FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Návrh a implementace modulu pro obousměrný převod mezi formáty x-definice a XML Schéma
Student:	Bc. Tomáš Šmíd
Vedoucí:	Mgr. Václav Trojan
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2020/21

Pokyny pro vypracování

X-definice je rozsáhlý framework určený k validaci a transformaci strukturovaných dat, jehož součástí může být dynamický přístup k externím zdrojům. Cílem této práce je návrh a implementace modulu pro vzájemnou transformaci mezi formáty x-definice a XML Schema.

Postupujte v těchto krocích:

1) seznamte se se stávající implementací frameworku x-definice, zejména se stávajícím modulem pro transformaci XMLSchema, který je nevyhovující,
2) seznamte se se stávajícím návrhem specifikace XML Schema,
3) navrhněte modul pro obousměrnou transformaci mezi formáty x-definice a XML Schema,
4) návrh implementujte, řádně zdokumentujte a otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 28. srpna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Návrh a implementace modulu pro obousměrný převod mezi formáty x-definice a XML Schéma

Bc. Tomáš Šmíd

Katedra softwarového inženýrství
Vedoucí práce: Mgr. Václav Trojan

6. ledna 2020

Poděkování

V první řadě bych chtěl poděkovat lidem, kteří mi pomohli nebo mi dali možnost se rozvíjet v oblasti informatiky. Taktéž bych chtěl poděkovat své rodině a blízkým přátelům za podporu jak v průběhu celého studia, tak i v soukromém životě.

Velké poděkování též patří vedoucímu mé diplomové práce, panu Mgr. Václavu Trojanovi, který mne uvedl nejen do problematiky X-definic, ale taktéž mi v obecném měřítku rozšířil obzory v oboru informatiky.

Na závěr taktéž děkuji Karolíně Böhmové za pomoc s hledáním chyb v textu této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 6. ledna 2020 .....

Odkaz na tuto práci

Šmíd, Tomáš. *Návrh a implementace modulu pro obousměrný převod mezi formáty x-definice a XML Schéma*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato diplomová práce se zabývá analýzou a návrhem algoritmu pro obousměrný převod formátů X-definice a XML schéma. Práce dále popisuje problémové části transformace zmíněných datových formátů. Výsledný algoritmus je implementován jako nadstavba volně dostupné knihovny X-definice od společnosti Syntea software group a.s. v programovacím jazyce Java.

Klíčová slova X-definice, XML schéma, XSD, XML, validace XML W3C XML Schéma 1.0, transformace datových formátů, Java

Abstract

This master's thesis describes analysis and design of bidirectional algorithm for transformation of X-definition and XML schema formats. Furthermore thesis discusses problematic parts of transformation of mentioned data formats. Final algorithm is implemented in Java programming language and using open source library named X-definition of company Syntea software group a.s as dependency.

Keywords X-definition, XML schema, XSD, XML, XML validation, W3C XML Schema 1.0, transformation of data formats, Java

Obsah

Seznam obrázků

Seznam ukázek

Úvod

Česká společnost se soudobým názvem Syntea software group a.s používá již skoro 20 let pro interní účely zpracování dat tzv. X-definice[1]. Tyto X-definice v původní implementaci zpracovávaly pouze datový formát XML, v dnešní době jsou však rozšiřovány i o datový formát JSON. X-definice slouží jak k validaci XML dat, tak i k jejich načtení do paměti počítače. K uložení dat v rámci paměti lze použít i tzv. X-komponenty[1], které jsou obdobou JAXB [2]. X-definice dále poskytují například konstrukční mód, který slouží k vytvoření předpisu X-definice dle zadaných XML dat. Konstrukční mód však není pro vypracování této práce klíčový.

Historie, aktuální stav a porovnání

V době vytváření knihovny X-definice, na přelomu roku 2000, existovalo několik rozšířených jazyků pro validaci datové XML struktury. Významnými zástupci těchto jazyků jsou DTD, *Schematron* a RELAX NG [3]. Vzhledem k požadavkům na validaci XML dat byly všechny uvedené jazyky pro interní potřeby společnosti naprosto nedostačující [4, 5, 6]. Ke vzniku prvního standardu XML schémat (zde a po zbytek celé této práce uvažujeme pouze o XML schématech od konsorcia W3C), který byl označen jako *Recommendation*, došlo v průběhu roku 2001 a k jeho ustálení došlo až v roce 2004 – čili po vzniku X-definic [7]. Následující vývoj X-definic probíhal již paralelně s existencí XML schémat.

XML schémata se dočkala nového standardu až v roce 2012. Tento standard se však dosud dostatečně neuchytil (a to i kvůli problémům s limitovanou zpětnou kompatibilitou s verzí z roku 2004), a proto jediným široce používaným standardem pro validaci XML dat zůstala do dnešního dne specifikace z roku 2004, jmenovitě verze 1.0 (pokud nebude řečeno jinak, budou veškeré následující výskyty spojení *XML schéma* zamýšleny právě ve stran-

[1] `https://www.syntea.cz/xdefnice/`

dardu z roku 2004). Rozšíření nového standardu z roku 2012 mohlo být dále výrazně negativně ovlivněno i vzestupem popularity alternativních datových formátů jako jsou například JSON a YAML.

V dnešní době jsou X-definice mnohonásobně komplexnější než XML schéma (v porovnání s libovolnou verzí), a proto není možné jednoduše nahradit X-definice pomocí XML schémat. X-definice navíc disponují mnoha propracovanými funkcionalitami navíc.

Motivace

X-definice poskytují flexibilnější vyjádření předpisu než XML schémata, a proto umožňují vytvoření mnohem komplexnějšího popisu XML struktury. Tento zápis je navíc lépe čitelný [8]. Dále X-definice disponují více datovými typy, které jsou v mnoha případech intuitivnější nebo uživatelsky přívětivější. Podporují taktéž pokročilou validaci (například práce s identifikátory, validační hooky) a propojení s externími třídami v jazyce Java. Z pohledu normalizace a použitelnosti je však v dnešní době nutné podporovat standardizovaný a rozšířený popis datového formátu, který je používán například v rozhraních různých informačních systémů. V této oblasti (pro popis datové XML struktury) se nabízí pouze jediný obstojný standard a tím jsou právě XML schémata.

Problémy

V současnosti knihovna X-definice obsahuje oboustrannou implementaci převodu formátu X-definic a XML schémat. Toto řešení však obsahuje mnoho chyb, a to zejména v transformaci X-definic do XML schémat, která je stěžejním bodem této diplomové práce. Zde je nutné připomenout čtenáři skutečnost, že prostor popisovaných XML struktur pomocí předpisu X-definice výrazně převyšuje (z pohledu obsáhlosti i přesnosti) prostor, který lze popsat XML schématy.

Průvodce textem

V kapitole *Analýza* se čtenář seznámí se základní charakteristikou X-definic a XML schémat a přednostmi těchto předpisů. Dále jsou čtenáři popsány základní stavební bloky jednotlivých předpisů a jejich transformační obrazy, resp. vzory v jednotlivých předpisech. V rámci třetí kapitoly, *Návrh*, je čtenáři předložena obecná podoba implementace nového řešení, na kterou navazuje návrh řešení komplexních transformací. V závěru této kapitoly se budu věnovat návrhu samotného algoritmu pro oboustrannou transformaci a náležitostem, které takto navržený algoritmus obnáší. Poslední kapitola zabývající se algoritmem transformace se nazývá *Implementace a realizace*. Jak již jistě čtenáři na-

povědělo, v této kapitole je podrobněji popsána výsledná realizace algoritmů, jak pro převod X-definic, tak i XML schémat. Čtenář se taktéž dozví o technických podrobnostech implementace, včetně konkrétních ukázek kódu. Na závěr je v kapitole *Testování* předložena metodika testování transformačních algoritmů.

Kapitola 1

Cíl práce

Výstupem této diplomové práce je návrh a implementace algoritmu pro obousměrnou transformaci předpisu X-definice a XML schéma. V případě transformace X-definic do XML schémat je hlavním omezujícím faktorem, aby všechna data, která jsou validní pro X-definici, byla zároveň validní i pro XML schéma. Naopak, pokud jsou vstupní data nevalidní pro X-definici, nemusí být nutně nevalidní i pro XML schéma[2]. Při transformaci XML schémat do X-definice se předpokládá, že výsledek validace bude pro oba typy předpisu XML struktury shodný.

Výstup praktické části bude dostupný jako samostatný modul, který bude používat aktuální verzi knihovny X-definice jako závislost. Praktická část bude dále obsahovat dokumentaci řešení, ukázku použití a dostatečnou testovací množinu dat. Cílem práce je vytvořit algoritmus oboustranné transformace uvedených formátů, který v případě transformace X-definice na XML schéma pokryje většinu běžně používaného prostoru X-definic a zároveň výstupní XML schéma bude mít vlastnosti výše uvedené.

[2]Tento fakt plyne z rozdílu popisovaných prostorů XML struktur. Tomuto problému se věnuje značná část této práce.

Kapitola 2

Analýza

Obsahem kapitoly je představení formátu předpisů X-definice a XML schéma. Dále kapitola zahrnuje popis stávajícího řešení pro oboustrannou transformaci formátů X-definice a XML schéma a problémy tohoto řešení. V závěru kapitoly se budu věnovat porovnání zmiňovaných formátů, a to jak z pohledu způsobu popisu XML struktury, tak i z pohledu validace.

2.1 X-definice

Název X-definice označuje jak knihovnu, tak i formát předpisu XML struktury. Knihovna jako taková operuje nejenom se samotnými předpisy, ale lze taktéž definovat kolekci X-definic, která umožňuje libovolné propojení jednotlivých předpisů X-definic. Dále lze využívat například již zmíněné X-komponenty. Knihovna X-definice umožňuje použití dvou následujících módů [9, 10]:

- **Validační mód** – Slouží ke zprocesování a validaci vstupních dat. Uživatel v rámci validačního módu načte předpis (konkrétní X-definici, resp. kolekci X-definic) a následně zvolí data (XML struktura), která mají být zvalidována vůči vybranému předpisu.
- **Konstrukční mód** – Ze vstupní datové XML struktury se vytvoří předpis X-definice, který odpovídá této XML struktuře, tj. vstupní data jsou validní vůči vytvořenému předpisu.

V rámci této diplomové práce je pro nás zajímavý pouze **validační mód**, neboť cílem je navržení a implementace oboustranného transformačního algoritmu pro převod formátů X-definice a XML schéma, kdy XML schéma slouží právě pouze k validaci XML struktury. Příklad zápisu jednoduché X-definice je součástí ukázky 2.1.

Knihovna X-definice je optimalizována pro zpracovávání datových XML struktur o velikosti v řádech desítek gigabytů. Při interním zpracování takto velkých souborů nedochází k vytváření DOM celého souboru – z důvodu

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/3.2"
  xd:name="basic_x-definition"
  xd:root="a">
  <xd:declaration>
    type a_Type string(1, 8);
  </xd:declaration>
  <a a="optional a_Type()"/>
</xd:def>
```

Ukázka 2.1: Příklad předpisu (X-definice)

paměťové optimality. Tato skutečnost je jednou z dalších výhod předpisu X-definic nad XML schématy.

2.2 XML schéma

XML schémata jsou standardizovaným formátem, jehož funkčnost je díky velkému zájmu a mnohaletému používání ověřena. Zápis XML schémat je oproti X-definicím výrazně obsahově rozsáhlejší, přičemž výsledně paradoxně pokrývá menší prostor popisu XML struktury než X-definice. Dále na rozdíl od X-definic slouží výhradně k validaci XML dat (i přesto by bylo možné vytvořit aplikaci obdobnou konstrukčnímu módu jako je tomu u X-definic). Vzhledem k trendu poslední doby, kdy je snahou používat pro popis čitelnější jazyky, které jsou zároveň „méně ukecané“ (například YAML), by byly v tomto ohledu X-definice vhodným nástupcem XML schémat.

Pravidla pro tvorbu XML schémat vypadají od pohledu triviálně. Tato pravidla jsou však doplněna o mnoho klauzulí, které definují jaké uzly za jakých podmínek mohou být vnořeny do jiných uzlů (obecná podoba předpisu XML schémat). Ve výsledku se tak z relativně jednoduchého předpisu podoby stromové struktury XML schématu stane „nekonečná“ změť pravidel, kterou je nutné do posledního bodu dodržet. Navíc s narustající komplexitou datové XML struktury, kterou chceme pomocí XML schémat popsat, přibývá mnoho dalších omezujících pravidel [11].

I přes nevýhody, které XML schémata mají vůči X-definicím, se jedná v mnoha případech o dostačující nástroj pro definici struktury a validaci XML dat. Pokud bychom však chtěli popsat komplexnější datovou XML strukturu, je možné, že bychom velmi rychle narazili na limity XML schémat.

Součástí ukázky 2.2 je XML schéma, které popisuje totožnou XML strukturu jako předchozí ukázka X-definice 2.1. Již od prvního pohledu je viditelné, že zápis potřebný v XML schématu je znatelně rozsáhlejší.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          attributeFormDefault="unqualified"
          elementFormDefault="unqualified">
   <xs:simpleType name="a_Type">
      <xs:restriction base="xs:string">
         <xs:minLength value="1"/>
         <xs:maxLength value="8"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="a">
      <xs:complexType>
         <xs:attribute name="a" type="a_Type" use="optional"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Ukázka 2.2: Příklad předpisu (XML schéma)

2.3 Současná implementace transformace

Aktuální verze knihovny X-definice obsahuje algoritmus pro oboustrannou transformaci formátů X-definice a XML schéma. Řešení je postavené čistě nad prostředky samotného jazyku Java a knihovny X-definice – bez dalších knihoven třetí strany. Hlavním problémem tohoto řešení je, že vstupní předpisy (v obou směrech transformace) jsou načítány a interpretovány přímo v algoritmu transformace pomocí zcela vlastní implementace. Tento způsob načítání předpisů je zejména problémovým pro transformaci z X-definic do XML schémat, neboť X-definice se v čase, na rozdíl od XML schémat, rozvíjejí. Z toho plyne, že nové konstrukty X-definice tento algoritmus nezná a je mnohdy obtížné je doplnit do algoritmu transformace. Navíc v samotném načítání a zpracování předpisů se nacházejí chyby – tudíž samotný algoritmus transformace nemůže následně pracovat správně.

Dalším problémem je relativně „plochá struktura“ algoritmu transformace obsahující velké řetězení funkcí, která v mnoha ohledech vede na dosti přímočarý převod. Takový přístup je však ve spoustě případů nedostačující, a to zejména tehdy, kdy je nutné vytvořit **validní** XML schéma. Tato situace nastává hlavně v případech, kdy obraz XML schémat je pro daný vzor X-definice (nebo kolekci X-definicí) velmi odlišný od původního předpisu nebo když například v rámci XML schémat musí dojít k rozpadu nějakého uzlu na více uzlů (k pokročilejším transformacím se věnuje sekce 3.3 v následující kapitole). V opačném směru transformace (XML schéma do X-definice) tato „plochá“ implementace není tolik problematická, neboť X-definice disponují dostačujícími prostředky pro vyjádření přímočaré transformace.

Pokud bych měl na současné implementaci transformace něco vyzdvihnout

(na základě čeho by se dalo inspritovat), pak je to transformace datových typů, která je přehledná a strukturalizovaná.

Požadavek pana Trojana, vedoucího této práce, byl takový, abych ze současné implementace čerpal minimum poznatků (zejména po stránce technické) a více se zaměřil na nastudování standardů obou typů předpisů a dále se pak především soustředil na převod komplexnějších X-definic do XML schémat. Z tohoto důvodu zde dále nerozebírám současnou implementaci, která je v důsledku nevyhovující.

2.4 Porovnání předpisů XML struktur

2.4.1 Syntaxe formátů

Jak X-definice tak i XML schéma jsou předpisy založené na jazyce XML (viz ukázkové příklady 2.1 a 2.2). Z této skutečnocsti vyplývá, že k předpisům samotným lze přistupovat pomocí DOM (konkrétně XML DOM)[12]. Oba zápisy předpisů tudíž podporují určité typy uzlů, atributů v uzlech a textových uzlů. Pro většinu uzlů lze oboustranně, případně pouze jednostranně, dohledat obrazy, viz 2.6.1. X-definice obsahují speciální atribut *xd:script*, který umožňuje přiřadit danému uzlu mimo jiné například i pokročilou validaci (více o validaci v sekci 2.5).

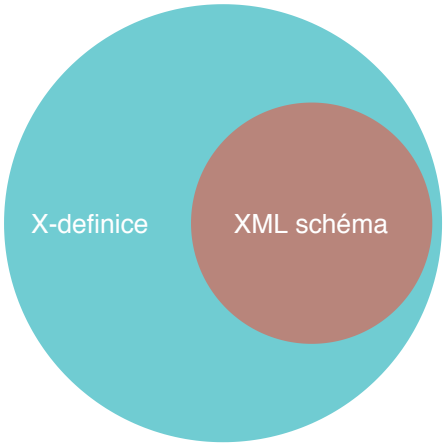
2.4.2 Pokrytí prostorů XML struktur

XML schéma pokrývá významně menší prostor popisu XML struktur než X-definice. Obecně platí předpoklad, že cokoliv co lze zapsat v XML schématech, lze následně zapsat i v X-definicích – čili prostor XML struktur, který lze zapsat pomocí XML schémat je podprostorem XML struktur, který lze definovat X-definicema, viz vizualizace 2.1. Modře vyznačený prostor XML struktur pro XML schéma kopíruje přesně určitou část podprostoru XML struktur X-definic, čili se jedná o takové předpisy, které lze bezztrátově oboustranně transformovat.

Naopak však toto tvrzení **neplatí**. Při transformaci X-definice do XML schéma je možné dohledat obecnější XML schéma, které se řídí určitými pravidly (viz dále). Z tohoto důvodu jsem při analýze transformace X-definic do XML schémat strávil velké množství času nad hledáním vhodného obrazu v XML schéma pro daný vzor předpisu X-definic. Vždy bylo potřeba dohledat obraz XML schémat takový, který splňuje následující:

- výsledný obraz XML schémat musí být validní s ohledem na všechna pravidla XML schémat[3],

[3]Toto pravidlo je náročné splnit zejména při transformacích, ve kterých z přímočarého převodu X-definice vznikne předpis v XML schématech takový, že jsou mnohokrát porušena pravidla struktury předpisu.



Obrázek 2.1: Vizualizace pokrytí prostorů XML struktur

- všechna validní data pro vzor předpisu X-definice jsou zároveň validní i pro obraz předpisu v XML schématu,

- XML schéma musí být takový předpis, aby počet shodných výsledků validace pro nevalidní data byl maximální – co nejvyšší.

Na první pohled se může tento úkol zdát jako triviální. Pokud se však k němu přidá fakt, že mnoho X-definic nelze převést přímočaře do XML schémat a je občas problém najít „rozumné“ řešení (rozumějme takové, aby pokrývalo co nejpodobnější prostor předpisů XML struktur) pro daný vzor X-definice, pak se z této úlohy stává časově náročná analýza.

2.5 Porovnání validace

Oba formáty předpisů kontrolují strukturu XML dat. Tato kontrola obnáší nejen validaci stromové struktury uzlů, ale taktéž validaci atributů, výskytů jednotlivých prvků a datových typů. Dále se kontrolují i jmenné prostory (více o jmenných prostorech viz 2.6.7). Jak X-definice tak i XML schéma umožňují kontrolovat unikátnost výskytu hodnot, přičemž v obecném měřítku mají X-definice mnohem robustnější implementaci kontroly a práce s unikátností hodnot (víc viz 2.6.8).

X-definice navíc umožňují použití doplňující pokročilejší validace, včetně například proměnných nebo validačních hooků. Tato validace je většinou řešena přes atribut *xd:script* (2.6.2.1). V tomto ohledu XML schémata nemají žádnou ekvivalentní funkcionalitu, a proto budou tyto validační a jiné předpisy v rámci transformace ignorovány.

```
<a xd:script="occurs 0..3">required typeString()</a>
```

Ukázka 2.3: Příklad předpisu elementu (X-definice)

```
<xs:element maxOccurs="3" minOccurs="0" name="a" type="typeString"/>
```

Ukázka 2.4: Příklad předpisu elementu (XML schéma)

2.6 Transformace dílčích předpisů

Tato sekce obsahuje popis analýzy oboustranné případně pouze jednosměrné transformace (pokud neexistuje relevantní obraz v XML schématu pro zápis X-definice) elementů a atributů. Součástí sekce jsou i ukázky přímočarých převodů jednotlivých dílčích částí. Je nutné si zde uvědomit, že **výsledná transformace komplexnějšího předpisu se může značně lišit od částečných transformací** a to zejména při převodu X-definice do XML schéma – většinou z důvodu doplňujících pravidel pro podobu struktury XML schématu.

Většinu informací zmíněných v této sekci jsem čerpal ze standardu XML schémat a příruček ke knihovně X-definice [11, 13, 14, 15, 10, 16].

2.6.1 Uzly

2.6.1.1 Element

Element neboli model je základní stavební prvek jak v X-definici, tak i v XML schématu. Určuje základní hierarchii popisované XML struktury. Zatímco v X-definici je název elementu definován jako název tagu, tak v XML schématech je použit atribut *name* v uzlu *xs:element*. Transformace je však bezztrátová v tomto případě, viz ukázky 2.3 a 2.4. Oboustranně podporované attributy (výčet atributů v XML schéma):

- *minOccurs* – u X-definice řešeno přes obsah atributu *xd:script*,
- *maxOccurs* – u X-definice řešeno přes obsah atributu *xd:script*,
- *type* – u X-definice řešeno přes obsah elementu,
- *ref* – u X-definice řešeno přes obsah atributu *xd:script*,
- *nillable* – příznak, zda elementu může být přiřazena hodnota *null*.

Podporované attributy v XML schéma při transformaci z X-definice:

- *form* – vyhodnocuje se na základě jména elementu v X-definici a nastavení jmenného prostoru XML schématu

```
<a>
    <b xd:script="occurs 0..3"/>
    <c/>
</a>
```

Ukázka 2.5: Příklad přímého vložení elementu do elementu (X-definice)

Mixed element Mixed element znamená, že uzel typu element může obsahovat jak textové uzly, tak zároveň i další typy uzlů. Zatímco v X-definici lze přesně popsat, jak bude takový kombinovaný obsah elementu vypadat (dokonce i jaký datový typ textu bude použit), tak u XML schémat lze pouze označit vložený uzel *xs:complexType*, do příslušného elementu, pomocí atributu *mixed* s hodnotou *true*.

V X-definici existuje více druhů zápisu mixed elementu:

- atribut *xd:text* – nejbližší ekvivalence k atributu *mixed* v XML schéma,
- atribut *xd:textcontent* – obdoba atributu *xd:text* v X-definici (více viz 2.6.2.2 a 2.6.2.3),
- explicitní zápis kombinovaného obsahu – nemá ekvivalenci v XML schéma, jedná se o **ztrátovou transformaci**.

Více k transformaci elementů s kombinovaným obsahem se věnuje subsekce 3.3.1.1 v rámci kapitoly *Návrh*.

Vložení uzlu Pokud element v XML struktuře obsahuje další element, je nutné, aby v XML schématech byl vkládaný element vložen navíc do uzlu definující typ partikulární skupiny elementů (viz 2.6.1.3) a tato celá skupina byla navíc vložena do uzlu *xs:complexType*. Z toho plyne, že předpisech v XML schémat **není možné** vkládat elementy přímo do dalšího elementu (příklad 2.6)[4,5]. Na základě tohoto „nešťastného“ pravidla velmi bobtná struktura předpisu v XML schématech (naštěstí X-definice tímto netrpí, příklad 2.5).

2.6.1.2 Definice skupiny elementů

Slouží k zadefinování skupiny elementů, na které se lze následně odkazovat (jako na skupinu). Zatímco v X-definici se tato skupina zapisuje pomocí uzlu *xd:mixed*, tak v XML schéma je tento uzel pojmenován *xs:group*. Pojmenováním však rozdíly nekončí.

V X-definici implicitně nezáleží na pořadí elementů umístěných v takové skupině, navíc můžeme určit počet jejich výskytů. V XML schématech musíme

[4] https://www.w3.org/TR/xmlschema-1/#declare-element
[5] https://www.w3.org/TR/xmlschema-1/#element-complexType

```
<xs:element name="a">
     <xs:complexType>
        <xs:sequence>
           <xs:element maxOccurs="3" minOccurs="0" name="b"/>
           <xs:element name="c"/>
        </xs:sequence>
     </xs:complexType>
</xs:element>
```

Ukázka 2.6: Příklad přímého vložení elementu do elementu (XML schéma)

```
<xd:mixed xd:name="m">
    <n/>
    <o/>
</xd:mixed>
```

Ukázka 2.7: Příklad definice skupiny elementů (X-definice)

```
<xs:group name="m">
   <xs:all>
      <xs:element name="n"/>
      <xs:element name="o"/>
   </xs:all>
</xs:group>
```

Ukázka 2.8: Příklad definice skupiny elementů (XML schéma)

do definice skupiny elementů vložit uzel určující typ partikulární skupiny[6]. Pokud bychom však zvolili v XML schématu partikulární skupinu takovou, aby nezáleželo na pořadí uzlů, tj. uzel *xs:all*, potom nemůžeme určit kardinalitu jednotlivých elementů uvnitř partikulární skupiny (více k uzlům partikulárních skupin v 2.6.1.3).

Jedná se proto opět **o potencionálně ztrátovou transformaci**, za předpokladu, že definice skupiny elementů obsahuje element mající maximální kardinalitu vyšší než jedna. O této transformaci se dodatečně zmiňuje i subsekce 3.3.1.3. V ukázkách 2.7 a 2.8 jsou k vidění příklady bezztrátových transformací definic skupin elementů.

Odkaz na definici skupiny elementů Odkaz na definici skupiny elementů probíhá analogicky. Opět se zde liší klíčové uzly, v rámci X-definice je použit opět uzel *xd:mixed* s atributem *xd:script* obsahující referenci na definici sku-

[6] `https://www.w3.org/TR/xmlschema-1/#declare-namedModelGroup`

```
<xd:sequence>
  <a/>
  <b/>
</xd:sequence>
```

Ukázka 2.9: Příklad sekvence množiny elementů (X-definice)

piny elementů. V XML schématech je použit uzel *xs:group* s atributem *xd:ref* obsahující referenci na definici skupiny elementů.

Standard XML schémat umožňuje pro uzel *xs:group* mající odkaz na definici skupiny elementů nastavit kardinalitu vyšší než jedna. Toto však není možné provést na úrovni X-definice a proto je v tomto případě nutné zvážit pokročilejší transformaci. Ve výsledku se však nejedná nutně o ztrátovou transformaci.

2.6.1.3 Partikulární skupina uzlů

Jedná se celkem o tři skupiny definující, v jakém pořadí se mohou vyskytovat uzly, které se nachází uvnitř těchto skupin, resp. uzlů. V zásadě první dvě skupiny jsou z pohledu oboustranné transformace beztrátové, viz ukázky pro uspořádanou množinu 2.9, 2.10 a výběr z množiny 2.11 a 2.12.

Uspořádaná množina (sekvence) Základní vlastností této množiny je, že uzly které obsahuje, musí být v rámci datové XML struktury ve stejném pořadí jako v předpisu.

V rámci X-definice je tato množina definována uzlem *xd:sequence*, resp. v XML schéma uzlem *xs:sequence*. V obou případech mohou uzly této skupiny obsahovat následující typy uzlů[7]:

- element,
- reference na definici skupiny elementů,
- partikulární skupina elementů.

Zároveň jsou pro oboustrannou transformaci podporované následující atributy:

- *minOccurs* – u X-definice řešeno přes obsah atributu *xd:script*,
- *maxOccurs* – u X-definice řešeno přes obsah atributu *xd:script*.

[7] `https://www.w3.org/TR/xmlschema-1/#element-sequence`

```
<xs:sequence>
  <xs:element name="a"/>
  <xs:element name="b"/>
</xs:sequence>
```

Ukázka 2.10: Příklad sekvence množiny elementů (XML schéma)

```
<xd:choice>
  <p/>
  <q/>
</xd:choice>
```

Ukázka 2.11: Příklad výběru elementu z množiny (X-definice)

```
<xs:choice>
   <xs:element name="p"/>
   <xs:element name="q"/>
</xs:choice>
```

Ukázka 2.12: Příklad výběru elementu z množiny (XML schéma)

Výběr uzlu z množiny Jak jistě název napovídá, v této množině uzlů lze vybrat určitý element z množiny. Počet vybraných elementů lze ovlivnit atributy kardinality. Vybrané elementy se tudíž mohou opakovat.

V předpisu X-definice definována uzlem *xd:choice*, resp. v XML schéma uzlem *xs:choice*. Obdobně jako u výše uvedené uspořádané množiny, může uzel výběru z množiny obsahovat tři typy uzlů (element, reference na definici skupiny elementů, partikulární skupina elementů)[8]. To stejné platí i pro atributy, tj. atributy kardinality jsou podporované jak v X-definici tak XML schéma.

Neuspořádaná množina V porovnání s předešlými dvěmi zástupci z partikulárních skupin elementů se jedná v mnoha ohledech o potencionálně **ztrátovou transformaci**, v závislosti jak na samotném zápisu daného uzlu a jeho obsahu, tak i na jeho pozici v rámci struktury předpisu. Jedná se v podstatě o jednu z nejproblémovějších transformací z X-definic do XML schémat (v opačném směru jsou problémy minimální).

V XML schématech neuspořádaná množina umožňuje vybrat každý uzel z množiny maximálně jednou[9]. Dále u této množiny v XML schématech nelze použít vyšší kardinalitu než jedna. To v důsledku znamená, že se jedná de

[8] `https://www.w3.org/TR/xmlschema-1/#element-choice`

[9] Explicitně lze kardinalitu jednotlivých uzlů změnit tak, aby jejich výskyt nebyl povinný.

```
<xd:mixed>
  <a/>
  <b/>
</xd:mixed>
```

Ukázka 2.13: Příklad neuspořádané množiny elementu (X-definice)

```
<xs:all>
  <xs:element name="a"/>
  <xs:element name="b"/>
</xs:all>
```

Ukázka 2.14: Příklad neuspořádané množiny elementu (XML schéma)

facto o výběr z množiny (popsaný výše), přičemz každý uzel může být vybrán maximálně jednou (a nebo nemusí být vybrán vůbec). Tyto vlastnosti se velmi liší vůči definici neuspořádané množiny v X-definicích.

V X-definici se tento uzel značí stejně jako definice skupiny elementů nebo reference na takovou definici, avšak bez udání jména/reference, konkrétně se jedná o uzel *xd:mixed*. V XML schéma je pro tento účel určen uzel *xs:all*.

Použití neuspořádané množiny elementů v rámci XML schémat se musí řídit mnoha pravidly, resp. omezeními[10]. Z tohoto důvodu se jedná v mnoha ohledech o ztrátovou transformaci, jejíž kompletní řešení je popsáno v subsekci 3.3.1.2. Příklad bezztrátové transformace dílčí části X-definice je součástí ukázek 2.13 a 2.14

2.6.1.4 Element any

Základní definicí elementu typu *any* je element s libovolným pojmenováním. Další části definice jsou odlišné z pohledu X-definice a XML schémat. V obou předpisech je značení obdobné, v X-definicích se používá uzel *xd:all* a v XML schématech *xs:all*.

Element any v X-definicích V základu se jedná pouze o element s libovolným jménem. Navíc je možné dodefinovat požadované atributy, které má takový element obsahovat. Element *any* lze dodatečně rozšířit pomocí atributu *xd:script* o následující hodnoty:

- *moreAttributes* – povoluje existenci dalších nedefinovaných atributů přímo v elementu *any*,

[10] `https://www.w3.org/TR/xmlschema-1/#coss-modelGroupl`

- *moreElements* – povoluje existenci nedefinovaných elementů vložených do elementu *any*,
- *moreText* – povoluje vložení textu do elementu *any*.

Element any v XML schématech Na rozdíl od X-definic nelze definovat požadované atributy. Není taktéž možné definovat konkrétní rozšíření elementu, lze však využít atributu *processContents*[11], jehož hodnoty umožňují následující:

- ***strict*** – *There must be a top-level declaration for the item available, or the item must have an xsi:type, and the item must be ·valid· as appropriate.*,
- ***skip*** – *No constraints at all: the item must simply be well-formed XML.*,
- ***lax*** – *If the item has a uniquely determined declaration available, it must be ·valid· with respect to that definition, that is, ·validate· if you can, don't worry if you can't.* [13].

V obecném měřítku se jedná o **potencionálně ztrátovou transformaci**, a to teoreticky pro obě strany transformace zároveň. Navíc element *xs:all* v rámci XML schémat má omezující pravidla na místo výskytu – například nelze takový element umístit do kořene XML schématu, zatímco v rámci X-definic toto provést lze.

Vzhledem k tomu, že element *any* se nevyskytuje ani v rozšířeném datasetu pro testování realizovaného algoritmu v rámci této práce a jedná se o relativně obsáhlou problematiku transformace[12], nebudu se nadále tématem transformace tohoto elementu zabývat.

2.6.2 Atributy X-definice

2.6.2.1 xd:script

Atribut X-definice umožňující doplnit mnoho dodatečných předpisů k danému uzlu. Přehled hlavních typů doplňujících předpisů, které lze do tohoto atributu vložit:

- **kardinalita** – definice výskytu daného uzlu,
- **reference** – definice odkazu na jiný element/skupinu elementů,
- **validace** – pokročilá logika validace (nelze uplatnit v transformaci do XML schéma, budou ignorovány), příklad: volání externí validační funkce z jazyka Java,

[11] `https://www.w3.org/TR/xmlschema-1/#Wildcards`

[12] Zejména z pohledu analýzy – které případy lze kdy, jak a za jakých podmínek transformovat.

```
<A xd:script="ref refA; occurs *" />
```

Ukázka 2.15: Příklad použití atributu xd:script (X-definice)

```
<xs:element maxOccurs="unbounded" minOccurs="0" name="A" type="refA"/>
```

Ukázka 2.16: Příklad transformace atributu xd:script (XML schéma)

- **akce** – doplňující akce, které se mají provést (nelze uplatnit v transformaci do XML schéma, budou ignorovány), příklad: *onTrue, onFalse, setErr, ...*,
- **proměnné** – definice proměnných, které mají být použity v rámci elementu a vložených uzlů do něj (nelze ve většině případů uplatnit v transformaci do XML schéma), příklad: *String role = ...*.

Výskyt uzlu a odkazy jsou standardní prvky, které se nachází taktéž v XML schématech. Lze je navíc oboustranně bezztrátově transformovat. Příklad bezztrátové transformace je součástí ukázek 2.15 a 2.16.

Více podrobností o transformaci kardinality, resp. reference, mezi předpisy X-definice a XML schéma lze dohledat v subsekci *Výskyty* (2.6.3), resp. *Odkazy* (2.6.6).

2.6.2.2 xd:text

Umožňuje do uzlu typu element vložit na libovolná místa libovolný textový uzel, čili element výsledně může obsahovat více textových uzlů. Obsah textových uzlů je následně možné validovat zvlášť.

V XML schématech neexistuje ekvivalentní atribut k *xd:text*, jedná se však o přibližnou ekvivalenci k atributu *mixed* (v rámci elementu v XML schématech), která se liší pouze ve formě rozšířené validace, která může být provedena pouze v X-definicích.

2.6.2.3 xd:textcontent

Umožňuje do uzlu typu element vložit na libovolná místa libovolný textový uzel, čili element výsledně může obsahovat více textových uzlů. Textové uzly jsou následně spojeny do jednoho a obsah je validován dohromady.

V XML schématech neexistuje ekvivalentní atribut k *xd:textcontent*, jedná se však o zdánlivě přibližnou ekvivalenci k atributu *mixed* elementu v XML schéma.

2.6.2.4 xd:attr

Jedná se obdobu elementu *xd:any*, avšak pro atributy. To znamená, že předpis umožňuje v daném uzlu použít atribut s libovolným jménem. Dále je možné v X-definici definovat datový typ daného atributu.

V XML schématech neexistuje ekvivalentní atribut k *xd:attr*, jedná se však o přibližnou ekvivalenci k uzlu *xs:anyAttribute* v rámci XML schémat. Tento uzel obdobně jako *xs:any* využívá atribut *processContents* (více o tomto atributu lze dohledat v paragrafu 2.6.1.4).

2.6.3 Výskyt uzlů

2.6.3.1 X-definice

Zápis výskytu uzlů je různorodý a z historických důvodů v dnešní verzi knihovny X-definice existuje v mnoha případech více způsobů, jak zapsat tutéž kardinalitu (příklady zápisů viz ukázka 2.17). Všechny zápisy lze však bezztrátově převést do XML schémat a naopak. Seznam klíčových slov pro výskyt uzlů je následující:

- **optional**, **?** – uzel se může vyskytnout maximálně jednou,
- **required** – uzel se musí vyskytnout právě jednou (výchozí stav),
- ***** – uzel se může vyskytnout neomezeně krát (minimálně ani jednou),
- **+** – uzel se musí vyskytnout minimálně jednou (maximálně neomezeně).

Dále je možné použít následující zápisy vyskytů:

- **n** – uzel se musí vyskytnout právě n-krát,
- **m..n** – uzel se musí vyskytnout minimálně m-krát a zároveň maximálně n-krát,
- **m..*** – uzel se musí vyskytnout minimálně m-krát.

Ve výchozím stavu je uzel povinný s kardinalitou 1.

2.6.3.2 XML schéma

Oproti X-definici je zápis výskytu v XML schématech méně variabilní, avšak zcela dostačující a pokrývá běžné scénáře. Pro zápis se používají pouze dva následující atributy:

- **minOccurs** – definice minimálního počtu výskytů daného uzlu, výchozí hodnota je 1,
- **maxOccurs** – definice maximálního počtu výskytů daného uzlu, výchozí hodnota je 1.

```
<a1 xd:script="occurs optional"/>
<a2 xd:script="occurs 0..1"/>
<b1 xd:script="occurs *"/>
<b2 xd:script="occurs 0..*"/>
<c1 xd:script="occurs +"/>
<c2 xd:script="occurs 1..*"/>
<d1 xd:script="occurs required"/>
<d2 xd:script="occurs 1"/>
```

Ukázka 2.17: Příklad předpisů pro definici výskytu elementů (X-definice)

Ve výchozím stavu je uzel povinný, tak jako tomu je u X-definic. Pokud vyžadujeme, aby daný uzel byl nepovinný (za předpokladu, že podporuje nastavení kardinality), stačí nastavit hodnotu atributu *minOccurs* na 0. Naopak pokud potřebujeme, aby uzel měl neomezený počet výskytů, potom nastavíme hodnotu atributu *maxOccurs* na *unbounded*.

2.6.4 Datové typy

Datové typy definují typy hodnot, které se mohou vyskytovat v atributech a textových uzlech. Pokud porovnáme X-definice s XML schématy, velmi rychle zjistíme, že X-definice mají mnohem četnější množinu datových typů. Navíc historicky snahou tvůrců knihovny X-definice bylo, aby tato knihovna obsahovala všechny (běžně používané) datové typy z XML schémat. Lze tak usoudit, že transformace datových typů z XML schémat do X-definic nebude problémová a navíc bude zcela bezztrátová, neboť všechny potřebné datové typy jsou již součástí knihovny X-definic. Pokud by se však náhodou objevil nějaký datový typ v XML schématech, který není součástí X-definic a je potřeba jej transformovat, nejjednodušší bude takový datový typ do X-definic doplnit.

2.6.4.1 Omezení datových typů

Abychom mohli hodnotu daného datového typu upřesnit, můžeme použít tzv. *constraints*, neboli česky omezení. Zatímco v případě X-definic jsou omezení zapisována přímo v deklaraci datového typu (viz ukázka 2.18, kde omezujeme datový typ *int* na rozsah hodnot 1–999), tak v XML schématech je nutné pro definici datového typu použít uzel *xs:restriction* (viz ukázka 2.19, obsahující bezztrátovou transformaci k ukázce 2.18). Pokud však žádná omezení na datový typ neexistují, je možné v rámci XML schémat použít i uzel *xs:extension*.

Druhy omezení na datové typy jsou shodné pro oba typy předpisů. Rozlišujeme následující omezení:

- **length** – délka obsahu datového typu (omezení shora i zdola),

```
int(1, 999)
```

Ukázka 2.18: Příklad zápisu datového typu s omezením (X-definice)

- **minLength** – minimální délka obsahu datového typu,
- **maxLength** – maximální délka obsahu datového typu,
- **pattern** – regulární výraz, který musí obsah datového typu splňovat (je možné použít více regulárních výrazů),
- **enumeration** – množina hodnot, kterých musí obsah datového typu nabývat,
- **maxExclusive** – maximální hodnota, které může obsah datového typu nabývat (pouze pro číselné datové typy),
- **maxInclusive** – maximální hodnota (včetně), které může obsah datového typu nabývat (pouze pro číselné datové typy),
- **minExclusive** – minimální hodnota, které může obsah datového typu nabývat (pouze pro číselné datové typy),
- **minInclusive** – minimální hodnota (včetně), které může obsah datového typu nabývat (pouze pro číselné datové typy),
- **totalDigits** – maximální počet číslic v celé části desetinného čísla (pouze pro číselné datové typy),
- **fractionDigits** – maximální počet desetinných číslic v desetinném čísle (pouze pro číselné datové typy),
- **whiteSpace** – způsob, jakým zpracovávat bílé znaky.

Dále X-definice používají omezení ***base*** a ***item***, která jsou v rámci XML schémat pokryta většinou atributem *base* v uzlu *xs:restriction*, nebo v případě datového typu *union* může být použit i atribut *memberTypes* umístěný v uzlu *xs:union*.

Z výše uvedené analýzy vyplývá, že omezení datových typů lze bez problému oboustranně bezztrátově transformovat, neboť pokrývají totožný prostor omezení.

```
<xs:restriction base="xs:integer">
  <xs:minInclusive value="1"/>
  <xs:maxInclusive value="999"/>
</xs:restriction>
```

Ukázka 2.19: Příklad zápisu datového typu s omezením (XML schéma)

```
<A
  e="double(); default '1.2'"
  f="double(); fixed '1.3'"
/>
```

Ukázka 2.20: Příklad zápisu výchozí a fixní hodnoty (X-definice)

```
<xs:element name="A">
  <xs:complexType>
    <xs:attribute default="1.2" name="e" type="xs:double"/>
    <xs:attribute fixed="1.3" name="f" type="xs:double"/>
  </xs:complexType>
</xs:element>
```

Ukázka 2.21: Příklad zápisu výchozí a fixní hodnoty (XML schéma)

2.6.4.2 Výchozí a fixní hodnota

Jak v X-definicích, tak i v XML schématech je možné určit výchozí nebo fixní hodnotu datového typu. Tato možnost se vztahuje tudíž pouze na uzly podporující určení datového typu, tj. textové uzly a atributy. Výchozí hodnota se v obou typech předpisů značí ***default*** a fixní hodnota ***fixed***. Jedná se o bezztrátovou obousměrnou transformaci (viz zmíněné ukázky dále). Zápisy se však v rámci předpisů liší, viz ukázka 2.20 pro X-definice a 2.21 pro XML schéma – v obou případech element A obsahuje atribut e s výchozí hodnotou a atribut f s fixní hodnotou.

2.6.4.3 X-definice

Vzhledem k faktu, že X-definice definují mnohem více datových typů, bude potřeba navrhnout a implementovat transformaci datových typů neznámých pro XML schéma. Vzhledem k omezeným možnostem XML schémat bude výstupem transformace nejčastěji datový typ *string* v kombinaci s použitím regulárního výrazu pro obsah. V rámci této práce budou řešeny pouze běžně používané datové typy v rámci X-definic. Datové typy mající přímý ekvivalent v XML schématech budou implicitně transformovány. I přesto jsem ve

spolupráci s vedoucím diplomové práce vytipoval datové typy, jejichž transformaci není nutné v rámci implementace řešit (viz dále). Jedná se o datové typy, které splňují alespoň jeden z následujících bodů:

- v současné době jsou již pouze historickým přežitkem,
- nelze jejich funkcionalitu v XML schématech zaručit (většinou se jedná o takové datové typy, které vyhodnocují strukturu XML),
- není nutné řešit jejich transformaci (minimálně používané datové typy).

Příkladem takových datových typů jsou: *QNameURI, QNameList, QNameURIList, pic.*

Taktéž jsem ve spolupráci s vedoucím práce vydefinoval některé datové typy, které v současné době mají odlišnou interní implementaci a z tohoto důvodu není možné je jednoduše transformovat. Jedná se o následující datové typy: *email, emailList, file, uri, uriList, url, urlList.*

Více podrobností k analýze a návrhu (pokročilých) transformací datových typů lze najít ve zbytku této subsekce a v sekci 3.5.2.1 (kapitola *Návrh*). Níže následuje výčet skupin datových typů X-definice a popis způsobu provedení jejich transformace.

Porovnávání části řetězce Jedná se o jednoduchou transformaci datového typu X-definice. V XML schématech bude potřeba použít datový typ *string* v kombinaci s příslušným regulárním výrazem.

Jedná se o následující datové typy: *contains, containsi, ends, endsi, eq, eqi, regex, starts, startsi.*

Desetinná čísla X-definice podporují podrobnější popis desetinného čísla než je v XML schématu. Navíc v X-definicích může být oddělovačem celé a desetinné části čárka. Z těchto důvodů nelze přímočaře transformovat desetinné číslo z X-definice do XML schéma.

Jedná se o následující datové typy: *dec.*

Datum a čas XML schéma podporuje mnoho formátů času a data, všechny jsou však založeny na ISO formátech. Pokud tedy chceme využít jiný typ formátu času nebo data v XML schématech, musíme použít datový typ *string*, který můžeme rozšířit o regulární výraz. Tato skutečnost je velkým handicapem XML schémat.

X-definice umožňují použití datových typů pro čas a datum společně s relativně standardizovanou maskou popisu formátu data/času. Tato maska je podrobně popsána v dokumentaci X-definic[10]. Příklad použití datového typu *xdatetime* v kombinaci s maskou formátu je součástí ukázky 2.22 (atribut *DatumCasDN*).

```
<ObjStrankaDN DatumCasDN="required xdatetime('d.M.yyyy H:mm[:ss]')"/>
```

Ukázka 2.22: Příklad použití datového typu *xdatetime* (X-definice)

Pro transformaci této masky bude použita třída *DateTimeFormatAdapter* z původní implementace transformace předpisu X-definice do XML schéma. Tato třída převádí vstupní masku na datový typ *string* s odpovídajícím regulárním výrazem.

Jedná se o následující datové typy: *xdatetime, dateYMDhms, emailDate.*

Case-insensitive V XML schématech neexistuje žádný implicitní datový typ podporující *case-insensitive*. Z tohoto důvodu bude nutné vytvořit transformaci, která dokaže ze vstupního regulárního výrazu, vytvořit *case-insensitive* regulární výraz.

Jedná se o následující datové typy: *containsi, endsi, eqi, startsi.*

Množina s parametrizovatelným oddělovačem Jedná se o takové datové typy X-definice, které umožňují zapsání množiny hodnot s použitím definovaného separátoru. Tyto datové typy by měly být řešeny převodem do datového typu *string* společně s použitím regulárního výrazu, který umožní jak zapsat prvky z dané množiny, tak i kontrolovat požadovaný oddělovač.

Často však samotné prvky množiny musejí nabývat konkrétních hodnot (například. ISOlanguage z XML schéma). Z tohoto důvodu se může jednat o **ztrátovou transformaci**.

Jedná se o následující datové typy: *ISOlanguages, NCNameList.*

Datové typy s pevným formátem Datové typy X-definice, které z definice mají známý a pevně stanovený formát, který lze vyjádřit regulárním výrazem.

Jedná se o následující datové typy: *an, MD5, num.*

List Zápis množiny prvků definovaných jedním konkrétním datovým typem. Prvky jsou odděleny mezerou a mohou se na ně vztahovat dodatečná omezení. Datový typ *list* lze bezztrátově transformovat do XML schémat (a naopak). Příklad oboustranně bezztrátové transformace datového typu *list* je součástí ukázek 2.23 a 2.24.

Union Na rozdíl od datového typu *list* lze určit více datových typů, které mají být vloženy do *union*. Naopak nelze například určit počet prvků, které mohou být v rámci obsahu vloženy. Obdobně jako pro datový typ *list* lze provést oboustranně bezztrátovou transformaci datového typu *union*, viz ukázky 2.25 a 2.26.

```
<a>
    list(%item=int(%minInclusive='1', %maxInclusive='10'),
        %length='3')
</a>
```

Ukázka 2.23: Příklad použití datového typu *list* (X-definice)

```
<xs:element name="a">
   <xs:simpleType>
      <xs:restriction>
         <xs:simpleType>
            <xs:list>
               <xs:simpleType>
                  <xs:restriction base="xs:int">
                     <xs:minInclusive value="1"/>
                     <xs:maxInclusive value="10"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:list>
         </xs:simpleType>
         <xs:length value="3"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Ukázka 2.24: Příklad použití datového typu *list* (XML schéma)

```
<a>
    union(%item=[
             int(%minInclusive='1', %maxInclusive='10'),
             string(%enumeration=['A', 'B', 'C'])
   ])
</a>
```

Ukázka 2.25: Příklad použití datového typu *union* (X-definice)

```
<xs:simpleType name="a_union_int">
  <xs:restriction base="xs:int">
     <xs:minInclusive value="1"/>
     <xs:maxInclusive value="10"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="a_union_string">
  <xs:restriction base="xs:string">
     <xs:enumeration value="A"/>
     <xs:enumeration value="B"/>
     <xs:enumeration value="C"/>
  </xs:restriction>
</xs:simpleType>

...

<xs:element name="a">
   <xs:simpleType>
      <xs:union memberTypes="a_union_int a_union_string"/>
   </xs:simpleType>
</xs:element>
```

Ukázka 2.26: Příklad použití datového typu *union* (XML schéma)

2.6.5 Deklarace

Deklarace slouží k zadefinování nového datového typu, který je následně možné použít v rámci X-definice, resp. XML schéma. Zatímco u X-definic deklarace slouží k definici datových typů, metod a proměnných, tak u XML schémat je možné pomocí deklarace zadefinovat uzly a datové typy vztahující se pouze k datové XML struktuře (proměnné a metody nejsou v XML schématech podporovány). X-definice navíc podporují definování rozsahu deklarace, rozlišujeme:

- **lokální rozsah** – deklarace je dostupná (může být referována) pouze v rámci X-definice, ve které je definována,
- **globální rozsah** – deklarace je dostupná v rámci celé kolekce X-definic.

Oba typy předpisů umožňují řetězení deklarací, tj. deklarace A se může odkazovat na deklaraci B. Způsob, jakým lze odkazovat na deklarace je popsán v subsekci 2.6.6, se nazývá reference.

Deklarace je možné vzájemně bezztrátově transformovat. Vzhledem k rozdílům, které plynou z analýzy předpisů, není vždy možné deklarace a reference na ně přímočaře transformovat. Z tohoto důvodu se transformaci deklarací a referencí věnuji důkladněji až v kapitole *Návrh*, viz subsekce 3.3.2.4.

```
<xd:declaration>
  type c_Type long(1, 10);
</xd:declaration>
```

Ukázka 2.27: Příklad předpisu deklarace (X-definice)

2.6.5.1 X-definice

Deklarace v X-definicích se definuje pomocí klíčového uzlu *xd:declaration*. Tento uzel musí ležet v kořeni X-definice a typicky obsahuje klíčové slovo *type*, které definuje nový datový typ. **Názvy datových typů musí být unikátní**[13]. Dále uzel *xd:declaration* může například obsahovat definici proměnné nebo *uniqueSet* (více o *uniqueSet* viz subsekce 2.6.8). Proměnné a další typy deklarací, které nejsou zmíněny v této práci, nejsou zároveň podporovány ze strany XML schémat – z tohoto důvodu nejsou nadále zmiňovaný a z pohledu transformace jsou ignorovány. V ukázce 2.27 je vidět příklad zápisu deklarace nového datového typu *c_Type*, který je typu *long* a může nabývat hodnot 1 až 10.

Deklarací může být implicitně i element nebo definice skupiny elementů. V obou případech se musí jednat o uzly, které leží v kořenu X-definice a zároveň nejsou kořenovými elementy X-definice (více o kořenových elementech X-definice viz 2.7.1). Není však nutné používat klíčový uzel *xd:declaration* nebo jiný speciální způsob zápisu.

2.6.5.2 XML schéma

XML schémata poskytují dva základní způsoby, jak vytvořit deklaraci, v závislosti na typu obsahu:

- **complexType** – Definice uzlu typu element, který může obsahovat další uzly (elementy, partikulární skupiny elementů) nebo atributy.
- **simpleType** – Definice „jednoduchého typu“, který lze pokládat za datový typ (lze použít pouze v rámci atributu a textového uzlu). Součástí definice může být i omezení datového typu.

Výše uvedené deklarace musí ležet v kořeni XML schématu, aby bylo možné na ně odkazovat. Z toho vyplývá, že v obou případech zápisu předpisu pro deklaraci musí být použit atribut *name*, jehož obsah definuje jméno deklarace. Stejně jako u X-definic musí být jména deklarací unikátní, omezení se však vztahuje pouze v rámci jmenného prostoru (včetně prázdného jmenného prostoru).

[13] V rámci různých rozsahů deklarací datových typů mohou být použity stejné názvy – v takovém případě je vždy preferován datový typ s deklarací v lokálním rozsahu.

```
<xs:simpleType name="c_Type">
  <xs:restriction base="xs:long">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="10"/>
  </xs:restriction>
</xs:simpleType>
```

Ukázka 2.28: Příklad předpisu deklarace (XML schéma)

Další vlastností XML schémat, která je analogickou k X-definici, je možnost použít element nebo definici skupiny elementů jako deklaraci (za stejných předpokladů jako u X-definice. Na rozdíl od X-definice, XML schémata nepovolují explicitně definovat kořenové elementy[14], tudíž může být takový předpis do značné míry matoucí. XML schémata oproti X-definici navíc umožňují použít jako deklaraci i kořenový atribut.

Příklad zápisu deklarace typu *simpleType* v XML schématech, který je ekvivalentní k ukázce X-definice z 2.27, je součástí ukázky 2.28.

Obecně vzato lze říct, že deklarace nového datového typu v X-definici lze transformovat na deklaraci *simpleType* v XML schématech a naopak. Transformace deklarace definice skupiny elementů je popsána v subsekci 2.6.1.2. Co se týče transformace deklarovaných uzlů typu *element* a *complexType* v XML schématech, tak zde záleží na kontextu předpisu a referencích na tyto deklarace. Tomuto problému se věnuji podrobněji v subsekci 3.3.2.4 v rámci kapitoly *Návrh*.

2.6.6 Odkazy

Odkazy, neboli **reference**, se obecně používají pro vložení datového typu do elementu nebo atributu. Tento datový typ bývá zpravidla používán na více místech, ale nejedná se o podmínku nutnou. V případě, že chceme vytvořit přehledný předpis datové XML struktury, který není přímo zatížen informacemi o datových typech, můžeme datové typy definovat pomocí deklarace (viz předchozí subsekce 2.6.5). Tento přístup nám zajistí oddělení předpisu struktury dat a samotných datových typů.

Reference se do značné míry liší, a to jak z pohledu zápisu, tak i výskytu v předpisu. Ve výsledku se však jedná o oboustranně bezztrátovou transformaci, kterou je nutné vyhodnocovat v mnoha případech v rámci daného kontextu, nikoliv samotného předpisu (viz dále).

Dalším problémem transformace referencí jsou jmenné prostory, které mají vliv na pozici definice deklarace (v XML schématech) a obsah reference samotné. Jmenné prostory více rozvádím v subsekci 2.6.7. Problémům souvi-

[14] Všechny kořenové uzly typu element jsou zároveň implicitně kořenovými uzly datové XML struktury.

```
<xd:declaration>
  type b_Type string(1, 8);
</xd:declaration>
<a b="optional b_Type()"/>
```

Ukázka 2.29: Příklad atributu s odkazem na deklaraci (X-definice)

sejících s transformací referencí a jmenných prostorů se věnuji zvlášť v subsekci 3.3.2.7 (kapitola *Návrh*).

2.6.6.1 X-definice

V rámci X-definice rozlišujeme následujících pět druhů základních referencí:

- **z elementu *a* na element *r*** – Element *a* má definován atribut *xd:script*, jehož součástí je zápis *ref <NAME_OF_r>*. Pokud element *r* leží v jiné X-definici než element *a*, potom hodnota atributu *xd:script* musí obsahovat *ref <XDEFINITION_NAME>#<NAME_OF_r>*. V obou uvedených případech uvažujeme v rámci jména elementu *r* i jmenný prostor. Element *a* navíc může obsahovat vlastní definici obsahu, tudíž výsledná datová XML struktura by měla obsahovat jak definici elementu *a* tak i definici elementu *r*. Technicky vzato, předpis elementu *r* je vložen do elementu *a*.

- **z elementu *a* na skupinu elementů *g*** – Do atributu *xd:script* elementu *a* je vložena reference (obdobně jako u bodu výše) na skupinu elementů *g*. Další použití této reference je popsáno v rámci subsekce 2.6.1.2.

- **z atributu na deklaraci** – Obsah atributu je definován nepřímo pomocí deklarace. Ukázka 2.29 obsahuje nepovinný atribut *b* v elementu *a*, který odkazuje na deklaraci *b_Type*.

- **z textového uzlu *a* na deklaraci** – Do elementu *a* je vložena reference na deklaraci, syntaxe zápisu odkazu na deklaraci je obdobná jako u příkladu výše uvedeného bodu.

- **z deklarace *e* na deklaraci *f*** – Deklarace datového typu *e* se interně odkazuje na jinou deklaraci *f*.

De facto reference u X-definic lze rozdělit do dvou skupin – reference na element a reference na deklaraci. Dle typu skupiny se následně liší zápis odkazu na referenci.

2.6.6.2 XML schéma

V XML schématech existuje více druhů referencí než u X-definic, které vyplývají z různých kombinací použití v závislosti na tom, jaký typ uzlu se na danou deklaraci odkazuje. V zásadě lze tyto skupiny rozdělit do dvou skupin:

- reference vkládající obsah (použití pomocí atributu *ref*),
- reference přiřazující odkaz na uzel (atribut *type*).

Zatímco první skupina může odkazovat pouze na elementy nebo atributy, resp skupiny elementů nebo atributů, druhá skupina odkazuje pouze na uzly *complexType* a *simpleType* (viz dále). V XML schématu tudíž rozlišujeme následující typy referencí:

- **atribut *a* na atribut *r*** – Předpis obsahu atributu je vložen z předpisu jiného atributu. Atribut *a* obsahuje atribut *ref*, který referuje atribut *r*.
- **atribut *a* na skupinu atributů *g*** – Předpis obsahu skupiny atributů je vložen do předpisu jiného atributu. Atribut *a* obsahuje atribut *ref*, který referuje skupinu atributů *g*.[15]
- **atribut *a* na deklaraci *s*** – Atribut *a* obsahuje atribut *type*, který referuje na deklaraci typu *simpleType* jmenující se *s*.
- **rozšíření/restrikce báze atributu *a*** – Předpis atributu *a* obsahuje ve stromové struktuře uzel typu *xs:extension*, resp *xs:restriction*, který se odkazuje pomocí atributu *base* na základní typ definovaný v XML schématech.
- **element *a* na element *r*** – Předpis obsahu elementu je vložen z předpisu jiného elementu. Element *a* obsahuje atribut *ref*, který referuje element *r*.
- **rozšíření báze elementu *a* o deklaraci *c*** – Předpis elementu *a* je rozšířen o deklaraci typu *complexType* s názvem *c*. Element *a* musí obsahovat ve stromové struktuře uzel typu *xs:extension*, který obsahuje atribut *base* referující uvedenou deklaraci.
- **element *a* na skupinu elementů *g*** – Element *a* obsahuje ve stromové struktuře uzel typu *group*, který obsahuje atribut *ref* referující definici skupiny elementů *g*. Další použití této reference je popsáno v rámci subsekce 2.6.1.2.
- **textový element *a* na deklaraci *s*** – Předpis elementu *a* obsahuje atribut *type*, který referuje na deklaraci typu *simpleType* jmenující se *s*.

[15] V rámci této práce není transformace zmíněného typu reference řešena.

- **rozšíření/restrikce báze textovové elementu *a*** – Předpis elementu *a* obsahuje ve stromové struktuře uzel typu *xs:extension*, resp *xs:restriction*, který se odkazuje pomocí atributu *base* na základní typ definovaný v XML schématech.

- ***complexType* deklarace *c1* na *complexType* deklaraci *c2*** – Deklarace typu *complexType* s názvem *c1* se interně odkazuje na deklaraci typu *complexType* s názvem *c2*.

- ***complexType* deklarace *c1* na *simpleType* deklaraci *s2*** – Deklarace typu *complexType* s názvem *c1* se interně odkazuje na deklaraci typu *simpleType* s názvem *s2*.

- ***simpleType* deklarace *s1* na *simpleType* deklaraci *s2*** – Deklarace typu *simpleType* s názvem *s1* se interně odkazuje na deklaraci typu *simpleType* s názvem *s2*.

Vzhledem ke komplexitě X-definic a mnoha omezujícím pravidel XML schémat, dojde při transformaci X-definice do XML schémat k využití všech typů výše zmíněných referencí – v závislosti na kontextu a podobě předpisu X-definice. Tato skutečnost je demostrací toho, že i přes minimalitu X-definic lze pomocí nich vytvořit předpis mnohem komplexnější datové XML struktury.

K problematice transformace referencí se věnuji v subsekcích 3.3.2.4 až 3.3.2.7 v rámci kapitoly *Návrh*.

2.6.7 Jmenné prostory

Jmenné prostory jsou koncept, který je součástí XML schémat od počátku. Naopak v rámci X-definic byly jmenné prostory doplněny až časem, tudíž lze o nich mluvit jako o doplňku X-definic. I přesto je implementace jmenných prostorů v X-definicích znatelně flexibilnější než v XML schématech.

Jmenný prostor, neboli **namespace**, se definuje pomocí dvojice prefix a URI (viz subsekce 2.7.1 pro X-definice, resp. 2.7.2 pro XML schéma). Prefix musí být v rámci jedné X-definice i XML schématu unikátní. Jmenné prostory určují, jaký jmenný prefix v rámci XML struktury mohou uzly a atributy používat. Samotné předpisy mohou definovat více různých jmenných prostorů.

Ukázka 2.30 obsahuje X-definici používající **jiný jmenný prostor** u elementu *Car*. Definice jmenného prostoru *car* se nachází na druhé řádce ukázky (prefix *car*, hodnota URI *http://example.com/carInfo*). V ukázce 2.31 se nachází bezztrátová transformace do XML schématu a ukázka 2.32 obsahuje validní data ke zmíněným předpisům. Už od pohledu je z ukázky 2.30 vidět, že zápis jmenných prostorů v X-definicích je velmi úsporný a zároveň dobře čitelný.

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/3.2"
   xmlns:car = "http://example.com/carInfo"
   xd:name   = "x-definition-namespace"
   xd:root   = "Cars" >
   <Cars>
       <car:Car>
           ...
       </car:Car>
   </Cars>
</xd:def>
```

Ukázka 2.30: Příklad použití jmenného prostoru (X-definice)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:car="http://example.com/carInfo"
          xmlns:shop="http://example.com/eshopInfo"
          attributeFormDefault="unqualified"
          elementFormDefault="unqualified">
   <xs:import namespace="http://example.com/carInfo"
             schemaLocation="external_car.xsd"/>
   <xs:element name="Cars">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="car:Car"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>

Soubor "external_car.xsd"

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:car="http://example.com/carInfo"
          xmlns:shop="http://example.com/eshopInfo"
          attributeFormDefault="qualified"
          elementFormDefault="qualified"
          targetNamespace="http://example.com/carInfo">
   <xs:import namespace="http://example.com/eshopInfo"
             schemaLocation="external_shop.xsd"/>
   <xs:element name="Car">
      ...
   </xs:element>
</xs:schema>
```

Ukázka 2.31: Příklad použití jmenného prostoru (XML schéma)

```
<Cars
    xmlns:car = "http://example.com/carInfo"
    xmlns:shop = "http://example.com/eshopInfo">
    <car:Car>
       ...
    </car:Car>
</Cars>
```

Ukázka 2.32: Příklad XML dat s použitím jmenného prostoru

Z důvodu použití rozdílného jmenného prostoru v kořeni XML schématu je nutné v rámci XML schémat vytvořit další XML schéma, které bude definovat požadovaný cílový jmenný prostor (více viz 2.6.7.1). Transformace předpisu jedné X-definice se tedy v tomto případě rozpadla na dvě XML schémata. Výsledný předpis navíc používá referenci a je daleko obsáhlejší než původní X-definice. Jedná se sice o bezztrátovou transformaci, nemusí však být nutně od prvního pohledu přímočará. Navíc se vzrůstajícím počtem různých jmenných prostorů může vzrůstat i počet výstupních XML schémat a jejich propojení pomocí referencí. Uvedená XML schémata v ukázce 2.31 lze taktéž bezztrátově nazpět převést do X-definice, nezískáme však původní předpis jako je uveden v ukázce 2.30.[16]

Datová XML struktura 2.32, která je validní vůči oběma předpisům z ukázek, je výsledně elegantní. Při bližším porovnání je navíc viditelné, že se výrazně z pohledu stromové struktury neliší od předpisu X-definice (ukázka 2.30), což může být v mnoha ohledech výhodou.

2.6.7.1 Cílové jmenné prostory

XML schémata umožňují definovat cílový jmenný prostor, který bude výchozí pro všechny uzly v rámci daného XML schéma. Definice cílového jmenného prostoru se může vyskytovat maximálně jednou v XML schéma. Provádí se pomocí atributu *targetNamespace* v kořenovém uzlu (*xs:schema*) XML schéma.

X-definice na rozdíl od XML schémat nedisponují definicí cílového jmenného prostoru. Tato skutečnost však ničemu nevadí, z předchozí ukázky 2.30 je viditelné, že jmenný prostor lze bez problému vložit přímo do předpisu daného uzlu X-definice – absence cílového jmenného prostoru nijak nepřekáží.

[16] Zajisté by šlo vytvořit pokročilejší analýzu předpisu XML schémat, která by umožňovala zjednodušit tento konstrukt v rámci X-definice při transformaci XML schémat.

2.6.8 Kontrola existence a unikátnosti hodnot

2.6.8.1 X-definice

X-definice k nastavení a kontrole unikátnosti používají proměnnou typu *uniqueSet*. Jedná se o jedinou proměnnou, která je v rámci transformace X-definic zpracovávána. *UniqueSet* interně používá tabulku unikátních hodnot. Tato proměnná může být buď součástí deklarace (lokální i globální rozsah) nebo definována jako součást atributu *xd:script* v elementu. V druhém případě je *uniqueSet* dostupný pouze v rámci podstromu daného elementu, navíc každý výskyt elementu validuje svůj podstrom zvlášť.

Typ *uniqueSet* umožňuje definovat jednotlivé interní proměnné. Každá z těchto proměnných má svůj **vlastní datový typ**. Dle zápisu lze určit, zda je daná proměnná součástí klíče *uniqueSetu*. Klíč v *uniqueSetu* se tudíž může skládat i z více proměnných, navíc některé proměnné mohou být nepovinné.

Pro práci s *uniqueSet* existuje v rámci X-definic mnoho funkcí. Zmiňuji zde proto jenom ty, které přímo souvisejí s validací unikátnosti nebo kontroly existence hodnot:

- **ID** – vkládá klíč do tabulky hodnot, vrátí chybu pokud klíč již existuje,
- **SET** – vkládá klíč do tabulky hodnot, nevrátí chybu pokud klíč již existuje,
- **IDREF** – kontroluje, zda klíč exituje v tabulce hodnot, vrátí chybu pokud klíč neexistuje,
- **IDREFS** – kontroluje, zda množina klíčů exituje v tabulce hodnot, vrátí chybu pokud některý z klíčů neexistuje,
- **CHKID** – kontroluje, zda klíč exituje v tabulce hodnot, nevrátí chybu pokud klíč neexistuje,
- **CHKIDS** – kontroluje, zda množina klíčů exituje v tabulce hodnot, nevrátí chybu pokud některý z klíčů neexistuje,

Transformace omezení na unikátnost z X-definic do XML schémat je obtížnou záležitostí, která by si sama o sobě zasloužila důkladnou analýzu (viz dále). XML schémata mají minimální prostředy pro kontrolu unikátnosti hodnot. Jak je vidět, X-definice oproti XML schématům (viz dále) mají daleko více možností, jak validovat unikátnost hodnot a to včetně možného potlačení chybové hlášky. Dalším rozdílem je, že v rámci X-definic je možné, aby jeden element obsahoval více atributů používajících stejnou interní proměnnou v rámci jednoho *uniqueSetu*. Tento zápis je v podstatě nemožné transformovat do XML schémat. XML schémaza též nemají žádnou (ani přibližnou) ekvivalenci pro funkce *IDREFS* a *CHKIDS*.

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/3.2" root='a'
    name="keyAndRef1">
    <xd:declaration>
       uniqueSet u {x: int()}
    </xd:declaration>
    <a>
       <b z='u.x.IDREFS' />
       <c x='u.x.ID()' y='u.x.ID'/>
    </a>
</xd:def>
```

Ukázka 2.33: Příklad použití kontroly unikátnosti (X-definice)

K problematice této transformace se více věnuji v rámci sekce 3.3.1.4, kapitola Návrh. I přesto, že se jedná o velmi často používanou validaci ze strany X-definic, tak vzhledem k omezeným možnostem XML schémat se jedná o okrajový koncept (po diskuzi s vedoucím práce), který v rámci této práce je řešen pouze jako *proof-of-concept*. Navíc bude řešena pouze jednosměrná transformace a to z X-definic do XML schéma[17]. Ukázka 2.33 obsahuje příklad X-definice, která používá kontrolu unikátnosti hodnot.

2.6.8.2 XML schéma

Jak již bylo zmíněno, XML schémata mají minimální, nebo spíš velmi základní, prostředky pro validaci unikátnosti a kontrolu existence hodnot. V rámci XML schémat neexistují žádné funkce určené pro tuto kontrolu, je proto potřeba k těmto účel použít uzly a atributy k tomu určené. V podstatě existují dva základní mechanismy, jak definovat unikátnost hodnot v XML schématech:

1. ***ID/IDREF*** *declarations are done at the level where they are used, and are, therefore, fully integrated with the pseudo-object-oriented features of W3C XML Schema,*

2. ***key/keyref*** *definitions are done at the level of a common ancestor and rely on the actual structure of the instance documents rather than on its object-oriented schema.* [17]

První zmiňovaný mechanismus je velmi triviální – tabulka hodnot je uložena pouze v rámci konkrétního uzlu (nezvtahuje se na podstrom) a nejsou nijak řešeny datové typy. Z toho vyplývá, že kontrola hodnot probíhá pouze a přímo v rámci daného uzlu. Tento mechanismus poskytuje následující tři atributy:

- **ID** – vkládá klíč do tabulky hodnot,

[17] Opačný směr transformace by měl být naprosto bezproblémový.

- **IDREF** – kontroluje, zda klíč existuje v tabulce hodnot,
- **IDREFS** – kontroluje, zda množina klíčů existuje v tabulce hodnot.

Druhý mechanismus je výrazně robustnější než první, který využívá pouze atributů. I přesto ho nelze srovnávat s implementací kontroly unikátnosti a existence hodnot v X-definicích. Druhý mechanismus využívá ke kontrole dotazovací jazyk XPath, který umožňuje provést kontrolu i skrze různé elementy. Přes tento mechanismus lze taktéž libovolně párovat klíče a reference na ně, včetně určení datových typů na základě atributů, které jsou použity v XPath. Toto jsou nesmírné výhody oproti prvnímu zmiňovanému mechanismu. Pro definici kontroly unikátnosti a existence hodnot pomocí XPath se používají následující uzly:

- **xs:unique** – vkládá klíč do své tabulky hodnot,
- **xs:key** – vkládá klíč do své tabulky hodnot,
- **xs:keyref** – referuje na uzel *xs:unique* nebo *xs:key*. Kontroluje, zda klíč existuje v tabulce hodnot daného uzlu.

Všechny výše zmiňované uzly musí mít definované jméno pomocí atributu *name* a obsahovat uzel *xs:selector*, který slouží k zadaní XPath hodnoty vedoucí na požadovaný uzel – hodnota cesty je relativní k pozici daného uzlu **v rámci datové XML struktury** (nikoliv XML schématu). Dále uzly musí obsahovat alespoň jeden uzel *xs:field*, který odkazuje na jednotlivé atributy požadovaného uzlu. Vzhledem ke skutečnosti, že počet uzlů *xs:field* může být více, lze vytvářet klíče pomocí tzv. kompozice, kdy je klíč složen z hodnot více atributů.

Hlavním rozdílem mezi uzly *xs:unique* a *xs:key* je, že existence atributu nebo jeho hodnoty, definovaného pomocí XPath v uzlu *xs:field*, není povinná v případě uzlu *xs:unique*. To znamená, že takový atribut může v datové XML struktuře chybět. Naopak v případě uzlu *xs:key*, pokud se nepodaří dohledat požadovaný atribut v rámci datové XML struktury, dojde k chybě při validaci.

U uzlu *xs:keyref* je potřeba navíc nastavit hodnotu atributu *refer*, která musí obsahovat kvalifikované jméno uzlu *xs:unique* nebo *xs:key*. Tím dojde ke zpárování klíčů a referencí, které následně umožní kontrolu unikátnosti. Ukázka 2.34 obsahuje základní příklad XML schématu, který používá kontrolu unikátnosti a existence hodnot.

Transformace omezení z XML schémat do X-definic nebude součástí implementace této práce, neboť se jedná o okrajový případ a z pohledu vedoucího této práce není nutné se touto transformací v současnosti zabývat. Z těchto důvodů nebudu dále věnovat pozornost transformaci kontroly existence a unikátnosti hodnot z XML schémat.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          attributeFormDefault="unqualified"
          elementFormDefault="unqualified">
   <xs:element name="a">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="b">
               <xs:complexType>
                  <xs:attribute name="m" type="xs:int" use="required"/>
               </xs:complexType>
            </xs:element>
            <xs:element maxOccurs="unbounded" name="c">
               <xs:complexType>
                  <xs:attribute name="o" type="xs:int" use="required"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
      <xs:key name="key_u_0">
         <xs:selector xpath="c"/>
         <xs:field xpath="@o"/>
      </xs:key>
      <xs:keyref name="ref_u_0_0" refer="key_u_0">
         <xs:selector xpath="b"/>
         <xs:field xpath="@m"/>
      </xs:keyref>
   </xs:element>
</xs:schema>
```

Ukázka 2.34: Příklad použití kontroly unikátnosti (XML schéma)

2.7 Kořenové uzly

2.7.1 X-definice

X-definice jako kořenový uzel používá *xd:def*. Tento uzel používá k definici výchozího jmenného prostoru atribut *xmlns:xd*, který v dnešní verzi knihovny X-definice obsahuje hodnotu *http://www.xdef.org/xdef/3.2*. Uzel *xd:def* umožňuje použití dalších atributů níže uvedených.

Jméno X-definice Určeno hodnotou atributu *xd:name*. V rámci kolekce X-definic se musí jednat o unikátní jméno. Tento atribut nemá v XML schématech obraz.

Kořenové elementy Seznam elementů, které mohou být vloženy do kořenu datové XML struktury. Určeno atributem *xd:root*, jednotlivá jména jsou od-

```
<xd:collection xmlns:xd="http://www.xdef.org/xdef/3.2">
    <xd:def xd:name="xdefinition_1" xd:root="a">
        <a>
            <b xd:script = "ref xdefinition_2#c;"></b>
        </a>
    </xd:def>
    <xd:def xd:name="xdefinition_2">
        <c/>
    </xd:def>
</xd:collection>
```

Ukázka 2.35: Příklad kolekce X-definic

dělena separátorem "|". Tento atribut nemá v XML schéma obraz[18], navíc se jedná o nepovinný atribut.

Vložení dalších X-definic Pomocí atributu *xd:include* lze do X-definice načíst další X-definice. Hodnota atributu obsahuje cesty (URI) k takovým souborům, jednotlivé cesty jsou odděleny separátorem ",".

Jmenné prostory V X-definici lze určit jmenné prostory, které mohou být v rámci dané X-definice používány. Syntaxe zápisu jmenného prostoru je následující

xmlns:<PREFIX_PROSTORU>=<URI_PROSTORU>

Všechny jmenné prostory v rámci dané X-definice musí mít unikátní prefix. V důsledku tohoto lze vytvořit neomezené množství unikátních jmenných prostorů[19]. Jmenné prostory lze přímočaře oboustranně bezztrátově transformovat.

2.7.1.1 Kolekce X-definic

Kolekce X-definic je definována elementem *xd:collection*. Obdobně jako kořenový uzel X-definice používá stejný výchozí jmenný prostor, tj. atribut *xmlns:xd* s hodnotou *http://www.xdef.org/xdef/3.2*. Ukázka 2.35 obsahuje jednoduchý příklad kolekce X-definic obsahující dvě X-definice.

[18] Všechny kořenové elementy v XML schématech jsou implicitně i kořeny datové XML struktury.

[19] Neomezené množství je v praxi omezeno prostředky systému. Obecně vzato však neexistuje důvod k vytváření obrovského množství jmenných prostorů.

2.7.2 XML schéma

Kořenový uzel XML schéma se definuje jako *xs:schema* a jeho součástí je taktéž (jako u X-definic) atribut výchozího jmenného prostoru *xmlns:xs*, který má hodnotu *http://www.w3.org/2001/XMLSchema*. Součástí kořenového uzlu mohou být další následující atributy.

Výchozí nastavení jmenného prostoru pro globální atributy a elementy Určuje, zda atributy, resp. elementy, umístěné přímo v kořeni XML schématu mají být implicitně vloženy do cílového jmenného prostoru, který dané XML schéma používá. Nastavení probíhá pomocí atributů kořenového uzlu *attributeFormDefault* pro atributy, resp. *elementFormDefault* pro elementy. Oba atributy mohou nabývat následujících hodnot:

- **qualified** – Kořenový atribut, resp. element patří do cílového jmenného prostoru,
- **unqualified** – Kořenový atribut, resp. element nepatří do cílového jmenného prostoru.

Zde je nutné si uvědomit, že pokud XML schéma nepoužívá cílový jmenný prostor, pak nemá význam používat zmíněné atributy s hodnotou *qualified* [18].

Oba zmíněné atributy kořenového uzlu nemají obraz v X-definicích. Při opačném převodu (z X-definic do XML schémat) bude nutné hodnoty těchto atributů vyhodnotit, viz 3.2.

Výše uvedené výchozí hodnoty použití jmenného prostoru pro kořenové uzly lze lokálně přepsat v rámci jednotlivých elementů a atributů. Lze tak dodatečně pro každý element a atribut zvlášť určit, zda leží nebo neleží v cílovém jmenném prostoru.

Jmenné prostory Jmenné prostory jsou v rámci XML schémat definovány totožně jako u X-definic, viz 2.7.1.

Cílový jmenný prostor Definován atributem *targetNamespace*, jehož hodnota je URI. Tato hodnota musí být z množiny definovaných jmenných prostorů v rámci daného XML schématu.

Tento atribut nemá v X-definici obraz, proto při převodu z X-definice do XML schématu bude nutné hodnotu tohoto atributu vyhodnocovat (obdobně jako u atributů *attributeFormDefault* a *elementFormDefault*), viz 3.2.

2.7.2.1 Kolekce XML schémat

Kolekce XML schémat nepoužívá žádný speciální element jako je tomu u X-definic. Stačí nám pouze znát tzv. kořenové XML schéma, ze kterého se sestavuje validační předpis. Kořenové schéma je určené jednoznačně a musí být právě jedno.

Abychom z jednoho XML schéma vytvořili kolekci schémat, musíme do kořenových uzlů (*xs:schema*) dílčích XML schémat přidat uzly, které vkládají do současného XML schéma další XML schéma (viz dále). Toto vkládání schémat je typicky lineárního nebo případně stromového charakteru. Může však existovat i cyklická závislost, kdy XML schémata tvoří v důsledku jednotlivých vložení kružnici předpisů.

Vložení XML schéma se stejným cílovým jmenným prostorem Pokud chceme do XML schéma vložit další XML schéma se stejným cílovým jmenným prostorem (pokud obě schéma nepoužívají cílový jmenný prostor, tak se toto pravidlo též aplikuje), potom použijeme uzel *xs:include*, který vložíme do kořenového uzlu našeho XML schématu. Jedinou hodnotou zmíněného uzlu je atribut *schemaLocation*, jehož obsahem je cesta (URI) k danému XML schématu.

Vložení XML schéma se odlišným cílovým jmenným prostorem Pokud chceme do XML schéma vložit další XML schéma s odlišným cílovým jmenným prostorem (pokud naše schéma nepoužívá cílový jmenný prostor a vkládané schéma používá, tak se toto pravidlo též aplikuje), potom použijeme uzel *xs:import*. Na rozdíl od předchozího případu, musíme navíc vyplnit atribut *namespace*, který nese hodnotu URI cílového jmenného prostoru vkládaného XML schématu.

Kapitola 3

Návrh

3.1 Nové řešení – Big picture

Nová implementace algoritmu pro oboustrannou transformaci by měla zohledňovat poznatky a chyby, ke kterým došlo v průběhu realizace původního řešení nebo byly časem objeveny při používání (zmíněné v sekci 2.3). Další snahou bude implementaci takovou, která bude celkově nejen dostatečně strukturovaná, ale bude i dobře udržitelná a rozšiřitelná do budoucna, čili flexibilní. Hlavními body (rozumějme zlepšeními), kterými by se návrh a implementace nového řešení měly řídit, jsou:

- **Načtení vstupních přepisů** – V případě X-definic bude vstupní předpis do algoritmu ve formátu zkompilované instance X-definice, resp. zkompilované kolekce X-definic. Pro načítání XML schémat bude využita knihovna třetí strany *Apache XmlSchema*[20]. Vstupem do algoritmu bude taktéž zkompilovaná instance, akorát v tomto případě se bude jednat o XML schéma nebo kolekci XML schémat. Původní řešení v obou případech pracovalo přímo se zápisem daných předpisů (čili obsahovalo vlastní syntaktický analyzátor), nikoliv s již načtenými a zkompilovanými objekty v paměti.
- **Fázování transformace** – Algoritmus by neměl být plochý jako v původním řešení. Minimálně při převodu X-definic do XML schémat bude rozdělen do několika fází:
  1. *Inicializace* – Inicializace kontextu převodu, vytažení základních informací z X-definice/kolekce X-definic, resp. XML schéma/kolekce XML schémat.
  2. *Preprocessing* – Průchod stromem předpisu a načtení důležitých dat pro fáze *transformace* a *postprocessing*. Možná částečná transformace uzlů, která však nesmí být duplicitní s fází *transformace*.

[20] `https://ws.apache.org/xmlschema/`

3. *Transformace* – Průchod stromem předpisu a transformace jednotlivých uzlů do cílového typu předpisu.
4. *Postprocessing* – Dodatečná transformace uzlů, které byly uloženy nebo označeny ve fázích *preprocessing* nebo *transformace*. Tyto uzly buď nebylo možné v daný moment převést nebo vyžadují pokročilou transformaci s ohledem na strukturu předpisu. Dále v této fázi mohou být provedeny doplňující transformace nad již existujícím cílovým předpisem.

- **Flexibilita a modularita** – Implementace by měla zohledňovat logickou strukturu algoritmu se snahou dodržet pravidlo *Maximize cohesion & Minimize coupling*, aby bylo možné algoritmus do budoucna jednoduše rozšiřovat [19].
- **Pokrytí prostoru předpisů** – Implementace by měla maximalizovat pokrytí předpisů, které dokáže transformovat. V případě X-definic nebude možné mnoho zápisů (zejména pro pokročilou validaci) převést, tudíž bude tyto transformace v některých případech **ztrátové** (v závislosti na předpisu). Pokrytí předpisů bude testováno podmnožinou příkladů, které již v knihovně X-definice existují (více viz kapitola Testování 5). V opačném směru transformace (XML schéma do X-definice) by převod měl být bezztrátový.

3.1.1 Knihovna Apache XmlSchema

Součástí diplomové práce je i zpracování a vytváření souborů ve formátu XML schéma. Abych tyto operace nemusel řešit vlastní implementací (tyto operace nejsou cílem této práce ani takové řešení není žádoucí – viz požadavky na nové řešení 3.1), zvolil jsem po dohodě s vedoucím práce řešení pomocí open-source knihovnou třetí strany, která je dlouhodobě rozvíjena, udržována a má důvěryhodné autory – organizaci *Apache*[21]. Samozřejmostí byl požadavek na licenci knihovny, která umožní použití v rámci knihovny X-definice.

Knihovna umožňuje pracovat s XML schématy na úrovni objektového modelu v jazyce Java. Lze pomocí ní jak schéma načíst, tak i upravit nebo vytvořit zcela nové. Knihovna nevyžaduje novější standardy jazyka Java, konkrétně vyžaduje verzi 1.5 a vyšší, tudíž je možné ji použít v současné implementaci knihovny X-definice, která používá standard Javy 1.6. Ke knihovně zároveň existuje i dokumentace a ukázky základních příkladu použití.

3.2 Transformace kořenových uzlů

Převod kořenových uzlů mezi formáty obnáší jistou transformaci hodnot a vyhodnocování dodatečných atributů, které nemusejí být explicitně definovány

[21] `http://www.apache.org/`

v původním předpisu. Jedinou implicitní transformací atributů kořenových uzlů jsou jmenné prostory, viz 2.7.

3.2.1 Vytvoření kořenového uzlu XML schématu

Kořenový uzel XML schématu je *xs:schema*, jak již bylo zmíněno v kapitole *Analýza* (více viz 2.7.2). V transformaci z X-definic do XML schémat je potřeba rozhodnout o hodnotách následujících atributů:

1. *targetNamespace* – 3.2.1.1,

2. *attributeFormDefault* a *elementFormDefault* – 3.2.1.2.

3.2.1.1 Cílový jmenný prostor

Určení cílového jmenného prostoru XML schéma vychází ze dvou následujících kroků:

1. analýza jmenných prostorů použitých v kořenových uzlech transformované X-definice,

2. pokud se v prvním bodě nepodaří získat konkrétní jmenný prostor, pak se kontroluje existence jmenného prostoru s prázdným prefixem.

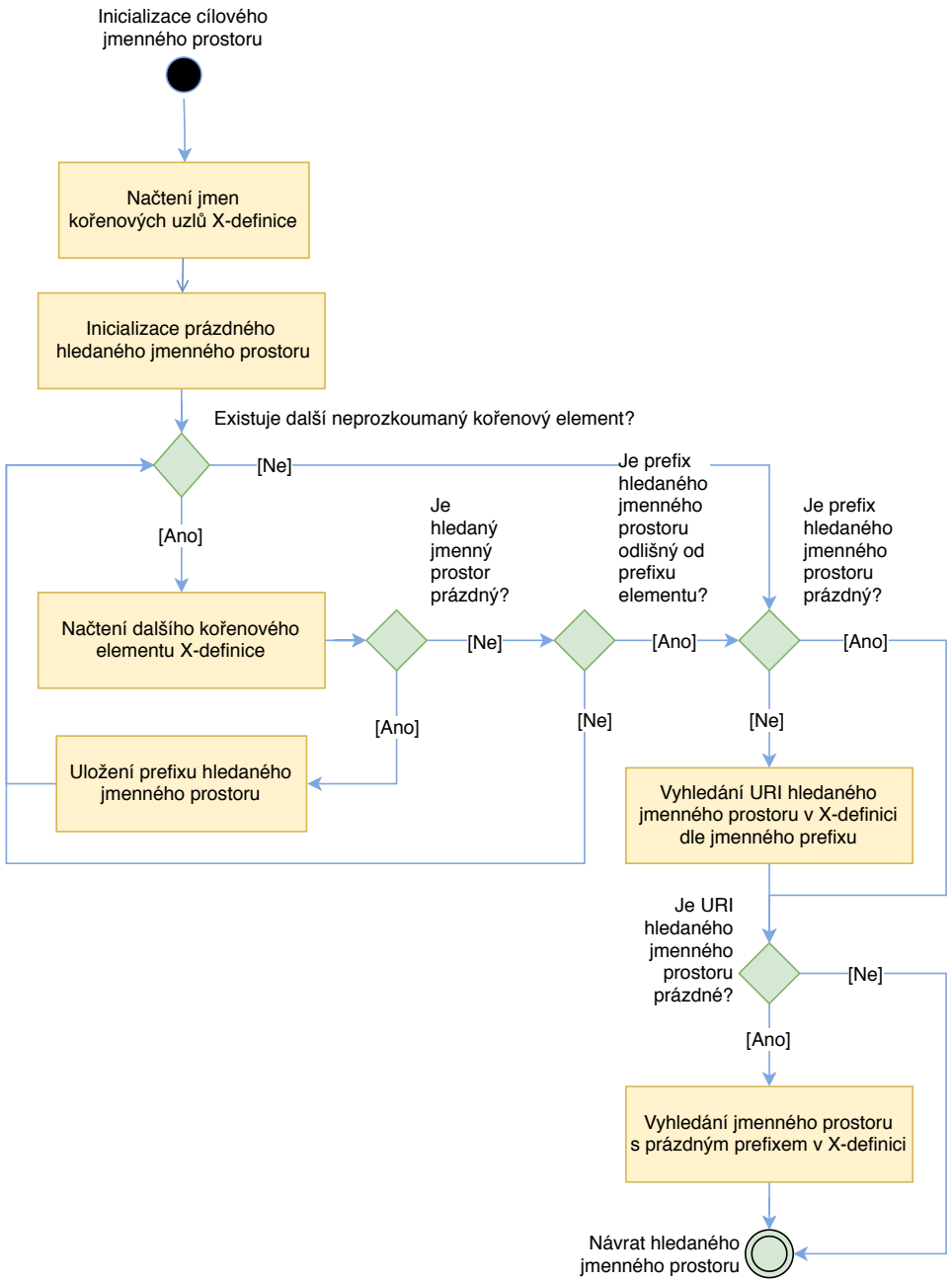
V první fázi je nutné zjistit, zda všechny kořenové elementy X-definice (jejich jména jsou definována v rámci atributu *xd:root* v kořenovém elementu *xd:def*) používají stejný jmenný prefix. Následně je vyhledáno URI jmenného prostoru v X-definici dle dohledaného jmenného prefixu. Pokud v X-definici neexistuje jmenný prostor používající daný prefix, potom dojde k vyhlášení chyby a použití prefixu jmenného prostoru bez URI. Pokud existuje alespoň jeden kořenový element používající jiný jmenný prefix, potom dojde k vyhlášení chyby a použití prvního nalezeného prefixu jmenného prostoru.

K druhé fázi dojde teprve tehdy, když se nepodaří získat žádný prefix jmenného prostoru (a tím pádem ani jeho URI). V takovém případě by algoritmus měl zkontrolovat, zdali náhodou není používán v X-definici jmenný prostor s prázdným prefixem, tj. zápis

xmlns=<URI_PROSTORU>.

Za předpokladu, že by takový jmenný prostor v X-definici existoval, potom by cílovým jmenným prostorem v XML schématu byl prostor s prázdným prefixem a stejným URI jako v X-definici, tj. hodnota <URI_PROSTORU>.

Celý diagram algoritmu určení cílového jmenného prostoru je zobrazen na obrázku 3.1.



Obrázek 3.1: Diagram algoritmu pro určení cílového jmenného prostoru XML schémat

3.2.1.2 Kvalifikovaná jména kořenových atributů a elementů

Atributy *attributeFormDefault* a *elementFormDefault* definují, zda kořenové neboli globální atributy a elementy mají implicitně patřit do cílového jmenného prostoru XML schéma nebo ne (více v kapitole *Analýza*, subsekce *Kořenové uzly – XML schéma* 2.7.2).

Obdobně jako při vyhodnocování cílového jmenného prostoru XML schématu je potřeba analyzovat kořenové elementy X-definice. Dalším vstupním parametrem je cílový jmenný prostor, který jsme zjistili již dříve v rámci transformace kořenového uzlu X-definice.

attributeFormDefault Pro zjištění hodnoty tohoto atributu je potřeba analyzovat všechny atributy všech kořenových elementů v X-definici. Přímo v atributech X-definice zjistíme, v jakém jmenném prostoru leží. Pokud některý z prohledávaných atributů leží ve jmenném prostoru ekvivalentním (z pohledu hodnoty prefixu jmenného prostoru) k cílovému jmennému prostoru, potom hodnota *attributeFormDefault* je *qualified*. V opačném případě je hodnota *unqualified*.

elementFormDefault Obdobně jako u předchozího atributu *attributeFormDefault* je nutné analyzovat všechny kořenové elementy X-definice. Pokud však XML schéma má definovaný neprázdný cílový jmenný prostor, potom je hodnota atributu *elementFormDefault* implicitně *qualified*. V případě prázdného cílového jmenného prostoru je potřeba získat jmenný prostor všech kořenových uzlů a ten následně porovnat s cílovým jmenným prostorem XML schématu. Pokud se hodnoty (prefixy) shodují, potom je hodnota atributu *qualified*. V opačném případě je hodnota *unqualified*.

3.2.2 Vytvoření kořenového uzlu X-definice

Na rozdíl od XML schéma může být kořenový uzel X-definice tvořen dvěma různými uzly na základě toho, zdali se jedná o kolekci X-definic nebo právě jeden předpis X-definice (i v tomto případě bychom mohli předpis X-definice vložit do kolekce, není to však žádoucí).

V případě kolekce X-definic je nutné použít jako kořenový uzel *xd:collection*. Tento uzel použijeme právě tehdy, když transformujeme kolekci předpisů XML schémat do X-definic. Uzel *xd:collection* navíc nevyžaduje žádné nastavení na základě XML schémat.

Kořenový uzel X-definice *xd:def* je naopak závislý na obsahu kořenového uzlu XML schéma. Ať už transformujeme jedno XML schéma nebo celou kolekci, vždy bude nutné pro každé XML schéma vytvořit právě jeden uzel *xd:def*. Při transformaci z XML schéma je nutné vyhodnotit obsah následujících atributů kořenového uzlu X-definice:

```
xd:root = "empty | text | attr"
```

Ukázka 3.1: Příklad zápisu atributu *xd:root* v X-definicích

- **xd:name** – jedná se o jméno X-definice, které lze zadat jako externí hodnotu nebo vyhodnotit ze jména souboru XML schématu,
- **xd:root** – z jmen všech kořenových elementů XML schéma vytvoříme řetězec, který bude jednotlivá jména elementů oddělovat pomocí znaku "|". Názorný příklad výsledku takové transformace pro XML schéma se třemi kořenovými elementy, pojmenovanými *empty*, *text* a *attr*, je součástí ukázky 3.1.

3.3 Pokročilejší transformace X-definice do XML schéma

Sekce pokročilejších transformací se věnuje problematice převodu uzlů z X-definice do XML schéma takových, které nelze přímočaře (partikulárně) převést, jak bylo popsáno v sekci 2.6. Tyto transformace je nutné většinou provést z důvodu porušení pravidel XML schémat po přímočaré transformaci uzlů.

3.3.1 Ztrátové transformace

Tato subsekce popisuje známé ztrátové transformace z předpisu X-definice do XML schéma, na které jsem narazil v průběhu vypracování této práce. Nejedná se tak nutně o kompletní množinu ztrátových transformací.

3.3.1.1 Mixed element

Transformace elementů podporujících kombinovaný obsah, tj. textové uzly a uzly elementů, je teoreticky oboustranně ztrátová. Hlavním problémem je, že XML schémata nepodporují validaci obsahu textových uzlů v rámci kombinovaného obsahu. Z toho vyplývá, že obsah textových uzlů nelze ani nijak omezit.

X-definice obsahují více různých prostředků (viz 2.6.1.1), jak zmíněnou kombinaci uzlů zadefinovat. Každý prostředek navíc s sebou přináší jiné možnosti validace. Avšak definice *mixed* elementu v rámci XML schémat nemá plnohodnotný ekvivalent k žádnému z prostředků X-definic.

Nezávisle na tom, který prostředek pro definici kombinovaného obsahu v rámci X-definic použijeme, musíme na straně XML schéma v rámci příslušného elementu, resp. uzlu *xs:complexType* uvnitř daného elementu, použít atribut *mixed* s hodnotou *true*.

O jediné bezztrátové transformaci *mixed* elementu lze uvažovat pouze v případech, kdy kombinovaný obsah v rámci X-definice je definován pomocí následujících atributů a jejich hodnot:

- xd:text="? string()"
- xd:textcontent="? string()"

Všechny ostatní případy zápisu kombinovaního obsahu jsou zcela určitě ztrátové. V obecném měřítku je nejvíce ztrátovým případem situace, kdy v X-definicích je definováno, na jakém místě a s jakým obsahem se má textový uzel vyskytovat.

3.3.1.2 Neuspořádaná množina elementů

V rámci dílčích transformací se jedná o převod uzlu *xd:mixed* na *xs:all*. Analýzou jsem dohledal čtyři hlavní problémy při transformaci z X-definic do XML schéma[22]:

- Uzel *xs:all* v XML schéma nemůže obsahovat jiný uzel než typu element, zatímco v X-definici takové omezení neexistuje. Pokud tedy takový uzel v X-definici obsahuje jiný uzel než je element, potom tato skutečnost znamená, že bude nutné provést transformaci z elementu *xd:mixed* na uzel *xs:choice* s příslušnou kardinalitou. Kardinalitu *xs:choice* je možné jednoduše dopočítat pomocí zanoření se do původního *xd:mixed*. Vzhledem k vysoké ztrátovosti takové transformace je možné využít i kardinalitu *unbounded*.
- Uzel *xs:all* nemůže obsahovat elementy mající nastavenou kardinalitu vyšší než jedna. Tento případ je v podstatě obdobným problémem jako výše uvedený bod (opět transformace uzlu *xd:mixed* na *xs:choice*).
- Uzel *xs:all* může mít maximální kardinalitu rovno jedné. Řešení tohoto problému je analogické jako výše jmenované.
- Uzel *xs:all* nemůže být vložený, ať už přímo nebo nepřímo (vztahuje se bohužel i na reference nebo definici skupiny elementů), v libovolné hloubce do uzlu *xs:sequence* nebo *xs:choice*. Řešení této situace je o něco složitější a bude vyžadovat pokročilejší transformaci na bázi postprocessingu – viz kapitola *Návrh* 3. V podstatě je nutné zajistit, když bude transformována takováto struktura předpisu X-definice, aby došlo opět k transformaci uzlu *xd:mixed* na *xs:choice*.

Výše uvedená řešení problémů obsahují ve všech případech transformace z X-definice do XML schéma přechod na uzel *xs:choice*. Je to z toho důvodu, že

[22] `https://www.w3.org/TR/xmlschema-1/#element-all`

```
<xd:choice>
  <xd:mixed>
    <p xd:script='2..3;'/>
    <q xd:script='0..1;'/>
  </xd:mixed>
  <b/>
  <c/>
</xd:choice>
```

Ukázka 3.2: Příklad ztrátové transformace uzlu *xd:mixed* (X-definice)

```
<xs:choice>
  <xs:choice maxOccurs="4" minOccurs="2">
     <xs:element name="p"/>
     <xs:element name="q"/>
  </xs:choice>
  <xs:element name="b"/>
  <xs:element name="c"/>
</xs:choice>
```

Ukázka 3.3: Příklad ztrátové transformace uzlu *xd:mixed* (XML schéma)

z pohledu XML schémat musí být zachována partikulární skupina elementů a jedinou další nabízející se možností je uzel *xs:sequence* (uspořádaná množina), u kterého jak jistě název napoví, nelze aplikovat na uzel typu neuspořádaná množina. Tudíž se v řešení těchto ztrátových transformacích nenabízí žádná jiná možnost než použít uzel *xs:choice*.

V X-definici je navíc možné vnořit do sebe uzly typu *xd:mixed*. Toto vnoření má specifické chování, jako je například zachování pořadí uzlů uvnitř *xd:mixed* v rámci celého předpisu. I v tomto případě nelze dosáhnout bezztrátové transformace do XML schéma. Jako nejlepší možné řešení se jeví spojení všech vnořených uzlů *xd:mixed*, resp. *xs:all* do jednoho uzlu daného typu.

Pokud nenastane při transformaci X-definice do XML schéma jedna z výše uvedených situací, pak je možné uzel *xd:mixed* převést bezztrátově. Avšak vzhledem ke komplexitě používaných X-definic v reálném provozu je taková šance minimální. Z této skutečnosti vyplývá, že **při transformaci komplexnější X-definice obsahující uzel *xd:mixed* existuje velmi vysoká šance, že dojde ke ztrátové transformaci**.

Ukázky 3.2 a 3.3 obsahují příklady ztrátové transformace uzlu *xd:mixed*, ke které došlo z následujících důvodů:

- stromová struktura XML schéma neumožňuje vložení uzlu *xs:all* do *xs:choice*,

- uzel *xs:all* nemůže obsahovat uzly s kardinalitou vyšší než 1.

V ukázkách je dále vidět, že vnitřní uzel *xs:choice* **nově obsahuje atributy výskytu na základě původního předpisu** X-definice, které pomáhají omezit prostor připuštěných validních datových XML struktur. Ztratila se však informace o tom, který element může být kolikrát použit.

3.3.1.3 Definice skupiny elementů

Ztrátovost této transformace plyne z 3.3.1.2, neboť definice skupiny elementů v rámci X-definic je definována jako neuspořádaná množina elementů. Pokud tedy dojde ke ztrátové transformaci elementu *xd:mixed* jakožto kořenového uzlu definující skupinu elementů, potom lze očekávat, že na straně XML schémat bude uzel *xs:group* obsahovat *xs:choice* namísto *xs:all*.

3.3.1.4 Kontrola existence a unikátnosti hodnot

Jak již bylo v rámci analýzy zmíněno (viz 2.6.8), omezení na existenci a unikátnosti hodnot je z pohledu implementace velmi odlišné pro každý předpis a navíc je řešení této problematiky v rámci X-definic daleko rozsáhlejší a robustnější. Z těchto důvodů je potřeba vytvořit omezující pravidla, kdy má smysl se snažit o transformaci zmiňované kontroly a kdy již nemá.

V první řadě je potřeba určit mapování metod X-definice, určených ke kontrole existence a unikátnosti hodnot, na uzly v XML schématu.

- Metoda **ID** se bude mapovat na uzel *xs:unique* nebo *xs:key* (viz dále).
- Metoda **SET** se bude mapovat na uzel *xs:unique* nebo *xs:key* (viz dále).
- Metoda **IDREF** se bude mapovat na uzel *xs:keyref*.
- Metoda **CHKID** se bude mapovat na uzel *xs:keyref*.
- Metody **IDREFS**, **CHKIDS** a další v této práci nezmíněné se nebudou mapovat – čili jsou v rámci transformace nepodporovány.

Z výše uvedeného mapování je viditelné, že u XML schémat nezvažujeme použití atributů **ID**, **IDREF**, **IDREFS**, které jsou zmíněny v subsekci 2.6.8.2, a to z toho důvodu, že tyto atributy umožňují pokrytí naprosto minimálního počtu případů[23].

Následně je potřeba určit hranice, kdy ještě jsme schopni povolené metody X-definice plnohodnotně (bezztrátově) transformovat do XML schémat. Níže se nachází výčet pravidel určujících omezení pro požadovanou transformaci. Tato pravidla se opírají o analýzu a poznatky získané v rámci této práce. Vzhledem k tomu, že se jedná o *proof-of-concept* (viz 2.6.8.1), nemusí být nutně tato množina pravidel finální pro budoucí rozvoj.

[23] Většinu těchto případů lze navíc pokrýt pomocí uzlů *xs:unique*, *xs:key* a *xs:keyref*.

- U převodu kontroly existence a unikátnosti hodnot nemá smysl uvažovat o takových transformacích, které jsou z pohledu unikátnosti ztrátové – pokud nejsme schopni zabezpečit unikátnost hodnoty, potom nejsme schopni ohlídat ani její existenci.

- Pokud existuje více klíčů (metody **ID** a **SET**), které leží v různých elementech, potom nelze transformaci provést.

- Pokud existuje více klíčů (metody **ID** a **SET**), které mají více výskytů v rámci jednoho elementu, potom nelze transformaci provést[24].

Pokud bychom shrnuli pravidla týkající se metod **ID** a **SET**, tak nám v podstatě říkají, že musí existovat v rámci X-definice právě jeden atribut definující klíč. Za předpokladu, že výše uvedená pravidla jsou splněna, je už potřeba rozhodnout pouze o tom, zda se metody **ID** a **SET** mají transformovat na uzel *xs:unique* nebo *xs:key* v XML schéma. Toto rozhodnutí závisí pouze a jen na tom, zda existuje metoda **IDREF** nebo **CHKID** (může jich být i více) odkazující se na příslušný klíč. Pokud existuje alespoň jedna taková metoda, potom se metody **ID** a **SET** transformují na uzel *xs:key*. V opačném případě se metody transformují na uzel *xs:unique* XML schéma.

Výše uvedené rozhodnutí o transformaci metod **ID** a **SET** je z důvodu čitelnosti, aby bylo jednoduše rozeznatelné (v XML schéma), zda se jedná o unikátní klíč s referencí nebo pouze unikátní hodnoty.

V neposlední řadě je potřeba vyřešit XPath cesty k atributům, ke kterým se klíče a reference na klíče vztahují. Tento problém je naštěstí možné řešit velmi jednoduše a elegantně, a to jen díky tomu, že

- zkompilovaná X-definice interně používá mechanismus obdobný XPath pro označení pozice uzlů a atributů,

- stromová struktura X-definic odpovídá stromové struktuře datového XML, které je k ní validní.

Na základě techto skutečností je možné velmi jednoduše získat potřebnou relativní XPath cestu k uzlům a atributům.

Vzhledem k počtu a charakteristice omezujících pravidel lze očekávat, že většina proměnných typu *uniqueSet* v rámci X-definic nebude v současnosti transformována do XML schémat (budou ignorovány) a to z důvodu komplexity jejich použití.

[24] Tento problém je pravděpodobně do určité řešitelný. Vyžaduje však další analýzu tzv. kompozitních klíčů v rámci X-definic a XML schémat.

3.3.2 Bezztrátové transformace

3.3.2.1 X-definice a kolekce XML schémat

Zatímco u X-definic je možné v rámci jednoho souboru definovat kolekci X-definic, kde každá z nich má své unikátní jméno, tak u kolekce XML schémat je nutné pro každé XML schéma zvlášť vytvořit právě jeden soubor. Teoreticky lze uvažovat o spojení předpisů XML schémat, které používají stejný cílový jmenný prostor – tato studie však není součástí ani cílem této práce. Proto dále uvažuji pouze model takový, kdy z každé X-definice bude vytvořeno minimálně jedno XML schéma (viz dále).

Pro vytvoření XML schéma je nutné znát název souboru. Tento název je buď možné generovat automaticky, nebo ještě lepší případ je přebrání názvu z předpisu X-definice. Vzhledem k tomu, že názvy X-definic jsou unikátní, potom i názvy souborů kolekce XML schémat musí být unikátní.

Tato situace platí pouze do chvíle, kdy vytváříme právě jedno XML schéma pro jeden předpis X-definice, viz další subsekce.

3.3.2.2 XML schéma a jmenné prostory

Vytvoření více XML schémat z jedné X-definice bylo již zmíněno v subsekci 2.6.7 v rámci kapitoly *Analýza*. Z uvedené analýzy jmenných prostorů vyplývá, že v případě použití elementů nebo atributů ležících v jiném jmenném prostoru (používající jiný jmenný prefix než kořenové elementy nebo atributy), je potřeba vytvořit takové XML schéma, které bude mít cílový jmenný prostor shodný s jmenným prostorem takových elementů a atributů. Do takto vytvořeného XML schéma budou přemístěny všechny elementy a atributy ležící v jiném jmenném prostoru a následně na ně bude vytvořena příslušná reference z původních míst v předpisu XML schémat.

Tato skutečnost má více dopadů na algoritmus transformace X-definice do XML schéma:

- Je nutné vytvořit takový algoritmus pojmenování XML schémat, aby nedošlo ke kolizi v případě, že vytváříme více XML schémat než bylo vstupních X-definic.
- Je nutné zajistit, aby v případě existence dvou a více stejně pojmenovaných elementů, resp. atributů, které mají být vloženy jako kořenové uzly do nového XML schéma (s příslušným cílovým jmenným prostorem), nedošlo ke kolizi jmen.

Z důvodu zjednodušení bude pojmenování vytvářených dodatečných XML schémat postaveno pouze na prefixu cílového jmenného prostoru. Jméno souboru XML schéma bude vytvářeno následujícím předpisem

”external_”+ <PREFIX_JMENNÉHO_PROSTORU>.

Řešení transformace vytváření dodatečných XML schémat, které budou obsahovat pouze uzly ležící v jiném jmenném prostoru v rámci X-definice, bude řešeno až ve fázi *Postprocessing* algoritmu transformace. Toto rozhodnutí plyne ze skutečnosti, že nejprve je nutné zjistit, které všechny uzly bude nutné přemístit do jiných XML schémat. Tato analýza musí být provedena až po rozhodnutí cílového jmenného prostoru příslušného XML schéma, které má být vytvořeno z dané X-definice. To znamená, že nejprve je nutné převést samotnou X-definici do XML schéma a až následně dořešit takto dotčené uzly. Analýza kterých uzlů se přesun týka, může být provedena přímo v rámci transformace stromové struktury X-definice.

3.3.2.3 Tvorba deklarací

Proces tvorby deklarací lze rozdělit do třech fází:

1. transformace deklarací X-definice (atributy a textové uzly),
2. transformace globálních elementů, které nejsou zároveň kořeny X-definice (jejich jména nejsou součástí atributu *xd:root*),
3. transformace vložení ostatních X-definic – vytvoření uzlů *xs:include* a *xs:import* na základě použitých referencí.

Transformace deklarací Výsledkem jsou uzly *simpleType* XML schématu vytvořeny na základě deklarací předpisu X-definice. Transformovány jsou pouze takové deklarace, které jsou v rámci dané X-definice použity (pomocí reference). Zároveň jsou z transformace vynechány všechny uzly ležící v jiném jmenném prostoru, které mají referenci na danou deklaraci. Takové uzly jsou poznamenány pro fázi algoritmu *Postprocessing*.

Pokud neexistuje žádné omezení na datový typ deklarace, potom uvnitř uzlu *simpleType* bude uzel *xs:extension*. V opačném případě bude součástí deklarace uzel *xs:restriction* se všemi náležitostmi a omezeními.

Transformace globálních elementů Výstupem této transformace jsou buď globální elementy XML schématu nebo uzly typu *complexType*. Pro transformaci je použit stejný algoritmus jako pro převod celé stromové struktury X-definice.

3.3.2.4 Elementy – Tvorba referencí

Z analýzy deklarací a referencí (viz 2.6.5 a 2.6.6) vyplývá, že i přes menší počet typů deklarací a referencí v X-definicích dochází pokrytí všech případů v XML schématech. To znamená, že musí docházet k logickému rozpadu jednotlivých případů v rámci X-definice na menší dílčí části.

Ve všech případech níže popsaných se snažíme získat kvalifikované jméno pro definici reference (zdrojový kód pro vytvoření kvalifikovaného jména reference je součástí přiložené ukázky D.2). Navíc platí, že elementy používající referenci nemají žádný svůj obsah (nedefinují žádný vnořený uzel ani uzel atributu). Až na poslední případ vkládáme v rámci XML schéma získané jméno do atributu *ref* příslušného elementu (obsahující referenci). V posledním zmiňovaném případě je použit atribut *type*, který musí odkazovat na uzel *simpleType* nebo *complexType*, zatímco atribut *ref* u elementu vždy odkazuje na další uzel typu element.

Element v jiném jmenném prostoru mající referenci Element obsahuje již informaci o URI jmenného prostoru. Výsledné kvalifikované jméno se skládá z uvedeného URI a jména elementu.

Reference na element v jiné X-definici v jiném jmenném prostoru Element obsahuje obdobu XPath pozice referovaného elementu v jiné X-definici. Z těchto informací můžeme zjistit, jaký jmenný prostor je používán referovaným elementem – konkrétně prefix jmenného prostoru. Na základně prefixu jmenného prostoru můžeme v cílové X-definici dohledat URI jmenného prostoru. Uvedené URI a jméno elementu (bez názvu X-definice a prefixu jmenného prostoru) jsou hledaným kvalifikovaným jménem reference.

Reference na element v jiné X-definici Obdobně jako v případě výše, element obsahuje obdobu XPath pozice referovaného elementu v jiné X-definici. Nepotřebujeme však znát URI, neboť element leží ve stejném jmenném prostoru. Prázdné URI a jméno elementu ve stejném formátu, jak je popsáno výše, jsou hledaným kvalifikovaným jménem reference.

Reference na element ve stejné X-definici a stejném jmenném prostoru Nejjednodušší případ ze všech, hledané URI je prázdné a jméno reference je opět bez názvu X-definice.

Dalším jednotlivým případem je, když element definuje svůj vlastní obsah a zároveň používá referenci. V takovém případě je nutné do tohoto elementu v rámci XML schémat vložit uzel *xs:complexContent*, který se pomocí atributu *base* odkazuje na referenci, jejíž kvalifikované jméno bylo získáno za podobných pravidel uvedených výše. **Navíc je nutné deklaraci reference převést z elementu na uzel *xs:complexType*.** Na ukázce 3.4 je vidět rozšíření obecného elementu, který jako bázi používá uzel *ct_root_Person* ve jmenném prostoru s prefixem *ext*. Dále tento element sám o sobě definuje vnitřní uzel *Hat* a atribut *workedId*.

V rámci práce s referencemi je dále nutné, aby XML schéma obsahující elementy, které odkazují na uzly v jiných XML schématech, obsahovalo taktéž

```
<xs:complexType>
  <xs:complexContent>
     <xs:extension base="ext:ct_root_Person">
        <xs:sequence>
           <xs:element name="Hat" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="workerId" type="xs:int" use="required"/>
     </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Ukázka 3.4: Příklad rozšíření báze elementu v XML schéma

uzly vkládající příslušná XML schémata do současného. Popis podoby a způsoby vložení jsou popsány v subsekci 2.7.2.1 v rámci kapitoly *Analýza*.

3.3.2.5 Atributy – Tvorba referencí

Atributy na rozdíl od elementů v X-definicích neumožňují použití referencí. To znamená v porovnání s elementy velké zjednodušení z pohledu rozpadu deklarací a referencí. Jediný případ, při kterém může dojít k vytvoření reference v rámci XML schéma, je totožný jak pro atributy tak i elementy a je popsán v následující subsekci 3.3.2.7.

Místo referencí se však uzly atributů v rámci XML schéma mohou potýkat s problémy použití atributu *type*, který obsahuje obdobně jako reference u elementů kvalifikované jméno. Mohou nastat následující případy:

1. atribut se odkazuje na uzel typu *simpleType*, který byl vytvořen v rámci procesu popsaného v 3.3.2.3,
2. atribut se odkazuje na základní datový typ XML schémat.

3.3.2.6 Textové uzly – Tvorba referencí

Jedná se o obdobný případ jako u atributů v XML schématech, neboť textové uzly též využívají uzly typu *simpleType*, akorát prostřednictvím uzlu *xs:extension* nebo *xs:restriction* uvnitř uzlu *xs:simpleContent*. Uzel *xs:extension* by měl být použit pouze v případech, kdy textový uzel nepoužívá atributy.

V případě jak atributů, tak i textových uzlů, by mělo vždy dojít k použití reference na uzel *simpleType*, který obsahuje *xs:restriction*, pokud existují další omezení na daný datový typ.

3.3.2.7 Elementy a atributy používající jiný jmenný prostor

Posledním větším případem, který je vhodný k většímu rozboru a zároveň již byl částečně zmíněn v předešlé subsekci 3.3.2.2, je situace, kdy element,

resp. atribut sice nepoužívá žádnou referenci, ale samotné jméno tohoto uzlu leží v jiném jmenném prostoru. V takovém předpisu je nutné zjistit URI jmenného prostoru z aktuální X-definice na základě prefixu jmenného prostoru použitého ve jménu elementu, resp. atributu. Dohledané URI a jméno elementu, resp. atributu (bez názvu X-definice a prefixu jmenného prostoru) jsou identifikátory, pomocí kterých se vytvoří kvalifikované jméno, které je následně vloženo do atributu *ref*. Obsah samotného elementu, resp. atributu není v takový okamžik transformován (dojde k poznamenání uzlu) a až v rámci fáze *Postprocessing* dojde k vytvoření daného uzlu v příslušném XML schémat.

3.4 Pokročilejší transformace XML schéma do X-definice

Vzhledem ke skutečnosti, že X-definice pokrývají větší prostor pro popis datových XML struktur, uvažujeme pouze bezztrátové transformace z XML schéma do X-definice.

3.4.1 Bezztrátové transformace

3.4.1.1 Mixed element

Pokud se v rámci XML schéma objeví uzel *xs:complexType* s atributem *mixed* a hodnotou *true*, potom je nutné zvolit vhodnou transformaci do předpisu X-definice, pomocí dostupných prostředků X-definice. Jako nejvhodnější se jeví do příslušného elementu vložit atribut *xd:text="? string()"* vzhledem k obecnosti validace atributu *mixed* v XML schématech. Zmíněný předpis umožňuje na libovolná místa v elementu vložit textový uzel obsahující řetězce o neomezené délce, což je nejbližší definice chování k atributu *mixed="true"* v XML schématech.

V rámci převodu si musíme pohlídat, do kterého uzlu v X-definici vložíme atribut *xd:text*. Zatímco v XML schéma je pozice atributu *mixed* jednoznačně určena, v X-definici existuje více možností, kam vložit zmíněný transformovaný atribut. S tímto souvisí i problém, že v X-definicích neexistuje ekvivalent k uzlu *xs:complexType*. Je proto potřeba brát do úvahy taktéž například reference v rámci schématu nebo složitější konstrukce.

3.4.1.2 Odkaz na skupinu elementů

V případě, kdy XML schéma obsahuje odkaz na skupinu elementů (uzel *xs:group* s atributem *ref*) s atributem *minOccurs* mající hodnotu vyšší než 1, pak je nutné tento uzel transformovat tak, aby atribut *minOccurs* měl hodnotu maximálně 1. Dalším omezením z pohledu X-definic je pozice uzlu pro odkaz na skupinu elementů (uzel *xd:mixed* s atributem *xd:script* obsahující klíčové

slovo *ref* – obraz transformace uzlu *xs:group*). Takový uzel není možné vložit na libovolnou pozici v rámci uzlů *element*, *sequence*, *choice*, ...

Z výše uvedených důvodů vyplývá, že pro zachování struktury předpisu (referencí) v případě použití odkazu na skupinu elementů, bude nutná pokročilá transformace s analýzou stromu předpisu, tak jako tomu bylo u převodu X-definic do XML schéma.

Vzhledem k tomu, že se jedná o jediný nalezený předpis v rámci této práce, který vyžaduje analýzu stromu předpisu při transformaci XML schémat do X-definic, a zároveň tento směr transformace má nižší prioritu a je obecném měřítku jednodušší, bude výše uvedený problém řešen zkratkou – namísto zachování reference na odkaz skupiny elementů, bude obsah skupiny elementů přímo vložen na pozici reference. V takovém případě lze přenést i hodnotu atributu *minOccurs* do uzlu uvnitř skupiny elementů[25]. Tento postup byl diskutován a schválen vedoucím práce.

3.5 Algoritmus transformace X-definice do XML schéma

Tato sekce obsahuje návrh realizace algoritmu pro transformaci X-definice. Zabývá se jak základním datovým modelem potřebným pro uskutečnění transformace, tak i procesem transformace samotné.

3.5.1 Datový model

Z analýzy problematiky a návrhu algoritmu pro transformaci předpisu X-definice do XML schéma vyplývá, že budou potřeba minimálně následující klíčové datové modely pro realizaci implementace.

3.5.1.1 SchemaNode

S přihlédnutím ke skutečnosti, že algoritmus bude vyžadovat pokročilejší *post-processing* na základě struktury vstupního předpisu X-definice, jehož výsledkem může být výrazně odlišný předpis typu XML schéma, je nutné, abychom měli možnost si zapamovat, který zdrojový uzel X-definice se mapuje na který výstupní uzel XML schéma.

Toto mapování je potřeba dělat pouze nad elementy, atributy a deklaracemi (včetně skupiny elementů). V podstatě jsou tímto výčtem pokryty všechny uzly, které mohou obsahovat věcnou informaci. Uzly jako *sequence*, *choice* a *xd:mixed*, resp. *xs:all* není potřeba mapovat. Nad rámec mapování uzlů vzoru a obrazu, bude vytvořena obousměrná relace mezi uzly, které obsahují reference, resp. deklaraci reference.

[25] Z pohledu XML schémat musí skupina elementů vždy obsahovat jeden z následujících uzlů: *all*, *choice* nebo *sequence*. Vzhledem k přímé transformaci těchto uzlů není problém na ně aplikovat atribut výskytu.

```
<xd:def xd:name="schemaNode" xd:root="a"
    xmlns:xd="http://www.xdef.org/xdef/3.2">
    <c d="int(2014)"></c>
    <a>
        <b xd:script = "ref c;"></b>
    </a>
</xd:def>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          attributeFormDefault="unqualified"
          elementFormDefault="unqualified">
   <xs:complexType name="c">
      <xs:attribute name="d" type="xs:int" use="required"/>
   </xs:complexType>
   <xs:element name="a">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="b" type="c"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

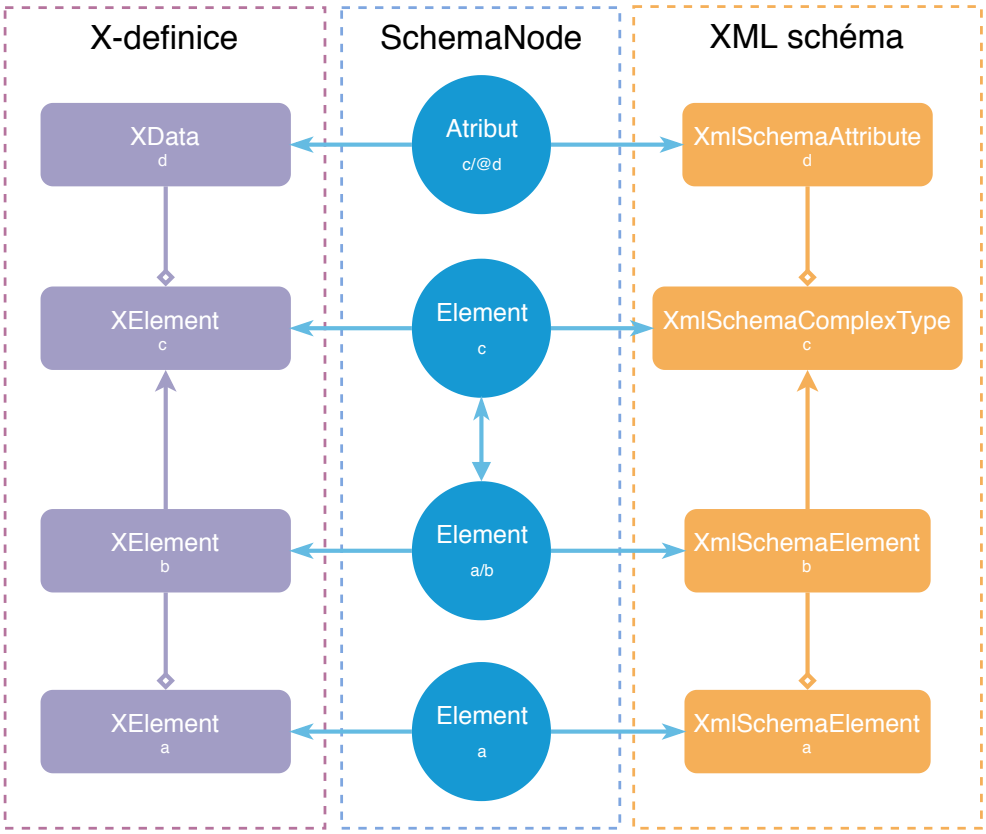
Ukázka 3.5: Příklad jednoduché X-definice a její transformace

Tento datový model může být vytvářen a aktualizován ve všech fázích algoritmu transformace. Jeho hlavním účelem je rychlý přístup ke zdrojovému uzlu v rámci X-definice, resp. k obrazu uzlu v XML schéma, a možná pokročilá analýza klíčových uzlů.

Celý datový model bude uložen v rámci kontextu transformace (viz 3.5.1.3). Součástí ukázky 3.5 je jednoduchá X-definice a její transformace do XML schéma. V průběhu transformace by měla vzniknout v paměti struktura uvedená na obrázku 3.2, která znázorňuje, co vše bude součástí datového modelu *SchemaNode*.

3.5.1.2 UniqueConstraint

Vzhledem k analýze kontroly existence a unikátnosti hodnot (viz 2.6.8) je nutné vytvořit model, který si uchová informace o místech a způsobech použití *uniqueSet* v X-definici. Tyto informace jsou získávány ve všech fázích algoritmu transformace. Na konci fáze *postprocessing* dojde k vyhodnocení použití jednotlivých *uniqueSet* a v případě splnění pravidel plynoucích z analýzy a návrhu transformace *uniqueSet* (viz 3.3.1.4) budou do XML schéma vloženy příslušné uzly pro kontrolu existence a unikátnosti hodnot.



Obrázek 3.2: Datový model *SchemaNode* vytvořený v rámci transformace X-definice z ukázky 3.5

Obdobně jako u datového modelu *SchemaNode* budou informace o *UniqueConstraint* uloženy v kontextu transformace.

3.5.1.3 Kontext transformace

Kontext transformace bude sloužit k uložení klíčových informací a dat, které jsou potřeba napříč algoritmem transformace. Tento kontext bude inicializován na začátku algoritmu a v jeho průběhu může být libovolně doplňován. Klíčem je, aby k tomuto kontextu měly přístup všechny třídy transformace – tím dojde k velkému ušetření výměny dat mezi jednotlivými třídami.

Hlavní informace a data, která by měla být součástí tohoto kontextu jsou:

- **Množina názvů XML schémat** – Knihovna *Apache XmlSchema* neposkytuje možnost, jak jednoduše zpětně získat jméno XML schéma. Dále tato množina bude kontrolovat unikátnost jmen XML schémat, aby nedocházelo k jejich kolizi.

- **Množina lokací XML schémat** – Pro každé XML schéma bude existovat záznam v této množině a to z důvod vytváření kolekce schémat, kdy mohou být použity uzly *xs:include* a *xs:import* vyžadující znalost lokace cílových XML schémat.

- **Výstupní kolekce XML schémat** – Umožňuje rychlý přístup do kolekce XML schémat, kterou právě vytváříme.

- **Množina uzlů určená k *postprocessingu*** – Pro každé XML schéma bude existovat množina uzlů X-definice, kterou je nutné transformovat až ve fázi algoritmu *postprocessing*.

- **Množina *SchemaNode*** – Pro každé XML schéma bude existovat mapování XPath cesty k uzlu (v rámci datové XML struktury) a modelu *SchemaNode*.

- **Množina *UniqueConstraint*** – Pro každé XML schéma bude existovat mapování XPath cesty k *uniqueSet* (v rámci X-definic) a modelu *UniqueConstraint*.

3.5.2 Návrh tříd

Návrh tříd je rozdělen do několika logických celků, které je potřeba v rámci implementace nutné vyřešit. Zmiňuji zde pouze ty nejdůležitější koncepty, zbytek považuji za implementační detaily, které jsou popsány v rámci kapitoly *Implementace a realizace* 4.

3.5.2.1 Datové typy

Jedním ze základních stavebních kamenů transformace je převod samotných datových typů a jejich případných omezení. Datové typy v rámci XML schéma jsou uloženy v uzlu *simpleType*. Tento uzel může obsahovat následující uzly: *xs:restriction*, *xs:list* a *xs:union*.

Pro sjednocení vytváření obsahu datových typů bude vytvořena třída *XsdSimpleContentFactory*, která bude poskytovat jednotné rozhraní pro vytvoření obsahu uzlu *simpleType*. Jejím výstupem tudíž bude instance třídy *XmlSchemaSimpleTypeContent*, která interně bude nést informaci o typu obsahu a obsahu samotném. V případě interního vytváření uzlu *xs:restriction* v rámci třídy *XmlSchemaSimpleTypeContent* je nutné transformovat taktéž omezující podmínky na datový typ. Řešení této transformace je popsáno v následující subsekci 3.5.2.2.

K samotné transformaci názvů datových typů bude použita pomocná třída obsahující mapování jednotivých datových typů X-definice na datový typ XML schémat.

3.5.2.2 Omezení datových typů

Vzhledem k tomu, že X-definice disponují mnohem větším počtem různých datových typů, které je navíc možné kdykoliv doplnit, je nutné, aby řešení transformace datových typů a zejména jejich omezení bylo dostatečně flexibilní.

Tyto požadavky lze řešit například vytvořením *interface* pro návrhový vzor *factory*, který bude sloužit k vytvoření omezení pro daný datový typ. Tento *interface* by měl poskytovat metodu pro převod vstupního předpisu X-definice, obsahující omezení datového typu, na seznam omezení v XML schématech (uzly *xs:facet*). Dále by měla existovat abstraktní *factory*, která bude obsahovat základní logiku převodu, která bude aplikována/dostupná pro transformaci všech omezení libovolného datového typu. Na základě těchto dvou předpokladů vznikne výchozí *factory* používaná pro převod libovolného datového typu. V případě, že datový typ požaduje speciální převod určitého omezení nebo musí být doplněn o omezení navíc, budou interně dostupné metody *customFacet*, resp. *extraFacets* pro řešení těchto případů. Dále by jednotlivé *factory* datových typů měly mít možnost přepsat výchozí transformaci omezení.

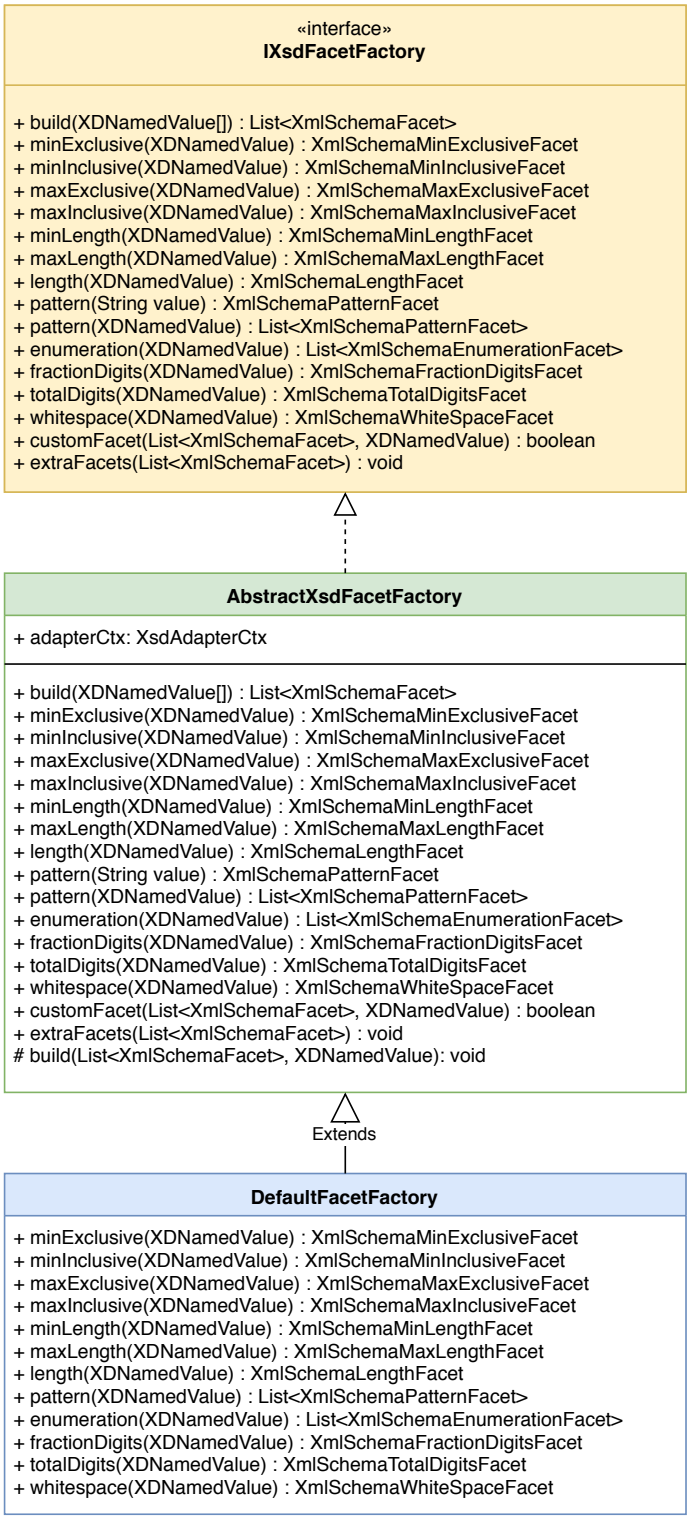
Ukázka 3.3 obsahuje návrh základní obecnou implementace tříd pro transformaci omezení datových typů[26]. Tato ukázka obsahuje pouze výchozí implementaci pro všechny datové typy, specifické datové typy mohou být vytvořeny stejným způsobem. Implementace metod vytvářející *facet* v rámci třídy *AbstractXsdFacetFactory* bude vracet pouze výjimku. Až na úrovni třídy *DefaultFacetFactory* bude existovat implementovace základního algoritmu pro transformaci příslušného *facet*. Tudíž implementace specifických datových typů může vycházet jak z třídy *AbstractXsdFacetFactory* tak i z *DefaultFacetFactory* a nemusí obsahovat implementaci transformace všech omezení.

3.5.2.3 Továrny objektů

Nedílnou součástí nového řešení algoritmu je i centrální (a mnohdy i kontrolované) vytváření instancí tříd.

XML schéma Vzhledem ke skutečnosti, že vytvoření prázdného XML schéma a následná jeho inicializace je složitější (rozumějme netriviální v ohledu počtu kroků, které je potřeba vykonat) proces (viz 3.2.1), bude vytvořena továrna *XsdSchemaFactory* pro tvorbu XML schémat. Toto řešení bude mít i výhodu v tom, že při vytváření dodatečných XML schémat nebude nutné implementovat vytváření XML schémat na více místech.

[26]Tento návrh neobsahuje *factory* pro jednotlivé datové typy. Lze si ji však představit buď místo *DefaultFacetFactory* nebo jako třídu, která rozšiřuje *DefaultFacetFactory*.



Obrázek 3.3: Návrh tříd pro vytváření omezení datových typů (XML schéma)

Uzly XML schémat Vytváření uzlů XML schémat je klíčovou funkcionalitou, kterou poskytuje knihovna *Apache XmlSchema*. Nad rozhraním této knihovny bude vytvořena továrna *XsdNodeFactory* pro vytváření prázdných i inicializovaných uzlů XML schémat. Účelem této továrny je poskytnou rozhraní, které vrací potřebné uzly pro transformaci předpisu X-definice. Veškeré vytváření uzlů XML schémat by se tudíž mělo odehrávat pouze ve zmíněné továrně.

Jména uzlů V průběhu transformace X-definice může z různých důvodů docházet k rozpadům uzlů, které je následně potřeba umístit do kořene XML schéma. Uzly umístěné v kořeni XML schémat musí mít unikátní jméno. Z tohoto důvodu, aby nedošlo ke konfliktu jmen v rámci XML schémat, je nutné vytvořit mechanismus, který bude vytvářet vždy unikátní jména pro kořenové uzly XML schémat (tento mechanismus se vztahuje i na kolekci XML schémat). Továrna *XsdNameFactory* bude tedy nejenom poskytovat rozhraní pro vytvoření nového jména uzlu, ale taktéž interně uchovávat všechny dosud vytvořená jména pro uzly v kořeni XML schémat pro budoucí kontrolu kolize.

SchemaNodeFactory Slouží k vytvoření, inicializaci a vytváření relací mezi jednotlivými modely *SchemaNode*. I přes jednoduchost modelu *SchemaNode* je nutné z uzlů X-definice zpracovat mnoho informací, abychom mohli instanci třídy *SchemaNode* vytvořit. Navíc vytváření relací mezi těmito uzly bude nutné provádět nad kontextem transformace, neboť deklarace reference může být do kontextu vložena dřív než samotná reference na ní.

Z těchto důvodů se jedná o komplexnější problematiku, kterou bude vhodné implementovat v oddělené třídě.

3.5.2.4 Třídy transformace

Třídy transformace obsluhují dílčí části převodu předpisu X-definice. Třídy jsou rozděleny do logických uskupení dle jejich zaměření. Jako celek utváří kompletní algoritmus transformace, který je volán skrze třídy *XDef2XsdAdapter* a *XdPool2XsdAdapter*. Obě zmíněné třídy slouží jako vstupní bod pro zahájení transformace předpisu a používají interně všechny níže uvedené třídy.

Vzhledem ke komplexnosti transformace X-definic do XML schémat rozlišujeme následující transformační třídy dle jejich zodpovědnosti:

- *Xd2XsdTreeAdapter*,
  - slouží k transformaci stromové struktury X-definice,
  - výstupem bude uzel z XML schéma obsahující stromovou strukturu,
  - interně bude používat *XsdNodeFactory*,

  - může být použita v libovolné fázi algoritmu,

- *Xd2XsdReferenceAdapter*,
  - slouží ke zpracování referencí,
    - transformace stromové struktury nekořenových globálních uzlů,
    - transformace deklarací X-definice,
    - vytvoření uzlů *xs:include* a *xs:import* na základě referencí,
  - interně bude používat *XsdNodeFactory*,
  - výstupem bude vložení uzlů *xs:complexType* a *xs:simpleType* do kořene XML schéma,
  - použita pouze po inicializaci XML schémau (před samotnou transformací stromové struktury),

- *Xd2XsdExtraSchemaAdapter*,
  - slouží k vytvoření dodatečných XML schémat,
  - obdobný přístup k vytvoření XML schéma jako ve třídě *XDef2XsdAdapter*[27],
  - výstupem je nové XML schéma (pokud ještě neexistovalo) nebo doplnění stávajícího schématu,
  - použita pouze ve fázi *Postprocessing* algoritmu,

- *Xd2XsdPostProcessingAdapter*,
  - slouží ke spuštění dodatečných *postprocessing* transformací,
    - doplnění uzlů na základě znalosti XML schémat,
    - aktualizace struktury XML schéma z důvodu dodržení pravidel XML schémat,
    - vytvoření kontroly existence a unikátnosti hodnot (viz 2.6.8),
  - interně používá pro účely doplňování uzlů transformační třídu *Xd2XsdExtraSchemaAdapter*,
  - interně používá pro aktualizaci struktury pomocnou třídu *XsdPostProcessor*,
  - použita pouze ve fázi *Postprocessing* algoritmu.

[27] Rozdíl spočívá v tom, že vstupem do této třídy může být více předpisů X-definice a seznam uzlů X-definice, které mají být v rámci daného XML schématu vytvořeny

3.5.2.5 Pomocné třídy

Pro další operace, které budou potřeba napříč algoritmem transformace, budou vytvořeny tzv. pomocné třídy, které budou dělit takové operace do logických celků. Z prvotní analýzy se jedná o třídy obsahující následující typy operací:

- **operace nad jmény uzlů** – slouží k získání potřebných formátů jmen z různých uzlů a atributů X-definice,
- **operace nad jmennými prostory** – slouží k získání a ověření jmenného prostoru z uzlu X-definice,
- **Postprocessing** – slouží k provedení pokročilých dílčích transformací na základě struktury předpisu XML schéma.

3.5.3 Návrh workflow

Obsah této subsekce se věnuje návrhu průběhu algoritmu transformace – jsou zde vysvětleny základní bloky jednotlivých fází algoritmu. Součástí obrázku 3.4 je digram zachycující průběh transformace kolekce X-definic a zodpovědnosti některých tříd ze subsekce *Návrh tříd* (3.5.2). Diagram transformace jedné X-definice je velmi obdobný uvedenému diagramu.

3.5.3.1 Inicializace a preprocessing

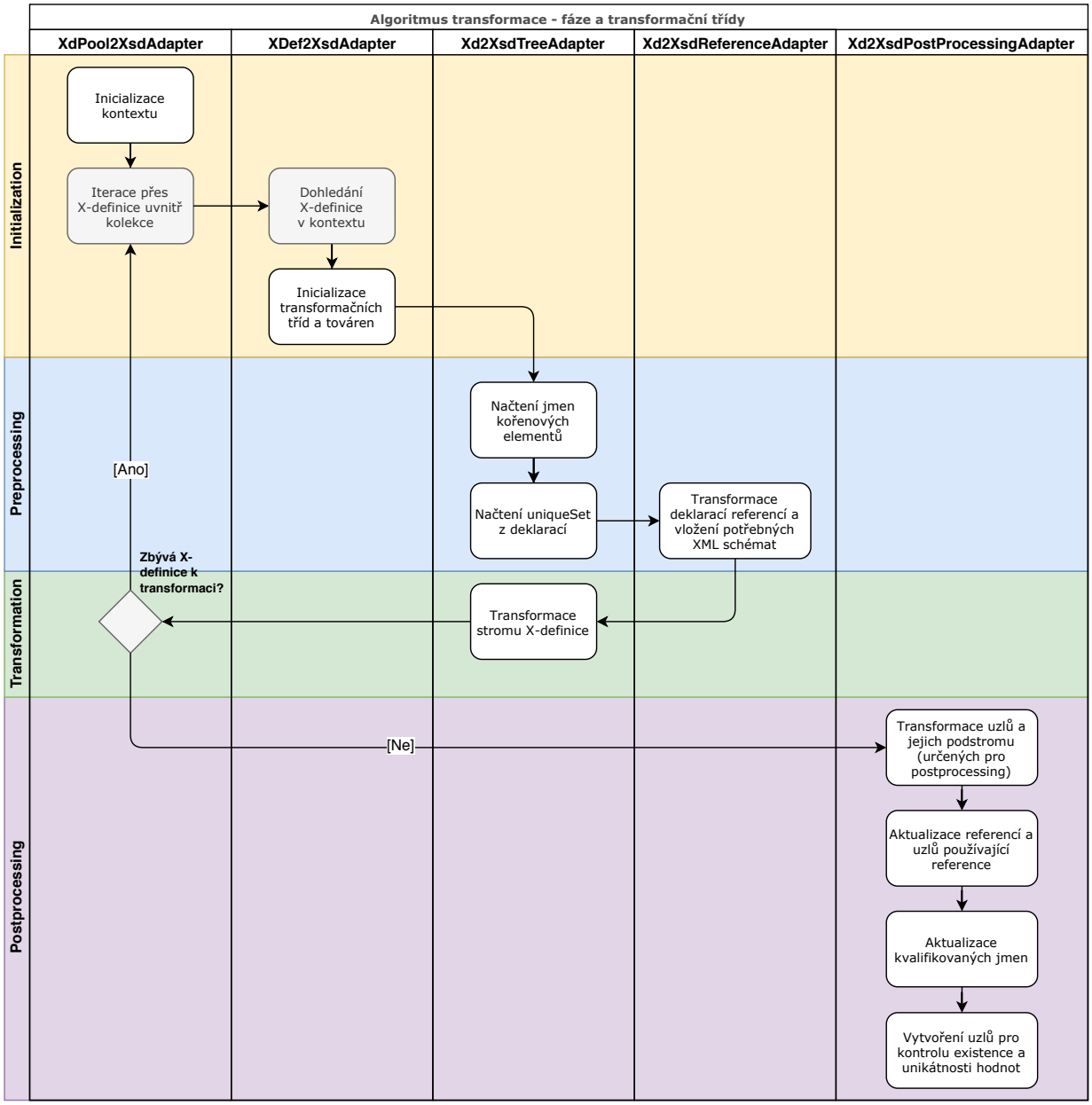
Tyto fáze slouží k načtení vstupních dat, inicializaci kontextu transformace a inicializace dalších tříd potřebných k provedení transformace. Dále se zde v rámci fáze *preprocessing* budou zpracovávat jmenné prostory, kořenové uzly a deklarace, cesty k jednotlivých XML schématům a proběhne taktéž samotná inicializace XML schémat.

3.5.3.2 Transformace

Fáze transformace bude mít dvě základní části:

1. **Transformace referencí** – budou transformovány deklarace referencí a vytvořena potřebná vložení ostatních XML schémat,
2. **Transformace stromové struktury předpisu** – transformace zbytku X-definice, stromová struktura začínající kořenovými uzly X-definice (definovány v rámci atributu *xd:root* v kořeném uzlu X-definice).

Výše uvedené rozdělení je z důvodu, že první část této fáze bude vytvářet de facto pouze deklarace včetně transformace jejich obsahu, tj. stromová struktura předpisu. Součástí této fáze bude kontrola i na unikátnost transformovaných uzlů, aby nedošlo k dvojité transformaci stejné deklarace (deklarace



Obrázek 3.4: Návrh workflow pro transformaci kolekce X-definic

může být v rámci předpisu použita na více místech). V rámci druhé fáze již budou existovat transformované deklace, tudíž se bude možné zaměřit pouze na stromovou strukturu předpisu X-definice. Reference na deklarace v této fázi stačí pouze zadefinovat pomocí kvalifikovaného jména.

Samotná transformace předpisu bude probíhat pomocí rekurzivního volání, které postupně prohledá celý strom předpisu, jehož kořen bude zadán na vstupu. V obou částech této fáze bude pro transformaci stromu předpisu použita třída *Xd2XsdTreeAdapter*.

3.5.3.3 Postprocessing

Postprocessing je poslední fází transformace uzavírající celý algoritmus převodu předpisu X-definice. Na rozdíl od předešlých fází má mnohem více rozmanitějších funkcí. V obecném měřítku bude mít tato fáze za úkol opravit XML schéma (aby bylo validní dle pravidel XML schémat) a doplnit některé uzly, které nemohly být vytvořeny v předešlých fázích. V této fázi budou:

1. transformovány uzly X-definice ležící v jiném jmenném prostoru,
2. aktualizovány deklarace referencí a uzly používající reference nad datovým modelem *SchemaNode*,
   - rozpad uzlů porušující pravidla XML schémat,
   - aktualizace referencí, tak aby dodržovala pravidla XML schémat,
3. aktualizována kvalifikovaná jména jednotlivých uzlů v XML schéma,
4. vytvořeny uzly pro kontrolu existence a unikátnosti hodnot (volitelně).

3.6 Algoritmus transformace XML schéma do X-definice

Na rozdíl od předchozí sekce, kde je popsán algoritmus pro transformaci X-definice, je převod XML schéma výrazně jednodušší a přímočarý. Tato skutečnost plyne z možností, kterými X-definice disponují. X-definice zároveň neobsahují tolik doplňujících pravidel pro podobu stromu předpisů jako XML schémata, tudíž přímočaře vytvořené předpisy X-definice jsou ve většině případů validní a není potřeba v tomto směru transformace fáze *postprocessing*.

3.6.1 Datový model

V transformaci XML schéma do X-definice nebude potřeba rozsáhlý datový model jako tomu bylo při opačném směru transformace (viz 3.5.1) a to zejména z důvodu přímé transformace formátu – v extrémním případě by se dalo uvažovat i o použití XSLT[20].

3.6.1.1 Kontext transformace

Stejně jako při převodu X-definice, bude kontext transformace sloužit k uložení klíčových informací a dat, které budou potřeba v rámci transformace na mnoha místech algoritmu.

Klíčové znalosti, která budou uloženy v kontextu jsou:

- **Množina názvů XML schémat** – Knihovna *Apache XmlSchema* neposkytuje cestu, jak jednoduše získat jméno vstupního XML schématu. Názvy X-definic budou odvozeny z této množiny.
- **Množina cílových jmenných prostorů X-definice** – Pro každou X-definici bude existovat záznam v této množině v případě, že zdrojové XML schéma používá cílový jmenný prostor. Tato množina bude následně použita pro generování jmen kořenových uzlů X-definice.
- **Množina jmenných prostorů X-definice** – Pro každou X-definici bude existovat množina jmenných prostorů, které byly součástí zdrojového XML schéma.
- **Mapování cílových jmenných prostorů na X-definici** – Pro každý použitý cílový jmenný prostor v XML schéma bude existovat záznam obsahující množinu X-definic, které tento cílový jmenný prostor používají.

3.6.2 Návrh tříd

Návrh tříd je rozdělen do stejných logických celků jako tomu bylo v subsekci 3.5.2 při opačném směru transformace.

3.6.2.1 Datové typy

Obdobně jako u transformace X-definice je nutné zajistit plnohodnotnou transformaci datových typů. Výhodou v tomto směru transformace je, že XML schémata obsahují menší počet datových typů. Z těchto datových typů jsou navíc běžně používány pouze ty nejzákladnější datové typy.

Tvorba datových typů je v X-definicích jednodušší než v XML schématech, je však nutné dodržet syntaxi zápisu v závislosti na tom, ve které části předpisu daný datový typ vytváříme. Rozlišujeme následující přístupy, jak můžeme datový typ deklarovat:

- **kořenová deklarace** – definice datového typu uvnitř uzlu *xd:declaration*,
- **deklarace textového uzlu** – definice datového typu uvnitř elementu, jehož součástí má být určitý typ textového obsahu,
- **holý datový typ** – definice datového typu pro použití v atributu nebo při definici typu položky v datovém typu *list* a *union*.

Za účelem jednotného vytváření datových typů bude vytvořena třída *XdDeclarationFactory*, která pomocí jednoduchého rozhraní bude poskytovat veškerou funkcionalitu pro vytvoření příslušného požadovaného datového typu. Vstupem do tohoto rozhraní bude inicializovaná třída *XdDeclarationBuilder*, která bude fakticky daný datový typ vytvářet. Inicializace zmíněné třídy bude mimo jiné vyžadovat, aby do ní byl vložen uzel *xs:simpleType* z XML schéma.

O *XdDeclarationFactory* se můžeme v tomto ohledu bavit jako o jakém si *wrapperu* logiky pro vytvoření datového typu.

Třída *XdDeclarationBuilder* bude interně používat rozhraní *IDeclarationTypeFactory*, jehož účelem bude transformovat uzly *xs:restriction* XML schéma do omezení v rámci X-definice, viz 3.6.2.2.

3.6.2.2 Omezení datových typů

Transformace omezení datových typů z XML schémat do X-definic je obdobná jako při opačném směru transformace, i přesto má své náležitosti. Hlavním rozdílem je, že v některých případech neexistuje univerzální zápis omezení. Ten se proto liší dle datového typu. Výsledně je však transformace XML schémat v tomto ohledu jednodušší.

Obdobně jako v subsekci 3.5.2.2 bude vytvořeno rozhraní poskytující klíčové metody pro vytvoření omezení datového typu. V tomto směru transformace se bude jednat o třídu *IDeclarationTypeFactory*. Nad touto třídou bude provedena abstraktní implementace továrny *AbstractDeclarationTypeFactory*, která poskytne základní výchozí algoritmus pro transformaci omezení.

Nezbytnou interní funkcionalitou *AbstractDeclarationTypeFactory* bude zajištění rozhraní, které dokáže určit, které *facet* z XML schéma již byly transformovány a které ještě ne. Tato funkcionalita se vyplatí zejména v místech, kde různé datové typy X-definice vyžadují (nebo umožňují) specifický zápis určitých omezení.

Následně nad abstraktní třídou *AbstractDeclarationTypeFactory* bude možné vytvořit konkrétní továrny dle potřeb datových typů X-definice (za předpokladu, že implementace *AbstractDeclarationTypeFactory* bude nedostačující). Tyto továrny budou mít možnost vybrat pouze potřebná omezení která chtějí transformovat a způsob implementace, jakým způsobem bude transformace probíhat.

3.6.2.3 Továrny objektů

Uzly X-definice Vytváření uzlů X-definice je obdobně jako u XML schéma klíčovou funkcionalitou. Tato továrna bude interně používat

- třídu *KXmlUtils* (z knihovny X-definice) – pro tvorbu kořenových uzlů X-definice a
- třídy z balíčku *org.w3c.dom* – pro tvorbu libovolného uzlu X-definice.

Továrna *XdNodeFactory* bude zajišťovat vytvoření veškerých potřebných uzlů X-definice, včetně nastavení jména nebo jmenného prostoru. Na jiných místech implementace by nemělo docházek k vytváření uzlů X-definice.

Atributy X-definice Vzhledem ke skutečnosti, že X-definice disponují mnoha typy atributů, bude vhodné tyto uzly vytvářet pomocí zastřešené funkcionality v továrně *XdAttributeFactory*. Zmíněná továrna bude poskytovat i rozhraní pro vytvoření speciálních atributů X-definice.

3.6.2.4 Třídy transformace

Zatímco u transformace X-definic jsme rozlišovali několik typů tříd transformace, tak u transformace XML schémat nám vystačí pouze jedna třída a to díky jednoduchosti převodu, který je možné de facto provést v rámci jednoho průchodu předpisu. Pro tyto učely bude vytvořena třída *Xsd2XdTreeAdapter*.

I přesto, že se jedná pouze o jednu třídu, bude vytvořena taktéž jako u převodu X-definic třída pro vstupní bod transformace – *Xsd2XDefAdapter*. Třída *Xsd2XdTreeAdapter* bude mít následující charakteristické vlastnosti:

- jejím účelem bude transformovat stromový předpis XML schéma,
- výstupem bude uzel obsahující transformovanou stromovou strukturu v X-definici,
- interně bude používat *XdNodeFactory*, *XdAttributeFactory*, *XdDeclarationFactory*,
- může být použita v libovolné fázi algoritmu.

3.6.2.5 Pomocné třídy

V pomocných třídách pro převod XML schémat rozlišujeme stejné logické celky jako u převodu X-definic, tj.

- **operace nad jmény uzlů** – slouží k získání potřebného formátu jmen,
- **operace nad jmennými prostory** – různorodé operace se jmennými prostory XML schémat.

3.6.3 Návrh workflow

Subsekce návrh workflow obsahuje návrh algoritmu transformace předpisu XML schéma do X-definice. V jednolivých částech jsou obecně vysvětleny základní stavební bloky algoritmu. Fáze *postprocessing* na rozdíl od transformace X-definic není potřeba.

3.6.3.1 Inicializace a preprocessing

V rámci fáze inicializace se načítají vstupní data a inicializuje se kontext transformace. Dále dojde k analýze vstupní kolekce XML schémat jejíž výsledkem bude množina XML schémat, která má být transformována. Tato analýza je potřebná z důvodu vzájemného stromového nebo kruhového vkládání XML schémat – v kolekci XML schémat se mohou objevovat duplicitní předpisy XML schémat.

Při *preprocessingu* proběhne pouze inicializace jmenných prostorů (včetně cílového jmenného prostoru) vstupního XML schéma. Všechny jmenné prostory jsou uloženy do kontextu transformace.

3.6.3.2 Transformace

Na rozdíl od převodu X-definice je fáze transformace pro XML schéma výrazně jednodušší. Transformace bude probíhat v následujících krocích:

1. **vytvoření kořenového uzlu** – může se jednat jak o uzel X-definice (*xd:def*) tak i uzel kolekce X-definic (*xd:collection*),
2. **transformace jmenných prostorů** – do kořenového uzlu X-definice budou přidány atributy definující jmenné prostory,
3. **transformace stromové struktury předpisu** – samotná transformace XML schéma.

Transformace stromové struktury XML schéma probíhá obecně nad všemi uzly, není zde žádný obdobný (*preprocessing*) jako je u transformace X-definice. Transformace tudíž probíhá lineárně a jsou postupně vytvářeny všechny uzly X-definice. Obdobně jako u transformace X-definice bude i tato transformace provedena pomocí rekurzivního volání a vstupem do tohoto volání bude uzel, který má být převeden (včetně celé stromové struktury, kterou obsahuje).

Kapitola 4

Implementace a realizace

Kapitola *Implementace a realizace* popisuje praktický výstup této diplomové práce opřený o analýzu a návrh uvedený v rámci tohoto textu. Implementační detaily se mohou částečně lišit, navíc mnoho implementační pasáží není součástí kapitoly *Návrh* z důvodu rozsahu implementace (zejména u algoritmu transformace X-definice do XML schéma). Všechny klíčové části samotného algoritmu transformace jsou však popsány v předešlých kapitolách.

4.1 Algoritmus transformace X-definice do XML schéma

Kompletní diagram tříd algoritmu pro transformaci X-definice, resp. kolekci X-definic, je součástí obsahu na přiloženém médiu ve složce *src/diagrams*.

4.1.1 Rozhraní algoritmu

Pro spuštění algoritmu zmíněné transformace existují dvě základní rozhraní a jejich implementace v závislosti na tom, zda je vstupem do algoritmu samotná X-definice (datový typ *XMDefinition*) nebo kolekce X-definice (datový typ *XDPool*).

V případě rozhraní třídy *XDef2SchemaAdapter* jsou k dispozici následující dvě metody implementované třídou *XDef2XsdAdapter*:

- *createSchema(XMDefinition)* – transformuje X-definici, která je zadaná jako parametr vstupu,
- *createSchema(XDPool)* – transformuje X-definici, kterou lze získat jako první ze vstupního parametru na vstupu (kolekce X-definic).

Rozhraní třídy *XdPool2SchemaAdapter* poskytuje pouze jen jednu metodu v rámci implementace *XdPool2XsdAdapter* a to

- *createSchemas(XDPool)* – transformuje kolekci X-definic, která je zadána jako parametr vstupu.

V obou výše uvedených případech je výstupem volaní metod kolekce XML schémat (datový typ *XmlSchemaCollection*). Důvod je takový, že i v případě transformace jedné X-definice může dojít k vytvoření více XML schémat, viz 2.6.7.

4.1.1.1 Načtení a zpracování vstupních dat

Vstupní data do algoritmu jsou již zkompilovaná a zvalidovaná, nejedná se tudíž o tzv. *plain* hodnotu. Důvodem tohoto způsobu implementace je, že zodpovědností algoritmu není kompilace ani validace vstupních dat. Tato zodpovědnost je přenesena na komponentu, která bude tento algoritmus využívat. Tento způsob implementace je jedním z hlavních bodů požadavků na novou implementaci.

Vstupní data jsou následně zpracovávána tak, že využíváme jednotlivé dostupné metody rozhraní knihovny X-definice a přistupujeme tak na položky již zkompilovaného objektu X-definice a uzlů uložených uvnitř tohoto objektu. Takto lze relativně jednoduše získat všechny potřebné informace o vstupní X-definici, resp. o kolekci X-definic.

Z pohledu algoritmu není nutné provádět dodatečnou validaci vstupních dat.

4.1.1.2 Výstup algoritmu

Výstupem algoritmu je vždy kolekce XML schémat. Tato kolekce je v počátku algoritmu inicializována jako prázdná a následně je v průběhu algoritmu postupně plněna. Tudíž je zaručeno, že vždy dojde k vrácení kolekce, pokud nedošlo k vyvolání výjimky v průběhu transformace.

4.1.2 Konfigurace algoritmu

4.1.2.1 Vlastnosti

Transformaci předpisů X-definice lze konfigurovat. Tato konfigurace lze rozdělit do třech hlavních skupin:

1. **vytváření uzlů** – uzly, které mohou být navíc v rámci transformace vytvořeny (částečně se může překrývat s některými nastaveními ze skupiny *postprocessing*),
2. **vlastnosti transformace** – upravuje chování transformace (například: vynucení tečky v desetinném čísle v rámci XML schéma),
3. **postprocessing** – povolení jednotlivých částí transformace ve fázi *postprocessing* algoritmu.

```
final XDef2XsdAdapter adapter = new XDef2XsdAdapter();
final Set<Xd2XsdFeature> features = Xd2XsdUtils.defaultFeatures();

features.add(Xd2XsdFeature.XSD_ANNOTATION);
features.add(Xd2XsdFeature.XSD_NAME_COLISSION_DETECTOR);
features.add(Xd2XsdFeature.POSTPROCESSING_KEYS_AND_REFS);

adapter.setFeatures(features);
```

Ukázka 4.1: Příklad konfigurace algoritmu pro převod X-definice

Jednotlivá nastavení konfigurace lze povolit nezávisle na sobě. Příklad nastavení je součástí ukázky 4.1. V první části ukázky je vytvoření třídy pro transformaci předpisu X-definice a získání výchozí konfigurace, která by měla být nastavena v rámci algoritmu transformace. Druhá část přidává následující dodatečná nastavení do výchozí konfigurace:

- **XSD_ANNOTATION** – povolí vytvoření uzlů *xs:annotation*, které obsahují dodatečnou informaci k transformaci (z konfigurační skupiny 1),
- **XSD_NAME_COLISSION_DETECTOR** – zapne dodatečnou kontrolu pro vytváření jmen uzlů v kořeni XML schémat, tak aby nedocházelo ke kolizi jmen (z konfigurační skupiny 2),
- **POSTPROCESSING_KEYS_AND_REFS** – v rámci fáze *postprocessing* algoritmu transformace budou vytvořeny uzly *xs:unique*, *xs:key* a *xs:keyref* pro kontrolu existence a unikátnosti hodnot (z konfigurační skupiny 3).

V rámci poslední části ukázky již pouze dochází k nastavení konfigurace do třídy *XDef2XsdAdapter*, která slouží k zahájení transformace.

4.1.2.2 Logování

V neposlední řadě lze nastavit úroveň logování, která má být použita v průběhu algoritmu transformace X-definice. Toto nastavení je obecné, probíhá pomocí třídy *SchemaLogger* a nevztahuje se ke konkrétní instanci třídy pro transformaci předpisu. Rozlišujeme následující úrovně logování, které jsou z pohledu dnešních implementací a návrhu standardní[21]:

1. **LOG_NONE** – logování vypnuto,
2. **LOG_ERROR** – zapnuto logování chybových zpráv, které vedou na neúspěšnou transformaci,
3. **LOG_WARN** – zapnuto logování upozorňovacích zpráv, které obsahují informaci o ztrátové transformaci,

4. **LOG_INFO** – zapnuto logování informačních zpráv, které obsahují informace o převedených uzlech,

5. **LOG_DEBUG** – zapnuto logování doplňujících zpráv, které obsahuje doplňující informace k transformaci uzlů,

6. **LOG_TRACE** – zapnuto logování doplňujících zpráv, které vypisují volání některých funkcí algoritmu.

4.1.3 Transformace – fáze preprocessing

Fáze tohoto algoritmu probíhá výhradně ve třídě *Xd2XsdReferenceAdapter*, která interně používá další třídy jako jsou například *XsdNodeFactory* a *Xd2XsdTreeAdapter*. Použití kontextu transformace je samozřejmostí, tak jako v ostatních třídách transformace.

Preprocessing v rámci třídy *Xd2XsdReferenceAdapter* probíhá ve dvou následujících fázích:

1. jednoduché datové typy (uzly *xs:simpleType*) a vložení XML schémat (uzly *xs:include* a *xs:import*),

2. komplexní datové typy (uzly *xs:complexType*).

V rámci ukázky 4.2 je metoda *extractRefsAndImports*. Jejím účelem je provést výše uvedené dvě fáze (viz komentáře v kódu). V obou případech jsou výsledkem transformace uzly, které jsou uloženy v kořeni XML schématu. Cílem této transformace je vytvořit všechny datové uzly, na které bude vytvořena reference v rámci zbytku předpisu[28].

Na začátku fáze *preprocessing* jsou navíc pomocí třídy *Xd2XsdTreeAdapter* z atributu *xd:root* v kořeni X-definice načtena jména kořenových uzlů. Načtená jména jsou v rámci algoritmu použita pro rozdělení fází transformace *preprocessing* a *transformace*. Dále jsou taktéž načteny *uniqueSet* ze všech deklarací v rámci dané X-definice.

4.1.3.1 Jednoduché datové typy a XML schémata

Strom předpisu je rekurzivně procházen. V rámci průchodu stromem X-definice se dohledávají:

- **atributy** – Pokud jméno atributu používá jiný prefix jmenného prostoru, je označen pro fázi *postprocessing*. V opačném případě, pokud atribut používá deklaraci, je tato deklarace transformována do *top-level* uzlu typu *xs:simpleType* pomocí volání třídy *XsdNodeFactory*. Při transformaci interně probíhá kontrola, zda již nebyl vytvořen uzel *xs:simpleType* se stejným jménem.

[28] Uzly vytvořené v této fázi algoritmu mohou obsahovat reference i mezi sebou.

```
private void extractRefsAndImports(final XDefinition xDef) {
    final Set<XMNode> processed = new HashSet<XMNode>();

    // Extract all simple types and imports
    for (XElement elem : xDef.getXElements()) {
        extractSimpleRefsAndImports(elem, processed, false);
    }

    // Extract all complex types
    final Set<String> rootNodeNames =
        adapterCtx.findSchemaRootNodeNames(schemaName);
    for (XElement elem : xDef.getXElements()) {
        if (rootNodeNames == null ||
            !rootNodeNames.contains(elem.getName())) {
            transformTopLevelElem(elem);
        }
    }
}
```

Ukázka 4.2: Ukázka kódu z fáze *preprocessing* algoritmu pro převod X-definice

Pokud atribut leží v jiném jmenném prostoru, než je deklarován cílový jmenný prostor výstupního XML schéma transformovaného atributu, potom dojde k vytvoření příslušného uzlu *xs:import*.

- **textové uzly** – Mají obdobně definovanou transformaci jako atributy s tou výjimkou, že pokud se jedná o složitější datový typ (s omezeními) a zároveň nepoužívají referenci, tak i přesto dojde k vytvoření uzlu *xs:simpleType* za stejných pravidel jako u atributu a vytvoření reference.
- **elementy používající referenci** – Pokud element používá referenci, potom se rozhoduje o tom, v jakém jmenném prostoru nebo X-definici leží. Na základě tohoto rozhodnutí se určí, zda a jaký uzel (*xs:include* nebo *xs:import*) by měl být vytvořen pro vložení příslušného XML schéma. V opačném případě, kdy element nepoužívá referenci, se kontroluje, zda jméno elementu používá jiný prefix jmenného prostoru (obdobně jako u atributů). Na základě této informace se rozhodne o tom, zda bude element vytvořen v rámci fáze *postprocessing* nebo dojde k vytvoření uzlu *xs:import* s příslušným XML schéma.

4.1.3.2 Komplexní datové typy

Transformace elementů X-definice ve fázi *postprocessing* se týká pouze elementů, které jsou uloženy v kořeni X-definice, ale zároveň nemohou být kořenem datové XML struktury, neboli tyto kořeny nejsou uvedeny v atributu *xd:root* v kořeni X-definice (viz ukázka 4.2).

```
private void transformXdef(final Xd2XsdTreeAdapter treeAdapter) {
    final Set<String> rootNodeNames =
        adapterCtx.findSchemaRootNodeNames(xDefinition.getName());

    if (rootNodeNames != null) {
        for (XElement elem : xDefinition.getXElements()) {
            if (rootNodeNames.contains(elem.getName())) {
                treeAdapter.convertTree(elem);
            }
        }
    }
}
```

Ukázka 4.3: Ukázka kódu z fáze *transformace* algoritmu pro převod X-definice

V rámci transformace těchto uzlů mohou vzniknout následující *top-level* uzly XML schématu:

- ***xs:element*** – Pokud transformovaný uzel X-definice neobsahuje uzel *xs:complexType*, potom bude v kořeni uložen jako uzel *xs:element*. V opačném případě dojde k dále uvedeným případům.
- ***xs:group*** – Pokud zdrojem transformace byl uzel *xd:mixed*, potom výsledkem transformace bude uvedený uzel *xs:group*.
- ***xs:complexType*** – Ve všech ostatních případech bude vytvořen tento uzel.

Ve všech výše uvedených případech je aplikován stejný algoritmus transformace. Tato transformace je prováděna pomocí třídy *Xd2XsdTreeAdapter*. Jedná se o stejný algoritmus transformace, jako je použit v rámci fáze *transformace* (viz 4.1.4). Jediným rozdílem je, že na úrovni třídy *Xd2XsdReferenceAdapter* se rozhodne, který uzel by měl být vložen do kořene XML schématu – na základě tohoto rozhodnutí se výstupní transformovaný obsah z třídy *Xd2XsdTreeAdapter* dodatečně upraví.

4.1.4 Transformace – fáze transformace

Fáze transformace je de facto jednoúčelová operace, jejímž účelem je převést stromovou strukturu mající za kořen jeden z elementů uvedených v atributu *xd:root* v kořeni X-definice. O nastavení vstupů (jednotlivých elementů) se stará přímo třída *XDef2XsdAdapter*, následně celá transformace je provedena pomocí třídy *Xd2XsdTreeAdapter*, viz ukázka 4.3.

4.1.4.1 Xd2XsdTreeAdapter

Hlavní zodpovědností třídy *Xd2XsdTreeAdapter* je transformace uzlů stromové struktury X-definice do stromové struktury XML schématu. Vstupem do metody *convertTree* je libovolný uzel X-definice, výstupem je libovolný uzel XML schéma. Jak vstup, tak výstup mohou obsahovat další vnořenou stromovou strukturu. Metoda *convertTree* následně interně volá metodu *convertTreeInt*, která je rekurzivní a prochází celý podstrom daného uzlu. Zdrojový kód metody *convertTreeInt* je součástí přiložené ukázky D.1.

Tato třída si interně pomocí množiny uzlů X-definice ukládá, které uzly již byly transformovány. Díky tomu nedochází k vícenásobným transformacím uzlů, které vznikají díky referencím. Třída *Xd2XsdTreeAdapter* dále interně používá zejména *XsdNodeFactory* a *XsdPostProcessor*. Důvodem pro použití třídy *XsdPostProcessor* je, že i v rámci přímé transformace stromu předpisu X-definice existují situace, kdy je občas zapotřebí provést drobnou dodatečnou úpravu výstupního předpisu XML schéma. Jedná se však výhradně o minimální zásahy do struktury XML schéma – například změna typu jednoho konkrétního uzlu.

Metoda *convertTreeInt* interně používá různé další metody pro vytvoření příslušných uzlů XML schématu na základě typu vstupního uzlu X-definice. Hlavní rozdělení uzlů a jejich transformace je popsána dále v této subsekci.

Transformace atributu

- V případě, kdy uzel definující atribut používá referenci, která leží v jiném jmenném prostoru, potom je do tohoto uzlu vložena reference pomocí kvalifikovaného jména a uzel deklarace reference je určen pro *postprocessing*.

- V opačném případě probíhají kontroly v následujícím pořadí (může nastat právě jedna situace):

  1. zda atribut používá *uniqueSet* – pokud ano, je datový typ atributu získán z *uniqueSet*,
  2. zda používá odkaz na deklaraci – pokud ano, je vložen do uzlu *xs:attribute* atribut *type* s příslušným kvalifikovaným jménem,
  3. zda používá datový typ s omezením – pokud ano, je vložen do uzlu opět atribut *type* (obdobně jako v případě výš), deklarace datového typu byla vytvořena již ve fázi *preprocessing*,
  4. zda používá fixní hodnotu – pokud ano, je vložen do uzlu atribut *fixed* s příslušnou hodnotou,
  5. pokud nenastala ani jedna z výše uvedených situací, potom dojde k vytvoření uzlu *xs:simpleType* uvnitř uzlu *xs:attribute*.

- Nakonec je doplněna výchozí hodnota do atributu XML schéma (atribut *default*), pokud uzel v X-definici používá v datovém typu *default*.

Transformace elementu

- Načtení použitých *uniqueSet* zadefinovaných v rámci atributu *xd:script* v transformovaném elementu.
- V případě, kdy element používá referenci a sám ji nijak nerozšiřuje, potom:
  - pokud reference
    - leží v jiném jmenném prostoru (porovnání URI jmenného prostoru),
    - používá odlišný prefix pro jmenný prostor,
    - leží v jiné X-definici,

    tak dojde k vytvoření reference atribut *ref* uvnitř transformovaného elementu. Reference je definována pomocí kvalifikovaného jména.
  - V opačném případě je v rámci elementu XML schéma vytvořen atribut *type*, jejímž použitím očekáváme, že daná reference je uzel typu *xs:complexType*. Stejně jako v předchozím případě je hodnotou atributu *type* kvalifikované jméno.
- V případě, kdy element používá referenci, kterou rozšiřuje svojí definicí o atributy nebo uzly, potom:
  1. dojde k vydefinování atributů, které jsou součástí definice elementu, nikoliv reference,
  2. dojde k vydefinování uzlů, které jsou součástí definice elementu, nikoliv reference.

  Předpokládáme, že existuje alespoň jeden atribut nebo uzel rozšiřující transformovaný element (v opačném případě by algoritmus vykonával předchozí větev). Pokud by však náhodou byl element X-definice rozšířen takovým způsobem, který transformační algoritmus nedokáže rozeznat a množina vydefinovaných atributů a uzlů by byla výsledně prázdná, potom by se s uzlem zacházelo jako s uzlem používající referenci bez rozšíření. V opačném případně následně dojde k:
  - vytvoření uzlu *xs:extension*, který obsahuje atribut *base* odkazující na referenci,
  - naplnění uzlu *xs:extension* atributy a uzly vydefinovanými jako součást aktuálně transformovaného elementu,

  - vložení vytvořeného uzlu *xs:extension* do nového uzlu *xs:complexContent*,
  - vložení vytvořeného uzlu *xs:complexContent* do nového uzlu *xs:complexType*,

  Uzel typu *xs:complexType* již lze použít jako obsah uzlu typu *xs:element*[29].

- V opačném případě, kdy element nepoužívá referenci, ale leží v jiném jmenném prostoru, potom je element označen k *postprocessingu* a aktuálně probíhající transformace je doplněna o referenci na element, který bude v rámci fáze *postprocessing* vytvořen.

- Pokud nenastal ani jeden z uvedených případů, pak dojde k převodu celého obsahu (včetně stromové struktury) transformovaného elementu.

  - Pokud element X-definice obsahuje pouze textový uzel (ani žádné atributy), pak dojde ke speciální transformaci textového uzlu – výstupní element bude naplněn uzlem *xs:simpleType*, který bude definovat požadovaný datový typ textového uzlu.
  - V opačném případě dojde uvnitř elementu k vytvoření uzlu *xs:complexType*, který bude obsahovat všechny uzly a atributy. Tato transformace je řešena pomocí metody *createComplexType* v třídě *Xd2XsdTreeAdapter*, která slouží právě a pouze k vytvoření uzlu *xs:complexType* uvnitř uzlu *xs:element*.

Transformace textového uzlu

- U transformace textového uzlu v případě, kdy element obsahuje i atributy, je potřeba, aby datový typ takového textového uzlu byl buď

  - zadefinován jako *top-level* uzel typu *xs:simpleType* nebo,
  - známý datový typ XML schéma.

  Následně se textový uzel odkazuje na takto vytvořenou deklaraci datového typu, resp. známý datový typ. V tomto případě se textový uzel deklaruje pomocí uzlu *xs:simpleContent*, jehož obsahem je uzel *xs:extension* s atributem *base* odkazujícím na deklaraci pomocí kvalifikovaného jména.

- V opačném případě, kdy element obsahuje další elementy a textový uzel zároveň, se jedná o ztrátovou transformaci, která je popsána v rámci subsekce 3.3.1.1 v kapitole *Návrh*. Tento případ transformace je řešen naprosto odlišně od výše uvedeného. Při této transformaci je nutné zachovat použití uzlu *xs:complexType* z rodičovského elementu, ve kterém je dodatečně doplněn atribut *mixed* s hodnotou *true*.

[29] `https://www.w3.org/TR/xmlschema-1/#declare-element`

Transformace skupiny elementů Tato transformace je jednou ze zajímavějších částí algoritmu pro transformaci předpisu X-definice. Samotná transformace uzlu skupiny elementů je triviální záležitost – stačí vytvořit příslušný uzel a do něj začít vkládat transformované elementy. Problém vzniká však ve chvíli, kdy dojde z pohledu XML schémat k porušení pravidel pro vložení uzlů do této skupiny nebo k porušení použití samotné skupiny (viz například 3.3.1.2). Dalším komplikací je, že z pohledu struktury zkompilované X-definice se skupina elementů tváří jako zásobník, nikoliv jako vnoření uzlů.

Celý problém transformace skupiny elementů je implementován v metodě *createComplexType* třídy *Xd2XsdTreeAdapter*. Algoritmus simuluje zásobník skupiny elementů jak je tomu v X-definici, zatímco výstupní XML schéma obsahuje standardně vložené skupiny elementů. Součástí této metody je volání *updateGroupParticles*, které zajišťuje validitu umístění skupin elementů dle pravidel XML schémat. To znamená, že se v tomto případě pokročilejší transformace neřeší v rámci fáze *postprocessing*, ale ihned při transformaci jednotlivých uzlů. Mimo to, volání metody *updateGroupParticles* může restrukturalizovat obsah celého zásobníku skupin elementů. Z tohoto důvodu je v rámci metody *createComplexType* zavedena proměnná *stackPopCounter*, která udržuje informaci o tom, kolik skupin elementů v rámci změn na zásobníku bylo vyhozeno. Tímto mechanismem lze zpětně validně zpracovávat zkompilovanou X-definici.

V rámci restrukturalizace obsahu zásobníku může dále dojít například k přesunování elementů mezi skupinami elementů. Z pohledu transformace se též jedná o triviální operace.

4.1.5 Transformace – fáze postprocessing

V průběhu všech předchozích fází, včetně té současné, je naplňován datový model *SchemaNode*, který je průběžně ukládán do kontextu transformace. Tento datový model je následně použit pro pokročilejší transformace v rámci fáze *postprocessing*. Tuto fázi zajišťuje třída *Xd2XsdPostProcessingAdapter* (viz schéma na obrázku 3.4), přičemž některé použité částečné algoritmy transformace v této třídě jsou součástí třídy *XsdPostProcessor* (viz dále).

4.1.5.1 Transformace uzlů

V rámci celého průběhu algoritmu, a to včetně této fáze, byly a stále jsou ukládány uzly, které leží v jiném jmenném prostoru, než je cílový jmenný prostor XML schéma, kam jsou dané transformované uzly ukládány. Tato skutečnost vede na to, že některé uzly je potřeba transformovat dodatečně. Dále z tohoto faktu plyne, že v rámci fáze *postprocessing* mohou vzniknout XML schémata navíc.

Celá dodatečná transformace uzlů vypadá tak, že se nejdřív zjistí, zda vůbec existují nějaké uzly a XML schémata určené k transformaci v rámci

fáze *postprocessing*. Pokud ano, dojde k iteraci přes všechny X-definice zadané na vstupu. Nad těmito X-definicemi je inicializován speciální adapter pro transformaci uzlů, třída *Xd2XsdExtraSchemaAdapter*. Implementace třídy *Xd2XsdExtraSchemaAdapter* obsahuje logiku, jak doplnit dodatečně transformované uzly do příslušných XML schémat. Výstupem volání této třídy je množina jmenných prostorů, které byly aktualizovány. Tato třída si dále interně drží informaci o kontextu jmenného prostoru původního XML schéma, než byl tento kontext modifikován zmíněnou třídou[30].

Pokud se v rámci dané instance třídy *Xd2XsdExtraSchemaAdapter* zjistí, že existují uzly, které je potřeba v dané vložené vstupní X-definici transformovat, pak dojde k internímu vytvoření transformačního kontextu, který zadefinuje nové transformační třídy a použije původní datové modely (vnořená třída *SchemaAdapter* v *Xd2XsdExtraSchemaAdapter*). Takto vytvořený transformační kontext najde nebo vytvoří nové potřebné XML schéma a následně do něj vloží všechny uzly určené k transformaci, resp. *postprocessingu*. **V rámci této operace může dojít opět k nalezení dalších uzlů určených ke zpracování ve fázi *postprocessing*.**

Při transformaci uzlů ve fázi *postprocessing* jsou postupně odebírány uzly, které byly transformovány a naopak jsou přidávány nově objevené uzly určené pro tento proces – v takovém okamžiku je nutné aktualizovat kontext uzlů, které zbývají k převedení. Fáze *postprocessing* nad danou X-definicí končí ve chvíli, kdy neexistuje žádný další uzel, který má být vyřešen v rámci fáze *postprocessing* pro příslušnou X-definici. Část zdrojového kódu transformace uzlů X-definice v *postprocessing* fázi je součástí přiložené ukázky D.3.

4.1.5.2 Aktualizace referencí

O transformaci, která aktualizuje reference a jejich použití, se stará již zmíněna třída *XsdPostProcessor*. Tato třída je i přesto provolána skrze *Xd2XsdPostProcessingAdapter*, neboť v ní dochází ke kontrole konfigurace algoritmu – zda je příslušná operace *postprocessingu* povolena. V rámci třídy *XsdPostProcessor* je nutné zavolat metodu *processRefs* pro provedení aktualizace referencí.

V průběhu implementace a testování jsem narazil na více typů referencí, které je nutné na konci transformačního algoritmu aktualizovat dle pravidel XML schémat. Tyto aktualizace většinou plynou z nedodržení pravidel XML schéma. Typickým zástupcem tohoto problému je například použití atributu *ref* v uzlu *xs:element*, který musí ukazovat na uzel typu *xs:element*, avšak v průběhu algoritmu bylo rozhodnuto, že deklarace reference bude uzel typu *xs:complexType*. V následujících odstavcích jsou popsány další situace, kdy nastane aktualizace referencí, a jejich jednotlivá řešení.

[30] S transformací nových uzlů mohou přibýt i nové potřebné jmenné prostory v rámci daného XML schéma.

***Top-level* nekořenový element, který je referován** Základní problém transformace, kdy X-definice umožňuje rozeznat pomocí atributu *xd:root*, jaké elementy mohou být kořenem datové XML struktury. Tuto funkcionalitu XML schémata postrádají – počítají se s tím, že veškeré kořenové uzly mohou být zároveň kořenem datové XML struktury. Tento problém by teoreticky šel řešit ihned ve fázi *transformace*. Avšak v návaznosti na takové řešení může vzniknout další problém, který souvisí s použitím referencí. Tento problém by byl nutný řešit až v této fázi (*postprocessing*). Z tohoto důvodu jsou nekořenové *top-level* elementy celkově řešeny až v této fázi.

V této transformaci je nutné zajistit rozpad uzlu typu *xs:element* na uzel *xs:complexType* (původní uzel bude z XML schéma odstraněn). Dále je nutné zajistit unikátnost jmen nově vytvořených uzlů a v případě, že nějaký uzel odkazoval na tento transformovaný uzel, potom je nutné změnit atribut z *ref* na *type*. Kvalifikované jméno (hodnota atributu *type*) by mělo zůstat nezměněno za předpokladu, že nový uzel bude mít stejné jméno a ležet ve stejném jmenném prostoru jako původní uzel typu *xs:element*.

Top-level* element, který používá atribut *ref Bez ohledu na to, zda se jedná nebo nejedná o kořenový element XML schéma, není dovoleno v rámci XML schémat, aby *top-level* uzel typu *xs:element* obsahoval atribut *ref*. Z tohoto důvodu je potřeba (obdobně jako v předchozím případě) zajistit převod transformovaného uzlu na uzel typu *xs:complexType*. Na rozdíl od předchozího případu nemá takovýto element žádný podstrom elementů, a proto je z toho důvodu tato *postprocessing* transformace výrazně jednodušší. Opět je však nutné zajistit aktualizaci atributů u uzlů referující na původní, resp. nový uzel.

Kvalifikovaný *top-level* element, který je referován z nekvalifikovaného elementu V porovnání s předešlým případem se jedná o trochu odlišnou situaci, kdy nebudeme mazat původní uzel z XML schéma. V tomto případě, pro dodržení všech pravidel a zároveň zachování pokrytí předpisu XML struktur, je nutné vytvořit nový uzel typu *xs:complexType*, který bude disponovat novým jménem a obsahem dotčeného elementu. Dále bude dotčený element odkazovat na nově vytvořený uzel pomocí atributu *type* a všechny uzly, které původně měly referenci na transformovaný element budou nově odkazovat na vytvořený uzel *xs:complexType*.

Aktualizace referencí používající špatný atribut V poslední řadě je nutné zkontrolovat, zdali nedošlo v rámci libovolné fáze algoritmu transformace k porušení pravidel XML schémat při použití referencí. Je nutné proto provést následující kontroly:

- uzel typu *xs:element* při referování *xs:element* vždy používá atribut *ref*,

- uzel typu *xs:element* při referování *xs:complexType* vždy používá atribut *type*.

4.1.5.3 Aktualizace kvalifikovaných jmen

Po doplnění uzlů v rámci fáze *postprocessing* může dojít k problémům s kvalifikovanými jmény, které vznikly na základě rozkladu jmenných prostorů. Z tohoto důvodu je nutné na závěr algoritmu transformace zkontrolovat a případně zaktualizovat následující:

- kvalifikovaná jména atributů, resp. elementů,
- použité výchozí jmenné prostory (atribut *form*) u atributů, resp. elementů,
- kvalifikovaná jména referencí.

4.1.5.4 Vytvoření uzlů pro kontrolu existence a unikátnosti hodnot

Dodatečně lze do XML schéma (nepovinně) doplnit uzly pro kontrolu existence a unikátnosti hodnot. Jedná se o uzly *xs:unique*, *xs:key* a *xs:keyref*. Návrh, za jakých podmínek mohou být tyto uzly vytvořeny a podoba jejich samotného mapování je součástí subsekce 3.3.1.4 v rámci kapitoly *Návrh*. Vzhledem ke skutečnosti, že se jedná o *proof-of-concept feature* nebudu zabíhat do velkých implementačních detailů.

Podstatnou částí implementace je, aby vytvořené uzly byly umístěny ve správných hloubkách, resp. uzlech. Toho lze docílit pomocí analýzy XPath jednotlivých použití proměnné *uniqueSet* a samotné definice této proměnné, kterou získáme z X-definice v předchozích fázích algoritmu. Dále je nutné zajistit unikátnost jmen těchto uzlů. Tento problém je řešen jednoduše pomocí čítače, který se inkrementuje při vytvoření klíče pro každou novou transformaci *uniqueSet*. Do jména uzlů je dále přidáváno i jméno samotného *uniqueSet*.

4.2 Algoritmus transformace XML schéma do X-definice

Na rozdíl od opačného směru transformace popisovaného v sekci 4.1, je tento směr transformace výrazně jednodušší a hlavně přímočarý (bez nutnosti použití fáze *postprocessing*).

Kompletní diagram tříd algoritmu pro transformaci XML schémat je součástí obsahu na přiloženém médiu ve složce *src/diagrams*.

4.2.1 Rozhraní algoritmu

Rozhraním algoritmu transformace XML schémat je třída *Schema2XDefAdapter*, jejíž jediná implementace *Xsd2XDefAdapter* poskytuje pouze následující metodu pro provedení požadované transformace. Jedná se o metodu *createXDefinition(XmlSchema, String)*, jejíž první argument je vstupní XML schéma určené k převodu a druhý argument definuje jméno výstupní kořenové X-definice.

4.2.1.1 Načtení a zpracování vstupních dat

Vstupem do algoritmu je odobně jako v předchozím případě zkompilovaný objekt, v tomto případě se však jedná o XML schéma. Toto XML schéma interně obsahuje kolekci XML schémat – tudíž i v případech, kdy potřebujeme transformovat více XML schémat v rámci kolekce, je možné použít výše uvedenou metodu. Důvody pro použití zkompilovaného objektu na vstupu do algoritmu jsou stejné jako v subsekci 4.1.1.1, tj. přenos zodpovědnosti na externí komponentu.

Vstupní data jsou opět zpracovávána pomocí dostupných metod v rozhraní knihovny *Apache XmlSchema*, která nám umožňuje získat informace o XML schématech a jejich obsahu. V první řadě je však nutné vyřešit, jaká všechna XML schémata je potřeba transformovat, neboť kolekce XML schémat může obsahovat i duplicitní schémata z důvodu vícenásobných nebo kruhových vložení a importů XML schémat.

Z pohledu algoritmu není nutné provádět dodatečnou validaci vstupních dat.

4.2.1.2 Výstup algoritmu

Metoda *createXDefinition(XmlSchema, String)* v rámci rozhraní *Schema2XDefAdapter* vrací řetězec, který obsahuje předpis X-definice, resp. kolekce X-definic v textovém XML formátu[31]. Výstup algoritmu je průběžně tvořen v rámci transformace.

4.2.2 Konfigurace algoritmu

Algoritmus transformace XML schémat lze konfigurovat obdobně jako tomu je při transformaci X-definice. Konfigurace logování se nijak neliší (popis viz 4.1.2.2) a konfigurace vlastností probíhá totožně, pouze definice samotných vlastností algoritmu transformace jsou odlišné (viz dále).

4.2.2.1 Vlastnosti

Vlastnosti, které lze nastavit u transformace XML schémat do X-definic jsou na rozdíl od opačného směru transformace spíše doplňující. Jedná se o násle-

[31] Existují i jiné způsoby zápisu X-definice, například binární (serializovaný)

dující vlasnosti:

- **XD_EXPLICIT_OCCURRENCE** – všechny uzly X-definice budou implicitně obsahovat v atributu *xd:script* hodnotu *occurs 1*,
- **XD_TEXT_REQUIRED** – všechny textové uzly v X-definici budou implicitně povinné,
- **XD_MIXED_REQUIRED** – všechny elementy v X-definici povolující kombinaci textu a uzlů (atribut *xd:text*) budou vyžadovat existenci textu.

Jednotlivá nastavení lze konfigurovat nezávisle na sobě. Použití je obdobné jako pro opačný směr konfigurace (viz ukázka 4.1), je potřeba však dosadit příslušné třídy – v tomto případě třídu *Xsd2XDefAdapter* za *Xdef2XsdAdapter* a třídu *Xsd2XdUtils* za *Xd2XsdAdapter*.

4.2.3 Transformace – fáze preprocessing

Fáze *preprocessing* je výrazně méně obsáhlá než při transformaci X-definice. Je potřeba provést pouze dva následující triviální kroky:

1. na základě analýzy vytvořit novou kolekci XML schémat, která má být transformována (vyřadit duplicitní XML schémata – viz 4.2.1.1),
2. načíst veškeré jmenné prostory (včetně cílového jmenného prostoru) XML schémat z kolekce vytvořené v předcházejícím bodě do kontextu transformace.

Vzhledem k nenáročnosti této fáze jsem v rámci implementace nevytvářel speciální třídu pro provedení fáze *preprocessing* a implementace této fáze je obsahem třídy *Xsd2XDefAdapter*.

4.2.4 Transformace – fáze transformace

Transformace stromové struktury XML schémat do X-definic je v podstatě přímočará operace, při které jsou v rámci XML schémat vynechány některé výskyty uzlů, které jsou pro X-definice nadbytečné (jako například uzel *xs:complexType* uvnitř uzlu *xs:element*). Zbytek uzlů je transformován 1:1. K samotné transformaci uzlů je použita třída *Xsd2XdTreeAdapter* (viz 4.2.4.1). Zmíněná třída společně s *Xsd2XDefAdapter* interně používají pro vytvoření uzlů X-definice třídu *XdNodeFactory*.

Transformace začíná v třídě *Xsd2XDefAdapter*, ve které probíhá vytvoření a inicializace kořenových uzlů X-definice v metodě *createXDef*, resp. kolekce X-definic, resp. *createPool*. V obou případech je interně použita třída *XdNodeFactory*. Počátek transformace X-definice je součástí ukázky 4.4.

```
Element xdRootElem;
if (schemasToBeProcessed.size() > 1) {
    xdRootElem = elementFactory.createPool();
    // First transform root XSD document
    xdRootElem.appendChild(createXDef(xDefName, rootSchema, true));

    for (XmlSchema schema : schemasToBeProcessed) {
        if (rootSchema.equals(schema)) {
            continue;
        }

        final String schemaName = adapterCtx.findXmlSchemaName(schema);
        xdRootElem.appendChild(createXDef(schemaName, schema, true));
    }
} else {
    xdRootElem = createXDef(xDefName, rootSchema, false);
}
```

Ukázka 4.4: Ukázka kódu z fáze *transformace* algoritmu pro převod XML schéma

O transformaci stromového předpisu XML schéma se stará metoda *transformXsdTree*, která postupně iteruje přes *top-level* uzly (*xs:complexType*, *xs:simpleType*, *xs:group* a *xs:element*). V rámci zmíněné iterace jsou tyto uzly transformovány pomocí třídy *Xsd2XdTreeAdapter* metodou *convertTree*.

4.2.4.1 Xsd2XdTreeAdapter

Jak již bylo zmíněno, hlavní odpovědností třídy *Xsd2XdTreeAdapter* je transformace stromové struktury z XML schéma do X-definice. Pro spuštění tohoto převodu slouží jediná veřejná metoda k tomu určená a to *convertTree*. Tato metoda je zároveň rekurzivní a slouží k transformaci celé stromové struktury XML schéma.

Na rozdíl od transformace X-definice není potřeba si interně pamatovat, které uzly již byly převedeny, neboť převod je zcela přímočarý. Jedinou složitější částí transformace jsou datové typy (viz kapitola Návrh 3.6.2.1).

Transformace elementu

- Vytvoření kostry uzlu X-definice obsahující název elementu, v případě potřeby i jmenný prostor. Pokud element z XML schéma používá referenci, tak dojde i k naplnění atributu *xd:script* příslušnou hodnotou pro vytvoření reference v rámci X-definice.

- Pokud transformovaný element obsahuje atribut *type*, pak

  - v případě, že se atribut *type* odkazuje na *xs:complexType*, potom dojde k vytvoření reference uvnitř uzlu X-definice,
  - v případě, že se atribut *type* odkazuje na *xs:simpleType*, potom dojde k vytvoření textového uzlu pomocí příslušného nastavení *XdDeclarationBuilder* a následného zavolání metody *createDeclarationContent* z třídy *XdDeclarationFactory*.

- Pokud element z XML schéma obsahuje uzel *xs:complexType* nebo *xs:simpleType*, pak
  - v případě, že se jedná o *xs:complexType*, potom dojde k transformaci komplexního datového typu (viz dále),
  - v případě, že se jedná o *xs:simpleType*, potom dojde k vytvoření textového uzlu obdobným způsobem jak je popsáno výše.
- Na závěr dojde k doplnění výskytu elementu do atributu *xd:script* a nastavení *nillable* pokud bylo použito v rámci XML schéma.

Transformace komplexních datových typů Tato transformace popisuje převod uzlu typu *xs:complexType*. Níže uvedený postup se vztahuje i na *toplevel* uzly tohoto typu s tou výjimkou, že tato transformace je navíc obalena do uzlu element X-definice.

- V první řadě dojde k transformaci atributů z *xs:complexType* do rodičovského uzlu typu element v rámci X-definice.
- Pokud uzel obsahuje tzv. *particle* uzel, pak:
  - v případě, že se jedná o partikulární skupinu elementů, dojde k transformaci popsané níže v rámci odstavce *Transformace partikulární skupiny elementů*,
  - v opačném případě dojde k transformaci stromové struktury voláním rekurzivní metody *convertTree*, do které jako vstupní argument vstupuje *particle* uzel.
- Pokud transformovaný uzel obsahuje atribut *mixed* s hodnotou *true*, potom dojde k vytvoření atributu *xd:text* v rodičovském uzlu typu element (více v kapitole *Návrh* 3.3.1.1).
- Pokud transformovaný uzel obsahuje uzel *xs:extension* uvnitř uzlu
  - *xs:complexType*, pak dojde k
    1. transformaci atribůtu z *xs:extension* do příslušného uzlu typu element v rámci X-definice,
    2. vytvoření reference v rámci příslušeného elementu, pokud uzel *xs:extension* používá atribut *base*,

3. transformaci uzlu definující partikulární skupinu elementů, pokud je takový uzel součástí *xs:extension*.

– *xs:simpleType*, pak dojde k

1. transformaci atributu z *xs:extension* do příslušného uzlu typu element v rámci X-definice,
2. vytvoření textového uzlu uvnitř příslušného elementu, používající datový typ založený na atributu *base* z uzlu *xs:extension*.

Transformace partikulární skupiny elementů Všechny transformace uzlů partikulárních skupin jsou přímočaré. Z pohledu X-definice je však například použití uzlu *xd:sequence* uvnitř uzlu *element* nadbytečné za předpokladu že uzel *xd:sequence* v XML schéma obsahuje pouze uzly typu *element*. Pokud tedy XML schéma obsahuje uzel *xs:sequence* jehož součástí jsou pouze *elementy*, pak výstupní X-definice neobsahuje uzel *xd:sequence*.

Transformace všech partikulárních skupin elementů probíhá v následujících krocích:

1. vytvoření prázdného příslušného uzlu v X-definici (*xd:sequence* – výjma pravidle uvedeného výše, *xd:choice* a *xd:mixed*),

2. transformace všech uzlů vložených do partikulární skupiny v rámci XML schéma pomocí metody *convertTree*.

3. transformace informace o výskytu partikulární skupiny elementů (součástí obsahu atributu *xd:script*).

Transformace definice skupiny elementů Transformace definice skupiny elementů probíhá pomocí výše uvedené transformace *Transformace partikulární skupiny elementů*. Tato transformace je navíc obalena do uzlu *xd:mixed* (viz 2.6.1.2).

Transformace reference na skupinu elementů Přímá transformace (zachovávající strukturu předpisu) by v tomto případě vyžadovala pokročilejší transformaci, která by analyzovala strukturu předpisu XML schéma nebo výstupního předpisu X-definice. Navíc existují další problémy související s přímou transformací této reference, jako je například kardinalita nebo pozice takového uzlu v rámci X-definice (více viz například v subsekci 3.4.1.2). Z těchto důvodů je zde aplikována zkratka, která se vyhýbá všem těmto problémům (včetně pokročilé transformace) – místo transformace reference na skupinu elementů je definice skupiny elementů přímo vkládána místo reference do výstupní X-definice. Uzel *xs:group* obsahující referenci je tudíž přeskočen a pomocí metody *convertTree* dojde k transformaci definice skupiny elementů, která je popsána výše.

```
public void createDeclaration(final XdDeclarationBuilder builder) {
    if (builder.parentNode == null) {
        return;
    }

    final Element xdDeclaration = xdFactory.createEmptyDeclaration();
    xdDeclaration.setTextContent(builder.build());
    builder.parentNode.appendChild(xdDeclaration);
}
```

Ukázka 4.5: Kód pro vytvoření deklarace v X-definicích

Transformace datového typu O transformaci datového typu se starají třídy *XdDeclarationFactory* a *XdDeclarationBuilder* (viz dále). K nastavení instance třídy *XdDeclarationBuilder* dochází v rámci třídy *Xsd2XdTreeAdapter* a následně je tato instance předána jako argument do *XdDeclarationFactory*.

4.2.4.2 XdDeclarationFactory

Poskytuje základní rozhraní, které umožňuje určit, jaký typ deklarace má být vytvořen (ukázka 4.5 obsahuje metodu *createDeclaration* z rozhraní). V rámci tohoto rozhraní může dojít i k vytvoření uzlů X-definice, které přímo souvisí s daným typem deklarace. Dále tato továrna poskytuje základní inicializaci třídy *XdDeclarationBuilder* a ukládá si informace o *top-level* deklaracích jednoduchých datových typů (uzel *xs:simpleType*), které již byly transformovány, aby nedocházelo k vícenásobným transformacím stejných uzlů z XML schéma.

4.2.4.3 XdDeclarationBuilder

Třída *XdDeclarationBuilder* obsahuje veškerou logiku pro vytvoření potřebného datového typu a případně i jeho deklarace. Následuje výčet klíčových parametrů, které lze v této třídě nastavit:

- **simpleType** – zdrojový uzel XML schéma, který má být transformován do deklarace v rámci výstupní X-definice,
- **type** – typ deklarace, který má být vytvořen (definice těchto typů lze dohledat v 3.6.2.1),
- **baseType** – kvalifikované jméno datového typu v XML schéma.

Po nastavení všech příslušných parametrů již stačí pouze zavolat metodu *build*, která interně zprocesuje všechny zadané parametry a na základě jejich hodnot vytvoří požadovaný datový typ. V rámci implementace není zmíněná metoda volána přímo, ale vždy skrze metody třídy *XdDeclarationFactory*.

4.3 Porovnání algoritmů

Struktura algoritmů je do značné míry stejná, včetně jejich průběhu. V implementaci transformace X-definic je navrženo a implementováno mnohem více funkcionalit, které souvisejí s komplexností X-definic. Cílem těchto funkcionalit je dohledat vhodný obraz v rámci XML schémat. Implementace zmíněných funkcionalit (a nejenom funkcionalit) je opřena o analýzu a návrh, které v rámci této práce byly vypracovány. Algoritmus pro transformaci XML schémat je znatelně štíhlejší i díky minimálnímu *preprocessingu* a nepotřebné fázi *post-processing*.

Fázi *transformace* považuji do velké míry pro obě strany převodu za ekvivalentní, neboť se většinou jedná pouze o přímočarý převod uzlů. V rámci této fáze se řeší i transformace datových typů. V obou případech transformace bylo nutné implementovat dostatečně robustní logiku k podchycení všech požadavků pro převody jednotlivých datových typů. Ze zdrojového kódu lze jednoduše rozpoznat, že implementace převodu datových typů X-definice byla náročnější a rozsáhlejší oproti datovým typům XML schémat. Tato skutečnost vyplývá i z faktu, že X-definice obsahují více datových typů než XML schéma.

Rozhraní algoritmů požaduje z pohledu logiky obdobné typy vstupních objektů a snahou bylo vytvořit rozhraní co nejjednodušší. Taktéž způsob konfigurace algoritmu je totožný pro oba směry transformace. Jediný rozdíl spočívá v návratovém typu, kdy v rámci transformace XML schéma je výsledkem *raw* (řetězcová) hodnota X-definice, která ze své vlastní definice není zkompilována, zatímco výsledkem transformace X-definice je kolekce zkompilovaných XML schémat.

Kapitola 5

Testování

5.1 Odlaďování testovacích případů

Pro odlaďování jednotlivých problémových testovacích případů jsem v průběhu vypracování této práce používal externí nástroje, které mi efektivně pomáhaly dohledávat chyby v jednotlivých výstupních předpisech z algoritmu transformace.

Pro transformaci X-definic do XML schéma existuje mnoho nástrojů (ať už dostupných online nebo jako aplikace), jak zkontrolovat XML schéma. Po otestování mnoha nástrojů jsem zvolil jako nejvhodnější pro mé potřeby následující nástroje:

- *Online XML Validator (XSD)*[32] od *Liquid Technologies*. Tato společnost navíc poskytuje například nástroje i pro generování XML dat na základě předpisu XML schéma.

- *XMLSpy*[33] od společnosti *Altova*. Tento nástroj je velmi pokročilý, umožňuje i validaci XML dat vůči kolekci XML schémat (tuto funkcionalitu první zmíněný nástroj postrádá). Dále například podporuje zobrazení diagramu XML schémat, což se se zvyšující se komplexitou X-definic ukázalo jako velmi užitečná funkcionalita.

V opačném směru transformace, XML schémat do X-definic, je jediným dostupným nástrojem pro validaci předpisu X-definic *X-definition validator*[34] od společnosti *Syntea software group a.s.*

[32] `https://www.liquid-technologies.com/online-xsd-validator`
[33] `https://www.altova.com/xmlspy-xml-editor`
[34] `http://xdef.syntea.cz/tutorial/examples/validate.html`

```
File xmlDataFile = getXmlDataFile(fileName, testingFile);
File xDefFile = getInputXDefFile(fileName);
ArrayReporter reporter = new ArrayReporter();
XDDocument xdDocument = XValidate.validate(null, xmlDataFile,
    (File[])Arrays.asList(xDefFile).toArray(), fileName, reporter);
assertTrue(xdDocument != null, "XML is not valid against
    x-definition. Test=" + fileName + ", File=" + testingFile);
assertFalse(reporter.errors(), "Error occurs on x-definition
    validation. Test=" + fileName + ", File=" + testingFile);
```

Ukázka 5.1: Ověření testovacích dat proti předpisu X-definice

5.2 Popis testovacího frameworku

V rámci testování jsem si implementoval vlastní jednoduchý testovací framework, který je postaven nad již existujícím testovacím frameworkem v rámci knihovny X-definice. Implementaci vlastní nadstavby frameworku jsem provedl z důvodu specifických požadavků na unit testy pro algoritmus transformace. Touto implementací zároveň vznikla množina regresních testů, které umožňovaly v rámci vývoje algoritmů rychlé podchycení nově vzniklých chyb.

Testovací framework jsem implementoval pro oba směry transformace, jedná se však o velmi obdobné implementace. Z tohoto důvodu oba frameworky dědí z třídy *TesterXdSchema*, která obsahuje základní společnou logiku pro testovaní obou směrů transformace.

Testování obou algoritmů obsahuje základní utility, jako je například načtení souborů, kompilace X-definice a XML schéma, atp. Těmto utilitám se v rámci kapitoly *Testování* věnovat nebudu, považuji je za triviální a zcela základní.

Celé testování je postaveno na následujícím principu (konkrétnější implementace toho pricipu je popsána v jednotlivých směrech transformace):

1. načtení vstupních předpisů ze souboru, resp. souborů,
2. uložení výstupního předpisu z algoritmu do souboru,
3. ověření testovacích dat (validních i nevalidních) vůči vstupním a výstupním předpisům (kontroluje se shoda výsledku validace).

5.2.1 Ověřování testovacích dat

Ověřování testovacích dat vůči předpisu X-definice je zajištěno implementací v rámci knihovny X-definice, třída *XValidate*. Příklad ověření validních XML dat vůči X-definici je součástí ukázky 5.1 (včetně aserce).

K testování dat vůči XML schéma jsem použil prostředky jazyku Java, který obsahuje pro tyto účely již implementované příslušné třídy. Konkrétní

implementace pro tento testovací framework se nachází v testovacím balíčku ve třídě *XmlValidator*, jejíž součástí je jednoduché ověření XML dat vůči XML schéma, resp. kolekci XML schémat.

5.3 Testování transformace X-definice do XML schéma

V rámci testování transformace X-definice je k dispozici více metod pro testování vstupních předpisů. Tyto metody se liší na základě toho zda:

- vstupní předpis je X-definice nebo kolekce X-definic,
- existuje reference výstupního souboru předpisu,
- je potřeba zapnout konkrétní *feature* v rámci daného testu.

Volání všech výše uvedených metod končí v metodě *convertXdDef2Xsd* při transformaci X-definice, resp. *convertXdPool2Xsd* při transformaci kolekce X-definic. Tyto obě metody mají z pohledu logiky velmi podobnou implementaci, liší se však v některých implementačních detailech které souvisí právě v rozdílu načtení a kompilace X-definice a kolekce X-definic.

5.3.1 Průběh testování

Unit test tohoto směru transformace v případě převodu kolekce X-definic (metoda *convertXdPool2Xsd*) probíhá v následujících krocích:

1. vytvoření a inicializace instance třídy *XdPool2XsdAdapter* pro provedení transformace,
2. načtení množiny vstupních souborů předpisů kolekce X-definice,
3. kompilace souborů kolekce X-definic do instance třídy *XDPool*,
4. **transformace** objektu *XDPool* do kolekce XML schémat (objekt *XmlSchemaCollection*),
5. uložení výstupního objektu *XmlSchemaCollection* do souborů,
6. (nepovinné) kontrola výstupu vůči referenčním souborům,
7. testování validních a nevalidních vstupních dat vůči vstupním předpisům kolekce X-definic (slouží k ověření očekávaných výsledků),
8. testování pozitivního scénáře (validní vstupní data) vůči výstupní kolekci XML schémat,

```
convertXdDef2XsdNoRef ("groupChoice1",
    Arrays.asList(new String[] {"groupChoice1_valid_1",
        "groupChoice1_valid_2"}),
    Arrays.asList(new String[] {"groupChoice1_invalid_1",
        "groupChoice1_invalid_2"})
);
```

Ukázka 5.2: Volání testovací funkce pro převod X-definice

9. testování negativního scénáře (nevalidní vstupní data) vůči výstupní kolekci XML schémat.

V případě, že dojde k chybě v libovolném výše uvedeném kroce, je test označen jako neúspěšný. Součástí ukázky 5.2 je volání testovací metody *convertXdDef2XsdNoRef*, která spouští unit test, jehož součástí je:

- kolekce X-definic s kořenovým názvem X-definice *groupChoice1* (hlavní soubor kolekce X-definic je uložen ve stejnojmenném souboru),
- testovací datové soubory *groupChoice1_valid_1* a *groupChoice1_valid_2* obsahující validní XML (implicitně je doplněna přípona *.xml*),
- testovací datové soubory *groupChoice1_invalid_1* a *groupChoice1_invalid_2* obsahující nevalidní XML (implicitně je doplněna přípona *.xml*).

Dále součástí testu není referenční výstupní soubor, tudíž šestý krok unit testu nebude vykonán.

5.4 Testování transformace XML schéma do X-definice

Pro testování transformace XML schéma, resp. kolekce XML schémat existují obdobné typy testovacích metod jako tomu bylo při transformaci předpisu X-definice (viz 5.3). Na rozdíl od předešlého případu nerozlišujeme vstupní data, zda se jedná o jedno XML schéma nebo kolekci XML schémat, ale naopak rozlišujeme typ výstupu, tj. zda se jedná o jednu X-definici nebo kolekci X-definic.

Volání všech testovacích metod v rámci testování tohoto směru transformace končí na metodě *convertXsd2XDef* ať už je očekávaným výstupem jedna X-definice nebo kolekce X-definic.

5.4.1 Průběh testování

V případě unit testů pro transformaci XML schémat probíhá testování velmi odobně jako pro opačný směr transformace. Kroky testování jsou následující:

1. vytvoření a inicializace instance třídy *Xsd2XDefAdapter* pro provedení transformace,
2. načtení množiny vstupních souborů kolekce XML schémat a jejich kompilace (objekt *XmlSchema*[35]),
3. **transformace** objektu *XmlSchema* do *raw* formátu X-definice (objekt *String*),
4. uložení výstupního objektu *String* do souboru,
5. (nepovinné) kontrola výstupu vůči referenčnímu souboru,
6. testování validních a nevalidních vstupních dat vůči vstupním předpisům kolekce XML schémat (slouží k ověření očekávaných výsledků),
7. testování pozitivního scénáře (validní vstupní data) vůči výstupní X-definici, resp. kolekci X-definic,
8. testování negativního scénáře (nevalidní vstupní data) vůči výstupní X-definici, resp. kolekci X-definic.

Obdobně jako při transfomaci X-definice, v případě výskytu chyby v libovolném kroce, bude daný test považován za neúspěšný. Pro unit testy existuje obdobné rozhraní jako v ukázce 5.2 s rozdílným pojmenováním metod (pořadí a typ parametrů je zachován).

[35]Bystrý čtenář si všimne, že v opačném směru transformace byl použit objekt typu *XmlSchemaCollection* pro kolekci XML schémat. Avšak v tomto směru transformace potřebujeme znát kořenové XML schéma (jako vstup do algoritmu). Naštěstí toto kořenové schéma již obsahuje informaci o celé kolekci XML schémat – viz 4.2.1.1.

Závěr

Cílem této diplomové práce byla analýza specifikací X-definic a XML schémat, dále návrh a implementace modulu pro obousměrnou transformaci těchto formátů. Součástí analýzy bylo mimo jiné i seznámení se se současnou implementací modulu, která řeší stejnou problematiku, avšak zcela nedostatečně a to zejména při transformaci X-definic do XML schémat.

V rámci analýzy jednotlivých formátů jsem narážel na mnohá úskalí, která tyto předpisy přináší, zejména potom XML schéma a jeho doplňující pravidla pro strukturu předpisu. Veškeré složitější koncepty transformace byly řešeny až v rámci návrhu.

Řešení problematických transformací, zejména potom těch ztrátových, byly diskutovány s vedoucím práce, kterému jsem vždy předložil možné řešení konkrétní transformace (v některých případech existovalo i více řešení).

K vypracování této práce bylo klíčovým krokem se seznámit se specifikacemi jednotlivých předpisů a následně se naučit sestavovat tyto předpisy tak, aby pokryly co největší prostor datových XML struktur, který byl popsán vzorovým předpisem.

Algoritmus, který je výstupem této práce, je připraven k použití a otestován na množině testovacích dat, která pokrývá větší množinu případů, než bylo původním cílem této práce. I přesto se s jistotou najdou takové předpisy (ať už X-definice nebo XML schéma), které se nepodaří tímto algoritmem správně transformovat. To však není chybou této práce, neboť jejím cílem bylo pokrýt co největší množství běžných případů použití X-definic a XML schémat. Tyto případy byly předem vytipovány pevně danou testovací množinou dat, kterou tento algoritmus umí korektně transformovat.

Výsledný algoritmus pro převod X-definice do XML schémat je z mého pohledu dostatečně robustní a flexibilní. Tato skutečnost plyne i z rozdělení algoritmu na jednotlivé fáze, které ve výsledku zjednodušují zpracování komplexních X-definic.

Vzhledem ke komplexitě obou předpisů lze tuto práci i nadále rozšiřovat – je možné analyzovat a implementovat okrajové případy transformací, které

nejsou součástí této práce. Dalším samotným tématem je analýza kontroly unikátnosti a existence hodnot, kterému se tato práce věnuje pouze okrajově. V neposlední řadě by bylo možné hlouběji analyzovat ztrátové transformace za účelem nalezení takového převodu, který ztratí co nejméně informací o původním předpisu. V této diplomové práci jsem zmíněné problematice věnoval velké množství času – i přesto může dojít k dohledání konkrétních předpisů takových, že jejich ztrátová transformace není řešena optimálně (v rámci možností) a lze ji tudíž provést s menší ztrátovostí.

Literatura

[1] Kocman, J.; Trojan, V.: *X-components 3.2*. Leden 2019, [cit. 2020-01-06].

[2] Wikipedia: Java Architecture for XML Binding. [online], 2018, [cit. 2020-01-02]. Dostupné z: `https://en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding`

[3] O'Reilly: A Short History of XML Schema Languages [online]. [online], 2002, [cit. 2019-12-11]. Dostupné z: `https://docstore.mik.ua/orelly/xml/schema/appa_03.htm`

[4] Wikipedia: Schematron. [online], 2019, [cit. 2020-01-02]. Dostupné z: `https://en.wikipedia.org/wiki/RELAX_NG`

[5] Wikipedia: Document type definition. [online], 2019, [cit. 2020-01-02]. Dostupné z: `https://en.wikipedia.org/wiki/Document_type_definition`

[6] Wikipedia: RELAX NG. [online], 2019, [cit. 2020-01-02]. Dostupné z: `https://en.wikipedia.org/wiki/Schematron`

[7] Wikipedia: XML Schema (W3C) — Wikipedia, The Free Encyclopedia. [online], 2019, [cit. 2019-12-11]. Dostupné z: `https://en.wikipedia.org/wiki/XML_Schema_(W3C)`

[8] Trojan, V.; Kamenický, J.; Měska, J.: Why All of Humanity Does Not Speak Esperanto. 2017, [cit. 2020-01-02]. Dostupné z: `http://xdef.syntea.cz/tutorial/en/userdoc/XMLPrague2007_eng.pdf`

[9] Trojan, V.: X-definition Tutorial. [online], 2019, [cit. 2020-01-06]. Dostupné z: `http://xdef.syntea.cz/tutorial/`

[10] Trojan, V.; Srp, J.: *X-definition 3.2 - Language description*. 2019, [cit. 2019-12-12].

[11] W3C: *XML Schema Part 2: Datatypes Second Edition*. Říjen 2004, [cit. 2019-12-12]. Dostupné z: https://www.w3.org/TR/xmlschema-2/

[12] Wikipedia: Document Object Model. [online], 2019, [cit. 2020-01-02]. Dostupné z: https://en.wikipedia.org/wiki/Document_Object_Model

[13] W3C: *XML Schema Part 1: Structures Second Edition*. Říjen 2004, [cit. 2019-12-12]. Dostupné z: https://www.w3.org/TR/xmlschema-1/

[14] W3Schools: XML Schema Tutorial. [online], 1999-2020, [cit. 2020-01-06]. Dostupné z: https://www.w3schools.com/xml/schema_intro.asp

[15] Webucator: Webucator's Free XML Schema Tutorial. [online], 2004-2020, [cit. 2020-01-06]. Dostupné z: https://www.webucator.com/tutorial/learn-xml-schema/index.cfm

[16] Trojan, V.: *X-definition 3.2 - Manual*. Říjen 2019, [cit. 2019-12-12].

[17] Vlist, E. V.: *XML Schema*. OREILLY MEDIA, 2002, ISBN 0596002521, [cit. 2019-12-15]. Dostupné z: https://www.ebook.de/de/product/3250456/eric_van_vlist_xml_schema.html

[18] Michels: XML Schema qualified, unqualified what's it all about? 2018, [cit. 2019-12-14]. Dostupné z: https://dzone.com/articles/logging-levels-what-they-are-and-how-they-help-you

[19] Xebia: Maximize cohesion Minimize coupling. [online], 2014, [cit. 2019-12-12]. Dostupné z: http://essentials.xebia.com/maximize-cohesion-minimize-coupling/

[20] Wikipedia: XSLT. [online], 2019, [cit. 2020-01-06]. Dostupné z: https://en.wikipedia.org/wiki/XSLT

[21] Dietrich, E.: Logging Levels: What They Are and How They Help You. 2017, [cit. 2020-01-06]. Dostupné z: http://michielv.eu/2012/11/12/xml-schema-qualified-unqualified-whats-it-all-about/

Příloha **A**

Seznam použitých zkratek

DTD Document type definition

DOM Document Object Model

JSON JavaScript Object Notation

ISO International Organization for Standardization

JAXB Java Architecture for XML Binding

RELAX NG REgular LAnguage for XML Next Generation

URI Uniform Resource Identifier

W3C World Wide Web Consortium

XML Extensible Markup Language

XPath XML Path Language

XSLT eXtensible Stylesheet Language Transformations

YAML Ain't Markup Language

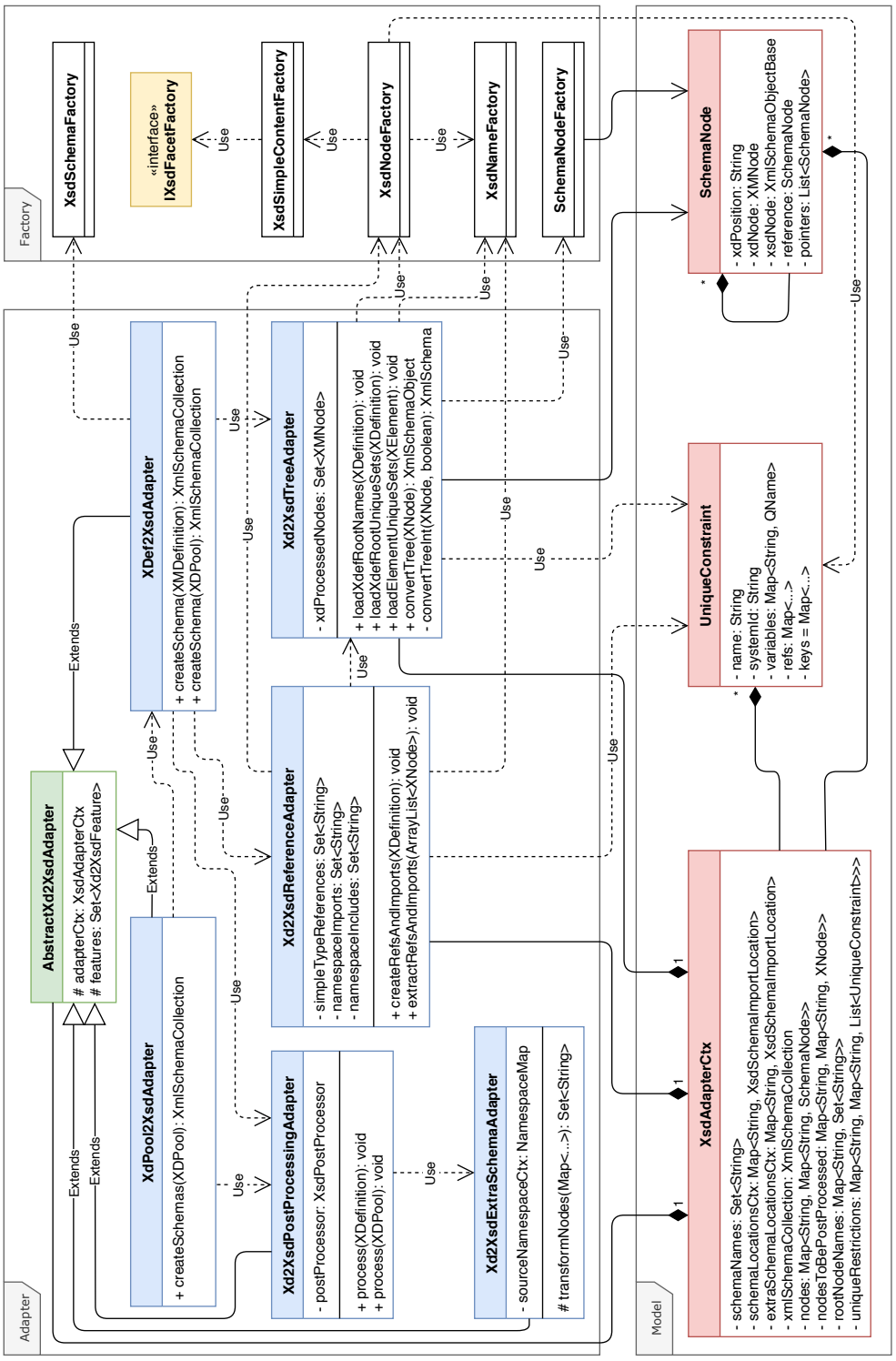
Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `algorithm` ..... zdrojové kódy implementace algoritmu
    - `impl` ..... zdrojové kódy implementace v rámci modulu Maven
    - `repository` ..... snapshot historie repozitáře knihovny X-definice
    - `xdef` ..... zdrojové kódy implementace a knihovny X-definice
  - `diagrams` ..... doplňující UML diagramy tříd
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Smid_Tomas_2020.pdf` ..... text práce ve formátu PDF

Příloha C

Přiložené obrázky



Obrázek C.1: Zjednodušený výřez diagramu tříd pro transformaci X-definice

Příloha D

Přiložené ukázky

```
private XmlSchemaObject convertTreeInt(final XNode xNode,
                                       boolean topLevel) {
  if (!xdProcessedNodes.add(xNode)) {
    SchemaLogger.printP(LOG_DEBUG, TRANSFORMATION, xNode, "Already
        processed. This node should be reference definition");
    return null;
  }

  short xdElemKind = xNode.getKind();
  switch (xdElemKind) {
    case XNode.XMATTRIBUTE: {
      return createAttribute((XData) xNode, topLevel);
    }
    case XNode.XMTEXT: {
      return
          xsdFactory.createSimpleContentWithExtension((XData)xNode);
    }
    case XNode.XMELEMENT: {
      return createElement((XElement) xNode, topLevel);
    }
    case XNode.XMSELECTOR_END:
       return null;
    case XNode.XMSEQUENCE:
    case XNode.XMMIXED:
    case XNode.XMCHOICE:
      return xsdFactory.createGroupParticle(xNode);
    case XNode.XMDEFINITION: {
      SchemaLogger.printP(LOG_WARN, TRANSFORMATION, xNode,
          "XDefinition node has to be only pre-processed!");
      return null;
    }
    default: {
      SchemaLogger.printP(LOG_WARN, TRANSFORMATION, xNode, "Unknown
          type of node. NodeType=" + xdElemKind);
    }
  }

  return null;
}
```

Ukázka D.1: Zdrojový kód metody *convertTreeInt* – rekurzivní transformace předpisu X-definice

```
private QName getRefQName(final XElement xElem) {
  final String refXPos = xElem.getReferencePos();
  final String xPos = xElem.getXDPosition();
  final String refSystemId =
      XsdNamespaceUtils.getSystemIdFromXPos(refXPos);
  final String refLocalName = XsdNameUtils.getReferenceName(refXPos);
  final String refNsPrefix =
      XsdNamespaceUtils.getReferenceNamespacePrefix(refXPos);

  if (XsdNamespaceUtils.isNodeInDifferentNamespace(xElem.getName(),
      xElem.getNSUri(), schema)) {
      final String nsUri = xElem.getNSUri();
      final String nsPrefix =
          schema.getNamespaceContext().getPrefix(nsUri);
      if (nsPrefix == null) {
          final XmlSchema refSchema =
              adapterCtx.findSchema(refSystemId, true, TRANSFORMATION);
          final String refNsUri =
              refSchema.getNamespaceContext().getNamespaceURI(refNsPrefix);
          if (XsdNamespaceUtils.isValidNsUri(refNsUri)) {
              XsdNamespaceUtils.addNamespaceToCtx((NamespaceMap)
                  schema.getNamespaceContext(), refNsPrefix, refNsUri,
                  refSystemId, POSTPROCESSING);
          }
      }
      return new QName(nsUri, xElem.getName());
  } else if
      (XsdNamespaceUtils.isRefInDifferentNamespacePrefix(refXPos,
      schema)) {
      final XmlSchema refSchema = adapterCtx.findSchema(refSystemId,
          true, TRANSFORMATION);
      final String nsUri =
          refSchema.getNamespaceContext().getNamespaceURI(refNsPrefix);
      return new QName(nsUri, refLocalName);
  } else if (XsdNamespaceUtils.isRefInDifferentSystem(refXPos, xPos))
      {
      return new QName(XSD_NAMESPACE_PREFIX_EMPTY, refLocalName);
  } else if (Xd2XsdUtils.isAnyElement(xElem)) {
      String anyLocalName = refLocalName;
      final int anyPos = anyLocalName.indexOf("$any/$any");
      if (anyPos != -1) {
          anyLocalName = anyLocalName.substring(0, anyPos);
      }
      return new QName(refNsPrefix, anyLocalName);
  }
  return new QName(refNsPrefix, refLocalName);
}
```

Ukázka D.2: Zdrojový kód metody *getRefQName* – získání kvalifikovaného jména z elementu X-definice

```
int lastSizeMap = schemasToResolve.size();
while (!schemasToResolve.isEmpty()) {
  Iterator<Map.Entry<String, XsdSchemaImportLocation>> itr =
      schemasToResolve.entrySet().iterator();
  while (itr.hasNext()) {
    final Map.Entry<String, XsdSchemaImportLocation> schemaToResolve
        = itr.next();
    final String schemaTargetNsUri = schemaToResolve.getKey();

    if (updatedNamespaces.contains(schemaTargetNsUri)) {
        itr.remove(); continue; }

    final Map<String, XNode> nodesInSchemaToResolve =
        allNodesToResolve.get(schemaTargetNsUri);

    if (nodesInSchemaToResolve != null) {
      final ArrayList<XNode> nodesToResolve = new
          ArrayList<XNode>(nodesInSchemaToResolve.values());
      final Iterator<XNode> itr2 = nodesToResolve.iterator();
      XNode n;
      while (itr2.hasNext()) {
          n = itr2.next();
          if (!sourceSystemId.equals(
              XsdNamespaceUtils.getSystemIdFromXPos(n.getXDPosition())))
                  { itr2.remove(); }
      }
      if (!nodesToResolve.isEmpty()) {
          final SchemaAdapter adapter = new
              SchemaAdapter(sourceXDefinition);
          adapter.setAdapterCtx(adapterCtx);
          adapter.createOrUpdateSchema(new NamespaceMap((HashMap)
              sourceNamespaceCtx.clone()), nodesToResolve,
              schemaTargetNsUri, schemaToResolve.getValue());
      }
      itr.remove();
    }
  }

  int currSchemasToResolve =
      adapterCtx.getExtraSchemaLocationsCtx().size();
  if (lastSizeMap < currSchemasToResolve) {
    schemasToResolve =
        (HashMap)((HashMap)adapterCtx.getExtraSchemaLocationsCtx()).clone();
  } else if (lastSizeMap <= schemasToResolve.size()) { break; }
  lastSizeMap = schemasToResolve.size();
}
```

Ukázka D.3: Vytah zdrojového kódu metody *transformNodes* – vytváření uzlů v *postprocessing* fázi algoritmu transformace X-definice