VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO VYTVÁŘENÍ TESTOVACÍCH VSTUPŮ V XML

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN OČENÁŠ
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO VYTVÁŘENÍ TESTOVACÍCH VSTUPŮ V XML

TOOL FOR GENERATING XML INPUTS FOT TESTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN OČENÁŠ
AUTHOR

VEDOUCÍ PRÁCE Ing. ALEŠ SMRČKA, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Cílem této bakalářské práce je představit nástroj pro generování testovacích vstupů ve formátu XML. Nástroj se u generovaného XML dokumentu zaměřuje na různé kombinace elementů, atributů a jejich hodnot. Uživatel nástroje má možnost volby 3 různých scénářů generování XML dat, každá z nich vyžaduje jinou úroveň znalosti.

Abstract

The aim of this bachelor's thesis is to present a tool for automatic generating a set of XML inputs for testing from a given XML scheme. While generating a XML document, the tool aims at different combinations of elements, attributes, and their values. From a user perspective, three scenarios of generating XML documents are provided, each of which requires different level of expertise.

Klíčová slova

XML, XSD, testování založené na syntaxi, testovací vstupy, PHP, generování vstupů

Keywords

XML, XSD, syntax based testing, test inputs, PHP, generating inputs

Citace

Martin Očenáš: Nástroj pro vytváření testovacích vstupů v XML, bakalářská práce, Brno, FIT VUT v Brně, 2015

Nástroj pro vytváření testovacích vstupů v XML

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doktora Aleše Smrčky.

.....
Martin Očenáš
18. května 2015

Obsah

Kapitola 1

Úvod

Testování je běžná činnost při vývoji každého softwaru. Pro otestování softwaru je potřeba vytvořit sadu testů, které budou spouštět daný software s určitým nastavením a vyhodnocovat výsledky jeho chování. Cílem testů je systematicky spouštět všechny části software a ověřit, že fungují podle specifikace. Pokud nefungují podle specifikace pak je účelem testů objevit a lokalizovat vadu. Vytváření takovýchto testů je práce, kterou musí testeři vykonávat, ve většině případů, manuálně.

Myšlenkou tohoto projektu je vytvořit nástroj, který dokáže vygenerovat testovací hodnoty, které bude možné využít jako vstupy pro testování programů. Takovýto nástroj by dokázal testerům výrazně usnadnit práci při testování systému jako celku. Pro každý testovací vstup by sice bylo nutné manuálně vytvořit odpovídající výstup, ale i přesto by se jednalo o výrazné usnadnění. Sada automaticky vygenerovaných testovacích vstupů je i bez znalosti korektních výstupů, schopna odhalit stavy, které vedou k nekorektnímu chování nebo v nejhorším případě pádu programu.

Cílem tohoto projektu je vytvořit nástroj, který bude generovat testovací vstupy ve formátu XML podle zadaného XML schematu (XSD). Uživatelem v tomto případě bude tester, který potřebuje otestovat svoji aplikaci, která jako vstup používá XML dokumenty. Cílem je vygenerovat XML testovací vstupy, které budou obsahovat všechny potenciálně nebezpečné hodnoty, které by mohly způsobit chybu v testovém systému pro dané XSD.

Pro různé potřeby testera má nástroj možnost, generovat různé úrovně výstupů:

Jeden testovací soubor Cílem je poskytnout testerovi jeden XML soubor, který odpovídá vloženému XSD a který je možné použít pro ověření základní funkčnosti testovaného systému

Sada testovacích souborů Cílem je poskytnout testerovi sadu testovacích XML souborů, které obsahují různé, potenciálně nebezpečné hodnoty a které všechny by měl testovaný systém zvládnout korektně zpracovat.

Sada souborů s dodatečným nastavením Cílem je poskytnout testerovi sadu testovacích XML souborů, ale nabídnout možnost blíže specifikovat obsah jednotlivých XML elementů.

V kapitole 2 jsou stručně popsány základy technologií XML, XSD a testování založeného na syntaxi, které jsou důležité pro vznik nástroje. V kapitole 3 je navržen a popsán princip fungování nástroje, jeho architektura, princip a postup generování testovacích hodnot. Kapitola 4 obsahuje popis důležitých části implementace vzniklého programu, úskalí, které bylo nutné při implementaci programu řešit a popis testování programu.

Kapitola 2

Současný stav aplikací využívající jazyk XML

Následující kapitola stručně popisuje technologie důležité pro tento projekt. Jedná se zejména o technologie XML a XSD, které tvoří podstatu projektu, dále pak část o testování založeném na syntaxi, které je základem toho, jak vytvářet výstupní hodnoty nástroje.

2.1 Jazyk XML

XML (Extensible Markup Language) je standardní, značkovací jazyk určený pro popis dat. Je navržen tak, aby byl obecný a aby bylo možné jej použít pro popis libovolných, serializovaných dat. Důvodem pro vznik XML byla potřeba strukturovaného formátu dat, který by byl čitelný pro člověka i počítač. XML je tedy zapisováno v textové podobě, čitelné člověkem. Pro specifikování významu jednotlivých částí textu jsou použity tzv. značky, které jsou snadno zpracovatelné i počítačem.

2.1.1 Formát XML

Pokud máme text, který obsahuje ucelená XML data, pak hovoříme o XML dokumentu (přesnější definice bude uvedena v kapitole 2.1.3). XML dokument je tvořen tzv. elementy, které se do sebe hierarchicky zanořují a tvoří tak hierarchicky strukturovaná data. Elementy jsou zapisovány pomocí značek, ty jsou zapisovány pomocí symbolů $<$ a $>$. Jedna značka sama o sobě může popisovat celý element, nebo může znamenat pouze začátek nebo konec elementu.

2.1.2 Zápis XML elementu

Element je základní složkou XML dokumentů, každý element by měl popisovat jeden objekt daného systému, nezávisle na tom, jestli se jedná o konkrétní elementární objekt nebo o abstrakci nad více objekty. V XML se může každý element skládat z atributů a obsahu, kde atributy jsou uvedeny ve značce, uvozující element a obsah je uveden mezi uvozující a ukončující značkou. Atributy jsou zapsány ve formátu *nazev-atributu="hodnota-atributu"*. Hodnota atributu se vždy vztahuje pouze k elementu, u nějž je zapsán. Standard XML rozděluje elementy podle způsobu zápisu na párové a nepárové.

Nepárové elementy jsou tvořeny jednou značkou, která element současně uvozuje i ukončuje. Značka, označující nepárový element, vždy obsahuje symbol / před ukončujícím symbolem

>. Volitelně může nepárový element obsahovat také atributy. Nepárový element pak může vypadat například takto:

```
<auto znacka="skoda"/>
```

Párové elementy jsou tvořeny uvozující a ukončující značkou, vše mezi těmito značkami tvoří obsah elementu. Uvozující značka má tvar <*Jméno-elementu*> a ukončující <*/Jméno-elementu*>, případné atributy jsou obsaženy v uvozující značce. Párového elementu autosalon bez atributů pak může vypadat například takto:

```
<autosalon>
        <auto znacka="skoda"/>
        <auto znacka="volvo"/>
</autosalon>
```

2.1.3 XML dokument

XML dokument je ucelená část XML dat, která se skládá z XML hlavičky a jednoho kořenového XML elementu. V hlavičce (xml declaration) musí být uvedená verze XML, volitelně použité kódování a informaci, jestli smí být dokument použit bez schématu. Příklad hlavičky:

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

Každý XML dokument musí obsahovat právě jeden kořenový element (tj. element, který není obsahem žádného jiného elementu). V tomto kořenovém elementu jsou obsaženy všechny ostatní elementy daného dokumentu. Příklad XML dokumentu:

```
<?xml version="1.0" encoding="UTF-8"?>
<mesto>
        <ulice jmeno="Masarykova">
                <autosalon>
                        <auto znacka="skoda"/>
                        <auto znacka="volvo"/>
                </autosalon>
        </ulice>
</mesto>
```

Standard XML definuje několik pravidel pro pojmenovávání elementů:

- Ve jménech elementů záleží na velikosti písmen
- Jméno elementu musí začínat písmenem nebo podtržítkem
- Jméno elementu nesmí začínat sekvencí "XML", v jakékoli kombinaci velikosti písmen
- Jméno elementu může obsahovat písmena, číslice, pomlčky, podtržítka a tečky
- Jméno elementu nesmí obsahovat mezery

XML standard definuje, že zanořené elementy musí být ukončeny přesně v opačném pořadí, než byly uvozeny. Následující zápis tedy není v XML možný:

```
<a><b>Some text</a></b>
```

Správně by taký text musel být zapsán:

```
<a><b>Some text</b></a>
```

2.2 XML schéma

XSD (XML schéma) je XML dokument s definovanou strukturou, který specifikuje možný obsah jiného XML dokumentu. Běžně je tedy vytvořen XSD dokument, který specifikuje, jaké elementy s jakým obsahem budou obsaženy v XML dokumentu. Standard XML sám o sobě nedefinuje obsah XML dokumentu, obecně je tedy možné v XML dokumentu vytvořit libovolné elementy s libovolným obsahem. Pokud je potřeba přesně definovat strukturu elementů v XML dokumentu, pak může být použito XSD, které tuto strukturu může přesně definovat. Další výhodou XSD je možnost strojově vyhodnotit, jestli určitý XML dokument vyhovuje danému XSD.
Vzorový XSD soubor je umístěn v příloze B.

2.2.1 Forma XSD

XSD dokument je sám o sobě také XML dokument a musí splňovat jeho pravidla, na rozdíl od XML jsou však v XSD definované elementy, které může obsahovat. Kořenovým element XSD je vždy element *schema*, který definuje strukturu XML dokumentu na kořenové úrovni, veškeré elementy, specifikované přímo v elementu *schema*, mohou být v XML dokumentu použity jako kořenové elementy.

2.2.2 Datový typ elementu

XSD vyžaduje ke každému elementu, kromě názvu, specifikovat i datový typ. Datový typ specifikuje atributy elementu, jeho obsah, možné podelementy a jestli je element párový nebo ne. Datové typy jsou v XSD rozděleny na dvě základní kategorie, jednoduché datové typy (*Simple type*) a komplexní datové typy (*Complex type*). Elementy jednoduchých datových typů nemohou mít atributy, jsou vždy párové a nemohou obsahovat žádné podelementy. Komplexní datové typy mohou obsahovat atributy, podelementy i vlastní text, mohou být také nepárové.

V XSD je možné využít vestavěné datové typy (viz. 2.2.3) a také definovat vlastní datové typy. Vytváření vlastních datových typů se věnuje kapitola 2.2.4.

2.2.3 Vestavěné datové typy

XSD definuje několik vestavěných datových typů, které je možné využít jako datové typy elementů. Jedná se pouze jednoduché datové typy. Ze základních typů jsou to například:

- **integer** - Celočíselný datový typ
- **decimal** - Datový typ pro desetinná čísla
- **float** - Datový typ pro čísla s plovoucí řádovou čárkou
- **boolean** - Datový typ pro booleovké hodnoty
- **string** - Datový typ pro libovolný řetězec
- **duration** - Datový typ specifikující časový interval
- **dateTime** - Datový typ pro určení data a času

Kompletní výčet vestavěných datových typů, včetně specifikace, lze nalézt na stránkách W3C [9]

2.2.4 Vytváření datových typů

XSD umožňuje autorovi dokumentu definovat vlastní datové typy. Datový typ může být definován přímo pod kořenových elementem XSD, potom je vytvořený datový typ globální a je možné jej využít pro kterýkoli element v daném XSD, v tomto případě musí být nově vzniklý datový typ pojmenovaný. Další možností je specifikace datového typu přímo v elementu, ve kterém bude použit, v takovém případě nesmí být pojmenovaný a může být použit pouze pro element, ve kterém je definován. Touto možností specifikace datových typů se více zabývá kapitola 2.2.5. Pravidla pro vytváření datových typů se liší pro jednoduché a komplexní datové typy, proto budou popsány zvlášť.

Jednoduché datové typy vznikají vždy odvozením od existujícího jednoduchého datového typu a to omezením možných hodnot rodičovského typu. Možná omezení jsou definována ve standardu XSD [9] a mohou se lišit pro různé datové typy. Obvyklá omezení jsou: maximální délka/hodnota, minimální délka/hodnota, určitá délka, regulární výraz, kterému musí výsledná hodnota odpovídat nebo přímo výčet hodnot, které mohou být ve výsledném datovém typu obsaženy. Na uvedeném příkladu je vytvořen nový datový typ *counter*, odvozený od datového typu *decimal* s omezením možných hodnot na interval <0;100).

```
<simpleType name="counter">
        <restriction base="decimal">
                <minInclusive value="0"/>
                <maxExclusive value="100"/>
        </restriction>
</simpleType>
```

Komplexní datové typy mohou vznikat více způsoby:

- Vytvořením zcela nového komplexního typu.
- Odvozením od existujícího typu rozšířením nebo omezením existujícího komplexního typu.
- Vytvořením nového komplexního datového typu, přičemž obsah elementu bude tvořen jednoduchým datovým typem.

Při vytvoření nového komplexního typu je implicitně nastaveno, že obsahem tohoto typu mohou být pouze další elementy, další možností je, že obsahem bude pouze text daný jednoduchým datovým typem a poslední možností je, že obsah bude tvořen kombinací elementů a textu. Pro vytvoření komplexního datového typu s podelementy a možným dalším textem, je třeba vytvořit komplexní typ s obsahující podlementy a v definici komplexního typu přidat atribut *mixed="true"*.

V následujících příkladech budou vytvořeny komplexní datové typy výše popsanými způsoby. Vytvoření zcela nového komplexního datového typu:

```
<complexType name="auto">
        <complexContent>
                <element name="ridic" type="osoba"/>
        </complexContent>
        <attribute name="pocet_sedadel" type="integer"/>
</complexType>
```

Vytvoření komplexního datového typu odvozením od existujícího:

```
<complexType name="auto_se_spolujezdcem">
        <complexContent>
                <extension base="auto">
                        <element name="spolujezdec" type="osoba"/>
                </extension>
        </complexContent>
</complexType>
```

Vytvoření komplexního datového typu s jednoduchým obsahem:

```
<complexType name="Kniha">
        <simpleContent>
                <extension base="string">
                        <attribute name="autor" type="string"/>
                        <attribute name="pocet_stran" type="integer"/>
                <extension>
        <simpleContent>
</complexType>
```

2.2.5 Definice elementu

Definice elementu v XSD udává, že ve výsledném XML dokumentu bude v tomto místě umístěn element. Pokud je element v XSD definován jako podelement *schema*, pak ve výsledném XML dokumentu může být použit jako kořenový element. Obdobně jako u definice datových typů platí, že elementy definované přímo pod kořenovým elementem XSD jsou globální a mohou být odkazovány kdekoli z daného XSD.

Pro každý element musí být specifikováno jméno a datový typ, to lze udělat několika způsoby:

- Přímým zadáním jména a datového typu:

  ```
  <element name="country" type="string"/>
  ```

- Odkazem na jiný element, který definuje jméno a datový typ:

  ```
  <element ref="country"/>
  ```

- Odkazem na jiný element, který definuje datový typ, jméno je zadáno přímo:

  ```
  <element name="zeme" substitutionGroup="country"/>
  ```

  Tento přístup se využívá například pro možnost používat stejný element, kde jméno elementu může být používáno ve více jazycích

2.2.6 Další možnosti XSD

Kromě výše zmíněných možností umožňuje XSD definovat a odkazovat i atributy, skupiny atributů a skupiny elementů. Umožňuje využít elementy *any* a *anyAttribute*, které umožňují na svoji pozici v XML dokumentu vložit libovolný element resp. libovolný atribut. XSD umožňuje použití indikátorů, které specifikují pořadí podelementů, nebo možnosti opakování určitých podelementů. Podrobnější popis XSD dokumentů je možné nalézt na stránkách W3school [10]

2.3 Testování založené na syntaxi

The essential characteristic of syntax-based testing is that a syntactic description such as a grammar or BNF is used. [1, p.170] Testovaní založené na syntaxi je oblast testování, která se zabývá testováním strukturovaných vstupních dat, jejichž formát je určitým způsobem popsán a je možné vstupní data validovat podle daného popisu. Vstupní data mohou být popsána mnoha způsoby, například:

- formální gramatika,
- regulární výraz,
- formátovací řetězec,
- DTD, XML schema.

Obecně mohou být vstupní data popsána i neformálně, ale pro účely testování založeného na syntaxi, je třeba tento popis formalizovat. Cílem formalizace popisu vstupních dat je možnost, pro všechna možná vstupní data jednoznačně rozhodnout, zda jsou validní nebo ne a jak by měl testovaný systém reagovat.

2.3.1 Kritéria pokrytí

Coverage criterion (kritérium pokrytí) je pravidlo nebo předpis pro systematické generování požadavků na test.[6, p.5] U použitého kritéria pokrytí je možné měřit míru, s jakou současné testy splňují dané kritérium. Kritéria jsou specifikována před vytvářením testů a následně jsou vytvářeny testy tak, aby pokryly dané kritérium v co největší míře.

Formální popis vstupních dat umožňuje, mimo jiné, specifikovat kritéria pokrytí pro vstupní data. Pro bezkontextové gramatiky existují např. dvě kritéria[7]:

- pokrytí všech terminálních symbolů,
- pokrytí všech přepisovacích pravidel.

Uvedená kritéria pokrytí lze snadno transformovat na jiné druhy popisu vstupních dat, jako např.: jiné gramatiky, XML schema, DTD dokument apod.

2.3.2 Validace dat

Možnost rozlišit mezi validními a neplatnými vstupními daty je velice důležitá pro tvorbu testovacích případů a vyhodnocování výsledků testů. Předpokladem pro korektní validaci vstupních dat je přesný a jednoznačný popis vstupních dat (viz. 2.3). Přesný popis je důležitý zejména pro testovaní dat, která jsou blízko hranici mezi validními a nevalidními daty.

Pokud je využit určitý formát popisu vstupních dat, pak je možné provádět strojovou validaci dat. Vhodné formáty popisu jsou zejména gramatiky, regulární výrazy, XML schémata, DTD dokumenty nebo formátovací řetězce, tyto formáty jsou obecně známé a rozšířené a jsou dostupné nástroje pro jejich strojové zpracování.

Pro některé formáty popisu je možné vstupní data i strojově generovat. Pro strojové generování vstupních dat jsou vhodné zejména bezkontextové gramatiky, kde je možné pro generování systematicky využívat jednotlivá derivační pravidla a vygenerovat tak testovací data, která pokryjí všechny možné derivační pravidla dané gramatiky.

2.3.3 Mutační testování

Mutační testování [7] je technika, jejíž podstatou je změna validních vstupních dat tak, aby vznikla data neplatná. Testovací případ, ve kterém byla takováto mutace provedena, musí skončit selháním, pokud se tak nestane, pak je v testovaném systému chyba.

Pro mutační testování je vhodný popis pomocí gramatik, kde je možné snadno pozměnit terminální symboly nebo derivační pravidla a následně podle upravené gramatiky generovat testovací data.

2.4 Kombinační testování

Kombinační testování je typ definice testovacích kritérií, kdy cílem testu obecně není otestovat jeden prvek v programu, ale kombinaci dvou a více takových prvků. Těmito prvky mohou být větve v programu, části podmínek, argumenty spuštění programu apod. Cílem testů potom není otestovat všechny tyto elementy, ale všechny možné kombinace dvou, nebo více, těchto prvků.

Při použití kombinačního testování roste šance na odhalení chyby podle toho, kolik prvků je zahrnuto do jedné kombinace. Zároveň také velmi výrazně roste počet potřebných testovacích případů.

Testovací vstupy pro kombinační testování je možné, při znalosti zdrojového kódu, strojově generovat. Složitost takovýchto generátorů je však také výrazně větší a z toho důvodu nebude kombinační testování v této práci dále uvažováno.

Kapitola 3

Návrh generátoru testovacích dat

Tato kapitola specifikuje požadavky na vytvořený program, dále popisuje návrh a princip fungování nástroje, pro generování XML testovacích vstupů. V části 3.4 jsou dále popsány principy generování hodnot jednoduchých datových typů.

3.1 Definice pojmů

V následující částech textu budou využívány následující pojmy s daným významem:

Vnitřní reprezentace Reprezentace vstupních nebo výstupních dat pomocí objektů v programu,

vstupní element jeden konkrétní XML element v XML schématu, zadanému na vstup programu,

výstupní element jeden Konkrétní XML element obsažený ve výstupním XML dokumentu,

podelement XML element, který je obsažen uvnitř jiného XML elementu,

testovací doména/dimenze část oboru hodnot jednoho datového typu, sdružující hodnoty které jsou si sémanticky blízké a je pravděpodobné, že chyby odhalené těmito hodnotami budou stejné bez ohledu na to, která konkrétní hodnota je použita,

dodatečná nastavení specifikace použitých dat a omezení v jednotlivých výstupních elementech, která nejsou specifikována ve vstupním XSD dokumentu, ale volitelně je nastavuje uživatel,

nastavení testů uživatelem specifikované chování nástroje, určuje kolik testovacích souborů bude obsaženo ve výstupu a jestli bude uživatel dotázán na dodatečná nastavení,

cesta k elementu řetězec, který jednoznačně identifikuje element nebo atribut elementu ve stromu XSD objektů. Princip vytváření cesty je popsán v kapitole 3.3.4.

3.2 Specifikace požadavků

Cílem projektu je vytvořit nástroj, který bude generovat XML dokumenty, které budou validní vzhledem k danému XSD dokumentu. Tyto XML dokumenty by měly pokrývat

všechny potenciálně nebezpečné vstupy, které by mohli způsobit selhání testovaného systému a které je možné vytvořit vzhledem k danému XSD. Navržený nástroj bude splňovat tyto požadavky:

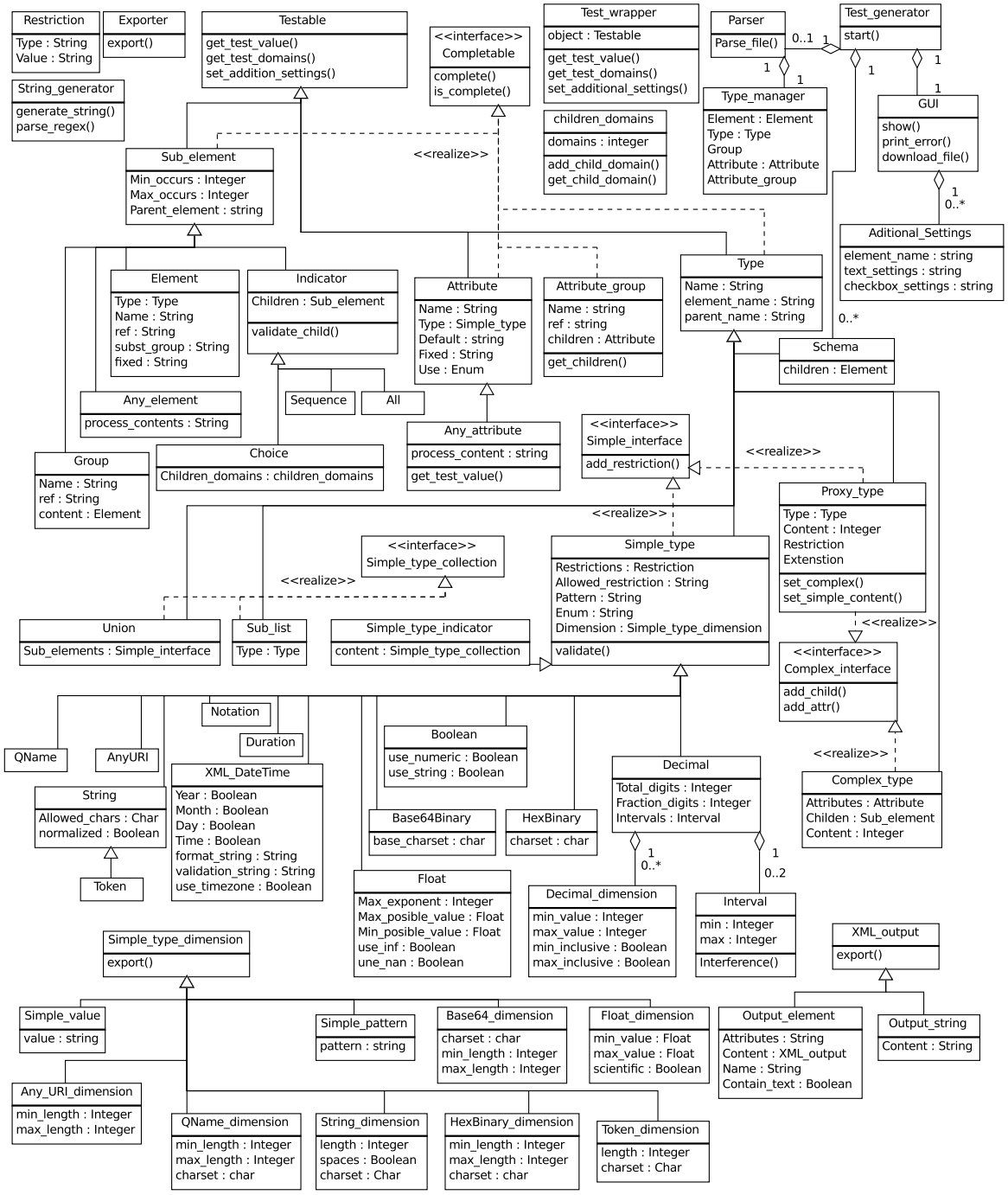
- validace vstupního XML schématu,
- odhalení vstupní omezení, které není možné naplnit,
- podpora generování testovacích hodnot pro všechny jednoduché datové typy uvedené v [9],
- možnost vygenerovat jeden XML dokument,
- možnost vygenerovat sadu XML dokumentů bez nutnosti zadávat další nastavení,
- možnost specifikovat nastavení, která nejsou zadaná XML schématem při generování XML dokumentů,
- možnost exportovat a importovat dodatečné nastavení,
- rozdělení oboru hodnot jednoduchých datových typů do tříd ekvivalence,
- vygenerování minimálně jedné hodnoty pro každou třídu ekvivalence každého elementu,
- rozdělení jednotlivých XML dokumentů do samostatných souborů,
- v případě vytvoření více souborů s testovacími daty, zabalení souborů do ZIP archivu.

3.3 Návrh programu

Tato sekce popisuje návrh samotného programu. Návrh je vytvořen pro jakýkoli objektově orientovaný, imperativní, programovací nebo skriptovací jazyk.

3.3.1 Architektura systému

Systém je tvořen podle paradigmatu třídního, objektově orientovaného programování. Vytvořené objekty jsou formovány do stromové struktury, kde kořenem této struktury je řídicí objekt programu. Podstromem této stromové struktury jsou objekty reprezentující data obsažené ve vstupním XSD dokumentu. Příklad vytvořené struktury objektů je obsažen v kapitole 3.3.1. Hierarchii použitých tříd popisuje následující diagram tříd:



Obrázek 3.1: Architektura systému

Následuje seznam jednotlivých tříd a jejich popis:

Aditional_settings Třída pro popis dodatečných nastavení. Každá instance uchovává dodatečná nastavení pro jeden element nebo atribut.

All Reprezentuje XSD indikátor All. Indikátor All definuje, že ve výstupu musí být vždy přítomny všechny jeho podelementy a to v libovolném pořadí.

Any_attribute Třída pro libovolný atribut. Generuje atributy s náhodným jménem a hodnotou.

Any_element Třída pro libovolný element. Generuje elementy s náhodným jménem i obsahem. Vygenerovaný element může být jednoduchého nebo komplexního datového typu a může obsahovat podelementy.

Attribute Třída pro popis atributu. Instance třídy reprezentuje jeden konkrétní atribut. Uchovává zejména jméno a datový typ atributu.

Attribute_group Třída pro popis skupiny atributů. Instance popisuje jednu konkrétní skupinu atributů. Skládá se zejména z pole jednotlivých atributů a názvu.

Completable Rozhraní pro všechny třídy, jejichž objekty jsou kompletovány ve více fázích (viz. 3.3.2). Specifikuje metody pro provedení kompletace.

Complex_type Reprezentuje komplexní datový typ. Každá instance reprezentuje jeden konkrétní datový typ, použitý u jednoho konkrétního elementu. Uchovává jméno, pole podelementů a pole atributů.

Complex_interface Rozhraní pro třídy, které mohou reprezentovat komplexní datové typy.

Children_domains Pomocná třída pro generování dat. Uchovává seznam počtů možných testovacích domén pro nastavené objekty. Dokáže rozdělit číslo generované testovací domény na index testovaného objektu a číslo testovací domény.

Choice Reprezentuje XSD indikátor Choice. Indikátor Choice definuje, že ve výstupu musí být vždy přítomen právě jeden z jeho podelementů.povolena

Element Třída pro popis XML elementu. Instance reprezentuje jeden výstupní XML element. Uchovává zejména informace o jménu elementu a datovém typu.

Exporter Knihovní třída (library class) která slouží pro exportování výstupních dat do souboru.

Group Třída pro skupiny elementů. Instance reprezentuje jednu skupinu elementů, definovanou ve vstupním XSD dokumentu. Uchovává zejména pole elementů a název skupiny.

GUI Třída pro komunikaci s uživatelem. Zpracovává vstupy od uživatele a posílá uživateli výstupy programu. Zajišťuje také získání a zpracování dodatečných nastavení.

Indicator Abstraktní třída reprezentující XSD indikátory. Indikátory tvoří obálku nad skupinou elementů a udávají kolik elementů a v jakém pořadí mohou být zobrazeny. Uchovává zejména seznam podelementů.

Output_element Třída pro výstupní elementy. Instance reprezentuje XML element, který již byl vygenerován a bude umístěn do výstupu. Uchovává jméno elementu, vygenerovaný obsah a atributy. Definuje metodu pro serializaci elementu.

Output_string Třída pro výstupní řetězec. Instance reprezentuje text, který bude umístěn do výstupu.

Parser Analyzátor vstupu. Validuje a následně parsuje vstupní XSD dokument a převádí jeho obsah do vnitřní reprezentace.

Proxy_type Třída pro dočasné reprezentace datových typů. Reprezentuje datový typ, který byl definován, ale ještě nejsou definovány všechny jeho součásti. Instance této třídy slouží jako dočasní zástupci konečného datového typu. Ve fázi kompletace jsou všechny instance této třídy odstraněny. Více o použití této třídy je popsáno v kapitole 3.3.3.

Restriction Přepravka (crate) pro uložení dat o jednom konkrétním omezení pro jeden konkrétní datový typ.

Schema Reprezentuje XSD objekt Schema, který je kořenem XSD dokumentu. Vnitřní reprezentace Schema se chová jako komplexní datový typ, kde všechny jeho podelementy jsou potenciální kořenové elementy výstupního XML dokumentu. Uchovává zejména pole podelementů.

Sequence Reprezentuje XSD indikátor Sequence. Indikátor Sequence definuje, že mohou být zobrazeny libovolné jeho podelementy, kde počet zobrazení je definován atributy elementů *minOccurs* a *maxOccurs*, pořadí podlementů je závazně definováno jejich pořadím v XSD dokumentu a přesně v tomto pořadí musí být i na výstupu.

Simple_interface Rozhraní pro třídy, které mohou reprezentovat jednoduché datové typy.

Simple_type Abstraktní třída která je nadtřídou pro všechny jednoduché datové typy. Tvoří kostru pro zpracování vstupů a generování výstupů pro všechny jednoduché datové typy. Jednotlivé jednoduché datové typy a postup generování jejich hodnot jsou popsány v kapitole 3.4.

Simple_type_collection Rozhraní pro objekty tříd *Union* a *Sub_list*. Existuje pouze kvůli typové kontrole.

Simple_type_dimension Abstraktní třída která je nadtřídou pro všechny třídy, které reprezentují jednu testovací dimenzi pro generování výstupních dat. Deklaruje metodu pro vygenerování jedné konkrétní výstupní hodnoty. Popis jednotlivých dimenzí je uveden v kapitole 3.4.

Simple_type_indicator Tvoří obálku nad objekty tříd *Union* a *Sub_list*, funguje jako běžný jednoduchý datový typ

String_generator Knihovní třída (library class) pro generování náhodných řetězců. Zvládá generovat zcela náhodné řetězce nebo řetězce podle regulárních výrazů.

Sub_element Abstraktní třída která je nadtřídou pro všechny třídy, které reprezentují vstupní XSD tag, který může být obsahem XSD elementu.

Sub_list Reprezentuje XSD objekt *List*. Sub_list vytváří indikátor nad jednoduchým datovým typem, který specifikuje, že daný datový typ může být ve výstupu několikrát opakován.

Testable Abstraktní třída která je nadtřídou pro všechny třídy, které generují výstupní data. Deklaruje metody pro generování výstupních dat.

Test_generator Hlavní třída celého programu. Instance třídy řídí celý program. Přímo vytváří a řídí objekty tříd Parser a GUI, také řídí proces generování výstupních dat.

Test_wrapper Instance této třídy slouží jako obálka nad testovatelným objektem. Jejím účelem je umožnit snadnější přístup ke generovaným datům pro rodičovský objekt. Více v kapitole 3.3.5.

Type Abstraktní třída která je nadtřídou pro všechny třídy, které reprezentují datové typy.

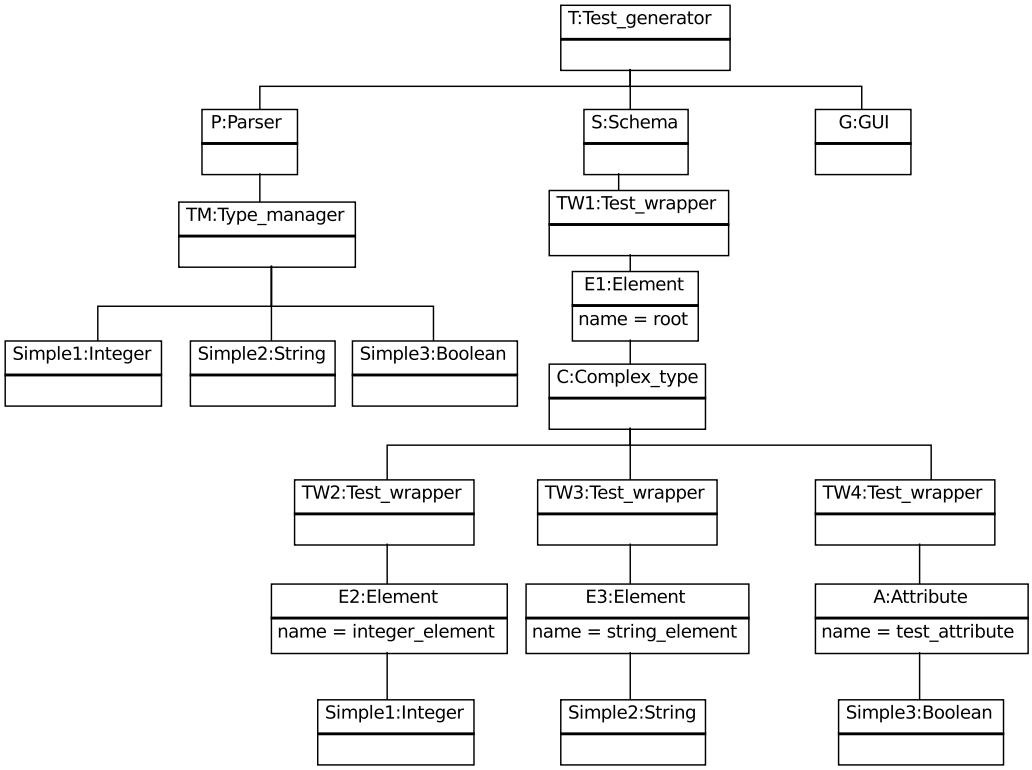
Type_manager Správce datových typů, návrhový vzor jedináček(singleton). Uchovává všechny pojmenované datové typy, elementy, atributy, skupiny atributů a skupiny elementů, které jsou definovány ve vstupním XSD na globální úrovni a vestavěné datové typy. Ostatním částem programu vydává kopie těchto objektů, identifikované názvem.

Union Reprezentuje XSD objekt *Union*. Union vytváří indikátor nad jednoduchými datovými typy, který specifikuje, že ve výstupu bude použit právě jeden z obalovaných datových typů.

XML_output Abstraktní třída která je nadtřídou pro všechny třídy, které reprezentují vygenerovaná data. Deklaruje metodu pro serializaci objektu.

Příklad vytvořené hierarchie objektů

Následující diagram zobrazuje příklad hierarchie objektů, pro jednoduchý vstupní XSD soubor:



Obrázek 3.2: Příklad objektů v programu

3.3.2 Fáze běhu programu

Postup programu se částečně odvíjí od zadaných vstupů, ty se skládají z XSD dokumentu, nastavení testů a povolení mutačního testování. Běh programu a generování výstupních dat probíhá v několika základních fázích:

1. zpracování vstupu 3.3.3,
2. (pouze při povolení dodatečných nastavení) zjištění a aplikování dodatečných nastavení 3.3.4,
3. generování testovacích hodnot 3.3.5,
4. generování výstupu 3.3.6.

Každé z těchto fází se detailněji věnuje jedna z následujících podkapitol.

3.3.3 Zpracování vstupu

Zpracování vstupu je fáze, ve které je vstupní XSD dokument převeden do vnitřní reprezentace objektů. V programu tuto činnost zajišťuje objekt třídy *Parser*. Samotné zpracování je rozděleno do dvou fází:

1. analýza vstupního dokumentu,
2. kompletace vstupních prvků.

Obě tyto fáze zajišťuje vytvořený program. Před zahájením analýzy je vhodné vstupní XSD dokument validovat pomocí knihovních funkcí, pokud jsou v daném programovacím jazyce k dispozici. Výsledkem zpracování vstupu je strom objektů, odpovídající datům v XSD dokumentu, kořenem tohoto stromu je objekt třídy *Schema*. Jako výsledek zpracování je navrácen ukazatel na kořenový objekt.

Analýza vstupního dokumentu

V této fázi program prochází vstupní XSD dokument a vytváří programové objekty, odpovídající XSD elementům. Do vnitřní reprezentace jsou převedeny pouze objekty, které mají význam pro generování odpovídajícího výstupu, nedůležité tagy nebo atributy jsou ignorovány.

Před zahájením fáze analýzy je vytvořen objekt třídy *Type_manager* (správce typů), v průběhu analýzy jsou do něj umisťovány všechny pojmenované XSD objekty, které jsou přímými potomky kořenového tagu *schema*.

V rámci XSD je možné se v definici objektů názvem odkazovat na jiný objekt a je možné aby odkazovaný objekt nebyl dosud deklarován a byl definován až v dalších částech XSD dokumentu. Z toho důvodu je nutné odkazované objekty převézt do vnitřní reprezentace pouze jako textové odkazy a s dalším propojením pokračovat až do dokončení analýzy celého vstupního XSD dokumentu.

V případě vytváření nových datových typů, které jsou odvozeny od jiných typů, které v daném okamžiku ještě nebyly definovány, je použit objekt třídy *Proxy_type*. Tento objekt uchovává záznam o názvu datového typu, od kterého je odvozen a zaznamenává všechny změny, které budou výsledný datový typ odlišovat od rodičovskému datovému typu. Tento objekt může být umístěn do správce typů a do fáze kompletace supluje výsledný objekt.

Po skončení fáze je analýzy, je možné, že celý strom XSD elementů bude zastupován pouze kořenovým objektem třídy *Schema* a všechny vnořené elementy budou pouze textově odkázány.

Kompletace vstupních prvků

V této fázi už není používán vstupní XSD dokument, pro provedení této fáze je použit objekt třídy *Schema* a objekty uložené ve správci typů.

Celá fáze je iniciována invokací metody *Complete* nad objektem *Schema*, jako argument této metody je předán ukazatel na správce typů. Metoda *Complete* je postupně rekurzivně volána na všechny potomky aktuálního objektu a vždy je jako argument předán ukazatel na správce typů. V rámci metody jsou u každého objektu získány objekty odkazované názvem (elementy, datové typy apod.) ze správce typů. Všechny objekty jako návratovou hodnotu metody *Complete* vrací ukazatel na zkompletovaný objekt.

Správce typů na vyžádání vydává objekty definované ve vstupním XSD dokumentu, identifikované názvem. Před navrácením objektu žadateli, zkontroluje správce typů jestli je

žádaný objekt kompletní, pokud ne provede jeho kompletaci a zkompletovaným objektem přepíše objekt původní, dále má uloženou pouze zkompletovanou verzi objektu. Žádajícímu objektu je následně vrácena kopie zkompletovaného objektu.

Ve fázi kompletace jsou postupně odstraněny objekty třídy *Proxy_type*. Objekty této třídy si při vlastní kompletaci vyžádají od správce typů instanci datového typu, na který odkazují a této instanci předají všechny změny oproti současnému stavu, které jsou v objektu uloženy. Jako návratovou hodnotu metody *Complete* je vrácen ukazatel na objekt cílového datového typu a objekt třídy *Proxy_type* je odstraněn.

Po skončení kompletace je dokončen a připraven strom objektů, reprezentující vstupní XSD data. Kořenem tohoto stromu je objekt třídy *Schema*.

3.3.4 Dodatečná nastavení

Dodatečná nastavení umožňují uživateli upřesnit některé věci, které nejsou přímo specifikovány ve vstupním XSD dokumentu. Dodatečná nastavení jsou použita pouze, pokud je nastavením testů povoleno jejich využití a současně vstupní XSD dokument obsahuje nespecifikovaná místa, které je možné specifikovat uživatelem.

Typy dodatečných nastavení

Dodatečná nastavení jsou rozdělena do dvou základních skupin: první jsou omezení zadaná textem, druhé jsou zadané pouze hodnotou ano nebo ne - typicky nastavena pomocí zaškrtávacích políček.

Omezení zadaná textem korespondují k omezením, které je možné k danému datovému typu zadat v XSD dokumentu:

- enumeration - výčet hodnot, kterých může daný datový typ nabývat,
- pattern - regulární výraz pro generování daného datového typu,
- minimální/maximální hodnota/délka - omezení specifické pro konkrétní datový typ.

Protože omezení typu *Enumeration* a *Pattern* vylučují možnost generování jakýchkoli jiných hodnot, tak při zadání jednoho z těchto omezení jsou veškerá další dodatečná nastavení ignorována. Primárně je použito omezení *Enumeration* a sekundárně *Pattern*.

Dodatečná nastavení zadaná pomocí zaškrtávacích políček obvykle upřesňují hodnoty, které mohou být použity v daném datovém typu. Pro tyto nastavení je typický datový typ *String*, kde jsou takto nastaveny znaky, které se mohou vyskytovat ve vygenerovaném řetězci. Konkretní nastavení jsou popsána u jednotlivých datových typů v kapitole 3.4.

Získání dodatečných nastavení

O dodatečná nastavení může požádat kterýkoli jednoduchý datový typ, obsažený v elementu nebo atributu. Daný datový typ vytvoří objekt třídy *Aditional_settings* ve kterém uvede cestu ke svému elementu/atributu a specifikuje dodatečná nastavení, které je mu možné nastavit. Cesta k elementu se skládá ze jmen všech nadelementů až ke kořenovému elementu oddělena znakem -. Cesta k elementu tak může vypadat například: *-root-element1-element2*. U atributu je nakonec přidán řetězec *.nazev_atributu*. Cesta k atributu vypadá například takto: *-root-element1-element2.attribute*. Při žádosti o dodatečná nastavení jsou uživateli zobrazovány cesty k jednotlivým elementům/atributům.

Pro možnost plošných nastavení je u každé žádosti o dodatečná nastavení uveden i textový název datového typu. Rozhraní, do kterého uživatel vyplňuje dodatečná nastavení, potom může nastavit dodatečná nastavení jednotně pro všechny objekty určitého datového typu.

Aplikace dodatečných nastavení

Po určení dodatečných nastavení uživatelem musí být tato nastavení analyzována a předána příslušným instancím datových typů. Pole všech dodatečných nastavení je předáno všem datovým typům v programu, každý datový typ si vezme nastavení příslušející jemu. Pro identifikaci cílového objektu je použita cesta, kterou datový typ uvedl při žádání o dodatečná nastavení.

3.3.5 Postup generování XML dat

Ve fázi generování jsou postupně vytvořeny konkrétní testovací data, odpovídající stromu objektů získaném ve fázi zpracování vstupu (viz. 3.3.3). Cílem je ve všech jednoduchých datových typech vytvořit testovací domény, tak aby pokrývali všechny potenciálně nebezpečné testovací hodnoty a následně vygenerovat tolik XML dokumentů, aby byly použity všechny vytvořené testovací domény všech datových typů. Tento proces se sestává z několika fází:

1. Vytvoření testovacích domén.
2. V případě povolených dodatečných nastavení, získání a aplikace dodatečných nastavení a znovuvytvoření testovacích domén.
3. Vygenerování dat.

Tyto jednotlivé fáze odpovídají metodám třídy *Testable*, každá fáze je zahájená řídícím objektem programu, zavoláním příslušné metody nad kořenem stromu XSD objektů, daná metoda je postupně hierarchicky volána nad všemi objekty stromu.

Vytvoření testovacích domén

Fáze vytvoření testovacích domén je iniciována metodou *get_test_domains* nad kořenovým objektem *Schema*. V této fázi jsou objektům postupně předávána Nastavení testů, informace o povolení nebo zakázání mutačního testování a objekt, do kterého mohou jednoduché datové typy ukládat požadavky na dodatečné nastavení. V průběhu procházení stromu touto metodou, jsou všem elementům a jejich datovým typům předávány názvy jejich rodičovských elementů a cesta k rodičovským elementům.

Metoda je nejdříve rekurzivně propuštěna až k jednoduchým datovým typům, na listech stromu. Tyto jednoduché datové typy na základě nastavení testů vygenerují požadavky na dodatečná nastavení, pokud dodatečná nastavení nejsou povolena, pak jsou vytvořeny jednotlivé testovací domény. V případě nastavení testů pouze na jeden výstupní soubor, jednoduchý datový typ vytváří pouze jednu testovací doménu. Každý jednoduchý datový typ, jako návratovou hodnotu metody *get_test_domains*, vrací počet testovacích domén, dostupných pro daný datový typ. V případě požadavku na dodatečná nastavení je navrácena konstanta 1.

Testovací domény komplexních datových typů jsou závislé na testovacích doménách jejich podelementů. Aby bylo možné pokrýt všechny testovací domény všech podelementů,

tak počet testovacích domén komplexního datového typu je roven maximálnímu počtu testovacích domén jednoho jeho podelementu.

Zvláštním případem jsou indikátory, které mohou tvořit mezivrstvu mezi komplexním datových typem a jeho podelementy. V jejich případě může být výpočet počtu testovacích domén proveden jinak, v závislosti na povaze indikátoru. Pro indikátory *All* a *Sequence* je možné použít stejný postup jako u komplexních datových typů, u indikátoru *Choice*, kde je do výsledku zahrnuta vždy jedna testovací doména jednoho podlementu, je počet testovacích domén roven součtu testovacích domén všech podelementů. Stejná situace jako pro indikátor *Choice* je i pro objekt *Schema*, kde je také vždy použita jedna testovací doména jednoho podelementu.

Aplikace dodatečných nastavení

Pokud jsou povolena dodatečná nastavení a vznikl alespoň jeden požadavek na dodatečná nastavení jsou nastavení vyžádána od uživatele a následně předána celému stromu XSD objektů prostřednictvím metody *set_additional_settings* objektu *Schema*, metoda je propagována ke všem objektům XSD stromu. Jednoduché datové typy akceptují dodatečná nastavení příslušející jim a podle nich vytvoří testovací domény.

Návratovou hodnotou této metody je počet existujících testovacích domén daného objektu. Určení počtu testovacích domén pro jednotlivé objekty probíhá stejně jako je popsáno v kapitole 3.3.5.

Generování dat

Ve fázi generování řídící objekt opakovaně volá metodu *get_test_value* nad kořenovým objektem *Schema*, jako argument jsou metody postupně předávány všechna nezáporná celá čísla menší než počet testovacích domén objektu *Schema*. Metoda je postupně propagována až k objektům jednoduchých datových typů na listech stromu. Při navracení ze zanořených metod je postupně vytvářen strom XML elementů obsahující testovací hodnoty.

Návratovou hodnotu metody *get_test_value* je objekt třídy *Output_element*, který reprezentuje XML element, obsahující vygenerovaná testovací data.

Jednoduché datové typy na základě čísla testovací domény generují testovací data, spadající do dané domény. U jednoduchých datových typů může být návratovou hodnotou metody *get_test_value* i objekt třídy *Output_string*, který bude začleněn do objektu třídy *Output_element* v nadřazeném objektu.

V rámci komplexních datových typů jsou vytvořeny objekty třídy *Output_element*, do kterých jsou postupně začleňovány návratové hodnoty metody *get_test_value* všech podelementů. V případě, že komplexní datový typ je nastaven jako *mixed*, pak jsou mezi návratové hodnoty jednotlivých podelementů vkládány objekty třídy *Output_string* obsahující náhodné řetězce.

Výsledkem této fáze je sada XML stromů obsahujících testovací data, počet vygenerovaných stromů odpovídá počtu testovacích domén kořenového objetu *Schema*.

Obálka nad generovanými daty

Pro usnadnění implementace komplexních datových typů a všech ostatních XSD objektů, které mohou obsahovat podelementy, je vytvořena třída *Test_wrapper*. Objekty této třídy vytváří mezivrstvu mezi rodičovským objektem a podelementem, cílem je usnadnit implementaci rodičovského objektu. Samotný objekt třídy *Test_wrapper* má dvě základní funkce.

První funkcí je zachytávání a uchovávání počtu platných testovacích domén pro daný podelement. V případě, že by rodičovský objekt požadoval vygenerování testovací domény, která by měla větší číslo, než je uchovávaný počet možných testovacích domén, pak by objekt třídy *Test_wrapper* požadavek změnil na náhodnou testovací doménu z platných testovacích domén podelementu.

Druhou funkcí je náhodné mapování požadovaných testovacích domén na skutečně vygenerované testovací domény. Po zjištění počtu platných testovacích domén podelementu, objekt třídy *Test_wrapper* vytvoří stálé mapování mezi vstupními požadavky a výstupními požadavky pro platné testovací domény, toto mapování je vytvořeno náhodně. Cílem je vytvořit různé kombinace testovacích domén do výstupních hodnot, pro různé běhy programu.

3.3.6 Tvorba výstupu

Po vygenerování testovacích dat jsou data serializována do textových XML souborů, serializace probíhá postupně pro jednotlivé vygenerované XML stromy testovacích dat. Každý vygenerovaný XML strom je uložen do samostatného souboru, tyto soubory jsou následně předány uživateli. V případě, že je vygenerováno více XML stromů, pak jsou výstupní XML soubory zabaleny do ZIP archivu.

Ve výstupním souboru je uvedena hlavička XML dokumentu a následně kořenový element a v něm zanořené veškeré další elementy. Výstup je tvořen strukturovaně s odsazením zanořených elementů. V případě elementů, které obsahují i vlastní text i podelementy, není použito odsazení.

3.4 Generování hodnot jednoduchých datových typů

V této sekci jsou popsány principy, podle kterých se vybírají testovací domény a jak se generují testovací data pro jednoduché datové typy existující v XSD. Do jednotlivých podsekcí je zahrnut daný datový typ a datové typy podobné nebo odvozené, u kterých se výběr testovacích domén a následné generování testovacích hodnot provádí stejně.

U většiny jednoduchých datových typů platí, že v rámci objektu datového typu jsou vytvořeny testovací domény, které jsou reprezentovány samostatnými objekty a samotné generování testovacích dat probíhá v objektech testovacích domén.

V případě, že jsou u daného datového typu specifikována omezení *Enumeration* nebo *Pattern*, pak se hodnoty generují podle nich. Hodnoty, vygenerované z těchto omezení, musí být validní hodnoty pro daný datový typ

3.4.1 Datový typ AnyURI

Datový typ *AnyURI* reprezentuje libovolnou URI adresu, v podstatě se jedná o řetězec odpovídající regulárnímu výrazu:

$$\hat{}([a\text{–}zA\text{–}Z0-9]+\backslash.)+[a\text{–}zA\text{–}Z]+(\backslash/([a\text{–}zA\text{–}Z0-9\backslash.])*(\backslash?.+=.+(\backslash\&.+=.+)*)?)?\$$$

Tento regulární výraz je použit pro validaci hodnot tohoto datového typu. Aby vyhovovali danému regulárnímu výrazu, musí mít vygenerované řetězce minimální délku 3 znaky, z toho důvodu není možné nastavit tomuto datovému typu omezení na maximální délku menší než 3.

V rámci programu jsou generovaná data omezená pouze na URL. Konkrétně pro délku dat větší než 7 jsou generována data odpovídající regulární výrazu:

$\hat{}\ldots\backslash..+\backslash\ldots\$$

Pro délku menší nebo rovno 7 podle regulární výrazu:

$\hat{}.\backslash..+\$$

3.4.2 Datový typ base64Binary

Datový typ *base64Binary* může obsahovat libovolný řetězec zakódovaný pomocí kódování base64. Pro generování testovacích hodnot je potřeba pouze vygenerovat libovolný náhodný řetězec a ten zakódovat pomocí kódování base64.

Problémem u tohoto datového typu je omezení na určitou délku vygenerovaných dat. Zakódování pomocí base64 prodlužuje původní řetězec o jednu třetinu délky, takže výsledný řetězec je dlouhý 4/3 délky původního řetězce. Z toho důvodu není možné spolehlivě vygenerovat řetězce o délce, která není dělitelná číslem 4.

3.4.3 Datový typ boolean

Datový typ *boolean* může obsahovat pouze 4 hodnoty: 0, 1, true, false. Tyto 4 hodnoty současně slouží jako samostatné testovací domény.

V rámci dodatečných nastavení je možné specifikovat, jestli mají být používány textové a číselné reprezentace daných hodnot.

3.4.4 Datový typ dateTime

Datový typ *dateTime* obsahuje řetězec specifikující datum,čas a volitelně časovou zónu. Formát zápisu je: *YYYY-MM-DDThh:mm:ss*, časová zóna může být specifikována konkatenováním řetězce určujícího časovou zónu, za řetězec určující datum a čas. Časová zóna může být specifikována písmenem "Z", které indikuje UTC, nebo pomocí řetězce *+/- hh:mm*, které indikuje časový posuv oproti UTC. Před datum může být umístěno znaménko minus které indikuje záporný časový údaj

Řetězec datového typu *dateTime* tedy musí odpovídat regulárnímu výrazu:

$$\hat{}([\backslash-\backslash+])?[0-9][0-9][0-9][0-9]\backslash-[0-9][0-9]\backslash-[0-9][0-9]$$
$$\mathrm{T}[0-9][0-9]\backslash:[0-9][0-9]\backslash:[0-9][0-9]$$
$$(?:\backslash.[0-9]+)?([\mathrm{Z}]|(?:[\backslash-\backslash+]([0-9][0-9]\backslash:[0-9][0-9])))?\$$$

Do testovacích dimenzí jsou zahrnuty hodnoty odpovídající minimálním a maximálním hodnotám (pokud jsou nastaveny). Následně je vypočten referenční čas, v případě že jsou nastaveny minimální a maximální hodnoty, pak je roven náhodné hodnotě mezi těmito čísly, v případě absence jednoho z nich je roven určené hodnotě posunuté o 5 let směrem do platného definičního oboru, v případě absence obou hodnot je roven aktuálnímu datu a času. Následně jsou vygenerovány testovací dimenze způsobem, kdy je k referenčnímu času přičítáno nebo odečítáno náhodné číslo v řádu let, měsíců, dní a času.

Stejný princip platí i pro podobné datové typy: *date, time, gYear, gYearMonth, gMonthDay, gDay a gMonth*, které používají pouze některé části obsažené v datovém typu *dateTime*.

V rámci dodatečných nastavení je možné povolit nebo zakázat použití časové zóny.

3.4.5 Datový typ decimal

Datový typ *decimal* reprezentuje libovolné reálné číslo, které je složeno z celé a volitelně z desetinné části, volitelně může být přidáno i předřadné znaménko plus nebo minus.

Na základě zadaných omezení je vytvořen interval, ze kterého se mohou generovat testovací hodnoty, v případě absence omezení zadaných ze strany uživatele, jsou použita omezení na maximální a minimální hodnotu definovaná v [9].

Problém jsou omezení *totalDigits* a *fractionDigits*, které specifikují celkový počet cifer resp. maximální možný počet desetinných cifer. V případě výskytu těchto omezení je nutné aby všechna čísla v intervalu generovaných hodnot měla takový počet cifer. V tomto případě ve výsledku pravděpodobně vzniknou 2 intervaly, zvlášť pro kladná a záporná čísla. Intervaly musí být vytvořeny tak, aby číslo s nejvyšší absolutní hodnotu mělo počet celých cifer roven hodnotě omezení *totalDigits* a číslo s nejnižší absolutní hodnotu počet celých cifer roven rozdílu hodnot omezení *totalDigits* a *fractionDigits*. Tyto intervaly mohou být navíc omezeny intervalem daným omezeními na minimální a maximální hodnotu čísla.

Testovací hodnoty budou generovány pouze z vypočtených intervalů, pokud by bylo vygenerováno číslo s počtem cifer menším než je uvedeno v omezení *totalDigits* bude doplněno desetinnými ciframi.

Do testovacích domén budou zahrnuta čísla (pokud to omezení umožní): 0, 1, minimální a maximální hodnota obou intervalů, náhodná hodnota z obou intervalů.

Stejný princip je použit i pro datový typ *Integer* a všechny datové typy z něj odvozené.

3.4.6 Datový typ duration

Datový typ *duration* reprezentuje časový interval o určité délce. Formát je následující: *PnYnMnDTnHnMnS*, kde n je přirozené číslo. Specifikován je rok, měsíc, den, hodina, minuta a sekunda, povinně se ve výsledné hodnotě musí vyskytovat alespoň jeden z těchto údajů. Výslednému řetězci může být předřazeno znaménko minus, indikující záporný časový interval. Hodnota tedy musí odpovídat regulárnímu výrazu:

```
^([\-\+])?P(?:([0-9]+)Y)?(?:([0-9]+)M)?(?:([0-9]+)D)?
(?:T(?:([0-9]+)H)?(?:([0-9]+)M)?(?:[0-9]+(?:\.[0-9]+)?S)?)?$
```

Pokud nejsou specifikovány maximální nebo minimální hodnoty, je jako hranice použita hodnota +/- 2^{30} sekund. V programu jsou všechny položky zadaných hodnot převedené na sekundy a dále je se s nimi pracuje v tomto formátu.

Jako testovací dimenze jsou zvoleny hodnoty na minimální a maximální hranici oboru hodnot a náhodné hodnoty v oboru hodnot. V testovacím výstupním řetězci jsou využity všechny položky časových údajů.

3.4.7 Datový typ float

Datový typ *float* reprezentuje libovolné reálné číslo. U těchto čísel je uvažováno, že jsou ukládány v aritmetice s plovoucí řádovou čárkou, tím pádem je možné aby tento datový typ nabývat i hodnot *INF, -INF* a *NaN*.

Pokud nejsou maximální nebo minimální hodnoty specifikované v omezeních, jsou použity hodnoty z [9]. Následně jsou z daného intervalu generovány testovací hodnoty. Do testovacích domén jsou zahrnuty hodnoty (pokud to omezení a nastavení povolují):

- 0,

- -0,
- 0.1,
- 1,
- Minimální a maximální hodnota,
- *INF* a *-INF*,
- *NaN*,
- Minimální hodnota + 0.1,
- Maximální hodnota - 0.1,
- Náhodné číslo z intervalu mezi minimální a maximální hodnotou.

V rámci dodatečných nastavení je možné nastavit jestli se mají do výstupu používat konstanty *INF,-INF* a *NaN*.

Stejný princip je použit pro datový typ *double*, jehož jediným rozdílem oproti typu *float* je větší obor hodnot.

3.4.8 Datový typ hexBinary

Datový typ *hexBinary* reprezentuje libovolný řetězec zakódovaný pomocí kódování hexBinary, toto kódování reprezentuje každý znak pomocí dvou hexadecimálních číslic, které reprezentují ASCII hodnotu daného znaku.

Pro omezení na délku výsledného řetězce je nutné, aby délka bylo sudé číslo, protože po zakódování je každý znak zdrojového řetězce reprezentován dvěma znaky. Pokud není specifikovaná délka výsledného řetězce, pak jako minimální délka je použito číslo 0 a jako maximální délka číslo 100.

Do testovacích domén jsou uloženy hodnoty:

- řetězec obsahující pouze čísla 0 o minimální délce,
- řetězec obsahující pouze čísla 0 o maximální délce,
- řetězec obsahující pouze čísla F o minimální délce,
- řetězec obsahující pouze čísla F o maximální délce,
- řetězec obsahující náhodná čísla o minimální délce,
- řetězec obsahující náhodná čísla o maximální délce,
- řetězec obsahující náhodná čísla o náhodné délce.

3.4.9 Datový typ NOTATION

Pro datový typ *NOTATION* je vyžadováno, aby hodnoty které může obsahovat byly definovány v omezení *Enumeration*, žádné další hodnoty není možné generovat.

3.4.10 Datový typ QName

Datový typ *QName* reprezentuje jméno, které je využíváno v určitém prostoru jmen. Pro generování testovacích dat je použit prostor XML.

Generovaná testovací data začínají prefixem *xml:* a následuje náhodně vygenerovaný řetězec. Pro omezení nad tímto datovým typem je nutné aby maximální délka vygenerované hodnoty byla minimálně 5.

3.4.11 Datový typ string

Datový typ *string* reprezentuje libovolný řetězec, který může obsahovat jakékoli znaky.

Jako testovací domény jsou použity délky výsledných řetězců a to minimální a maximální délka a několik řetězců náhodné délky v rozmezí mezi minimální a maximální délkou. Vždy je vytvořeno alespoň 10 testovacích domén. V dalších testovacích doménách je délka řetězce náhodná a je zakázáno do řetězce vkládat mezery.

V rámci dodatečných nastavení je možné upřesnit používanou znakovou sadu a to konkrétně jestli se má používat:

- malá anglická abeceda,
- velká anglická abeceda,
- čísla,
- XML entity,
- speciální znaky (znaky: (,), # ,@,˜apod.),
- bílé znaky (konce řádků, tabulátory).

Ve výchozím nastavení se používá malá a velká anglická abeceda, čísla a bílé znaky.

Odvozený typ *normalizedString* je velice podobný, jediným rozdílem je, že neumožňuje vkládat bílé znaky.

3.4.12 Datový typ token

Datový typ *token* je odvozen od datového typu *normalizedString* a dědí většinu jeho vlastností. Rozdílem oproti typu *normalizedString* je, že hodnota datového typu *token* nesmí začínat ani končit libovolným nenulovým počtem mezer a kdekoli v hodnotě se nesmí vyskytovat 2 mezery po sobě.

Veškeré principy generování testovacích dat jsou převzaty ze třídy *String*. Stejné principy platí i pro všechny datové typy odvozené od typu *token*.

3.4.13 Indikátory jednoduchých datových typů

Indikátor je objekt, který obsahuje podelementy a definuje určitý způsob, jakým se mohou tyto podelementy objevit ve výstupním dokumentu. V XSD se vyskytují 2 objekty, které je možné označit za indikátory jednoduchých datových typů a to *Union* a *List*. Tyto indikátory mohou fungovat jako obsah uživatelsky definovaných jednoduchých datových typů.

Indikátor *Union* může obsahovat neomezené množství dalších jednoduchých datových typů a ve výstupu je využit právě jeden z nich. Ve vygenerovaných testovacích hodnotách jsou postupně použity všechny testovací domény všech vnořených datových typů.

Indikátor *List* obsahuje právě jeden jednoduchý datový typ a ve výstupu je možné do elementu vložit libovolný počet hodnot tohoto datového typu. Ve vygenerovaných testovacích hodnotách je vložen náhodný počet hodnot.

3.5 Použití mutačního testování

V programu existuje možnost použití mutačního testování. Při použití mutačního testování jsou u datových typů *Decimal, Float, HexBinary* a typů z nich odvozených, do testovacích domén přidány i hodnoty, které přesahují maximální/minimální povolené hodnoty, žádná další podpora mutačního testování není vytvořena.

Důvodem je předpoklad, že cílový testovaný systém validuje vstupní XML dokumenty vůči XML schématu. Nevalidní XML dokument pak pouze testuje XML validátor a nikoli testovaný systém.

Kapitola 4

Implementace generátoru testovacích dat

Program je implementován jako webová aplikace, kde na straně je serveru je použit jazyk PHP, na straně klienta HTML. Využití webových technologií výrazně zjednodušuje implementaci programu na straně uživatelského rozhraní, které zajistí webový prohlížeč.

Aplikace je navržena tak, že klientská část aplikace, ve webovém prohlížeči, slouží pouze jako uživatelské rozhraní, hlavní část aplikace běží na straně serveru.

4.1 Implementace serverové části

Na straně serveru je použit skriptovací jazyk PHP ve verzi 5.4, není použit žádný framework. PHP je imperativní, objektově orientovaný jazyk, proto je možné návrh (viz. kapitola 3) snadno implementovat.

PHP je interpretovaný jazyk a bez použití překladové cache (která v projektu použita není), je nutné program při každém běhu analyzovat a následně interpretovat. V kombinaci s tím, že v PHP každý požadavek od klienta vyžaduje další běh programu, takže opět analýzu a interpretaci kódu, tento přístup podstatně snižuje rychlost aplikací vytvořených v PHP.

4.1.1 Adresářová struktura

Program je vytvořen jako modulární a jeho soubory jsou rozloženy do několika adresářů. Struktura adresářů a jejich význam je přibližně takováto:

app Adresář obsahující zdrojové soubory serverové části aplikace.

- **app/ReverseRegex** Adresář obsahuje zdrojové kódy knihovny ReverseRegex.
- **app/Testable** V adresáři jsou zdrojové kódy objektů, které generují výstupní data.
  - **app/Testable/Simple_type** Adresář obsahuje třídy jednoduchých datových typů.

output Adresář pro dočasné soubory obsahující výstupní data.

www Adresář pro všechny soubory dostupné webovému prohlížeči, jsou zde umístěny obrázky, CSS styly, javascriptové soubory a soubory pro spuštění aplikace.

Konfigurace programu

V rámci aplikace je možné omezeně překonfigurovat nastavení. Nastavení je uloženo v souboru *app/init.php*. Je možné přenastavit adresář pro dočasné výstupní soubory, výstupní kódování, počáteční hodnotu generátoru náhodných čísel, hlavičku výstupního XML dokumentu a některé výchozí hodnoty, použité při generování.

4.1.2 Perzistence objektů

Pro PHP je typické, že jeden běh aplikace odpovídá jednomu požadavku ze strany webového prohlížeče, tento přístup značně komplikuje práci s uživatelskými proměnnými v průběhu několika požadavků od klienta.

Ve vytvořené aplikaci je situace, kdy je uživatel žádán o dodatečná nastavení, v této situaci je nutné rozdělit generování výstupu do dvou běhů programu. V prvním je analyzován vstupní XSD dokument a vytvořeny požadavky na dodatečná nastavení. V druhém běhu jsou dodatečná nastavení předána objektům, reprezentující XSD objekty a jsou vygenerována výstupní data. Problémem je přetrvání stromu XSD objektů z prvního běhu programu do druhého, aby nebylo nutné ve druhém běhu programu znovu analyzovat vstupní XSD dokument.

Pro perzistenci objektů je využit mechanismus PHP relací (session), který naváže relaci s konkrétním prohlížečem a umožňuje ukládat textová data do speciálních proměnných a v dalším běhu programu, při požadavku od stejného prohlížeče, je opět načíst. Do těchto proměnných je uložen strom XSD objektů, pomocí funkce *serialize*, která objekty převede do textové reprezentace, v dalším běhu programu jsou objekty opět vytvořeny pomocí funkce *unserialize*.

4.1.3 Generování náhodných hodnot

Generování náhodných hodnot je velice důležitou částí programu, zásadní pro generování testovacích hodnot. V programu je pro generování náhodných hodnot využit generátor *Mersenne Twister* dostupný standardně v PHP. Na počátku běhu programu je generátor inicializován na hodnotu aktuálního času serveru, s využitím funkce *microtime*, která čas určuje s přesností na mikrosekundy.

Tento přístup zajišťuje generování různých hodnot pro každý běh programu.

4.1.4 Vytváření souborů

V rámci projektu je potřeba vytvářet soubory s výstupními daty, proto je potřeba aby na straně serveru existoval adresář, do kterého může aplikace zapisovat a je možné zde tyto soubory vytvářet.

Problém je pojmenovávání vytvořených souborů, protože souborů může být mnoho a nesmí se vzájemně ovlivňovat, tento problém je vyřešen pomocí standardní funkce *tempnam*, která v zadaném adresáři vytvoří soubor, s náhodným jménem a zajistí jeho unikátnost. Při generování výstupu je každý jeden XML dokument umístěn do samostatného souboru. V případě generování více výstupních XML dokumentů pro jednoho klienta jsou výstupní soubory zabaleny do ZIP archivu.

Soubory s výstupními daty jsou uloženy v adresáři, který není přímo přístupný z webové stránky, ale musí být na výstup vystaveny aplikací. Při vygenerování souboru je název

souboru uložen v PHP relaci a klientovi je umožněno stáhnout pouze tento soubor, žádný jiný. Tento přístup znemožňuje klientům získat přístup k souborům jiných uživatelů.

Soubory jsou na straně serveru ponechány do okamžiku, kdy jsou vystaveny uživateli ke stažení, poté jsou smazány. Problém může nastat v okamžiku, kdy si klient nechá vygenerovat výstupní soubor a prohlížeč nepožádá o stažení souboru. Tento soubor zůstane na serveru uložen a nebude aplikací smazán.

4.1.5 Informace o chybách

Aplikace využívá výjimek pro propagování informace o chybovém stavu při běhu aplikace. Používané výjimky jsou definované v souboru *app/Exceptions.php*. V případě vyhození výjimky v průběhu programu je výjimka zachycena v řídicím objektu třídy *Test_generator*, text výjimky je následně zobrazen uživateli a běh programu je ukončen.

4.2 Uživatelské rozhraní

Aplikace je vytvořena jako webová, takže interakce s aplikací probíhá skrz webový prohlížeč. Technologie použité pro uživatelské rozhraní jsou HTML5, CSS3 a javascript s použitím knihovny jQuery.

Webová stránka je koncipována tak, aby byla jednoduchá a přehledná, bez pokročilé grafiky. Po otevření stránky je veškerá další komunikace se serverem prováděna pomocí technologie AJAX tak, aby nebylo nutné další obnovování stránky v průběhu užívání aplikace.

Stránka umožňuje vložit XSD dokument jako soubor nebo jako text. V případě souboru je na straně klienta soubor otevřen a přečten, v obou případech je na server XSD dokument poslán jako textový řetězec.

4.2.1 Komunikace se serverem

Při používání aplikace je nutné předávat data ze strany klienta k serveru a opačně, jedná se o XSD dokumenty, nastavení testů, dodatečná nastavení a výstupní soubory. Veškerá tato komunikace probíhá asynchronně bez nutnosti obnovovat stránku v prohlížeči.

Zprávy přenášené mezi klientem serverem jsou posílány oběma směry a v obou směrech jsou data přenášena ve formátu JSON.

Od klienta k serveru jsou přenášeny 2 typy zpráv. První typ zprávy je použit při předávání zadání generátoru, tato zpráva obsahuje 3 položky: XSD dokument, nastavení testů a povolení mutačního testování. Druhý typ zprávy je použit při přenosu dodatečných nastavení na server.

Každá zpráva přenášená ze serveru ke klientovy má 2 položky: *type* a *data*, kde položka *type* specifikuje typ přenášené zprávy a položka *data* vlastní obsah zprávy. Ve směru ze serveru ke klientovy jsou přenášeny 3 typy zpráv:

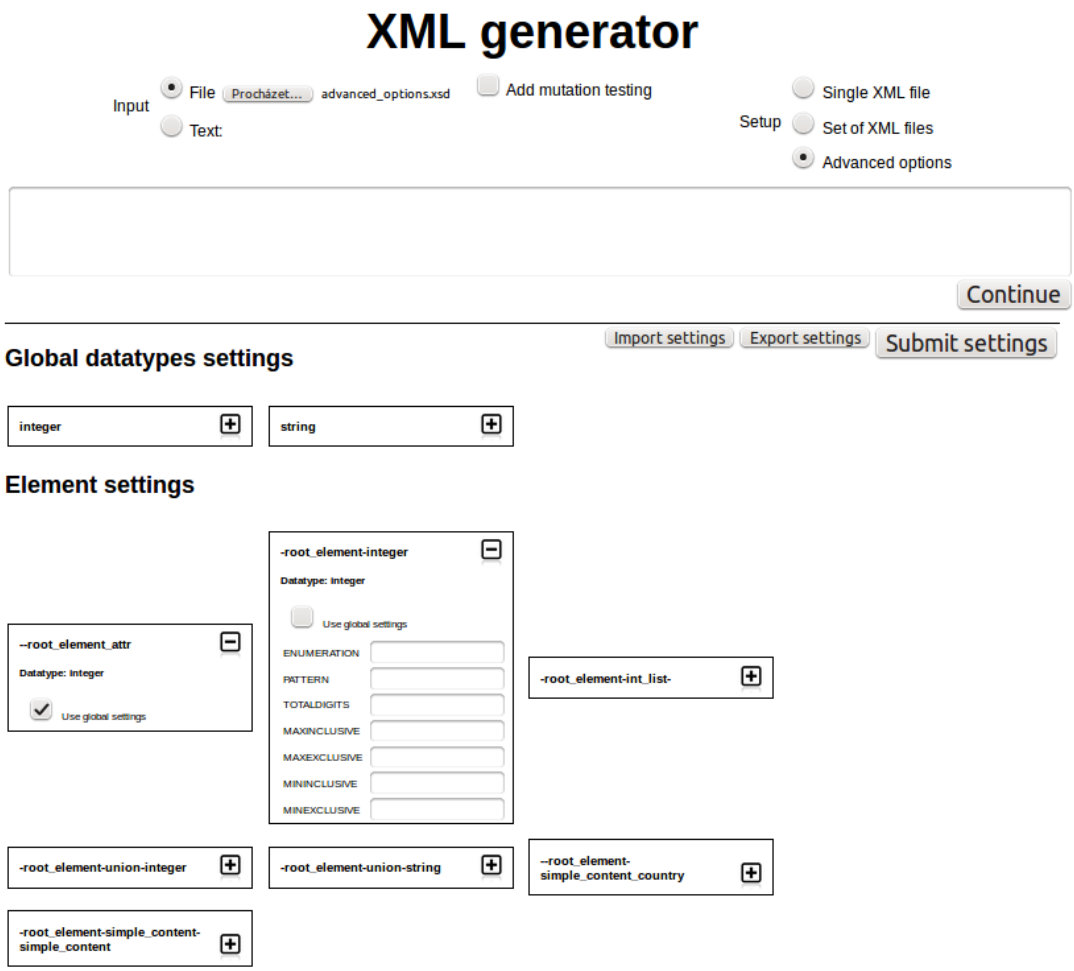
settings Výzva o dodatečná nastavení, položka data obsahuje dostupná nastavení.

message Chybová zpráva pro uživatele, položka data obsahuje popis chyby.

download Zpravuje klienta o tom, že je připraven výstupní soubor a že má klient zahájit stahování.

4.2.2 Popis uživatelského rozhraní

Vzhled uživatelského rozhraní zobrazuje následující obrázek, jedná se o snímek obrazovky nástroje:



Obrázek 4.1: Snímek obrazovky nástroje

Na tomto snímku je použit vstupní soubor C.3 s použitím dodatečných nastavení.

Horní část rozhraní slouží pro zadání vstupního souboru a nastavení testů. Spodní část rozhraní je použita pouze v případě dodatečných nastavení.

V horní části je vlevo umístěn přepínač pro volbu způsobu vložení vstupního XSD dokumentu. XSD dokument může být vložen jako soubor nebo jako text. Uprostřed horní části je zaškrtávací tlačítko pro povolení mutačního testování. V horní části vpravo je umístěn přepínač pro volbu nastavení testů. Posledním prvkem v horní části je tlačítko *Continue*, pomocí kterého se vstupy odesílají dále ke zpracování.

Ve spodní části jsou zejména 2 skupiny dodatečných nastavení: globální nastavení datových typů a nastavení jednotlivých elementů. V těchto nastaveních je možné specifikovat jaké hodnoty se mají v jednotlivých elementech používat. Implicitně si všechny elementy berou nastavení, které je specifikováno pro jejich datový typ v globálním nastavení datových typů.

Ve spodní části jsou dále umístěna 3 tlačítka: import a export dodatečných nastavení

a odeslání dodatečných nastavení. Tlačítka pro import a export jsou více popsána v sekci 4.2.3. Tlačítko *Submit settings* odešle dodatečná nastavení k dalšímu zpracování.

4.2.3 Dodatečná nastavení

Při přenosu dodatečných nastavení je pro každý jeden element a atribut specifikována výzva o jeho dodatečná nastavení, v každé výzvě je uveden název, datový typ a požadovaná nastavení. V položce datového typu je vždy uveden základní datový typ, v případě nově definovaného datového typu je zde veden základní datový typ, od kterého byl nový typ odvozen. Všechny výzvy jsou vždy přenášeny v jedné zprávě.

Specifikování datového typu u každé výzvy umožňuje v uživatelském rozhraní vytvořit globální dodatečná nastavení pro daný datový typ, které funguje jako globální implicitní dodatečné nastavení pro všechny výzvy s daným datovým typem. U každé jednotlivé výzvy je možné nastavit, jestli se má řídit tímto globálním nastavením nebo, jestli bude pro danou výzvu dodatečné nastavení specifikováno individuálně. Použití těchto globálních nastavení je pouze záležitostí uživatelského rozhraní, na server jsou dodatečná nastavení odeslána plnohodnotně pro každou jednu výzvu.

Export a import dodatečných nastavení

Do uživatelského rozhraní je přidaná možnost exportování a následného importování dodatečných nastavení. Export je proveden z hodnot, které jsou aktuálně nastaveny v uživatelském rozhraní. Výsledkem exportu je textový soubor ve formátu JSON, ve kterém jsou uloženy nastavení všech dodatečných nastavení. Při generování hodnot ze stejného XSD dokumentu může být soubor s dodatečným nastavením znovu nahrán do uživatelského rozhraní. Import způsobí nastavení všech elementů v uživatelském rozhraní, pro zadávání dodatečných nastavení stejně, jako byla v okamžiku exportu.

Soubor s dodatečným nastavením je možné editovat a měnit v něm uložená nastavení.

4.3 Použité knihovny

V rámci projektu jsou použity standardní i další veřejně přístupné knihovny. Použité knihovny jsou popsány v následujících částech textu.

Knihovna ReverseRegex

ReverseRegex je knihovna pro PHP pro generování řetězců podle regulárního výrazu. Jedná se o otevřenou knihovnu, dostupnou z [2]. Knihovna je schopna analyzovat regulární výrazy používané v XSD a vygenerovat textové řetězce odpovídající danému regulárnímu výrazu.

Tato knihovna je použita ve třídě *String_generator*, která, mimo jiné, tvoří abstrakci nad touto knihovnou. Generování testovacích řetězců z regulárních výrazů je v aplikaci závislé na schopnostech této knihovny.

Tuto knihovnu je možné použít v libovolné verzi.

Knihovna SimpleXML

SimpleXML je standardní knihovna PHP určená pro práci s XML dokumenty. V aplikaci je využita pro validaci vstupního XSD dokumentu a pro parsování XML tagů v tomto dokumentu. Další zpracování XSD dokumentu zajišťuje sama aplikace.

Tuto knihovnu je možné použít v libovolné verzi.

Knihovna libzip

libzip je standardní PHP knihovna pro práci se ZIP archivy. V aplikaci je využívána pro tvorbu výstupních souborů ve formátu ZIP.

Tuto knihovnu je možné použít v libovolné verzi.

Knihovna jQuery

jQuery je knihovna pro Javascript pro obecné usnadnění práce v tomto jazyku. V aplikaci je použita ve verzi 2.1.3. V uživatelském rozhraní je využita pro tvorbu skriptů řídících činnost rozhraní. Knihovna je dostupná z [15].

4.4 Testování funkčnosti generátoru

Pro ověření funkčnosti implementované aplikace je vytvořená sada automatických testů. Cílem testů je odhalit chyby, vyskytující se v programu. Pro testování aplikace je vybráno testovací kritérium *Line coverage* tj. kritérium pro pokrytí všech řádků zdrojového kódu.

Testována byla pouze serverová část aplikace. Pro testování aplikace je použit framework PHPUnit, jedná se o volně dostupný testovací framework pro jednotkové testování, založený na architektuře xUnit. PHPUnit je dostupný z [13].

4.4.1 Jednotkové testování

Základním použitým principem testování je jednotkové testování (unit testing), v rámci tohoto testování je vytvořena velká sada testů, kde cílem každého testu je ověřit správnou funkčnost jedné elementární části programu, pro jeden definovaný případ.

Pomocí tohoto principu byly testovány zejména třídy jednoduchých datových typů, kde je testována zejména správná reakce na všechny varianty nastavených omezení. Dále byl tento princip použit na testování detekce nevalidních vstupů a stavů programu.

4.4.2 Modulové testování

Pro testování aplikace jako celku je použito modulové testování, ve kterém je vytvořen řídicí objekt programu a jsou simulovány vstupy od uživatele. Aplikaci jsou předkládány XSD dokumenty s určitým uživatelským nastavením, po vygenerování jsou převzaty a validovány vygenerované XML soubory. Pro validaci správných výsledků je využita standardní PHP knihovna *SimpleXML*.

Cílem těchto testů je ověřit správnost řídicích konstrukcí programu a schopnost vytvářet XML dokumenty, které odpovídají zadaným XSD dokumentům.

V rámci těchto testů jsou testovány všechna nastavení testů, kde každému nastavení přísluší jiný XSD dokument. Simulovány jsou veškeré vstupy od uživatele, včetně dodatečných nastavení.

Použitá schémata

Pro modulové testování je použito několik schémat, která jsou uvedená v příloze C. Jednotlivá schémata jsou určena pro různé nastavení a testují různou část programu.

Význam jednotlivých schémat:

C.1 Testuje nastavení jednoho výstupního souboru a schopnost aplikace správně zpracovat různé XSD objekty.

C.2 Testuje nastavení pro sadu výstupních souborů a aplikaci různých omezení uvedených přímo v XSD dokumentu.

C.3 Testuje dodatečná nastavení, schéma vytváří strom XSD objektů, na který je možné aplikovat dodatečná nastavení. Pro ověření zadaných dodatečných nastavení se vygenerované XML soubory validují proti schématu C.3.1

4.4.3 Výsledky testování

Celkově testy pokrývají 3717 z 3806 logických řádků programu. Všechny chyby nalezené v těchto částech programu byly opraveny.

Některé části zdrojového kódu nejsou pomocí testů pokryty, důvodem je přímá závislost řízení toku programu na věcech, které je problematické ovlivnit v rámci testů, např.: generování náhodných čísel nebo volání operačního systému. Další části nejsou pokryty kvůli špatné sémantické dosažitelnosti.

4.5 Známá omezení

Implementovaná aplikace zvládá zpracovat většinu objektů, které se mohou vyskytovat ve validním XSD dokumentu. Existují konstrukce, které se mohou ve validním XSD dokumentu vyskytovat, se kterými si aplikace nedokáže správně poradit. Následuje výčet a popis známých omezení aplikace.

Rekurzivní zanořování elementů

V rámci XSD dokumentů je možné definovat elementy, které jsou v sobě vzájemně zanořeny. Příslušná část XSD může vypadat například takto:

```
<xsd:complexType name="A">
    <xsd:sequence>
        <xsd:element name="child" type="B"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="B">
    <xsd:sequence>
        <xsd:element name="subelement" type="A"/>
    </xsd:sequence>
</xsd:complexType>
```

Vytvořit XML dokument, ve kterém by se měl vyskytovat elementu jednoho z definovaných datových typů je nemožné. V takovémto případě by při generování došlo k nekonečné smyčce a pádu programu. Aplikace je schopna podobnou situaci detektovat, vypsat chybovou hlášku a ukončit provádění programu.

Generování řetězců podle regulárních výrazů

Aplikace je schopna generovat testovací řetězce podle zadaných regulárních výrazů. Schopnosti aplikace jsou v tomto případě omezeny schopností externí knihovny používané pro generování řetězců (viz. 4.3). Známé konstrukce regulárních výrazů, které knihovna nezvládá jsou: \w, \s. Problematické jsou i konstrukce pro opakování znaků + a *, které jsou v některých případech generovány příliš dlouho.

Práce s více soubory

Aplikace není schopna zpracovat XSD objekt *include*, který do současného XSD dokumentu importuje další XSD dokument, ve kterém mohou být definovány další prvky, dále použité v původním XSD dokumentu.

Omezení striktní varianty elementu any

V návaznosti na předchozí nedostatek, práce s více soubory, není aplikace schopna korektně generovat XSD objekty *any* a *anyAttribute* s nastaveným atributem *processContents=strict*, přičemž tohle nastavení tohoto atributu je implicitní. Toto nastavení definuje pro daný objekt striktní režim.

Ve striktním režimu je nutné, aby jakýkoli element/atribut, který bude do výsledného XML dokumentu vložen na místo any objektů byl definován na globální úrovni v XSD dokumentu. Kterékoli elementy definované na globální úrovni v původním XSD dokumentu mohou být použity jako kořenové elementy výsledného XML dokumentu. Bez možnosti používat elementy z importovaných schémat by používání těchto elementů vedlo k možné nekonečné smyčce při generování.

Kapitola 5

Závěr

Cílem projektu bylo vytvořit nástroj, který by dokázal zjednodušit testování aplikací, které využívají jako vstup XML dokumenty. Hlavním cílem nástroje je generovat XML dokumenty, odpovídající danému XML schématu, přičemž vygenerované dokumenty by měly pokrývat co nejvíce možných hodnot, které by mohly způsobit chybu při vložení do testovaného systému. Tyto XML dokumenty mohou využít testeři aplikací jako testovací vstupy.

Nejdříve bylo nutné nastudovat formát XML a všechny možnosti v jeho zápisu a následně formát a možnosti XML schémat. Podle možností XML schémat bylo potřeba určit na co se bude zaměřovat generování testovacích XML a jakým způsobem se budou generovat testovací hodnoty.

Dalším krokem bylo definovat požadavky na výsledný program. Na základě požadavků byla navržena architektura a funkčnost programu. V navrženém programu bylo nutné analyzovat XML schéma, vytvořit objekty, odpovídající schématu a následně z těchto objektů generovat a exportovat testovací hodnoty.

Nejdůležitější částí návrhu bylo navrhnout princip generování testovacích hodnot, pro jednoduché datové typy, používané v XML. Právě tyto hodnoty jsou použity jako obsah základních XML elementů a v důsledku těchto hodnot může nastat selhání testovaného systému.

Nástroj byl implementován jako webová aplikace, kde většina činnosti probíhá na straně serveru. Pro implementaci byly použity jazyky PHP, HTML, CSS a javascript. Vzniklá aplikace nezvládá všechny možnosti, které se mohou v XML schématech objevit, známá omezení aplikace jsou popsána v kapitole 4.5.

Konečným krokem vývoje bylo testování výsledné aplikace. Testována byla pouze serverová část aplikace. Byl použit princip jednotkového testování, pro který byl použit testovací framework PHPUnit. V průběhu testování bylo nalezeno a opraveno přibližně 10 chyb.

Aplikaci je možné rozšířit o podporu práce s více XML schématy, která v současné verzi není. Další možností je získání lepší knihovny pro generování testovacích řetězců podle regulárních výrazů.

Další možností pro rozšíření je interakce aplikace s dalšími systémy. V současné verzi je jedinou možností pro přístup k aplikaci využití webového rozhraní. Aplikace by mohla být rozšířena o API umožňující přímou interakci s jinými systémy.

Literatura

[1] Amman, P.; Offutt, J.: *Introduction to software testing*. Cambridge University Press, 2008, iSBN 978-0-511-39330-3.

[2] icomefromthenet: ReverseRegex. `https://github.com/icomefromthenet/ReverseRegex/`, 2014-11-11 [cit. 2015-02-06].

[3] Myers, G. J.: *The Art of Software Testing*. John Wiley & Sons, 2004, iSBN 0-471-46912-2.

[4] Pecinovský, R.: *Návrhové vzory*. Computer Press, 2007, iSBN 978-80-251-1582-4.

[5] Rábová, Z.; Hanáček, P.; Peringer, P.; aj.: Užitečné rady pro psaní odborného textu [online]. `http://www.fit.vutbr.cz/info/statnice/psani_textu.html`, 2008-11-01 [cit. 2015-04-08].

[6] Smrčka, A.: ITS - Testování se znalostí kódu, kritéria pokrytí. Materiály k přednášce. Interní dokument. [online]. https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ITS-IT/lectures/2-pokryti.pdf, 2014 [cit. 2015-04-16].

[7] Smrčka, A.: ITS - Testování založené na syntaxi. Materiály k přednášce. Interní dokument. [online]. https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ITS-IT/lectures/6-syntax.pdf, 2014 [cit. 2015-04-16].

[8] Spillner, A.; Ling, T.; Schaefer, H.: *Software testing foundations*. Rocky Nook, 2007, iSBN 978-1-933952-08-6.

[9] WWW stránky: XML Schema Part 2: Datatypes[online]. `http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#built-in-datatypes`, 2001-05-02 [cit. 2014-10-12].

[10] WWW stránky: XML Schema Tutorial[online]. `http://www.w3schools.com/schema/`, 2001-05-02 [cit. 2014-10-12].

[11] WWW stránky: W3C XML Schema. `http://www.w3.org/TR/xmlschema11-1/`, 2014-04-13 [cit. 2014-10-18].

[12] WWW stránky: PHP: Hypertext Preprocessor. `http://php.net/`, 2015.

[13] WWW stránky: PHPUnit - The PHP Testing Framework. `https://phpunit.de/`, 2015.

[14] WWW stránky: W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. http://www.w3.org/TR/xmlschema11-1/, 2015-04-05.

[15] WWW stránky: jQuery. https://jquery.com/, 2015 [cit. 2015-04-14].

Příloha A

Obsah CD

Přílohou bakalářské práce je CD s následujícím obsahem:

app Zdrojové kódy projektu,

- **coverage_cover** výstup pokrytí zdrojových kódu testy,
- **src** zdrojové kódy aplikace,
- **tests** zdrojové kódy testů,
- **Makefile** makefile pro spuštění testů,
- **phpunit.phar** zdrojový kód knihovny PHPUnit,
- **phpunit.xml** konfigurační souboru PHPUnit,

text-src zdrojové kódy textové části práce,

README.txt popis instalace, užívání a testování aplikace,

xocena04-BP.pdf textová část práce.

Příloha B

Vzorový XSD dokument

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
    <xsd:element name="root_element" type="root_type"/>

    <xsd:complexType name="root_type">
        <xsd:sequence>
            <xsd:element name="myDecimal" type="myDecimal"/>
            <xsd:element name="myDateTime" type="xsd:dateTime"/>
            <xsd:element name="myTime" type="xsd:time"/>
            <xsd:element name="myDuration" type="xsd:duration"/>
            <xsd:element name="simplecontent" type="simple_content"/>
            <xsd:element name="local_customer" type="customer"/>
            <xsd:element name="simple_list" type="simple_list"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="myDecimal">
        <xsd:restriction base="xsd:decimal">
            <xsd:totalDigits value="10"/>
            <xsd:fractionDigits value="3"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="simple_list">
        <xsd:list itemType="xsd:integer"/>
    </xsd:simpleType>

    <xsd:complexType name="simple_content">
        <xsd:simpleContent>
            <xsd:extension base="xsd:integer">
                <xsd:attribute name="country" type="xsd:string" />
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
```

```
    <xsd:complexType name="customer">
        <xsd:sequence>
            <xsd:element name="firstname" type="xsd:string"/>
            <xsd:element name="lastname" type="xsd:string"/>
            <xsd:element name="country" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="attr" type="xsd:integer"/>
    </xsd:complexType>
</xsd:schema>
```

Příloha C

Testovací XSD dokumenty

V této části příloh jsou uvedeny XSD dokumenty, které jsou použity pro testování aplikace. Názvy kapitol reprezentují nastavení testů, pro které je daný dokument určen.

C.1 Jeden výstupní soubor

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
    <xsd:notation name="mynotation" public="image/gif">
        <xsd:annotation>
            <xsd:documentation>
                Test text
            </xsd:documentation>
        </xsd:annotation>
    </xsd:notation>

    <xsd:annotation>
        <xsd:appinfo>
            Any attr annotation
        </xsd:appinfo>
    </xsd:annotation>

    <xsd:element name="root" type="Configuration"/>
    <xsd:element name="alt_root" type="Configuration"/>

    <xsd:element name="root_string" type="root_string"/>
    <xsd:element name="inner_string" substitutionGroup="root_string"/>

    <xsd:complexType name="Configuration">
        <xsd:sequence maxOccurs="unbounded" minOccurs="1">
            <xsd:element name="string1" type="xsd:string"/>
            <xsd:element name="string2" type="xsd:string"/>
            <xsd:element name="integer" type="xsd:integer"/>
            <xsd:element name="datetime" type="xsd:dateTime"/>
            <xsd:element name="float" type="xsd:float"/>
```

```
<xsd:element name="double" type="xsd:double"/>
<xsd:element name="time" type="xsd:time"/>
<xsd:element name="hexbinary" type="xsd:hexBinary"/>
<xsd:element name="base64" type="xsd:base64Binary"/>
<xsd:element name="duration" type="xsd:duration"/>
<xsd:element name="anyuri" type="xsd:anyURI"/>
<xsd:element name="qname" type="xsd:QName"/>
<xsd:element name="token" type="xsd:token"/>
<xsd:element name="intlist" type="valuelist"/>
<xsd:element name="loggerlevel" type="LoggerLevels"/>
<xsd:sequence>
  <xsd:element name="integer" type="xsd:integer"/>
  <xsd:element name="next_int" type="xsd:int"/>
</xsd:sequence>
<xsd:element name="contextListener" type="ContextListener"/>
<xsd:element name="Timestamp" type="Timestamp"/>
<xsd:element name="myDateTime" type="myDateTime"/>
<xsd:element name="myDateTime2" type="myDateTime2"/>
<xsd:element name="myDateTime3" type="myDateTime3"/>
<xsd:element ref="root_string"/>
<xsd:element ref="inner_string"/>
<xsd:element name="ComplexContent">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="myType">
                <xsd:sequence>
                    <xsd:element name="int" type="xsd:int"/>
                    <xsd:element name="decimal"
                    type="xsd:decimal"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>

<xsd:element name="MixedContent">
    <xsd:complexType>
        <xsd:complexContent mixed="true">
            <xsd:extension base="omg">
                <xsd:attribute name="myattr"
                type="xsd:integer"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>

<xsd:element name="SimpleContent">
    <xsd:simpleType>
```

```
                <xsd:restriction base="xsd:integer">
                    <xsd:maxExclusive value="10000"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:choice>
            <xsd:any minOccurs="1" processContents="lax">
                <xsd:annotation>
                    <xsd:documentation>
                        In the any element
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:any>
        </xsd:choice>
        <xsd:any processContents="lax"/>
        <xsd:element name="TriggeringPolicy"
            type="TriggeringPolicy"/>
        <xsd:group ref="myGroup"/>
        <xsd:element name="foo" type="myAll"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="attrGroup"/>
</xsd:complexType>


<xsd:simpleType name="root_string">
    <xsd:restriction base="xsd:string">
        <xsd:maxLength value="10"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDateTime">
    <xsd:restriction base="xsd:dateTime">
        <xsd:maxInclusive value="2012-05-30T09:00:00"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDateTime2">
    <xsd:restriction base="xsd:dateTime">
        <xsd:minExclusive value="2002-05-30T09:00:00"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDateTime3">
    <xsd:restriction base="xsd:dateTime">
        <xsd:minExclusive value="2002-05-30T09:00:00"/>
        <xsd:maxExclusive value="2012-05-30T09:00:00"/>
    </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:complexType name="omg">
    <xsd:attribute name="omg" type="xsd:int"/>
</xsd:complexType>

<xsd:complexType name="myAll">
    <xsd:all>
        <xsd:element name="allel2" type="xsd:int" minOccurs="0"/>
        <xsd:element name="allel3" type="xsd:int" minOccurs="0"
            maxOccurs="1"/>
        <xsd:element name="allel4" type="xsd:int" minOccurs="1"/>
    </xsd:all>
</xsd:complexType>

<xsd:group name="myGroup">
    <xsd:choice>
        <xsd:element name="group_element" type="xsd:integer" />
        <xsd:element name="unionsimpletype" type="UnionSimpleType"/>
    </xsd:choice>
</xsd:group>

<xsd:attributeGroup name="attrGroup">
    <xsd:attributeGroup ref="attrGroup2"/>
    <xsd:attribute name="scanPeriod" type="xsd:normalizedString"
            use="optional"/>
    <xsd:attribute ref="myAttr"/>
    <xsd:anyAttribute processContents="lax">
        <xsd:annotation>
            <xsd:appinfo>
                Any attr annotation
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:anyAttribute>
</xsd:attributeGroup>

<xsd:attributeGroup name="attrGroup2">
    <xsd:attribute name="debug" type="xsd:boolean"
            use="optional" default="false"/>
    <xsd:attribute name="scan" type="xsd:string"
            use="optional"/>
    <xsd:attribute name="fixed" type="xsd:string"
            use="required" fixed="omgwtf"/>
</xsd:attributeGroup>

<xsd:attribute name="myAttr">
    <xsd:simpleType>
        <xsd:restriction base="xsd:decimal">
            <xsd:totalDigits value="10"/>
```

```
            <xsd:fractionDigits value="5"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>

<xsd:simpleType name="valuelist">
    <xsd:annotation>
        <xsd:documentation>
            Any other test text
        </xsd:documentation>
    </xsd:annotation>
    <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>

<xsd:complexType name="myType">
    <xsd:choice>
        <xsd:element name="foo" type="xsd:string"/>
    </xsd:choice>
    <xsd:anyAttribute processContents="lax"/>
</xsd:complexType>

<xsd:complexType name="ContextListener" mixed="true">
    <xsd:choice maxOccurs="unbounded">
        <xsd:element name="resetJUL" type="xsd:boolean"/>
    </xsd:choice>
    <xsd:attribute name="class" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="Timestamp">
        <xsd:attribute name="key" type="xsd:string"
            use="optional"/>
        <xsd:attribute name="datePattern" type="xsd:string"
            use="optional"/>
        <xsd:attribute name="timeReference" type="xsd:string"
            use="prohibited"/>
</xsd:complexType>

<xsd:complexType name="TriggeringPolicy">
        <xsd:sequence>
                <xsd:element name="maxFileSize" minOccurs="0"
                    maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="class" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="MySimpleContent">
    <xsd:simpleContent>
        <xsd:extension base="xsd:integer">
```

```
            <xsd:attribute name="country" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="UnionSimpleType">
    <xsd:union memberTypes="xsd:integer xsd:int"/>
</xsd:simpleType>

<xsd:simpleType name="ListsimpleType">
    <xsd:list>
        <xsd:simpleType>
            <xsd:restriction base="xsd:int"/>
        </xsd:simpleType>
    </xsd:list>
</xsd:simpleType>

<xsd:simpleType name="LoggerLevels">
    <xsd:union>
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="OFF"/>
                <xsd:enumeration value="ALL"/>
                <xsd:enumeration value="INHERITED"/>
                <xsd:enumeration value="NULL"/>
                <xsd:enumeration value="ERROR"/>
                <xsd:enumeration value="WARN"/>
                <xsd:enumeration value="INFO"/>
                <xsd:enumeration value="DEBUG"/>
                <xsd:enumeration value="TRACE"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:pattern value="[Oo][Ff]{2}"/>
                <xsd:pattern value="[Aa][Ll]{2}"/>
          <xsd:pattern value="[Ii][Nn][Hh][Ee][Rr][Ii][Tt][Ee][Dd]"/>
                <xsd:pattern value="[Nn][Uu][Ll]{2}"/>
                <xsd:pattern value="[Ee][Rr]{2}[Oo][Rr]"/>
                <xsd:pattern value="[Ww][Aa][Rr][Nn]"/>
                <xsd:pattern value="[Ii][Nn][Ff][Oo]"/>
                <xsd:pattern value="[Dd][Ee][Bb][Uu][Gg]"/>
                <xsd:pattern value="[Tt][Rr][Aa][Cc][Ee]"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType>
          <xsd:restriction base="xsd:integer">
            <xsd:minExclusive value="10"/>
```

```
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>

    <xsd:complexType name="noNameType"/>
</xsd:schema>
```

C.2 Sada výstupních souborů

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
    <xsd:element name="root_element" type="root_type"/>

    <xsd:complexType name="root_type">
        <xsd:sequence>
            <xsd:element name="myDecimal" type="myDecimal"/>
            <xsd:element name="myInteger" type="myInteger"
                maxOccurs="unbounded"/>
            <xsd:element name="myString" type="myString"/>
            <xsd:element name="patternString" type="patternString"/>
            <xsd:element name="enumString" type="enumString"/>
            <xsd:element name="myDateTime" type="myDateTime"/>
            <xsd:element name="myDateTime2" type="myDateTime2"/>
            <xsd:element name="myDateTime3" type="myDateTime3"/>
            <xsd:element name="myTime" type="myTime"/>
            <xsd:element name="myDuration" type="myDuration"/>
            <xsd:element name="myDuration2" type="myDuration2"/>
            <xsd:element name="myDuration3" type="myDuration3"/>
            <xsd:element name="myDuration4" type="myDuration4"/>
            <xsd:element name="myDuration5" type="myDuration5"/>
            <xsd:element name="myDuration6" type="myDuration6"/>
            <xsd:element name="myDuration7" type="myDuration7"/>
            <xsd:element name="myDuration8" type="myDuration8"/>
            <xsd:element name="simplecontent" type="simple_content"/>
            <xsd:element name="simplecontent2" type="simple_content2"/>
            <xsd:element name="simplecontent3" type="simple_content3"/>
            <xsd:element name="customer" type="Norwegian_customer"/>
            <xsd:element name="local_customer" type="local_customer"/>
            <xsd:element name="demanding_customer"
                type="demanding_customer"/>
            <xsd:element name="simple_list" type="simple_list"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="myDecimal">
        <xsd:restriction base="xsd:decimal">
            <xsd:totalDigits value="10"/>
```

```
        <xsd:fractionDigits value="3"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myInteger">
    <xsd:restriction base="xsd:integer">
         <xsd:minInclusive value="-150000"/>
         <xsd:maxInclusive value="150000"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myString">
    <xsd:restriction base="xsd:string">
        <xsd:minLength value="15"/>
        <xsd:maxLength value="74"/>
    </xsd:restriction>
</xsd:simpleType>

 <xsd:simpleType name="patternString">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[Oo][Ff]{2}"/>
        <xsd:pattern value="[Aa][Ll]{2}"/>
        <xsd:pattern value="[Ii][Nn][Hh][Ee][Rr][Ii][Tt][Ee][Dd]"/>
        <xsd:pattern value="[Nn][Uu][Ll]{2}"/>
        <xsd:pattern value="[Ee][Rr]{2}[Oo][Rr]"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="enumString">
    <xsd:restriction base="xsd:string">
                <xsd:enumeration value="INHERITED"/>
                <xsd:enumeration value="NULL"/>
                <xsd:enumeration value="ERROR"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDateTime">
    <xsd:restriction base="xsd:dateTime">
        <xsd:minInclusive value="2002-05-30T09:00:00"/>
        <xsd:maxInclusive value="2012-05-30T09:00:00"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDateTime2">
    <xsd:restriction base="xsd:dateTime">
        <xsd:maxExclusive value="2012-05-30T09:00:00"/>
    </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="myDateTime3">
    <xsd:restriction base="xsd:dateTime">
        <xsd:minExclusive value="2002-05-30T09:00:00"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myTime">
    <xsd:restriction base="xsd:time">
        <xsd:minInclusive value="00:10:00"/>
        <xsd:maxInclusive value="20:00:00"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDuration">
    <xsd:restriction base="xsd:duration">
        <xsd:minExclusive value="P12M"/>
        <xsd:maxInclusive value="P12Y18DT20H15M20S"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDuration2">
    <xsd:restriction base="xsd:duration">
        <xsd:maxExclusive value="P12Y18D"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDuration3">
    <xsd:restriction base="xsd:duration">
        <xsd:minInclusive value="P12MT15H18M12S"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDuration4">
    <xsd:restriction base="xsd:duration">
        <xsd:minExclusive value="-P12M"/>
        <xsd:maxExclusive value="P12Y18D"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDuration5">
    <xsd:restriction base="xsd:duration">
        <xsd:minInclusive value="-P5Y12M"/>
        <xsd:maxInclusive value="-P3Y18D"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDuration6">
```

```
    <xsd:restriction base="xsd:duration">
        <xsd:maxInclusive value="-P12Y18D"/>
    </xsd:restriction>
</xsd:simpleType>


<xsd:simpleType name="myDuration7">
    <xsd:restriction base="xsd:duration">
        <xsd:minInclusive value="-P5Y12M"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="myDuration8">
    <xsd:restriction base="xsd:duration">
        <xsd:minInclusive value="P12Y17D"/>
        <xsd:maxInclusive value="P12Y18D"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="simple_list">
    <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>

<xsd:complexType name="simple_content">
    <xsd:simpleContent>
        <xsd:extension base="xsd:integer">
            <xsd:attribute name="country" type="xsd:string" />
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="simple_content2">
    <xsd:simpleContent>
        <xsd:restriction base="simple_content">
            <xsd:maxInclusive value="150"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="simple_content3">
    <xsd:simpleContent>
        <xsd:extension base="simple_list">
            <xsd:attribute name="country" type="xsd:string" />
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="customer">
```

```
    <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
        <xsd:element name="country" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="attr" type="xsd:integer"/>
</xsd:complexType>

<xsd:complexType name="Norwegian_customer">
    <xsd:complexContent>
        <xsd:restriction base="customer">
            <xsd:sequence>
                <xsd:element name="firstname" type="xsd:string"/>
                <xsd:element name="lastname" type="xsd:string"/>
                <xsd:element name="country"
                    type="xsd:string" fixed="Norway"/>
            </xsd:sequence>
            <xsd:attribute name="attr" type="xsd:integer" fixed="40"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>

<xsd:group name="replacing_group">
    <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
</xsd:group>

<xsd:attributeGroup name="replacing_attr_group">
    <xsd:attribute name="attr" type="xsd:integer"
            fixed="40"/>
</xsd:attributeGroup>

<xsd:complexType name="local_customer">
    <xsd:complexContent>
        <xsd:restriction base="customer">
            <xsd:group ref="replacing_group"/>
            <xsd:attributeGroup ref="replacing_attr_group"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>

<xsd:group name="exten_group">
    <xsd:sequence>
        <xsd:element name="middle_name" type="xsd:string"/>
    </xsd:sequence>
</xsd:group>
```

```
    <xsd:attributeGroup name="exten_attr_group">
        <xsd:attribute name="attr2" type="xsd:integer"
                fixed="40"/>
    </xsd:attributeGroup>

    <xsd:complexType name="demanding_customer">
        <xsd:complexContent>
            <xsd:extension base="customer">
                <xsd:group ref="exten_group"/>
                <xsd:attributeGroup ref="exten_attr_group"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

C.3 Pokročilá nastavení

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                elementFormDefault="qualified">
    <xsd:element name="root_element" type="root_type"/>

    <xsd:complexType name="root_type">
        <xsd:sequence>
            <xsd:element name="integer" type="myInteger"/>
            <xsd:element name="int_list" type="int_list"/>
            <xsd:element name="union" type="myunion"/>
            <xsd:element name="simple_content" type="simple_content"/>
        </xsd:sequence>
        <xsd:attribute name="attr" type="attr_integer"/>
    </xsd:complexType>

    <xsd:simpleType name="myInteger">
        <xsd:restriction base="xsd:integer">
            <xsd:whiteSpace value="preserve"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="attr_integer">
        <xsd:restriction base="xsd:integer">
            <xsd:whiteSpace value="preserve"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="int_list">
        <xsd:list itemType="listInteger"/>
    </xsd:simpleType>
```

```
    <xsd:simpleType name="listInteger">
        <xsd:restriction base="xsd:integer">
            <xsd:whiteSpace value="preserve"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="myunion">
        <xsd:union>
            <xsd:simpleType>
                <xsd:restriction base="xsd:integer">
                    <xsd:whiteSpace value="preserve"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:whiteSpace value="preserve"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>

    <xsd:complexType name="simple_content">
        <xsd:simpleContent>
            <xsd:extension base="myInteger">
                <xsd:attribute name="country" type="xsd:string" />
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:schema>
```

C.3.1 Validace pokročilých nastavení

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                elementFormDefault="qualified">
    <xsd:element name="root_element" type="root_type"/>

    <xsd:complexType name="root_type">
        <xsd:sequence>
            <xsd:element name="integer" type="myInteger"/>
            <xsd:element name="int_list" type="int_list"/>
            <xsd:element name="union" type="myunion"/>
            <xsd:element name="simple_content" type="simple_content"/>
        </xsd:sequence>
        <xsd:attribute name="attr" type="attr_integer"/>
    </xsd:complexType>

    <xsd:simpleType name="myInteger">
        <xsd:restriction base="xsd:integer">
            <xsd:whiteSpace value="preserve"/>
```

```
        <xsd:minInclusive value="150"/>
        <xsd:maxExclusive value="500"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="attr_integer">
    <xsd:restriction base="xsd:integer">
        <xsd:whiteSpace value="preserve"/>
        <xsd:minExclusive value="200"/>
        <xsd:maxInclusive value="468"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="int_list">
    <xsd:list itemType="listInteger"/>
</xsd:simpleType>

<xsd:simpleType name="listInteger">
    <xsd:restriction base="xsd:integer">
        <xsd:whiteSpace value="preserve"/>
        <xsd:minInclusive value="50"/>
        <xsd:maxInclusive value="150"/>
    </xsd:restriction>
</xsd:simpleType>

 <xsd:simpleType name="myunion">
    <xsd:union>
        <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
                <xsd:whiteSpace value="preserve"/>
                <xsd:minInclusive value="-50"/>
                <xsd:maxExclusive value="50"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:whiteSpace value="preserve"/>
                <xsd:length value="50"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>

<xsd:complexType name="simple_content">
    <xsd:simpleContent>
        <xsd:extension base="myInteger">
            <xsd:attribute name="country" type="xsd:string" />
        </xsd:extension>
```

```
            </xsd:simpleContent>
        </xsd:complexType>
</xsd:schema>
```