UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Datová struktura silniční a železniční sítě

Ondřej Hrnčíř

Bakalářská práce

2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Ondřej Hrnčíř**
Osobní číslo:	**I08058**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Datová struktura silniční a železniční sítě**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce naprogramování aplikace, která bude filtrovat mapových podkladů pouze údaje o silniční a železniční sitích.

1. Požaduje se v rešeržní části práce popsat dostupné zdroje mapových podkladů a jejich formáty z pohledu silniční a železniční sitě.

2. Navrhnout vlastní jednoduchou datovou strukturu silniční a železniční sítě v podobě schématu XML.

3. Do navržené datové struktury z bodu 2 převést silniční a železniční síť z volně přístupného mapového zdroje, například ze serveru Openstreet, pomocí vlastního programu v jazyce Java.

4. Program bude vypracován ve dvou verzích a) pro příkazovou řádku b) z jednoduchým grafickým rozhraním.

5. Minimální povinné spouštecí parametry programu jsou tyto: definice vstupního, výstupního souboru, volba typu sitě.

6. Vstupní i výstupní soubor budou ve formátu XML.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. Neil Bradley,XML kompletní průvodce, 2000, Grada, ISBN 80-7169-949-7

2. Herout, Pavel, Java a XML, Kopp, 2007, ISBN 978-80-7232-307-4

3. Václav Bárta, Programový generátor trendů, bakalářská práce 2011, UPCE-FEI

Vedoucí bakalářské práce: **Ing. Karel Šimerda**
Katedra softwarových technologií

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na mou práci vztahují práva a povinnosti vyplývající ze zákona č. 121/200 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako Školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou, nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 16. 8. 2013 Ondřej Hrnčíř

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce Ing. Karlu Šimerdovi za jeho odbornou pomoc, cenné rady a poskytnuté materiály, které mi pomohly při zpracování bakalářské práce.

Anotace

Cílem této práce je vytvoření vlastní aplikace, která bude sloužit k filtrování jednotlivých vrstev mapových podkladů. Práce bude zaměřena pouze na filtrování silniční a železniční sítě. Zdrojem mapových podkladů pro tuto práci bude použit server openstreetmap.org, který jako jeden z mála poskytuje mapové podklady ve vektorovém formátu bezplatně.

Klíčová slova

GIS, OSM, XML, DOM, SAX, DTD, XSD, node, way, relation

Title

The data stucture of road and railway networks

Annotation

The aim of this work is to create a custom application that will be used to filter out individual layers of map data. Work will focus only on filtering road and rail networks. The source of maps for this work will be used server openstreetmap.org, who is one of the few providing maps in vector format free of charge.

Keywords

GIS, OSM, XML, DOM, SAX, DTD, XSD, node, way, relation

Obsah

Seznam zkratek

ČR	Česká republika
DBF	Data Base File
DGN	Design
DOM	Document Object Model
DTD	Dokument Type Definition
DWG	DraWinG
DXF	Data eXchange Format
ESRI	Enviromental Systems Research Institute, Inc.
GeoJSON	Geographic JavaScript Object Notation
GIS	Geographic information system
GML	Geography Mark-Up Language
GPS	Global Positioning System
GPX	GPS eXchange Format
HTML	HyperText Markup Language
CHKO	Chráněná krajinná oblast
IBM	International Business Machines
IGDS	Intergraph´s Interactive Graphics Design System
ISFF	Intergraph Standard File Format
JAXB	Java Architecture for XML Binding
JOSM	Java OpenStreetMap Editor
JPEG	Joint Photographic Expert Group
JTS	Jednotná trigonometrická síť
JTSK	Jednotná trigonometrická síť katastrální
OSM	Open Street Map
PDF	Portable Document Format
PNG	Portable Network Graphics
PSČ	Poštovní směrovací číslo
ŘSD ČR	Ředitelství silnic a dálnic České republiky
S-42	Pulkovo 1942
SAX	Symple API for XML
SGML	Standard Generalized Markup Language
SHP	shapefile
S-JTSK	souřadnicový systém Jednotné trigonometrické sítě katastrální
StAX	Streaming API for XML
SVG	Scalable Vector Graphics
UCS	Universal Character Set
UTF-8	UCS Transformation Format—8-bit[
UTM	Universal Transverse Mercator
WGS84	World Geodetic System 1984
WKB	Well Known Binary
XML	Extensible Markup Language
XSD	XML Schema Definition

Úvod

Cílem této práce je vytvoření vlastní aplikace, která bude sloužit k filtrování jednotlivých vrstev mapových podkladů. Práce bude zaměřena pouze na filtrování silniční a železniční sítě. Zdrojem mapových podkladů pro tuto práci bude použit server *openstreetmap.org*, který jako jeden z mála poskytuje mapové podklady ve vektorovém formátu bezplatně.

Práce je rozdělena do tří kapitol. První kapitola vymezuje základní pojmy a pojednává o některých dostupných zdrojích mapových podkladů ve vektorovém formátu. Tato kapitola je dále zaměřena na popis jednotlivých formátů, které jsou běžné pro distribuci mapových podkladů.

Druhá kapitola je zaměřena na formát XML, je zde možné nalézt popis struktury XML souboru, strukturu jednotlivých elementů i jejich obsah. V této kapitole je dále zmíněna možnost procházení a validování souborů tohoto formátu. Vzhledem k zaměření práce, je v této kapitole také rozebírán formát OSM, který využívá server *openstreetmap.org* k jeho distribuci. V této kapitole je také rozebrána spojitost mezi OSM a XML formáty.

Poslední kapitola je zaměřena na tvorbu vlastní datové struktury a její obsluhu. V této části jsou také popsány obě verze programu, využívající této datové struktury. První verze je vytvořena pro příkazovou řádku, druhá verze programu je vytvořena s jednoduchým grafickým rozhraním. Aby bylo možné programy zaměřit na silniční a železniční síť, je součástí této kapitoly také popis filtrování datové struktury.

1 Mapové podkady a jejich formáty

1.1 Vymezení základních pojmů

Geografie

Geografie je věda, která se zabývá studiem mnoha jevů na zemi. Například studiem zemského povrchu, krajiny, jejích vzájemných souvislostí a historickým vývojem.

Zdroj:[5]

Kartografie

Kartografie je jednou z vědních disciplín geografie. Název vychází z latinského slova Chartés, jež ve volném překladu znamená rytí či psaní na papír.

Kartografie je tedy věda o zobrazování a zkoumání rozmístění, spojitostí a vzájemných vazeb úkazů přírody a obyvatelstva pomocí zvláštních obrazově znakových prvků, které jsou obecně známy.

Zdroj:[5]

Mapa

ICA definuje pojem mapa jako reprezentaci vybraných materiálních nebo abstraktních znaků území, které se nacházejí na povrchu Země nebo se k zemskému povrchu vztahují, zobrazuje povrch Země obvykle v měřítku a na plochém médiu.

Charakteristické znaky map:

a) oproti realitě je mapa vždy zmenšená a zevšeobecněná,
b) mapa obsahuje vysvětlující informace,
c) mapa vyjadřuje prostorovou, atributovou a časovou informaci o vlastnostech území.

Zdroj:[6]

Trať je ideální cesta mezi dvěma body.

Dráha je reálný obraz trasy pohybu objektu ve skutečných podmínkách. Díky překážkám na trase vzniká odchylka dráhy od ideální tratě. V ideálním případě by se trať a dráha shodovaly, v realitě se to stává velmi zřídka.

Zdroj:[7]

1.1.1 Rastrové a vektorové vyjádření mapových podkladů

Rastr neboli rastrová mřížka je jedním ze základních prvků rastrové grafiky. Rastr se skládá z bodů, jejichž hlavním nositelem informací je barva. Nejčastějším uskupením bodů je rozmístění do čtvercové mřížky. Takovéto rozložení se osvědčilo jako nejefektivnější zejména v informačních technologiích pro práci s grafickým zobrazením. Jednotlivé body této rastrové mřížky se zde nazývají pixely, jsou to nejmenší části vyobrazeného předmětu.

Rastrová grafika využívá rastru jako základního prvku k zachycení informací o obrazu. Vyobrazení se skládá z pixelové mřížky, každý jednotlivý bod obsahuje právě jednu barvu. Rastrová grafika je nejjednodušším způsobem ukládání informací o obrazu. Takto zpracované vyobrazení nese informaci o každém pixelu zvlášť, znamená to tedy, že je velice paměťově náročný. Při práci s větším množstvím dat, je-li to možné, je efektivnější využívat vektorovou grafiku.

Vektor lze definovat jako přímku, která má určitou velikost a daný směr. U vektoru je možno dále stanovit počáteční a koncové body.

Vektorová grafika je využívána k zobrazování obrazových dat, stejně jako grafika rastrová, ovšem neobsahuje zvlášť informace o jednotlivých bodech, ale soubor informací o vektorech. Na těchto vektorech se grafické zobrazení zakládá. K výhodám tohoto zobrazení patří dynamické zobrazení předmětů, které mohou mít kdykoliv změněn tvar a velikost nezávisle na ostatních vektorech. Zpracování dat, která jsou uložena pomocí vektorové grafiky, je mnohem efektivnější, rychlejší a také je nenáročné na paměťový prostor, v porovnání s rastrovou grafikou. Nevýhodou vektorového zobrazení je velká náročnost při získávání dat, jelikož o každém prvku zkoumaného objektu je nutno zaznamenat informace o poloze a dalších jeho vlastnostech. Na rozdíl od grafiky rastrové tedy nestačí objekt pouze vyfotografovat či naskenovat.

Rastrové mapy si lze vyobrazit jako obrazy složené z pravidelných či nepravidelných částí a buněk, které tvoří určitý tvar. Abychom rastr mohli definovat, je nutné stanovit počátek souřadnicového systému a velikost krokové vzdálenosti na jednotlivých osách, tato vzdálenost určuje podrobnost, neboli rozlišení zobrazované mapy.

Vektorové mapy jsou složeny z vektorů a bodů, které lze pro jednoduchost představovat jako uzly a hrany, pomocí kterých je definován každý objekt mapy.

Objekty jsou skládány z tzv. řetězců. Každý řetězec má počáteční a koncový bod, které spojuje množina vektorů. Proto je u vektorů nutno udávat směr, tak aby bylo možné daný řetězec přesně definovat. Pokud daný řetězec má totožný počáteční a koncový bod, nazýváme ho z topologického hlediska polygonem, z geometrického hlediska lze tento řetězec představit jako n-úhelník.

U takto specifikovaných objektů musíme také určovat jejich polohu vůči ostatním objektům. Pokud se jednotlivé objekty překrývají, je také nutno zaznamenat prioritu jejich zobrazování.

Zdroj:[5]

1.2 GIS - Geografický informační systém

GIS je systém, který slouží k získávání, ukládání, analýzu a vizualizaci dat, která souvisí s povrchem Země. Tato data nazýváme geodata. GIS umožňuje pomocí dostupných softwarových a hardwarových prostředí zpracovávat modely částí Zemského povrchu. Tímto způsobem vytvořený GIS model lze využít například pro evidenci katastru nemovitostí, předpověď počasí, určování záplavových zón, plánování a výstavbu různých technologických objektů a komunikací.

Mapové vrstvy GIS

Geodata vyjadřující stejné téma se sdružují a ukládají do mapových vrstev. Jednotlivými vrstvami může být například síť silnic, vodstvo, nadmořská výška a jiné.

Takovýmto dělením geodat do mapových vrstev se usnadňuje analýza a zpracování dat.

Zdroj:[8]

1.3Zdroje mapových podkladů

Vzhledem k tomu, že využívání rastrové grafiky pro zpracování dat ve výpočetní technice je nevýhodné, popis mapových podkladů se bude dále týkat pouze mapových zdrojů, které poskytují své mapové podklady převážně ve vektorové grafice.

1.3.1 Mapy.cz, s. r. o.

Jedním ze zdrojů mapových podkladů je například firma Mapy.cz, s. r. o., kterou lze nalézt na adrese *http://firma.mapy.cz*. Tato firma pro účely GIS státní i komerční sféry nabízí vektorové vrstvy ze svých mapových podkladů. Nabízené datové vrstvy jsou pozemní komunikace, veřejná doprava, hranice, využití půdy, vodstvo, výškopis, topografie, zájmové body, texty nebo také turistická značení. Tyto vrstvy jsou vhodné pro doplnění nebo aktualizaci všech GIS aplikací nebo územně analytických podkladů.

Zde je možno sivybrat jakoukoliv vrstvu samostatně nebo i více vrstev v balíku dat, dle požadavku zákazníka.

Data jsou poskytována v měřítku 1:10.000, mapy využívají souřadnicový systém UTM, WGS-84, JTSK nebo S-42. Hlavními používanými formáty jsou SHP či DXF, firma dle dohody může dodat i jiný formát.

Firma Mapy.cz, s. r. o. neposkytuje volně dostupné mapové podklady ve vektorovém formátu.

Zdroj:[9]

1.3.2 CSmap

Firma CSmap se zabývá GIS, optimalizací dopravy, převodem a transformací dat. Firma dále poskytuje komplexní služby softwarových řešení, vývoje aplikací na zakázku, zajištění potřebných mapových podkladů, dále poskytuje profesionální služby zahrnující konzultace, školení nebo zpracování analýz.

Na stránkách firmy *CSmap.cz*, v sekci mapy a data, jsou ke stažení vektorové mapy České i Slovenské Republiky. Dle potřeb zákazníků lze na objednání vytvořit i mapové podklady Evropy a celého světa.

Na stránkách jsou dostupné balíky, například administrativní členění ČR, které je vhodné pro sociodemografické, statistické, marketingové a obchodní analýzy. Další balík nabízí mapu PSČ, mapová sada obsahuje regiony a body poštovních směrovacích čísel ČR. Další nabízený balík zobrazuje kupní sílu obyvatel, tato mapa zobrazuje údaje o počtu obyvatel a indexů kupní síly obyvatel v dané obci. Pro účel této práce jsou z nabídky této firmy zajímavé mapy*Route 100* a *Mapy měst a obcí*.

Route 100 je topologickou vektorovou mapou v měřítku 1:100 000. Jedná se o vysoce kvalitní mapu ČR, která je přesná a je proto vhodná pro využití při určování tras a plánování cest. Mapa je zpracována v těchto mapových vrstvách: kompletní silniční síť do úrovně III. třídy silnic; zpevněné, lesní a polní cesty; železnice s nádražími a zastávkami; zastavěné plochy; lesy, zeleň a CHKO; vodní toky a vodní plochy. Tato vektorová mapa je dodávána v souřadnicovém systému WGS84 nebo dle přání zákazníka.

Mapy měst a obcí jsou rozsáhlou kolekcí velmi podrobných map měst v měřítku 1:10.000. Tyto mapy jsou dodávány ve vektorové nebo rastrové verzi. Vektorové mapy měst poskytují kvalitní a přesný podklad například pro řešení dopravních situací v rámci daného města či tvorbu analytických dat. Mapy jsou standardně dodávány v souřadnicovém systému WGS84, nebo JTSK, S-42, UTM a jiných dle přání zákazníka.

Na stránkách firma poskytuje obdobné mapy také pro Slovenskou Republiku.

Zdroj:[10]

1.3.3 CEDA - Central European Data Agency, a. s.

Tato firma se zabývá GIS, zpracováním digitálních mapových podkladů a prostorových databází, digitální kartografií a navigací, také zpracovává dynamické dopravní informace. Firma poskytuje mapové soubory ve vektorových i rastrových formátech.

CEDA poskytuje mapové podklady ČR v mnoha variantách, které jsou detailněji popsány na webových stránkách firmy*ceda.cz*. Mapy jsou poskytovány v souřadnicových systémech S-JTSK nebo WGS84, S-42 ve formátu dat SHP nebo v DBF formátu, či ESRI Shape File. V měřítku 1:10 000v 15 až 23 vrstvách.

Konkrétně mapa České Republiky je poskytována v měřítku 1:150 000, a 14 vrstvách. Mapa poskytuje řešení základních dopravních úloh na silniční síti, tvorbu přehledných mapových výstupů, základní geomarketingové analýzy a mnoho dalších využití. Jednotlivé vrstvy mají připojeny databázové informace, kterými mohou být typ, číslo a třída silnic, název a typ vodních ploch, název obce nebo či části obce. Jednotlivé vrstvy datových sad jsou prodejné i samostatně.

Další ze zajímavých vektorových map na tomto serveru, je mapa Global Network vznikající pod záštitou ŘSD ČR. Tato datová sada kompletně pokrývá území ČR a obsahuje routovatelnou silniční a uliční sít v měřítku 1:10 000 v 15 vrstvách.

Další z vektorových map nabízených touto společností je databáze Tele Atlas (TomTom) Multinet. Jedná se o vysoce přesnou digitální mapu tvořenou silniční sítí s přesností na úroveň ulic a místních komunikací. K jejím vrstvám patří i informace o využití území a zemním pokryvu (např.: zástavba, zeleň, vody, lesy, atd.). Dle tohoto serveru je databáze Multinet ideálním zdrojem pro aplikace orientované na síťové a prostorové analýzy. Tuto databázi je vhodné používat například pro výrobce a provozovatele systémů pro sledování vozidel, výrobce personálních navigátorů, provozovatele dopravy, poskytovatele logistických a dopravně-telematických služeb a systémů, správce dopravní infrastruktury, záchranné sbory a složky krizového řízení a asistenční služby. Přesnost mapy je neustále kontrolována a opravována terénními pracovníky nebo automobily pro mobilní mapování. Aktualizace této mapy probíhá každé tři měsíce.

Zdroj:[11]

1.3.4 Marushka

Mapový aplikační server *Marushka.cz* představuje prostředek pro publikaci a využívání dat GIS pomocí internetu. Přístup k mapovým podkladům probíhá přes stejnojmennou aplikaci Marushka. Nedílnou součástí je také aplikace Marushka design, která umožňuje v intuitivním prostředí konfiguraci a správu konkrétní mapové sady. Tato aplikace má přístup k datovým skladům, ze kterých jsou poskytovány mapové podklady v rastrové i vektorové formě. Technologie Marushka pracuje s datovým formátem WKB.

Mezi další podporované vektorové formáty patří Open GIS Consortium GML, dále Intergraph/Bentley DGN, ESRI Shape File formát SHP a formát Topografic GPX.

Jednou z organizací, která využívá technologii Marushka, je Český úřad zeměměřičský a katastrální, který má na tomto systému založeno nahlížení do katastru nemovitostí ČR.

Zdroj:[12]

1.3.5 OpenStreetMap

OpenStreetMap je projekt zaměřený na vytváření svobodných geografických dat. Jelikož ostatní volně dostupné mapové podklady jsou právně a technicky omezené, vznikl tento projekt, který lidem umožňuje volně pracovat s aktuálními geografickými daty bez

dalších nákladů a omezení. Na rozdíl od výše zmiňovaných mapových podkladů se jedná o celosvětový projekt, který může být upravován registrovanými uživateli.

OpenStreetMap je distribuován za podmínek Open Data Commons Open Database License (ODbL). Data se smí kopírovat, distribuovat, sdělovat veřejnosti a upravovat pouze pokud bude uveden zdroj OpenStreetMap.

Pomocí jednoduchého on-line editoru lze exportovat zobrazenou mapu jako OSM soubor, který je založen na XML technologii. Mapu je dále možno exportovat jako obrázek mapy veformátu PNG, JPEG, SVG či PDF, nebo jako vkladatelný HTML kód. Pro účel této práce bude nadále rozebírán pouze XML formát, který jako jediný vektorově založený formát tento projekt tímto způsobem nabízí.

OpenStreetMap nabízí nástroj JOSM, tento program je desktopovou aplikací sloužící k editaci mapových podkladů dostupných z OpenStreetMap. Základním formátem JOSM pro distribuci map je formát OSM. Aplikace JOSM umožňuje pracovat i s dalšími formáty, například GPX nebo GeoJSON.

Zdroj:[13]

1.4 Mapové formáty

1.4.1 SHP

SHP neboli Shapefile je formát vyvíjený firmou ESRI. Tento formát slouží pro ukládání prostorových dat, společně s těmito daty jsou ukládána i data atributová. Tato prostorová data jsou ukládána po jednotlivých vrstvách. Jednotlivá vrstva je tvořena třemi soubory:

a) `vrstva.shp` - soubor obsahuje geometrické popisy prostorových prvků dané vrstvy,
b) `vrstva.dbf` - soubor obsahuje atributová data vztahující se k prostorovým prvkům dané vrstvy,
c) `vrstva.shx` - soubor obsahuje uložení prostorového indexu nad geometrickými popisy prvků dané vrstvy.

1.4.2 Formát GML - Geography Mark-Up Language

Tento formát byl vytvořen na základě technologie XML. GML byl vyvíjen od roku 1998 Ronem Lake, za mezinárodní normu byl přijat až v roce 2007.

GML dokument je popsán pomocí aplikačního GML schématu. Schéma popisuje strukturu dat a definuje elementy a atributy. GML dokument je složen z GML prvků, které jsou analogií objektů reálného světa.

1.4.3 Formát DGN

DGN, neboli design, je formát vyvíjený firmou Bentley Systems. Tento datový formát má v současné době dvě verze, těmito jsou DGN V7 a DGN V8. Tento formát slouží především k ukládání vektorových dat, ale je možné v něm ukládat i data rastrová nebo popisná. Standardní příponou souboru je DGN.

Formát **DGN V7** vznikl v polovině osmdesátých let minuléhostoletí jako formát DGN, označení DGN V7 získal až s příchodem jeho novější verze DGN V8. DGN V7 byl vyvinut firmou Intergraph na základě formátu ISFF (Intergraph Standard File Format) systému IGDS (Intergraph´s Interactive Graphics Design System).

Formát **DGN V8** byl uveden v roce 2001. Oproti předchozí verzi došlo k překonání omezení například v maximálním počtu vrstev, nebo velikosti souboru. Formát mapuje historii úprav v datech. Tento formát není zpětně kompatibilní se svou předchozí verzí.

1.4.4 Formáty DXF, DWG

DXF je vektorový formát Drawing Interchange File Format vyvíjený firmou AutoDesk, slouží k ukládání dat o 2D a 3D rozměrech.

Vektorová data lze ukládat v tomto formátu v textové nebo binární formě. Binární soubory jsou rozlišovány příponou DXB (Drawing Binary Interchange File Format).

Formát **DWG** je obdobný jako DXF, pouze s tím rozdílem, že se jedná o interní formát firmy AutoDesk.

Zdroj:[14]

1.4.5 Formát GPX

GPX (GPS eXchange Format) je formátem založeným na XML technologii. Lze ho využít k zaznamenání cílových bodů tratě (waypoints), dále drah (tracks) a cest (routes). GPX je otevřený formát, nemusí se tedy hradit licenční poplatky. GPX v sobě zaznamenává údaje o poloze, času a výškových rozdílech. Proto jeho využití nalezneme na mnohých zařízeních pozičních systémů, které vyhodnocují svou polohu například vůči zemskému povrchu. GPX je soubor založený na XML technologii označený příponou GPX. Tento soubor má pevně danou strukturu.

Kořenovým elementem takového souboru je element GPX, který obsahuje informace o typu GPX souboru, hlavičku s metadaty, cílovými body, trasami a cestami. Soubor se skládá z následujících elementů:

a) element `Metadata` obsahuje informace o daném souboru, autorovi a autorských právech,
b) element `Wpt` nese informace o cílových bodech, bodech zájmu, či pojmenovaných bodech mapy,
c) element `Rte`, představuje trať, jedná se o uspořádaný seznam cílových bodů a seznam odbočovacích bodů vedoucích k cílovému bodu,
d) element `Trk` značí dráhu, což je uspořádaný seznam bodů, zaznamenávající cestu,
e) element `Extension` umožňuje rozšíření GPX vlastními elementy,
f) element `Trkseg` udržuje seznam bodů dráhy, jež jsou logicky pospojovány v daném pořadí,
g) element `Copyright` obsahuje informace o licenčních podmínkách,
h) element `Link` umožňuje zaznamenat cesty k externím zdrojům,
i) element `Email` uchovává emailovou adresu z bezpečnostních důvodů rozdělnou zvlášť na id a doménu,
j) element `Person` slouží k identifikaci osoby nebo organizace,
k) element `Pt` označuje geografický bod, který je zastoupený zeměpisnou šířkou a délkou, popřípadě výškovým rozdílem a časovým záznamem,
l) element `Ptseg` zaznamenává posloupnost bodů,
m) element `Bouds` udává rozsah elementu (šířka/délka),
n) element `Latitude/Longitude` obsahuje zeměpisnou šířku a délku bodu v desítkových stupních,

o) element `Degrees` obsahuje desítkové stupně používané pro azimut a směr,
p) element `Fix` zaznamenává typ opravy GPS,
q) element `DgpsStation` určuje diferenciální GPS stanici.

Zdroj:[7]

1.4.6 Formát OSM

Tento formát je založený na XML technologii s kódováním UTF-8. OSM používá jako základní souřadnicový systém WGS84. Data jsou uspořádána v následujících prvcích:

a) uzly jsou body zaznamenávané v daném systému,
b) cesty vyjadřují posloupnost uzlů, vytvářejí tím polylinii či uzavřený polygon,
c) relace je skupina uzlů, cest a jiných relací, které spojuje určitá vlastnost,
d) atributy jsou přiřazovány uzlům, cestám a relacím a upřesňují tak jejich vlastnosti.

Práce je dále zaměřena na zpracování map ze serveru OpenStreetMap. Pro distribuci map z tohoto serveru slouží především formát OSM. Tento formát bude podrobněji popsán v kapitole 2.7 Formát OSM podrobněji.

Zdroj:[13]

1.4.7 Formát GeoJSON

Jedná se o formát, který slouží k zaznamenávání geografických datových struktur. Tento formát umožňuje zaznamenávat informace ve více podobách, například Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon nebo GeometryCollection.

Pozice objektu je zaznamenávána pomocí souřadnicového systému WGS84.

Zdroj:[15]

1.4.8 Formát WKB

Tento formát lze použít pro zaznamenávání souřadnicových dat, které jsou reprezentovány jakobinární tok řetězců. Řetězec se skládá ze tří částí. První část udává způsob ukládání dat (jejich pořadí), druhá část představuje typ objektu, třetí část tvoří souřadnice objektu pomocí hodnot X a Y. Tyto tři části mají pevně danou velikost a tvoří tím řetězec. Posloupností takovýchto řetězců je tvořen celý WKB soubor.

Zdroj:[17]

1.5 souřadnicové systémy

JTSK

Jedná se o systém jednotné trigonometrické katastrální sítě. Tato síť je definována Besselovým elipsoidem a je založena na prvcích sítě dřívější vojenské triangulace (z let 1862-1898), tj. rozměr, orientace a poloha na elipsoidu. Dále je založena na JTS (z let 1920-1957) avyužívá Křovákovo zobrazení. Toto Křovákovo zobrazení je založeno na konformním kuželovém zobrazení v obecné poloze, které v roce 1922 navrhl inženýr Josef Křovák.

S-42

Tento souřadnicový systém využívá Krasovského elipsoid a Gaussovo válcové zobrazení (UTM) 6° poledníkových pásů.

UTM

Jedná se o Gaussovo zobrazení, kde je vyobrazován elipsoid přímo do roviny.

WGS84

Souřadnicový systém WGS-84 používá elipsoid WGS-84 a transverzální válcové zobrazení UTM 6° poledníkových pásů. Tento souřadnicový systém se jeví jako nejvyužívanější.

Zdroj:[16]

2 XML technologie

Jméno XML je zkratka z anglického *Extensible Markup Language*. Tato technologie není ve vlastnictví žádného komerčního subjektu. Byla vyvinuta konsorciem W3C. XML je ideální pro ukládání strukturovaného a semistrukturovaného textu. XML dokument je složen z tagů (značek), elementů a entit, které vymezují jeho části. Dokument XML je založen na logické i fyzické struktuře. Logická struktura se dělí do pojmenovaných jednotek a podjednotek zvaných elementy. Fyzická struktura umožňuje pojmenovat a uložit samostatně části dokumentu, které nazýváme entity. Logická struktura má několik omezení, která musí být striktně dodržena, aby byl XML dokument validní. Validitu lze kontrolovat pomocí parseru (analyzátor). Pro XML neexistuje předdefinovaný seznam elementů, využívání prvků je tedy svobodné. Aby nedošlo k nesprávnému pojmenování elementů, lze pomocí DTD (Dokument Type Definition) mechanismu předdefinovat názvy elementů, které se mohou v konkrétním dokumentu vyskytovat. Na základě tohoto je možné pomocí parseru a těchto DTD dat určit, zda je dokument validní.

XML soubory mohou být editovány pomocí běžných textových editorů, existují pro ně ovšem i speciální editory. Na rozdíl od klasických textových editorů tyto speciální editory využívají DTD mechanismu, aby nedošlo k syntaktickým chybám, ani k chybám ve struktuře dokumentu.

XML je výhodný formát pro výměnu dat mezi aplikacemi. Tento formát může být například využit pro výměnu dat v relačním databázovém systému. XML je navrhnut, aby přímo uchovával metadata, i přesto jsou ukládána odděleně od dokumentů.

2.1 Struktura XML souboru

Struktura XML souboru vychází z HTML a SGML jazyka, na základě toho vzniká jednoduchý a velice účinný mechanismus sloužící k ukládání, zpracování a šíření informací.

XML soubor je tvořen elementy, element je složen z počátečního a koncového tagu, mezi kterými jsou specifikována data. Elementy XML mohou obsahovat další vnořené elementy. Celý dokument vychází z jediného elementu, vytváří tím stromovou strukturu. Každý element musí být plně vnořen do jiného elementu.

Při popisu vztahů mezi elementy užíváme terminologii z rodokmenů. U elementu můžeme určit sourozence, předka a potomky. Z toho vychází, že můžeme u každého elementu určit jednoho rodiče, libovolný počet sourozenců (kromě vlastního dokumentu). Elementy, které nemají potomky, nazýváme listy.

Každý element může obsahovat mimo svého jména i další informace, kterými lze specifikovat jeho obsah. Tyto informace o informacích nazýváme metadata. Tato jsou ukládána v atributech. Každý element může obsahovat i více atributů, každý atribut musí být pojmenovaný.

2.2 Struktura elementů

Element je, jak již bylo zmíněno, ohraničen tagy. Počáteční tag je ohraničen lomenými závorkami (`<počáteční tag>`), ukončující tag, na rozdíl od počátečního tagu, je doplněn za první lomenou závorkou standardním lomítkem (`</koncový tag>`). Ve jménech elementů je nutno rozlišovat velká a malá písmena jsou, tzv. case sensitive. Jelikož se lomené závorky (`<, >`) používají jako oddělovače, neměly by se tedy v datech nikdy objevit. Pokud se tyto znaky v datech musí užívat, je nutné místo nich použít únikový kód `&lt`, nebo `&gt`, které tyto znaky reprezentuje.

2.3 Obsah elementů

Elementy mohou obsahovat data, popřípadě dětské elementy i text. Nazýváme je kontejnerovými elementy. Element neobsahující text, ale pouze dětské elementy, nazýváme elementovým obsahem.

Atributy elementu nesou dodatečné informace o elementu. Atributy nalezneme v počátečním tagu elementu, každý atribut tvoří název atributu a hodnoty atributy. Hodnota atributu je k jeho názvu přiřazena rovnítkem. Hodnota atributu je ohraničena uvozovkami. Častěji užívané uvozovky je možno nahradit apostrofy.

2.4 Deklarace XML

Počátek dokumentu by měl obsahovat speciální tag pro uchování některých důležitých informací o dokumentu. Tento tag může obsahovat tři parametry:

a) parametr `version` obsahuje použitou verzi XML, pokud není informace uvedena, předpokládá se užití verze XML 1.0,
b) parametr `encoding` určuje použitou znakovou sadu dokumentu, pokud není uvedena znaková sada, předpokládá se užití znakové sady UTF-8,

c) parametr `standalone` nese informaci, zda je nutné ke správné interpretaci obsahu dokumentu použít externě definovaných deklarací značkování.

Zdroj:[1]

2.5 Validita souboru

Při validaci se ověřuje, zda XML dokument odpovídá specifikovaným omezením uvedeným ve schématu. Lze říci, že součástí schématu je i popis struktury XML dokumentů.

DTD - Document Type Definition

Pomocí DTD lze definovat XML dokument. Jedná se o velice podporovaný způsob definice dokumentů XML, ale v poslední době se přechází na XML Schema. I přesto je DTD často používaným způsobem definice.

Jednou z nevýhod DTD je nemožnost definovat datové typy a jejich omezení.

XML Schema

XML Schema obsahuje konkrétní popis struktury XML dokumentu, dále specifikuje jednotlivé typy elementů a jejich možnost výskytu. Na základě tohoto lze XML soubor jednoznačně definovat. XML Schema je nástupcem DTD systému.

XML Schema je vhodné užívat pro datově orientované XML dokumenty. XML Schema nabízí několik základních datových typů, například`String`, `Boolean`, `Decimal`, `Float`, `Double`, `DateTime`, `Time`, nebo `Date`.

Zdroj:[4]

2.6 Podpora XML v programovacím jazyku Java

2.6.1 Parser

Parser je nástroj, který slouží v programovacím jazyku Java ke zpracování XML dokumentů. Parser by měl obecně umožňovat načtení a validaci XML dokumentu. Parser dále umožňuje přímý přístup a zpracování elementů, atributů a jejich hodnot.

Parsery dělíme do dvou skupin. První skupinu představují parsery s proudovým zpracováním dat, druhou skupinu představují parsery se stromovou reprezentací dat. Do skupiny s proudovým zpracováním dat patří například parsery SAX a StAX, do skupiny se stromovou reprezentací patří DOM a JAXB.

Zdroj:[3]

2.6.2 DOM

DOM (Document Object Model) načítá XML dokument do paměti celý, data jsou tak reprezentována stromovou strukturou. Díky tomu je DOM vysoce paměťově náročný parser, ale naopak výhodou je libovolný pohyb v této datové struktuře.

Implementace v Javě

Jedním z představitelů DOM parserů je parser od společnosti SUN, pro jeho implementaci v programovacím jazyce Java je nutné užití následujících importů balíčků `org.w3c.dom.*`, `com.sun.xml.tree.*` a také `org.xml.sax.*`. Pro zpracování XML dokumentů poskytuje SUN třídu `XmlDocument`, tato třída obsahuje metodu `CreateXmlDocument`. Tato třída `XmlDocument` poskytuje informace o zpracovaném dokumentu a odkazu na kořenový element, který je základem pro stromovou datovou strukturu, jejímž datovým typem je uzel. Každý uzel obsahuje informace o sobě, typu, názvu a atributech a také informace o pozici. K těmto informacím lze přistupovat pomocí následujících metod:

a) `short getNodeType ()`,
b) `String getNodeName ()`,
c) `String getNodeValue () throws DOMException`,
d) `void setNodeValue (String nodeValue) throws DOMException`,
e) `boolean hasChildNodes ()`,
f) `NamedNodeMap getAttributes ()`,
g) `Document getOwnerDocument ()`.

Aby bylo možné se pohybovat v hierarchii dokumentu, každý uzel obsahuje metody, jejichž návratová hodnota obsahuje odkaz na sousední uzly.

a) `Node getFirstChild ()`,
b) `Node getLastChild ()`,
c) `Node getNextSibling ()`,
d) `Node getPreviousSibling ()`,
e) `Node getParentNode ()`,
f) `NodeList getChildNodes ()`.

Metoda `getFirstChild` zpřístupní prvního potomka daného uzlu. Metoda `getLastChild` zpřístupní posledního potomka daného uzlu, metoda `getChildNodes` slouží pro iteraci všech dětských elementů. Metoda `getNextSibling` a

`getPreviousSibling` slouží k pohybu mezi sourozenci uzlů. Metoda `getParentNode` slouží ke zpřístupnění předchůdce daného uzlu.

Tuto stromovou strukturu je možné upravovat následujícími metodami:

a) `Node removeChild (Node oldChild) throws DOMException`,
b) `Node insertBefore (Node newChild, Node refChild) throws DOMException`,
c) `Node appendChild (Node newChild) throws DOMException`,
d) `Node replaceChild (Node newChild, Node oldChild) throws DOMException`,
e) `Node cloneNode (boolean deep)`.

Metoda `removeChild` slouží k odpojení od jeho umístění ve stromu, odkaz na odstraněný uzel je předán jako návratová hodnota této metody. Metoda `appendChild` slouží k přidávání nových dětských elementů uzlu, do stromů lze také přidat i uzly, které byly dříve metodou `removeChild` odstraněny.

Tento parser od firmy SUN obsahuje také třídu `TreeWalker`, která slouží k procházení struktury. Tato třída obsahuje následující metody:

a) `void TreeWalker (Node startPoint)`,
b) `Node getCure ()`,
c) `Node getNext ()`,
d) `Node getNextElement (String tagName)`.

Metoda `TreeWalker` slouží k předání odkazu na kořenový uzel, metoda `getNext` vrací odkaz na další uzel, pokud je návratová hodnota `null`, je procházení u konce.

Zdroj:[1]

2.6.3 SAX

SAX (Symple API for XML) je běžné rozhraní pro zpracovávání XML dat řízené událostmi. SAX zpracovává data proudově, při jejich načítání dochází k událostem. Události vyvolávají příslušné metody, pomocí těchto metod dochází k postupnému zpřístupnění dat. Při načtení není možnost k návratu k částem, které již byly načteny v průběhu parsování. Pokud je to nutné, parsování dokumentu musí proběhnout znovu od začátku.

Na základě přístupu řízených událostí patří mezi výhody tohoto zpracování malá paměťová náročnost, ze které také vychází vyšší rychlost zpracování.

Implementace v Javě

Jeden z možných parserů je parser od firmy IBM, pro jeho využití musíme importovat balíčky `org.xml.sax.*` a `com.ibm.xml.parsers.*`. Tento parser poskytuje nejen třídu pro zpracování dokumentu, ale také třídu zvanou `ValidatingSAXParser`, který jak název napovídá, umožňuje validaci dokumentu. Třída parser je implementována rozhraním, které definuje následující metody:

a) `void parse (InputSource src) throws SAXException, IOException,`
b) `void parse (String src) throws SAXException, IOException,`
c) `void setDocumentHandler (DocumentHandler doch),`
d) `void setErrorHandler (ErrorHandler errh),`
e) `void setDTDHandler (DTDHandler dtdh),`
f) `void setEntityResolver (EntityResolver entres),`
g) `void setLocale (Locale loc) throws SAXException.`

Metody začínající `set` jsou užívány k aplikaci k registraci tříd. Po registraci aplikace volá jednu z metod `parse`, parser tak začne číst XML data. Pokud parser dojde v dokumentu na nějaký objekt, je pomocí hlavní aplikace vyvolána příslušná metoda objektu. Další čtení dokumentu pokračuje návratem z této metody.

Pro správné fungování aplikace musí být vytvořena třída, která implementuje rozhraní `DocumentHandler`. Pomocí tohoto rozhraní jsou definovány následující metody, ve kterých dochází ke zpracování daných událostí:

a) `void startDocument () throws SAXException,`
b) `void endDocument () throws SAXException,`
c) `void startElement (String name, AttributeList atts) throws SAXException,`
d) `void endElement (String name) throws SAXException,`
e) `void characters (char ch[ ]. int start, int length) throws SAXException,`
f) `void ignorableWhitespace (char ch[ ]. int start, int length) throws SAXException,`
g) `void processingInstruction (String target, String data) throws SAXException,`

h) `void setDocumentLocator (Locator myLoc).`

Při zpracování dokumentu je nejprve volána metoda `startDocument`, jako poslední je volána metoda `endDocument`. Tyto metody lze užít pro obecnou kontrolu, inicializaci proměnných nebo otevírání či uzavírání souborů.

Metoda `startElement` je volána pokaždé, když se při zpracování dat dosáhne počátečního tagu elementu. Tato metoda zpřístupňuje jméno elementu a jeho atributy. Metoda `endElement`, jak již název napovídá, je volána pokud při zpracování dat parser dosáhne koncového tagu.

Metoda `characters` je volána při nalezení textového řetězce, metoda předá textový řetězec pomocí znakového pole.

Metoda `ignorableWhitespaces` je volána pokud parser nalezne řetězec ignorovatelných bílých znaků.

Zdroj:[1]

2.7 Formát OSM podrobněji

Vzhledem k tomu, že OpenStreetMap je jeden z mála distributorů volně přístupných map, bude práce dále zaměřena na zpracování mapových formátů této organizace.

Jak již bylo zmíněno, OpenStreetMap distribuuje své mapy v OSM formátu, který je založený na XML technologii. V následujícím textu bude podrobněji popsán princip, jakým způsobem jsou mapy do tohoto formátu ukládány.

Jako základní kořenový element ve struktuře OSM nalezneme tag s názvem `osm`, v jeho atributech nalezneme informaci o verzi, informace o vzniku mapy a informace o licenčních podmínkách.

Tento OSM element obsahuje tři základní elementy, představované základními datovými primitivy, označovanými jako `nodes`, `ways` a `relations`. Ve volném překladu se jedná o body, předměty a vztahy, ze kterých se mapový formát skládá.

Při pohledu na OSM strukturu zjistíme, že OSM element také obsahuje element označovaný jako `bounds`, tento element představuje hranice zaznamenané mapy ve čtyřech hodnotách. Tyto hodnoty označované jako `minlat`, `minlon`, `maxlat`, `maxlon` představují rozsah zeměpisné šířky a délky zaznamenané mapy.

Zdroj:[13]

2.7.1 Node

Node (bod) je jedním ze základních prvků OSM modelu. Jedná se o geoprostorový bod, který je vyjádřen pomocí zeměpisné šířky a délky, jako nepovinný parametr může obsahovat i nadmořskou výšku. Wikiopenstreetmap uvádí, že jejich mapy v roce 2013 obsahují téměř 2 miliardy bodů.

Struktura bodu se skládá z `id`, které je představováno jedinečným přirozeným číslem. `Id` přiřazené bodu je stávající a neměnné, pokud je bod odstraněn, nesmí být jeho jednoznačný identifikátor již použit, pouze pokud se jedná o obnovení odstraněného bodu.

Zeměpisná šířka a délka (latitude a longtitude) je představována jako datový typ `float`. Pro zeměpisnou šířku se jedná o hodnoty od -90 po 90 s přesností na 7 desetinných míst, hodnoty představují úhel ve stupních, měřený od rovníku. Jedná-li se o kladné číslo, bod leží na severní polokouli, jedná-li se o záporné číslo, bod leží na jižní polokouli. Pro zeměpisnou délku se jedná o hodnoty od -180 po 180 s přesností na 7 desetinných míst, hodnoty představují úhel ve stupních, měřený od nultého poledníku Greenwich. Je-li hodnota kladná, jedná se o bod ležící na východní polokouli, je-li hodnota záporná, leží bod na západní polokouli.

Mezi další informace, které jsou o bodu zaznamenávány, můžeme řadit informace o uživateli, který prvek zaznamenal, ve tvaru `user` a `uid`. `User` je představován datovým typem `String`, `uid` je představováno jednoznačným celočíselným identifikátorem. Ve struktuře bodu dále můžeme najít informace o viditelnosti, označované jako `visible`. `Visible` je datového typu `boolean`, nese tak informaci o tom, zda se bod má či nemá zobrazovat. Jako další z informací nalezneme údaje o verzi, označovaný jako `version`, dále informaci v jaké sadě byly provedeny změny, označované jako `changeset` a v neposlední řadě také o časovém označení, kdy byl bod zaznamenán, označovaný jako `timestamp`.

Bod stejně jako ostatní elementy může obsahovat další dodatečné informace představované tagy, ve kterých je možné specifikovat detailnější informace o bodu.

Zdroj:[13]

2.7.2 Way

Element `way` představuje objekty z reálného života. Každý objekt je na mapě představován jako křivka, která prochází body. Každý objekt je složen z minimálně dvou až dvou tisíc bodů. Každý tento element obsahuje seznam `id` bodů, kterými je definován. Křivky dělíme do dvou skupin na otevřené a na uzavřené. Typickým představitelem uzavřené křivky je například stavba. Otevřenou křivkou můžeme znázornit například cesty a trasy.

Stejně jako u bodů můžeme u elementu `way` nalézt parametry o jeho identifikétoru, viditelnosti, časové známce, verzi a uživateli, který objekt pojmenoval. Pro specifikaci bodů, kterými je objekt určen, slouží vnořené elementy označené jako `nd`, pro specifikaci typu a dalších detailnějších informací slouží vnořený element označený `tag`.

Těmito elementy jsou:

a) `Aerialway` (nadzemní doprava), příkladem může být lanovka, sedačková lanovka, lyžařský vlek, ale i přepravní výtah pro zboží,
b) `aeroway` (letecká doprava), jedná se například o letiště, přistávací plochy pro vrtulníky, hangáry, přistávací dráhy, terminály a další,
c) `amenity` (infrastruktura) jedná se především o místa určená nejen pro obyvatele, ale i pro turisty. Tato místa jsou rozdělena do několika skupin, jedná se o `Suestance` (zdroje potravin – kavárny, restaurace), dále `Education` (vzdělání – knihovny, školy, univerzity), také `Transportation` (doprava – půjčovny kol, autopůjčovny, nabíjecí zařízení pro elektromobily, parkoviště), dále `Financial` (finance – banky, bankomaty, směnárny), dále `Healthcare` (zdravotnictví – nemocnice, lékárny, kliniky), také `Entertainment`, `Arts` and `Culture` (zábava, umění a kultura – kino, divadlo, kluby, studia) a další (lavičky, krypty, policejní stanice, pošty, toalety),
d) `Barrier` (překážky), slouží k popisu potenciálních překážek, například městské hradby, příkopy, ploty, živé ploty, zdi, obrubníky, turnikety),
e) `Boundary` (hranice), mezi které můžeme zařadit administrativní hranice (jedná se například o hraniční území obcí a měst), námořní hranice, přírodní parky, poštovní okrsky apod,
f) `Building` (stavby), hotely, domy, průmyslové podniky, sklady, katedrály, kostely, nemocnice, nádraží, mosty, garáže, kůlny, chaty a další,

g) `Craft` (řemesla) mezi která lze řadit kováře, včelaře, loděnice, elektrikáře, sklenáře, hodináře,

h) `Emergency` (pohotovost), používá se k popisu umístění nouzových zařízení, například ambulance, požární stanice, hydranty, nouzové telefony, sirény,

i) `Geological`, jedná se o geologický popis oblasti,

j) `Highway` (silnice), slouží k popisu cest, chodníků, dálnic, silnic I. II. a III. tříd, cyklostezek, kruhových objezdů a dalších,

k) `Historic`, slouží k popisu archeologických lokalit, vraků, zřícenin, bojišť,pevností, památníků a dalších,

l) `Landuse`, nese informace o využití krajiny, jedná se o lesy, skládky, louky, pastviny, lomy, stavební pozemky, vinice a další,

m) `Leisure`, používá se označení rekreačních a sportovních zařízení, zahrad, golfových hřišť, dětských hřišť, rekreačních bazénů, plováren, a dalších,

n) `ManMade` jedná se o uměle vytvořené objekty v krajině, například štoly, majáky, komíny, vodárny, větrné mlýny, průmyslové stavby a další,

o) `Military`, využívá se pro zobrazení půdy a objektů využívaných k armádním účelům (bunkry, kasárna, střelnice, vojenská letiště),

p) `Natural` (přírodní), používá se pro popis fyzikálních vlastností půdy (bažiny, louky, pláže, vodní plochy, útesy, ledovce, hřebeny hor),

q) `Office`, zobrazuje kanceláře sloužící k podnikání, například právnické kanceláře, vládní budovy, cestovní kanceláře, telekomunikační společnosti, a další,

r) `Places` (místa), nese informace o městech, obcích, kontinentech, ostrovech, atd.,

s) `Power`, používá se k mapování elektrické a distribuční sítě (elektrárny, vedení vysokého napětí, rozvodny, trafostanice),

t) `Publictransport`slouží k popisu veřejné dopravy,

u) `Railway` (železnice) zobrazuje všechny druhy železnic (metra, tramvaje, lanové dráhy, železniční koleje pro osobní i nákladní dopravu),

v) `Route` (cesta) se používá k popisu cest například cyklostezek, sjezdovek,

w) `Shop` slouží k popisu prodejen a obchodních řetězců,

x) `Sport` poskytuje informace o umístění například hřišť, stadionů, závodišť, plaveckých bazénů, tenisových kurtů, posiloven,

y) `Tourism` (cestovní ruch) slouží k popisu objektů vyhledávaných turisty,

z) `Waterway` (vodní díla), slouží k popisu řek, kanálů, loděnic, přehrad a ostatních vodních děl.

Pro všechny tyto objekty lze formou tagů upřesnit informace o adrese, popisu,emailu, kontaktních údajích, zdrojích webových stránek, plochách pokrytí, přístupnosti, časovém omezení, apod.

Zdroj:[13]

2.7.3 Relations

Relations, neboli vztahy, jsou posledním základním stavebním prvkem v OSM struktuře. Skládají se z jednoho nebo více tagů anebo jednoho nebo více bodů. Tyto vztahy slouží k definování logických nebo geografických závislostí mezi jednotlivými objekty.

Zdroj:[13]

2.7.4 Validování OSM struktury

Aby bylo možné ověřit správnost map, nejen co se týče syntaxe, ale i sémantiky, nalezneme proto v JOSM funkci validace mapy. Tato funkce v JOSM byla dříve jako plugin, nyní tuto funkci v tomto programu nalezneme mezi základními funkcemi. Tato funkce JOSM zamezuje duplicitě bodů i objektů, stará se o nekompletní objekty apod.

Pro ověření mapových formátů bez užití JOSM lze využít DTD či XSD (XML schéma), pomocí těchto mechanismů je definována struktura a formát OSM.

XSD (XML schéma) soubory pro validaci OSM map pomocí XML schéma jsou přístupné v různých verzích na adrese *wiki.openstreetmap.org/wiki/XSD*.

XML schéma na této adrese jsou velmi rozsáhlé. Jako následující ukázka bude použita část kódu, která slouží pro ověření bodů.

```
<xs:elementname="node">
 <xs:complexType>
  <xs:complexContent>
   <xs:extensionbase="osmBasicType">
    <xs:sequence>
     <xs:elementref="tag"maxOccurs="unbounded"minOccurs="0">
     </xs:element>
    </xs:sequence>
    <xs:attributename="lat"type="xs:string"use="required"/>
    <xs:attributename="lon"type="xs:string"use="required"/>
    <xs:attributename="action"type="xs:string"use="optional"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

Zdroj:[13]

Pro validaci pomocí **DTD** systému lze nalézt DTD soubory na adrese *wiki.openstreetmap.org/wiki/DTD.*

Pro názornost je níže zobrazena starší verze DTD souboru pro validaci z výše uvedeného odkazu.

```
<!ELEMENT osm ((node|relation|way)*)>
<!ATTLIST osm version   (0.5) #REQUIRED>
<!ATTLIST osm generator CDATA #REQUIRED>
<!ELEMENT node (tag*)>
<!ATTLIST node id        CDATA #REQUIRED>
<!ATTLIST node lat       CDATA #REQUIRED>
<!ATTLIST node lon       CDATA #REQUIRED>
<!ATTLIST node visible   CDATA #IMPLIED>
<!ATTLIST node user      CDATA #IMPLIED>
<!ATTLIST node timestamp CDATA #IMPLIED>
<!ELEMENT way (tag*,nd,tag*,nd,(tag|nd)*)>
<!ATTLIST way id        CDATA #REQUIRED>
<!ATTLIST way visible   CDATA #IMPLIED>
<!ATTLIST way user      CDATA #IMPLIED>
<!ATTLIST way timestamp CDATA #IMPLIED>

<!ELEMENT nd EMPTY>
<!ATTLIST nd ref         CDATA #REQUIRED>
<!ELEMENT relation ((tag|member)*)>
<!ATTLIST relation id        CDATA #REQUIRED>
<!ATTLIST relation visible   CDATA #IMPLIED>
<!ATTLIST relation user      CDATA #IMPLIED>
<!ATTLIST relation timestamp CDATA #IMPLIED>

<!ELEMENT member EMPTY>
<!ATTLIST member type (way|node|relation) #REQUIRED>
<!ATTLIST member ref  CDATA  #REQUIRED>
<!ATTLIST member role CDATA  #IMPLIED>
<!ELEMENT tag EMPTY>
<!ATTLIST tag  k         CDATA #REQUIRED>
<!ATTLIST tag  v         CDATA #REQUIRED>
```

Zdroj:[13]

3 Vlastní datová struktura silniční a železniční sítě

Pro účely této práce byla vytvořena vlastní programovádatová struktura, která je s kompletním programem v příloze této práce.

Tato struktura vychází z OSM formátu a je zaměřena na silniční a železniční síť. Jak již bylo zmíněno, základním stavebním prvkem jsou prvky `nodes`, `ways` a `relations`. Prvek `nodes` (**body**) je v této struktuře představován třídou `Point`, dále prvek `ways` (**předměty**) je v této struktuře představován třídou `Object`. Prvky `relations` (**vztahy**) nejsou z pohledu této práce podstatné, proto z této datové struktury byly vypuštěny.

3.1 AbstractList

Jako základní datová struktura, která je využita pro hlavní datovou strukturu, je použit vlastní jednosměrný, lineární, cyklický, generický seznam. Tento seznam je realizován jako třída `AbstractList<T>`, která implementuje rozhraní `IAbstracList<T>`. V tomto rozhraní nalezneme následující metody, sloužící zejména pro naplňování tohoto seznamu:

a) `void zrus ()`, tato metoda slouží k vyprázdnění celého seznamu,
b) `boolean jePrazdny ()`, pokud seznam neobsahuje žádný prvek, tato metoda vrací `true`, pokud se v seznamu nachází jeden a více prvků, vrací `false`,
c) `void vlozPrvni (T obj)`, tato metoda vkládá prvek na první pozici v seznamu, pokud je první pozice obsazena, každý prvek seznamu je posunut na následující pozici a tím má možnost tento prvek být vložen jako první, tato metoda automaticky rozšiřuje seznam o jeden prvek,
d) `void vlozPosledni (T obj)`, tato metoda automaticky rozšiřuje seznam o jeden prvek, který je vložen na poslední pozici seznamu,
e) `T zpristupniPrvni ()`, návratovou hodnotou této metody je odkaz na první prvek v seznamu,
f) `T zpristupniPosledni ()`, návratovou hodnotou této metody je odkaz na poslední prvek v seznamu,
g) `T odeberPrvni ()`, pokud seznam obsahuje alespoň jeden prvek, tato metoda automaticky snižuje seznam o první prvek, který je následně vrácen jako návratová hodnota, pokud je návratovou hodnotou `null`, znamená to, že seznam je již prázdný a nelze z něj žádný prvek odebrat,

h) `T odeberPosledni ()`, pokud seznam obsahuje alespoň jeden prvek, tato metoda automaticky snižuje seznam o poslední prvek, který je následně vrácen jako návratová hodnota, pokud je návratovou hodnotou `null`, znamená to, že seznam je již prázdný a nelze z něj žádný prvek odebrat,
i) `int getPocet ()`, při volání této metody, metoda vrací aktuální počet prvků v seznamu,
j) `IIterator vytvorIterator ()`, návratovou hodnotou této metody je iterátor, který slouží k procházení tohoto seznamu.

Jelikož je tato kolekce vytvořena pro obecné použití, nebudou všechny tyto metody pro účel této práce využity.

K procházení prvků této kolekce slouží iterátor, který je jako podtřída součástí třídy `AbstractList<T>` a je implementován rozhraním `IIterátor<T>`, který obsahuje následující metody:

a) `boolean hasNext ()`, tato metoda patří mezi standardní metody iterátoru, slouží k zjištění informace o tom, zda se iterátor může posunout na další prvek, pokud je iterátor na posledním prvku, tato metoda vrací `false`,
b) `T getNext ()`, tato metoda patří mezi standardní metody iterátoru, slouží k posunutí iterátoru na další prvek v kolekci. Její návratová hodnota je odkaz na nově posunutý prvek, pokud je iterátor na konci kolekce, tato metoda vrací `null`,
c) `void vlozNaslednika (T obj)`, tato metoda slouží pro vkládání prvků do kolekce, prvek je vkládán jako následující prvek za prvek, na který iterátor ukazuje. Tato metoda automaticky rozšiřuje seznam o jeden prvek,
d) `T zpristupniAktualni ()`, tato metoda slouží k zpřístupnění prvku, na který iterátor ukazuje,
e) `T zpristupniNaslednika ()`, tato metoda slouží k zpřístupnění následujícího prvku, na který iterátor ukazuje, na rozdíl od metody `getNext()`, která také zpřístupňuje následující prvek, tato metoda iterátor neposouvá,
f) `T odeberAktualni ()`, pokud seznam obsahuje alespoň jeden prvek, tato metoda automaticky snižuje počet prvků, a to tak, že odebírá prvek, na který iterátor ukazuje. Jelikož iterátor ukazuje na prvek, který je z kolekce odebírán, je iterátor automaticky posunut na prvek předchozí. Problém by mohl nastat při odebírání prvku

prvního, protože se jedná o seznam cyklický, v tomto případě je iterátor posunut na poslední prvek seznamu tak, aby při volání metody `getNext()` byl zpřístupněn první prvek seznamu. Iterátor tak sice ukazuje na poslední prvek seznamu, ale v tomto případě přivolání metody `hasNext ()`, tato metoda vracítrue. Návratovou hodnotou této metody je právě odebíraný prvek,

g) `T odeberNaslednika ()`, pokud seznam obsahuje alespoň jeden prvek, tato metoda automaticky snižuje počet prvků, a to tak, že odebírá prvek, který je následníkem prvku, na který iterátor ukazuje. Pokud seznam neobsahuje žádný prvek, návratovou hodnotou je `null`,

h) `void vynulujIterator ()`, tato metoda slouží k nastavení iterátoru tak, aby byl připraven procházet kolekci od začátku.

3.2 Třída Point

Bod z OSM formátu je v této vlastní datové struktuře představován třídou`Point`. Jeho proměnné jsou:

a) `private long id`, tato proměnná slouží k zaznamenání jedinečného identifikátoru bodu,

b) `private float lat`, tato proměnná slouží k zaznamenání pozice bodu, konkrétně jeho zeměpisné šířky bodu,

c) `private float lon`, tato proměnná slouží k zaznamenání pozice bodu, konkrétně jeho zeměpisné délky,

d) `private String user`, tato proměnná slouží k zaznamenání názvu uživatele, který tento bod zaměřil,

e) `private long uid`, tato proměnná slouží k zaznamenání jedinečného identifikátoru uživatele, který daný bod zaměřil,

f) `private int version`, tato proměnná nese informaci o verzi.

Pro nastavování a získávání hodnot těchto proměnných slouží příslušné metody (`set` a `get`), které tato třída obsahuje. Ostatní informace o bodu jsou v OSM struktuře nepovinné a pro účel této práce nepodstatné, proto nebyly do této struktury zahrnuty.

3.3 Třída Object

Mezi privátní proměnné této třídy patří:

a) `private long id`, tato proměnná slouží k zaznamenání jedinečného identifikátoru objektu,
b) `private String user`, tato proměnná slouží k zaznamenání názvu uživatele, který tento objekt zmapoval,
c) `private long uid`, tato proměnná slouží k zaznamenání jednoznačného identifikátoru uživatele, který tento objekt zmapoval,
d) `private int version`, tato proměnná nese informace o verzi,
e) `private String type`, tato proměnná slouží k zaznamenání typu objektu, pro účel této práce je do této proměnné zaznamenáván, pokud se jedná o silniční cesty, řetězec `highway`, pokud se jedná o železniční trať, řetězec `railway`, pro ostatní objekty je automaticky vkládán řetězec `other`,
f) `private String subType`, tato proměnná slouží k bližšímu upřesnění, o který typ železniční nebo silniční cesty se jedná, pro ostatní objekty je automaticky vkládán řetězec `other`,
g) `private IAbstractList<point> points = new AbstractList<point> ()`, tato kolekce slouží k ukládání bodů, kterými je objekt definován.

Pro nastavování a získávání hodnot některých těchto proměnných slouží příslušné metody (`set` a `get`), které tato třída obsahuje.

Další metody, které tato třída obsahuje, jsou následující:

a) `public void addPoint (point point)`, tato metoda slouží k přidávání bodů do její privátníkolekce `points`, při volání této metody se předávaný bod automaticky vkládá na konec této kolekce. Protože se nepředpokládá, že by více procesů zároveň přistupovalo k této kolekci, byl pro procházení kolekce vytvořen iterátor uvnitř třídy,
b) `public boolean hasNext ()`, při prvním volání této metody se automaticky v této třídě vytvoří iterátor, který pomocí této a následující metody umožní projití všech bodů, které objekt obsahuje. Pokud iterátor je na konci kolekce, vrací tato metoda `false`,

c) `public point getNext ()`, tato metoda slouží k postupnému zpřístupnění procházených bodů iterátorem,
d) `public void resetIterator ()`, tato metoda slouží k posunutí iterátoru na začátek kolekce tak, aby bylo možno body objektů projít od začátku.

3.4 Vlastní datová struktura

Vlastní datová struktura, která je schopna pojmout informace o mapě jako celku z pohledu silniční a železniční sítě, je v tomto případě představována třídou `MyDataStructure`. Tato třída obsahuje následující privátní proměnné a veřejné metody:

a) `private float version`, tato proměnná slouží k zaznamenání verze ukládané mapy,
b) `private String copyright`, tato proměnná obvykle obsahuje informaci jakými autorskými právy je tato mapa vázána,
c) `private String attribution`, tato proměnná obvykle obsahuje webový odkaz na autorská práva,
d) `private String license`, tato proměnná obvykle obsahuje webový odkaz na licenční podmínky,
e) `private float minlat, minlon, maxlat, maxlon` - tyto proměnné slouží k zaznamenání plochy vyjadřované konkrétní mapou,
f) `private IAbstractList<Object> objects = new AbstractList<Object> ()` tato kolekce slouží k ukládání jednotlivých objektů mapy,
g) `private IAbstractList<Point> points = new AbstractList<Point> ()` tato kolekce bodů slouží k dočasnému zaznamenání všech použitých bodů převáděné mapy, po dokončení převodu je tato kolekce automaticky vyprázdněna. Body každé mapy jsou rozděleny mezi jednotlivé objekty tak, aby tyto body byly součástí jednotlivých objektů. Tím je při odstraňování jednotlivých objektů ulehčeno odstraňování bodů, které byly třeba pro vymezení tohoto objektu, tímto je ulehčeno hlavně filtrování jednotlivých vrstev mapy.

Metody této třídy je možné rozdělit do několika skupin. První skupinou jsou metody, které slouží k nastavování a získávání hodnot již zmíněných privátních proměnných této třídy. Jejich podrobnější rozepisování není nutné, protože již podle názvu je zřejmé, k čemu tyto metody slouží. Jedná se o následující metody:

a) `public void setAttribution (String attribution),`
b) `public void setCopyright (String copyright),`
c) `public void setLicense (String license),`
d) `public void setMaxlat (float maxlat),`
e) `public void setMaxlon (float maxlon),`
f) `public void setMinlat (float minlat),`
g) `public void setMinlon (float minlon),`
h) `public void setVersion (float version),`
i) `public String getAttribution (),`
j) `public String getCopyright (),`
k) `public String getLicense (),`
l) `public float getMaxlat (),`
m) `public float getMaxlon (),`
n) `public float getMinlat (),`
o) `public float getMinlon(),`
p) `public float getVersion ().`

Druhou skupinu představují metody pro přidávání jednotlivých bodů do kolekce `private IAbstractList<Point> points`. Z této kolekce jsou následně body ukládány do vytvářených objektů, po dokončení vkládání do této datové struktury je pomocí metody `public void removeTemporaryPoints ()` tato kolekce vyprázdněna.

Převod jednotlivých bodů probíhá vytvořením prázdného bodu pomocí metody `public void CreatePoint ()` v kolekci `points` a následovným doplněním informací o tomto bodě následujícími metodami. Tyto metody nastavují jednotlivé informace poslednímu vytvořenému bodu:

a) `public void setLastPointId (long id),`
b) `public void setLastPointUid (long uid),`
c) `public void setLastPointUser (String user),`

d) `public void setLastPointVersion (int version),`
e) `public void setLastPointlon (float lon),`
f) `public void setLastPointlat (float lat).`

Třetí skupinu představují metody pro přidávání jednotlivých objektů. Jelikož získávání informací o jednotlivých objektech probíhá sekvenčně, je nutné vytvořit prázdný objekt pomocí metody `public void CreateObject ()`, kterému následně budou pomocí níže zmíněných metod, postupně doplňovány jeho informace. Principem těchto metod je nastavení jednotlivých atributů posledního vytvořeného objektu. Jakmile je vytvořen nový objekt, nelze ho již pomocí těchto metod upravovat.

a) `public void setLastObjectId (long id),`
b) `public void setLastObjectSubType (String subType),`
c) `public void setLastObjectType (String type),`
d) `public void setLastObjectUid (long uid),`
e) `public void setLastObjectUser (String user),`
f) `public void setLastObjectVersion (int version),`
g) `public void setLastObjectPoint (Long id).`

Poslední zmíněná `metoda setLastObjectPoint` automaticky vybere,podle předávaného`id`, z kolekce `points`konkrétní bod a přidá ho k naposledy vytvořenému objektu.

Čtvrtá skupina metod je tvořena metodami pro postupné procházení této datové struktury, sloužící zejména pro převedení jednotlivých objektů a bodů zpět do OSM struktury.

a) `public Point getAllPointsSequentially ()`, tato metoda slouží k projití a zpřístupnění všech použitých bodů v této datové struktuře. Opětovným voláním této metody postupně získáme všechny použité body. Pokud je návratová hodnota `null`, byly již předány všechny body,
b) `public Object getAllObjectsSequentially ()`, tato metoda slouží k projití a zpřístupnění všech objektů v této datové struktuře. Opětovným voláním této metody postupně získáme všechny objekty. Pokud je návratová hodnota `null`, byly již všechny objekty předány.

Poslední velice zajímavou metodou této třídy je metoda

```
public void filter (boolean highway, boolean railway).
```

Tato metoda projde všechny objekty této datové struktury a odstraní z ní objekty, které nevyhovují této podmínce

```
(!(((object.getType().equals("railway")) && railway) ||
((object.getType().equals("highway")) && highway))).
```

Tato podmínka je vytvořena tak, aby dle předávaných parametrů byly z kolekce odstraněny nežádoucí objekty. Pokud první parametr této metody je nastaven na `true`, budou odstraněny všechny objekty mimo silničních cest, pokud je druhý parametr nastaven na `true`, budou odstraněny všechny objekty mimo železniční tratě. Pokud jsou oba parametry nastaveny na `true`, budou odstraněny všechny objekty, mimo železniční a silniční sítě. Jestliže oba dva parametry jsou nastaveny na `false`, vznikne prázdná mapa bez jakéhokoliv objektu. Jestliže tato metoda mezi načítáním a exportováním mapy nebude volána, bude mapa obsahovat všechny objekty jako mapa originální, s tím rozdílem, že objekty, které nepatří silniční a železniční síti, budou označeny jako `other`.

3.5 ControlClass

Pro obsluhu vlastní datové struktury (`MyDataStructure`) v programu lze nalézt třídu `ControlClass.java`. Tato třída si vytváří instanci od třídy `MyDataStructure.java` a slouží k její obsluze. V této obslužné třídě nalezneme tři základní metody, které slouží pro načtení, filtrování a ukládání mapy.

Metoda pro načítání

Pro načtení souboru map nám slouží metoda:

```
public void load (String XML_INPUT_FILENAME) throws
ParserConfigurationException, SAXException, IOException.
```

Tato metoda očekává odkaz na mapový soubor, který bude touto metodou načten a převeden do vlastní datové struktury. Odkaz na soubor je očekáván jako textový řetězec (`String`). Tato metoda je ošetřena příslušnými výjimkami. K vlastnímu načítání souboru slouží parser SAX, který je importován pomocí:

```
import javax.xml.parsers.SAXParser,
import javax.xml.parsers.SAXParserFactory.
```

Tento parser pro čtení XML souboru (OSM souboru) využívá Handler, který je jako vnitřní třída součástí třídy `ControlClass`. Třída `Handler` je odvozena od třídy `DefaultHandler`, v této třídě nalezneme metody, které jsou volány, pokud se při načítání

souboru parser narazí na odpovídající objekt, viz kapitola 2.6.3 SAX – implemenentace v javě.

Metody obsažené v Handleru slouží k postupnému naplňování datové struktury. Jedná se zejména o metodu `startElement`, jež zpřístupňuje postupně obsah jednotlivých elementů. Jelikož se OSM struktura skládá z do sobě vnořených předem známých elementů (`OSM`, `bounds`, `node`, `way`, `nd`, `tag`), je metoda `startElement` rozdělena pomocí podmínek do tomu odpovídajících větví.

V OSM struktuře je nutno při načítání respektovat správné vnoření elementů. K tomuto slouží jednoduchý mechanismus čtyř proměnných typu `boolean`, které svým názvem odpovídají příslušným elementům. Tyto proměnné jsou na začátku načítání souboru nastaveny všechny na `false`. Při vstupu do příslušného elementu se příslušná proměnná nastaví na `true` a při opouštění se opět nastaví na `false`. Jednotlivé větve této metody tyto informací využívají. Při ukončení čtení souboru, pokud je datová struktura konzistentní, musí mít tyto proměnné typu `boolean` hodnoty`false`, což je ověřováno jak při volání metody `startDocumnet`, tak hlavně při volání metody `endDocument`. Pokud by některá z těchto proměnných typu `boolean` v těchto případech měla hodnotu `true`, je považováno, že při načítání nastala chyba a datová struktura není konzistentní. V tomto případě je vyvolána výjimka `VerifyError ()`.

Metoda pro filtrování

K filtrování datové struktury slouží následující metoda:

```
public void filter (boolean highway, boolean railway);
```

Předávaným parametrem této metody jsou informace o tom, jaká vrstva mapy bude vyfiltrována. Filtrační schopnosti této metody odpovídají zaměření této práce, a to na železniční a silniční síť. K tomu jsou očekávány dvě informace typu boolean, které je této metodě nutno předat. První informace nese údaj o filtrování silniční sítě, druhá informace o filtrování železniční sítě.

Pokud bude tato metoda volána s parametry (`true`, `true`), budou z datové struktury odstraněny všechny objekty a jejich příslušné body, které nejsou součástí silniční a železniční sítě.

Jestliže při volání metody budou použity parametry (`true`, `false`) budou odstraněny z datové struktury všechny objekty a jejich příslušné body, které nejsou součástí silniční sítě.

Pokud bude metoda volána s parametry (`false`, `true`), budou odstraněny z datové struktury všechny objekty a jejich příslušné body, které nejsou součástí železniční sítě.

Volání metody s parametry (`false`, `false`) se nedoporučuje, protože dojde k odstranění všech objektů i bodů, vznikne tak prázdná mapa.

Pokud mezi načítáním a ukládáním mapy, nedojde k volání této metody, budou všechny objekty, nenáležící silniční ani železniční síti, označeny jako `other`.

Metoda pro uložení

K vytvoření výstupního souboru, který bude obrazem datové struktury, slouží metoda:

```
public void stow (String XML_OUTPUT_FILENAME) throws
IOException, XMLStreamException;
```

Vstupním parametrem je název popřípadě i cesta, která je specifikací k vytvoření výstupního souboru. Tato metoda je ošetřena příslušnými podmínkami, pokud metoda bude volána s parametrem `null`, bude výstupní soubor pojmenován jako `Default.xml`, k vytvoření a zapisování do výstupního souboru je použito

```
XMLOutputFactory.
```

Pro zapisování jednotlivých částí souboru je užit `XMLStreamWriter`, pomocí kterého se postupně volají jeho metody s příslušnými parametry odpovídající OSM struktuře. Nejprve je volána metoda pro vytvoření hlavičky XML dokumentu, upřesňující kódování a verzi, následovně jsou volány metody

```
writeStartElement ()
writeEndElement ().
```

Mezi těmito metodami, jsou pomocí metody `writeAttribute ()` vkládány příslušné atributy.

Po zapsání hlavičky následuje zápis elementu OSM, který obsahuje atributy `version`, `copyright`, `attribution`, `license`, poté následuje zápis elementu `bounds` a atributů definujících vymezený prostor mapy a koncového elementu, příslušejícímu k elementu `bounds`.

Následovně budou do výstupního souboru vkládány pomocí cyklu všechny body, které byly užity ve struktuře mapy. Vkládání bude probíhat pomocí volání metody `writeStartElement` s parametrem `node`, příslušnými atributy a koncovým elementem, který odpovídá elementu `node`.

Poté budou do dokumentu stejným způsobem jako body vkládány objekty z dané struktury. Nakonec stačí zapsat párový element k elementu OSM, zavolat metodu pro vytvoření konce dokumentu a soubor uzavřít.

Pro ověření konzistentnosti struktury zde slouží proměnná `immersion` typu `int`, která je zpočátku nastavena na nulu, při každém volání metody `writeStartElement` je tato proměnná inkrementována a při každém volání metody `writeEndElement` je tato proměnná dekrementována. Pokud by po ukončení zápisu do souboru tato proměnná obsahovala jinou hodnotu než nula, nelze obsah souboru považovat za správný. Pro tento případ bude vyvolána výjimka `IndexOutOfBoundsException ()`.

3.6 Vlastní XML schéma

Aby bylo možné výstupní soubor validovat, bylo vytvořeno vlastní XML schéma, které slouží pouze pro tento účel.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="osm">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="bounds">
     <xs:complexType>
      <xs:attribute type="xs:float" name="minlat"/>
      <xs:attribute type="xs:float" name="minlon"/>
      <xs:attribute type="xs:float" name="maxlat"/>
      <xs:attribute type="xs:float" name="maxlon"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="node" maxOccurs="unbounded">
     <xs:complexType>
      <xs:attribute type="xs:long" name="id"/>
      <xs:attribute type="xs:float" name="lat"/>
      <xs:attribute type="xs:float" name="lon"/>
      <xs:attribute type="xs:string" name="user"/>
      <xs:attribute type="xs:long" name="uid"/>
      <xs:attribute type="xs:int" name="version"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="way" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="nd" minOccurs="2" maxOccurs="unbounded">
        <xs:complexType>
         <xs:attribute type="xs:long" name="ref"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="tag">
        <xs:complexType>
         <xs:attribute type="xs:string" name="k"/>
         <xs:attribute type="xs:string" name="v"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:long" name="id"/>
      <xs:attribute type="xs:string" name="user"/>
      <xs:attribute type="xs:long" name="uid"/>
      <xs:attribute type="xs:int" name="version"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute type="xs:float" name="version"/>
   <xs:attribute type="xs:string" name="copyright"/>
   <xs:attribute type="xs:string" name="attribution"/>
   <xs:attribute type="xs:string" name="license"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

3.7 Program verze CMD

Jedna z verzí programu byla vypracována pro příkazovou řádku. Program očekává tři spouštěcí parametry. Jako první parametr je očekáván název (popřípadě i cesta) ke vstupnímu souboru. Jako druhý parametr je očekáván název (popřípadě i cesta) k výstupnímu souboru. Pro oba případy platí, že pokud nebude uvedena cesta, ale pouze název, bude se umístění považovat v adresáři, ve kterém se program nachází. Pokud bude program spouštěn z vývojového prostředí, bude se jednat o adresář projektu. Třetím parametrem, který je zde očekáván, je číslo v rozmezí od jedné do tří. Tento parametr slouží k nastavení filtrování mapové struktury. Pokud má parametr hodnotu jedna, budou ze struktury odstraněny všechny objekty a jejich příslušné body mimo silniční síť. Pokud bude mít parametr hodnotu dva, budou ze struktury odstraněny všechny objekty a jejich příslušné body, mimo železniční síť. Pokud bude mít parametr hodnotu tři, budou odstraněny všechny objekty a jim příslušející body, mimo silniční a železniční síť.

3.8 Program verze GUI

Základní funkce tohoto programu můžeme rozdělit do tří částí. První část slouží k načítání souboru, k tomu slouží tlačítko označené**Načti**, druhá část programu slouží k filtrování načtených dat. K tomu slouží dvě zaškrtávací políčka a tlačítko **Filtruj**. Třetí funkcí programu je uložení zpracovaných dat do výstupního souboru, k tomu slouží jedno editační pole a tlačítko **Ulož.**

K obsluze tohoto programu slouží čtyři tlačítka, k tomu dvě zaškrtávací políčka a jedno editovatelné pole. První tlačítko**Načti** nejprve vyvolá nové okno pro výběr vstupního souboru, vstupní soubor je očekáván ve formátu OSM nebo XML. Po vybrání souboru a jeho potvrzení následuje proces pro načtení tohoto souboru do datové struktury. Jelikož načítání rozsáhlejších map může trvat několik desítek sekund, po které by se program mohl jevit jako nefunkční, je proto vyvoláno nové nemodální okno, které uživatele upozorňuje k vyčkání. Po skončení načtení je toto okno automaticky uzavřeno. Tlačítko **Filtruj** slouží k odstranění nežádoucích vrstev mapy, které jsou představovány datovou strukturou. K nastavení filtru slouží dvě zaškrtávací políčka, pokud je příslušné políčko vybráno, bude příslušná dopravní síť ve struktuře ponechána. Třetí tlačítko**Ulož** převede datovou strukturu do souboru, který bude pojmenován dle obsahu editačního pole. Defaultní název pro výstupní soubor je `map_out.xml`. Pokud by toto pole obsahovalo prázdný řetězec, bude výstupní soubor pojmenován jako `map_out.xml`. Čtvrté tlačítko **Konec** slouží k ukončení aplikace.

3.9 Adresář projektu

Ve zdrojovém adresáři projektu (`MyMapFiler`), který je přílohou této práce, lze nalézt několik zdrojových balíčků. V jeho podadresáři `src` (`sourcepackages`) nalezneme pět základních balíčků, do kterých je tento projekt rozdělen:

a) balíček `abstractList` - tento balíček obsahuje třídy sloužící k vytvoření a obsluze vlastního lineárního seznamu,
   - `AbstractList.java`, viz kapitola 3.1
   - `IAbstractList.java`, viz kapitola 3.1
   - `IIterator.java`, viz kapitola 3.1

b) balíček `DataStructure` - tento balíček obsahuje třídy k vytvoření a obsluze vlastní datové struktury,
   - `Point.java`, viz kapitola 3.2
   - `Object.java`, viz kapitola 3.3
   - `MyDataStructure.java`, viz kapitola 3.4
   - `ControlClass.java`, viz kapitola 3.5

c) balíček `CMD` - tento balíček obsahuje verzi programu pro příkazovou řádku,
   - `CMD.java`, viz kapitola 3.7

d) balíček `GUI`- tento balíček obsahuje verzi programu s jednoduchým grafickým rozhraním.
   - `JFrame`, viz kapitola 3.8

e) balíček `XML_schema` - tento balíček obsahuje vlastní XML schema,
   - `schema.xsd`, viz kapitola 3.6

Závěr

Cílem této práce bylo vytvoření vlastní aplikace, která bude sloužit k filtrování jednotlivých vrstev mapových podkladů. Práce byla zaměřena pouze na filtrování silniční a železniční sítě. Zdrojem mapových podkladů pro tuto práci byl použit server *openstreetmap.org*, který jako jeden z mála poskytuje mapové podklady ve vektorovém formátu bezplatně.

Vlastní datová struktura byla postavena na vlastním, jednosměrném, cyklickém, generickém, lineárním seznamu. Vývoj programu byl nejprve postaven na datové struktuře `ArraiList`, která je součástí balíčku `java.util.ArrayList`. K procházení datové struktury byl užit iterátor z `java.util.iterator`. Problémy nastaly při odstraňování jednotlivých částí struktury. Na takto navržené datové struktuře se nedařilo efektivně odstraňovat nežádoucí části struktury při procházení pomocí iterátoru. Následovně byla celá struktura přepracována a postavena na vlastním lineárním seznamu, který využívá vlastnoručně naprogramované metody iterátoru. Tento iterátor byl doplněn zejména metodou pro bezproblémové odstraňování prvků při iterování datové struktury.

Při testování programu vytvořeného k této práci byla zjištěna velká časová náročnost na zpracování rozsáhlejších mapových podkladů. Časová náročnost mnohdy přesahovala více jak několik desítek sekund. Mohlo by se zdát, že tato neefektivnost je způsobena užitím vlastního lineárního seznamu. Ovšem efektivnost programu ve vývojové verzi, která byla založena, na struktuře `java.util.ArrayList`, nebyla bez výrazných změn.

Dle OpenStreetMap měl být použit datový typ pro jedinečný identifikátor bodu Integer, ale při převádění některých map nastal problém, protože hodnoty překračovaly limity dané tímto datovým typem, proto v této práci byl pro jednoznačný identifikátor bodů i objektů užit datový typ Long. Tímto byl problém odstraněn.

Cíle této práce byly splněny. V první části této práce byly vymezeny základní pojmy a také zde bylo uvedeno seznámení s problematikou map, mapových zdrojů a jejich formátů zejména ve vektorovém formátu. V druhé části byla vymezena problematika XML technologie a její spojitosti s vektorovými mapovými formáty. Ve třetí části byla popsána tvorba a struktura dvou verzí programů, které jsou náplní této bakalářské práce. Zde byla v neposlední řadě také popsána tvorba vlastní jednoduché datové silniční a železniční sítě v podobě schématu XML.

Literatura

1. **BRADLEY, Neil.***XML: kompletní průvodce*. 1. vyd. Praha: Grada, 2000, 537 s. ISBN 80-7169-949-7.

2. **HEROUT, Pavel.***Java a XML*. 1. vyd. České Budějovice: Kopp, 2007, 313 s. ISBN 978-80-7232-307-4.

3. **BÁRTA, Václav.***Programový generátor trendů*. Pardubice, 2011. Bakalářská práce. Univerzita Pardubice Fakulta elektrotechniky a informatiky. Vedoucí práce Ing. Karel Šimerda.

4. **KAŠPAR, Jan.***Tvorba a zpracování souborů XML v Javě*. Pardubice, 2010. Diplomová práce. Univerzita Pardubice Fakulta elektrotechniky a informatiky. Vedoucí práce Ing. Zdeněk Šilar.

5.**SOUČEK, Dominik.***Použití shape files jako standardních formátů mapových dat a jejich využití v praxi*. Univerzita Pardubice, 2012. Bakalářská práce. Univerzita Pardubice. Vedoucí práce Ing. Jaroslav Štroch.

6. **KOZEL, Jiří.***Mapa kritický rozbor definice podle ICA*. 2006. Dostupné z: http://www.fi.muni.cz/usr/richter/lekce/u02rozbordefinicemapy.pdf.

7. **ŠPILAR, Martin.***Emulátor výstupu přijímače GPS ve formátu GPX a NMEA*. Pardubice, 2011. Bakalářská práce. Univerzita Pardubice. Vedoucí práce Ing. Karel Šimerda.

8. **FIEDLER, Jiří.***Geodetické práce: GIS* [online]. 2013 [cit. 2013-07-14]. Dostupné z: http://www.fiedler-geo.cz/gis.

9. ***Mapy.cz:*** *Nabidka vektorovych dat* [online]. 2013 [cit. 2013-07-14]. Dostupné z: http://firma.mapy.cz/nabidka-vektorovych-dat.html.

10. **CSMAP, s. r. o.***CSmap: GIS řešení a mapové podklady pro geografickou analýzu a vizualizaci dat* [online]. Brno, 2008 [cit. 2013-07-14]. Dostupné z: http://www.csmap.cz/gis-a-mapy.

11. ***CEDA:*** *Central European Data Agency, a.s.* [online]. Praha, 2011 [cit. 2013-07-14]. Dostupné z: http://www.ceda.cz/cs/produkty/vektorove-mapy/.

12.***Marushka:*** *mapový server* [online]. Pardubice, 2006 [cit. 2013-07-14]. Dostupné z: http://marushka.geostore.cz/

.

13. ***OpenStreetMap:*** *Otevřená wiki-mapa světa* [online]. 2013 [cit. 2013-07-14]. Dostupné z: http://wiki.openstreetmap.org/wiki/Main_Page.

14. **ŠMÍDOVÁ, Kateřina.***Export dat SGI ISKN do formátu ESRI shapefile a jejich import do RDBMS za pomoci WKT/WKB*. Praha, 2009. Diplomová práce. České vysoké učení technické. Vedoucí práce Ing. Petr Souček, Ph.D.

15**. *GeoJSON*** [online]. 2009 [cit. 2013-07-14]. Dostupné z: http://geojson.org/

16. **JEŽEK, Jan.***Vývoj programového modulu pro převod souřadnic mezi kartografickými zobrazeními*. Praha, 2003. Diplomová práce. České vysoké učení technické v Praze.

17. **ORACLE.***Well-Known Binary (WKB) format* [online]. 2013 [cit. 2013-07-14]. Dostupné z: docs.oracle.com/cd/E17952_01/refman-5.5-en/gis-wkb-format.html

Seznam příloh

Příloha A - AbstractList.java;

Příloha B - IAbstractList.java;

Příloha C - IIterator.java;

Příloha D - Point.java;

Příloha E - Object.java;

Příloha F - MyDataStructure.java;

Příloha G - ControlClass.java;

Příloha H - CMD.java;

Příloha I - JFrame;

Příloha J - schema.xsd.

Příloha A

```
package abstractList;

public class AbstractList<T> implements IAbstractList<T> {

    private Prvek<T> prvni = null;
    private Prvek<T> posledni = null;
    private int pocet = 0;

    public void zrus() {
        prvni = null;
        posledni = null;
        pocet = 0;
    }

    public int getPocet() {
        return pocet;
    }

    public boolean jePrazdny() {
        return (prvni == null);
    }

    private void prvniVlozeni(T obj) {
        Prvek pom = new Prvek(obj, null);
        pocet++;
        prvni = pom;
        prvni.nasledujici = prvni;
        posledni = prvni;
    }

    private Prvek vloz(T obj, Prvek nasledujici) {
        Prvek pom = new Prvek(obj, nasledujici);
        pocet++;
        return pom;
    }

    public void vlozPrvni(T obj) {
        if (jePrazdny()) {
            prvniVlozeni(obj);
        } else {
            Prvek pom = vloz(obj, prvni);
            posledni.nasledujici = prvni;
            prvni = pom;
        }
    }

    public void vlozPosledni(T obj) {
        if (jePrazdny()) {
            prvniVlozeni(obj);
        } else {
            Prvek pom = vloz(obj, prvni);
            posledni.nasledujici = pom;
            posledni = pom;
        }
    }
```

```
public T zpristupniPrvni() {
    if (prvni != null) {
        return prvni.data;
    }
    return null;
}

public T zpristupniPosledni() {
    if (posledni != null) {
        return posledni.data;
    }
    return null;
}

public T odeberPrvni() {
    Prvek<T> smazany = prvni;
    if (jePrazdny()) {
        return null;
    }
    if (prvni == posledni) {
        zrus();
        return smazany.data;
    } else {
        prvni = prvni.nasledujici;
        posledni.nasledujici = prvni;
        pocet--;
        return smazany.data;
    }
}

public T odeberPosledni() {
    Prvek<T> smazany = posledni;
    if (jePrazdny()) {
        return null;
    }
    if (prvni == posledni) {
        zrus();
        return smazany.data;
    } else {
        Prvek<T> p = najdiPredchudce(posledni);
        p.nasledujici = prvni;
        posledni = p;
        pocet--;
        return smazany.data;
    }
}

private Prvek<T> najdiPredchudce(Prvek<T> prvek) {
    Prvek<T> p = prvek;
    while (p.nasledujici != prvek) {
        p = p.nasledujici;
    }
    return p;
}
```

```
private class Prvek<T> {

    T data;
    Prvek<T> nasledujici;

    public Prvek(T data, Prvek<T> nasledujici) {
        this.data = data;
        this.nasledujici = nasledujici;
    }
}

public IIterator<T> vytvorIterator() {
    return new Iterator();
}

public class Iterator implements IIterator<T> {

    Prvek<T> aktualni = null;
    Prvek<T> pom = null;
    private int i = 0;

    public boolean hasNext() {
        return (i < pocet);
    }

    public T getNext() {
        if (hasNext() == false) {
            return null;
        }
        i++;
        if (aktualni == null) {
            aktualni = prvni;
            return prvni.data;
        }
        aktualni = aktualni.nasledujici;
        return aktualni.data;
    }

    public void vynulujIterator() {
        aktualni = null;
        i = 0;
    }

    public T zpristupniAktualni() {
        if (aktualni != null) {
            return aktualni.data;
        }
        return null;
    }

    public T zpristupniNaslednika() {
        return aktualni.nasledujici.data;
    }

    public T odeberAktualni() {
        if (aktualni != null) {
            return odeberNaIndexu(aktualni);
        }
        return null;
    }
```

```
        public T odeberNaslednika() {
            if (aktualni != null) {
                return odeberNaIndexu(aktualni.nasledujici);
            }
            return null;
        }

        private T odeberNaIndexu(Prvek<T> prvek) {
            if (jePrazdny()) {
                return null;
            }
            i--;
            if (prvek.nasledujici == prvek) {
                zrus();
                aktualni = null;
            } else if (prvek == prvni) {
                odeberPrvni();
                aktualni = posledni;
            } else if (prvek == posledni) {
                odeberPosledni();
                aktualni = posledni;
            } else {
                Prvek<T> po = najdiPredchudce(prvek);
                po.nasledujici = po.nasledujici.nasledujici;
                aktualni = po;
                pocet--;
            }
            return prvek.data;
        }

        public void vlozNaslednika(T obj) {
            if (jePrazdny()) {
                prvniVlozeni(obj);
                aktualni = prvni;
            } else {
                Prvek p = vloz(obj, aktualni.nasledujici);
                aktualni.nasledujici = p;
                if (aktualni == posledni) {
                    posledni = p;
                }
                aktualni = p;
            }
        }
    }
}
```

Příloha B

```
package abstractList;

public interface IAbstractList<T> {

    void zrus();

    boolean jePrazdny();

    void vlozPrvni(T obj);

    void vlozPosledni(T obj);

    T zpristupniPrvni();

    T zpristupniPosledni();

    T odeberPrvni();

    T odeberPosledni();

    public int getPocet();

    IIterator vytvorIterator();

}
```

Příloha C

```
package abstractList;

public interface IIterator<T> {

    boolean hasNext();

    T getNext();

    void vlozNaslednika(T obj);

    T zpristupniAktualni();

    T zpristupniNaslednika();

    T odeberAktualni();

    T odeberNaslednika();

    void vynulujIterator();

}
```

Příloha D

```
package DataStructure;

public class Point {
    private long id;
    private long uid;
    private float lat;
    private float lon;
    private String user;
    private int version;

    public Point() {
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public long getUid() {
        return uid;
    }

    public void setUid(long uid) {
        this.uid = uid;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public int getVersion() {
        return version;
    }

    public void setVersion(int version) {
        this.version = version;
    }

    public float getLat() {
        return lat;
    }

    public void setLat(float lat) {
        this.lat = lat;
    }

    public float getLon() {
        return lon;
    }
```

```
    public void setLon(float lon) {
        this.lon = lon;
    }

    @Override
    public String toString() {
        return "point{" + "id=" + id + ", uid=" + uid + ", lat=" + lat + ",
lon=" + lon + ", user=" + user + ", version=" + version + '}';
    }

}
```

Příloha E

```
package DataStructure;

import abstractList.AbstractList;
import abstractList.IAbstractList;
import abstractList.IIterator;

public class Object {
    private long id;
    private long uid;
    private String user;
    private int version;
    private String type = "Other";
    private String subType = "Other";
    private IAbstractList<Point> points = new AbstractList<Point>();

    IIterator<Point> iter = null;
    public void resetIterator(){
        iter = null;
    }

    public boolean hasNext(){
        if(iter==null){
            iter=points.vytvorIterator();
        }
        return iter.hasNext();
    }

    public Point getNext(){
        return iter.getNext();
    }

    public void addPoint(Point point) {
        points.vlozPosledni(point);
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getSubType() {
        return subType;
    }

    public void setSubType(String subType) {
        this.subType = subType;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }
```

```
    public long getUid() {
        return uid;
    }

    public void setUid(long uid) {
        this.uid = uid;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public int getVersion() {
        return version;
    }

    public void setVersion(int version) {
        this.version = version;
    }
}
```

Příloha F

```
package DataStructure;

import abstractList.AbstractList;
import abstractList.AbstractList.Iterator;
import abstractList.IAbstractList;
import abstractList.IIterator;

public class MyDataStructure {

    private float version;
    private String copyright;
    private String attribution;
    private String license;
    private float minlat, minlon, maxlat, maxlon;
    private IAbstractList<Object> objects = new AbstractList<Object>();
    private IAbstractList<Point> points = new AbstractList<Point>();
    IIterator<Object> iter = null;
    IIterator<Object> iter2 = null;
    Object object = null;

    public Point getAllPointsSequentially() {
        if (iter == null) {
            iter = objects.vytvorIterator();
        }
        while (true) {
            if (object == null) {
                if (iter.hasNext()) {
                    object = iter.getNext();
                } else {
                    return null;
                }
            }
            if (object.hasNext()) {
                return object.getNext();
            } else {
                object = null;
            }
        }
    }

    public Object getAllObjectsSequentially() {
        if (iter2 == null) {
            iter2 = objects.vytvorIterator();
        }
        if (iter2.hasNext()) {
            return iter2.getNext();
        } else {
            return null;
        }
    }

    public String getAttribution() {
        return attribution;
    }

    public String getCopyright() {
        return copyright;
    }
```

```
public String getLicense() {
    return license;
}

public float getMaxlat() {
    return maxlat;
}

public float getMaxlon() {
    return maxlon;
}

public float getMinlat() {
    return minlat;
}

public float getMinlon() {
    return minlon;
}

public float getVersion() {
    return version;
}

public void setAttribution(String attribution) {
    this.attribution = attribution;
}

public void setCopyright(String copyright) {
    this.copyright = copyright;
}

public void setLicense(String license) {
    this.license = license;
}

public void setMaxlat(float maxlat) {
    this.maxlat = maxlat;
}

public void setMaxlon(float maxlon) {
    this.maxlon = maxlon;
}

public void setMinlat(float minlat) {
    this.minlat = minlat;
}

public void setMinlon(float minlon) {
    this.minlon = minlon;
}

public void setVersion(float version) {
    this.version = version;
}

public void CreateObject() {
    objects.vlozPosledni(new Object());
}
```

```
    public void setLastObjectId(long id) {
        objects.zpristupniPosledni().setId(id);
    }

    public void setLastObjectSubType(String subType) {
        objects.zpristupniPosledni().setSubType(subType);
    }

    public void setLastObjectType(String type) {
        objects.zpristupniPosledni().setType(type);
    }

    public void setLastObjectUid(long uid) {
        objects.zpristupniPosledni().setUid(uid);
    }

    public void setLastObjectUser(String user) {
        objects.zpristupniPosledni().setUser(user);
    }

    public void setLastObjectVersion(int version) {
        objects.zpristupniPosledni().setVersion(version);
    }

    public void setLastObjectPoint(Long id) {

        IIterator<Point> iter = points.vytvorIterator();
        Point point;
        while (iter.hasNext()) {
            point = iter.getNext();
            if (id == point.getId()) {
                objects.zpristupniPosledni().addPoint(point);
            }
        }
    }

    public void removeTemporaryPoints() {
        points.zrus();
    }

    public void filter(boolean highway, boolean railway) {

        IIterator<Object> it = objects.vytvorIterator();
        Object object;
        while (it.hasNext()) {
            object = it.getNext();
            if (!(((object.getType().equals("railway")) && railway) ||
((object.getType().equals("highway")) && highway))) {
                it.odeberAktualni();
            }
        }
    }

    public void CreatePoint() {
        points.vlozPosledni(new Point());
    }

    public void setLastPointId(long id) {
        points.zpristupniPosledni().setId(id);
    }
```

```
    public void setLastPointUid(long uid) {
        points.zpristupniPosledni().setUid(uid);
    }

    public void setLastPointUser(String user) {
        points.zpristupniPosledni().setUser(user);
    }

    public void setLastPointVersion(int version) {
        points.zpristupniPosledni().setVersion(version);
    }

    public void setLastPointlon(float lon) {
        points.zpristupniPosledni().setLon(lon);
    }

    public void setLastPointlat(float lat) {
        points.zpristupniPosledni().setLat(lat);
    }
}
```

Příloha G

```
package DataStructure;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.Locator;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;

public class ControlClass {

    MyDataStructure mapa;

    public ControlClass() {
        mapa = new MyDataStructure();
    }

    public void load(String XML_INPUT_FILENAME) throws
ParserConfigurationException, SAXException, IOException {
        if (XML_INPUT_FILENAME == null) {
            throw new FileNotFoundException();
        }
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser parser = spf.newSAXParser();
        Handler handler = new Handler();
        XMLReader reader = parser.getXMLReader();
        reader.setContentHandler(handler);
        handler.setMapa(mapa);
        reader.parse(new InputSource(new FileReader(XML_INPUT_FILENAME)));
    }

    public void filter(boolean highway, boolean railway) {
        mapa.filter(highway, railway);
    }
```

```
    public void stow(String XML_OUTPUT_FILENAME) throws IOException,
XMLStreamException {
        if (XML_OUTPUT_FILENAME == null) {
            XML_OUTPUT_FILENAME = "Default.xml";
        }
        FileWriter outputFile = new FileWriter(XML_OUTPUT_FILENAME);
        XMLOutputFactory xmlof = XMLOutputFactory.newInstance();
        XMLStreamWriter writer = xmlof.createXMLStreamWriter(outputFile);

        writer.writeStartDocument("UTF-8", "1.0");
        int immersion = 0;
        Point point;
        Object object;

        writer.writeStartElement("osm");
        immersion++;
        writer.writeAttribute("version",String.valueOf(mapa.getVersion()));
        writer.writeAttribute("copyright",
String.valueOf(mapa.getCopyright()));
        writer.writeAttribute("attribution",
String.valueOf(mapa.getAttribution()));
        writer.writeAttribute("license",
String.valueOf(mapa.getLicense()));
        writer.writeStartElement("bounds");
        immersion++;
        writer.writeAttribute("minlat", String.valueOf(mapa.getMinlat()));
        writer.writeAttribute("minlon", String.valueOf(mapa.getMinlon()));
        writer.writeAttribute("maxlat", String.valueOf(mapa.getMaxlat()));
        writer.writeAttribute("maxlon", String.valueOf(mapa.getMaxlon()));
        writer.writeEndElement();
        immersion--;
        point = mapa.getAllPointsSequentially();

        while (point != null) {
            writer.writeStartElement("node");
            immersion++;
            writer.writeAttribute("id", String.valueOf(point.getId()));
            writer.writeAttribute("lat", String.valueOf(point.getLat()));
            writer.writeAttribute("lon", String.valueOf(point.getLon()));
            writer.writeAttribute("user", point.getUser());
            writer.writeAttribute("uid", String.valueOf(point.getUid()));
            writer.writeAttribute("version",
String.valueOf(point.getVersion()));
            writer.writeEndElement();
            immersion--;
            point = mapa.getAllPointsSequentially();
        }

        object = mapa.getAllObjectsSequentially();
        while (object != null) {
            writer.writeStartElement("way");
            immersion++;
            writer.writeAttribute("id", String.valueOf(object.getId()));
            writer.writeAttribute("user", object.getUser());
            writer.writeAttribute("uid", String.valueOf(object.getUid()));
            writer.writeAttribute("version",
String.valueOf(object.getVersion()));
```

```
            object.resetIterator();
            while (object.hasNext()) {
                writer.writeStartElement("nd");
                immersion++;
                writer.writeAttribute("ref",
String.valueOf(object.getNext().getId()));
                writer.writeEndElement();
                immersion--;
            }

            if (object.getType() != null) {
                writer.writeStartElement("tag");
                immersion++;
                writer.writeAttribute("k", object.getType());
                writer.writeAttribute("v", object.getSubType());
                writer.writeEndElement();
                immersion--;
            }

            writer.writeEndElement();
            immersion--;
            object = mapa.getAllObjectsSequentially();
        }

        writer.writeEndElement();
        immersion--;
        writer.writeEndDocument();
        writer.flush();
        if (immersion != 0) {
            throw new IndexOutOfBoundsException();
        }
    }

    static class Handler extends DefaultHandler {

        MyDataStructure mapa;
        boolean osm = false;
        boolean bounds = false;
        boolean node = false;
        boolean way = false;

        public void setMapa(MyDataStructure mapa) {
            this.mapa = mapa;
        }
```

```
        @Override
        public void startElement(String uri, String localName, String
qName, Attributes attributes) throws SAXException {
            if (qName.equals("osm")) {
                osm = true;
                for (int i = 0; i < attributes.getLength(); i++) {
                    if (attributes.getLocalName(i).equals("version")) {

mapa.setVersion(Float.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("copyright")) {
                        mapa.setCopyright(attributes.getValue(i));
                    }
                    if (attributes.getLocalName(i).equals("attribution")) {
                        mapa.setAttribution(attributes.getValue(i));
                    }
                    if (attributes.getLocalName(i).equals("license")) {
                        mapa.setLicense(attributes.getValue(i));
                    }
                }
            }
            if (qName.equals("bounds") && osm) {
                bounds = true;
                for (int i = 0; i < attributes.getLength(); i++) {
                    if (attributes.getLocalName(i).equals("minlat")) {

mapa.setMinlat(Float.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("minlon")) {

mapa.setMinlon(Float.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("maxlat")) {

mapa.setMaxlat(Float.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("maxlon")) {

mapa.setMaxlon(Float.valueOf(attributes.getValue(i)));
                    }
                }
            }
            if (qName.equals("node") && osm) {
                node = true;
                mapa.CreatePoint();
                for (int i = 0; i < attributes.getLength(); i++) {
                    if (attributes.getLocalName(i).equals("id")) {

mapa.setLastPointId(Long.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("lat")) {

mapa.setLastPointlat(Float.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("lon")) {

mapa.setLastPointlon(Float.valueOf(attributes.getValue(i)));
                    }
```

```
                    if (attributes.getLocalName(i).equals("user")) {
                        mapa.setLastPointUser(attributes.getValue(i));
                    }
                    if (attributes.getLocalName(i).equals("uid")) {
mapa.setLastPointUid(Long.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("version")) {
mapa.setLastPointVersion(Integer.valueOf(attributes.getValue(i)));
                    }
                }
            }
            if (qName.equals("way") && osm) {
                way = true;
                mapa.CreateObject();
                for (int i = 0; i < attributes.getLength(); i++) {
                    if (attributes.getLocalName(i).equals("id")) {
mapa.setLastObjectId(Long.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("uid")) {
mapa.setLastObjectUid(Long.valueOf(attributes.getValue(i)));
                    }
                    if (attributes.getLocalName(i).equals("user")) {
                        mapa.setLastObjectUser(attributes.getValue(i));
                    }
                    if (attributes.getLocalName(i).equals("version")) {
mapa.setLastObjectVersion(Integer.valueOf(attributes.getValue(i)));
                    }
                }
            }
            if (qName.equals("nd") && way && osm) {
                for (int i = 0; i < attributes.getLength(); i++) {
                    if (attributes.getLocalName(i).equals("ref")) {
mapa.setLastObjectPoint(Long.valueOf(attributes.getValue(i)));
                    }
                }
            }
            if (qName.equals("tag") && way && osm) {
                for (int i = 0; i < attributes.getLength(); i++) {
                    if (attributes.getLocalName(i).equals("k")) {
                        if (attributes.getValue(i).equals("railway") ||
attributes.getValue(i).equals("highway")) {
                            mapa.setLastObjectType(attributes.getValue(i));
                            try {
                                if (attributes.getLocalName(i +
1).equals("v")) {
```

```
mapa.setLastObjectSubType(attributes.getValue(i + 1));
                                }
                            } catch (IndexOutOfBoundsException e) {
                                System.err.println("Hodnotu atributu se
nepodarilo ziskat");
                            }
                        }
                    }
                }
            }
        }

        @Override
        public void endElement(String uri, String localName, String qName)
throws SAXException {

            if (qName.equals("osm")) {
                osm = false;
            }
            if (qName.equals("bounds")) {
                bounds = false;
            }
            if (qName.equals("node")) {
                node = false;
            }
            if (qName.equals("way")) {
                way = false;
            }
        }

        @Override
        public void setDocumentLocator(Locator locator) {
        }

        @Override
        public void startDocument() throws SAXException {
            if (osm || bounds || node || way) {
                throw new VerifyError();
            }
        }

        @Override
        public void endDocument() throws SAXException {
            mapa.removeTemporaryPoints();
            if (osm || bounds || node || way) {
                throw new VerifyError();
            }
        }
    }
}
```

Příloha H

```
package CMD;

import DataStructure.ControlClass;
import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.stream.XMLStreamException;
import org.xml.sax.SAXException;

public class CMD {
    public static void main(String[] args) throws
ParserConfigurationException, IOException, SAXException, XMLStreamException
{

        boolean Highway = true;
        boolean Railway = false;
        ControlClass cc = new ControlClass();

        if (args.length == 3) {
            int filtr = Integer.valueOf(args[2]);
            switch (filtr) {
                case (1):
                    Highway = true;
                    Railway = false;
                    break;
                case (2):
                    Highway = false;
                    Railway = true;
                    break;
                case (3):
                    Highway = true;
                    Railway = true;
                    break;
                default:
                    throw new IndexOutOfBoundsException();
            }
            cc.load(args[0]);
            cc.filter(Highway, Railway);
            cc.stow(args[1]);
        } else {
            System.out.println("Nesprávný počet parametrů");
            System.out.println("První parametr představuje vstupní
soubor");
            System.out.println("Druhy parametr představuje výstupní
soubor");
            System.out.println("Třetí parametr upřesňuje filtrační
podmínky");
            System.out.println("pro filtrování silniční sítě zadejte jako
třetí parametr číslo 1");
            System.out.println("pro filtrování železniční sítě zadejte jako
třetí parametr číslo 2");
            System.out.println("pro filtrování obou sítí zadejte jako třetí
parametr číslo 3");
        }
    }
}
```

Příloha I

```
package GUI;

import DataStructure.ControlClass;
import java.awt.Color;
import java.awt.Label;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.stream.XMLStreamException;
import org.xml.sax.SAXException;

public class JFrame extends javax.swing.JFrame {

    boolean Highway = true;
    boolean Railway = false;
    ControlClass cc = new ControlClass();
    String XML_INPUT_FILENAME;
    String XML_OUTPUT_FILENAME = "map_out.xml";

    public JFrame() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jButtonLoad = new javax.swing.JButton();
        jButtonStow = new javax.swing.JButton();
        jButtonFilter = new javax.swing.JButton();
        jCheckBoxHighway = new javax.swing.JCheckBox();
        jButtonExit = new javax.swing.JButton();
        jCheckBoxRailway = new javax.swing.JCheckBox();
        jTextField = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jButtonLoad.setText("Načti");
        jButtonLoad.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonLoadActionPerformed(evt);
            }
        });

        jButtonStow.setText("Ulož");
        jButtonStow.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonStowActionPerformed(evt);
            }
        });

        jButtonFilter.setText("Filtruj");
```

```
        jButtonFilter.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonFilterActionPerformed(evt);
            }
        });

        jCheckBoxHighway.setSelected(true);
        jCheckBoxHighway.setText("Silniční síť");
        jCheckBoxHighway.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jCheckBoxHighwayActionPerformed(evt);
            }
        });

        jButtonExit.setText("Konec");
        jButtonExit.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonExitActionPerformed(evt);
            }
        });

        jCheckBoxRailway.setText("Železniční síť");
        jCheckBoxRailway.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jCheckBoxRailwayActionPerformed(evt);
            }
        });

        jTextField.setText("map_out.xml");

        jLabel1.setText("Výstupní soubor:");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
ING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jButtonLoad)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 86,
Short.MAX_VALUE)
                        .addComponent(jButtonStow))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jCheckBoxHighway)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 66,
Short.MAX_VALUE)
                        .addComponent(jButtonFilter))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jCheckBoxRailway)
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 48,
Short.MAX_VALUE)
                        .addComponent(jButtonExit))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel1)
                        .addGap(10, 10, 10)
                        .addComponent(jTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, 105, Short.MAX_VALUE)))
                .addContainerGap())
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                    .addComponent(jButtonLoad)
                    .addComponent(jButtonStow))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                    .addComponent(jCheckBoxHighway)
                    .addComponent(jButtonFilter))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
ING)
                    .addComponent(jCheckBoxRailway)
                    .addComponent(jButtonExit))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                    .addComponent(jLabel1)
                    .addComponent(jTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

pack();
    }// </editor-fold>

private void jCheckBoxHighwayActionPerformed(java.awt.event.ActionEvent
evt) {
    Highway = jCheckBoxHighway.isSelected();
}
```

```
private void jButtonLoadActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser JFCH = new JFileChooser();
    FileNameExtensionFilter filtr = new FileNameExtensionFilter("Formát
osm, xml", "osm", "xml");
    JFCH.setFileFilter(filtr);
    int vysledek = JFCH.showOpenDialog(this);
    if (vysledek == JFileChooser.APPROVE_OPTION) {
        try {
            XML_INPUT_FILENAME = JFCH.getSelectedFile().getAbsolutePath();
        } catch (Exception e) {
        }
        JDialog dialog = new JDialog();
        dialog.setModal(false);
        dialog.setTitle("pozor");
        dialog.setSize(200, 150);
        dialog.setBackground(Color.white);
        Label lbl = new Label("Prosím čekejte");
        lbl.setForeground(Color.red);
        dialog.add(lbl);
        dialog.setVisible(true);
        try {
            cc.load(XML_INPUT_FILENAME);
        } catch (ParserConfigurationException ex) {
            Logger.getLogger(JFrame.class.getName()).log(Level.SEVERE,
null, ex);
        } catch (SAXException ex) {
            Logger.getLogger(JFrame.class.getName()).log(Level.SEVERE,
null, ex);
        } catch (IOException ex) {
            Logger.getLogger(JFrame.class.getName()).log(Level.SEVERE,
null, ex);
        }
        dialog.setVisible(false);
    }
}

private void jButtonStowActionPerformed(java.awt.event.ActionEvent evt) {
    if (jTextField.getText().equals("")) {
        XML_OUTPUT_FILENAME = "map_out.xml";
    } else {
        XML_OUTPUT_FILENAME = jTextField.getText();
    }
    try {
        cc.stow(XML_OUTPUT_FILENAME);
    } catch (IOException ex) {
        Logger.getLogger(JFrame.class.getName()).log(Level.SEVERE, null,
ex);
    } catch (XMLStreamException ex) {
        Logger.getLogger(JFrame.class.getName()).log(Level.SEVERE, null,
ex);
    }
}

private void jButtonFilterActionPerformed(java.awt.event.ActionEvent evt) {
    cc.filter(Highway, Railway);
}

private void jButtonExitActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}
```

```
private void jCheckBoxRailwayActionPerformed(java.awt.event.ActionEvent
evt) {
    Railway = jCheckBoxRailway.isSelected();
}
    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(JFrame.class.getName()).log(java.util.lo
gging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(JFrame.class.getName()).log(java.util.lo
gging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(JFrame.class.getName()).log(java.util.lo
gging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(JFrame.class.getName()).log(java.util.lo
gging.Level.SEVERE, null, ex);
        }
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new JFrame().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButtonExit;
    private javax.swing.JButton jButtonFilter;
    private javax.swing.JButton jButtonLoad;
    private javax.swing.JButton jButtonStow;
    private javax.swing.JCheckBox jCheckBoxHighway;
    private javax.swing.JCheckBox jCheckBoxRailway;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JTextField jTextField;
    // End of variables declaration
}
```

Příloha J

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="osm">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="bounds">
     <xs:complexType>
      <xs:attribute type="xs:float" name="minlat"/>
      <xs:attribute type="xs:float" name="minlon"/>
      <xs:attribute type="xs:float" name="maxlat"/>
      <xs:attribute type="xs:float" name="maxlon"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="node" maxOccurs="unbounded">
     <xs:complexType>
      <xs:attribute type="xs:long" name="id"/>
      <xs:attribute type="xs:float" name="lat"/>
      <xs:attribute type="xs:float" name="lon"/>
      <xs:attribute type="xs:string" name="user"/>
      <xs:attribute type="xs:long" name="uid"/>
      <xs:attribute type="xs:int" name="version"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="way" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="nd" minOccurs="2" maxOccurs="unbounded">
        <xs:complexType>
         <xs:attribute type="xs:long" name="ref"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="tag">
        <xs:complexType>
         <xs:attribute type="xs:string" name="k"/>
         <xs:attribute type="xs:string" name="v"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:long" name="id"/>
      <xs:attribute type="xs:string" name="user"/>
      <xs:attribute type="xs:long" name="uid"/>
      <xs:attribute type="xs:int" name="version"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute type="xs:float" name="version"/>
   <xs:attribute type="xs:string" name="copyright"/>
   <xs:attribute type="xs:string" name="attribution"/>
   <xs:attribute type="xs:string" name="license"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```