie prebehlo v poriadku
    - výstupným dokumentom je mytest.xml
    - vygeneroval sa 1 dokument (je možnosť generovať viac dokumentov).
    - v 1. dokumente sa mal vygenerovať 1 element (element *mainelement*)
    - v 1. dokumente vygeneroval sa 1 element.
    - deliacou čiaru „----------“, ktorá sa zobrazuje až na záver generovania všetkých dokumentov
    - generovanie sa dokončilo v poriadku („Generating done“), ktoré inak pri prerušení je indikované správou „Generating canceled“.
    - počet vygenerovaných elementov vo všetkých dokumentoch
    - čas, ktorý zabralo generovanie všetkých dokumentov

2. stav generovania, ktorý pomocou progress baru graficky zobrazuje, v akom štádiu generovania sa práve nachádzame
3. počet doposiaľ vygenerovaných elementov
4. čas doposiaľ strávený generovaním
5. odhadovaný čas, ktorý ostáva k dokončeniu generovania. Ten je dopočítaný podľa aktuálnej priemernej rýchlosť generovania[3].
6. tlačítko, ktoré zastavuje generovanie (len pri jeho priebehu), a po jeho skončení zatvára okno.

V ukážke 14 je obsah vygenerovaného XML dokumentu.

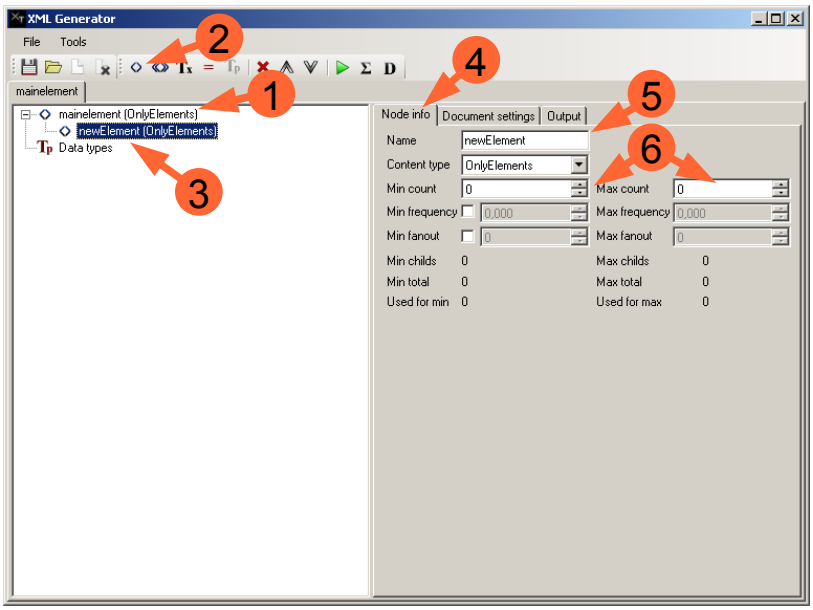
```
<?xml version="1.0" encoding="UTF-8" ?>
<mainelement/>
```

Ukážka 10: príklad výstupu prázdneho dokumentu

3 Priemerná rýchlosť generovania sa počíta zo stráveného času a počtom doposiaľ vygenerovaných elementov.

4.6. Pridanie uzlu typu element

4.6.1. Nový element s nastavením Min count a Max count



Obrázok 11: pridanie elementu

V kapitole 4.5 sme si ukázali, ako možno vygenerovať skoro prázdny XML dokument. Ak chceme, aby vo výstupe koreňový dokument nebol prázdny a obsahoval aspoň nejaký element, môžeme ho pridať tak, že si vyberieme *mainelement* (šípka 1 na obrázku 11) ako rodičovský element nového elementu a klikneme na tlačítko *New element* v *paneli nástrojov* (šípka 2 na obrázku 11).

Ak chceme premenovať *mainelement*, alebo novo vytvorený element, vyberieme ho a v pravej časti okna klikneme na záložku *Node info* (šípka 4 na obrázku 11) a zmeníme jeho *Name* (šípka 5 na obrázku 11).

Element by sa nám ale nevygeneroval, pretože má nastavený *Min count*[4] na 0 (šípka 6 na obrázku 11). Ten musíme nastaviť na 1, ak chceme, aby sa nám vygeneroval aspoň jeden element tohto typu. Pri nastavení *Min count* sa automaticky nastavil aj *Max count*[5]. Aplikácia takto automaticky zabráni neželanému konfliktu, kedy *Max count* bol menši ako *Min count*. A naopak pokiaľ nastavíme *Max count* menší ako *Min count*, aplikácia opraví *Min count*. Hodnoty, ktoré tu môžeme nastavovať sú v rozmedzí 0 až 10^9.

Pre príklad si nastavme počet na 10^6 a vygenerujme dokument. Ukážka 11 zobrazuje nami vygenerovaný XML dokument.

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

4 *Minimal count* - Minimálny počet výskytu daného element pod rodičovským elementom.

5 *Maximal count* - Maximálny počet výskytu daného element pod rodičovským elementom.

```
   <newElement/>
   <newElement/>
   <newElement/>
… cca 1000 000 elementov newElement
   <newElement/>
</mainelement>
```

Ukážka 11: príklad výstupu s jedným elementom

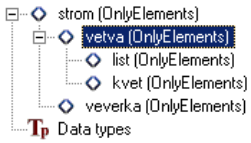
V kapitole 4.3 sme si predstavili štruktúru šablóny. Väčšinou nás zaujíma, ako vyzerá *mainelment*, alebo nejaký jeho detský uzol. Ukážka 11 má zdrojovú šablónu:

```
<mainelement name="mainelement" mincount="1" maxcount="1">
    <element name="newElement"
           mincount="1000000" maxcount="1000000">
    </element>
</mainelement>
```

Ukážka 12: dáta šablóny s jedným definovaným elementom

Všimneme si, že *Min count* sa nastavuje atributom *mincount* a *Max count* atributom *maxcount*.

Pre lepšiu názornosť sme vytvorili zložitejšiu šablónu, ktorej štruktúra je na o brázku 12 (zobrazené v GUI) a výpis zo šablóny (súboru, kde je šablóna uložená) v ukážke 13.



Obrázok 12: príklad štruktúry zo samých elementov

```
<mainelement name="strom" mincount="1" maxcount="1">
    <element name="vetva" mincount="50" maxcount="50">
      <element name="list" mincount="0" maxcount="10" />
      <element name="kvet" mincount="1" maxcount="2" />
    </element>
    <element name="veverka" mincount="1" maxcount="3" />
  </mainelement>
```

Ukážka 13: dáta šablóny s rozvetvenejšou štruktúrou

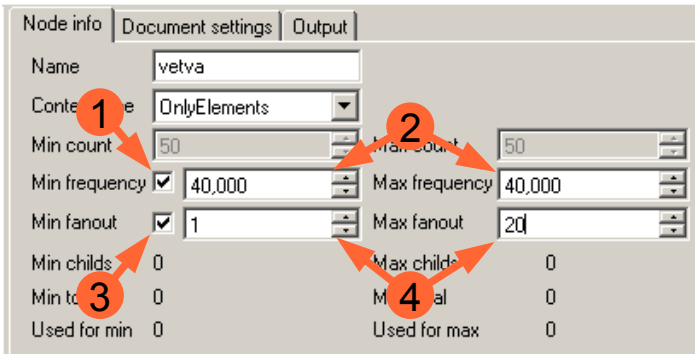
Vygenerovaný dokument podľa šablóny z ukážky 13 má mať koreňový element *strom*, v ňom presne 50 elementov *vetva*, minimálne 1 a maximálne 3 elementy *veverka*. Element *vetva* má obsahovať maximálne 10 elementov list a 1, alebo 2 elementy *kvet*. Ukážka 14 obsahuje začiatok a koniec z vygenerovaného XML dokumentu.

```
<?xml version="1.0" encoding="UTF-8" ?>
<strom>
   <vetva>
      <list/>
```

```
      <list/>
      <list/>
      <kvet/>
   </vetva>
   <vetva>
      <list/>
      <list/>
…
      <list/>
      <kvet/>
      <kvet/>
   </vetva>
   <veverka/>
   <veverka/>
   <veverka/>
</strom>
```

Ukážka 14: výstup zo šablóny s rozvetvenejšou štruktúrou

4.6.2. *Frequency a fanout*



Obrázok 13: Nastavenie fekvencie a fanoutu

V prípade, že nechceme definovať počet elementov počtom (*Min/Max count*), môžeme definovať jeho frekvenciu. K tomu slúžia parametre *Min frequency* a *Max frequency*, ktoré sa v súbore šablóny definujú atribútmi elementu *minfrequency* a *maxfrequency*. Aby sme ich ale mohli použiť, musíme najprv zaškrtnúť checkbox pre aktiváciu (šípka 1 na obrázku 13). Zaškrtnutím sa zneplatní editácia *Min/Max count* a povolí editácie *Min/Max frequency* (viditeľné na obrázku 13). V súbore šablóny sa zaškrtnutie indikuje atribútom *usefrequency*, ktorého hodnota sa musí rovnať 1. Frekvencia výskytu sa zapisuje v percentách, kde nastavené hodnoty sú v rozmedzí 0 až 100 s presnosťou na 3 desatinné miesta.

Fanout je počet detských elementov, ktoré má daný element obsahovať. Tento počet môžeme nadefinovať pomocou *Min fanout* a *Max fanout* (šípka 4 na obrázku 13). V súbore šablóny sa definujú ako atribúty *minfanout* a *maxfanout* elementu. Rovnako ako definovanie frekvencie, aj fanout musí byť povolený, a povoľuje sa zaškrtnutím checkboxu (šípka 3 na obrázku 13).

Zaškrtnutím, alebo nezaškrtnutím fanoutu alebo frekvencie vznikajú kombinácie, pri ktorých generátor mení spôsob výpočtu výsledného minimálneho a maximálneho počtu elementov, ktoré vygeneruje.

Prípad, keď v rodičovskom elemente nie je zaškrtnutý fanout a aspoň jeden element používa frekvenciu:

- v prípade, že všetky súrodenecké elementy používajú len frekvenciu, generátor vypíše užívateľovi chybu, pretože v tomto prípade je fanout potrebný (generátor nemá možnosť vypočítať, koľko elementov predstavuje nastavená frekvencia)

- v prípade, že aspoň jeden súrodenecký element nepoužíva frekvenciu, ale *Min/Max count*, a tento počet je väčší ako 0 (inak je to chyba), počty ostatných elementov sa pri generovaní automaticky dopočítavajú.

V prípade, že fanout zaškrtnutý je:

- v prípade, že žiadny element nepoužíva frekvenciu, generátor len skontroluje rozsahy (súčty *Min/Max count*) s Min/Max fanout-om.

- V prípade, že všetky súrodenecké elementy používajú frekvenciu, najprv sa prepočítajú ich minimálne počty (v podstate počítame *Min count*) podľa svojej minimálnej frekvencie a minimálneho fanoutu. V prípade, že súčet prepočítaných minimálnych frekvencií (teda suma vypočítaných *Min count-ov*) je menší ako minfanout, generátor použije ako rezervu *Max frequency*, z ktorej „ukrojí“ tak, aby výsledný súčet bol minimálne rovný *Min fantout*.

- V prípade, že niektoré súrodenecké elementy používajú frekvenciu a iné nie, najprv sa prepočíta minimálny počet elementov pre frekvencie zo súčtu *Min count* ostatných elementov. V prípade nezrovnalostí sa použijú „rezervy“ z *Max frequency*.

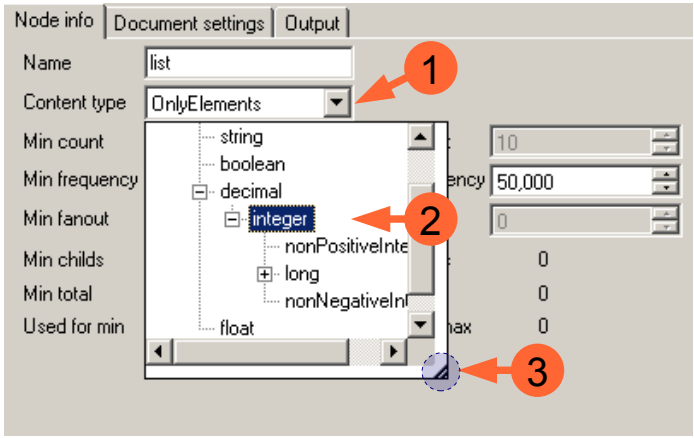
4.6.3. Content type

V kapitole 4.6.1 sme si ukázali, ako definovať element obsahujúci elementy. V prípade, že chceme, aby element, obsahoval dáta, zvolíme si, aký typ dát má element obsahovať. Element má *Content type* nastavený na *OnlyElements*. To znamená, že element nemôže obsahovať dáta, ale len uzly typu element, elements alebo text node [6]. A naopak, pokiaľ element má nastavený *Content type*, nemôže obsahovať elementy. V prípade, že element má nadefinované detské uzly, tieto uzly zmiznú v prípade, že mu nastavíme nejaký dátový typ[7]. *Content type* nastavíme kliknutím na komponentu pre výber dátového typu (šípka 1 na obrázku 14), ktorá zobrazí strom hierarchicky usporiadaných dátových typov. Výber

6 Uzly typu elements a text node sú vysvetlené v kapitolách 4.10. a 4.11.

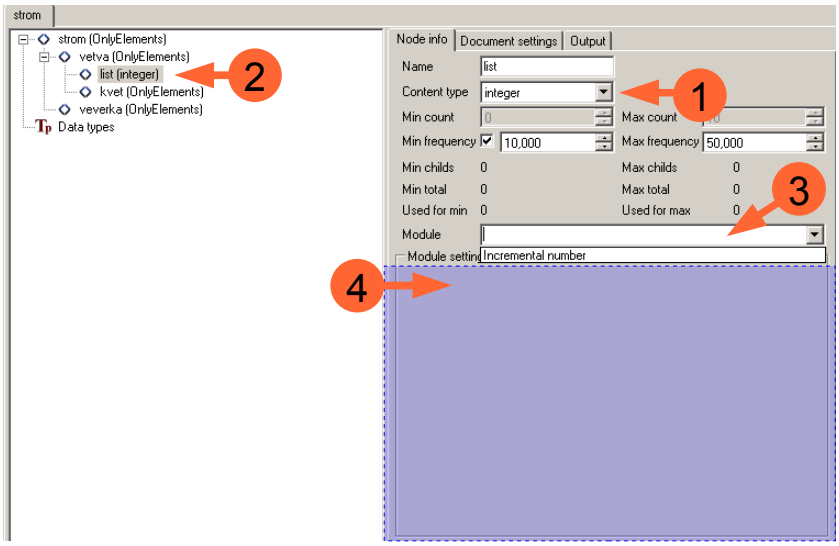
7 Dátový typ je uzol v hierarchickej stromove štruktúre, kde koreňom je *any*. ElementsOnly nie je dátový typ.

dátového typu v strome sa robí pomocou dvojkliku (šípka 2 na obrázku 14). Ak nám nevyhovuje veľkosť zobrazeného stromu, môžeme ho zväčšiť uchopením za pravý dolný roh (šípka 3 na obrázku 14).



Obrázok 14: výber dátového typu elementu

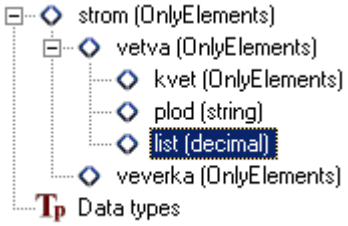
Po vybraní dátového typu sa zmenil obsah komponenty *Content type* (šípka 1 na obrázku 15), jeho popis v zátvorkách v paneli zobrazujúcom šablónu (šípka 2 na obrázku 15), objavila sa komponenta *Module* (šípka 3 na obrázku 15) pre výber modulu, ktorý generuje dátové typy a panel s jeho nastavením (šípka 4 na obrázku 15). Na obrázku 15 je tiež zobrazené, z akých modulov si môžeme vybrať. Vidíme, že pre dátový typ *integer*, si môžeme vybrať len jeden modul, a to „Incremental number“.



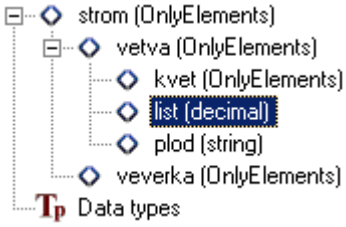
Obrázok 15: element s nastaveným dátovým typom

4.6.4. *Editácia poradia a mazanie*

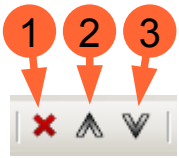
Pri generovaní sa vytvárajú elementy a atribúty podľa poradia[8], ktoré je definované v šablóne. Toto poradie môžeme meniť pomocou tlačítok posunu v paneli nástrojov (šípky 2 a 3 na obrázku 18).



Obrázok 16: príklad a)



Obrázok 17: príklad b)

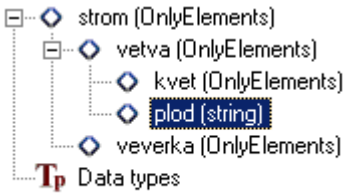


Obrázok 18: tlačíka posunu a mazania

Najprv vyberieme uzol, ktorý chceme posunúť (uzol list na obrázku 16), a pomocou tlačítka *Move element/attribute up*, uzol posunieme nahor (výsledok je zobrazený na obrázku 17). Podobne pomocou tlačítka *Move element/attribute down* posunieme uzol nadol. Táto zmena je ilustrovaná na obrázkoch 17 a 16, kde obrázok 16 predstavuje stav pred zmenou a obrázok 17 stav po zmene. Uzly sa môžu presúvať len medzi súrodeneckými uzlami, tj. Uzly pri posune nepreskakujú cez nadradené uzly.

Ak chceme uzol zo šablóny vylúčiť, môžeme to vykonať pomocou tlačítka *Remove node* v paneli nástrojov (šípka 1 na obrázku 18).

Na príklade zobrazenom na obrázku 17 vymažeme uzol *list*. Výslená schéma je zobrazená na obrázku 19.

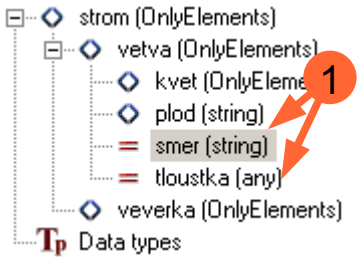


Obrázok 19: schéma po zmazaní uzlu

4.7. Pridanie uzlu typu attribute

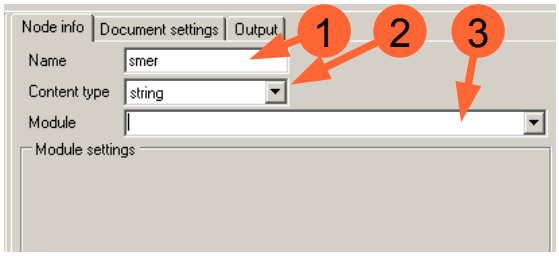
Uzol typu *attribute* špecifikuje atribút vo vygenerovanom súbore. Môže sa nachádzať len ako detský uzol elementu, alebo *mainelmentu*. Tlačítko paneli nástrojov má ikonu =. Rovnakú ikonu má aj v paneli, ktorý zobrazuje štruktúru šablóny (na obrázku 20).

8 Detské uzly elemntu sa generujú v tom poradí, ako sú nadefinované v šablóne. Pre generovanie detských uzlov v náhodom poradí sa používa uzol typu *elements* popísaný v kapitole 4.10. .



Obrázok 20: atribúty

Atribúty sa generujú tiež podľa poradia, v akom sú nadefinované v šablóne. Ich poradie rovnako ako poradie elementov môžeme meniť pomocou tlačítok a . Musia však byť v zozname ako posledné. Aplikácia nepovolí posunúť atribút pred element (alebo iný detský uzol nadradeného elementu) a naopak element za atribút.



Obrázok 21: parametre atribútu

Už pri výbere atribútu zistíme, že záložka *Node info* sa zmenila (obrázok 21). Pretože atribútu sa v rodičovskom elemente môžu vyskytovať len raz, jediné parametre, ktoré im môžeme nastaviť, sú meno (šípka 1 na obrázku 21), Content type (šípka 2 na obrázku 21) a modul (šípka 3 na obrázku 21), ktorý danému atribútu generuje dáta.

Pridaním atribútu do elementu sa v súbore šablóny pridá element *attributes*, ktorý obsauje len elementy *attribute*. Každý element *attribute* obsahuje podelement *settings*, v ktorom sa nachádzajú parametre pre modul, ktorý mu nastavíme.

Z ukážky 15 vidíme, že jediný element *kvet* má atribúty *smer* a *tloustka*. Atribút *tloustka* je typu *string* a pre generovanie dát používa modul *stringconstant*.

```
<element name="vetva" mincount="50" maxcount="50"
     … >
   <element name="kvet" …  />
    <element name="plod" type="string" … module="stringconstant">
       <setting text="jablko" />
    </element>
    <attributes>
       <attribute name="smer" type="string"
            module="stringconstant">
          <setting text="jih"/>
     </attribute>
     <attribute name="tloustka" type="integer"
          module="incrementNumber">
        <setting step="1" />
```

```
    </attribute>
  </attributes>
</element>
```

Ukážka15: dáta šablóny s definovanými atrubúrmi

Popis modulov je možné nájsť v kapitole 4.9, kde sú rozpísané všetky moduly, ich parametre a typy, ktoré môžu generovať.

Príklad vygenerovaného súboru je v ukážke 16.

```
<?xml version="1.0" encoding="UTF-8" ?>
<strom>
   <vetva smer="jih" tloustka="0">
      <kvet/>
      ...
      <kvet/>
      <plod>jablko</plod>
   </vetva>
   <vetva smer="jih" tloustka="1">
      <kvet/>
      ...
      <kvet/>
      <plod>jablko</plod>
   </vetva>
...
```

Ukážka 16: vygenerovaný súbor podľa šablóny z ukážky 15

4.8. Pridanie nového dátového typu

Skôr ako moduly, si musíme popísať význam užívateľsky definovaných dátových typov, pretože niektoré moduly ich akceptujú a iné nie.

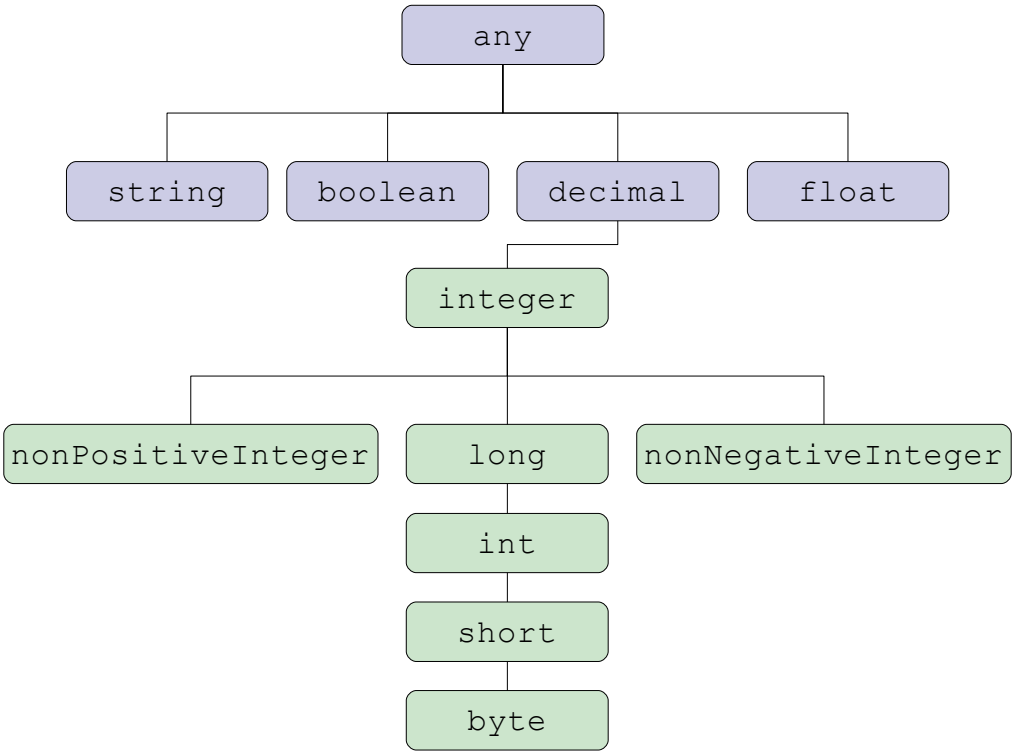
V prípade, že nám nestačí východzia sada dátových týpov, ktoré poskytuje naša aplikácia, môžeme si nadefinovať vlastný dátový typ. Dátové typy sú hierarchicky rovnako usporiadané, ako určuje XML Schema [5]. Z nej sú ale použité len niektoré. Všetky východzie typy, ktoré aplikácia poskytuje sú zobrazené na obrázku 22, kde sú hierarchicky usporiadané.

Na obrázku 22 sú farebne odlíšené dátové typy. Modrá farba predstavuje základné dátové typy a zelená dátové typy, ktoré sú zdedené pomocou obmedzení[10].

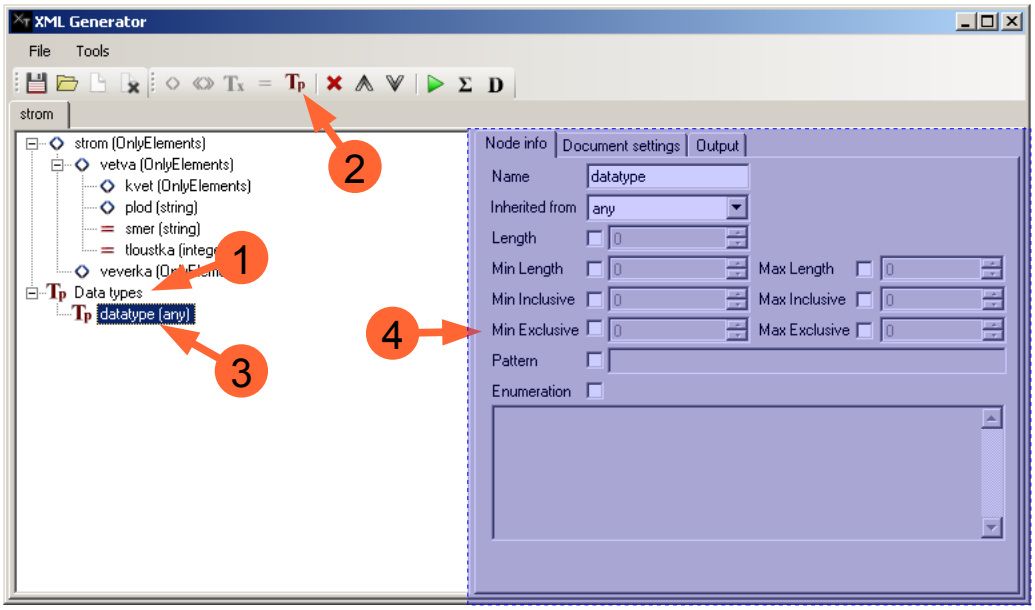
Vlastný dátový typ pridáme kliknutím na uzol *Data types*, alebo už vytvorený dátový typ (šípka 1 a 3 na obrázku 23), čím povolíme tlačítko  *New data type* v paneli nástrojov. Následným kliknutím na toto tlačítko pridáme nový dátový typ (šípka 3 na obrázku 23). Novo vytvorený dátový typ sa automaticky vyberie[9]. V záložke *Node info* nastavujeme jeho parametre, resp. obmedzenia[10] (šípka 4 na obrázku 23).

9 Ako by sme na neho sami klikli

10 Z angl. *restriction*, ktoré sa používa v XML Schema

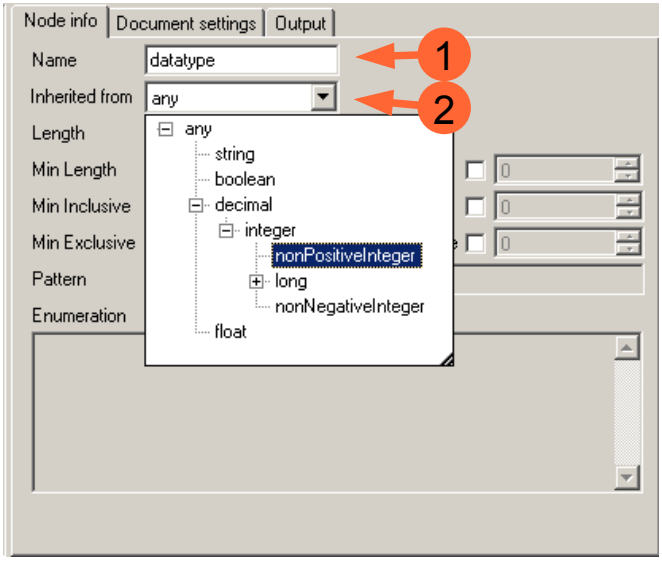


Obrázok 22: východzie dátové typy



Obrázok 23: pridanie dátového typu

Základnými parametrami, ktoré nastavujeme vlastnému dátovému typu je meno (šípka 1 na obrázku 24) a typ (šípka 2 na obrázku 24) , z ktorého dedí. Ako je vidieť z obrázku 24, dediť môžeme len z východzích dátových typov. Aplikácia tak predchádza komplikáciám s násobným dedením, pretože pri generovaní poskytuje modulu práve definíciu dátového typu, u ktorého nie je zaručené, že bude vedieť pracovať so zdedenými typmi do x-tej úrovne.



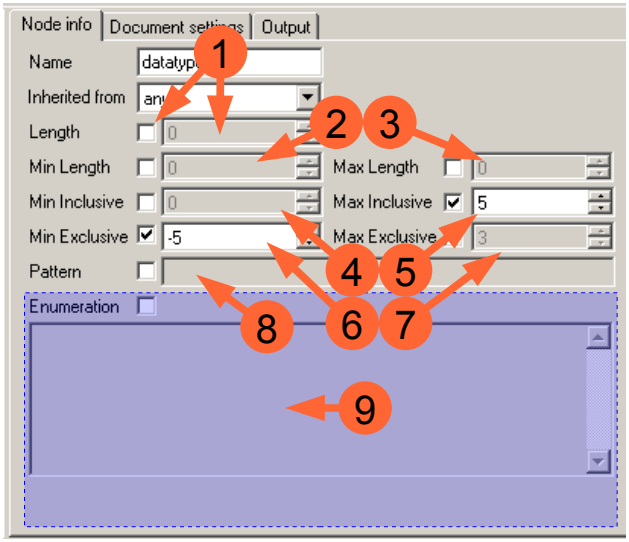
Obrázok 24: výber dátového typu pre denenie a nastavenie mena dátového typu

Na obrázku 25 vidíme, že každé obmedzenie má okrem komponenty pre svoju editáciu aj checkbox. Tým nastavujeme, aby dané obmedzenie bolo akceptované. Podľa šípok na obrázku 25 si popíšeme jednotlivá obmedzenia:

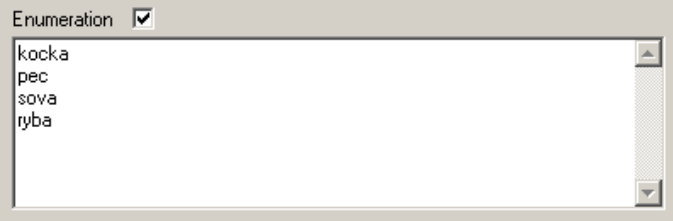
1. Length – nastavujeme ním presný počet znakov, ktoré majú mať dáta tohto typu
2. Min Length – minimálny počet znakov
3. Max Length – maximálny počet znakov
4. Min Inclusive – obmedzenie uplatnené hlavne v číselných dátových typoch. Je to spodná hranica, kde číslo musí byť väčšie alebo rovné tomuto obmedzeniu.
5. Max Inclusive – podobne ako Min Inclusive, ale je to horná hranica, kde číslo musí byť menšie alebo rovné tomuto obmedzeniu
6. Min Exclusive – podobne ako Min Inclusive s tým rozdielom, že číslo musí byť ostro väčšie ako toto obmedzenie.
7. Max Exclusive – podobne ako Min Exclusive, ale číslo musí byť ostro menšie ako toto obmedzenie.

8. Pattern – zadáva sa ako regulárny výraz (viď kapitola 2.8). a z hľadiska generovania je toto obmedzenie najzaujímavejšie, pretože sa s ním dá nadefinovať najviac typov, s ktorými sa pri práci a dátami najviac stretávame. Dajú sa ním nedefinovať emaily, PSČ, telefónne čísla atď.

9. Enumeration – je zoznam reťazcov, kde dáta tohto typu môžu obsahovať len jeden z jeho reťazcov. V komponente sa zoznam definuje tak, že každý reťazec je napísaný v novom riadku. Príklad je zobrazený obrázku 26.



Obrázok 25: parametre dátového typu



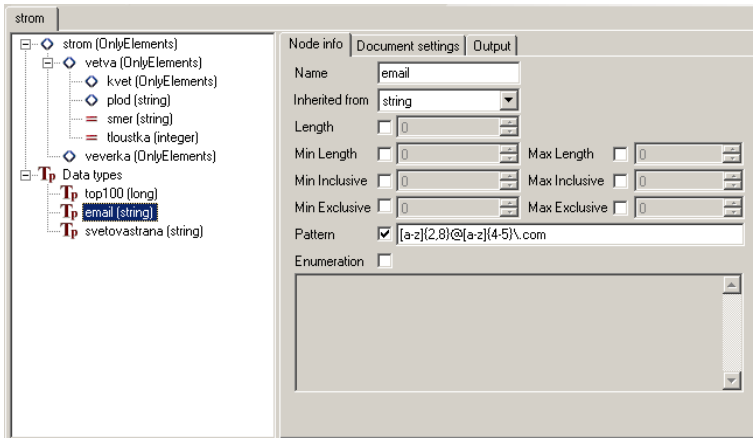
Obrázok 26: príklad obmedzenia Enumeration

Jednotlivé obmedzenia je možné kombinovať. Aplikácia pri zistení kombinácie, ktorá sa ťažko spĺňa, zobrazí do Log Formu (obrázok 7 v kapitole 4.4). Kombinácie, ktoré aplikácia vyhodnocuje ako ľahko splnitelné, sú:

- Lengh, Min Lengh, Max Length
- Min Inclusive, Max Inclusive, Min Exclusive, Max Exclusive

Všetky ostatné kombinácie (skladajúce sa z aspoň dvoch obmedzení), aplikácia vyhodnocuje ako ťažko splniteľné.

V súbore šablóny sú dátové typy uložené v elementoch *datatype* pod elementom *datatypes*, ktorý je detským elementom *xdoc*. Pre predstavu, ukážka 17 odpovedá dátovým typom z obrázku 27 .



Obrázok 27: príklad nadefinovaných dátových tpov

```
<xdoc>
  <settings>
      …
  </settings>
  <mainelement … >
      …
  </mainelement>
  <datatypes>
    <datatype name="top100" length="0" … pattern=""
     mininclusive="0" maxinclusive="100"
     usemininclusive="1"  parentType="long">
      <enumerations />
    </datatype>
    <datatype name="email" length="0" … maxexclusive="0"
     pattern="[a-z]{2,8}@[a-z]{4-5}\.com" usepattern="1"
     parentType="string">
      <enumerations />
    </datatype>
    <datatype name="svetovastrana" length="0 … usepattern="0"
     useenumeration="1" parentType="string">
      <enumerations>
        <enumeration>vychod</enumeration>
        <enumeration>zapad</enumeration>
        <enumeration>sever</enumeration>
        <enumeration>jih</enumeration>
      </enumerations>
    </datatype>
  </datatypes>
</xdoc>
```

Ukážka 17: definícia vlastných dátových typov

4.9. Moduly

Moduly, generujúce dáta patria medzi základ celej aplikácie. Tie majú na starosti generovanie dát pre elementy, atribúty a textové elementy[11], alebo spracovať vygenerovaný dokument.

11 Uzly typu text node popísané v kapitole 4.11.

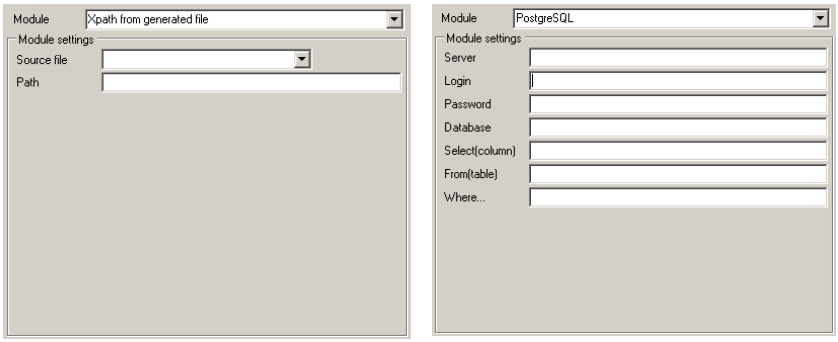
Pri inicializácií aplikácie, je prechádzaný celý adresár modules (kapitola 4.2.1), v ktorom z každej knižnice sa vyberú a prefiltrujú všetky triedu, z ktorých sa vytvoria inštancie modulov. Inštancie sa rozdelia do dvoch už spomenutých kategórií, at to moduly, ktoré generujú dáta a moduly, ktoré spracúvajú vygenerovaný súbor. V tejto kapitole si popíšeme len to, ako moduly používať.

Aplikácia si od každého modulu, ktorý generuje dáta, vyžiada informácie, v ktorých je uvedené:

- meno modulu, ktoré sa zobrazuje v GUI
- jeho id (to sa ukladá do súboru šablóny a ním aj identifikuje modul)
- typ, ktorý dokáže generovať
- obmedzenia, ktoré modul dokáže akceptovať
- jeho vstupné parametre

Z týchto informácií sa každému dátovému typu, vrátane užívateľom nadefinovaných dátových typov, určí, ktorý modul ho dokáže generovať.

Každý modul má iné vstupné parametre. Aplikácia pre daný modul vytvorí panel s komponentami, ktoré odpovedajú jednotlivým parametrom. Preto pri zmene modulu sa GUI dynamicky mení. Príkladom je obrázok 28, kde sme len zmenili modul.



Obrázok 28: dynamická zmena GUI pri zmene modulu

V súbore šablóny sa potom jednotlivé nastavenia modulu ukladajú do elementu settings, pod elementom, ktorý prislúcha uzlu v šablóne[12]. V ukážke 18 je zobrazené nastavenie atribútu *smer*, typu string, ktorý má nastavený modul *stringconstant* s jediným parametrom, a to parametrom *text*, ktorého hodnota je *jih*.

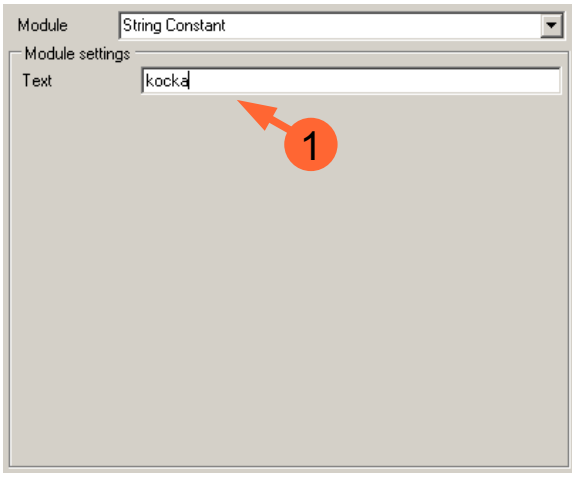
```
<attribute name="smer" type="string"
      module="stringconstant">
    <setting text="jih"/>
</attribute>
```

Ukážka 18: nastavenie modulu v súbore šablóny

12 Myslíme tým že atribút má v súbore šablóny elment attribute, element má element a pod.

4.9.1. Modul String Constant

Tento modul je najjednoduhší zo všetkých. Jediným vstupným parametrom, je *Text* (šípka 1 na obrázku 29), ktorý vracia ako vygenerovanú hodnotu. Podľa obrázku 29 bude teda jeho výstupom *kocka*. Modul neakceptuje žiadne obmedzenia a typ, ktorý generuje je string.



Obrázok 29: modul String Constant

Ukážka 19 zobrazuje jeho nastavenie v súbore šablóny a ukážka 20, čo je jeho výstupom.

```
  <attribute name="stringconstantattribute"
             module="stringconstant">
    <setting text="kocka" />
  </attribute>
```

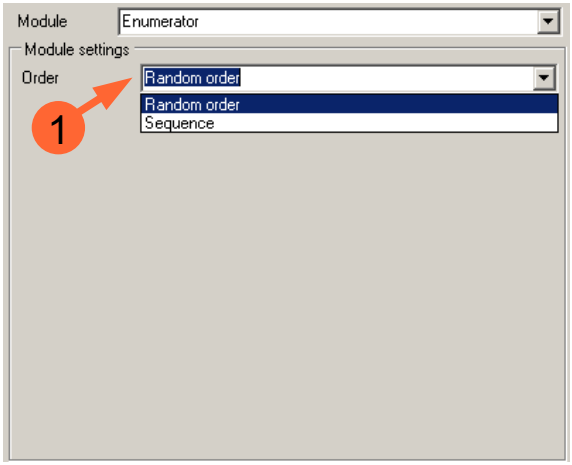
Ukážka 19: nastavenie modulu String Constant v súbore šablóny

```
<example stringconstantattribute="kocka"/>
<example stringconstantattribute="kocka"/>
<example stringconstantattribute="kocka"/>
```

Ukážka 20: príklad vygenerovaných dát pomocou modulu String Constant

4.9.2. Modul Enumerator

Modul *Enumerator* je použitelný len pre užívateľom nastavené dátové typy zdedené od typu string, pretože tento modul pracuje s obmedzením *Enumeration*. Obmedzenie totiž používa ako zdroj dát. Jediným parametrom je *Order* (šípka 1 na obrázku 30), ktorý určuje, či z obmedzenia *Enumeration* má brať reťazce náhodne, alebo po poradí (v prípade, že sa dostane na koniec zoznamu, tak ho začne čítať odznova). Okrem obmedzenia *Enumeration*, modul neakceptuje žiadne iné obmedzenie. Ako už bolo spomenuté, modul je možná použiť tam, kde dátový typ string, alebo nejaký jeho predok.



Obrázok 30: Modul Enumerator

Ukážka 21 zobrazuje jeho nastavenie v súbore šablóny a ukážka 22, čo je jeho výstupom, keď dátový typ atribútu je má v obmedzení *Enumeration* nastavené reťazce vychod, zapad, sever, jih :

```
<attribute name="enumeratorattribute" module="enumenator">
    <setting order="random" />
</attribute>
```

Ukážka 21: nastavenie modulu Enumeratort v súbore šablóny

```
<example enumeratorattribute="jih"/>
<example enumeratorattribute="sever"/>
<example enumeratorattribute="vychod"/>
```

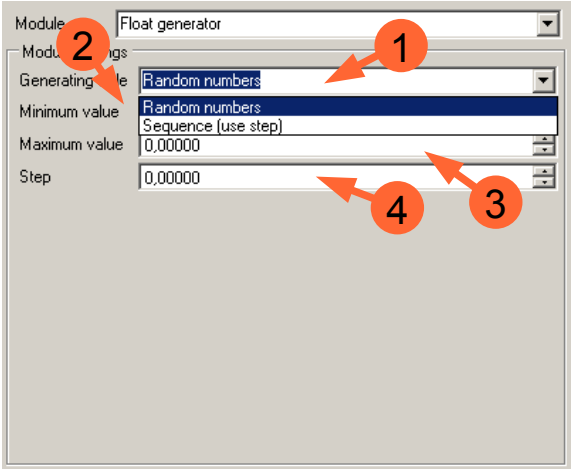
Ukážka 22: príklad vygenerovaných dát pomocou modulu Enumerator

4.9.3. Modul Float generator

Moldul *Float generator* pracuje s dátovým typom *float* a neakceptuje žiadne obmedzenia. Jeho parametre sú podľa šípok na obrázku 31 :

1. Generating style – môže nadobúdať hodnoty :
    - Random numbers (v súbore šablóny *random*), kde čísla budú v rozmedzí Minimum value a Maximum value generovane náhodne
    - Sequence (v súbore šablóny *sequence*), kde čísla budú generované postupne od Minimum value, s rozdielom Step, a pokiaľ presiahnu hodnotu Maximum value, budú generované odznova od Minimum value.
2. Minimum value – dolná hranica pre vygenerované čísla
3. Maximum value – horná hranica pre vygenerované čísla

4. Step – tento parameter sa využíva len vtedy, ak Generating style má hodnotu Sequence. Čísla sa generujú pričítaním hodnoty Step k naposledy vygenerovanému číslu.



Obrázok 31: modul Float generator

Ukážka 23 zobrazuje jeho nastavenie v súbore šablóny a ukážka 24, čo je jeho výstupom.

```
<attribute name="floatgeneratorattribute"
           module="floatGenerator">
    <setting order="random" minnum="0" maxnum="100,00000"
           step="1" />
  </attribute>
```

Ukážka 23: nastavenie modulu Float generator v súbore šablóny

```
<example floatgeneratorattribute="40,114580811986"/>
<example floatgeneratorattribute="32,1410083827288"/>
<example floatgeneratorattribute="44,4689030034788"/>
```

Ukážka 24: príklad vygenerovaných dát pomocou modulu Float generator

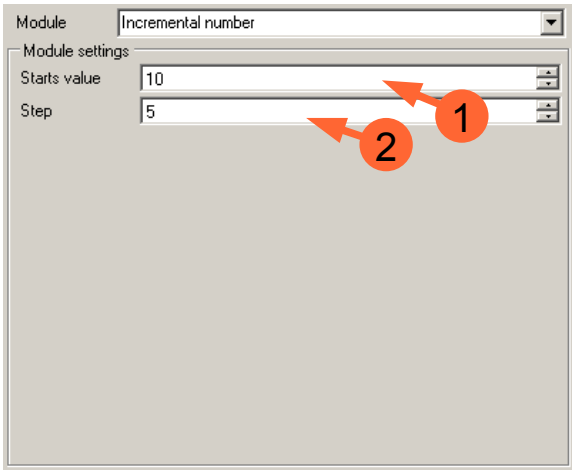
4.9.4. Modul Incremental number

Modul *Incremental number* pracuje s dátovým typom *long* a akceptuje obmedzenia Length, Min Length, Max Length, Min Inclusive, Max Inclusive, Min Exclusive a Max Exclusive. Nedokáže ale pracovať s ich kombináciami. V prípade kombinácií, uprednostňuje Min/Max Inclusive/Exclusive a až potom Min/Max Length.

Poľa obrázku 32 vidíme, že má dva parametre (podľa šípok) :

1. Startvalue – je to prvé číslo, od ktorého generuje dáta
2. Step - k naposledy vygenerovanému číslu pripočíta hodnotu Step.

V prípade, že generátor prekročí niektoré obmedzenie, začne znovu generovať čísla od Start value. V prípade, že start value je menšie ako niektoré obmedzenie, nastaví sa na start value na najmenšie možné číslo.



Obrázok 32: Modul Incremental number

Ukážka 25 zobrazuje jeho nastavenie v súbore šablóny a ukážka 26, čo je jeho výstupom.

```
<attribute name="incrementalnumberattribute"
           module="incrementNumber">
    <setting step="5" startnum="10" />
</attribute>
```

Ukážka 25: nastavenie modulu Incremental number v súbore šablóny

```
<example incrementalnumberattribute="25"/>
<example incrementalnumberattribute="30"/>
<example incrementalnumberattribute="35"/>
<example incrementalnumberattribute="40"/>
```

Ukážka 26: príklad vygenerovaných dát pomocou modulu Incremental number

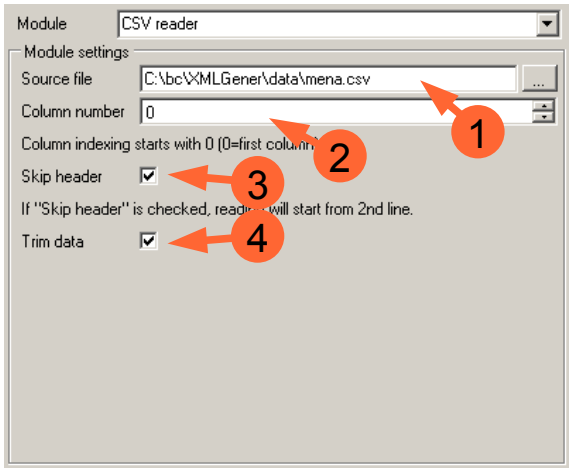
4.9.5. Modul CSV Reader

Modul *CSV Reader* pracuje s dátovým typom string a neakceptuje žiadne obmedzenia. Jeho hlavným parametrom je cesta k CSV súboru[13] (šípka 1 na obrázku 33), ktorý otvorí, nahrá si ho do pamäte (kôli rýchlosti) a postupne vracia dáta z stĺpca s indexom *Column number* (šípka 2 na obrázku 33), kde prvý stĺpec má index 0. Ďalšími parametrami je :

- *Skip header* (šípka 3 na obrázku 33) - v prípade, že je tento checkbox zaškrtnutý, generátor preskakuje prvý riadok v načítanom súbore

13 CSV súbor popísaný v kapitole 2.12.

- *Trim* (šípka 4 na obrázku 33) - v prípade, oreže sprava a zľava načítaný reťazec o neviditeľné znaky a medzery a v prípade, že orezaný výsledný reťazec je prázdny, zahodí ho a načíta ďalší v poradí.



Obrázok 33: Modul CSV Reader

Ukážka 27 zobrazuje jeho nastavenie v súbore šablóny a ukážka 28, čo je jeho výstupom.

```
<attribute name="csvreaderattribute" module="csvreader">
    <setting text="" source="C:\bc\XMLGener\data\mena.csv"
            column="0" skip="1" trim="1" />
</attribute>
```

Ukážka 27: nastavenie modulu Incremental number v súbore šablóny

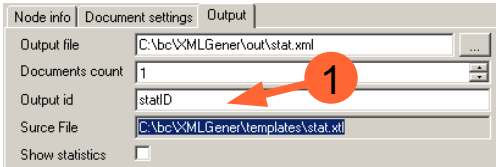
```
<example csvreaderattribute="Robert"/>
<example csvreaderattribute="Vanda"/>
<example csvreaderattribute="Livia"/>
<example csvreaderattribute="Lenka"/>
```

Ukážka 28: príklad vygenerovaných dát pomocou modulu Incremental number

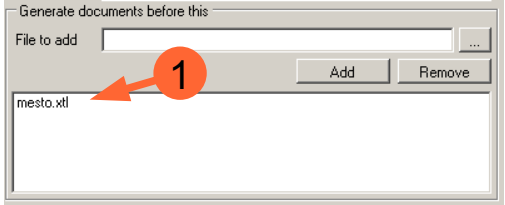
4.9.6. Modul XPath from generated file

Modul *XPath from generated file* pracuje s dátovým typom *string* a neakceptuje žiadne obmedzenia.

Používa sa len v prípade, že do aktuálne editovanej šablóny pridáme šablónu, ktorá sa má vygenerovať ešte pred tým, ako začne generovanie aktuálne editovanej šablóny. Príklad: Naša šablóna sa volá *mytest* a šablóna, ktorá sa má predgenerovať, sa volá *stat*. Najrv otvoríme šablónu *stat*, kde v záložke Output nastavíme Output Id na *statID* (šípka 1 na obrázku 34). Šablónu uložíme a otvoríme si šablónu *mytest*. V záložke Document Settings ju pridáme do zoznamu predgenerovaných šablón (šípka 1 obrázok 35) tak, že najprv klikneme na tlačítko „…“ pre výber predgenerovaných šablón a potom na tlačítko *Add*.

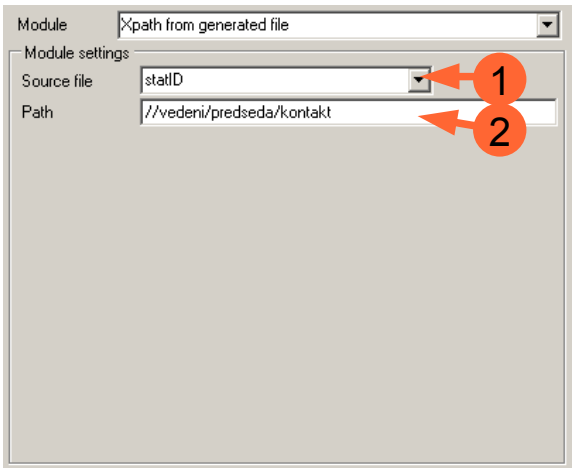


Obrázok 34: nastavenie Output id



Obrázok 35: predgenerované šablóny

Následne sa (musíme aspoň prekliknúť na uzol kvôli obnoveniu) zobrazí statID v ponuke (šípka 1, obrázok 36). Musíme predpokladať, že vygenerovaný súbor je XML Dokumentom. Potom zdrojom dát pre modul *XPath from generated file*, je zoznam uzlov, ktoré vyberie pomocou XPath.



Obrázok 36: Modul XPath From generated file

Ukážka 29 zobrazuje jeho nastavenie v súbore šablóny a ukážka 30, čo je jeho výstupom.

```
<attribute name="xpathfromgeneratedfileattribute"
           module="xpahtFromGenerated">
   <setting xpath="//vedeni/predseda/kontakt" />
 </attribute>
```

Ukážka 29: nastavenie modulu XPath From generated file v súbore šablóny

```
<example xpathfromgeneratedfileattribute="qizp87@pgtki.com"/>
<example xpathfromgeneratedfileattribute="iitbyd60@uxdrx.com"/>
<example xpathfromgeneratedfileattribute="nrek@gicyq.com"/>
<example xpathfromgeneratedfileattribute="pyfr@dumuv.com"/>
```

Ukážka 30: príklad vygenerovaných dát modulom XPath From generated file

4.9.7. Modul PostgreSQL

Modul *PostgreSQL* pracuje s dátovým typom string a neakceptuje žiadne obmedzenia. Modul pri inicializácii sa pripojí k databázi na *PostgreSQL* servri (viď kapitola 2.10), vykoná príkaz pre výber dát z tabuľky, výsledok si uloží do medzipamäte a ukončí spojenie. Pri generovaní modul vracia dáta z medzipamäte cyklicky, to znamená že po poslednom riadku sa vráti na prvý riadok.

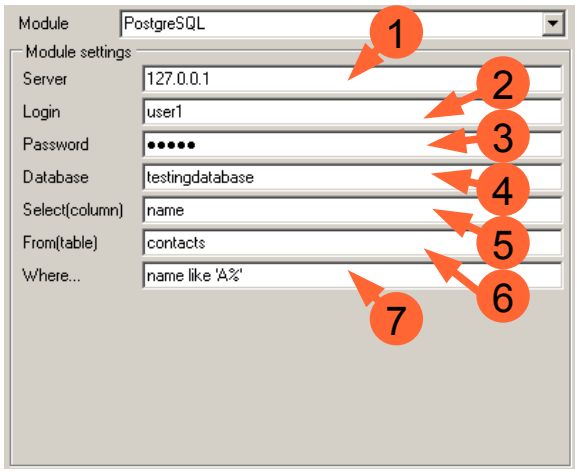
Databáza, ku ktorej sa modul pripája je určená parametrami:

- adresa, alebo meno servera (šípka 1 na obrázku 37).
- meno užívateľa (šípka 2 na obrázku 37), ktorý má právo prihlásiť sa do databáze

- heslo (šípka 3 na obrázku 37), s ktorým sa užívateľ prihlasuje do databáze
- meno databáze (šípka 4 na obrázku 37)

Príkaz pre výber dát (z tabuľky) sa poskladá z parametrov:

- *Select (column)* (šípka 5 na obrázku 37) - stĺpec, alebo výraz, z ktorého budú brané dáta. Modul vyberá dáta len z prvého stĺpca a to aj v prípade, že bude zadaných viac stĺpcov.
- *From (table)* (šípka 6 na obrázku 37) - tabuľa ,resp. virtuálna tabuľka, ako napríklad *view*, viacero tabuliek alebo funkcia vracajúca tabuľku
- *Where* (šípka 7 na obrázku 37) - podmienka, ktorá filtruje riadky z tabuľky. Tento parameter ako jediný nie je povinný.



Obrázok 37: Modul PostgreSQL

Ukážka 31 zobrazuje poskladaný dotaz podľa parametrov z obrázku 37.

```
SELECT name FROM contacts WHERE name like 'A%';
```

Ukážka 31: príklad dotazu pre výber dát do databáze na PostgreSQL serveri

Ostatné parametre dotazu, ktoré nasledujú za *WHERE*, ako napríklad *OFFSET*, *LIMIT, GROUP BY* alebo *ORDER BY* sa píšu do parametra *Where* za podmienku. Tá v tomto prípade musí byť vyplnená, pretože poskladaný výraz by nebol platný. Príkladom je ukážka 32.

```
SELECT name FROM contacts WHERE LIMIT 100;
```

Ukážka 32: príklad neplatného dotazu

Môžeme to ale obísť použitím podmienky, ktorá je splnená vždy. Teda *true*, alebo *1=1*, čoho výsledkom je dotaz v ukážke 33 .

```
SELECT name FROM contacts WHERE true LIMIT 100;
```

Ukážka 33: príklad správne poskladaného dotazu

Ukážka 34 zobrazuje jeho nastavenie v súbore šablóny a ukážka 35, čo je jeho výstupom.

```
<attribute name="postgresqlattribute" module="pgsql">
   <setting where="name like 'A%'" table="contacts"
      col="name" database="testingdatabase" password="tajne"
      login="user1" server="127.0.0.1" />
</attribute>
```

Ukážka 34: nastavenie modulu PostgreSQL v súbore šablóny

```
<example postgresqlattribute="Adriana"/>
<example postgresqlattribute="Andrea"/>
<example postgresqlattribute="Amy"/>
<example postgresqlattribute="Alex"/>
```

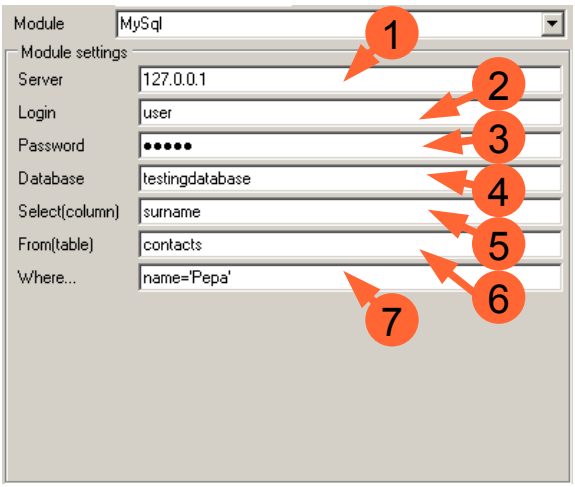
Ukážka 35: príklad vygenerovaných dát modulom PostgreSQL

4.9.8. Modul MySQL

Modul *MySQL* pracuje s dátovým typom string a neakceptuje žiadne obmedzenia. Jeho parametre a jeho spôsob generovania dát sú rovnaké, aké má modul *PostgreSQL* (kapitola 4.9.7). Rozdielom je, že sa pripája k MySQL serveru. Postupuje rovnako ako modul *PostgreSQL*, tj. pripojí sa k serveru, vykoná dotaz, výsledok uloží do medzipamäte a odpojí sa. Pri generovaní vracia dáta cyklicky, teda po poslednom zázname vracia opäť prvý záznam.

Z obrázku 38 je vidieť jeho parametre (podľa čísla šípok):

1. adresa serveru
2. meno užívateľa
3. heslo
4. meno databáze
5. stĺpec – vyberá len 1. stĺpec v prípade, že ich je zadaných viac
6. tabuľka
7. podmienka



Obrázok 38: Modul MySQL

Rovnako ako v module PostgreSQL sa ostatné parametre dotazu, ktoré nasledujú za *WHERE*, ako napríklad *OFFSET*, *LIMIT, GROUP BY* alebo *ORDER BY* píšu do parametra *Where* za podmienku. Návod je v kapitole 4.9.7.

Ukážka 36 zobrazuje jeho nastavenie v súbore šablóny a ukážka 37, čo je jeho výstupom.

```
<attribute name="mysqlattribute" module="mysql">
   <setting where="name='Pepa'" table="contacts" col="surname"
      database="testingdatabase" password="tajne" login="user"
      server="127.0.0.1" />
</attribute>
```

Ukážka 36: nastavenie modulu MySQL v súbore šablóny

```
<example mysqlattribute="Novak"/>
<example mysqlattribute="Nezval"/>
<example mysqlattribute="Krc"/>
<example mysqlattribute="Moravec"/>
```

Ukážka 37: príklad vygenerovaných dát modulom MySQL

4.9.9. Modul Pattern generator

Modul *Pattern* pracuje s dátovým typom string a akceptuje obmedzenie pattern.

Jeho parametrom (šípka 1 na obrázku 39) je regulárny výraz, ktorý je popísaný v kapitole 2.8.1. Modul ignoruje parameter v prípade, že dátový typ má nastavený *Pattern*.

Obrázok 39: Modul Pattern generator

Pre generovanie používa knižnicu popísanú v kapitole 2.8.1. Generovanie pomocou tohto modulu je o čosi pomalšie, než u ostatných modulov. To tiež ale závisí od parametra (napríklad 1000 000 reťazcov, kde parametrom bol výraz `[a-z]{3,5}@[a-z]{5}\.com` vygeneroval na našej zostave za 20 sekúnd).

Ukážka 38 zobrazuje jeho nastavenie v súbore šablóny a ukážka 39, čo je jeho výstupom.

```
<attribute name="patterngeneratorattribute"
      module="patternGenerator">
    <setting text="[0-9]{3} [0-9]{2}" />
</attribute>
```

Ukážka 38: nastavenie modulu MySQL v súbore šablóny

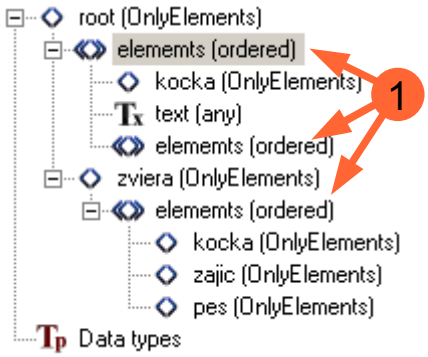
```
<example  patterngeneratorattribute="170 45"/>
<example  patterngeneratorattribute="069 64"/>
<example  patterngeneratorattribute="620 24"/>
<example  patterngeneratorattribute="594 40"/>
```

Ukážka 39: príklad vygenerovaných dát modulom MySQL

4.10. Pridanie uzlu typu elements

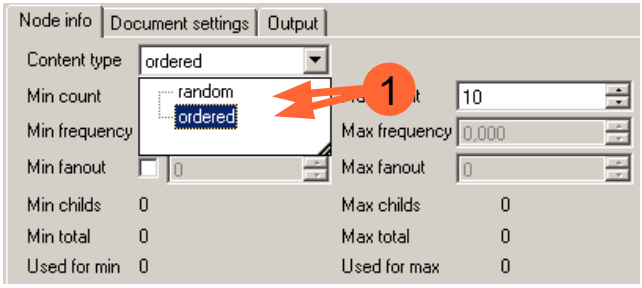
Uzol typu *elements* funguje ako akási zátvorka pre elementy. Môže obsahovať len uzly typu *element*, text *node* alebo *elements*, jeho parametrami sú *Min/Max Count/Fanout/Frequency*, ale vo vygenerovanom súbore sa zobrazujú len elementy/text vygenerovaný z jeho detských uzlov.

Elements sa môže nachádzať len ako detský uzol *elementu*, *mainelementu* alebo uzlu typu *elements*. Pridáva sa tláčítkom v paneli nástrojov. Rovnakú ikonu má aj v paneli, ktorý zobrazuje štruktúru šablóny (šípka 1 na obrázku 40).



Obrázok 40: uzol typu elements

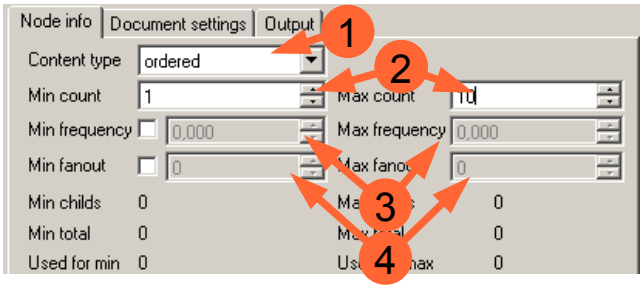
Rovnako ako uzly typu element, sa aj uzly typu elements generujú podľa poradia. Možno im preto pomocou tlačítok a meniť poradie. To však nemusí platiť pre ich potomkov. Parameter *Content Type* (šípka 1 na obrázku 41) určuje, v akom poradí sa majú detské elementy generovať. Čiže v náhodom poradí, alebo v tom, v akom sú definované.



Obrázok 41: nastavenie Content type ulzu typu elements

Podľa obrázka 42 vidíme, že až na parameter *Content type* (šípka 1 na obrázku 42), má všetky ostatné parametre zhodné s uzlom typu *element.* Jeho pramatrami teda sú (podľa čísel šípok):

1. *Content type* – určuje, v akom poradí sa budú generovať detské uzly
2. *Min/Max count* - minimálny/maximálny počet vygenerovaných uzlov, ktoré sa ale nezobrazujú vo výsledku (viď kapitola 4.6.1)
3. *Min/Max frequency* – minimálna/maximálna frekvencia výskytu (viď kapitola 4.6.2)
4. *Min/Max fanout* – počet detských uzlov (viď kapiotola 4.6.2)



Obrázok 42: parametre uzlu typu elements

Ukážka 40 zobrazuje súbor šablóny v ktorej sú použité uzly typu *elements* a ukážka 41 súbor, vygenerovaný podľa tejto šablóny.

```
<xdoc>
  <settings>
    <output module="" schemeType="dtd"
      outfile="out\elements.xml" />
    <secondoutput module="" />
  </settings>
  <mainelement name="root" mincount="1" maxcount="1">
    <elements mincount="1" maxcount="1">
      <element name="prvy" mincount="2" maxcount="2" />
      <elements mincount="5" maxcount="5" type="random">
        <element name="aaa" mincount="1" maxcount="1" />
        <element name="bbb" mincount="1" maxcount="1" />
      </elements>
    </elements>
  </mainelement>
  <datatypes />
</xdoc>
```

Ukážka 40: súbor šablóny, ktorá obsahuje uzly typu elements

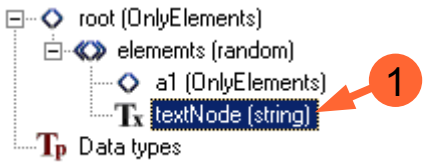
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE root SYSTEM "elements.dtd" >
<root>
   <prvy/>
   <prvy/>
   <bbb/>
   <aaa/>
   <bbb/>
   <aaa/>
   <bbb/>
   <aaa/>
   <bbb/>
   <aaa/>
   <bbb/>
   <aaa/>
</root>
```

Ukážka 41: súbor vygenerovaný podľa šablóny z ukážky 40

4.11. Pridanie uzlu typu text node

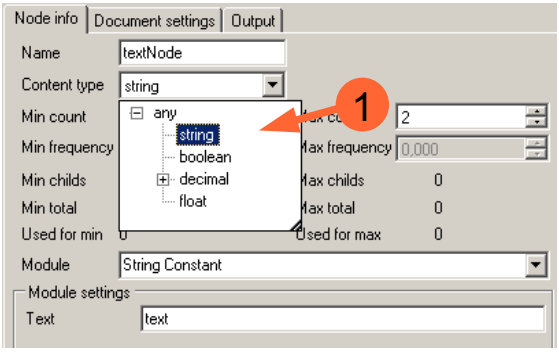
Uzol typu *text node* slúži prie potrebu generovania zmiešaného obsahuj. Myslí sa tým obsah elementu, ktorý tvorí text aj elementy. Je to v podstate uzol typu *element* bez atribútov s nastaveným dátovým typom (viď kapitola 4.6.3), z ktorého sa generuje len jeho obsah, tzn. bez ohraničujúcich značiek elementu < a >.

Text node nemá žiadne detské uzly. Jeho nadradenými uzlami môžu byť len uzly typu *element*, *mainelement* alebo *elements*. Pridáva sa pomocou tlačítka  v paneli nástrojov. Rovnakú ikonu má aj v paneli, ktorý zobrazuje štruktúru šablóny (šípka 1 na obrázku 43).



Obrázok 43: uzol typu text node

Ako je vidieť z obrázku 44, nastavenia jeho parametrov sa oproti *elementu* líšia len vo výbere *Content Type*, kde sú na výber len dátové typy (šípka 1 na obrázku 44). Jeho parametre sú popísané už v kapitole 4.6, preto sa nimi ďalej nebudeme zaoberať.



Obrázok 44: parametre uzlu typu text node

Ukážka 42 zobrazuje súbor šablóny v ktorej sú použité uzly typu *text node* a ukážka 43 súbor, vygenerovaný podľa tejto šablóny.

```
<xdoc>
  <settings>
    <output module="" schemeType="dtd"
      outfile="out\textnodeExample.xml" />
    <secondoutput module="" />
  </settings>
  <mainelement name="root" mincount="1" maxcount="1">
    <elements mincount="4" maxcount="4" type="random">
      <element name="a1" mincount="2" maxcount="2" />
      <textnode name="textNode" type="string"
          module="stringconstant" mincount="2" maxcount="2">
        <setting text="text" />
      </textnode>
    </elements>
  </mainelement>
  <datatypes />
</xdoc>
```

Ukážka 42: súbor šablóny, ktorá obsahuje uzly typu elements

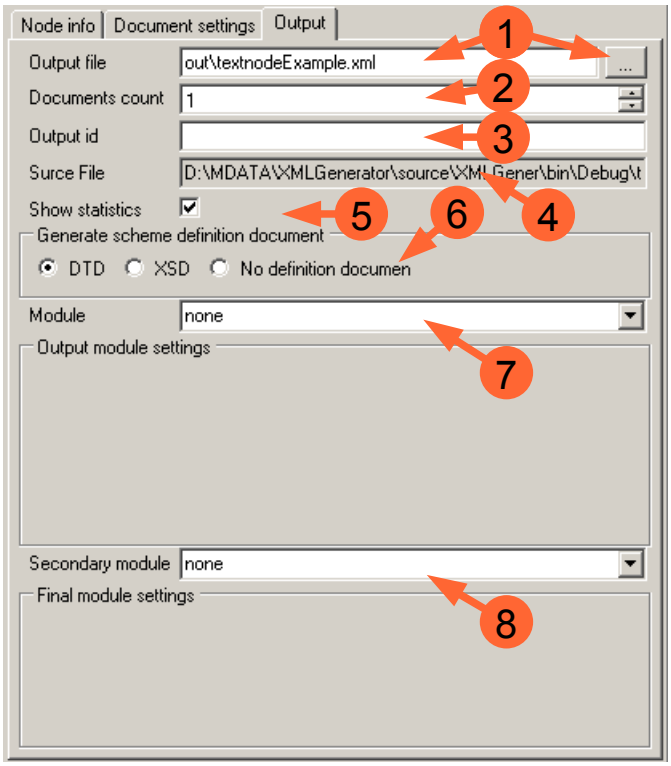
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE root SYSTEM "textnodeExample.dtd" >
```

```
<root>
   <a1/>
   <a1/>
text
text
text
   <a1/>
text
   <a1/>
text
   <a1/>
   <a1/>
text
   <a1/>
text
text
   <a1/>
</root>
```

Ukážka 43: súbor vygenerovaný podľa šablóny z ukážky 42

4.12. Parametre výstupu

Doposiaľ bolo popísané, ako vygenerovať XML dokument. Všetky parametre, ktoré sa týkajú vygenerovaného dokumentu sa nastavujú v záložke *Output* (obrázok 45).



Obrázok 45: záložka output

Podľa šípok na obrázku 45 sú parametrami dokumentu:

1. *Output file* – je meno výstupného súboru. XML Generátor podporuje zápis relatívnej (príklad je na obrázku) aj absolútnej cesty. Cesta môže byť zadaná jej vpísaním do textového poľa, alebo pomocou tlačítka „...“, ktoré zobrazí dialógové okno.

2. *Documents count* – je číslo určujúce počet vygenerovaných dokumentov. V prípade, že dokument je len jeden, meno vygenerovaného dokumentu bude zhodné s hodnotou parametra *Output file*. Inak XML Generátor dosadzuje číslo určujúce poradie dokumentu pred príponu s ohľadom na celkový počet vygenerovaných elmenetov (dosadzuje nuly pred číslom). Keb počert vygenerovaných dokumentov bol 100, podľa parametru z obrázku 45 by druhý vygenerovaný dokument bol uložený v súbore `out\textNodeExample002.xml`.

3. *Output id* - je to identifikátor prvého vygenerovaného súboru. V prípade, že šablóna vygeneruje viac XML dokumentov, identifikuje sa ním len prvý dokument. Jeho použite je určené pre moduly generujúce dáta, ktoré z vygenerovaného súboru potrebujú získať nejaké údaje. Vzor použitia je popísaný v kapitole 4.9.6.

4. Komponenta *Source file* slúži len pre zobrazenie úplnej cesty k súboru šablóny.

5. Parameter *Show statistics* určuje, či sa má zobraziť štatistika o vygenerovaných súboroch. Generátor vždy k vygenerovaným dokumentom vytvára štatistickú tabuľku, ktorá je uložená v jednom súbore (obsahuje dáta zo všetkých vygenerovaných súboroch) s rovnakým názvom ako parameter *Output file*, len miesto prípony *xml* má príponu *csv*. Je to CSV súbor (viď kapitola 2.12), ktorého stĺpcami sú:

   index dokumentu, meno elementu, počet elementov v hĺbke 1, počet elementov v hĺbke 2 atď., ako je uvedené v ukážke 44.

```
document index; element name/type;Depth 0;Depth 1;Depth 2
1;root;1;0;0
1;a;0;1031;0
1;y1;0;0;103100
1;y2;0;0;103001
1;y3;0;0;103000
```

Ukážka 44: štatistika k vygenerovaným súborom

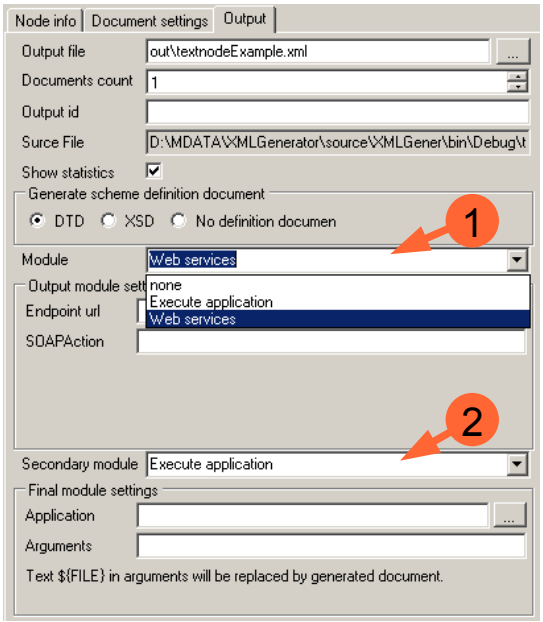
6. Parameter *Generate scheme definition document* určuje, aký typ schémy sa má k vygenerovaným XML dokumentom vytvoriť. Súbor obsahujúci schému sa generuje nie z vygenerovaného XML dokumentu, ale zo šablóny. V prípade, že je vybraná XSD alebo DTD schéma, XML Generátor vloží odkaz na súbor, v ktorom je vygenerovaná schéma, do vygenerovaného XML dokumentu tak, aby bol validný,

7. Parameter *Module* určuje primárny modul výstupu, ktorý sa spúšťa vždy a nad každým vygenerovaným XML dokumentom. XML Generátor obsahuje tri základné moduly. Prvým z nich je None, druhým Execute Application a tretím je Web services. Všetky sú popísané v kapitole 4.13. Tie dostanú na vstup vygenerovaný XML súbor a meno súboru, do ktorého majú zapísať výsledok svoj výstup (použitie je uvedené v kapitole 4.9.6).

8. Parameter *Secondary module* určuje sekundárny modul výstupu, ktorý by mal vygenerovaný súbor (resp. súbor spracovaný primárnym modulom) len zobraziť. Ten sa spúšťa len súbormi, ktoré sú vygenerované z „aktuálne generovanej“ šablóny. To znamená, že sa nespúšťa nad pedgenerovanými šablónami[14].

4.13. Moduly pre zpracovanie vygenerovaných dát

Nastavenia výstupných modulov sú umiestnené v záložke *Output* (viď kapitola 4.12). Po vygenerovaní XML dokumentu zavolá generátor primárny výstupný modul (šípka 1 na obrázku 46), ktorý ho spracuje a výsledok uloží do súboru, ktorého meno dostane od generátora, resp. meno vzniknuté podľa parametra *Output file* (šípka 1 na obrátku 45 v kapitole 4.12). Generátor najprv vygeneruje dočasný súbor (má v názve *_tmp*, ako napríklad *mytest_tmp.xml*), ktorý predá modulu. Modul má následne spracovať tento súbor a výstup zapísať do súbora, ktorého meno určuje tiež XML generátor (určuje ho podľa parametra *Output file*). Tento súbor je potom braný ako výstupný súbor vygenerovaný zo šablóny.



Obrázok 46: nastavenie výstupných modulov

14 Myslia sa tým vygenerované XML dokumenty zo šablón, ktoré sa majú generovať ešte pre tým, ako započne generovanie aktuálnej šablóny

Sekundárny výstupný modul (šípka 1 na obrázku 46) by mal vygenerovaný súbor len zobraziť (nie je to podmienkou), pretože ten na vstupe dostane len meno vygenerovaného súboru.

V kapitolách 4.13.1 až 4.13.3 sú popísané východzie výstupné moduly, ktoré XML Generátor ponúka.

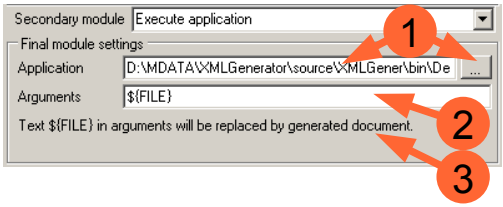
4.13.1. Výstupný modul None

Modul *None* je virtuálny, ktorý ponúka XML Generator. Pri výbere tohto modulu totiž XML Generator nevytvára dočasný súbor, ale dáta generuje priamo do výstupného súboru. V prípade, že je zvolený ako sekundárny modul, s vygenerovaným súborom nič nerobí.

4.13.2. Výstupný modul Execute Application

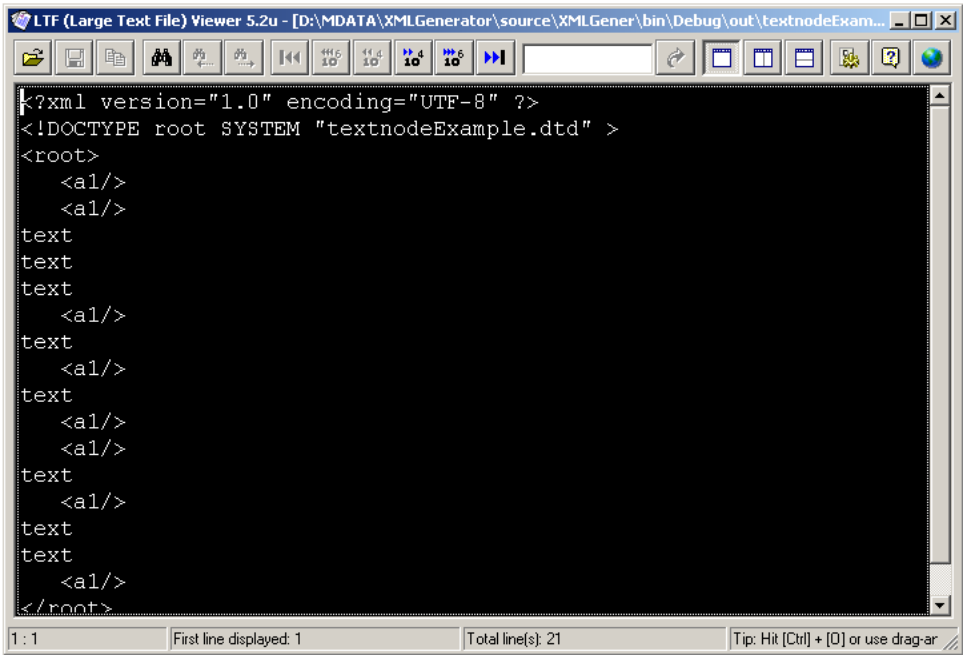
Modul je určený pre výber sekundárneho modulu. Jeho parametrami sú (podľa šípok na obrázku 47):

1. *Application* – je to cesta k spúšťateľnému súboru. V prípade, že nie je vyplnený, použije aplikáciu *explorer.exe*.
2. *Arguments* – čo sú argumenty aplikácie. Ako je vidieť z obrázku 47 (šípka 3), modul informuje o spôsobe použitia. V parametri *Arguments* nahradzuje reťazec `${FILE}` menom vygenerovaného súbora.



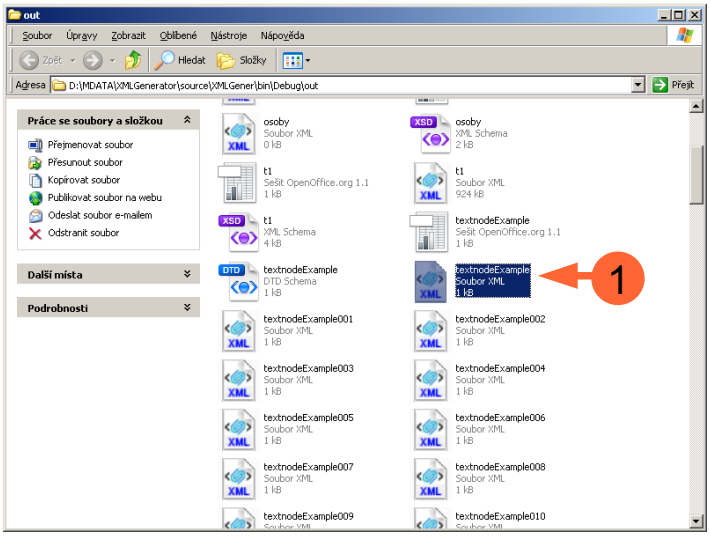
Obrázok 47: nastavenie výstupného modulu Execute Application

Na obrázku 47 je nastavený parameter *Application* na aplikáciu *LTFViewr*, ktorá je umiestná v adresári *Tools* (viď kapitola 4.2.1). Tá dokáže zobrazovať súbory väčšie ako operačná pamäť. Jej nevýhodou ale je, že nedokáže správane zobraziť súbory v kódovaní UTF-8. Na obrázku xx je znázornené, ako aplikácia zobrazuje vygenerovaný súbor (aplikácia je automaticky spustená z modulu).



Obrázok 48: aplikácia LTFViewr

V prípade, ak parameter nie je vyplnený, modul spúšťa aplikáciu *exporer.exe* s parametrami */select menosúboru*. Následkom toho je otvorenie okna a v ktorom je vygenerovaný dokument vybraný (šípka 1 na obrázku 49).



Obrázok 49: vygenerovaný súbor zobrazený v okne

Jeho nastavenie, podľa obrázku 47, ako sekundárneho modulu, zobrazuje ukážka 45.

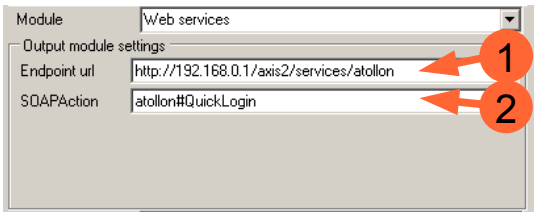
```
<xdoc>
  <settings>
    <output schemeType="dtd" outfile="out\lidi.xml" module="" />
    <secondoutput module="executeapplication">
      <setting
application="D:\MDATA\XMLGenerator\source\XMLGener\bin\Debug\tool
```

```
s\LTFViewr\LTFViewr5u.exe" arguments="${FILE}" />
    </secondoutput>
  </settings>
…
</xdoc>
```

Ukážka 45: Nastavenie modulu Execute application ako sekundárneho modulu

4.13.3. Výstupný modul Web services

Module *Web services* je určený pre použite ako primárny modul. Ten sa pripája na adresu *Endpoint url* (šípka 1 na obrázku 50), na ktorej sú poskytované webová služba *SOAP Action* (šípka 2 na obrázku 50).



Obrázok 50: nastavenie výstupného modulu Web services

Do tela dotazu (do elementu `<body>`), ktorý odosiela na server, vkladá vygenerovaný dočasný súbor z XML Generatora (viď úvod kapitoly 4.13) a odpoveď zo servera ukladá do výstupného súboru, ktorého meno mu určuje XML Generator. Pri tomto module musíme dbať na to, aby k nemu nebola vygenerovaná žiadna schéma. Tá by totiž bola vložená do dočasného súboru, čo môže spôsobovať odmietnutie dotazu zo strany servera.

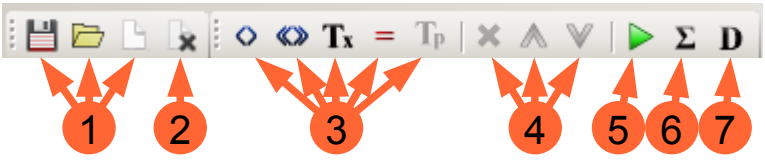
Je ho nastavenie v súbore šablóny, ako primárneho výstupného modulu, je zobrazené v ukážke 46.

```
 <xdoc>
  <settings>
    <output schemeType="none" outfile="out\Login.xml"
module="webservice" doccount="1">
      <setting url="http://192.168.0.1/axis2/services/atollon"
action="atollon#QuickLogin" />
    </output>
    <pregenerate>
    </pregenerate>
    <secondoutput module="" />
  </settings>
…
</xdoc>
```

Ukážka 46: Nastavenie modulu Web services ako sekundárneho modulu

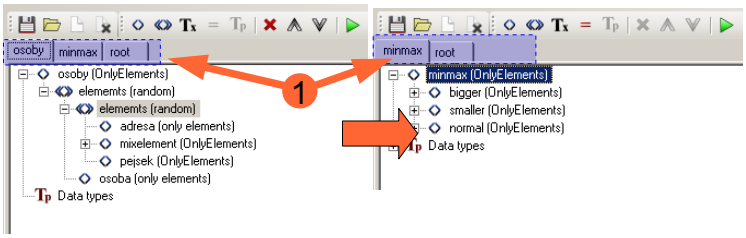
4.14. Panel nástrojov

Na obrázku 51 je zobrazený celý ovládací panel. Funkcia tlačítok, označených šípkami 1, 3, 4 a 5 boli popísané v kapitolách 4.6 až 4.11.



Obrázok 51: panel nástrojov

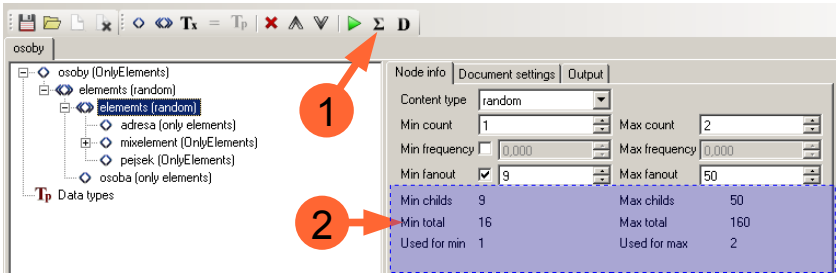
Tlačítko *Close document* (šípka 2 na obrázku 51) zatvára aktuálne editovanú šablónu. Rozdiel je vidieť na obrázku 52 (šípka 1).



Obrázok 52: zatváranie dokumentu

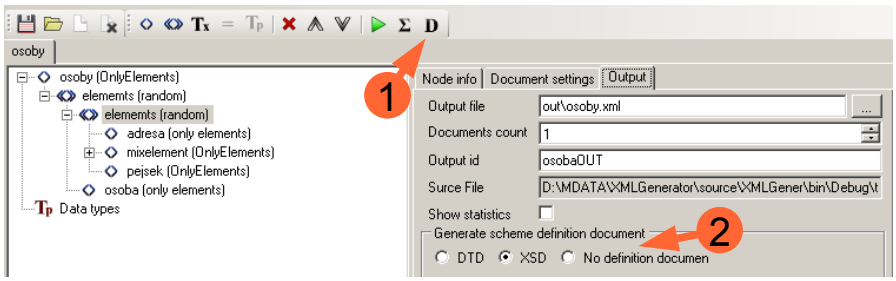
Tlačítko *Calculate Ranges* (šípka 6 na obrázku 51) je určené pre ladenie generovania. Pred generovaním sa z nastavených parametrov *Min/Max count*, *Min/Max frequency* a *Min/Max fanout* prepočítava pomocou funkcie, ktorú toto tlačítko spúšťa, horné a dolné hranice jednotlivých elementov (šípka 2 na obrázku 53). Význam jednotlivých čísel je :

- *Min childs*, *Max childs* – horná a dolná hranica počtu detských uzlov (uzlov typu *element*, *elemnts* a *text node*)
- *Min total*, *Max total* - horná a dolná hranica počtu uzlov (typu *element*, *elemnts* a *text node*) v celom strome vrátane vybraného uzlu
- *Used for min*, *Used for max* – je horná a dolná hranica výskytu vybraného uzla v nadradenom uzli



Obrázok 53: prepočítavanie

Tlačítko *Generate scheme* (šípka 1 na obrázku 54) generuje schému podľa šablóny bez nutnosti generovanie XML dokumentu. Typ schémy sa určuje v záložke *Output* výberom jednej z možnosti: *XSD* alebo *DTD* (šípka 1 na obrázku 54). V prípade výberu *No definition document*, sa nevygeneruje žiadny dokument.



Obrázok 54: tlačítko pre generovanie schémy

5. Programátorská dokumentácia

5.1. Vývojové prostredie a použité knižnice

Aplikácia XML Generator bola vytvorená pre operačný systém Windows XP a vyššie v prostredí Microsoft Visual C# 2008 Express Edition na platforme .Net Framework 3.5. Prostredie plne podporuje prácu s XML dokumentami a formulárovými komponentami. Potreby XML Generatora sú kôli rozmanitosti modulov presahujú podporu štandardných knižníc (ako komunikácia s PostgreSQL a MySQL serverom, alebo generovanie reťazcov z regulárneho výrazu), čo bolo vyriešené importom externých knižníc, ktoré tieto potreby pokrývajú. Všetky knižnice a aplikácie použité v projekte sú voľne dostupné na internete.

Následujúci zoznam popisuje mená knižníc použitých v aplikácii a ich význam:

- Npgsql.dll – knižnica pre komunikáciu s PostgreSQL serverom [20]
- Mono.Security.dll, policy.2.0.Npgsql.dll – knižnice vyžadované knižnicou Npgsql.dll pre pripojenie sa cez SSL port [20]
- MySql.Data.dll – knižnica pre pripojenie sa k MySQL serveru [21]
- RegexDataGenerator.dll – knižnica, ktorá na podľa regulárneho výrazu vygeneruje reťazec [11]

5.2. Štruktúra aplikácie

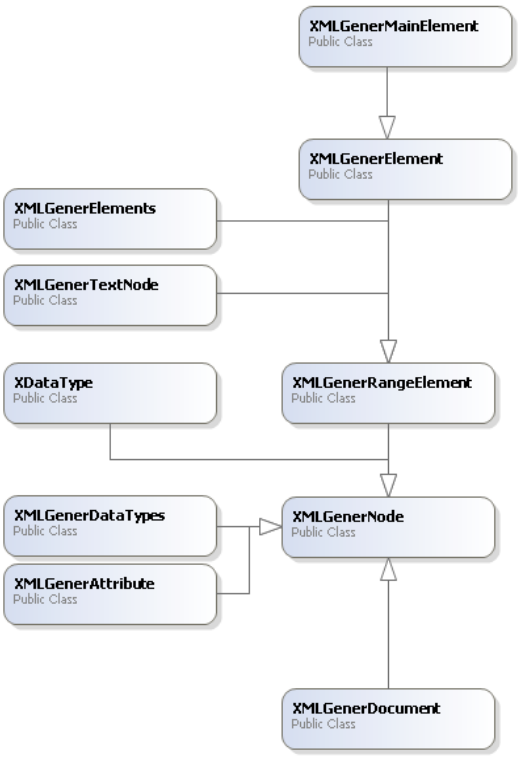
Celé riešenie implementácie pozostáva z 3 projektov:

- jadro, ktoré sa nachádza v knižnici/projekte XMLGenerLib.dll a je nezávislé od zbytku
- aplikácia s GUI rozhraním, ktoré sa nachádza v spustitelnom súbore XMLGener.exe, ktorá je ale závislá na jadre, nie však na moduloch
- základné moduly, ktoré sa nachádzajú v knižnici/projekte XbasicModules.dll je závislé na jadre

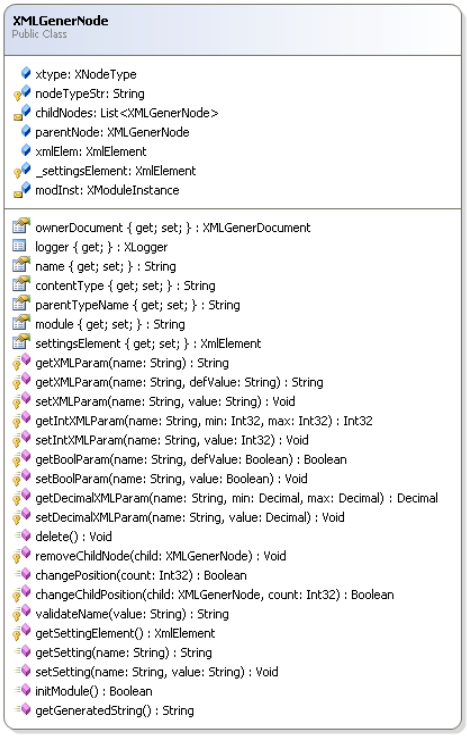
Na štruktúru aplikácie možno nazerať dvoma spôsobmi. Z pohľadu editovania šablóny (kapitola 5.3) a z pohľadu procesu generovania (kapitola 5.5). V následujúcej kapitole 5.2 sú popísané hlavné triedy a ich hierarchia.

5.2.1. Popis hlavných tried

Hierarchia tried XML Generatora je inšpirovaná DOM [6] modelom. Základným stavebným prvkom *XMLGenerNode* (obrázok 56), ktorý je predkom každého uzla (uzla typu *element*, *elements*, *attribute* atď.). Tá je zobrazená na obrázku 55.



Obrázok 55: hierarchia dedenia základných tried



Obrázok 56: Hlavné vlastnosti a funckie triedy XMLGenerNode

5.2.2. Trieda XMLGenerNode

Nasledujúci zoznam v krátkosti popisuje vlastnosti a funkcie triedy XMLGenerNode:

- *xtype* – výčtový typ, ktorý určuje typ uzla, kde každý zdedený potomok má svoj vlastný typ.
- *nodeType* – meno XML elementu (v XML dokumente), ktorý si každý potomok určí sám (podľa sa pomenuje element v XML dokumente)
- *childNodes* – zoznam detských elementov
- *parentNode* – nadradený uzol typu *XMLGenerNode*
- *xmlElem* – referencia na element inštancie *XmlDocument*, (viď kapitola 5.2.11)
- *ownerDocument* – referencia na XMLGenerDocumet (podobne ako *XmlNode* má referenciu O*wnerDocument*)
- *logger* – referencia na objekt, do ktorého sa posielajú všetky správy (chybové hlášky, upozornenia a pod.). Tá sa nastaví len v *ownerDocument*-e, ktorého detské uzly si túto referenciu sami preberú
- *name* – meno uzla vo vygenerovanom XML dokumente
- *contentType* – typ obsahu. Pre potomka triedy XMLGenerNode môže nadobúdať iný význam, ako napríklad pre *XMLGenerAttribute* znamená dátový typ (rovnako tak pre *XMLGenerElement*), ale pre XMLGenerElements znamená, v akom poradí budú generované detské uzly
- *parentTypeName* – tento parameter sa využíva pri dedení dátových typov. Označuje predka, z ktorého je dátový typ (trieda *XdataType*) zdedený
- *module* – je *id* modulu, ktorý generuje dáta pre tento uzol
- *settingsElement* – je referencia na XmlElement, ktorá je detským uzlom elementu *xmlElem* (viď kapitola 5.2.11)
- *getXMLParam* – funkcia pre získanie hodnoty atribútu z elementu *xmlElem*. Jej pridanou hodnotou je, parameter *defValue* (čiže východzia hodnota), ktorý vracia v prípade, že atribút neexistuje. Funkcia vracia string. Následujúce funkcie *getBoolParam*, *getIntXMLParam* a *getDecimalXMLParam* používajú túto funkciu pre získavanie dát.
  - *getBoolParam* – funkcia vracia typ bool
  - *getIntXMLParam* – funkcia s definovaním rozsahu, v ktorom sa výsledok môže nachádzať. Vracia typ int
  - *getDecimalXMLParam* – funkcia s definovaním rozsahu, v ktorom sa výsledok môže nachádzať. Vracia typ decimal
- *setXMLParam* – funkcia inverzná k funkcii *getXMLParam*. Čiže nastavuje hodnotu atribútu elementu *xmlElem*. Následujúce funkcie

setIntXMLParam, *setBoolParam* a *setDecimalXMLParam* ju používajú k ukladaniu dát (po konverzii hodnoty do stringu).

- *setIntXMLParam* – nastavuje hodnotu typu int
- *setBoolParam* – nastavuje hodnotu typu bool
- *setDecimalXMLParam* – nastavuje hodnotu typu decimal

- *delete*, *removeChildNode* – funkcia *delete* vymaže aktuálny uzol (vrátane referencie do *xmlElem*). Funkcia *removeChildNode* vymaže detský uzol
- *changePosition*/*changeChildPosition* – funkcie pre zmenu poradia aktuálneho/detského uzla (vrátane zmeny poradia *xmlElem*). Jej používanie je spomenuté v kapitole 4.6.4
- *getSettingElement* – XML element určený pre ukladanie nastavení parametrov z modulu
  - *getSetting* – vráti parameter modulu z elementu *getSettingsElement*
  - *setSetting* – ukladá parameter modulu do elementu *getSettingsElement*
- *initModule* – funkcia vytvára inštanciu modulu. Vráti true, ak inicializácia prebehla v poriadku
- *getGeneratedString* – vracia string, ktorý je vygenerovaný modulom

5.2.3. Trieda XMLGenerRagneElement

Táto trieda je použitá len ako základ pre triedy *XMLGenerElement*, *XMLGenerElements* a *XMLGenerTextNode*. Jej pridanou hodnotou oproti *XMLGenerNode* je nastavovanie atribútov *Min/Max count*, *Min/Max frequency* a *Min/Max fanout* (viď kapitola 4.6.1 a 4.6.2).

5.2.4. Trieda XMLGenerElement

Trieda ktorá je oproti predkovi *XMLGenerRangeElement* rozšírená o zoznam atribútov. Od tejto triedy sa očakáva, že jej inštancia bude mať nadradený uzol.

5.2.5. Trieda XMLGenerMainElement

Tejto triede je umožnené nemať nadradený uzol, pretože sa používa ako koreň stromu.

5.2.6. Trieda XMLGenerElements

Trieda *XMLGenerElements* je špecifická v tom, že jej *Content type* (viď kapitola 4.10) nie je dátový typ, a vo vygenerovanom súbore sa nezobrazuje. Funguje ako akási zátvorka.

5.2.7. Trieda XMLGenerTextNode

Trieda *XMLGenerTextNode* sa správa ako *XMLGenerElement*, s tým rozdielom, že vo výstupe sa zobrazí len jeho obsah. (viď kapitola 4.11).

5.2.8. Trieda XMLGenerAttribute

Trieda *XMLGenerAttribute* má na starosti generovanie atribútu v elemente. Neobsahuje žiadne detské uzly a naopak žiadny uzol nemôže mať túto triedu ako nadradený uzol. Jediným možným nadradeným uzlom je trieda typu *XMLGenerElement*, ktorá má pre uzly typu *XMLGenerAttribute* oddelený zoznam, v ktorom ich udržiava (viď kapitola 5.2.4).

5.2.9. Trieda XDataType

Táto trieda predstavuje užívateľom definovaný dátový typ. Oproti predkovi má navias parametre určujúce obmedzenia: *length*, *minLength*, *maxLengt*, *minInclusibe*, *maxInclusive*, *minExclusive*, *maxExclusive*, *pattern* a *enumeration*. Trieda má nadradené uzol *XMLGeneerDataTypes*, a nemôže mať žiadne detské uzly.

5.2.10. Trieda XMLGenerDataTypes

Trieda *XMLGenerDataTypes* slúži len ako zoznam, ktorého detské uzly sú tiredy *XDataType*.

5.2.11. Trieda XMLGenerDocument

Trieda *XMLGenerDocumet* predstavuje centrum všetkých referencií a funkcionalít. Má na starosti vytváranie (a ukladanie) stromu zo súboru šablóny, spúšťanie generovania, obsluhuje moduly výstupu, ponúka zoznamy dátových typov pre všetky uzly, prepočítava horné a dolné hranice uzlov (viď kapitola 4.14, tlačítko *Calculate ranges*), spúšťa predgenerovania importovaných šablón, obsahuje všetky parametre, ktoré sú v GUI v záložke *Document settings* a *Output* (viď kapitola 4.12).

Trieda *XMLGenerDocument*, má na starosti načítanie šablóny (tj. XML dokumentu) a udržiava jej inštanciu (tj. inštanciu triedy *XmlDocument* z knižnice *System.Xml*), z ktorej vytvorí strom pozostávajúci z tried potomkov *XMLGenerNode*.

Pre jednoduché ukladanie si *XMLGenerNode* drží referenciu na odpovedajúci XML element v strome, v kde koreňovým ulzom je *mainelement*. Väzba XML element a XMLGenerNode je 1:1, pokiaľ nepočítame elementy settings. Táto referencia v triede *XMLGeneNode* je uložená v parametri *xmlElem* (obrázok 56).

5.2.12. XLogger

Trieda *XLogger* funguje ako rozhranie pre zobrazovanie správ. Jej funkcie *addLog* a *addErrorLog* spôsobia volanie udalosti *XlogEvent*, alebo *XErrorEvent*. *XLogger* má definovanú vlastnú statickú inštanciu, na ktorú *LogForm* (viď

kapitola 4.4) nastaví svoje obsluhujúce metódy (tzv. „povesí hendlery“). To má za efekt, že všetky správy sa zobrazujú v *LogForme*.

Pri vytváraní inštancie triedy *XMLGenerDocument* sa jej do konštruktora nastavuje *XLogger*, do ktorého všetky detské uzly vypisujú správy. V prípade, že *XMLGenerDocument* nemá nastavený *XLogger*, používa jej statickú inštanciu. Takýmto spôsobom sa pre každý *XMLGenerDokument* nastaví, kde sa budú vypisovať jeho správy.

5.3. Editácia a grafické rozhranie

Hlavnou riadiacou časťou je formulár s ovládacím panelom, ktorý do komponenty *TabControl* (z knižnice *System.Windows.Forms*) pridáva záložku (obsahujúca komponentu *XTemplatePanel*) pre každú novootvorenú šablónu.

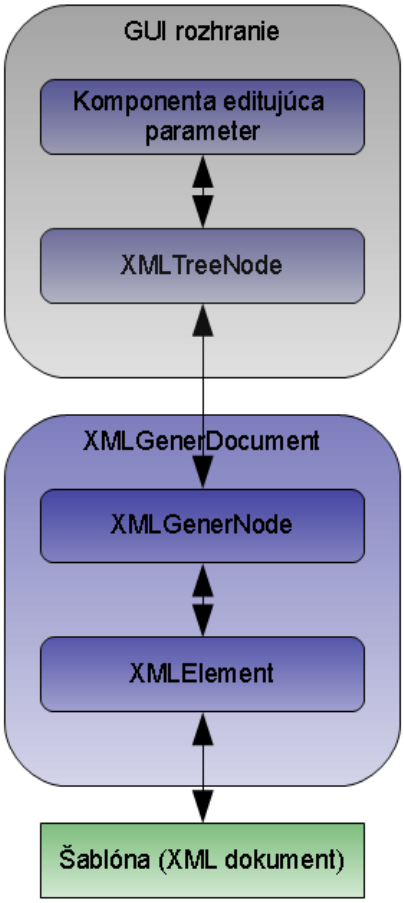
XTemplatePanel je hlavnou časťou pri editácii uzlov. Pre formulár poskytuje funkcie pre ukladanie dokumentu, vytvorenie a mazanie uzlov, posun uzlov (menenie ich poradia), generovanie XML dokumentov, generovanie schém podľa šablóny a prepočítavanie horných a dolných hraníc elementov (viď kapitola 4.14). Hlavný formulár obsluhuje len otváranie a zatváranie záložiek (panelov).

XTemplatePanel sa delí na dve časti. Jedna časť zobrazuje stromovú štruktúru šablóny a druhá zobrazuje detail a edituje vybraný uzol a parametre dokumentu.

Stromová štruktúra šablóny sa zobrazuje v komponente TreeView (z knižnice *System.Windows.Forms*). Pre zobrazenie uzlov šablóny, bola vytvorená trieda XTreeNode, ktorá je potomkom triedy *TreeNode* (z knižnice *System.Windows.Forms*). *XtreeNode* si drží odkaz na XMLGenerNode, z ktorého berie (pre svoje potreby) meno uzla (*name*) a jeho typ (*contentType*), ktoré si nastavuje ako parameter *Text* (ten sa zobrazuje v komponente *TreeView*). Obrázok 57 ukazuje vrstvy, cez ktoré sa prenášajú dáta. Najspodnejšou vrstvou je súbor, a najvyššou komponenta, ktorá dáta zobrazuje resp. edituje. Sivá vrstva predstavuje GUI rozhranie (resp. *XTemplatePanel*), modrá jadro generátora a zelená systémový súbor (XML dokument).

Pri editácii sa údaje nastavujú priamo až do triedy typu XMLElementu (čo ilustruje obrázok 57). Ukladanie je preto veľmi jednoduché. Stačí na inštancii *XMLDocument* použiť funkciu *save*.

XtemplatePanel ma premennú *selectedNode* typu *XTreeNode*, ktorá sa nastavuje pri kliknutí na uzol v strome (ktorý zobrazuje štruktúru šablóny). Pri nastavovaní sa zavolá funkcia pre obnovenie záložky *Node info*, kto zobrazí, alebo schová komponenty, ktoré sú špecifické pre vybraný uzol. Funkcia tiež volá funkciu pre zostavenie editačného panelu pre modul (viď kapitola 5.4.4).



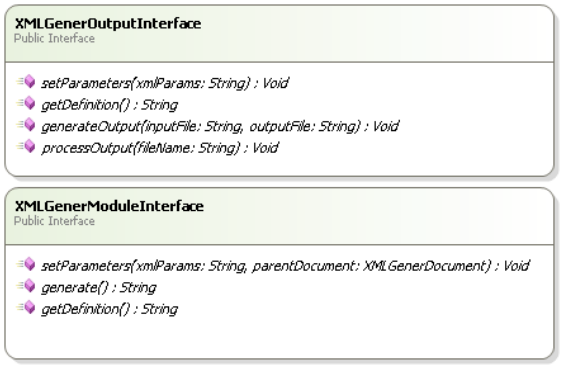
Obrázok 57: štruktúra ediácie

5.4. Moduly

Moduly, ktoré sú použité pri generovaní výsledného dokumentu sa rozdeľujú do 2 skupín:

- moduly pre generovanie obsahu, ktoré implementujú interface *XMLGenerModuleInterface*
- moduly pre zpracovanie vygenerovaného XML dokumentu, nazývané moduly výstupu, ktoré implementujú interface *XMLGenerOutpurInterface*

Program pri inicializácií prejde celý adresár modules, v ktorom otvorí každú knižnicu a v prípade, že nájde triedu, ktorá implementuje interface *XMLGenerModuleInterface* alebo *XMLGenerOutputInterface* pridá si do zoznamu. Zoznam potom prechádza a z tried vytvára inštancie, z ktorých určuje ich parametre, typ, ktorý generujú atď. (viď kapitola 4.9)



Obrázok 58: interface pre moduly

5.4.1. Moduly pre generovanie obsahu

Trieda takéhoto modulu musí implementovať následujúce metódy (z interfacu XMLGenerModuleInterface):

- *void setParameters(string xmlParams, XMLGenerDocument parentDocument)* – metóda zavolaná generátorom pre inicializáciu modulu. V nej predá generátor nastavenia pre modul (z uzla *XMLGenerNode* hodnotu *getSettingsElement()*, pre ktorý je modul určený) v XML formáte a referenciu na dokument, ktorý sa generuje. Referenciu mu predá aby modul mohol podľa O*utId* získať skutočné meno vygenerovaného XML dokumentu (použitie je popísané v kapitole 4.9.6).
- *string generate()* - modul vráti vygenerovaný text.
- *string getDefinition()* - modul predáva XML Generatou informácie o sebe. Je to reťazec obsahujúci XML dáta (viď kapitola 5.4.3)

5.4.2. Moduly výstupu

Generátor volá modul výstupu, ak nastavený :

- nie je (resp. modulom výstupu je *None*, viď kapitola 4.13.1) generátor generuje dáta priamo do výstupného súboru.
- je, generátor najprv vygeneruje dočasný súbor, ktorého cestu predá modulu a predá mu zároveň cestu k výstupnému súboru.

Trieda modulu výstupu musí implementovať interface *XMLGenerOutputInterface*:

- *void setParameters(string xmlParams)* – metóda zavolaná generátorom pre inicializáciu dokumentu. V nej predá generátor nastavenia pre modul v XML formáte .

- *void generateOutput(string inputFile,string outputFile)* – modul zpracuje súbor *inputFile* a vygeneruje súbor *outputFile*. Táto metóda sa spúšťa len v prípade, že modul je nastavený ako primárny

- *void processOutput(string fileName)* – táto metóda sa volá len v prípade, keď je modul nastavený ako sekundárny modul

- *string getDefinition()* – modul predáva XML Generatou informácie o sebe. Je to reťazec obsahujúci XML dáta (viď kapitola 5.4.3)

5.4.3. Definícia a nastavenie modulu

Definícia (metóda getDefunution) a nastavenie parametrov modulu z generátora funguje pri oboch typoch modulov rovnako. Generátor najprv získa z modulu jeho definíciu, na základe ktorej dokáže vygenerovať GUI pre editáciu parametrov (viď kapitola 5.4.4) , meno modulu, *id* modulu, v prípade modulu pre generovanie obsahu, typ výstupu a obmedzenia (*restriction*), ktoré modul dokáže akceptovať.

Príklad definície je zobrazený v ukážke 47.

```
<function>
    <name>Pattern generator</name>
    <id>patternGenerator</id>
    <type>string</type>
    <attributes>
       <attribute>
          <name>Pattern</name>
          <id>text</id>
          <type>text</type>
       </attribute>
    </attributes>
   <acceptrestriction>pattern</acceptrestriction>
</function>
```

Ukážka 47: Príklad definície modulu

Podľa ukážky 47 je:

- meno modulu (ktoré sa zobrazí užívateľovi) *Pattern generator*

- *id* modulu, ktoré sa ukladá do šablóny pre daný uzol *patternGenerator*

- má jediný atribút:

  - ktorého meno je Pattern (zobratí sa užívateľovi)

  - id parametra je *text*

  - typ parametra je *text*

- akceptuje obmedzenie *pattern*

Ukážka 48 zobrazuje jeho uloženie súbore šablóny.

```
<attribute name="patterngeneratorattribute"
     module="patternGenerator">
    <setting text="[0-9]{3} [0-9]{2}" />
</attribute>
```

Ukážka 48: nastavenie modulu MySQL v súbore šablóny

Pri nastavovaní parametrov (funkciou *setParameters*) si modul musí sám dáta rozparsovať a nastaviť. XML Generator volá metódu *setParameters* pri inicializácii generovania, takže je vhodné by si modul v prípade potreby nakešoval dáta. XML Generator potom na module volá len funkciu *generate()*, ktorá by mala podľa možností pracovať čo najrýchlejšie.

5.4.4. Funkcia pre zostavenie editačného panelu pre modul

XML Generator z modulu získa informácie o module pomocou funkcie *getDefinition()*. Tá vráti reťazec obsahujúci XML dáta, ktorých schéma je popísaná v ukážke 49.

```
<!ELEMENT function (name,id,type,attributes?,acceptrestriction*)>
<!ELEMENT name (#PCDATA) >
<!ELEMENT id (#PCDATA) >
<!ELEMENT type (#PCDATA) >
<!ELEMENT attributes (attribute*) >
<!ELEMENT attribute (name,id,type,items?) >
<!ELEMENT acceptrestriction (#PCDATA) >
<!ELEMENT items (item*) >
<!ELEMENT item EMPTY >
<!ATTLIST item
              name CDATA #IMPLIED
              value CDATA #IMPLIED >
```

Ukážka 49: schéma XML dát, ktoré vracia modul

XTemplatePanel predá tieto dáta spolu s editovaným uzlom triedy *XMLGenerNode* komponenete *XModuleEditor*, ktorá si podľa elementov *attribute* zostaví panel pre editáciu parametrov. Panel zostavuje z komponent určených na editáciu atributov modulu. V kapitolách 5.4.4.1 až 5.4.4.7 sú popísané všetky typy *type* a komponenty z ktorých *XModuleEditor* vytvára gui. *XModuleEditor* každej komponenete predá *XMLElement* (ktorý získa z *XMLGenerNode*) s menom parametra a jeho *id*, ktoré označuje meno atribútu v elemente. Komponenta si sama získa aktuálny údaj a tiež sa sama stará o jeho nastavenie.

5.4.4.1. Typ text a password

XModuleEditor pre oba typy používa komponentu XTextItem (na obrázku 59). V prípade typu *password* len nastaví textovému poľu, aby zobrazoval heslo.

Obrázok 59: príklad komponenty XTextItem

Ukážka 50 popisuje, ako vyzerá element *attribute*, z ktorého tieto komponenty vygenerujú.

```
<attribute>
   <name>Server</name>
   <id>server</id>
   <type>text</type>
</attribute>
<attribute>
   <name>Password</name>
   <id>password</id>
   <type>password</type>
</attribute>
```

Ukážka 50:príklad definície parametru typu text a password

5.4.4.2. Typ number a float

XModuleEditor pre oba typy používa komponentu *XNumberItem (na obrázku 60)*, kde len nastaví, desadinné miesta.

Obrázok 60: príklad komponenty XNumberItem

Ukážka 51 popisuje, ako vyzerá element *attribute*, z ktorého tieto komponenty vygenerujú.

```
<attribute>
   <name>Starts value</name>
   <id>startvalue</id>
   <type>number</type>
</attribute>
<attribute>
   <name>Step</name>
   <id>step</id>
   <type>float</type>
</attribute>
```

Ukážka 51:príklad definície parametru typu text a password

5.4.4.3. Typ generated file

XModuleEditor pre typ *generatedFile* používa komponentu *XGeneratedFileSelector*, ktorá je zobrazená na obrázku *61*.

Obrázok 61: príklad komponenty XGeneratedFileSelector

Ukážka 52 popisuje, ako vyzerá element *attribute*, z ktorého je táto komponenta generovaná.

```
<attribute>
    <name>Source file</name>
    <id>sourceId</id>
    <type>generatedFile</type>
</attribute>
```

Ukážka 52:príklad definície parametru typu generatedFile

5.4.4.4. Typ file

XModuleEditor pre typ *file* používa komponentu *XFileSelector*, ktorá je zobrazená na obrázku 62. Tá otvorí dialógové okno pre výber súboru. Meno súboru je tiež možné písať do textového poľa.

Obrázok 62: príklad komponents XFileSelector

Ukážka 53 popisuje, ako vyzerá element *attribute*, z ktorého je táto komponenta generovaná.

```
<attribute>
   <name>Source file</name>
   <id>source</id>
   <type>file</type>
</attribute>
```

Ukážka 53:príklad definície parametru typu file

5.4.4.5. Typ select

XModuleEditor pre typ *select* používa komponentu *XComboItem*, ktorá je zobrazená na obrázku 63.

Obrázok 63: príklad komponenty XComboItem

Ukážka 54 popisuje, ako vyzerá element *attribute*, z ktorého je táto komponenta generovaná.

```
<attribute>
   <name>Generating style</name>
   <id>order</id>
   <type>select</type>
   <items>
      <item name="Random numbers" value="random"/>
      <item name="Sequence (use step)" value="sequence"/>
   </items>
</attribute>
```

Ukážka 54:príklad definície parametru select

5.4.4.6. Typ bool

XModuleEditor pre typ *bool* používa komponentu *XBooleanItem*, ktorá je zobrazená na obrázku 64.

Obrázok 64: príklad komponenty XBooleanItem

Ukážka 55 popisuje, ako vyzerá element *attribute*, z ktorého je táto komponenta generovaná.

```
<attribute>
   <name>Skip header</name>
   <id>skip</id>
   <type>bool</type>
</attribute>
```

Ukážka 55:príklad definície parametru bool

5.4.4.7. Typ info

XModuleEditor pre typ *info* používa komponentu *XInfoItem*, ktorá je zobrazená na obrázku 65 (šípka 1). Tá nenastavuje žiadny parameter. Je určená len zobrazenie nejakek informácia (ako napríklad to, že indexovanie stĺpcov začína od 0).

Obrázok 65: príklad komponenty XInfoItem

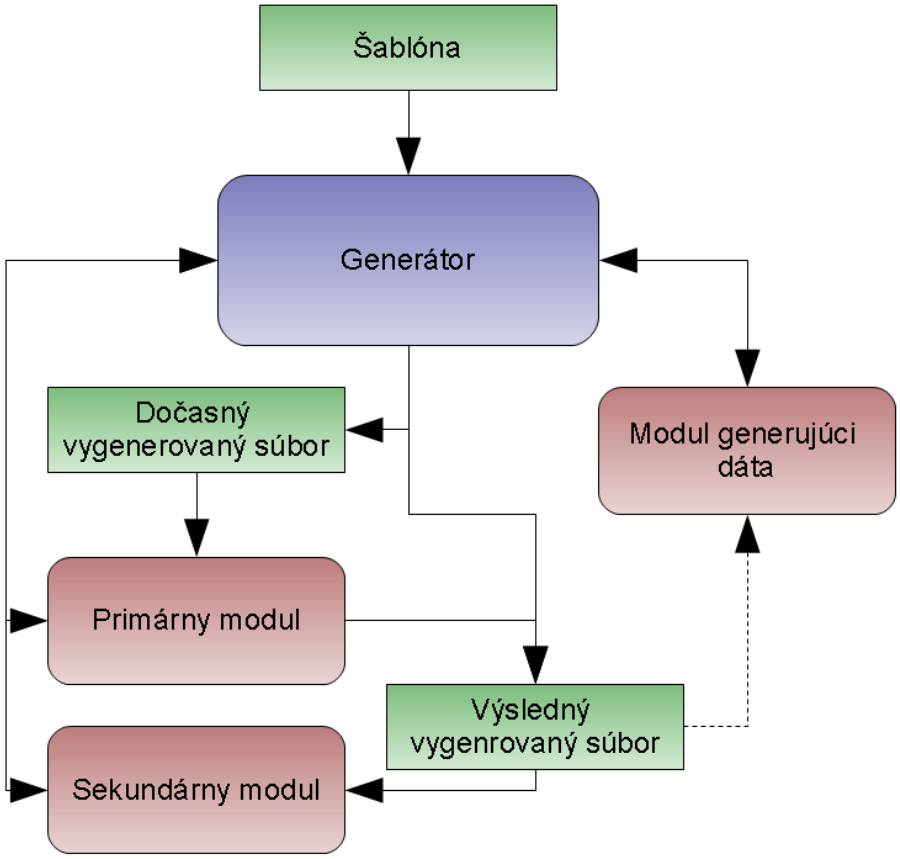
Ukážka 56 popisuje, ako vyzerá element *attribute*, z ktorého je táto komponenta generovaná.

```
<attribute>
```

```
   <name>Column indexing starts with 0 (0=first column).</name>
   <id>i0</id>
   <type>info</type>
</attribute>
```

Ukážka 56:príklad definície parametru typu info

5.5. Generovanie XML Dokumentov



Obrázok 66: schéma procesu genrovania

Obrázok 66 zobrazuje schému, ako spolu komunikujú moduly a generáto, zelené objekty predstavujú súbory.

Generovanie sa spúšťa v hlavnom formulári. Ten vytvorí nové okno tiredy *ProcessForm*, ktoré pomocou komponenty *BackGroundWorker* (z knižnice *System.ComponentModel*) spustí paralelný proces. *ProcessForm* vytvorí vlastnú inštanciu triedy XLogger, a inštanciu triedy *XMLGenerDocument*, ktorej nastaví svoj vlastný *XLogger* (viď kapitola 5.2.12). Takto si zaistí vypisovanie správ do vlastného okna. Potom spustí načítanie šablóny zo súboru (metódou *XMLGenerDocument.Load*) zkontroluje správnosť nastavených parametrov a prepočítanie (metódou *XMLGenerDocumentCheck*) a následne nato generovanie.

Celé generovanie zaisťuje trieda *XMLGenerDocument.* Tá najprv spusti generovanie šablón, ktoré má nastavené v zozname pre predgenerovanie (viď

kapitola 4.9.6), potom vytvorí dokument, ktorý obsahuje schému (typu DTD alebo XSD), zapne stopky a začína generovať XML dokumenty (podľa nastaveného počtu) metódou *XMLGenerDocument.generateDocument*,a nad vygenerovanými dokumentami následne spúšťa moduly výstupu(moduly spracúvajúce vygenerovaný XML dokument). Po dokončení zastaví stopky, a zobrazí nameraný čas.

Metóda *XMLGenerDocument.generateDocument*, ktorá má ako parameter index dokumentu (podľa ktorého vytvorí meno výstupného súboru), už priamo generuje XML dokument (viď kapitola 5.5.2).

5.5.1. Algoritmus pre určenie minimálneho a maximálneho počtu elementov

Tento krok pripravuje dokument pre generovanie .V každom uzle (triede zdedenej od triedy *XMLGenerRangeElement*) je zadefinovaný aspoň jeden z parametrov *Min/Max Count*, *Min/Max frequency* a *Min/Max fanout.* Tieto parametre je ale potrebné vyčísliť, o čo sa stará funkcia *XMLGenerDocument.preCalculate*. (tá sa spúšťa vo funkcii *XMLGenerDocument.check*, ktorá je spustená v okne *ProcessForm* vždy pred generovaním).

Cieľom je vypočítať *Min total* a *Max total* (viď kapitola 4.14).

Celý výpočet sa robí na 2 kroky.

1. Výpočet výsledného fanoutu podľa hodnôt detských uzlov v prípade, že fanout nemá zadaný (pseudoalgoritmus v ukážke 57)
2. Poskladaj z vypočítaných hodnot M*in total* a M*ax total (pseudoalgoritmus v ukážke 58)*

```
stack.push( mainelement )
while(stack.count > 0 ){
   el=stack.pop();
   foreach (childNode in el.childNodes){
      if (childNode.useFrequency){
          // súčet frekvencií
          el.childUsesFreq = true;
          el.sumMinChFreq += childNode.minFrequency;
          el.sumMaxChFreq += childNode.maxFrequency;
      }else{
          // súčet počtov
          el.childUsesCount = true;
          el.sumMinChCount += childNode.minCount;
          el.sumMaxChCount += childNode.maxCount;
      }
      stack.Push(chNode);
  }
  kontrola:
    -hodnota súčtu frekvencií v detských elementoch <=100
    -ak detské elemnty používaju len frekvenciu,
     musí byť zapnutý fanout
    -ak súčet  minimálnych frekvencií == 100,
     sumMinChCount s musí rovnať 0
```

```
  // vypočítaj dolnú hranicu
  if (el.sumMinChFreq == 100)
      el.sumMinChMixCount = el.minFanout;
  else //
      el.sumMinChMixCount =
        Math.Round(el.sumMinChCount +
          (el.sumMinChCount * el.sumMinChFreq) /
          (100 - el.sumMinChFreq)); ;
  if (el.useFanout){
     -> error
  }

  // vypočítaj hornú hranicu
  if (el.sumMaxChFreq < 100){
      el.sumMaxChMixCount = Math.Round(el.sumMaxChCount +
                    (el.sumMaxChCount * el.sumMaxChFreq) /
                    (100 - el.sumMaxChFreq));
      if (el.sumMaxChMixCount < el.minFanout && el.useFanout){
        ->error
      }
  }else if (!el.useFanout)
          ->error

    }

   // výsledok je v el.sumMaxChMixCount a el.sumMinChMixCount
}
```

Ukážka 57: algoritmus pre výpočet výsledného fanoutu uzlov

```
stack.push( mainelement )
while(stack.count > 0 ){
  el = stack.Peek();

  -ak element neobsahuje detské uzly,
   nastav mu minTotal a maxTotal na 1, vyber element z fronty a
   pokračuj v cykle

  -ak neboli prepočítané všetky detské uzly, vlož do
   stacku tie, ktoré ešte prepočítané neboli a pokračuj odznova

  minimálnaRezerva= el.minFanout;
  maximálnaRezerva= el.maxFanout;
  el.minTotal = 1;
  el.maxTotal = 1;

  -prepočítaj minimálne a maximálne frekvencie detských uzlov na
   počet podla minimálnaRezerva (ktorá presdavuje 100%)(elementom
   zadaných frekvenciou tam vznikne parameter
   minCount a maxCount)

  // odpočítame potrebné minimum
  foreach(child in el.childs){
     child.reserver= child.maxCount-child.minCount;
```

```
    el.minTotal+= child.minCount * child.minTotal;
  }
  //tu už rozdelíme rezervy
  -od minimálnaRezerva a maximálnaReyerva odpočítaj minCount
   každéhodetského uzla

  minList = zoraď detské elementi podľa minTotal od najmenšieho

  // vypočítame minTotal
  foreach (child in el.childs){
     min = Math.Min(minimálnaRezerva, child.reserver );
     minimálnaRezerva -= min;
     el.minTotal += min * n.minTotal;
     if ( minimálnaRezerva < 1)
       break;
  }

  // počítame maxTotal ========
  -prepočítaj minimálne a maximalne frekvencie detských uzlov na
   počet podla maximálnaRezerva (ktorá presdavuje 100%)(elementom
   zadaných frekvenciou tam vznikne parameter minCount
   a maxCount)

  maxList = zoraď detské elementi podľa maxTotal od najväčšieho

  // odpočítame potrené minumim
  foreach(child in el.childs){
     child.reserver= child.maxCount-child.minCount;
     el.minTotal+= child.minCount * child.minTotal;
  }

  foreach (child in el.childs){
     min = Math.Min(maximálnaRezerva, child.reserver );
     maximálnaRezerva -= min;
     el.maxTotal += min * n.maxTotal ;
     if ( maximálnaRezerva < 1)
       break;
  }
}
```

Ukážka 58: algoritmus pre výpočet výsledného Min/Max total

Týmto spôsobom bol pre každý uzol vypočítaný minimálny a maximálny počet elementov v celom jeho podstrome, jeho fanout a *Min/Max Count.*

Zložitosť vypočítania minimálneho a maximálneho počtu:

- Pri výpočte 1. kroku prechádzame celý strom, pričom s každým uzlom pracujeme O(1) krát. Čiže 1. krok má zložitosť O(N).
- Pri priechode v 2. kroku s každým uzlom pracujeme 1 krát, ale pre zistenie najmenšieho a najväčšieho možného počtu, ktorý môže daný uzol

vygenerovať, je použité triedenie (pre usporiadanie od najväčšieho po najmenší). Z toho dostávame zložitosť 2. kroku O(N*log(N))

5.5.2. Generovanie

Pri generovaní začíname od koreňa. Prechádzame stromom do hĺbky: pri priechode stromom je na vstupe pre aktuálne prechádzaný uzol celkový počet elenetov, ktorý má byť rozdelený pre jeho potomkov (resp. počet -1, pretože odčítame aktuálny uzol). Aby bola štruktúra čo najrozmanitejšia, algoritmus sa snaží rozdeliť počet čo najnáhodnejšie. Algoritmus je popísaný v ukážke 59.

```
//vstupom je číslo total a element , ktoré máme rozdeliť medzi
detské uzly
  minFanout = el.minFanout
  maxFanout = el.maxFanout

  // cyklus 1
  // cyklus hladá vhodný fanout, aby počet total mohol
  // byť vygenerovaný
  while(1){
    if (minFanout>=maxFanout)
       break;
    rnd= random(minFanout,maxFanout)

    - pre číslo rnd vypočítaj min/max total algoritmom
      z ukážky 58
    if ( min total<= total <= max total)
      break;
    if (total < min total)
       maxFanout= maxfaout – (maxFanout - minFanout)/2
    else //  teda ak total > max total
       minFanout = minFanout + (maxFanout - minFanout)/2
  }
  child.reserve = child.maxCount – child.minCount;
  minFanoutSum = 0;

  // cyklus 2
  // každému detskému elementu priraď náhodne počet
  // elementov z rezervy
  foreach(child in el.childs){
     child.randomValue = random.nextDouble() *
                                  child.fanoutReserve;
     randomSum+=child.randomValue;
     minFanoutSum+=child.minCoun;

  }

  // cyklus 3
  //rozdel fanout medzi detské uzly
  totalReserve= total – minFanoutSum;
  foreach(child in el.childs){
     if (randomSum > 0)
        tmp=child.randomValue * fanoutReserve / randomSum;
     else
        tmp = 0;
     tmp=min(tmp,child.reserve);
     child.thisElementCount = tmp + child.minCount;
```

```
    randomSum -= child.randomValue;
    totalReserve -= tmp;
}

// cyklus 4
- over a ošetri rozdelenie fanoutu:
    - V prípade, že toto rozdelenie nestačí, na vygenerovanie
      total-u, polením intervalu sa pre každý detský uzol
      približuj k jeho maxFanout-u
    - V prípade, že toto rozdelenie presiahlo total,  zmenšuj
      pridelený fanout polením intervalu k minFanout-u
- opakuj krok „over a ošetri rozdelenie fanoutu", dokiaľ
  rozdelenie nebude v poriadku

// cyklus 5
// v tomto bode už má každý detský uzol pridelený správny
// fanout
minsum =0;
maxsum =0;
foreach(child in el.childs){
  minsum+= child.fanout * child.minTotal;
  maxsum+= child.fanout * child.maxTotal;
  child.countReserve = child.maxCount-child.minCount;
}


// cyklus 6
//následujúci cyklus rovnomerne rozdelí total
//jednotlivým detským uzlom

res = maxsum-minsum;
tot = total -minsum;
foreach(child in el.childs){
  tmp = child.fanout *child.countReserve;
  if (res>0)
    addReserver = tot * tmp / res;
  else
    addReserver = 0;
  tot-=addReserve;
  res-=tmp;
  child.total = child.fanout * chlild.mincount + addReserve;
  child.reserverForDistribution = addReserve;

}
```

Ukážka 59: algoritmus rozdelenia počtu generovaných uzlov medzi detské uzly

Zložitosť algoritmu z ukážky 59:

N označuje počet detských elementov.

- Cyklus 1 pre nájdenie vhodného fanout-u prebehne najviac log(N) krát.

- V ňom je použitá funkcia pre pre zistenie, či nájdený fanout je v poriadku (algoritmus z ukážky *58*). Tá trvá N*log(N), takžé výsledná zložitosť cyklu je $N*\log^2(N)$

- Cyklus 2 prebehne v čase O(N)
- Cyklus 3 prebehne v čase O(N)
- Cyklus 4 sa polením intervalu bude opakovať najviac O(log N) krát. Každý cyklus pracuje prejde všetky detské uzly, takže výsledná zložitosť bude O(N.log(N))
- Cyklus 5 aj 6 prebehnú v čase O(N)

Výsledná zložitosť rozdelenia počtu uzlov, ktoré sa majú v strome vygenerovať, mezdi detské uzly je teda:

$$O(N*\log^2(N) + N + N + N*\log(N) + 2N) = O(N*\log^2(N))$$

6. Záver

Cieľom práce bola implementácia aplikácie, ktorá dokáže generovať dostatočne zložité XML dokumenty, väčšie ako operačná pamäť, kde vygenerované dokumenty majú zachovávať užívateľom nadefinovanú štruktúru. Ďalšími atribútmi, ktoré má aplikácia realizovať boli počet elementov, ich frekvencia , fanout alebo generovanie rôznych dátových typov. Po dokončení má k vygenerovaným dokumetom vytvoriť DTD alebo XSD dokument a namerané výsledky zobraziť v príjemnej podobe. Aplikácia tiež mala zaisťovať automatické spúšťanie aplikácií nad vygenerovanými dokumentami.

XML Generator poskytuje príjemné GUI, ktoré je prispôsobené pre čo najrýchlejšiu a čo najpohodlnejšiu prácu. Aplikácia sama opravuje chyby, prípadne upozorní užívateľa na možné problémy, ako napríklad zlé parametrov v dátovom type. Koncept predlohy z ktorej sa generujú XML dokumenty (šablóna) vychádza z jazyka XML Schema, čo umožní užívateľovi, ktorý je s ňou oboznámený, veľmi rýchle zžitie sa s aplikáciou.

V porovnaní s komerčným alebo inými dlho vyvíjanými konkurenčnými aplikáciami, je jej nevýhoda v slabej podpore všetkých vlastností, ktoré XML Schema popisuje. Na druhú stranu ale má veľké výhody v podpore jednoduchej a veľmi účinnej rozšíriteľnosti a rýchlosti generovania, ktorými sa žiadny iný generátor nemôže chváliť (vyplýva z výsledkov testo, ktoré sa nachádzajú v prílohe B). Práve na tieto vlastnosti bol počas vývoja aplikácia kladený najväčší dôraz.

Počas písania tejto práce bolo zistené, že je takmer nemožné vygenerovať dokument obsahujúci všetky extrémy, ktoré sú zadefinované v šablóne. Často sa objavovali problémi s nedostatočným rozsahom čísel, ktoré jazyk C# ponúka, pretože už pri malej hĺbke dokumentu sa ľahko prekračuje číslo $7{,}8.10^{24}$, čo je maximálna hodnota typy decimal. V práci sme sa snažili podporovať dedenie elementových typov, čo ale veľmi úzko súvisí s hĺbkou dokumentu, ktorej implementácia je veľmi náročná do súčasnej aplikácie a nepodarilo sa ju realizovať.

Veľkou výhodou použitého prostredia C# bola buď to už priama, alebo nepriama podpora XML Regexpu, PostgreSQL a vôbec všetkých nástrojov a technológií, ktoré sú použité v moduloch.

Ďalšia práca na aplikácií by mohla zahrňovať:

- vytvorenie šablóny z DTD, XSD alebo XML dokumentu
- podporu dedenia typov elementov a nastavenie hĺbky vygenerovaného dokumentu
- podporu všetkých konštruktov pri vytváraní DTD a XSD dokumentov
- vypočítanie výskytu elementov podľa štatistického rozdelenia, akými sú napríklad geometrické, alebo normálne rozdelenie

Použitá literatúra

[1] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, http://www.w3.org/TR/REC-xml/

[2] World Wide Web Consortium, *XML Schema Part 0: Primer Second Edition*, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

[3] World Wide Web Consortium, *Extensible Markup Language (XML)*, http://www.w3.org/XML/

[4] Aaron Skonnard, Martin Gudgin: *XML, pohotová referenční příručka(2006)*

[5] World Wide Web Consortium, *XML Schema Part 2: Datatypes Second Edition*, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[6] World Wide Web Consortium, *Document Object Model (DOM)*, http://www.w3.org/DOM/

[7] World Wide Web Consortium, *Simple Object Access Protocol (SOAP) 1.1*, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[8] World Wide Web Consortium, *XML Path Language (XPath)*, http://www.w3.org/TR/xpath/

[9] Tony Stubblebine: *Regular expression 2nd Edition(2007)*

[10] Roman Barták, *Automaty a gramatiky*, http://kti.mff.cuni.cz/~bartak/automaty/

[11] Goran Siska, *Regular Expression Random Data Generator*, http://code.google.com/p/rxrdg/

[12] Pokorný J., Halaška I.: *Databázové systémy: Vybrané kapitoly a cvičení(1998)*

[13] John C. Worsley and Joshua D. Drake: *Practical PostgreSQL(2002)*

[14] Paul DuBois: *MySQL Cookbook(2003)*

[15] Progress Software Corporation, *Stylus Studio*, http://www.stylusstudio.com

[16] Progress Software Corporation, *Stylus Studio - XML Generator*, http://www.stylusstudio.com/xml_generator.html

[17] Liquid Technologies Limited, *Liquid XML Studio*, http://www.liquid-technologies.com/

[18] Database Group of the University of Toronto, *ToXGene*, http://www.cs.toronto.edu/tox/toxgene

[19] World Wide Web Consortium, *XML Schema Part 1: Structures Second Edition*, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[20] The Npgsql Development Team, *Npgsql - .Net Data Provider for Postgresql*, http://npgsql.projects.postgresql.org/

[21] Michael Widenius, *MySQL*, http://dev.mysql.com/

Príloha A

Obsah priliženého CD

K tejto práci je priložené CD, ktoré obsahuje inštalačné súbory, testovacie dáta, elektronickú verziu tohto textu a zdrojové súbory.

Popis súborov na disku:

- XMLGener.msi – inštalačný balíček aplikácie XML Generator
- dotNetFx35setup.exe – inštalačný súbor prostredia .NET framework 3.5
- source.zip – súbor obsahujúci zdrojové súbory
- XMLGenerator.pdf – je elektronická podoba tejto práce.

Po nainštalovaní XML Generatora sú v jeho adresári templates pripravené príklady.

Príloha B

Táto príloha obsahuje výsledky testov rýchlosti generovania, ktoré boli namerané. Test konkurenčnej aplikácie a XML Generator a sa prevádzal na rovnakom počítači.

Z uvedených konkurenčných programov spomenutých v kapitole 3, jediný ToXgene dokáže vygenerovať veľké XML dokumenty tak podľa užívateľom definovanej schémy. Ostatne totiž nedokážu generovať dokumenty väčšie ako operačná pamäť (Stylus Studio, Liquid Studio), alebo nedokážu generovať dokument podľa predlohy (DataGen). Preto je jedinou aplikáciu, s ktorou má zmysel porovnávať sa je ToXgene..

Pre dobré zrovnanie mali XML Generator aj ToXgene čo najpodobnejšie predlohy, teda aby vygenerovaný dokument mal rovnakú schému. Pre porovnanie rýchlosti generovania boli vykonané dva testy.

Test 1 sa prevádzal nad dátami z ukážok 60 a 61. Jeho výsledky sú zapísané v tabuľke 1 a 2.

Test 2 sa prevádzal nad dátami z ukážok 62 a 63. Jeho výsledky sú zapísané v tabuľke 3 a 4.

Z výsledkov testoch je jasne vidieť že XML Generator bol v týchto testoch približne 20 – krát rýchlejší ako ToXgene.

Test 1

```
<tox-template>
  <tox-document name="test">
    <element name="root">
      <complexType>
        <element name="a" minOccurs="10000" maxOccurs="10000">
          <complexType>
            <element name="x" minOccurs="100" maxOccurs="100"
             type="long"/>
            <element name="y" minOccurs="100" maxOccurs="100"
             type="long"/>
            <element name="z" minOccurs="100" maxOccurs="100"
             type="long"/>
          </complexType>
        </element>
      </complexType>
    </element>
  </tox-document>
</tox-template>
```

Ukážka 60: šablóna pre ToXgene, test 1

```
<xdoc>
  <settings>
    <output outfile="out\moj.xml" module="" schemeType="dtd" />
    <secondoutput module="" />
  </settings>
  <mainelement name="root" mincount="1" maxcount="1">
```

```
    <element name="a" mincount="10000" maxcount="10000">
      <element name="x" mincount="100" maxcount="100" type="long"
         module="incrementNumber">
        <setting step="1"  startnum="10000000" />
      </element>
      <element name="y" mincount="100" maxcount="100" type="long"
         module="incrementNumber">
        <setting step="1"  startnum="10000000"/>
      </element>
      <element name="z" mincount="100" maxcount="100" type="long"
         module="incrementNumber">
        <setting step="1" startnum="10000000" />
      </element>
    </element>
  </mainelement>
  <datatypes />
</xdoc>
```

Ukážka 61: šablóna pre XMLGenerator, test 1

Výsledky testu boli:

XML Generator

Číslo testu	Čas (v sekundách)	Veľkosť súboru V Bytoch	parametre modulu
1	14,18	62836763	startnum="0", step="1"
2	14,01	62836763	startnum="0", step="1"
3	14,13	62836763	startnum="0", step="1"
4	13,96	62836763	startnum="0", step="1"
5	15,52	75170093	startnum="1000000000", step="1"
6	14,85	75170093	startnum="1000000000", step="1"
7	14,71	75170093	startnum="1000000000", step="1"
7	15,86	87471323	startnum="10000000000", step="12345678
7	15,48	87471323	startnum="10000000000", step="12345678
7	15,51	87471323	startnum="10000000000", step="12345678

Tabuľka 1: Výskedky merania XML Generatora, test 1

ToXgene

Číslo testu	Čas (v sekundách)	Veľkosť súboru v
1	294,08	81857476
2	296,03	81857855
3	293,42	81857822

Tabuľka 2: Výsledky merania ToXgene, test 1

Test 2

```
<xdoc>
  <settings>
    <output outfile="out\moj.xml" module="" schemeType="dtd" />
    <secondoutput module="" />
  </settings>
  <mainelement name="root" mincount="1" maxcount="1">
    <element name="a" mincount="10000" maxcount="10000">
```

```
      <element name="e1" type="year" module="enumenator"
         mincount="100" maxcount="100">
        <setting order="sequence" />
      </element>
      <element name="e2" type="year" mincount="100"
         maxcount="100" module="enumenator">
        <setting order="sequence" />
      </element>
      <element name="e3" type="year" mincount="100"
         maxcount="100" module="enumenator">
        <setting order="sequence" />
      </element>
    </element>
  </mainelement>
  <datatypes>
    <datatype name="year" useenumeration="1" parentType="string">
      <enumerations>
        <enumeration>1942</enumeration>
      </enumerations>
    </datatype>
  </datatypes>
</xdoc>
```

Ukážka 62: šablóna pre XMLGenerator, test 2

```
<tox-template>
  <simpleType name="year">
    <restriction base="string">
      <tox-value>1942</tox-value>
    </restriction>
  </simpleType>
  <tox-document name="moj3">
    <element name="root">
      <complexType>
        <element name="a" minOccurs="10000" maxOccurs="10000" >
          <complexType>
            <element name="e1" minOccurs="100" maxOccurs="100"
type="year" />
            <element name="e2" minOccurs="100" maxOccurs="100"
type="year" />
            <element name="e3" minOccurs="100" maxOccurs="100"
type="year" />
          </complexType>
        </element>
      </complexType>
    </element>
  </tox-document>
</tox-template>
```

Ukážka 63: šablóna pre ToXgene, test 2

Číslo testu	Čas (v sekundách)	Veľkosť súboru v
1	11,45	63170093
1	11,31	63170093
1	11,35	63170093
1	11,73	63170093

Tabuľka 3: Výskedky merania XML Generatora, test 2

Číslo testu	Čas (v sekundách)	Veľkosť súboru v
1	289,81	42080135
2	291,34	42080135
3	290,66	42080135

Tabuľka 4: Výsledky merania ToXgene, test 2