Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Využití moderních metod k tvorbě uživatelské dokumentace na základě Open API standardu

Diplomová práce

Autor: Bc. Václav Suchánek
Studijní obor: Informační management

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2019

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně s použitím uvedené literatury.

V Hradci Králové dne 19. 4. 2019 ………………………………….

Poděkování

Děkuji docentu Filipovi Malému za odborné vedení, aktivní spolupráci a cenné rady poskytnuté při vypracování této diplomové práce.

Anotace

Práce se zabývá automatickým generováním dokumentace, jejímž obsahem je Open API specifikace, tzn. obsahem jsou nabízené REST API služby/metody prostřednictvím daného serveru. Pro tuto práci je využit testovací vzorek Open API zvaný PetStore, který je nabízen v rámci projektu Swagger. Generovací proces je sestaven na základě testovacího vzorku. Daný konfigurační soubor Open API je v prvé řadě transformován do XML souboru, jenž je validním dokumentem vůči schématu DocBook DTD. Tento dokument je následně transformován v rámci XSLT do HTML podoby a v rámci XSL-FO do FO dokumentu. Dokument nesoucí formátovací objekty je posléze formátován pomocí formátovacího procesoru do podoby PDF dokumentu.

Annotation

Title: Utilizing Modern Methods to Create User Documentation Based on the Open API Standard

This thesis deals with automatically generated documentation describing Open API specifications, i.e. REST API services/methods provided by a server. The work uses a test sample of Open API specification called PetStore. This sample is freely available within the Swagger project and is processed in several steps, as described in the thesis. Firstly, the Open API configuration file is transformed into a XML document, which is valid for the DocBook DTD schema. In following step, the XML document is transformed into HTML (via the application of XSLT) and into a FO document (via the application of XSL-FO). In the end, the FO document (containing formatting objects) is formatted via the formatting processor and the final PDF document is generated.

Obsah

Úvod ... 1

1 REST API ... 3

1.1 Metody ... 3

1.2 URI Formát ... 4

1.3 Autentifikace ... 5

2 Open API ... 6

2.1 Základní objekty ... 6

2.2 Nástroje Open Api ... 8

Swagger Editor ... 8

OpenAPI.NET ... 8

VSCode/openapi-lint ... 8

Atom/linter-swagger ... 8

3 XML ... 9

3.1 Historie XML ... 9

3.2 Základy psaní XML dokumentu ... 11

Deklarace XML ... 11

Kořenový element ... 11

Elementy a atributy ... 12

Entity ... 12

Sekce CDATA ... 13

Procesní instrukce ... 14

Komentáře ... 14

3.3 DTD ... 14

Tvorba DTD ... 15

Validace DTD ... 23

3.4 XML Schéma ... 24

Vytvoření XML Schéma ... 25

Jednoduché a komplexní typy ... 26

3.5 Jmenné prostory ... 37

Použití v XML ... 37

Použití v XML Schéma ... 38

3.6 XPath ... 40

4 Docbook DTD standard ... 41

4.1 Editory XML ... 42
Emacs ... 42
oXygen ... 42
XMLMind XML Editor ... 42
5 XSL ... 43
5.1 XSLT ... 43
Základní použití ... 43
Procesory XSLT ... 45
5.2 XSL-FO ... 45
Základní použití ... 45
Procesory XSL-FO ... 48
6 Ukázka generovacího workflow ... 49
6.1 Swagger procesor ... 49
6.2 Transformace XSLT ... 50
6.3 Transformace XSL-FO ... 50
6.4 Formátování XSL-FO ... 50
7 Výsledky ... 51
8 Závěr ... 54
9 Seznam použité literatury ... 56
10 Seznam tabulek ... 58
11 Seznam obrázků ... 59
12 Seznam zdrojových kódů ... 60

Úvod

V této práci je pojednáváno o technologiích, pomocí kterých je snahou docílit automatizovaného procesu. Z nadpisu této práce je zřejmé, že se jedná o vytváření dokumentace, jejíž obsahem je Open API, tzn. nabízené REST API služby/metody daným serverem. Technologií, pomocí kterých lze docílit takového procesu, existuje celá řada, proto byly použity pouze základní často používané, osvědčené nebo ty, u kterých probíhá vývoj dodnes. Některé použité technologie se mohou považovat za zastaralé, avšak standardy, které např. pocházejí od konsorcia W3C nebo z projektu Apache se na trhu podílejí desítky let a stále jsou aktivně využívány, neboť se na jednu stranu dají označit jako „zaběhlé“ a na druhou stranu málokdo má potřebu vyvíjet technologie lepší.

V první části této práce je popsáno REST API. Jsou zde rozebrány příslušné metody, kterých může uživatel využívat. Poté je naznačen formát, pomocí kterého se dají zmíněné metody volat. A na závěr jsou naznačeny kroky autentifikace, neboli způsoby, pomocí kterých můžou být uživateli přiděleny práva k používání daných metod.

Dále je rozebrána specifikace Open API, která vznikla na základě REST API. Jsou zde popsány základní objekty, kde je možné porovnat rozdíly mezi Open API a REST API (resp. přidanou hodnotou Open API). Poté následují vybrané nástroje (editory, validátory) Open API, kde Swagger editor je označen jako klíčový, neboť pochází od autorské společnosti Open API specifikace.

Následující kapitola pojednává o XML. Tato kapitola je rozsáhlejší, neboť problematika XML a příslušné schéma jsou klíčové z pohledu primárního dokumentu, ve kterém se nachází zdroj veškeré dokumentace. Je zde nahlédnuto do historie o vývoji XML jako takového. Dále následují základy psaní XML dokumentu bez aplikovaného schématu (syntaxe). Poté je zde rozebrána definice a aplikace XML dokumentu, jenž je zapsána formou DTD. XML Schema DTD je zde popsáno jako základní schéma, neboť z něho vycházejí ostatní XML schémata. V rámci schémat je popsáno dané XML schéma s názvem XML Schema, které je aplikovatelné do praxe (narozdíl od DTD). V rámci XML ještě nemohou být opomenuty jmenné prostory, neboť ty zasahují do problematiky XML ve všech úrovních. A naposled je zde naznačena problematika XPath, jenž se jeví jako velice šikovná technologie pro práci s XML (navíc je nezbytná pro práci s XSL).

Následuje kapitola DocBook DTD, neboli definované XML schéma pro psaní knih. Tato problematika je velice rozsáhlá, avšak v této práci je nastíněna pouze okrajově, neboť použití se odvíjí od problematik předešlých a použití jednotlivých elementů (ze sémantického hlediska) není nutné rozebírat, protože samotná specifikace obsahuje bohatý popis jednotlivých prvků. Nastíněny jsou zde i některé editory, jenž zahrnují podporu pro toto schéma.

Poslední rozebraná problematika je o XSL, zahrnujíc XSLT a XSL-FO. Tyto technologie představují především procesory, které jsou zodpovědné za transformaci jednoho

formátu do druhého a případné formátování. Je zde naznačeno základní použití v rámci vlastně vytvořených šablon pro transformaci a to pro transformaci do podoby HTML a do podoby FO dokumentu. Následují vybrané procesory.

V závěrečné části této práce je rozebrána ukázková aplikace, jejíž účelem je zahrnout všechny výše uvedené technologie a vytvořit tak automatizovaný proces pro generování Open API dokumentace.

1 REST API

Metodologie zvaná Representational state transfer (REST) představuje architekturu rozhraní. Jinými slovy se jedná o zprostředkování webové služby. Tato služba umožňuje obousměrnou komunikaci založenou na architektuře client-server využívající HTTP volání. Rozhraní REST API má jasně definovanou strukturu ve 3 základních bodech. Prvním bodem jsou operace (metody), které umožňují číst nebo manipulovat s daty. O této problematice bude více pojednáno v kapitole Metody. Druhým bodem je HTTP response nesoucí status (identifikátor) a obsah (popis, HTTP hlavičky). Posledním bodem jsou URIs. Tento pojem představuje unikátní označení pro zdrojová data, které jsou součástí REST API. [1] [2]

1.1 Metody

Datové operace korespondují s CRUD operacemi. Tato zkratka představuje 4 základní operace nad daty v oblasti programování nebo databází. Jedná se o vytvoření, čtení, změnění nebo smazání. V případě REST API bylo snahou metodu *UPDATE* rozšířit a dále specifikovat. Proto namísto této metody vznikly další 2 metody zvané: *PUT* a *PATCH*. [1]

Název metody	Operace	Response status (kolekce objektů)	Response status (konkrétní objekt)
POST	Vytvoření	201 (kolekce id)	404 (nenalezeno), 409 (již existující objekt)
GET	Čtení	200 (kolekce objektů)	200 (objekt), 404 (nenalezeno)
PUT	Nahrazení	405 (nepovolená metoda)	200 (objekt), 204 (prázdný obsah), 404 (nenalezeno)
PATCH	Modifikace	405 (nepovolená metoda)	200 (objekt), 204 (prázdný obsah), 404 (nenalezeno)
DELETE	Smazání	405 (nepovolená metoda)	200 (objekt), 404 (nenalezeno)

Tabulka 1: Popis REST API metod a jejich doporučených návratových hodnot. [1] [upraveno]

V předešlé tabulce jsou znázorněny response statusy daných metod při zpracování kolekce objektů nebo objektu samotného. Pokud je status vyšší než 400, tak zpravidla bývá obsahem dané response zpráva o neúspěšné operaci. V úspěšném případě bývá v těle response navrácený objekt nebo kolekce objektů. Objekty jsou zpravidla formátovány jako JSON nebo jako XML, záleží na specifikaci. [1]

1.2 URI Formát

Součástí HTTP volání REST API metod (request) je URL adresa volané metody. V URL se nachází doména serveru, který nabízí REST API službu. Dále následuje cesta dané metody (URI), jejíž součástí bývají dané parametry. [3]

Následující ukázka znázorňuje obecný tvar REST API metody logistického cloudu (aplikace).

```
http://host/logisticRestApi/resources/{version}/{resourceName}
```

Zdrojový kód 1: Ukázka obecného tvaru REST API metody logistického cloudu. [3 str. 11]

Po zavolání metody (resp. její URI) je očekávána návratová hodnota (response). Již bylo uvedeno, že návratová hodnota může být pouhá zpráva, a to v neúspěšném requestu nebo u metod, kde není očekáván objekt nebo kolekce objektů, nebo struktura ve formátu JSON nebo XML. Je možné použít i jiné formáty, záleží na specifikaci. Při deklarování návratových hodnot daných metod je doporučeno nabízet oba formáty současně. [3] [1]

V nadcházející ukázce je uveden způsob volání REST API metody (*GET*) v prostředí Lightning platformy (služba), kde objektem requestu jsou informace ohledně verze používané Salesforce infrastruktury. Ihned poté následuje další ukázka s návratovou hodnotou (JSON) volané metody.

```
Curl https://yourInstance.salesforce.com/services/data
```

Zdrojový kód 2: Ukázka způsobu volání REST API metody v prostředí Lightning platform. [2 str. 2]

```
[
    {
        „version“:“20.0“,
        „url“:“/services/data/v20.0“,
        „label“:“Winter ,11“
    }
    …
]
```

Zdrojový kód 3: Ukázka návratové metody volané REST API metody v prostředí Lightning platform. [2 str. 2]

1.3 Autentifikace

Již bylo uvedeno, jakým způsobem je možné provést HTTP volání, které operace je možné využít a jaký tvar mají návratové hodnoty. Avšak ne vždy jsou REST API služby nabízeny volně (komukoliv). V těchto případech je vyžadováno oprávnění k užívání nabízených REST API služeb. Způsobů autentifikace může být mnoho. V případě logistického cloudu (aplikace společnosti Oracle) je využíváno 2 základních metod. První známou metodou je HTTP Basic Authentication. Při užití této metody se v hlavičce HTTP requestů uvádí uživatelské ID a příslušné heslo, které jsou kódované pomocí base64. Poté je následně přijatý request (ze strany serveru) přijat či nikoliv. Druhou metodou je využití SSO (Single Sign-On). Tato metoda není metodou autentifikace v pravém slova smyslu, neboť zde probíhá pouze dotazování servery/aplikacemi třetích stran na to, zda již byl uživatel autentifikován. Existují zde i jiné možnosti k ověření daného oprávnění, např. společnost Salesforce ve své službě zvané Lightning platform využívá protokolu OAuth. Tento protokol je zde zprostředkován skrze příslušné endpointy, pomocí kterých se může autorizovat k užívání daných služeb na daném serveru. [3] [2]

2 Open API

Na základě uvedeného REST API bylo snahou vytvořit jednoduše čitelný standard/formát jak z pohledu strojů, tak z pohledu lidí. Vznikla specifikace zvaná: The OpenAPI Specification (OAS), jenž představuje jednotnou specifikaci popisující REST API metody. Název Open API bývá často zaměňován za Swagger API. Z historického hlediska při vývoji této technologie, byl prvotní projekt pojmenován jako: The Swagger API project. Později tento projekt dosahoval díky své flexibilitě značných úspěchů a vzbudil zájem u velkých společností. Později byla Swagger specifikace přejmenována na Open API specifikaci. Tímto přejmenováním specifikace nebyla změněna, pouze začala být nabízena jako open-source řešení prostřednictvím GitHub úložiště. [4] [5]

2.1 Základní objekty

Open API je definováno ve formátu JSON nebo YAML. JSON představuje objektový tvar zápisu využívaný napříč různými platformami. Je především založen na JavaScriptu. Jazyk YAML zase slouží pro serializaci dat složité struktury (nabývá většího významu). V notaci JSON je syntaxe založena na stylu: „klíč-hodnota“, kdežto v jazyce YAML je datová hierarchie řešena indentací. [4] [6]

Název objektu	Popis
Metadata	Sekce metadat zahrnuje informace o dané specifikaci. Jedná se především o verzi Open API. V současné době je k dispozici verze „3.0.0“, je možné použít i starší verzi: „2.0“. Je zde také k dispozici sekce „info“, ve které se definuje nadpis a krátký popis k specifikaci jako celku.
Servery	Všechny cesty v Open API jsou relativní vůči URL specifikované v tomto objektu. Zde se definují jednotlivé base URL příslušných serverů, které mohou být využívány.
Cesty	Cesty označují relativní URL (endpoint), které mohou být použity vůči definovaným base URLs. Cesty jsou doplněné o danou REST metodu, která zajišťuje provedení dané akce. Příkladem může být relativní cesta: „/users“ doplněná o metodu *GET*. Výsledkem volání této cesty by poté byl seznam daných uživatelů.
Parametry	Metody daných cest mohou být dále specifikovány díky parametrům. Dodatečný parametr může být specifikován v URL cestě nebo v hlavičce HTTP requestu. Specifikování daného parametru v cestě může vypadat následovně: „/user/{userId}“. Výsledkem volání této cesty by poté byl uživatel s daným ID. U tohoto parametru je potřeba dále specifikovat, kde se daný parametr nachází (cesta nebo hlavička requestu), zda je povinný nebo volitelný, případně popis a tvar těla (struktura) requestu.

Request Body	V starší verzi Open API bylo Request Body součástí parametrů. V novější verzi se uvádí na stejné úrovni s parametry. Tato sekce slouží k definování metod jako: *POST*, *PUT* nebo *PATCH*. U těchto metod bývají uvedeny povinné parametry, které jsou předmětem deklarace Request Body.
Responses	Již bylo uvedeno, REST technologie je založená na client-server architektuře, ve které je využíváno způsobu komunikace request a response. K definování response slouží právě tato sekce. Sekce zahrnuje status (pozitivní jako 200 nebo negativní jako 404), případný popis a případné tělo (strukturu) návratové hodnoty v daném formátu (např. XML nebo JSON).

Vstupní a výstupní modely	Tyto modely jsou označovány jako schémata. Jde o globální deklaraci znovu použitelné struktury, která je předmětem volání daných metod. Toto schéma bývá často referencováno z deklarace responses. Mohlo by se očekávat, že daná response s deklarovaným statusem: „200“ by vracela definovanou JSON strukturu. Příkladem může být deklarovaná cesta: „/users/{userId}“, jejíž response (se statusem „200“) má referencované schéma, které je znázorněno v následující ukázce:
	`components:`
	`  schemas:`
	`    User:`
	`      properties:`
	`        id:`
	`          type: integer`
	`        name:`
	`          type: string`
	`      # Both properties are required`
	`      required:`
	`          -  Id`
	`          -  name`
	Zdrojový kód 3: Ukázka deklarace globálního schématu dle Open API specifikace v jazyce YAML. [52]

Autentifikace	V této sekci jsou deklarovány využívané autentifikační metody v rámci dané specifikace. Můžou zde být uvedeny různé metody autentifikace jako basic HTTP autentification, API key (zasílaný skrze hlavičku requestu, skrze parametr nebo vedený v rámci cookies), OAuth 2 nebo OpenID Connect Discovery.

Tabulka 2: Popis základní struktury Open API specifikace.

2.2 Nástroje Open Api

Pro práci s Open API by bylo vhodné představit možné nástroje, kterými je možné danou specifikaci editovat, validovat, dokonce i testovat. Dostupné jsou různé konvertory (např. konvertování verzí Open API), nástroje generující dokumentaci (např. transformace Open API do podoby HTML), různé textové a visuální editory, API pro implementaci Open API na straně serveru, různé parsery nebo validátory. [7]

Swagger Editor

Jak už napovídá název, jedná se o open-source editor společnosti, která stála za vznikem Open API specifikace (SmartBear Software). V současné době podporuje Open API verze 2.0 a 3.0.0. Tento editor poskytuje možnosti validace, editace, testování a generování dokumentace. Editor nabízí interaktivní prostředí, ve kterém je možné přímo editovat Open API specifikaci a přitom pozorovat generovanou dokumentaci v HTML formátu. Součástí je i validátor. Interaktivní prostředí je možné naleznout zde: https://editor.swagger.io/. Součástí jsou i testovací data, volně nabízené na následující adrese: https://petstore.swagger.io/v2/swagger.json, které představují smyšlený obchod s domácími mazlíčky. Editor dále nabízí možnosti automatické generování serverů nebo clientů různých platforem. [8]

OpenAPI.NET

Tento nástroj představuje parser, jehož úlohou je reprezentovat Open API model jako .NET objekty. Nástroj je volně dostupný, je možné s ním zpracovat Open API zapsané jak v JSON nebo v YAML a je zde podpora pro Open API 2.0 i 3.0.0 specifikaci. [9]

VSCode/openapi-lint

Společnost Microsoft nabízí source-code editor zvaný Visual Studio Code. Tento editor zahrnuje rozšíření s názvem: openapi-lint. Jedná se o možnost validace a editace Open API (2.0 nebo 3.0.0) specifikace v YAML nebo v JSONu. Součástí je i konvertor Open API verzí z 2.0 do 3.0.0. [10]

Atom/linter-swagger

Atom představuje source-code editor, který je vyvíjen prostřednictvím služby GitHub. Do editoru je možné přidat plugin linter-swagger, pomocí kterého je dále možné validovat a editovat Open API specifikaci. Podobně jako u předchozích nástrojů, je zde podpora pro Open API verze 2.0 a 3.0.0 a oba způsoby zápisu (JSON a YAML). Editor je volně dostupný a je možné ho využít v rámci různých operačních systémů. [11]

3 XML

Jazyk XML (eXtensible Markup Language) je jazyk pro vytváření dokumentů s jednoduchým obsahem. Byl vyvinut konsorciem W3C (World Wide Web Consortium), jehož hlavní úlohou bylo překonání omezujících prvků v jazyce HTML (Hypertext Markup Language). Pomocí tohoto jazyka je možné kombinovat text a další média (obrázky, hudbu nebo videa) v rámci obsahu elektronických dokumentů. Struktura XML dokumentu je sestavena za pomoci značek, které jsou oproti značkám v jazyce HTML volně vázané. Za více striktní věc je považováno umístění a tvar značek, neboli XML dokument musí být správně formulován. Správnou formulaci zajišťuje specifikace popisu dokumentu DTD (Document Type Definition) a validující parser. Termín parser představuje tzv. analyzátor nebo také procesor XML, který porovnává příslušný dokument a jeho pravidla psaná v DTD. Proto se XML považuje za metajazyk, neboť slouží k popisu jazyků ostatních. [12] [13] [14]

3.1 Historie XML

Počátkem 60. let 20 stol. začala vznikat koncepce obecného značkování jako reakce na postupný vývoj internetu. V té době byl veliký nátlak na vynalezení univerzálního standardu pro výměnu dat. Postupem času byl vyvinut jazyk SGML (Standard Generalized Markup Language), na jehož vývoji se podílelo několik zaměstnanců společnosti IBM a řádově stovky lidí po celém světě. Tento jazyk byl přijat v roce 1986 jako standard ISO č. 8879. SGML byl velice robustný jazyk a jeho specifikace byla příliš pokročilá. Neexistoval téměř žádný program, který by dokázal tento jazyk implementovat. Proto byly často využívány pouhé podmnožiny tohoto jazyka, avšak podmnožiny byly často nekompatibilní, programy vykazovaly vysokou chybovost a implementace SGML byla finančně náročná. Mnoho expertů poptávalo vyvinutí zjednodušené verze SGML. [13] [12]

Postupem času se stal Internet populárním médiem pro zisk a výměnu dat a informací. Bylo zapotřebí jazyka, který by zajistil propojení mezi zařízeními ve společné síti a také byla požadována možnost základního formátování. Objevil se jazyk HTML jakožto jedna z aplikací jazyka SGML. HTML sloužil pro prezentaci prostřednictvím webových stránek, avšak uměl pouze prezentovat dokumenty nebo data a postrádal principy jazyka SGML. HTML měl pevně danou syntaxi, omezenou množinu tagů a nebyla zde možnost jazyk rozšířit o vlastnoručně vytvořené tagy. V roce 1996 se začala vyvíjet zjednodušená verze jazyka SGML a v roce 1998 byla vydána první verze jazyka XML. [12] [13]

Poté byly vyvinuty jmenné prostory, které zajišťovaly bezkonfliktnost různých dokumentů. Mohlo se stát, že různé XML dokumenty se spojovaly a tudíž nesly společný obsah. Avšak v této se situaci mohl objevit element nebo elementy, které byly stejně pojmenované, ale lišily se svým významem. Jak uvádí E. Castro [15], jeden dokument může používat element „zdroj“ jako určení zdroje na webové stránce a v druhém dokumentu může tento element vyjadřovat určení hlavního pramene řeky. V tomto okamžiku se vyplatí použití tzv. XML *namespace*, neboli soubor názvů elementů, které spolu souvisejí. [13] [15]

V tento okamžik byl k dispozici XML dokument, který mohl nést různorodé informace. V následující fázi se vývoj začal zaměřovat na prezentaci dat. Objevil se jazyk XSL (eXtensible Stylesheet Language), který sloužil k transformaci XML dokumentů. XSL se brzy rozdělil na 2 podmnožiny (aplikace). Jednou z nich byl XSLT (Transformation), který sloužil k transformaci XML dokumentu do dokumentu jiného. Často to bývala HTML struktura, která byla doprovázena CSS (Cascading Style Sheets), jenž zajišťoval stylování dokumentu. Druhou aplikací byl XSL-FO (XSL Formatting Objects), který sloužil k tvorbě tištěných stránek. Výstupem této podmnožiny býval obvykle dokument s formátem PDF (Portable Document Format). [13] [14]

Dále bylo pro práci s XML dokumenty nezbytně nutné linkování. Tento pojem znamená možnost propojení jednoho nebo více dokumentů do hypertextové sítě. Tuto možnost přinesl odkazovací jazyk XLL (eXtensible Linking Language). Tento jazyk se postupem času rozdělil do dvou samostatných standardů. Jednalo se o standardy XLink a XPointer. XLink deklaruje v DTD definici linkovacích elementů. Linky mohou nést atributy, které odpovídají standardu XLink a linky mohou být dvojího typu – jednoduché, které explicitně propojují jeden zdroj a jednu destinaci, a rozšířené, které mohou propojovat více zdrojů a jednu destinaci. K intervalové adresaci částí dokumentu XML slouží XPointer. Tento standard poskytuje přístup k elementům a jejich hodnotám, přitom nezáleží na jejich umístění. Avšak k identifikaci jednotlivých elementů byl XPointer příliš složitý a vzhledem k použitelnosti pro XSLT byl mnohdy nekompatibilní. Proto byla vytvořena XPath. Na rozdíl od XPointeru, XPath nepracuje s intervalovými částmi XML dokumentu, ale s výčtem jeho částí, které jsou označovány jako nodes. Jednotlivé nody mají několik typů, nejfrekventovanější jsou: element nodes, attribute nodes, text nodes apod. [13] [16]

V následující fázi bylo zapotřebí, aby aplikace napsané v jazycích jako: Java, JavaScript či C++ měly k dispozici rozhraní, skrze které by mohly přistupovat k obsahu XML dokumentu. Dokument byl ve většině případů interpretován jako DOM (Document Object Model), jenž se skládal z objektů. Tento model se stal univerzálním standardem, který představoval rozhraní, skrze které se mohlo přistupovat k HTML i XML dokumentům. [12] [13]

V další etapě bylo zapotřebí XML dokumenty zpracovat, načítat (v přiměřených dávkách) a validovat. Každý, kdo potřeboval zpracovat XML dokument, tak využil parser - vhodný pro aplikaci, se kterou pracoval. Parsery byly různého charakteru vytvořené pro účely objektově orientovaných jazyků (Java, Javascript, C++).

Každý dodavatel parseru vyvíjel vlastní API. Ale různá API byla většinou málo odlišná, neboť bylo zapotřebí stejných operací při zpracovávání XML dokumentů (načíst element, atribut, sečíst počet daných elementů apod.). Proto v roce 1998 diskuzní skupina XML-DEV vyvinula standard SAX (Simple API for XML). Na základě tohoto standardu postupně začala vznikat řada parserů hlavně pro Java aplikace. Standard se ujal a v roce 2000 byl představen SAX2. [12] [13]

Následně se základní specifikace XML začala obohacovat o různé rozšíření. Mezi ně se může řadit: XFragment – pro zlegalizování částí dokumentu, které by jinak nebyly označeny jako validní. XML Schemas, neboli XSD (XML Schema Definition), slouží jako rozšíření pro DTD – tedy slouží pro popis povolených prvků v dokumentu. XHTML (eXtensible Hypertext Markup Language) se využívá jako redefinice jazyka HTML, která může posloužit aplikaci XML. XQL (XML Query Language) – pro vyhledávání elementů v XML při stanovených kritériích. Canonical XML představuje algoritmus, který dokáže porovnávat XML dokumenty mezi sebou. Je zde možné ignorovat nepodstatné detaily jako např. rozdílnost v jednoduchých a dvojitých uvozovek, jenž obalují hodnoty atributů. Mezi rozšíření XML specifikace patří i XML Signatures, které jsou využívané pro digitální podpisy v XML dokumentech. [13]

3.2 Základy psaní XML dokumentu

Je vhodné uvést, že XML dokument je textový dokument a tudíž neobsahuje binární data. Proto je možné s XML dokumenty pracovat za pomoci libovolných textových editorů nebo programů, které dokáží načíst textové soubory. [13]

Deklarace XML

Soubor, který nese XML obsah byl měl mít koncovku *.xml* a měl by zahrnovat deklaraci XML dokumentu. Deklarace se zapisuje jako procesní instrukce (více v kapitole: Procesní instrukce) a nese číslo verze jazyka XML.
Deklarace může být zapsána v tomto tvaru: `<?xml version="1.0" ?>`. [13] [15]

Kořenový element

Je nutné, aby XML dokument zahrnoval tzv. kořenový element. Tento element se v dokumentu nachází pouze jednou a je párový (více v následující kapitole). Tento element je rodičem všech ostatních elementů. [14] [15]

Správné použití kořenového elementu je znázorněno v následující ukázce:

```
<?xml version="1.0"?>
<položka>
   <jméno>Jan Moudrý</jméno>
   <email href="mailto:jan.moudry@nejakavelkafirma.cz"/>
</položka>
```

Zdrojový kód 4: Ukázka správného použití kořenového elementu. [14 str. 49]

Elementy a atributy

Elementy v XML mají obdobné použití jako v jazyce HTML. Elementy mohou být párové nebo nepárové. Párové jsou specifické počátečním a koncovým tagem. Rozdílem je zpětné lomítko, které se přidává před název tagu. Příkladem počátečního tagu může být: *<calendar>* a příslušným koncovým tagem bude poté: *</calendar>*. Nepárové elementy se vyznačují tím, že nezahrnují obsah datového proudu. Nepárový element lze vytvořit pomocí zpětného lomítka, které se přidá před koncový znak tagu. Příkladem nepárového tagu může být: *<calendar/>*. [13]

Již zde byl zmíněn pojem: datový proud. Tento pojem představuje množinu obsahu, která je obklopena párovými tagy. Pomocí datového proudu je XML schopno identifikovat jednotlivé objekty, které jsou zahrnuty v jeho obsahu. Pokud je uveden např. tento element: `<jméno>Scarborough</jméno>`, tak datovým proudem tohoto elementu je text: Scarborough. [12]

Elementy často tvoří hierarchii. V rámci hierarchie se poté používají následující pojmy: rodič, potomek, sourozenec a další. Příkladem hierarchie může být struktura knih. V knize se obvykle nacházejí kapitoly a podkapitoly. Kniha jako celek představuje rodiče. Kapitola představuje potomka, která může mít sourozence – další kapitoly. Je možné, že rodič má více potomků, v nichž mohou být zanořeni další potomci. V této situaci se dá hovořit o přímém/nepřímém potomku/rodiči. Skupina nepřímých potomků se označuje jako potomci a skupina nepřímých rodičů se označuje jako předci. Vzhledem k zmíněným knihám, uvažuje-li se kapitola, tak její potomci budou jednotlivé podkapitoly a jejich odstavce, a součástí předků by mohla být samotná kniha a kolekce. [12]

Pokud je zapotřebí blíže specifikovat obsah daného elementu, tak lze využít atributů. Atributy se vkládají za název tagu odděleného mezerou. Atribut nese název a hodnotu. Hodnota se uzavírá pomocí uvozovek a spojovacím znakem hodnoty a názvu je „=“. Např. pokud by bylo žádané, aby daný element měl explicitně specifikovaný jazyk, element by mohl zaujmout následující tvar: `<name language=“English“>Tiger</name>`. Množina atributů, které může element nést je omezena. Omezení je deklarováno za pomoci DTD, kde je specifikována množina názvů atributů příslušných elementů a jejich hodnotových typů. Hodnota atributu je ve výchozím tvaru považována za textový řetězec a je možné použít různé datové typy. Atributy mohou zahrnovat výchozí hodnoty – pokud uživatel neuvede hodnotu atributu, tak je poté použita hodnota výchozí. Výčtový atribut zase disponuje seznamem hodnot, které mohou být použity jako hodnota atributu. [12] [15]

Entity

V textovém obsahu elementů je občas zapotřebí použít symbol jako je např. levá ostrá závorka: „<“. Jak již bylo řečeno, tento symbol je interpretován jako počáteční symbol pro vytvoření XML elementu a nikoliv textu. V programovacích jazycích se symboly tohoto typu běžně escapují pomocí přídavného znaku, který se píše před daný symbol. Častým přídavným znakem bývá zpětné lomeno: „\“. V XML dokumentu se na místo přídavných znaků používají entity. Deklarace libovolných entit je definována v DTD (více o deklaraci

bude vysvětleno v kapitole o DTD). Entita se používá v následujícím tvaru: začátek entity symbolizuje znak ampersand: „&“, poté následuje název entity a konec entity uzavírá středník. XML nabízí několik základních entit. Příkladem jsou: < (levá ostrá závorka), & (ampersand), > (pravá ostrá závorka), " (dvojité rovné uvozovky) a ' (jednoduchá uvozovka). [13] [14]

Příklad použití XML entity je znázorněn v následující ukázce, kde autorovým záměrem je vložení apostrofu (jednoduché uvozovky) do alternativního popisu obrázku:

```
<image source=‘oreilly_koala3.gif‘ width=‘122‘ height=‘66‘
   alt=‘Powered by O&apos;Reilly Books‘
/>
```

Zdrojový kód 5: Příklad použití XML entity. [13 str. 19]

Sekce CDATA

V předchozí kapitole bylo nastíněno, jakým způsobem zacházet se nevalidními znaky. Nyní by mohlo být nežádoucí, kdyby se v textovém obsahu vyskytovalo XML entit příliš mnoho nebo by se projevila snaha o zachování bílého místa a zalamování řádků. Tento jev se často vyskytuje v případě, když se v XML dokumentu objeví ukázka zdrojového kódu. Např. by se mohlo jednat o ukázku jiné XML struktury. Pro tyto účely se používá sekce *CDATA*. Použití *CDATA* je následující: do textového obsahu se vloží počátek „<[CDATA[“, poté následuje obsah a sekce je zakončena pomocí: „]]>“. Pokud XML procesor prochází *CDATA*, tak ignoruje jakékoliv značkování. Výjimkou jsou značky pro *CDATA*, neboť se předpokládá, že *CDATA* nebudou obsahovat jiné *CDATA*. [13] [14]

V následujícím příkladu autor použil *CDATA*, aby vložil ukázku XML dokumentu do XML dokumentu:

```
<?xml version=“1.0“?>
<příklad>
<[CDATA[
<?xml version=“1.0“?>
<položka>
   <jméno>Jan Moudrý</jméno>
   <email href=“mailto:jan.moudry@nejakavelkafirma.cz“/>
</položka>
]]>
</příklad>
```

Zdrojový kód 6: Ukázka použití CDATA. [14 stránky 54-55]

Procesní instrukce

Může nastat situace, že bude zapotřebí vložit do XML dokumentu jiný jazyk. To se může jevit jako rozporuplné, co se týče XML standardu, ale v praxi je použití procesních instrukcí výhodné. Procesní instrukce často nesou informaci, která se předává aplikaci, jenž zpracovává XML strukturu. Informace může být psána v libovolném tvaru, neboť není zpracována pomocí XML procesoru. Instrukce se uzavírá pomocí znaků: „<?“ a „?>“, bezprostředně za prvním znakem následuje název procesní instrukce (někdy zvaný jako PITarget) a poté následuje textový obsah oddělený mezerou. Příkladem obsahu procesní instrukce může být meta-značka, která vypovídá o tom, zda se příslušný obsah má indexovat pro vyhledávací roboty v rámci HTML. Dále může být vložen PHP skript nebo reference na import CSS stylů, a mnoho dalšího. [12] [13] [14]

V následující ukázce je autorovým záměrem použití importovaných CSS stylů pro osobu. Neboli je cílem, aby aplikace, která zpracovává XML, aplikovala stylování pro osobu/osoby dle uvedeného CSS souboru.

```
<?xml-stylesheet href="person.css" type="text/css"?>
<person>
   Alan Turing
</person>
```

Zdrojový kód 7: Ukázka použití procesní instrukce. [13 str. 21]

Komentáře

Podobně, jak je tomu v jazyce HTML, je možné vkládat do XML komentáře pomocí uzavíracích znaků: “<!--” a “-->”. Komentáře je výhodné používat pro autory nebo spoluautory XML dokumentů, neboť za jejich pomoci mohou označovat místa v XML (např. kapitoly knihy), které je potřeba zeditovat/upravit/prověřit. Není výhodné používat komentáře pro předávání dat mezi dokumentem a aplikací. V případě některých XML procesorů dochází při zpracování XML k vymazání komentářů. Komentáře nejsou z hlediska XML považovány za elementy, tudíž mohou být použity na jakékoliv úrovni. Avšak je nelze vkládat do deklarace elementů nebo seznamu atributů, a žádný komentář nemůže obsahovat další komentáře. Příkladem jednoduchého komentáře může být: `<!-- Toto je komentář -->`. [12 str. 41] [13] [14]

3.3 DTD

V předešlých kapitolách bylo pospáno, jakým způsobem je možné používat jednotlivé prvky XML dokumentu (elementy, atributy apod.). Nyní bude vysvětleno, jak je možné tyto prvky nadefinovat. XML je sám o sobě velice volným jazykem a to nemusí být vždy výhodou. Aplikace, které dokumenty zpracovávají mohou vyžadovat striktnější způsob popisování obsahu. Pomocí DTD lze obsah XML dokumentu přizpůsobit tak, aby byl snadno uchopitelný ze strany jiných procesorů nebo aplikací. [12] [13]

Použití DTD může být výhodné především pro XML procesor, který může autora již při editaci dokumentu upozorňovat na nevalidní prvky, které je nutno opravit. Dále je možné, že XML editor může využít předem definované struktury elementů a již při zadávání XML prvků ze strany autora využít našeptávání. Procesor XML může využít DTD jako nápovědu, díky které rozpozná odsazení a samotný obsah. Pomocí DTD je možné specifikovat výchozí hodnoty atributů. To může v důsledku snížit velikosti dokumentů. [14]

Tvorba DTD

DTD je možné deklarovat interně nebo externě. V případě interní DTD, je obsah specifikován přímo v XML dokumentu. Uvádí se přímo za XML specifikací, kde je deklarace zahájena v následujícím tvaru: `<!DOCTYPE root [`. `root` symbolizuje kořenový element XML dokumentu. Následuje samotná deklarace DTD a blok je uzavřen pomocí: `]>`. [15]

V následující ukázce autor znázorňuje použití interní DTD deklarace v XML dokumentu, jehož obsah pojednává o ohrožených druzích zvířat:

```
<?xml version="1.0" ?>
<!DOCTYPE endangered_species [

]>

<endangered_species>
<animal>
```

Zdrojový kód 8: Ukázka použití interního DTD v XML dokumentu. [15 str. 36]

V případě externí DTD se v XML dokumentu specifikuje reference na externí dokument s koncovkou .dtd. Externí DTD se v praxi používají častěji a vyplácejí se v případě využití pro více obsahově souvisejících dokumentů. Podobně jako u interní DTD se reference na externí DTD specifikuje přímo za XML deklarací. Příkladem reference může být:

`<!DOCTYPE person SYSTEM „http://ibiblio.org/xml/dtds/person.dtd">`.

Z ukázky je zřejmé, že se jedná o absolutní adresu. Je možné použít i adresu relativní, za předpokladu že se dokument nachází na stejném uložišti jako specifikace DTD. Pokud se DTD i XML nacházejí ve stejném adresáři, poté by reference mohla mít následující tvar: `<!DOCTYPE person SYSTEM „person.dtd">`. [13] [15]

DTD může nabývat dvojí podoby. První podoba se označuje jako datově orientovaná a druhá se orientuje na sdělení. Datově orientované DTD je velice striktní a přímočaré. Struktura se sestavuje směrem „shora dolů“ a bývá jednodušší tuto strukturu pochopit. Tento způsob zápisu je přívětivý pro aplikace, které zpracovávají XML dokument. Druhá podoba může nést název: DTD orientované na sdělení. V tomto případě se jedná o volnější způsob zápisu pravidel, kde autor nebývá tolik omezen jako u datově orientovaného DTD. Struktura se sestavuje směrem „odspoda nahoru“, tedy se postupuje od definování nejmenších elementů a pokračuje se k největším. Tento styl DTD se poté

vyznačuje větší složitostí, rozmanitostí, a proto jsou složitější na pochopení. Příkladem tohoto volného stylu může být psaní životopisů. [13]

Deklarace elementu

Prostřednictvím DTD je možné nadefinovat pravidlo psaní XML elementu následujícím způsobem. Obecný tvar pravidla je:
<!ELEMENT název_elementu (kontextový model)>. Názvem elementu může být jakýkoliv název splňující pravidla XML. Dále kontextový model symbolizuje pravidlo nebo pravidla daného elementu. Jedná se o pravidlo sekvenční, které charakterizuje množinu potomků, kteří mohou nést další vlastnosti ohledně jejich omezení. Více o omezení potomků je k nalezení v kapitole: Omezení potomků. Je důležité definovat „správné" pořadí prvků v kontextovém modelu. Pořadí je striktně specifikováno a když by použití neodpovídalo dané definici, tak by se XML dokument stal nevalidním. [13] [16]

Deklarace atributu

Deklarace atributu se zásadně liší od deklarace a elementu v tom, že každý atribut nebo výčet atributů je definován pro specifický element, resp. název elementu. Obecná definice výčtu atributů je následující:
<!ATTLIST název_elementu definice_atributu>. Název elementu specifikuje unikátní název elementu, ke kterému je výčet atributů vázán. Jinými slovy, u daného elementu je možné použít dané atributy. Definice atributu se skládá z dalších 3 vlastností. Obecný tvar definice atributu je: „název_atributu typ_atributu implicitní_hodnota". První z nich je název atributu. Název atributu tvoří platný název atributu v rámci XML a je unikátní v rámci dané definice elementu. Druhá a třetí vlastnost budou popsány v následujících podkapitolách. [13] [15] [16]

V následující ukázce je znázorněna deklarace atributu charakterizující rok daného elementu: populace. Jsou zde povoleny pouze 2 hodnoty a tento atribut je povinný.

```
<!ELEMENT population (#PCDATA)>
<!ATTLIST population year (1999|2000) #REQUIRED>
```

Zdrojový kód 9: Ukázka definice atributu pro daný element. [15 str. 50]

Typy atributu

Následuje tabulka s přehledem všech typů, které mohou být pro daný atribut definovány, a jejich krátký popis.

Typ atributu	Popis
CDATA	Tento typ představuje libovolný textový řetězec, který splňuje obecnou deklaraci XML dokumentu. Jedná se o nejfrekventovanější použití typu atributu. V praxi jsou často *CDATA* blíže specifikovány pomocí XML schémat. Obvykle poté *CDATA* nesou definici pro ceny, adresy nebo URI.
NMTOKEN	Jedná se o tzv. názvový symbol, který představuje XML název. Jsou zde však patřičné výjimky. Rozdílnost může spočívat v tom, že XML název musí začínat abecedním znakem nebo interpunkčním znaménkem a *NMTOKEN* nemusí (*NMTOKEN* může začínat číslicí). Stejně důležitým pravidlem pro oba prvky je to, že se v názvu nesmí vyskytovat prázdné místo.
NMTOKENS	Tento typ představuje rozšířený *NMTOKEN*. Rozšíření představuje možnost použití vícero hodnot specifikovaných jako *NMTOKEN*, oddělených prázdným místem. Následuje ukázka použití *NMTOKENS* pro specifikování barvy atributu foreground: <!ATTLIST body foreground NMTOKENS „Green, Yellow, Orange“ > **Zdrojový kód 10: Ukázka použití atributového typu NMTOKENS. [16 str. 65] [upraveno]**
VÝČET	*VÝČET* charakterizuje přesně definovaný výčet hodnot, které se mohou v atributu vyskytnout. Tyto hodnoty jsou omezeny obecnou XML definicí a hrají zde roli velká a malá písmena.
ID	Tento typ slouží pro deklarování unikátní hodnoty (nemohou být duplicitní) v rámci XML dokumentu. *ID* představuje XML název, proto nemůže začínat číslicí.

<table>
<tr><td>IDREF</td><td>Navazujícím typem na typ ID je IDREF, který slouží jako reference. Tato reference odkazuje na typ ID (platný XML název). Z hlediska kardinality jde o spojení 1:N, kde 1 typ ID může být cílem nekonečně mnoho referencí.
V následující ukázce je znázorněno použití typu IDREF, kde autor poukazuje na referenci mezi projektem a členem týmu prostřednictvím id_projektu:

<!ATTLIST osoba rodné_číslo ID #REQUIRED>
<!ATTLIST projekt id_projektu ID #REQUIRED>

<!ATTLIST člen_týmu osoba IDREF #REQUIRED>
<!ATTLIST přiřazeno id_projektu IDREF #REQUIRED>

Zdrojový kód 11: Ukázka použití atributového typu IDREF. [13 str. 44]</td></tr>
<tr><td>IDREFS</td><td>Tento typ představuje rozšířený IDREF. Rozšíření charakterizuje možnost použití vícero hodnot specifikovaných jako IDREF, oddělených prázdným místem.</td></tr>
<tr><td>ENTITY</td><td>Typ ENTITY deklaruje použití XML entity, která představuje často opakovaný textový řetězec, který je definován jako entita. Jedná se o entity neanalyzované. Více informací o entitách je zahrnuto v dalších kapitolách.</td></tr>
<tr><td>ENTITIES</td><td>Tento typ umožňuje použití vícero ENTITY typů najednou. Jednotlivé typy jsou oddělené prázdným místem.</td></tr>
<tr><td>NOTATION</td><td>Typ NOTATION představuje výčet definovaných notací v DTD. Tento typ je hodně podobný XML entitám a slouží ke specifikování typu obsahu. Notace se v praxi využívají takřka ojediněle.
Následující ukázka zahrnuje definici výčtu 4 notací pro daný atribut a definici notace samotné:

<!NOTATION png SYSTEM „image“png“>
<!ATTLIST image type NOTATION (gif|tiff|jpeg|png)
#REQUIRED>

Zdrojový kód 12: Ukázka použití atributového typu NOTATION. [13 str. 46]</td></tr>
</table>

Tabulka 3: Typy atributu s popisem. [13] [15] [16]

Typy výchozí deklarace atributu

Následující tabulka zahrnuje výčet všech dostupných výchozích deklarací atributu a jejich popis. Deklarace může být označena i jako implicitní hodnota atributu, která specifikuje dodatečné vlastnosti ohledně omezení daného atributu.

Typ výchozí deklarace	Popis
#IMPLIED	Tento typ uvádí atribut jako volitelný, proto atribut může být uveden a nebo nemusí.
#REQUIRED	Tento typ uvádí atribut jako povinný. Atribut musí vždy být součástí daného elementu.
#FIXED	Fixní hodnota charakterizuje neměnnou a konstantní hodnotu atributu. Atribut poté musí vždy obsahovat uvedenou hodnotu. Tento typ je vhodné využít v případě, kdy je nežádoucí změna hodnoty daného atributu ze strany uživatele.
hodnota	Tento typ se nespecifikuje jako klíčové slovo. Na místo klíčového slova je specifikována konkrétní hodnota textového řetězce uvedena v uvozovkách. Pokud daný atribut není specifikován, tak je posléze přidělen danému elementu s příslušnou výchozí hodnotou.

Tabulka 4: Typy výchozí deklarace atributu s popisem. [13] [16]

Deklarace obecných entit

XML procesor již má předem definovanou sadu vestavěných entit. (viz kapitola: Entity) Pokud je zapotřebí danou sadu rozšířit, tak je možné vytvořit entity vlastní. Entity se často využívají v oblastech, kde je potřeba vytvořit doprovodnou proměnnou pro často opakující se text, ve kterém se snadno může udělat gramatická chyba, nebo když je zapotřebí vložit do textu symbol, který je v rámci textového nodu nepovolen. Např. symbol: „<“. [12] [13] [15]

Následující ukázka zahrnuje definici entity s názvem: „myentity“, které přísluší textový obsah: „obsah entity ‚myentity‘“.

```
<!ENTITY myentity „obsah entity ‚myentity‘“>
```

Zdrojový kód 13: Ukázka definice entity. [12 str. 47]

Názvy entit mohou být v rámci DTD duplicitní. V případě duplicitních názvů je použita vždy první deklarovaná definice a ostatní definice jsou ignorovány. Název entit musí být platným XML názvem a není možné definovat výchozí hodnoty entit. Dále není možné pomocí výše uvedené ukázky definovat vzájemné definice entit. Ale je možné definovat entity jako zástupné konstanty pro názvy atributů. K takové definici slouží jiný druh entit (parametrické entity), o kterém se více informací nachází v následující kapitole. [12] [13]

Deklarace parametrických entit

Jak již bylo naznačeno v předchozí kapitole, parametrické entity jsou typem entit, které umožňují referencovat názvy atributů. V praxi se běžně stává, že je definována velká sada elementů, z nichž každý element zahrnuje často opakující se názvy atributů. Což může být nevýhodné, neboť když se např. změní název jednoho z daných atributů nebo do pomyslné sady atributů přibyde atribut nový, tak se tato změna musí projevit u všech souvisejících elementů. Proto je výhodné využít parametrické entity, které často opakující se sadu atributů mohou nahradit pomocí konstanty. Parametrická konstanta může být nadále modifikována na jednom místě. Parametrické entity jsou podobné entitám obecným, akorát namísto znaku „&" se používá znak „%", a parametrické entity je možné použít pouze uvnitř DTD. [12] [13] [15]

V následující ukázce je znázorněn vyřešený problém s často opakující se sadou atributů pomocí parametrické entity. Jsou vytvořeny 3 konstanty/kategorie vlastností, kterou jsou dále použity pro definici elementů symbolizujících obytné nemovitosti.

```
<!ENTITY % popisné_údaje „adresa, plocha, místnosti, koupelen>"
<!ENTITY % podmínky_pronájmu „nájem">
<!ENTITY % podmínky_prodeje „cena">

<!ELEMENT byt (%popisné_údaje;, %podmínky_pronájmu;)>
<!ELEMENT podnájem (%popisné_údaje;, %podmínky_pronájmu;)>
<!ELEMENT chata (%popisné_údaje;, %podmínky_prodeje;)>
<!ELEMENT nájemní_dům (%popisné_údaje;, %podmínky_prodeje;)>
<!ELEMENT rodinný_dům (%popisné_údaje;, %podmínky_prodeje;)>
```

Zdrojový kód 14: Ukázka definice a použití parametrické entity. [13 str. 52]

Sekvence potomků

Jak již bylo naznačeno, v kontextovém modelu se deklarují jednotlivé elementy, které jsou odděleny čárkou. Sekvence potomků uvádí, že pořadí, ve kterém je zápis potomků proveden, je relevantní. Pořadí zápisu elementů v kontextovém modelu musí být shodný s pořadím zápisu elementů v dané lokaci XML dokumentu. Obecný tvar kontextového modelu „(a, b, c)" udává, že na první pozici bude uveden element a, po kterém bude následovat element b a na konci bude uveden element c. [12] [13]

Příklad použití nesprávného pořadí elementů je znázorněn v následující ukázce. Z ukázky je zřejmé, že nebyla dodržena pravidla při psaní XML dokumentu a tudíž se dokument stává nevalidním.

```
<!ELEMENT name (first_name, last_name)>

<name>
   <last_name>Ciccone</last_name>
   <first_name>Madonna</first_name>
<name/>
```

Zdrojový kód 15: Ukázka nesprávného pořadí elementů. [13 str. 35]

Omezení potomků

V rámci definování kontextového modelu, tj. součást deklarace elementu, existují operátory, pomocí kterých je možné specifikovat dodatečné omezení. Daná omezení se většinou týkají nastavení volitelného nebo povinného elementu a dále jeho četnosti. Četnost není deklarována explicitně, pouze vyjadřuje, zda daný element může být použit právě jednou nebo vícekrát. Jsou k dispozici 3 následující symboly operátorů: „?“, „*“ a „+“, jejichž funkcionalita bude popsána v následujícím odstavci. [12] [13] [14]

Prvním nabízeným operátorem je: „?“. Tento operátor zajišťuje, že příslušný element bude volitelný. To znamená, že se element na deklarované pozici bude moct vyskytovat jednou nebo ani jednou. Příkladem může být deklarace elementu jméno. Takový element bude vždy zahrnovat křestní jméno a příjmení a bude moci a zároveň nebude muset zahrnovat jméno prostřední. Zápis do DTD této ukázky bude následující: `<!ELEMENT name (first_name, middle_name?, last_name)>`. [12] [13]

Operátor „+“ deklaruje příslušný element jako povinný. Tudíž se element musí vyskytnout na dané pozici právě jednou nebo se může vyskytnou vícekrát. Vhodným příkladem je definování elementu kniha, který musí zahrnovat alespoň 1 kapitolu, přičemž může obsahovat kapitol více. Deklarace tohoto příkladu bude následující: `<!ELEMENT book (chapter+)>`. [12] [13]

Poslední nabízený operátor je „*“. Tento operátor se dá také vyjádřit jako ekvivalentní kombinace 2 předešlých operátorů a to ve tvaru: „?+“, neboť vyjadřuje možnost výskytu žádného, jednoho nebo více elementů. Příkladem může být deklarace odborného článku, který může mít jednoho, žádného nebo více autorů. Zápis této ukázky bude následující: `<!ELEMENT article (author*)>`. [12] [13]

Výběr potomků

Výběr potomků je dodatečnou definicí kontextového modelu, která deklaruje výčet potomků, ze kterého je možné vybrat pouze jednoho. K této definici slouží operátor svislé čáry: „|“. V případě složitějších definic je možné využít seskupování v rámci kulatých závorek. Ukázkovým příkladem může být: `<!ELEMENT methodResponse (params | fault)>`. Příklad uvádí, že response metoda (element) musí obsahovat právě jednoho potomka a ten potomek musí být buď typu params nebo typu fault. [13]

Při použití výběru potomků je třeba brát ohledy na pravidla týkajících se pořadí jednotlivých elementů (viz sekce: Sekvence potomků). Dodržování pravidel je významné především při sestavování složitějších deklarací, kde může být použita kombinace sekvence potomků, výběru potomků a dokonce i omezení potomků. Při použití výběru potomků je důležité, aby výběrový operátor „|“ nebyl zkombinován se sekvenčním operátorem „,“ v rámci jednoho kontextového modelu. Deklarace tohoto typu by poté byla neplatná. Pokud v deklaraci elementu je vhodné použít výběrových i sekvenčních pravidel, poté musí být množina kontextového modelu, obsahující výběr potomků, vhodně definována pomocí kulatých závorek. Příkladem může být deklarace elementu

kruh, který nese údaj o jeho středu a musí nést též údaj o jeho poloměru nebo průměru. Pokud by se uživatel striktně držel zápisu, tak by poté vznikla následující deklarace: <!ELEMENT circle (center, radius | diameter)>, avšak tato deklarace je v rámci DTD neplatná. Neplatnost spočívá v tom, že v rámci modelu není zřejmé, zda element diameter je alternativním elementem k sekvenci elementů center a radius, nebo je alternativou pouze k elementu radius. V kontextovém modelu jsou použity oba dva typy operátorů, což způsobuje nevalidní část DTD, a k vyřešení tohoto problému je nutné rozdělit kontextový model na vnitřní a vnější pomocí kulatých závorek. Výsledkem bude deklarace v následujícím tvaru: <!ELEMENT circle (center, (radius | diameter))>, jenž je deklarací validní a je vhodná k použití v rámci uvedeného příkladu. [12] [13]

ANY, EMPTY a kombinovaný obsah

Kontextový model může být deklarován pomocí speciálních klíčových slov *EMPTY* nebo *ANY*, a jak již bylo nastíněno, může být složen ze sekvence nebo výběru prvků, přičemž definované prvky mohou být různého datového typu – v tomto případě by se jednalo o kombinovaný obsah. [12] [13]

V případě, že je požadováno definovat element, který je prázdný (nenese žádný obsah), tak se využívá klíčového slova *EMPTY*. Příkladem může být: <!ELEMENT graphic EMPTY>, kde element symbolizuje libovolný grafický prvek. Pro takový prvek jsou většinou relevantní jeho vlastní atributy a samotný obsah není důležitý. Pokud je definována taková konstrukce, tak zaniká smysl pro použití párových tagů a je využito tagů nepárových. [12] [13] [17 str. 126]

Další klíčový slovem je slovo *ANY*. Toto klíčové slovo udává, že deklarovaný element může obsahovat jakýkoliv jiný element, který je definovaný v příslušném DTD. Mnoho publikací uvádí, že tento operátor v praxi je velice málo využíván. Důvodem nízkého počtu užití je fakt, že tento operátor nevypovídá žádné informace ohledně definované XML struktury, která by měla být do jisté míry striktní. Dále se s tímto operátorem může uživatel setkat v rozpracovaných DTD strukturách, kde autor, příslušné specifikace, nemá ještě zcela kompletní představu o tom, jak bude finální XML struktura vypadat. Avšak v rámci XSL procesorů, kterými se zabývá tato práce, bývá operátor *ANY* užíván v praxi frekventovaněji, neboť XML dokument může nést nějakou informaci. Příkladem může být procesní instrukce, která je zpracovávána aplikací třetí strany. Pokud je brán v úvahu následující příklad: <!ELEMENT endangered_species ANY>, tak element symbolizující ohrožené druhy nevypovídá zcela žádnou informaci o jeho obsahu. Jediné, co lze vydedukovat, je fakt, že tento element obsahuje jakékoliv elementy nebo text, které jsou součástí společného DTD. [13] [14] [15 str. 42]

Kombinovaným obsahem se rozumí obsah daného elementu, ve kterém je možné použít kombinaci elementu a textového řetězce na stejné úrovni. K použití této definice je možné využít kontextového modelu ve specifickém tvaru/zápisu. Tato definice bývá využívána např. v případě odstavců, zahrnujících text, jenž může obsahovat další elementy, specifikující dodatečné formátování daného textu, příkladem může být kurzíva. Na ukázkovém příkladu: <!ELEMENT para (#PCDATA | emphasis | xref)*> je vhodné

nastínit specifickou strukturu kontextového modelu. Důležité je, že textový řetězec (`#PCDATA`) musí být deklarovaný jako první v pořadí. Následovně je deklarován výběr jednotlivých elementů a společně všechny 3 prvky mají definovanou volitelnou četnost. Jinačími slovy by se o příkladu dalo říct, že element odstavce může zahrnovat text, který může zahrnovat pomocné formátování kurzívy nebo hypertextové odkazy. [17 str. 126] [13]

Validace DTD

Nyní už je zřejmé, jaká pravidla je třeba dodržovat při psaní XML dokumentu a jakým způsobem deklarovat pravidla pro daný dokument. Samotná validace představuje „spojení“ mezi XML dokumentem a DTD pravidly, jenž je koncipováno jako proces, jehož úkolem je aplikovat pravidla DTD na XML strukturu. Jak již bylo nastíněno, výsledkem tohoto procesu je informace o tom, zda daný XML dokument je správně formulovaný, tudíž jestli je validní nebo nevalidní. Dle příslušného validačního parseru bývá nevalidní dokument doplněn o chybovou hlášku, které může nést informace např. o syntaktické nebo kontextové chybě. V rámci validace se uplatňuje principu: „co není povoleno, to je zakázáno“. Vypovídající schopnost DTD o XML struktuře musí být tudíž velice rozsáhlá. [13] [17]

Nicméně validační proces není schopen na základě DTD pravidel rozeznat následující 4 úkazy [13 str. 27]:

- Bližší informaci o kořenovém elementu v XML dokumentu.
- Četnost jednotlivých typů elementů.
- Strukturu znakových dat uvnitř elementu.
- Informace o sémantickém použití daného elementu (Odpovídá obsah elementu jeho symbolickému významu?).

V následující ukázce bude naznačen výstup validačního procesu konkrétního validačního parseru. Bude se jednat o parser nsgmls, který vyhodnotil daný dokument jako nevalidní. Výstupem je chybová hláška, která poukazuje na 2 chyby. První z nich je kontextového charakteru, která vypovídá o tom, že odrážkový seznam se nachází na umístění, které mu není povoleno. Následuje syntaktická chyba, která upozorňuje na symbol „<“, jenž je nativně zakázáno používat mimo XML tagy. Jednotlivé hlášky jsou doprovázeny informací o příslušném názvu dokumentu a řádku i sloupci, kde je chyba nalezena.

```
% nsgmls -sv /usr/local/sp/pubtext/xml.dcl book.xml
/usr/local/prod/bin/nsgmls:I: SP version "1.3.3"
/usr/local/prod/bin/nsgmls:ch01.xml:54:13:E: document type does not
allow element "itemizedlist" here
/usr/local/prod/bin/nsgmls:ch01.xml:57:0:W: character "<" is the first
character of a delimiter but occurred as data
/usr/local/prod/bin/nsgmls:ch01.xml:57:0:E: character data is not
allowed here
```

Zdrojový kód 16: Ukázka výstupu validačního procesu z parseru nsgmls. [17 str. 23]

Xerces

Jedná se o validační parser vyvinutý společností The Apache Software Foundation. Tento parser představuje kolekci parsovacích knihoven, pomocí kterých je možné validovat, serializovat a různými způsoby manipulovat s XML dokumentem. Projekt je implementován v rámci jazyků: Java, C++ a Perl. Je zde podpora pro standardní API pro parsování XML dokumentu mezi než patří: DOM, SAX a SAX2. [17] [18]

Nsgmls

Nsgmls je parser a validátor, jehož autorem je James Clark. Tento nástroj je určen pro práci s SGML dokumenty, ale je kompatibilní i s XML dokumenty, neboť XML tvoří podmnožinu SGML. Parser je rychlý a dokáže zpracovat širokou škálu různých struktur. Je zde využito tzv. systémových identifikátorů, kteří mají za úkol identifikovat dokument, procesující konstrukt, jenž zprostředkovává přístup k objektům uložiště, v nichž jsou uloženy jednotlivé entity dokumentu. Systémové identifikátory se dělí na 2 typy a to na jednoduché a formální. Jednoduché zahrnují primitivní URL odkaz (v případě XML např. na příslušné DTD) nebo odkaz na soubor. Formálních identifikátorů se využívá v případech mapování. Mapování přísluší na jedné straně entitám dokumentu a jejich objektům uložiště. V rámci mapování se dají specifikovat relace (jedna k mnoho, mnoho k jedné), distribuované úložiště nebo úložné kontejnery. [17] [19] [20] [21]

XML4J a XML4C

Jedná se o XML validátory, na jejichž tvorbě se podílela společnost IBM. Jak již napovídá název obou nástrojů, jeden je kompatibilní s jazykem Java a druhý s jazykem C++. [17]

Co se týče historie, v roce 1999 společnost The Apache Software Foundation zahájila tvorbu open source XML řešení. Společnost IBM tento projekt podpořila tím, že darovala do projektu technologie jako XML4J, XML4C a LotusXSL. XML4J byl později přejmenován na Xerces a LotusXSL byl přejmenován na Xalan. Od té doby se IMB soustředí svou činnost na vývoj technologie Xerces, a proto v XML4J verzi 3.0.x došlo k výrazným změnám. Je možné si povšimnout, že hlavní .jar soubory nesou pojmenování „Xerces“ ve svých názvech. Zajímavým faktem může být to, že při tvoření technologie Xerces-C, neboli Xerces kompatibilní s jazykem C++, byla značně omezena množina všech podporovaných kódování. Xerces-C podporuje kódování jako ASCII, UTF-8, Windows-1252 a pár dalších. Zatímco XML4C podporuje více než 100 odlišných kódování, které jsou spadající pod ICU (International Components for Unicode). [22] [23]

3.4 XML Schéma

Nejprve je třeba si ujasnit dvojí význam XML Schéma. XML Schéma může být chápáno ve smyslu obecného jazyka pro popis pravidel XML. V tomto případě by DTD představovalo konkrétní typ XML Schéma. Druhý význam pro XML Schéma je, že XML Schéma reprezentuje konkrétní typ XML stejně tak jako v případě DTD. O tomto konkrétním typu XML Schéma, který je alternativou k DTD, bude pojednáváno v této kapitole. [16]

Jelikož DTD je velice rozmanité, tak přináší i řadu nevýhod, co se týče použitelnosti. Každý element, atribut nebo jakýkoliv obsah musí být striktně deklarován a to vyžaduje rozsáhlou definici. V případě objemnějších struktur poté začíná být DTD hůře čitelné a míra složitosti do jisté míry stoupá. Údržba kódu takové definice je poměrně komplikovaná. Dále DTD nerozlišuje definici lokálních a globálních proměnných, neboli v DTD se nesmí vyskytnout element se stejným názvem. Tento případ nastává v okamžiku, kdy uživatel chce definovat daný element ve 2 odlišných kontextech, ale bohužel narazí na názvovou schodu. Co se týče validace DTD, tak jednotlivé deklarace mají odlišný způsob zápisu syntaxe, než je tomu u XML. Proto na DTD nelze uplatnit logiku XML parserů. Dalším nežádoucím kritériem ohledně DTD je kontrola obsahu. DTD má jasně stanovené typy atributů, které lze využít. Pokud by ale uživatel chtěl předdefinovat datové typy, tak by mohl maximálně využít atributu CDATA, pomocí kterého by definoval daný typ atributu jako libovolný textový řetězec. Poslední z nevýhod může být fakt, že v DTD nelze uplatnit pravidla pro použití jmenných prostorů (viz kapitola: Jmenné prostory). Na základě těchto nevýhod bylo vytvořeno XML Schéma. [15] [16] [24]

XML Schema Definition (XSD) je alternativní XML schéma k definici DTD. Bylo vytvořeno v roce 2001 jako standard W3C. XSD je napsáno v jazyce XML. Podporuje validaci datových typů. Dokáže rozlišovat lokální a globální proměnné. XSD poskytuje přehlednější a lepší kontrolu nad obsahem daného XML dokumentu. [15] [16] [24]

Vytvoření XML Schéma

Podobně jako DTD, XML Schéma (XSD) se definuje v externím souboru, který má koncovku ve tvaru: .xsd. Tento soubor se poté aplikuje v rámci daného XML dokumentu. Poté je možné zahájit validaci daného obsahu XML dokumentu pomocí příslušného validátoru. [15]

Nejprve k tvoření externího XML schéma souboru. Elementy v XSD uplatňují logiku jmenných prostorů. Je tomu tak především proto, neboť XSD je psáno v jazyce XML. Jmenné prostory pomohou rozlišit elementy, které náleží XML Schématu a které nikoliv. Pokud by logika nebyla aplikována, mohlo by dojít k vzájemné kolizi názvů elementů (více informací je k nalezení v kapitole: Jmenné prostory). [15] [24]

V následující ukázce je znázorněno, jakým způsobem deklarovat XSD dokument. Element schema představuje deklaraci daného XML Schéma a příslušné vnořené elementy tvoří již danou definici. Příkladem může být element name, který nese datový typ: textový řetězec.

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

Zdrojový kód 17: Ukázka vytvoření XML Schéma dokumentu. [24 str. 20] [upraveno]

Ohledně obsahu XML dokumentu, je zapotřebí uvést referenci na dané XML. Na rozdíl od DTD, kde se v referenci používalo *DOCTYPE* definice, pro XML Schéma je využito skupiny atributů, které charakterizují příslušný XSD soubor a popř. ještě daný jmenný prostor. [25]

V následující ukázce je uvedena část XML dokumentu, ve kterém je použita reference na XSD dokument (`note.xsd`). Dokument obsahuje kořenový element *note*, na kterém jsou aplikovány pravidla XML Schema náležící externímu souboru. Atribut *xmlns* představuje výchozí jmenný prostor, který bude aplikován na všechny elementy, u kterých jmenný prostor nebude uveden. Dále atribut *xmlns:xsi* uvádí, že všechny elementy nebo atributy nesoucí jmenný prefix „xsi“ spadají do tohoto daného jmenného prostoru. Nakonec atribut *xsi:schemaLocation* obsahuje cestu k XSD dokumentu.

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

<note
xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/xml/note.xsd">
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Dont't forget met his weekend!</body>
</note>
```

Zdrojový kód 18: Ukázka reference na XSD dokument v XML dokumentu. [25]

Jednoduché a komplexní typy

Předtím než se začne definovat obsah XSD dokumentu by bylo dobré poznamenat, že každý element nebo atribut v definici může být jednoduchý nebo komplexní. V případě jednoduchých typů daný element může obsahovat pouze text nebo atributy. Kritérium pouhého textu se tudíž vztahuje i na atributy, proto se atributy dají považovat rovněž za jednoduché typy. Jednoduché typy jsou zcela nezávislé na jiných definicích. Ohledně komplexních typů, se zde předpokládá, že obsahem elementu budou další elementy. V jiném smyslu zde bude tvořena struktura požadovaného XML dokumentu. Každý z těchto 2 typů disponuje vlastní sadou definic, která je vázaná na obsah. U jednoduchých typů to může být použití omezení charakterizované skrze regulární výraz. U komplexních typů je možné se setkat s definicí sekvence elementů a jinými definicemi. [15] [24]

Jednoduché typy

Aby se v rámci XSD dalo vhodně pracovat s datovými typy, tak společnost W3C stanovila řadu pravidel, dle kterých se jednotlivé datové typy budou řídit, deklarovat a rozkládat/procházet. Pro lepší pochopení XML Schéma je vhodné uvést 4 prostory, které jsou zohledňovány v rámci deklarace/parsování jednotlivých datových typů. Důležitým z nich je lexikální prostor. Ten kontroluje a řídí sadu validních znaků, které mohou být použity v rámci datového typu. Např. pokud v definovaném elementu se vyskytne

hodnota 100 nebo hodnota 1.0E2, tak tyto hodnoty mají stejný význam, avšak v druhé hodnotě se vyskytuje alfabetický znak, který za nepřítomnosti tohoto prostoru by mohl být považován za nevalidní v rámci definování celých čísel. Další z uvedených je hodnotový prostor. Tento prostor disponuje rozsahem všech akceptovatelných hodnot v rámci daného datového typu. Můžou se zde objevit hodnoty jako části textu, datum, číslo nebo ustálené slovní spojení. Při nahlédnutí do podrobnějších mechanismů, je možné narazit na serializační prostor. Ten pracuje na úrovni bytů, neboli validuje a shromažďuje informace o použitých bytech a to v rámci celého dokumentu i na úrovni jednotlivých elementů nebo atributů, resp. jejich hodnot. Posledním prostorem je parsovací prostor, jenž při procházení jednotlivých hodnot datových typů konvertuje jednotlivé znaky do znakové sady Unicode a stará se o to, aby bílá místa byly normalizovány. [16] [24]

Datové typy

Mezi běžné datové typy patří: textové typy, numerické typy, typy data a času nebo výčtové typy. Aplikace jednotlivých typů se liší. Existují zde pravidla, dle kterých se jednotlivé typy řídí. Dají se i omezit pomocí fasetů. Základní pravidla jednotlivých datových typů a jejich omezení budou rozebrány v následujících kapitolách. [15]

Textové typy

Největší dilema u textových typů je postavení vůči normalizaci textu. Jelikož v XML dokumentu je možné využívat formátování, tak se může docílit nechtěného obsahu. Tímto obsahem se myslí odsazení textu, nadbytečné bílé místo, nadbytečné konce řádků apod. Proto vzniklo procesování bílého místa, jehož úlohou je eliminovat všechny tyto výskyty a docílit žádoucího obsahu. Tento proces se skládá ze dvou fází. Nejprve jsou všechny tabulátory, bílá místa, konce řádků nahrazeny jednotným znakem bílého místa: „#x20“. Poté jsou navazující množiny tohoto jednotného znaku nahrazeny pouze jedním znakem a to opět zmíněným znakem. [24]

Podle toho, které fáze normalizace textu je potřeba aplikovat, se dají využít různé textové typy. V tomto ohledu úplně volným textovým typem je *xs:string*, který neuplatňuje žádné kroky normalizace textu. Dále zde figuruje *xs:normalizedString*, který využívá pouze 1. fázi normalizace, tj. nahrazení tabulátorů nebo konců řádků jednotným znakem bílého místa. [24]

Na následující ukázce je možné si povšimnout, jakým způsobem se aplikovala normalizace textu (1. fáze normalizace). Jednotlivé tabulátory byly nahrazeny odpovídající množinou znaků bílého místa, avšak poté tato množina již nebyla zredukována. Následná redukce množiny je označována jako 2. fáze normalizace.

```
<title lang="en">
    Being a Dog Is
    A Full-Time Job
</title>

    Being a Dog Is    a Full-Time Job
```

Zdrojový kód 19: Ukázka normalizace textu při použití textového typu xs:normalizedString. [24 str. 35]

Pro využití obou fází normalizace slouží typ *xs:token*. Tento typ se dá považovat za rozšířený *xs:normalizedString* o redukování bílého místa. [24]

V nadcházející ukázce je znázorněna normalizace textu při aplikování *xs:token*. Jednotlivé tabulátory byly nahrazeny odpovídající množinou znaků bílého místa jako v případě předešlém a poté došlo k jejímu zredukování.

```
<title lang="en">
    Being a Dog Is
    A Full-Time Job
</title>

Being a Dog Is a Full-Time Job
```

Zdrojový kód 20: Ukázka normalizace textu při použití textového typu xs:token. [24 str. 37]

Další typy jsou většinou derivovány z typu *xs:token* nebo se uplatněním normalizace textu není potřeba zabývat. Příkladem může být *xs:language* nesoucí kódové hodnoty různých světových jazyků specifikovaných dle RFC 1766 nebo *xs:Name*, který specifikuje platný název v rámci XML dokumentu. Kromě normalizace textu zde figuruje i escapování znaků. S tím je možné se setkat u definování URL adres, resp. používáním *xs:anyURI*. Název působí otevřeně, neboť podstatou tohoto typu je podpora URI popsané ve specifikacích RFC 2396 a RFC 2732. Mnoho znaků, které nepatří do ASCII, je potřeba escapovat. V případě specifikování obrázků, které jsou kódovány pomocí base64, je možné využít textový typ *xs:base64Binary*. Tento typ kategorizuje znaky do skupin, které mají délku 6 bitů, a mapuje je vůči poli, které může nést 64 tisknutelných znaků. Vedle tohoto typu se ještě nabízí *xs:heyBinary*, který překládá binární oktet (skupina 8 bitů) na hexadecimální tvar o dvou znacích. [24]

Numerické typy

Jako prvním numerický typ se nabízejí decimální typy. Hlavním představitelem decimálních typů je *xs:decimal*, jenž představuje jakékoliv číslo, jehož části jsou tvořené pouze číslicemi 0-9, tečkou (charakterizující desetinný oddělovač) nebo znaménky: „+“ a „-“. Zkrácený tvar nul, jako je např. „E+2“, a bílé místa jsou zakázány. Nadbytečné číslice nuly, které jsou zapsány před danou hodnotou, jsou ignorovány. Od *xs:decimal* je odvozeno mnoho podtypů, které přinášejí dodatečné omezení. Příkladem může být typ *xs:integer*, který specifikuje celé číslo. Celá čísla se dají dále omezovat. Je možné využít typů: *xs:nonPositiveInteger* nebo *xs:negativeInteger*, jejichž účelem je deklarace toho, zda hodnota „0“ bude validní či nikoliv. [15] [24]

Celá čísla mají libovolnou délku, což není přívětivé v rámci zpracování mikroprocesorů. Proto zde figurují další typy, které nesou omezení délky. Základem omezení je počet jednotlivých bitů, které může daný textový výraz využít. Číselné výrazy jsou mnohdy brány jako textový řetězec. Dle počtu využívaných bitů se zde nabízejí jednotlivé typy. Nejjednodušším a z pohledu bitu nejmenším typem je *xs:byte*, který využívá množinu 8 bitů. Jelikož zde nefiguruje omezení, které by vylučovalo záporná čísla, tak výsledná hodnota může nabývat hodnot z intervalu od -128 do 127. Pro využití pouze kladné množiny čísel je zapotřebí využít podtypu *xs:unsignedByte*, jehož výsledné hodnoty patří do intervalu od 0 do 255. Obdobně zde figurují další typy, které si liší pouze počtem využívaných bitů. Např. *xs:short* využívá 16 bitů, *xs:int* využívá 32 bitů a více známý *xs:long* využívá 64 bitů. [24]

Numerické typy *xs:float* a *xs:double* taktéž mohou naplňovat příslušnou množinu bitů (32 a 64 bitů), avšak se zde dá využít robustnější množiny čísel. Je tomu tak, neboť tyto typy navíc podporují speciální techniky, díky nimž lze číselný zápis zkrátit. Jedna z technik se nazývá: „síla desítky“. Jedná se o reprezentaci plovoucí desetinné čárky, kde základem exponentu je 10. K využití této techniky se nabízí písmeno „e“, které se uvádí na konci výrazu, za kterým se uvádí kladné nebo záporné celé číslo, pro posun desetinné čárky buď vlevo nebo vpravo a s tím související počet míst, o které má být čárka posunuta. Příkladem může být hodnota: “-1.2344e56“. Další přidanou metodou je rozlišování pozitivní (0) a negativní nuly (-0), hodnota nekonečna, taktéž kladná (INF) nebo záporná (-INF) a hodnota „NaN“, která deklaruje, že daný výraz není číselným výrazem. [24]

Mezi numerické typy se řadí i *xs:boolean*. Tento typ může nést buď hodnotu „true“ nebo hodnotu „false“, proto se tento typ jeví jako nenumerického typu, avšak tyto hodnoty se interpretují jako 0 a 1, které se rovněž dají přímo využít při specifikování konkrétní hodnoty. [24]

Typy data a času

Definování formátů data a času se opírá o standard ISO 8601. Podstatou tohoto standardu je podpoření jednotného datového a časového formátu v rámci různých zemí. Datumový formát využívá Gregoriánského kalendáře. Na tomto základě je poté specifikováno mnoho variací výsledného podporovaného formátu v rámci

W3C XML Schema. Základem formátu času je UTC (Coordinated Universal Time). Tato zkratka představuje koordinovaný mezinárodní čas, jehož časová hodnota je shodná s časem, který je místním časem v Greenwichi (jedná se o park nebo také o observatoř nacházející se v Londýně). Od tohoto centrálního času se odečítají nebo přičítají hodiny dle odpočítaných časových pásem dané země. Tyto odpočty jsou zahrnuty v příslušných časových typech a také mají obecné označení, jenž symbolizuje časovou odchylku dané země od centrálního času. Příkladem může být Rusko, které nese v tomto kontextu označení UTC +3. [15] [24]

Nyní k jednotlivým typům. Prvním používaným datumovým a časovým typem je *xs:dateTime*. Tento typ představuje určení data a času, jehož formát vypadá následovně: „CCYY-MM-DDThh:mm:ss“. U tohoto formátu i u formátů jiných datumových a časových typů je využito počátečních písmen jednotlivých datumových/časových úseků, jenž jsou převzaté z angličtiny. Např. symbol „C“ představuje století a symbol „M“ představuje měsíc. Formát typu *xs:dateTime*, resp. všechny jeho části, musí být specifikovány. Proto např. hodnota „2001-10-26T21:32“ je nekompletní a tudíž nevalidní. Ačkoliv to není na formátu tohoto typu viditelné, tak je možné specifikovat i hodnoty pro rozlišení časových pásem a dokonce i hodnoty pro specifikování letního a zimního času. Pokud daný čas je UTC, tak se na konci hodnoty uvádí písmeno „Z“. Pokud daný čas není UTC, tak se na konci hodnoty uvádí hodinový rozdíl. Proto validní a správnou hodnotou pro Rusko je např. „2001-10-26T21:32:52+03:00“. [15] [24]

Typ data a času *xs:dateTime* je robustný a proto je zde mnoho typů, které tvoří jeho podmnožiny. Pro využití podmnožiny datumu je zde *xs:date*. Jeho příslušný formát je zapsán ve tvaru: „CCYY-MM-DD“. Příkladem může být definice zapsána následovně: `<xsd:element name=“birthdate“ type=“xsd:date“/>`. Pokud v rámci této definice je použito následující hodnoty: `<birthdate>1999-03-14</birthdate>`, tak daný výskyt bude považován za validní. [15 str. 78] Dále se nabízejí další typy, které představují jednotlivé fragmenty datumu. Může to být *xs:gYearMonth*, který představuje gregoriánský kalendářní měsíc. Podobně tak gregoriánský kalendářní rok: *xs:gYear* a další. [15] [24]

Podmnožina datumu již byla uvedena a nyní k podmnožině času. *xs:time* charakterizuje typ opakující se doby dne. Příslušný formát je psán ve tvaru: „hh:mm:ss.sss“, jehož upravená část se používá k určení příslušného časového pásma, ve tvaru „+hh:mm“ nebo ve tvaru „-hh:mm“. Je nutné poznamenat, že indikátor časové zóny je volitelný. [15] [24]

Speciálním typem spadajícím do časové kategorie je *xs:duration*. Tento typ představuje časové trvání, jenž disponuje zcela odlišným formátem v následujícím tvaru: „PnYnMnDTnHMnS“. Symbol „P“ (Period) rozlišuje časový a odpočtový formát. Symbol „T“ značí volitelnou část času a „n“ symbolizuje počet jednotek. Na rozdíl od formátu *xs:dateTime*, tento formát není striktně definován, uživatel není povinen specifikovat všechny jednotlivé části uvedené v tvaru formátu, namísto toho může specifikovat pouze potřebné indikace. Např. hodnota „P1Y2MT123s“ symbolizuje dobu trvání, resp. že daná událost trvá 1 rok, 2 měsíce a 123 vteřin. Tento typ lze použít i v opačném slova smyslu,

neboli, že nebude představovat časové trvání, ale časový odpočet. V tomto případě se na začátku hodnoty uvádí pomlčka např. takto: „-P1Y“. [15] [24]

Výčtové typy

Jedná se o seznam položek oddělených bílým místem. Jednotlivé položky představují jednoduché datové typy. Proto je každá položka seznamu validována vůči specifikovanému datovému typu. Existují 3 předdefinované výčtové typy. Prvním z nich je *xs:NMTOKENS*, jehož obsahem jsou položky typu *xs:NMTOKEN*, tzn. že každá položka musí být validní vůči typu *xs:token*, který koresponduje se správně zapsaným tvarem XML názvu. Dále zde figuruje *xs:IDREFS*. Tento výčtový typ zahrnuje položky typu *xs:IDREF* (oddělené bílým místem), z čehož každá tato položka musí referencovat datový typ *xs:ID*, který je součástí totožného XML dokumentu. Posledním výčtovým typem je *xs:ENTITIES*, který obdobně nese seznam položek typu *xs:ENTITY*. Každá entita náležící tomuto seznamu musí být definována v rámci schématu daného XML. [24]

Fasety

Jednoduché typy se řídí určitými pravidly. Některé z nich byly již uvedeny, jedná se o 4 prostory, které souvisí s parsování datových typů. Vedle těchto prostorů jako je lexikální a hodnotový figurují další dodatečné omezení zvané fasety. S fasety je možné se setkat především u deklarace omezení celých čísel nebo u specifikování datumu. [15] [16]

Fasety se dělí na 2 druhy: na řazené a neřazené. Mezi řazené patří: *minExclusive*, *maxExclusive*, *minInclusive*, *maxInclusive*, *scale* a *precision*. Mezi neřazené patří: *length*, *period*, *minLength*, *maxLength*, *enumeration*, *pattern* a *encoding*. Rozdílem mezi těmito 2 druhy je, že při použití řazených fasetů v určité kombinaci je možné docílit předdefinované sekvence hodnot. Např. při specifikování omezení datového typu: celé číslo, je možné omezit množinu používaných hodnot na stanovený interval celých čísel: <0;100> pomocí: `<minInclusive = „0“/>` a `<maxInclusive = „100“/>`. V případě neřazených fasetů této specifické sekvence (intervalu hodnot) nelze dosáhnout.

Příkladem může být použití: `<minLength value="5"/>` a `<maxLength value="8"/>` pro stanovení minimálního možného a maximálního možného počtu znaků, které může nést heslo, jenž je charakterizováno jako textový řetězec (datový typ). [16] [26]

V následující ukázce je znázorněno použití neřazeného fasetu enumeration. Jedná se o definici elementu představující vozidlo, jenž smí nést pouze jednu s definovaných hodnot: „Audi", „Golf" nebo „BMW", tak aby patřičný úsek XML dokumentu byl validní.

```
<xs:element name="car" type="carType"/>

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

Zdrojový kód 21: Ukázka použití neřazeného fasetu enumeration v rámci definování jednoduchého datového typu XML Schema. [26]

Neřazený faset *pattern* využívá metodiky regulárních výrazů. Problematika regulárních výrazů je velice obsáhlá a zahrnuje široké pole uplatnění. Použití této technologie tzv. regex je možné uplatnit validování dat v oblasti data-miningu (kde je možné data filtrovat a vyhledávat tak relevantní výskyty), k čištění dat (způsob transformace ze „surových" dat na data v přehlednějším tvaru), parsování textových řetězců (získávání parametrů z URL adresy), k nahrazování znaků a textových řetězců, k zvýrazňování syntaxe apod. [27] V případě XML Schema fasetů se regex využívá k validaci dat. Regulární výraz nedbá na problematiku datových typů. Pracuje pouze s textovým řetězcem a operuje na úrovni jednotlivých znaků. Při popisování jednotlivých datových typů byly taktéž uváděny jejich formáty, např. *xs:date* je psán ve tvaru: „CCYY-MM-DD". Takový zápis se dá díky regulárnímu výrazu omezit ještě menší sadu znaků. O takovém způsobu omezení bude následující ukázka. [15]

Následující ukázka nastiňuje způsob použití regulárního výrazu jako dodatečného omezení. Regulární výraz je v rámci neřazeného fasetu *pattern* aplikován na *xsd:timeDuration*. Původní zápis *xsd:timeDuration* vypadá následovně: „PnYnMnDTnHMnS", což je tvar uplatňovaný i pro *xsd:duration*, záleží na dané verzi XML Schema. V tomto výchozím tvaru lze pro dané časové trvání použít roky, měsíce, dny, hodiny, minuty a sekundy. Regex uplatňuje v tomto případě uplatňuje vzorec „P\d+D", kterým omezuje původní tvar tak, aby uživatel mohl časové trvání zapsat pouze pomocí dnů. [15]

```
<xsd:element name="gestation">
<xsd:simpleType>
   <xsd:restriction base="xsd:timeDuration">
   <xsd:pattern value="P\d+D"/>
   </xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

Zdrojový kód 22: Ukázka použití regulárního výrazu pro omezení jednoduchého typu timeDuration XML Schema. [15 str. 84]

Komplexní typy

Jak již bylo u uvedeno, komplexní element je element obsahující jiný element nebo elementy, nebo kterému je povoleno nést nějaké atributy. Komplexní element může mít 4 podoby. Jednou podobou může být prázdný element nebo element obsahující pouze elementy. Dále element obsahující pouze text nebo element, který obsahuje kombinaci elementů a textu. Jelikož komplexní typ představuje tvoření struktury výsledného XML dokumentu, tak obdobně jako při deklaraci DTD je zde nabízena sada operátorů, pomocí kterých je možné elementy deklarovat. [15] [28] V následujících kapitolách budou rozebrány některé z nich.

Sekvence elementů

Již ve zmiňovaném DTD byla sekvence elementů deklarována jako výčet elementů oddělených čárkami. Tato sekvence udává přesné pořadí, ve kterém musejí být definované elementy použity, jinak daná formulace nebude validní. Logika u XML Schema sekvence zůstává neměnná, liší se pouze použití. K dispozici je element *<xsd:sequence>* v němž jsou uvedeny všechny definované elementy, jenž figurují v rámci komplexního elementu. V takovém pořadí, ve kterém jsou elementy uvedeny v definici, musejí být i v takové pořadí použity. Je nutné podotknout, že jednotlivé nadefinované elementy se mohou použít ani jednou až vícekrát. Pro definování škály výskytu elementů, resp. o omezení počtu elementů ještě bude pojednáno v jedné z následujících kapitol. [15] [16] [26]

V následující ukázce je znázorněno použití sekvence dvou elementů. Komplexní typ zde představuje zvířecí druh, který nese údaje o jeho hrozbách a váze. Při specifikování by v tomto případě musely být nejprve specifikovány hrozby a poté až váha.

```
<xsd:complexType name="animalType">
<xsd:sequence>
<xsd:element name="threats" type="threatsType"/>
<xsd:element name="weight" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
```

Zdrojový kód 23: Ukázka použití sekvence elementů v rámci komplexních typů XML Schema. [15 str. 95] [upraveno]

Volba elementů

Pro volbu elementů (tzn. nadefinování elementů tak, aby uživatel si mohl vybrat buď jeden nebo druhý element) se v DTD oddělovaly jednotlivé elementy pomocí svislé čáry „|“. V případě XML Schema se pro definování této metody využívá operátor *<xsd:choice>*. Ohledně DTD se nesmí zkombinovat sekvenční a výběrový operátor. Proto DTD využívá logických závorek, aby jasně definovala použití sekvence a použití výběru. Tyto konstrukce mohou být v případě složitějších struktur hodně nepřehledné a v některých případech i omezující. V případě XML Schema je možné zkombinovat tyto 2 operace tak, že volba elementů může obsahovat elementy nebo sekvence elementů. [15] [16]

Následující ukázka zahrnuje definici volby elementů. V rámci volby elementů figuruje element „poddruhy“ a sekvence elementů. Sekvence nenese žádný název, ale v tomto případě by se dala pojmenovat jako „žádné_poddruhy“. Volba totiž poukazuje na to, že pokud daný zvířecí druh má poddruhy, tak budou uvedené poddruhy. Pokud poddruhy nezahrnuje, tak poté by bylo užitečné uvést alespoň elementy oblast a populaci v rámci definované sekvence.

```
<xsd:complexType name="animalType">
…
<xsd:choice>
   <xsd:element name="subspecies" type="subspeciesType"/>
   <xsd:sequence>
   <xsd:element name="region" type="xsd:string"/>
   <xsd:element name="population" type="popType"/>
   </xsd:sequence>
</xsd:choice> …
```

Zdrojový kód 24: Ukázka použití volby elementů v rámci komplexních typů XML Schema. [15 str. 96]

Libovolné pořadí elementů

Další nabízenou metodou je *xsd:all*. Mnohdy není důležité pořadí elementů a v případech, kde není na to nutné dbát, je možné využít této metody. Obecně se uvádí, že všechny elementy, které jsou součástí *xsd:all*, musí být použity alespoň jednou v libovolném pořadí. Je možné definovat i omezení četnosti výskytů. Slouží k tomu pomocné atributy *minOccurs* a *maxOccurs*, které budou rozebrány v jedné z následujících kapitol. [16] [29]

V následující ukázce je nastíněn způsob, kterým lze definovat informace o dané osobě. V tomto případě je předpokladem, že osoba bude mít vždy jméno a příjmení. Může mít i titul, ale ten nemusí být uveden. Všechny 3 elementy mohou být uvedeny v jakémkoliv pořadí.

```
<xs:element name="osoba">
   <xs:complexType>
      <xsl:all>
         <xs:element name="jméno" type="xs:string"/>
         <xs:element name="příjmení" type="xs:string"/>
         <xs:element name="titul" type="xs:string" minOccurs="0"/>
      </xs:all>
   </xs:complexType>
</xs:element>
```

Zdrojový kód 25: Ukázka použití libovolného pořadí elementů v rámci komplexních typů XML Schema. [29]

Reference elementů

V rámci komplexních typů lze z dané sekvence referencovat již definované elementy. Nezáleží na tom, zda referencovaný element je typu jednoduchého nebo komplexního. Je nutné dbát důraz na to, zda referencovaný element je deklarovaný lokálně nebo globálně. K definici reference slouží atribut *ref*, jehož hodnotou je název referencovaného elementu. Příkladem může být: `<xsd:element ref="label" minOccurs="1"/>`, kde „label“ představuje název elementu, který je deklarován globálně. [15]

Omezení počtu elementů

Již v DTD figurovaly operátory: „+“, „?“ a „*“ k definování četnosti. Jednalo se o definování daného počtu, kolikrát se může daný element vyskytnout v příslušné lokaci. Tyto DTD operátory slouží v DTD hlavně pro stanovení volitelnosti výskytu daného elementu. Jedná se o výskyt jednou nebo ani jednou, výskyt jednou a vícekrát, nebo o kombinaci obou předešlých variant. V XML Schema je tato problematika více rozvedena. Uživatel může definovat přímo dané počty příslušných výskytů. K této definici slouží 2 atributy. Prvním z nich je *minOccurs="n"*, kde hodnota „n“ představuje minimální počet výskytů příslušného elementu v dané lokaci. A na druhé straně se nabízí atribut *maxOccurs="n"*, jehož hodnota definuje maximální počet výskytů příslušného elementu v dané lokaci. Je nutné podotknout, že výchozí hodnota obou těchto atributů je „1“. Proto, když atributy nejsou uvedeny, tak daný element se musí vyskytnout v dané lokaci právě jednou. [15] [16]

Deklarace a reference skupin elementů

Již bylo uvedeno, že v komplexním typu lze definovat sekvence, volbu elementů, libovolné pořadí elementů, reference na element a elementy samotné. Dále je možné definovat tzv. skupinu elementů nebo její referenci. Skupinu elementů lze využít např. u často opakující se množiny elementů, která se nachází v daném schématu. Může se jednat příkladem o atributy nějaké entity, váha a výška mohou být elementy pro osobu nebo zvíře. Skupina elementů se definuje jako *<xsd:group>*. Taková skupina posléze může nabývat málo rozeznatelného obsahu od komplexního typu. Uvnitř skupiny lze uvést sekvenci, volbu elementů, libovolné pořadí elementů, referenci na element, element samotný apod. Skupina elementů nese atribut *name*, jehož hodnota indikuje název skupiny (obdobně jako název elementu). Skrze tento název lze na příslušnou skupinu referencovat. Reference na skupinu může vypadat následovně: *<xsd:group ref="label">*, kde atribut *ref* nese hodnotu, jenž odkazuje na definici příslušné skupiny. V tomto případě na skupinu s názvem: „label“. [15] [16]

Následující ukázka znázorňuje definici skupiny elementů v rámci XML Schema. Skupina představuje fyzické rysy: váhu, výšku apod. Tato skupina fyzických rysů by mohla figurovat jako reference elementu indikující zvíře v obecném slova smyslu nebo jako reference indikující konkrétní zvíře.

```
<xsd:schema>
<xsd:group name="physical_traits">
<xsd:sequence>
<xsd:element name="weight" type="xsd:string"/>
<xsd:element name="length" type="xsd:string"/>
<xsd:element name="gestation" type="xsd:timeDuration"/>
<xsd:element name="distinguishing" type="xsd:string"/>
</xsd:sequence>
</xsd:group>
…
```

Zdrojový kód 26: Ukázka definice skupiny elementů v rámci komplexních typů XML Schema. [15 str. 98]

Deklarace a reference skupin atributů

Atributy patří mezi jednoduché typy. Avšak v rámci komplexních typů lze definovat skupiny atributů. Skupina atributů se posléze dá vykládat jako komplexní typ, neboť představuje definici struktury dokumentu. V XML Schema může atribut být vyžadován, být volitelný nebo může být zakázaný. K této definici slouží atribut use, který může nabývat hodnot: „required", „optional" nebo „prohibited". Obdobně jako v DTD je zde možnost využít fixní nebo implicitní hodnoty. Fixní hodnota představuje hodnotu, která musí být uvedena v atributu. Příkladem použití může být následující: `use="fixed" value="101"`. Příklad uvádí dva definované atributy v rámci daného atributu. Dle této definice definovaný atribut musí nést hodnotu „101". Obdobným způsobem je možné využít implicitní hodnoty, neboli výchozí hodnoty. Výchozí hodnota ve tvaru: `use="default" value="101"` udává hodnotu „101" jako počáteční hodnotu. Avšak tato hodnota může být změněna. [15] [16]

Podobně jako u skupiny elementů, je vhodné využít skupinu atributů v případě, že se nabízí použití stejné množiny atributů v rámci více rozdílných kontextů. Skupiny atributů se definují pomocí: *<xsd:atributeGroup>*. Tato definice posléze musí nést atribut *name* nebo atribut *ref*. Atribut *name* indikuje pojmenování dané skupiny atributů a *ref* představuje referenci na danou hodnotu atributu *name*, jenž je součástí odkazované skupiny atributů. [15] [16]

V následujícím příkladě je naznačena definice skupiny atributů, která představuje atributy obrázku. Danými atributy zde jsou: název souboru a souřadnice x a y. Tato skupina může usnadnit opakované použití definice vlastností daných elementů, která charakterizuje obrázky.

```
<xsd:atributeGroup name="imageAtts">
   <xsd:attribute name="filename" type="xsd:uri-reference"/>
   <xsd:atribute name="x" type="xsd:integer"/>
   <xsd:atribute name="y" type="xsd:integer"/>
</xsd:attributeGroup>
```

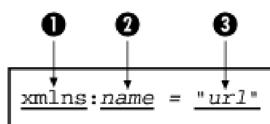
Zdrojový kód 27: Ukázka definice skupiny atributů v rámci komplexních typů XML Schema. [15 str. 111]

3.5 Jmenné prostory

Problematika jmenných prostorů se prolíná napříč celou problematikou XML. Jelikož XML Schema (alternativa DTD) je taktéž založeno na XML struktuře, jmenné prostory zde zaujímají důležité postavení. Jmenné prostory, označovány jako *namespaces*, se využívají především proto, aby nedošlo ke kolizi stejnojmenných elementů. Kolizí se myslí situace, kdy dojde ke zkombinování 2 různých XML dokumentů nebo jejich schémat. V obou dvou zkombinovaných částech bude figurovat stejnojmenný element např. source. V jednom XML dokumentu by mohl tento element nabývat významu ve smyslu „knižní zdroj“ nebo „URL adresa“ a v jiném XML dokumentu by mohl představovat „pramen řeky“. V tomto případě po vzájemné kombinaci může dojít k narušení významového obsahu. Obranou proti této kolizi jsou jmenné prostory. [15] [17]

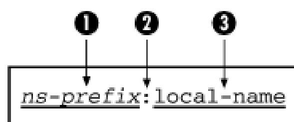
Použití v XML

Na obrázku č. 1 je znázorněna deklarace jmenného prostoru v XML dokumentu. Již při deklaraci DTD nebo XML Schema bylo možné si povšimnout použití této deklarace v konkrétních případech. Definice se uvádí jako součást XML elementu. Často bývá deklarována na kořenovém elementu, neboť bývá cílem mít pokrytý celý XML dokument určitým schématem. Důležité je klíčové slovo: „xmlns“ (označení pro XML *namespace*). Jakmile je toto slovo uvedeno v rámci atributů nějakého elementu, pro XML parser je to příznak pro deklaraci jmenného prostoru. Tzn. že na každém vnořeném elementu bude aplikován specifikovaný jmenný prostor. Dále se uvádí název jmenného prostoru, který mívá zkratkovitou povahu a je oddělen dvojtečkou od předešlého příznaku. V předcházejících kapitolách bylo možné si povšimnout, že název pro XML Schema byl v jednom případě „xs“ a ve druhém „xsd“. Avšak to je v pořádku, neboť název je libovolný. Po specifikování názvu jmenného prostoru následuje URL adresa. Tato adresa odkazuje na daný *namespace*, který se má aplikovat pro daný název. [15] [17]



Obrázek 1: Deklarace syntaxe jmenného prostoru. [17 str. 39]

Následující obrázek naznačuje použití *namespace* v rámci XML. Ať už se jedná o název elementu nebo název atributu, jmenný prostor (resp. jeho zkratka) se uvádí bezprostředně před název elementu nebo atributu. Tyto dva názvy jsou odděleny tečkou. [17]



Obrázek 2: Aplikování jmenného prostoru na XML názvu. [17 str. 38]

Použití v XML Schéma

V rámci XML Schéma je použití *namespace* rozsáhlejší. Z pravidla se zde prolíná vícero jmenných prostorů najednou. Jedním jmenným prostorem bývá právě *namespace* pro dané XML Schéma. Již bylo řečeno, v mnoha předešlých ukázkách XML Schema (alternativa DTD) bylo použito zkratky jmenného prostoru: „xsd“, které představuje zkrácený tvar od: XML Schema Definition. Dále se používají jmenné prostory elementů, které náleží do daných jmenných prostorů v rámci XML Schema definice. Této metodě se někdy přezdívá „obývání“ jmenného prostoru nebo také specifikace cílového jmenného prostoru. [15] [16]

Deklarace jmenného prostoru pro XML Schema se provádí stejným způsobem jako v případě XML. Specifikace *namespace* pro cílový jmenný prostor a pro použití daných operátorů je odlišná. Na element *<xsd:schema>* se deklaruje atribut: `targetNamespace=“URL“`, kde hodnota atributu představuje cílový jmenný prostor, který se má aplikovat. Nyní však do cílového *namespace* spadají pouze deklarace na globální úrovni (1. úroveň elementů v *<xsd:schema>*). Pro aplikaci cílového jmenného prostoru i pro více vnořené úrovně se deklaruje dodatečný atribut: `elementFormDefault=“qualified“`. Avšak tato deklarace se týká pouze elementů, pro atributy je zde další dodatečný atribut: `attributeFormDefault=“qualified“`, který je potřeba specifikovat pro aplikování cílového schématu na atributy, které jsou součástí vnořené úrovně. Opakem hodnot těchto speciálních atributů je hodnota: „unqualified“, která je nastavena jako výchozí hodnota. Tato hodnota může být nápomocna v případech výjimek. Za předpokladu, že bude specifikovaný `elementFormDefault=“qualified“`, budou všechny elementy spadat do cílového *namespace*. Avšak u všech elementů toto nemusí být žádoucí. Proto je možné cílový jmenný prostor na daných elementech tzv. „vypnout“ a to deklarací: `form=“unqualified“` přímo na konkrétním elementu XML Schéma. [15] [24]

Pokud již jsou v XML Schema zavedeny jmenné prostory je vhodné, pro lepší přehlednost cílového schématu, definovat předponu. Tyto předpony se často používají i k vymezení komplexních typů schématu pomocí reference. Předpony se však uvádět nemusejí a jak již bylo několikrát uvedeno, pokud bude specifikován pouze atribut *xmlns*, tak hodnota tohoto atributu, resp. referencované *namespace*, bude použita na všechny element/atributy, na kterých nebyl aplikován jmenný prostor. Takto definovaný jmenný prostor se nazývá: *implicitní namespace*. [15] [16]

Následující ukázka znázorňuje použití implicitního jmenného prostoru. V ukázce figurují metody aplikování tohoto jmenného prostoru pro elementy a atributy. Cílový namespace představuje knihovnu, jejíž prvky, neboli prvky tohoto jmenného prostoru, budou aplikovány pro komplexní typ zvaný: typ knihy.

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://dyomedea.com/ns/library"
   elementFormDefault="qualified" attributeFormDefault="unqualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://dyomedea.com/ns/library">
      <xs:element name="library">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="book" type="bookType"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      …
```

Zdrojový kód 28: Ukázka aplikace implicitního jmenného prostoru "knihovna" v rámci XML Schema. [24 str. 175]

Do stávajícího jmenného prostoru lze libovolně přidávat další externí definice pomocí *xsd:include*. V tomto elementu figuruje atribut *schemaLocation*, který nese odkaz na externí soubor s koncovkou *.xsd*. Zahrnutý jmenný prostor musí odpovídat definovanému jmennému prostoru v *xsd:schema* deklaraci, jinak externí soubor bude ignorován. Tímto způsobem se dají jednotlivé úseky XML Schema rozdělit do externích souborů. Tato metoda je efektivní hlavně z hlediska znovupoužitelnosti a přehlednosti daných deklarací. [15] [30]

xsd:include se často zaměňuje s *xsd:import*. Zásadní rozdíl mezi těmito dvěmi vloženími externích souborů je v použitém jmenném prostoru. Jak již bylo uvedeno, deklarace *xsd:include* musí být přímou množinou nebo podmnožinou *namespace* deklarovaného v atributu *targetNamepsace* (definice *namespace* pro XML Schema). Zatímco *xsd:import* může nést deklaraci odlišných jmenných prostorů. Tento element nese stejný atribut jako *xsd:include*, a to: *schemaLocation*, ve kterém se nachází odkaz importované XSD definice. Navíc se uvádí další atribut s názvem: *namepsace*, jehož obsahem je použitý jmenný prostor. [15] [31]

3.6 XPath

Vhodný nástrojem pro práci s XML je jazyk XPath, který přestavuje speciální jazyk pro práci s XML dokumentem. Tento jazyk slouží pro identifikaci částí XML dokumentu a na samotném dokumentu je zcela nezávislý. Již při představování XML dokumentu bylo pojednáváno o tzv. nodes, které představují množiny nebo podmnožiny XML struktury. XPath pracuje právě na úrovni těchto nodů. Jinými slovy XPath funguje jako vyhledavač, jehož nalezené výskyty představují seznam nodů. Nejkomplikovanější složkou jazyka XPath je jeho syntaxe, která pracuje na úrovni elementů, atributů, nodů, komentářů, procesních instrukcí, jmenných prostorů, kořenového elementu, identifikátorů a speciálně navržených funkcí. Ve výsledku je jazyk XPath natolik flexibilní, jednoduše použitelný a výkonný, že poslouží pro práci s XML dokumentem editorům i programátorům. Dále je nedílnou součástí jazyka XSLT, který bude uveden v jedné z následujících kapitol. [13] [32] [33]

Pro nastínění základní funkcionality je vhodné uvést, že jazyk XPath pracuje na daných úrovních XML dokumentu. K pohybu po úrovních je využíváno dopředných lomítek. Jedno lomítko: „/“ znamená, že XPath bude pracovat na úrovni aktuálního nodu. Pokud je lomítko hned na začátku, tak aktuální úroveň je rovna kořenovému elementu. Pokud jsou uvedena lomítka dvě následovně: „//“, tak zpracovávaná úroveň se nachází kdekoliv v dokumentu. Tzn. že XPath bude procházet všechny nody, pokud nebude specifikováno více informací. Ve většině případů za lomítkem následuje název hledaného elementu, který způsobí, že XPath bude hledat výskyty s daným názvem. Pokud není záměrem posun v jednotlivých úrovních dokumentu, ale je třeba pouze deklarovat podmínku dané lokace, tak se místo lomítek používají hranaté závorky. Obsahem hranatých závorek bývá opět syntaxe jazyka XPath. Proto je zde možné očekávat, že se bude hledat vnořený element, node na určité pozici, počet uvedených elementů, daný atribut apod. Právě hledání atributů bývá zde velice frekventované, proto je vhodné uvést syntaxi pro identifikaci atributu. Proto, aby se atributy rozeznaly od elementů, tak u atributů se uvádí pomocný znak „@“, jako předpona názvu atributu. Dále je možné zjišťovat, zda daný atribut nese konkrétní hodnotu. Podobně jako u deklarace atributů, se pro přiřazení hodnoty používá znak „=“ a hledaná hodnota vepsaná do uvozovek. Příkladem hledaného výrazu může být: `//moznost[@hlasu="5"]`. Po spuštění XPath, s tímto vstupem, nad daným XML dokumentem, bude XPath zpracovávat všechny jeho nody. Výsledkem zpracovávání bude seznam nodů, které budou korespondovat s hledaným výrazem, a to: element s názvem „moznost“, který má atribut „hlasu“, který nese hodnotu „5“. [13] [33]

4 Docbook DTD standard

V předešlých kapitolách byly popisovány pouze obecné pojmy, standardy, technologie a jejich uplatnění nebo způsob použití. Nyní bude popsáno konkrétní XML Schéma, které bude uplatněno pro výstupní dokumentaci.

V roce 1991 DocBook vznikl jako SGML orientovaný formát. Jeho hlavní úlohou byla tvorba unixové dokumentace. Tento systém je vyvíjen dodnes a slouží především pro tvorbu hlavně technických dokumentů. DocBook disponuje možností konverze jednoho vstupního formátu do podoby mnoha výstupních formátů jako HTML, PDF, Postscript, RTF apod. Standard je možné nalézt na: https://docbook.org/, kde je možné si stáhnout příslušné verze. V této práci bude uplatněna DocBook verze 4.x. DocBook představuje obsah knih. Prvky tohoto obsahu (kapitoly, apendix, rejstřík, titulní strana, systém sudých a lichých stránek apod.) je možné aplikovat v rámci technické dokumentace různé povahy. [34] [35] [36]

Docbook nabízí řádově stovky elementů, které je možné využít. Pokud by záměrem použití DocBook standardu bylo vytvoření dokumentace využívající prvky knižního obsahu, tak by struktura mohla být následující: kniha -> kapitola -> sekce -> obsah sekce. S použitím DocBooku by kořenový elementem byl *<book>*, jenž by nesl metadata zahrnuté v těle elementu *<bookinfo>* a na stejné úrovni by se nacházely příslušné kapitoly: *<chapter>*. Posléze každá kapitola by nesla nadpis *<title>*, následoval by volitelný obsah kapitoly a poté příslušné sekce: *<section>*. Kromě předchozích elementů, lze využít i dalších jako např.: *<toc>* pro definování obsahu, pro deklaraci rejstříku lze využít *<index>*, *<appendix>* pro charakterizování appendixu, *<glossary>* pro glosář apod. Nejvíce ustáleným elementem je *<para>*, který nese obsah odstavce. Příkladem takového odstavce může být: `<para>Toto je obsah elementu para</para>`. Je možné si povšimnout, že obsahem tohoto elementu může být teoreticky jakýkoliv text. Avšak dle hlediska sémantiky je vhodné, aby odpovídající element měl odpovídající obsah. DocBook nabízí popis každého použitelného prvku. Proto lze vypozorovat, zda obsah odpovídá dané deklaraci elementu. Příkladem může být element: *<filename>*, jehož popis je k nalezení zde: https://tdg.docbook.org/tdg/4.5/filename.html. Dle uvedeného popisu by obsahem tohoto elementu měl být název souboru. Avšak, dle DTD, obsahem může být libovolný textový řetězec. [37] Pro XML Schéma nebo validátory obecně je prakticky nemožné takový obsah řádně zvalidovat. Proto je nutné dbát na hledisko sémantiky již při editaci dokumentu a řídit se dle popisů použitého XML Schéma. [34] [35]

Aplikování DocBook schématu pro dané XML je stejné jako aplikování DTD schématu. Ihned za deklarací procesní instrukce nesoucí údaje o verzi XML a případném kódování se uvádí DOCTYPE s následujícím kořenovým elementem, který koresponduje s definicí aplikovaného schématu. Dané schéma se deklaruje jako URL odkazující na daný DTD dokument. Po definování schématu již následuje kořenový element a požadovaná XML struktura. [34]

Příkladem může být následující deklarace XML dokumentu, kde je aplikován DocBook DTD verze 4.5 a hlavní kořenový element představující knihu.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE book PUBLIC „-//OASIS//DTD DocBook XML V4.5//EN"
   http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd>
<book lang="cs">
   …
</book>
```

Zdrojový kód 29: Ukázka aplikace DocBook schématu v rámci XML dokumentu. [34] [upraveno]

4.1 Editory XML

Nyní již jsou známy problematiky ohledně XML, XML Schema, aplikace daného XML Schema (DocBook) a pomocný nástroj pro editaci XML, XPath. Bylo by vhodné uvést několik editorů, které je možné použít při editaci XML dokumentu, kde by byla uplatněna DocBook struktura.

Emacs

Emacs představuje multiplatformní textový editor. Je volně dostupný a rozšiřitelný. Nabízí přepínání do nXML módu, který je určený pro editování XML dokumentů. Součástí tohoto módu je podpora pro RELAX NG (alternativa XML Schema). Pokud je XML Schema prioritní, je možné nalézt externí konvertory pro transformování RELAX NG do XML Schema. [38] [39]

oXygen

Sada produktů oXygen nabízí XML řešení pro vývoj, editaci aj. XML Editor disponuje XML autoringem. Jedná se o speciální mód, pomocí kterého je možné přepnout XML strukturu do „finální" podoby výstupu. Např. DocBook tabulky, odstavce, nadpisy budou naformátovány. Dokonce obrázky budou vyobrazeny. V tomto módu je možné i nadále editovat XML strukturu. XML Editor dále nabízí integrovanou XPath, XML validaci aj. Používání tohoto softwaru je zpoplatněno. [40]

XMLMind XML Editor

Jedná se o XML editor, který nabízí WYSIWYG prostředí pro práci se schématy jako: DITA, DocBook nebo XHTML. Některé edice nabízejí addony jako XMLmind XSL-FO Converter, pro konverzi z XML do formátů: DOCX, RTF, ODT nebo WML. Nebo addon XMLmind Word To XML, pro konverzi z DOCX do XML. Tento software je zpoplatněn. [41]

5 XSL

Jazyk XSL (eXtensible Stylesheet Language) slouží k popisu transformace XML souborů. XML soubor je možné převádět do podoby HTML, JavaScriptu, SQL, XSL, TXT, RTF a dalších podob. XSL představuje „pouhý“ popis XML transformace, samotná transformace již spadá do XSLT, ve kterém je využíváno příslušných transformačních procesorů. Pro složitější výstupy z transformace je využito jazyka XSL-FO, neboli tzv. formátovacích objektů, pomocí kterých je možné popsat výslednou podobu dokumentu ve tisknutelném formátu jako je např. PDF. [42]

5.1 XSLT

XSLT (eXtensible Stylesheet Language Transformation) je označení pro širší specifikaci jazyka XSL. XSLT se specializuje na provedení daných transformací XML dokumentu do podoby jiné. Často to bývá transformace do podoby HTML. Provedení XSLT je založeno na datových strukturách, které mohou být označeny za známé. Jedná se o struktury, jejichž definice náleží konzorciu W3C. Transformované vstupy mohou být vícejazyčné (s diakritikou), neboť je zde podpora pro kódování v UTF-8. XSLT nabízí vysokou přístupnost ke zdrojovému kódu. Specifikace je psána v jazyce XML, kde je využito příslušného jmenného prostoru. Nabízené procesory XSLT bývají multiplatformní a veliká řada programovacích jazyků nabízí sama o sobě knihovny pro využívání této specifikace. [42] [43]

Základní použití

Obsah XSL dokumentu tvoří XML, které využívá jmenného prostoru specifikace XSLT. Proto je potřeba uvést procesní instrukci udávající verzi XML a popřípadě kódování. Dále se definuje *<xsl:stylesheet>*, na který je aplikován *namespace*, který je referencován na danou specifikaci XSLT. Poté následují doplňující nastavení. Příkladem může být nastavení výstupní metody: „xml“, která udává, že daný XML dokument bude pomocí šablon transformován do jiného XML dokumentu. [42]

Následující ukázka naznačuje způsob prvotní deklarace XSL dokumentu, který je možné použít k dané definici transformace.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl=http://www.w3.org/1999/XSL/Transform>

<xsl:output method="xml" indent="yes"/>

…

</xsl:stylesheet>
```

Zdrojový kód 30: Ukázka prvotní deklarace XSL dokumentu. [42 stránky 26,27] [upraveno]

Šablona

Celou definici transformace tvoří převážně šablona, neboli *<xsl:template>*. Šablona je základním prvkem XSLT. Využívá jazyka XPath (viz kapitola: XPath), který slouží k orientaci jednotlivých šablon. Principem takové šablony je transformovat hledaný XML element (resp. XML node). Transformační proces probíhá tak, že pokud šablona (resp. její XPath) narazí na hledaný XML node, tak jej transformuje do podoby, která je definovaná v těle dané šablony. Avšak jakým způsobem budou XML fragmenty procházeny? Často bývá procházení zahájeno na úrovni kořenového elementu XML dokumentu. Poté bývají šablony aplikovány sestupným směrem, tedy od rodičovského elementu až po vnořené struktury. K definování procházení od kořenového elementu slouží opět XSL šablony. Šablony mohou ve svém těle aplikovat jiné šablony pomocí: `<xsl:apply-templates/>`. Tzn. že zpracování daného nodu není v těle šablony ukončeno. Pokud pro definovanou úroveň existuje deklarovaná šablona, tak posléze bude aplikována. [42] [43]

V následující ukázce je znázorněna deklarace šablony, která slouží k zahájení aplikace šablon na kořenovém elementu XML dokumentu.

```
<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>
```

Zdrojový kód 31: Ukázka deklarace XSL šablony pro aplikaci šablon od kořenového elementu. [42 str. 112]

Pro deklaraci výstupní podoby danou šablonou se výstupní objekty/značky vepisují do těla šablony. Jako výstupní formát může být použit jazyk HTML. Definování výsledného tvaru HTML jazyka je stejné jako sestavování HTML značek při vytváření webových stránek. Avšak indentace je ignorována, indentace se řídí nastavením výstupu. Pokud je záměrem vypsat hodnotu z XML, používá se: *<xsl:value-of>*. Pomocí tohoto výpisu je možné vybrat daný node a vypsat jeho hodnotu. [43]

Následující ukázka zahrnuje deklaraci šablony, která je aplikována v rámci elementu s názvem: „greeting“. Při aplikaci je daný element transformován do specifikované HTML podoby, přičemž obsahem hlavního nadpisu bude hodnota tohoto elementu.

```
<xsl:template match="greeting">
   <html>
      <body>
         <h1>
            <xsl:value-of select="."/>
         </h1>
      </body>
   </html>
</xsl:template>
```

Zdrojový kód 32: Ukázka deklarace XSL šablony pro daný element, který bude transformován do HTML podoby. [43 str. 26]

Procesory XSLT

K dispozici je mnoho procesorů, které jsou vytvořené v jazyce Java. Proto je potřeba mít dostatečnou znalost Java prostředí, resp. umět zacházet s proměnným prostředí: *classpath*, znát způsoby zacházení s *.jar* souborem a vědět, jakým způsobem volat příslušné parametry. [42]

Saxon

Autorem tohoto procesoru je Michael Kay. Saxon patří mezi první XSLT procesory. V dnešní době je k dispozici ve 3 nabízených podobách. Saxon-HE (Home Edition), který představuje open-source pro osobní použití. Dále Saxon-PE (Professional Edition), který je zpoplatněn, nabízí navíc možnosti lokalizace a podporu pro dodatečné objektové modely. Posledním nabízeným je Saxon-EE (Enterprise Edition), který je také zpoplatněn a navíc zahrnuje mnoho dalších technologií. Tento procesor je podporovaný v jazyce Java a .NET. Poté ještě existují ucelené verze, z nichž např. Saxon-JS je kompatibilní s JavaScript prostředím. [42] [44]

Xalan

Jedná se o velice často využívaný XSLT procesor. Spadá do projektu Apache (The Apache Xalan Project) a je vysoce rozšířený v prostředí Java. Je volně dostupný a je podporovaný v rámci programovacích jazyků Java a C++. Zajímavostí je, že název Xalan je odvozen od afrického hudebního nástroje zvaného Xalam a tento nástroj je také vyobrazen jako součást loga tohoto projektu. [42] [45]

5.2 XSL-FO

Již byla popsána technologie XSLT, která představuje způsob, pomocí kterého je možné transformovat data do podoby formátovacích objektů. Z transformační fáze XSLT tak mohlo vzniknout RTF, XML, HTML apod. Vedle XSLT se nachází XSL-FO, které je schopné pracovat s formátovacími objekty. Tzn. že je dokáže nastylovat a procesovat jejich strukturu, vytvořit odkazy, vytvořit navigaci dokumentu a mnoho dalšího. XSL-FO se dá považovat za samostatnou specifikaci formátovacích objektů, protože zahrnuje mnoho vlastních technologií a je daleko rozsáhlejší než XSLT. [42] [46]

Základní použití

První krokem je vytvoření XSL-FO dokumentu. Tento dokument obsahuje množinu formátovacích objektů, které spadají do specifikace XSL-FO. Jeho vytvoření je totožné s vytvářením např. HTML výstupu v rámci XSLT šablon, neboť jde o transformaci XML elementů (resp. nodů) do podoby jiné. Takto vytvořený XSL-FO dokument je potřeba dále procesovat a to nikoliv pomocí XSLT procesoru, ale pomocí tzv. XSL-FO formátovacího procesoru. Několik ukázek XSL-FO procesorů bude popsáno v následujících kapitolách. Daný procesor má za úkol XSL-FO dokument „vyobrazit“ (naformátovat) do výsledné podoby, takovou podobou může být dokument PDF. [42] [47]

Co se týče použití formátovacích objektů v rámci XSL šablon, tak nejprve je potřeba deklarovat příslušný jmenný prostor. Deklarace může mít následující podobu: `xmlns:fo=http://www.w3.org/1999/XSL/Format`. Poté již je možné používat elementy z rodiny XSL-FO v rámci XSL šablon. [47]

Příkladem může být formátování odstavce. V následující ukázce šablona bude transformovat element *<para>* na element *<fo:block>*. Tento element bude představovat blok textu, jenž bude zahrnovat deklarovaný font: „Times“, bude mít definovanou velikost písma, bude zarovnán do bloku apod.

```
<xsl:template match="para">
 <fo:block
   font-family="Times"
   font-size="12pt"
   space-before="12pt"
   space-after="12pt"
   text-align="justify">
       <xsl:apply-templates/>
 </fo:block>
</xsl:template>
```

Zdrojový kód 33: Ukázka deklarace XSL šablony pro daný element, který bude transformován do XSL-FO podoby. [47] [upraveno]

Předchozí ukázka nastiňuje vytvoření elementu *<fo:block>*. Podobně jako jazyk HTML má XSL-FO specifikace definovanou strukturu, jenž musí být validní pro správné zobrazení. Hlavní rozdílem XSL-FO specifikace a např. HTML spočívá v účelu použití. HTML jazyk slouží pro sestavení struktury webových stránek. Tato struktura je nadále formátována pomocí CSS. Webové stránky se zobrazují pomocí monitorů (nebo jiných zařízení), jejichž základní jednotkou „zobrazení“ je pixel. Síť pixelů poté udává rozlišení a výslednou kvalitu obrazu. Zatímco XSL-FO specifikace slouží pro tiskařské účely. Tzn. že výstup je uzpůsoben tak, aby bylo možné jej vytisknout. Proto se v XSL-FO namísto pixelů objevuje jednotka point (pt), palec (in) nebo jednotky metrické soustavy (cm). Výstupní podobou mohou být listy ve formátu A4. Proto XSL-FO nabízí možnost konfigurace záhlaví, těla a zápatí. Dále definování vzdálenosti okrajů stránky od obsahu, číslování, nastavování sekvencí stránek (pro nastavení formátu sudé a liché stránky) apod. [42] [47]

Následující ukázka znázorňuje základní specifikaci stránky XSL-FO dokumentu. Element *<fo:root>* značí kořenový element XSL-FO, jehož obsahem je deklarace pravidel obsahu a obsah samotný. V *<fo:layout-master-set>* je k nalezení deklarace pravidel stránky pojmenovaná jako: „only“, která nese definici stránky jako celku a poté definici jejího těla, záhlaví a zápatí. Po nastavení pravidel následuje deklarace sekvence stránky, která je referencována zmíněnou sadou pravidel. V sekvenci je deklarován tok těla stránky, do kterého bude vepsán blok s příslušným obsahem. [47]

```
<?xml version="1.0" encoding="utf-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <fo:layout-master-set>
      <fo:simple-page-master
      page-height="11in"
      page-width="8.5in"
      master-name="only">
         <fo:region-body
      region-name="xsl-region-body"
            margin="0.7in" />
      <fo:region-before
            region-name="xsl-region-before"
            extent="0.7in" />
         <fo:region-after
      region-name="xsl-region-after"
           extent="0.7in" />
      </fo:simple-page-master>
   </fo:layout-master-set>

   <fo:page-sequence master-reference="only" format="A">
      <fo:flow flow-name="xsl-region-body">
         <fo:block >Some base content, containing an inline warning,
           <fo:inline >Warning: </fo:inline>Do not touch blue paper,
           a fairly straightforward piece requiring emphasis
          <fo:inline font-weight="bold">TEXT</fo:inline>, and
           some instructions which  require presenting in a different
           way, such as <fo:inline font-style="italic">Now light
           the blue paper</fo:inline>.
         </fo:block>
      </fo:flow>
   </fo:page-sequence>
</fo:root>
```

Zdrojový kód 34: Ukázka základní specifikace stránky v rámci XSL-FO. [47] [upraveno]

Procesory XSL-FO

Procesorů je celá řada. Existují procesory pro konvertování XSL-FO do podoby RTF. Dále XSL-FO zahrnuje částečnou podporu pro sázecí systém TeX (knihovna maker). Poté jsou zde konvertory, které převádějí formátovací objekty do podoby PDF, které jsou předmětem této práce. [42] [47]

FOP

FOP (Formatting Objects Processor) je formátovací procesor, který je navržen pro práci s XSL-FO specifikací. Vznikl v rámci projektu The Apache FOP project a jeho stvořitelem byl James Tauber. Jedná se o open-source koncipovaný jako Java aplikace. Je velice oblíbený a jeho vývoj probíhá dodnes. [42] [48]

XEP

Primární funkcionalitou XEP procesoru je převádění XSL-FO dokumentu do podoby PDF (stejně jako u FOP procesoru). Personální nebo akademická verze jsou volně k dispozici, verze pro komerční použití je zpoplatněna. Tento procesor je napsaný v jazyce Java a lze ho použít v prostředí, které zahrnuje JDK/JRE. XEP vznikl jako projekt společnosti RenderX a je hodně využívaný v podnikových sférách. [47] [49]

6 Ukázka generovacího workflow

V přiložených souborech, které jsou součástí této práce, je k dispozici ukázka generovacího workflow. Vstupní data pocházejí z testovacího prostředí Swagger editoru, jenž představují obchod s domácími mazlíčky. Data jsou volně k dispozici na následující adrese: https://petstore.swagger.io/v2/swagger.json. Obchod s domácími mazlíčky, resp. definovaná specifikace, se skládá ze 3 základních objektů: uživatel, mazlíček a objednávka. Každý z objektů má k dispozici svou vlastní sadu cest a příslušných metod. Jednotlivé metody zahrnují definici (záleží na vybrané metodě, zda definici může zahrnovat) produkovaných/konzumovaných dat, parametrů, příslušné responses nebo autorizační metody.

Zmíněné schéma je zpracováno v rámci generovacího workflow. Jinými slovy je vykonáno několik transformačních a formátovacích procesů, pomocí kterých vznikne uživatelská dokumentace, jejímž obsahem bude zmíněné schéma. Jednotlivé zpracovací procesy jsou rozebrány v následujících podkapitolách.

V přiložených souborech jsou k nalezení stažená vstupní data, složka výstupních dat, nástroje a informace o testovacím prostředí, které zahrnuje informace o operačním systému, použitých technologiích a editorech. Součástí je soubor `run.bat`, který po spuštění vygeneruje všechny příslušné výstupy. Ve složce s nástroji jsou jednotlivé transformační/formátovací fáze rozděleny do příslušných *.bat* souborů.

6.1 Swagger procesor

Jako první se ve workflow spouští přiložený soubor `swagger_processor_run.bat`, který zahájí tento procesor. Procesor představuje skriptovací soubory (psané v Node JS), které převádějí prvky obsažené v Open API na DocBook elementy, atributy, procesní instrukce apod. V těle procesoru dochází k přeskupení Open API sktruktury do struktury jiné, která je vhodnější pro zpracování. Dále dochází k nahrazení referencí jejich destinacemi, jenž jsou tvořeny zpravidla definovanými strukturami schémat. Poté je výsledná struktura transformována do DocBook struktury, která je obsažena v XML dokumentu. Procesor je navržen pro zpracování souboru ve formátu JSON a je kompatibilní pro verze Open API 2.0 a 3.0.0.

6.2 Transformace XSLT

Tato fáze zahrnuje transformační šablony a saxon procesor. Jsou zde použity originální DocBook XSL transformační šablony a jejich příslušné modifikace. Díky těmto modifikacím jsou finální výstupy vzhledově a strukturově upraveny tak, aby vizuální stránka Open API byla na odpovídající úrovni. Takto stavěný systém šablon je použit i pro následující transformaci XSL-FO a originální šablony je možné získat z následující adresy: https://sourceforge.net/projects/docbook/files/docbook-xsl/1.78.0/. Spuštěním přiloženého souboru: `saxon_html_run.bat` je inicializován saxon procesor, který převede DocBook strukturu na HTML strukturu dle definovaných transformačních šablon a vznikne tak webová stránka nesoucí Open API manuál. Ve zdroji je k dispozici externí CSS soubor, který je k finálnímu výstupu překopírován.

6.3 Transformace XSL-FO

Tato fáze je velice podobná transformaci XSLT, neboť je využito stejného procesoru s názvem saxon, u kterého se liší použité parametry. XSL-FO používá předdefinovaných transformačních šablon, avšak výstup těchto šablon je rozdílný. Transformuje se zde DocBook struktura do podoby formátovacích objektů. Výsledkem je tedy další XML dokument nesoucí strukturu, jenž je validní ke schématu formátovacích objektů. Tento soubor je užitečný pouze jako definice finálního vyobrazení daných prvků. Nese koncovku *.fo* a je zpracován v následující fázi.

6.4 Formátování XSL-FO

Poslední fáze je zahájena pomocí přiloženého `fop_run.bat` souboru. Je zde využito procesoru FOP, který slouží ke konvertování formátovacích objektů do podoby tisknutelných formátů. V tomto případě se bude jednat o formát PDF. Procesor využívá několika Java knihoven a definovaných fontů písma. Všechny náležitosti jsou součástí přiložených souborů.

7 Výsledky

V této kapitole se nachází několik ukázek finálních výstupů. Jedná se o obrázky, které zachycují podobu několika vybraných komponent Open API dokumentace. Je zde zahrnut HTML i PDF výstup. Oba výstupy jsou součástí přiložených souborů této práce.

V následujících dvou ukázkách je znázorněna metoda *PUT* pro definovanou cestu */pet*. Součástí jsou zdrojové informace nesoucí údaje o konzumovaných a produkovaných formátech v rámci této metody. Jinými slovy se jedná o formáty, které je možné použít v rámci HTTP request volání (Consumes), nebo které je možné obdržet v rámci HTTP response (Produces). Dále je součástí metoda autorizace, jenž zde představuje odkaz směřující na kapitolu, která zahrnuje informace o možném způsobu autorizace.

PUT /pet

Abstract

Update an existing pet

Resource Information

Consumes	application/json, application/xml
Produces	application/xml, application/json
Authorization	petstore_auth

Obrázek 3: Ukázka zdrojových informací metody *PUT* v rámci vygenerovaného HTML výstupu. [zdroj vlastní]

/pet

Update an existing pet

Resource Information

Consumes	application/json, application/xml
Produces	application/xml, application/json
Authorization	petstore_auth

Obrázek 4: Ukázka zdrojových informací metody *PUT* v rámci vygenerovaného PDF výstupu. [zdroj vlastní]

Následující ukázka se týká objektu typu uživatel a příslušné metody *POST*. Z ukázky je zřejmé, že zobrazená tabulka nese parametry, které jsou součástí HTTP request volání. V těle volání bude uveden parametr nesoucí název: „body“. Červená hvězdička indikuje, že tento parametr je povinný a musí být uveden při volání této metody. Málo výrazný popisek: „BODY“ udává, že daný parametr se bude nacházet v těle HTTP requestu. Datovým typem těla volání je objekt. Ve třetím sloupci s popisem je informace o tom, že daná metoda slouží k vytváření objektu uživatel. Dále je naznačen ukázkový tvar těla requestu. V tomto případě se jedná o JSON, přičemž všechny klíčové atributy jsou vygenerovány na základě Open API specifikace, jenž je součástí komponent. Přiřazené hodnoty jsou výchozími hodnotami generovanými skrze Swagger procesor. Tyto hodnoty jsou vygenerovány na základě příslušných datových typů. Příkladem může být klíčový atribut „id“, jehož datovým typem v rámci Open API je celé číslo. V procesoru je pro celé číslo definována výchozí hodnota: „0“ a tudíž je použita pro generování této ukázky.

Parameters

Name	Data Type	Description

body * BODY	Object	Created user object
		
		{
		"id": 0,
		"username": "string",
		"firstName": "string",
		"lastName": "string",
		"email": "string",
		"password": "string",
		"phone": "string",
		"userStatus": 0
		}

Obrázek 5: Ukázka parametrů metody *POST* v rámci vygenerovaného PDF výstupu. [zdroj vlastní]

V následující ukázce jsou znázorněny responses, které se týkají objektu objednávka a příslušné metody *GET*. Jinými slovy se jedná o návratové hodnoty, které jsou součástí HTTP response. Součástí je status „200“, který je označen jako: „úspěšná operace“. V příslušném popisu je dále vygenerováno tělo metody. Toto tělo je zahrnuto v HTTP response, neboli tato informace je po úspěšném volání obdržena. Toto tělo je generováno stejným způsobem jako v předešlé ukázce, avšak je možné si povšimnout, že zde v jednom případě nebyla použita výchozí hodnota. Jedná se o klíčový atribut „status“, jehož hodnoty jsou definovány jako konkrétní výčet textových řetězců. V rámci zpracování Swagger procesoru byly jednotlivé výčtové hodnoty vyňaty a pospojovány pomocí svislých čar. Zajímavým klíčovým atributem může být: „datum expedice“, jehož definovaným datovým typem je *dateTime*. V rámci zpracování došlo k použití výchozí hodnoty tohoto datového typu, která představuje datum generované ukázky. Zbylé dva statusy: „400“ a „404“ se dají označit za neúspěšné, neboť v jednom případě je navrácena zpráva o nevalidním zadaném ID a v druhém případě o nenalezené objednávce.

Responses

Code	Description
200	successful operation **Example 7. Response body** `{ "id": 0, "petId": 0, "quantity": 0, "shipDate": "2019-04-16T17:25:30.039Z", "status": "placed|approved|delivered", "complete": true }`
400	Invalid ID supplied
404	Order not found

Obrázek 6: Ukázka responses metody *GET* v rámci vygenerovaného HTML výstupu. [zdroj vlastní]

8 Závěr

Tato práce pojednává o využití moderních metod, které jsou využity k tvorbě uživatelské dokumentace, jejíž obsahem je Open API. Pod pojmem moderní metody se skrývá koncept single source publishing, kde je editace pouze jednoho primárního zdroje hlavním kritériem. Na základě tohoto zdroje jsou vytvářeny exporty do jiných formátů, s kterými se již stýká koncový uživatel. Kromě editace primárního zdroje, další výhodou tohoto konceptu může být lokalizace. Primární zdroj může být vícejazyčně překládán, navíc může být využito exportovací fáze k různým optimalizacím z pohledu lokalizace. Pokud by bylo využíváno pro překlad primárního zdroje externí lokalizační agentury, je možné zamýšlet, že první překlad daného manuálu/dokumentace bude mít nejvyšší náklady. Poté se při každé aktualizaci budou překládat pouze dané změny, rozdíly mezi zastaralou a aktuální verzí. Jelikož primární zdroj by byl zapsán v přívětivém formátu (XML), tak nalezení daných rozdílů by bylo z technického hlediska považováno za snadné. Lokalizační agentury využívají znalostní báze. Tím se myslí databáze, v níž figurují často překládané slova, výrazy, fráze apod. Znalostní báze jsou vytvářeny v rámci dané oblasti/společnosti. Tímto způsobem je možné docílit nižší nákladovosti, neboť obsah takovéto databáze se nepřekládá. Kdyby se daná společnost držela již zmíněného konceptu single source publishing a využívala by metody primárního zdroje, poté by bylo možné často využívané/překládané pojmy, výrazy a fráze dohledat a poskytnout dané výskyty lokalizační agentuře.

Byl zmíněn koncept single source publishing a následná lokalizace. Koncept výrazně snižuje nákladovost a eliminuje duplicity. Avšak zmíněné moderní metody přinášejí výhody i pro samotné editory. Řada společností si v dnešní době s uživatelskou dokumentací lidově řečeno „hlavu neláme“. Poté vznikají dokumentace prostřednictvím textového editoru Microsoft Word, generované dokumentace pomocí nástrojů jako Javadoc (dokumentace zdrojového kódu) nebo je dokumentace zcela opomíjena či vytěsněna (outsourcing). Výhody pro samotné editory mohou být především využití sémantického aparátu, možnosti pokročilého strukturování dokumentu (přehlednost), rozsáhlé možnosti úprav primárního zdroje (flexibilita). Z hlediska využití sémantiky, kterou se rozumí, že daná struktura odpovídá danému obsahu, je zde dbáno na to, aby datový obsah byl co nejvíce „vypovídající“. Což opět zlepšuje autorovu orientační schopnost a úroveň porozumění.

Nastíněny byly výhody ohledně samotné struktury dat a výhody ohledně flexibilních možnostech editace. V tomto bodě již je možné sestavit dokumentační tým, zahájit návrhy primárních zdrojů (uživatelská dokumentace, developerská příručka, ...) a vytvořit týmové workflow, které bude zaměřené na aktualizace daných dokumentů (editace - korektura - lokalizace - korektura - distribuce). Avšak tato práce je z části zaměřena na problematiku Open API. Proč tomu tak je? Snahou je z nastíněného procesu zcela vyloučit editora a vytvořit tak zcela automatizovaný proces. Open API představuje standard s pevně danou strukturou, jenž se opírá o specifikaci. Jelikož se takto daná struktura dá označit za „stálou“, neboli je využíváno pouze daných komponent,

tak je poté možné strukturu transformovat do podoby primárního dokumentu.

Pokud bude zohledněno vše výše uvedené, vznikne automatizovaný proces. Účastníkem tohoto procesu bude vývojář, který bude definovat konfigurační soubor Open API. Konfigurační soubor bude využíván implementovaným REST API prostředím na daném serveru. Server poté bude nabízet REST API služby koncovým uživatelům. Avšak koncoví uživatelé si nejsou vědomi všech nabízených REST API služeb/metod, proto může vývojář na základě vytvořeného konfiguračního souboru sestavit uživatelskou dokumentaci (nabízených REST API služeb), kterou může uživatelům nabídnout. Proto je poté vývojář jediným účastníkem při tvorbě REST API dokumentace. Avšak je potřeba dbát na to, aby byly řádně vyplněny popisy (description) jednotlivých komponent, neboť popisy jsou základem výstupní dokumentace.

9 Seznam použité literatury

1. **TODD, Fredrich.** REST API Tutorial. *Learn REST: A RESTful Tutorial.* [Online] RestApiTutorial.com, 2019. [Citace: 24. Březen 2019.] https://www.restapitutorial.com/.
2. **salesforce.com, inc.** REST API Developer Guide - api_rest.pdf. *REST API Developer Guide.* [Online] 4. Prosinec 2018. [Citace: 23. Březen 2019.] https://resources.docs.salesforce.com/216/latest/en-us/sfdc/pdf/api_rest.pdf.
3. **Oracle Corporation.** docs.oracle.com. *REST API Getting Started Guide.* [Online] 2019. [Citace: 23. Březen 2019.] https://docs.oracle.com/cloud/latest/otmcs_gs/OTMRA/OTMRA.pdf.
4. **SmartBear Software.** Swagger. *About Swagger Specification.* [Online] 2019. [Citace: 30. Březen 2019.] https://swagger.io/docs/specification/about/.
5. **GitHub, Inc.** OAI/OpenAPI-Specification. *GitHub.* [Online] 2019. [Citace: 30. Březen 2019.] https://github.com/OAI/OpenAPI-Specification.
6. **GRUENBAUM, Peter.** Documenting RESTful APIs Using Swagger. [Online] 26. Leden 2018. [Citace: 30. Březen 2019.] http://sdkbridge.com/tc/tcworkbook.pdf.
7. **APIs You Won't Hate.** OpenAPI.Tools. [Online] 2019. [Citace: 30. Březen 2019.] https://openapi.tools/.
8. **SmartBear Software.** Swagger Editor. [Online] 2019. [Citace: 31. Březen 2019.] https://swagger.io/tools/swagger-editor/.
9. **Microsoft Corporation.** OpenAPI.NET. *GitHub.* [Online] 2019. [Citace: 31. Březen 2019.] https://github.com/Microsoft/OpenAPI.NET.
10. —. openapi-lint. *Visual Studio | Marketplace.* [Online] 2019. [Citace: 31. Březen 2019.] https://marketplace.visualstudio.com/items?itemName=mermade.openapi-lint.
11. **GitHub, Inc.** linter-swagger. *ATOM.* [Online] 2019. [Citace: 31. Březen 2019.] https://atom.io/packages/linter-swagger.
12. **BRADLEY, NEIL.** *XML: kompletní průvodce.* Praha : Grada, 2000. ISBN 80-7169-949-7.
13. **HAROLD, RUSTY, ELLIOTE a MEANS, SCOTT.** *XML v kostce.* Praha : Computer Press, 2002. ISBN 80-7226-712-4.
14. **MARCHAL, BENOIT.** *XML v příkladech.* Praha : Computer Press, 2000. ISBN 80-7226-332-3.
15. **CASTRO, ELIZABETH.** *XML pro World Wide Web.* Praha : SoftPress, 2001. ISBN 80-86497-07-0.
16. **STURM, JAKE.** *Developing XML solutions.* Washington : Microsoft Press, 2000. ISBN 0-7356-0796-6.
17. **RAY, Erik T.** *Learning XML.* Cambridge : Mass.: O'Reilly, 2001. ISBN 0-59600-046-4.
18. **The Apache Software Foundation.** The Apache Xerces™ Project - Charter. [Online] The Apache Software Foundation, 1999. [Citace: 16. Únor 2019.] http://xerces.apache.org/charter.html.
19. **CLARK, James.** NSGMLS. [Online] 2001. [Citace: 16. Únor 2019.] http://www.jclark.com/sp/nsgmls.htm.
20. —. SP - System identifiers. [Online] 2000. [Citace: 16. Únor 2019.] http://www.jclark.com/sp/sysid.htm.
21. **International Organization for Standardization.** Formal System Identifier Definition Requirements (FSIDR). [Online] 1997. [Citace: 16. Únor 2019.] http://140.120.7.21/DSSSL/hytime/n1920/html/clause-A.6.html.
22. **IBM.** *XML4J Information.* [Online] 1999, 2000. [Citace: 17. Únor 2019.] http://www.cs.sfu.ca/CourseCentral/470/qyang/XML4J-3_1_1/docs/html/index.html.
23. **The Apache Software Foundation.** *Distributing XML4C.* [Online] 2001. [Citace: 17. Únor 2019.] https://www-01.ibm.com/software/htp/tpf/pubs/xml4ic/xml4c351/html/faq-distrib.html.
24. **VLIST, Eric van der.** *XML Schema.* California : O'Reilly, 2002. ISBN 0-596-00252-1.
25. **World Wide Web Consortium.** XML Schema Tutorial. [Online] 1999-2019. [Citace: 3. Březen 2019.] https://www.w3schools.com/xml/schema_intro.asp.
26. **Refsnes Data.** w3schools.com. *XSD Restrictions/Facets.* [Online] 2019. [Citace: 16. Březen 2019.] https://www.w3schools.com/xml/schema_facets.asp.
27. **FOX, Jonny.** *Regex tutorial — A quick cheatsheet by examples.* [Online] 2017. [Citace: 16. Březen 2019.] https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285.
28. **Refsnes Data.** w3schools.com. *SD Complex Elements.* [Online] 2019. [Citace: 16. Březen 2019.] https://www.w3schools.com/xml/schema_complex.asp.
29. **KOSEK, Jiří.** *XML schémata.* [Online] 10. Listopad 2014. [Citace: 17. Březen 2019.] https://www.kosek.cz/xml/schema/.

30. **Refsnes Data.** w3schools.com. *XML Schema include Element.* [Online] 2019. [Citace: 23. Březen 2019.] https://www.w3schools.com/xml/el_include.asp.
31. —. w3schools.com. *XML Schema import Element.* [Online] 2019. [Citace: 23. Březen 2019.] https://www.w3schools.com/xml/el_import.asp.
32. **tutorialspoint.** XPath - xpath_tutorial.pdf. *www.tutorialspoint.com.* [Online] 2012. [Citace: 23. Březen 2019.] https://www.tutorialspoint.com/xpath/xpath_tutorial.pdf.
33. **interval.cz, Redakce.** Základy jazyka XPath. *interval.cz.* [Online] 4. Duben 2004. [Citace: 23. Březen 2019.] https://www.interval.cz/clanky/zaklady-jazyka-xpath/.
34. **KOSEK, Jiří.** Stručný úvod do tvorby a zpracování dokumentů. *DocBook.* [Online] 7. Květen 2007. [Citace: 23. Březen 2019.] https://www.kosek.cz/xml/db/.
35. **WALSH, Norman; MUELLNER, Leonard.** DocBook: The Definitive Guide. *tdg.docbook.org.* [Online] 11. Říjen 2006. [Citace: 23. Březen 2019.] https://tdg.docbook.org/tdg/4.5/docbook.html. ISBN 156592-580-7.
36. **BOENDER, Ferry.** Really quick guide to DocBook. *www.electricmonk.nl.* [Online] 27. Červenec 2003. [Citace: 23. Březen 2019.] https://www.electricmonk.nl/docs/really_quick_guide_to_docbook/really_quick_guide_to_docbook.pdf.
37. **O'Reilly & Associates, Inc.** tdg.docbook.org. *filename.* [Online] 2003. [Citace: 23. Březen 2019.] https://tdg.docbook.org/tdg/4.5/filename.html.
38. **Free Software Foundation, Inc.** www.gnu.org. *nXML Mode.* [Online] 2018. [Citace: 23. Březen 2019.] https://www.gnu.org/software/emacs/manual/html_node/nxml-mode/.
39. **FANG, Lungang.** lgfang.github.io. *Editing XML in Emacs.* [Online] 5. Duben 2016. [Citace: 23. Březen 2019.] http://lgfang.github.io/mynotes/emacs/emacs-xml.html.
40. **SyncRO Soft SR.** www.oxygenxml.com. *XML Editor.* [Online] 2019. [Citace: 23. Březen 2019.] https://www.oxygenxml.com/xml_editor.html.
41. **XMLmind Software.** XMLmind XML Editor. *What is XMLmind XML Editor?* [Online] 20. Únor 2019. [Citace: 23. Březen 2019.] http://www.xmlmind.com/xmleditor/what_is_xxe.html.
42. **HOLZNER, Steven.** *XSLT: příručka internetového vývojáře.* Praha : Computer Press, 2002. ISBN 80-7226-600-4.
43. **TIDWELL, Doug.** *Mastering XML Transformations.* místo neznámé : O'Reilly Media, 2008. ISBN 978-0596527211.
44. **Saxonica.** Our Products. *SAXONICA.* [Online] 2019. [Citace: 4. Duben 2019.] https://www.saxonica.com/products/products.xml.
45. **The Apache Software Foundation.** The Apache Xalan Project. *Apache Xalan Project.* [Online] 2014. [Citace: 4. Duben 2019.] http://xalan.apache.org/index.html.
46. **LAPEYRE, Deborah Aleyne a USDIN, B. Tommie.** Introduction to XSL-FO Concepts. *Mulberry Technologies, Inc.* [Online] Leden 2006. [Citace: 4. Duben 2019.] http://www.mulberrytech.com/papers/Intro2XSL-FO/Intro2XSL-FO.pdf.
47. **PAWSON, Dave.** *XSL-FO.* Sebastopol, California : O'Reilly, 2002. ISBN 0-596-00355-2.
48. **The Apache Software Foundation.** The Apache FOP Project. *The Apache XML Graphic Project.* [Online] 2016. [Citace: 6. Duben 2019.] https://xmlgraphics.apache.org/fop/.
49. **RenderX, Inc.** XEP Engine. *The Future of Your Document Is Here.* [Online] 2016. [Citace: 6. Duben 2019.] http://www.renderx.com/tools/xep.html.
51. **Free Software Foundation, Inc.** www.gnu.org. *GNU Emacs documentation - GNU Project.* [Online] 2015. [Citace: 23. Březen 2019.] https://www.gnu.org/software/emacs/documentation.html.
52. **SmartBear Software.** Basic Structure | Swagger. *Swagger.* [Online] 2019. [Citace: 30. Březen 2019.] https://swagger.io/docs/specification/basic-structure/.
53. **Mulberry Technologies, Inc.** Introduction to XSL-FO Concepts. *Mulberry.* [Online] 2019. [Citace: 31. Březen 2019.] http://www.mulberrytech.com/papers/Intro2XSL-FO/Intro2XSL-FO.pdf.

10 Seznam tabulek

Tabulka 1: Popis REST API metod a jejich doporučených návratových hodnot. [1] [upraveno] 3
Tabulka 2: Popis základní struktury Open API specifikace. 7
Tabulka 3: Typy atributu s popisem. [13] [15] [16] 18
Tabulka 4: Typy výchozí deklarace atributu s popisem. [13] [16] 19

11 Seznam obrázků

Obrázek 1: Deklarace syntaxe jmenného prostoru. [17 str. 39] 38
Obrázek 2: Aplikování jmenného prostoru na XML názvu. [17 str. 38] 38
Obrázek 3: Ukázka zdrojových informací metody *PUT* v rámci vygenerovaného HTML výstupu. [zdroj vlastní] 51
Obrázek 4: Ukázka zdrojových informací metody *PUT* v rámci vygenerovaného PDF výstupu. [zdroj vlastní]51

12 Seznam zdrojových kódů

Zdrojový kód 1: Ukázka obecného tvaru REST API metody logistického cloudu. [3 str. 11] 4
Zdrojový kód 2: Ukázka způsobu volání REST API metody v prostředí Lightning platform. [2 str. 2]................ 4
Zdrojový kód 3: Ukázka deklarace globálního schématu dle Open API specifikace v jazyce YAML. [52] 7
Zdrojový kód 4: Ukázka správného použití kořenového elementu. [14 str. 49].. 11
Zdrojový kód 5: Příklad použití XML entity. [13 str. 19] 13
Zdrojový kód 6: Ukázka použití CDATA. [14 stránky 54-55] 13
Zdrojový kód 7: Ukázka použití procesní instrukce. [13 str. 21]... 14
Zdrojový kód 8: Ukázka použití interního DTD v XML dokumentu. [15 str. 36] 15
Zdrojový kód 9: Ukázka definice atributu pro daný element. [15 str. 50].. 16
Zdrojový kód 10: Ukázka použití atributového typu NMTOKENS. [16 str. 65] [upraveno] 17
Zdrojový kód 11: Ukázka použití atributového typu IDREF. [13 str. 44]... 18
Zdrojový kód 12: Ukázka použití atributového typu NOTATION. [13 str. 46].. 18
Zdrojový kód 13: Ukázka definice entity. [12 str. 47] 19
Zdrojový kód 14: Ukázka definice a použití parametrické entity. [13 str. 52] 20
Zdrojový kód 15: Ukázka nesprávného pořadí elementů. [13 str. 35] 20
Zdrojový kód 16: Ukázka výstupu validačního procesu z parseru nsgmls. [17 str. 23].................................. 23
Zdrojový kód 17: Ukázka vytvoření XML Schéma dokumentu. [24 str. 20] [upraveno] 25
Zdrojový kód 18: Ukázka reference na XSD dokument v XML dokumentu. [25].. 26
Zdrojový kód 19: Ukázka normalizace textu při použití textového typu xs:normalizedString. [24 str. 35]..... 28
Zdrojový kód 20: Ukázka normalizace textu při použití textového typu xs:token. [24 str. 37] 28
Zdrojový kód 21: Ukázka použití neřazeného fasetu enumeration v rámci definování jednoduchého datového typu XML Schema. [26] 32
Zdrojový kód 22: Ukázka použití regulárního výrazu pro omezení jednoduchého typu timeDuration XML Schema. [15 str. 84]... 32
Zdrojový kód 23: Ukázka použití sekvence elementů v rámci komplexních typů XML Schema. [15 str. 95] [upraveno] 33
Zdrojový kód 24: Ukázka použití volby elementů v rámci komplexních typů XML Schema. [15 str. 96] 34
Zdrojový kód 25: Ukázka použití libovolného pořadí elementů v rámci komplexních typů XML Schema. [29] 34
Zdrojový kód 26: Ukázka definice skupiny elementů v rámci komplexních typů XML Schema. [15 str. 98] ... 36
Zdrojový kód 27: Ukázka definice skupiny atributů v rámci komplexních typů XML Schema. [15 str. 111] ... 37
Zdrojový kód 28: Ukázka aplikace implicitního jmenného prostoru "knihovna" v rámci XML Schema. [24 str. 175]... 39
Zdrojový kód 29: Ukázka aplikace DocBook schématu v rámci XML dokumentu. [34] [upraveno] 42
Zdrojový kód 30: Ukázka prvotní deklarace XSL dokumentu. [42 stránky 26,27] [upraveno] 43
Zdrojový kód 31: Ukázka deklarace XSL šablony pro aplikaci šablon od kořenového elementu. [42 str. 112] 44
Zdrojový kód 32: Ukázka deklarace XSL šablony pro daný element, který bude transformován do HTML podoby. [43 str. 26] 44
Zdrojový kód 33: Ukázka deklarace XSL šablony pro daný element, který bude transformován do XSL-FO podoby. [47] [upraveno] 46
Zdrojový kód 34: Ukázka základní specifikace stránky v rámci XSL-FO. [47] [upraveno] 47