Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Automatické generování základní struktury dopravní sítě pro simulaci na základě veřejně dostupných zdrojů

Plzeň 2013 Martin Štulc

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 8. května 2013

Martin Štulc

Abstract

A distributed simulator of road traffic has been developed at Department of Computer Science of University of West Bohemia. In order to enable the simulation of real areas (e.g. Plzen city), the simulator requires the model of the road traffic network. The aim of this thesis is to find a publicly available source of road traffic network descriptions and to create a tool, which will transforms the obtained descriptions to a XML file, which can be used for creation of the model of the road traffic network for the simulator.

Poděkování

Rád bych poděkoval Ing. Tomáši Potužákovi, Ph.D. za trpělivost, rady a vedení v celém průběhu této práce.

Obsah

1 Úvod

Na katedře informatiky a vypočetní techniky Západočeské univerzity byl vyvinut distribuovaný simulátor městské dopravy. Aby však mohla fungovat simulace realných oblastí (např. města Plzeň), je kromě simulačního programu nutné získat rovněž podklady pro tvorbu modelu existujících silničních sítí.

Cílem této práce je najít vhodný, veřejně dostupný zdroj dat se strukturou dopravní sítě a vytvořit aplikaci, která tento zdroj zpracuje. Výstupem aplikace bude soubor XML, který bude obsahovat informace o dopravní síti a bude moci být použit jako zdrojová data pro simulaci dopravy.

V této práci je poukázáno na výhody a nevýhody různých zdrojů, na jejichž základě je vybrán jeden, který je dále analyzován a použit. Také si ukážeme úskalí tohoto zdroje, způsob jak je vyřešit a navrhneme kostru aplikace. Dále je uvedeno jakým způsobem byla aplikace implementována a na závěr si projdeme vzorový příklad, který ukáže správnou funkci celé aplikace.

2 Možnosti získání struktury dopravní sítě

Tato kapitola se zabývá možnostmi zdrojů dat pro dopravní síť a různými způsoby, jak tato data získat.

2.1 Možné zdroje dat

Následují různé zdroje dat s popisem toho, co poskytují, a v jakém formátu jdou data získat.

2.1.1 Mapy.cz

Jednou z možností získání dat je ze serveru *mapy.cz*. Získávání dat z tohoto serveru by mělo výhodu v pokrytí celé Evropy, s nejlepším pokrytím právě České republiky.

Nalezneme zde mimo jiné tyto informace:

- názvy ulic;
- popisná čísla domů;
- informace o současných i plánovaných uzavírkách silnic;
- rozlišení tříd silnic;
- označení silnic;
- struktura sítě silnic, pěších zón i železnice;
- informace o jednosměrných ulicích;
- kruhových objezdech;
- semaforech.

Informace potřebné pro generování struktury dopravní sítě jsou dostačující, první problém nastává při snaze získat data v nějakém rozumném formátu. Tento server totiž umožňuje pouze uložení obrázku mapy ve formátu *JPEG* a to pouze pro osobní potřebu, nebo zobrazení mapy na webu pomocí *Mapy API* [7], což nám ale užitek nepřinese.

Dostáváme se tedy do fáze, kdy jsou dvě možnosti získání dat z tohoto serveru. První je analýza obrazu (viz kapitola 2.2.1) a druhá je pokusit se odchytit data, která posílá server *mapy.cz* internetovému prohlížeči (viz kapitola 2.2.2).

Výsledkem odchytu je, že server posílá prohlížeči čtverce map ve formátu *GIF*. Tím jsme sice získali jiný formát, ale nezbyde nám stejně jiná možnost než analýza obrazu.

Jak už ale bylo uvedeno, i tato možnost porušuje licenční podmínky serveru *mapy.cz*. Ty dovolují použití pouze pro osobní potřebu vyjma použití *Mapy API* [7], ze kterého ovšem nezískáme potřebná zdrojová data, ale pouze grafický výstup. Ten ovšem máme právo pouze zobrazit [6].

2.1.2 Google maps

Další možností je získání dat z adresy *maps.google.cz*. Tyto mapy jsou sice na území ČR méně přesné než *mapy.cz*, ale v celosvětovém měřítku je předčí.

Nalezneme zde mimo jiné tyto informace:

- názvy ulic;
- popisná čísla domů (nejsou na mapách zobrazena, nejsou tedy dostupná při analýze obrazu);
- rozlišení tříd silnic;
- označení silnic;
- struktura sítě silnic, pěších zón i železnice;
- informace o jednosměrných ulicích;
- kruhových objezdech.

Tento zdroj tedy také poskytuje dostatek informací pro generování základní struktury dopravní sítě. Ale podobně jako v předchozím případě je možnost získat mapy pouze jako obrazový materiál, a to jak klasickým exportem, tak odchytáváním dat (kapitola 2.2.2) – google předává prohlížeči čtverce map ve formátu *PNG*.

Jsou zde sice možnosti jako export dat do formátu *KML* či *GPX*, to je ovšem možné jen pro uživatelsky definované trasy a body. Není možné exportovat celou mapu, která by obsahovala požadovaná data. Jediná možnost získání dat je tedy opět analýza obrazu (viz kapitola 2.2.1).

I zde ale narazíme na licenční podmínky, které zakazují kopírování, překládání, úpravy nebo vytváření odvozeného díla obsahu nebo jeho části [4].

Stejně jako u *mapy.cz*, i zde máme API pro zobrazení mapy na svých stránkách, ale k datům se opět nedostaneme.

2.1.3 Geografický informační systém (GIS)

GIS je organizovaný soubor softwaru a geografických údajů navržený pro efektivní získávání, ukládání, upravování, obhospodařování, analyzování a zobrazování všech forem geografických informací. Více informací lze nalézt v [3].

Pro tuto práci je důležité, že právě *GIS* by mohl být potenciálním zdrojem dat. *GIS* nepředepisuje jaká data by měl obsahovat, předepisuje pouze různé typy datových modelů uložení dat (viz [3]). Pro tuto práci by byl *GIS* vhodný. Je ale potřeba získat zdrojová data, která tento model implementují. To je ovšem poměrně problematické, jelikož tato data nejsou poskytována zdarma.

Hlavní nevýhodou je, že pro jednotlivé lokality je nutné hledat vždy nové zdroje, jelikož každý poskytovatel se specializuje pouze na určitou geografickou oblast. Dále se budeme zabývat tím, že chceme získat pouze dopravní síť města Plzeň.

Nejvhodnějším kandidátem je pak *gis.plzen.eu/uzemnisprava*. Nalezneme zde mimo jiné tyto informace:

- názvy ulic;
- popisná a orientační čísla domů;

- ostrůvky uvnitř silnic;
- struktura sítě silnic a železnice;
- informace o kruhových objezdech.

Jedná se ovšem pouze o zobrazení těchto dat, získání samotných dat je zpoplatněno. Možnost, která zde přichází v úvahu je opět analýza obrazu (viz kapitola 2.2.1). Předchozí dva zdroje jsou ovšem pro tuto metodu vhodnější, zejména z důvodu jednoho zdroje dat pro velkou oblast.

Pokusíme-li se odchytávat data (kapitola 2.2.2) zjistíme, že mapa je posílána jako jeden obrázek formátu *PNG*, což nám opět nepomůže. Chceme-li se tedy vyhnout analýze obrazu, nezbývá než zdrojová data GISu koupit.

V případě koupě dat by dalším postupem byla jejich analýza. Možnosti následného využití nemohl autor prozkoumat, jelikož nákup zdroje je mimo rozsah této práce.

2.1.4 Open Street Map

Další možnou volbou je server *OpenStreetMap.org*. Mapy z tohoto serveru pokrývají celý svět a jsou vytvářeny uživateli pod licencí *Open Data Commons Open Database License* [9]. To zaručuje, že data zde získaná můžeme pro nekomerční účely použít v případě, že uvedeme jako zdroj *OpenStreetMap.org*.

Data na tomto serveru upravují uživatelé, není tedy zaručena jejich přesnost a správnost (i když změna dat prochází schválením administrátorů), ale na druhou stranu díky tomu jsou tyto mapy schopné reagovat na jakoukoli změnu velice rychle.

Je zde možnost zadat poměrně velké množství dat a je pouze na uživatelích, kteří mapu vytvářejí či editují, jaká data vyplní. Uživatelé mohou vyplnit mimo jiné tyto informace:

- názvy ulic;
- popisná čísla a jména budov;
- struktura sítě silnic, pěších cest, tramvají a železnice;

- informace o kruhových objezdech;
- rozlišení tříd silnic;
- informace o jednosměrných ulicích;
- maximální povolená rychlost;
- označení silnic lokálním i mezinárodním značením;
- relativní vertikální umístění silnice;
- počet pruhů;
- materiál povrchu silnic;
- informace, zda se jedná o tunel či most;
- informace zákazu vjezdu motorovým vozidlům, koním a podobně.

Informací je tedy více než dost. Důležité ale je, že všechny tyto informace můžeme získat ve formátu *XML*, které je pro zpracování dat ideální. Dále je možný export do formátů *PNG*, *JPEG*, *PDF*, *SVG* a vkladatelného *HTML*.

V případě *PNG* by se jednalo o analýzu obrazu (viz kapitola 2.2.1), která není nejvhodnější a ani by nešly získat veškeré možné informace. Analyzovat *SVG*, by bylo o mnoho jednodušší, jelikož se jedná o vektorový formát, ale opět nelze získat všechny informace.

Nejvhodnější variantou je tedy export do *XML* a jeho následná analýza (viz kapilola 2.2.3).

2.1.5 Shrnutí

Porovnáme-li zde uvedené zdroje, můžeme dojít k následujícím závěrům:

- Ze serveru *mapy.cz* můžeme získat mapy pouze ve formátu *JPEG* a jako metodu použít analýzu obrazu. Navíc použití tako získaných dat odporuje licenčním podmínkám.
- S *google maps* je situace podobná, liší se pouze obrázkový formát, který je zde *PNG*.

- *GIS* by byl dobrý kandidát, ale získání zdrojových dat je zpoplatněno.
- *OpenStreetMap.org* poskytuje veškeré potřebné informace pod licencí, která umožňuje použití získaných dat, ale nemůžeme se spolehnout na korektnost informací. Data z tohoto serveru je možné získat ve formátu *XML*.

Po zvážení všech možností byl jako nejlepší varianta zvolen export dat do formátu *XML* ze serveru *OpenStreetMap.org* (podrobně v kapitole 3).

2.2 Metody získávání dat

Máme-li vybraný zdroj dat, musíme zvolit metodu, jak tato data získat. Tím se zabývá tato kapitola.

2.2.1 Analýza obrazu

Pro získání základní struktury dopravní sítě metodou analýzy obrazu je nejprve nutné získat mapový podklad, který bude obrazově znázorňovat veškeré informace, které chceme získat. To znamená, že třídy silnic budou označeny různou barvou či tloušťkou, budou zde viditelné popisky jmen ulic atd.

Dále na základě barvy jednotlivých pixelů a jejich shluků nalezneme nejpravděpodobnější místa, kde se nalézají jednotlivé silnice daných tříd.

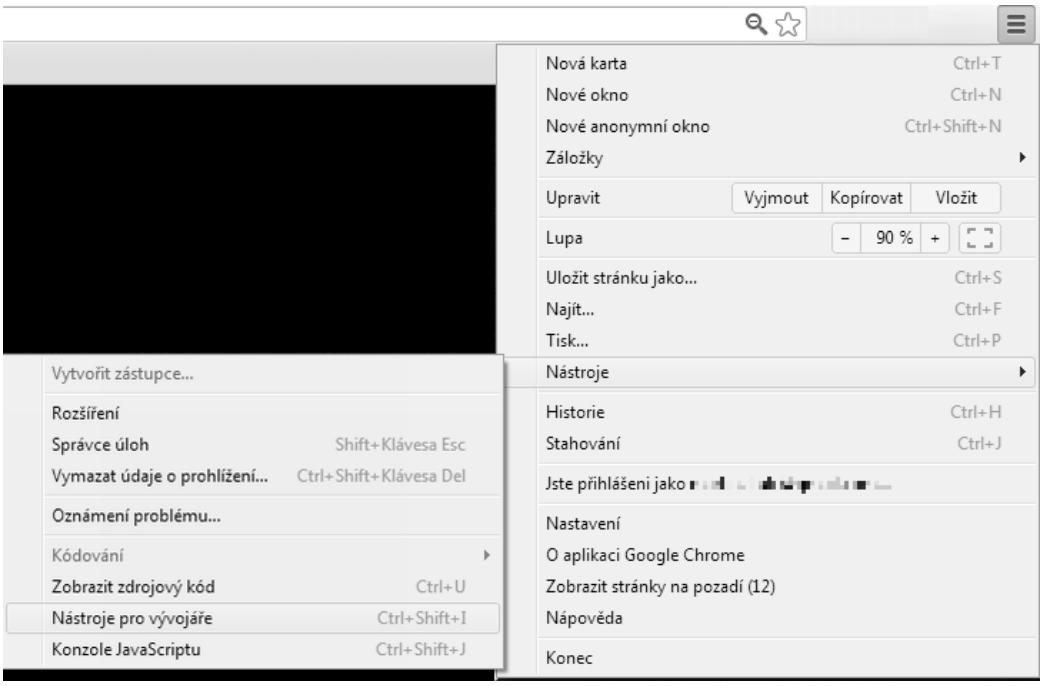
S názvy ulic by byla situace horší. Bylo by potřeba nejprve zjistit, kde se nachází text, dále ho analyzovat a následně přiřadit ke správné ulici.

Podrobnější informace o analýze obrazu můžeme nalézt v [5] a [8].

2.2.2 Odchytávání dat mezi serverem a prohlížečem

Touto metodou se rozumí odchytávání dat, které posílá server prohlížeči jako odpověď na požadavek zobrazení stránky.

Možností je samozřejmě více. Zde použitou možností je funkce aplikace *Google Chrome Verze 24.0.1312.57 m* zvaná *nástroje pro vývojáře*.



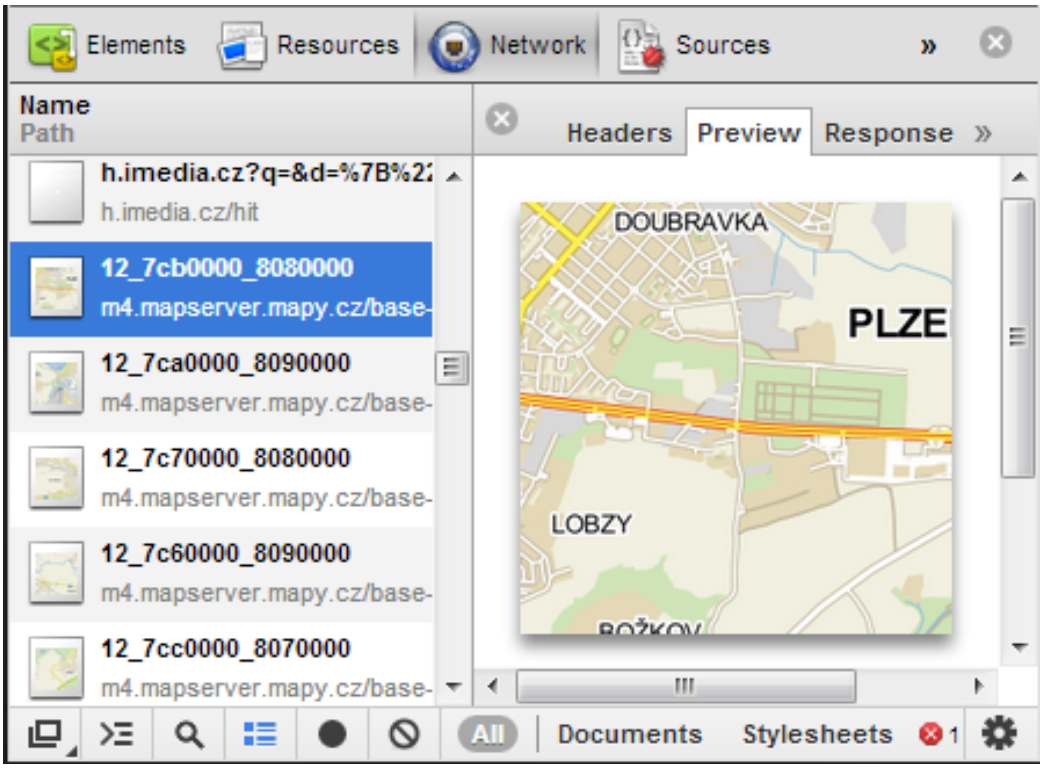
Obrázek 2.1: Zobrazení nástrojů pro vývojáře.

Odchyt dat provedeme následně:

1. Zobrazení nástrojů pro vývojáře – Nastavení – Nástroje – Nástroje pro vývojáře (viz obrázek 2.1).
2. Vymazat zaznamenaná data tlačítkem `Clear` (⊘) vlevo dole.
3. Do adresního řádku vložit adresu serveru. Po potvrzení Chrome zobrazí data přenášená mezi serverem a prohlížečem (viz obrázek 2.2).
4. Z toho lze pak vyčíst jakým způsobem se přenáší celá mapa – v případě obrázku 2.2 rozkouskovaná na jednotlivé čtverce ve formátu *JPEG*.

2.2.3 Analýza XML dokumentu

Analýza *XML* dokumentu se z uvedených metod jeví nejjednodušší a nejpřesnější. Při aplikaci je nutné prozkoumat daný dokument a zjistit, jaké informace jsou v něm uloženy a jak.



Obrázek 2.2: Zobrazení přenesených dat.

Ve chvíli, kdy známe strukturu dat v dokumentu, můžeme je zpracovat například pomocí technologií *SAX* či *DOM* (pro Javu nalezneme informace v [12, 13]).

3 Open Street Map

Open Street Map je otevřená wiki-mapa světa dostupná na adrese `http://www.openstreetmap.org`.

3.1 Export dat

Ještě předtím, než začneme pracovat se samotným programem, je potřeba získat zdrojová data map. Tato data jdou ze serveru `http://www.openstreetmap.org` získat v různých formách, jak již bylo uvedeno dříve. Nás bude zajímat formát *XML*.

Stáhneme tedy požadovaná data jako XML s příponou OSM. Podrobný návod získání dat nalezneme v příloze B.1.

3.2 Popis exportovaných dat

Tato kapitola popisuje strukturu XML souboru s příponou *.OSM* získaného dle návodu v příloze B.1. Podrobné informace o struktuře těchto dat lze nalézt v [10].

Veškeré použité příklady jsou upravené a zjednodušené pro potřebu vysvětlení struktury.

3.2.1 Element osm

Příklad elementu `osm`:

```
1 <osm version="0.6" generator="CGImap 0.0.2" copyright="
      OpenStreetMap and contributors" attribution="http://www.
      openstreetmap.org/copyright" license="http://opendatacommons.
      org/licenses/odbl/1-0/">
2     ...
3     ...
4 </osm>
```

Jedná se o úvodní element obsahující verzi celého dokumentu, generátor, pomocí kterého byl tento *osm* dokument vygenerován, a samozřejmě licence, pod kterou může být distribuován.

Uvnitř tohoto elementu se nalézají elementy `bounds`, `node`, `way` a `relation`.

3.2.2 Element bounds

Příklad elementu `bounds`:

```
1 <bounds minlat="49.7384900" minlon="13.3612400" maxlat="
    49.7514800" maxlon="13.3769300"/>
```

Tento element vymezuje obdélník mapy, která je obsažena v tomto dokumentu. Kde `minlat` a `maxlat` je minimální a maximální zeměpisná šířka v GPS souřadnicích a `minlon` a `maxlon` je minimální a maximální zeměpisná délka.

3.2.3 Element node

Příklad elementu `node`:

```
1 <node id="296805180" lat="49.7462014" lon="13.3739563" user="
    Parkis" uid="95101" visible="true" version="3" changeset="
    3223939" timestamp="2009-11-26T21:30:26Z">
2     <tag k="addr:conscriptionnumber" v="316"/>
3     <tag k="addr:housenumber" v="316/6"/>
4     <tag k="addr:postcode" v="30100"/>
5     <tag k="addr:street" v="Sady Pětatřicátníků"/>
6     <tag k="addr:streetnumber" v="6"/>
7     <tag k="source:addr" v="uir_adr"/>
8     <tag k="uir_adr:ADRESA_KOD" v="24573698"/>
9 </node>
```

Element `node` je jedním bodem na mapě určeným GPS souřadnicemi. Tento bod může být součástí cesty, budovy, parku a dalších. V podstatě každý objekt na mapě je reprezentován jedním nebo více body.

Každý bod má:

- své unikátní `id`, pomocí kterého je odkazován (integer >= 1);
- zeměpisnou šířku (`lat`) (float >= -90.0 and <= 90.0 zaokrouhleno na 7 desetinných míst);
- zeměpisnou délku (`lon`) (float >= -90.0 and <= 90.0 zaokrouhleno na 7 desetinných míst);
- uživatele (`user`), který ho vytvořil;
- verzi (`version`) – při každé změně bodu musí být verze zvýšena (integer >= 1);
- viditelnost v mapě (`visible`) (true/false);
- datum a čas vytvoření/úpravy (`timestamp`);
- atribut `changeset` (integer >= 1), který udává id změny, ve které byl bod uložen. V každé změně může být vytvořeno nebo upraveno více elementů.

Node může mít také speciální vlastnosti, které jsou definované elementem `tag`. V tomto elementu pak atribut `k` je klíčem a atribut `v` hodnotou této vlastnosti.

V uvedeném příkladu se jedná o dům s adresou *Sady Pětatřicátníků 316/6*, PSČ *30100*.

3.2.4 Element way

Příklad elementu `way`:

```
1  <way id="4750996" user="Tomas Pajonk" uid="305367" visible="true
       " version="10" changeset="11568302" timestamp="2012-05-11
       T14:03:10Z">
2      <nd ref="30053414"/>
3      <nd ref="1583016919"/>
4      <tag k="hgv_avg_speed" v="30"/>
5      <tag k="highway" v="primary"/>
6      <tag k="int_ref" v="E 53"/>
7      <tag k="maxspeed" v="50"/>
8      <tag k="name" v="Klatovská"/>
9          <tag k="lanes" v="2"/>
10     <tag k="oneway" v="yes"/>
```

```
11 |     <tag k="ref" v="27"/>
12 |</way>
```

Element `way` je křivka definovaná několika body na mapě (element `node`). Křivka může být buď otevřená nebo uzavřená, kde se poslední bod propojí s prvním.

Každá křivka má:

- své unikátní `id` pomocí kterého je odkazována (integer >= 1);
- uživatele (`user`), který ji vytvořil;
- verzi (`version`) – při každé změně křivky musí být verze zvýšena (integer >= 1);
- viditelnost v mapě (`visible`) (true/false);
- datum a čas vytvoření/úpravy (`timestamp`);
- atribut `changeset` (integer >= 1), který udává id změny, ve které byl bod uložen. V každé změně může být vytvořeno nebo upraveno více elementů;
- reference na body, přes které tato křivka vede (element `nd`). Tyto body jsou odkazovány svým id v atributu `ref`. Záleží na pořadí těchto bodů.

Steně jako `node`, tak i `way` má speciální vlastnosti definované elementy `tag`, kde `k` je klíč vlastnosti a `v` je hodnota vlastnosti. Klíčů těchto vlastností je celá řada, uvedeme zde jen některé:

- hgv_avg_speed – průměrná rychlost (integer >= 1; jednotka záleží na regionu);
- maxspeed – maximální rychlost (integer >= 1; jednotka záleží na regionu);
- highway – třída cesty (primary, secondary, footway, residential, ...);
- int_ref – mezinárodní značení silnic (např. E 53);
- name – jméno ulice (např. Klatovská);

- oneway – jedná se o jednosměrnou ulici (v=”yes”pro jednosměrku ve směru pořadí bodů, v=”reverse”pro jednosměrku proti směru pořadí bodů);
- ref – české značení silnic (např. 27);
- area – jedná se o uzavřenou křivku (v=”yes”);
- railway – jedná se o koleje (v=”tram”pro tramvaj, v=”rail”pro vlak);
- building – jedná se o budovu (v=”yes”);
- lanes – počet pruhů (integer >= 1).

Silnice jsou vždy vedeny jen přes ty body, na kterých platí všechny tagy. Změní-li se tedy například maximální povolená rychlost, již se nejedná o stejnou silnici, přestože může mít stejné jméno ulice.

Také pokud je silnice víceproudá, ale mezi jednotlivými jízdními pruhy je ostrůvek, již se nejedná o jednu víceproudou silnici, ale o více jednosměrných silnic.

V příkladu se pak jedná o jeden směr silnice v ulici Klatovská, který má dva jízdní pruhy. Průměrná rychlost je 30 km/h, maximální povolená rychlost je 50 km/h, označení v českém značení má 27 a v mezinárodním E 53. Tato silnice prochází přes dva body, je jednosměrná, jelikož se jedná pouze o jednu polovinu silnice, protože mezi protisměrnými pruhy je ostrůvek.

3.2.5 Element relation

Příklad elementu `relation`:

```
1  <relation id="1956768" user="Pavel Cvrček" uid="571360" visible=
       "true" version="7" changeset="14636931" timestamp="2013-01-13
       T16:40:10Z">
2      <member type="node" ref="299756679" role="stop"/>
3      <member type="node" ref="1581227574" role="platform"/>
4      <member type="node" ref="1587430359" role="stop"/>
5      <member type="node" ref="1587430361" role="platform"/>
6      ...
7      <member type="way" ref="144582001" role=""/>
8      <member type="way" ref="27494738" role=""/>
9      <member type="way" ref="23011096" role=""/>
10     <member type="way" ref="23173180" role=""/>
11     ...
```

```
12      <tag k="complete" v="yes"/>
13      <tag k="from" v="Lobzy, Pod Vrchem"/>
14      <tag k="name" v="Trolejbus 15: Lobzy, Pod vrchem =&gt;
            Borská pole, Teslova"/>
15      <tag k="ref" v="15"/>
16      <tag k="route" v="trolleybus"/>
17      <tag k="source" v="survey"/>
18      <tag k="to" v="Borská pole, Teslova"/>
19      <tag k="type" v="route"/>
20  </relation>
```

Relace používáme, chceme-li definovat jeden objekt, který se týká více elementů `node`, `way` či samotné `relation`. Poslednímu případu, kdy je relace prováděna nad jinou relací se pak říká super-relace [10].

V příkladu uvedeném výše se jedná o relaci týkající se hromadné dopravy, konkrétně linky číslo 15 ve směru *Lobzy, Pod vrchem – Borská pole, Teslova*.

První elementy `member` typu *node* určují nody označující zastávky této linky. Konkrétně *role="stop"* určují místo na silnici, kde autobus zastavuje (tento bod je součástí silnice) a *role="platform"*pak umístění zastávky (mimo silnici). Následují elementy typu *way*, určující přes které silnice trolejbus této linky jezdí.

Nyní vysvětlíme jednolivé tagy:

- *complete* – celá trasa trolejbusu je zde uvedena;
- *from* – počáteční stanice;
- *to* – cílová stanice;
- *name* – jméno linky;
- *ref* – číslo linky;
- *route* – typ linky;
- *type* – typ relace.

3.2.6 Komplexní příklad

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <osm version="0.6" generator="CGImap 0.1.0" copyright="
       OpenStreetMap and contributors" attribution="http://www.
       openstreetmap.org/copyright" license="http://opendatacommons.
       org/licenses/odbl/1-0/">
3      <bounds minlat="49.7441760" minlon="13.3722500" maxlat="
           49.7443670" maxlon="13.3725610"/>
4      <node id="31662298" lat="49.7409716" lon="13.3715439" user="
           javlada" uid="699970" visible="true" version="4"
           changeset="15374375" timestamp="2013-03-15T14:58:16Z" />
5      <node id="31662299" lat="49.7403513" lon="13.3713636" user="
           Parkis" uid="95101" visible="true" version="3" changeset=
           "3944872" timestamp="2010-02-22T16:30:10Z"/>
6      <node id="611664637" lat="49.7439309" lon="13.3723386" user=
           "Pavel Cvrček" uid="571360" visible="true" version="3"
           changeset="10532048" timestamp="2012-01-29T16:44:46Z">
7          <tag k="name" v="U Práce, Klatovská"/>
8          <tag k="public_transport" v="stop_position"/>
9          <tag k="wheelchair" v="yes"/>
10     </node>
11     <node id="644881043" lat="49.7413675" lon="13.3716641" user=
           "Parkis" uid="95101" visible="true" version="1" changeset
           ="3909423" timestamp="2010-02-18T14:43:55Z"/>
12     <way id="5719251" user="pschonmann" uid="5748" visible="true
           " version="16" changeset="13767204" timestamp="2012-11-05
           T22:34:18Z">
13         <nd ref="31662299"/>
14         <nd ref="31662298"/>
15         <nd ref="644881043"/>
16         <tag k="hgv_avg_speed" v="30"/>
17         <tag k="highway" v="primary"/>
18         <tag k="int_ref" v="E 53"/>
19         <tag k="maxspeed" v="50"/>
20         <tag k="name" v="Klatovská"/>
21         <tag k="ref" v="27"/>
22     </way>
23 </osm>
```

V tomto příkladu jsou čtyři *nody*. První, druhý a čtvrtý nejsou ničím zvláštní. Jsou to obyčejné body na mapě, na které může být odkazováno.

Třetí bod již definuje více. Jedná se označení místa, kde staví veřejná doprava (nikoli zastávka, ta je mimo komunikaci). Přilehlá zastávka má označení *U Práce, Klatovská* a je bezbariérově upravena.

Dále je v příkladu jeden element *way*. Jelikož chybí atribut *area*, jedná se o neuzavřenou křivku. Dle atributu *highway* vidíme, že se jedná o silnici, dokonce o silnici první třídy. Název ulice, ve které se silnice nalézá, je Klatovská. Dále zde vidíme české označení silnice, které je 27, a mezinárodní označení E 53. Maximální rychlost je zde 50 km/h, ale průměrná rychlost je zde jen 30 km/h. Všechny tyto vlastnosti platí na spojnici prvního, druhého a čtvrtého bodu.

Třetí bod by mohl být součástí relace, která by například sdružovala všechny zastávky daného spoje jako v příkladu v kapitole 3.2.5.

4 Analýza programu

V tomto okamžiku již máme vybraný zdroj dat a zbývá vytvořit program, který bude tato data zpracovávat. Jeho analýza je popsána v této kapitole.

4.1 Specifikace požadavků

Zde je popsána specifikace požadavků na výsledný program.

4.1.1 Účel a rozsah práce

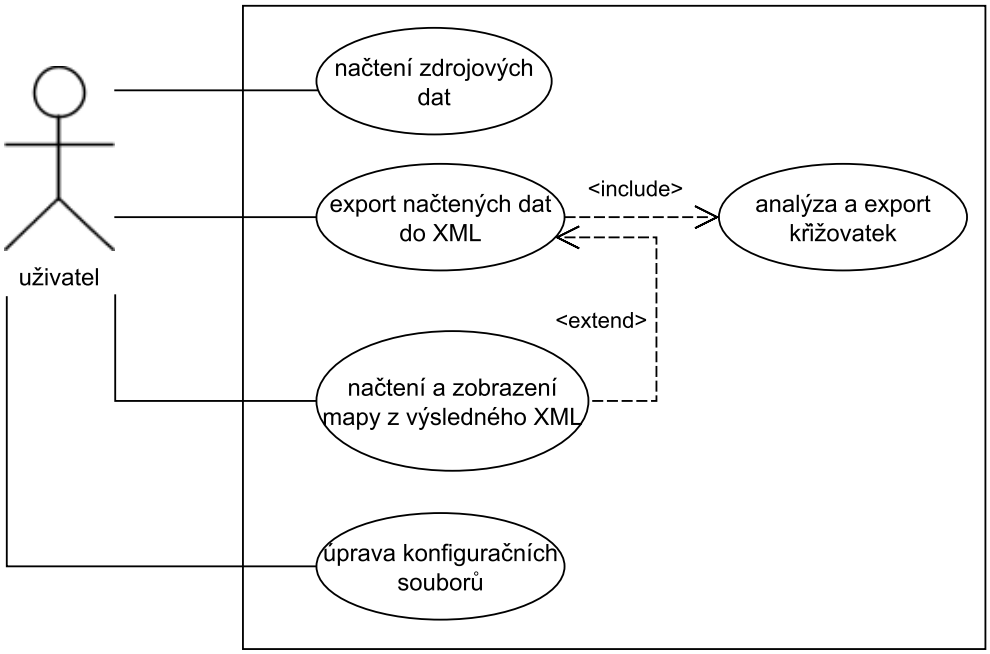
Účelem práce je vytvořit aplikaci, která zpracuje zdrojová data z vybraného zdroje, vybere z nich data týkající se dopravní sítě a exportuje je do XML dokumentu. Do tohoto dokumentu také vypíše seznam křižovatek s vypočítaným středem. Tento XML dokument je možné následně načíst za účelem prohlížení dopravní sítě.

Výsledný XML dokument bude sloužit jako zdojová data pro již existující systém sloužící pro simulaci silniční dopravy.

4.1.2 Základní přehled funkcí

Základní operace, které budou uživateli umožněny jsou následující:

- načtení zdrojových dat – umožní uživateli načíst data z jednoho nebo více zdrojových souborů;
- export načtených dat do XML – data, která uživatel načetl projdou restrukturalizací, složením do jednoho celku, vypočítáním křižovatek a uložením výsledku do XML (křižovatky nejsou součástí zdrojových souborů a musejí být získány jinou cestou);
- načtení a zobrazení mapy z výsledného XML – uživatel bude moci načíst výsledné XML a zobrazí se mu mapa uložené dopravní sítě;



Obrázek 4.1: Diagram případů užití.

- úprava konfiguračních souborů – uživateli bude umožněno měnit konfigurační soubory přímo z aplikace pomocí GUI.

Popsané funkce ukazuje diagram případů užití na obrázku 4.1.

4.1.3 Požadavky na rychlost

Na rychlost běhu aplikace nebyly vzneseny žádné požadavky. Předpokladem ale je, že nad zdrojovými daty obsahujícími jedno průměrné české město získáme výsledek maximálně v řádu minut.

4.1.4 Předpoklady a závislosti

Předpokladem úspěšného běhu aplikace je možnost stažení vstupních dat ze serveru `openstreetmap.org` nebo vlastnění zdrojových dat offline.

4.1.5 Požadavky na uživatelské rozhraní

Program bude mít grafické uživatelské rozhraní umožňující generování a zobrazení mapy. Dále bude možné spustit generování výsledného souboru z konzole (pro případ dávkového zpracování).

4.2 Analýza struktur

Datová struktura by měla umožňovat získávání dat z různých zdrojů a export dat do různých formátů výstupu.

Bude tedy vhodné rozdělit program na dvě části: první bude provádět import dat do vlastních struktur a druhá bude provádět export dat z vlastních struktur do výsledného formátu. Mezi tyto části by bylo vhodné vložit další, která bude vytvářet křižovatky na základě dostupných dat. Dále je vhodné vytvořit aplikaci, kterou je možno spouštět jak z konzole (pro případ dávkového zpracování), tak z GUI pro pohodlnost uživatele.

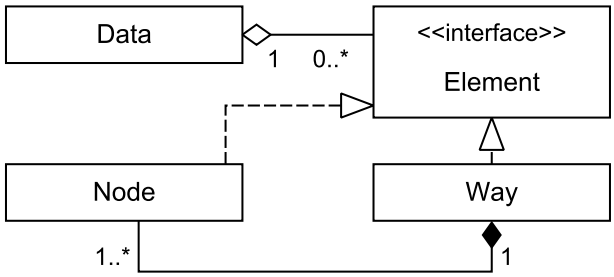
Vezmeme-li v úvahu tyto věci a budeme-li chtít dodržet třívrstvou architekturu, která je zde na místě, můžeme udělat návrh struktury balíků:

- data;
- import;
- export;
- console;
- gui;

kde balík `data` zastupuje datovou vrstvu, `import` a `export` aplikační vrstvu, `gui` a `console` zobrazovací vrstvu.

4.2.1 Návrh datové vrstvy

Datová vrstva bude poměrně jednoduchá a ukazuje ji obrázek 4.2.



Obrázek 4.2: Návrh datové struktury.

4.2.2 Návrh importu

Zdrojový formát z `openstreetmap.org` s příponou *osm* je ve skutečnosti XML. Tento soubor může být různě velký. `Openstreetmap.org` omezují velikost souboru počtem *nodů* (bodů), které tento soubor obsahuje a to na 50000. Přiblížíme-li se této hranici, je velikost souboru přibližně 10 MB. Těchto souborů si navíc můžeme stáhnout více a program je musí umět načíst všechny a sloučit je do jedné datové struktury. Musíme tedy zvolit správnou metodu parsování xml dokumentu. V úvahu přicházejí tyto možnosti:

- DOM;
- JAXB;
- SAX;
- vlastní parsování.

Co se týče DOMu, ten je na takto velký dokument nevhodný – načítání trvá příliš dlouho a zabírá mnoho paměti. JAXB je podobný případ, navíc již máme navrženou vlastní datovou strukturu. Jelikož zdrojová data potřebujeme pouze číst a to jen jednou, je nejvhodnějším řešením parsování *osm dokumentu* metodou SAX [12].

4.2.3 Návrh exportu

Při exportu dat se vyberou data o dopravní síti. Tato data pak zapíšeme do XML dokumentu v takové struktuře, jakou potřebujeme (viz kapitola

4.2.4). Musíme také vybrat, jakou metodou budeme data do XML zapisovat. Opět můžeme zvolit *DOM* a vytvářet strom z elementů nebo budeme zapisovat XML přímo do textového streamu jako text. Potom ovšem nemáme jistotu validního XML. To lze napravit tím, že po jeho vytvoření tento dokument validujeme a formátujeme.

Předchozím postupem jsme získali informace o dopravní síti, ale nikoli o křižovatkách. Ty musíme získat sami na základě známých poloh silnic. Více v kapitole 4.3.

4.2.4 Návrh struktury výsledného XML

Do XML budeme chtít ukládat mapu dopravní sítě. Tato mapa bude mít nějaký popis, okraje, dále pak silnice a křižovatky. Nazvěme tedy kořenový element mapy `map` a popis mapy `description`. Okraje mapy budeme definovat dvěma body, které vytvoří pomyslný obdélník. První bude levý horní bod mapy a druhý bude pravý dolní roh mapy. Okraje tedy nazvěme `bounds` a jednotlivé body `node`. Každý bod musí být nějak definován. V našem případě to budou GPS souřadnice. Proto každému bodu přidáme atributy pro zeměpisnou šířku (`latitude`) a pro zeměpisnou délku (`longitude`). Pro případ, že budeme chtít okraje rozšířit o další vlastnosti než body, uzavřeme oba body do elementu `nodes`.

Nyní máme definovány obecné vlastnosti mapy a přichází na řadu silnice. Každá silnice bude definována elementem `road` a bude mít tyto vlastnosti – id, maximální rychlost, jméno ulice, informace zda se jedná o jednosměrnou ulici, počet pruhů, třídu silnice, české a mezinárodní značení, relativní vertikální polohu a body, přes které tato silnice prochází. Tyto vlatnosti namapujeme na následující atributy s tím, že pokud některá vlastnost nebude známá, atribut nebude přítomen:

- id na atribut `id` (Integer > 0),
- maximální rychlost na atribut `maxspeed` (Integer > 0),
- jméno ulice na atribut `name` (String),
- informace o jednosměrné ulici na atribut `oneway` (Boolean),
- počet pruhů na atribut `lanes` (Integer > 0),

- třídu ulice na atribut `roadClass` (String),
- české značení silnice na atribut `roadNumber` (String),
- mezinárodní značení silnice na atribut `roadInternationalNumber` (String),
- relativní vertikání polohu na atribut `layer` (Integer),
- body, přes které silnice prochází, zadáme v již definovaných `node`, které budou uzavřeny v elementu `nodes`.

Následující XML ukazuje návrh výsledku.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <map source="OpenStreetMap and contributors" attribution="http:
       //www.openstreetmap.org/copyright">
3    <description>
4      ???
5    </descrtiption>
6    <bounds>
7      <nodes>
8        <node latitude="???" longitude="???"/>
9        <node latitude="???" longitude="???"/>
10     </nodes>
11   </bounds>
12   <road id="???" maxspeed="???" name="???" oneway="???" lanes="
         ???"
13       roadClass="???" roadInternationalNumber="???" roadNumber="
             ???" layer="???">
14     <nodes>
15       <node latitude="???" longitude="???"/>
16       <node latitude="???" longitude="???"/>
17       <node latitude="???" longitude="???"/>
18       <node latitude="???" longitude="???"/>
19     </nodes>
20   </road>
21   ...
22   <crossroad id="???">
23     <position>
24       <node latitude="???" longitude="???"/>
25     </position>
26     <roads>
27       <ref id="???" name="???"/>
28       <ref id="???"/>
29       <ref id="???" name="???"/>
30       <ref id="???"/>
31     </roads>
```

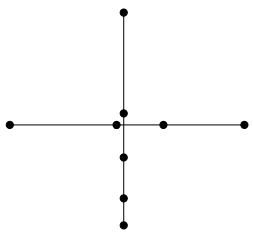
```
32 |   </crossroad>
33 |   ...
34 |</map>
```

4.3 Analýza algoritmu pro vytváření křižovatek

Jedním z požadavků bylo, že ve výstupním XML budou křižovatky. Ty ovšem nejsou uvedeny ve zdrojových datech a musíme je tedy získat sami z dopravní sítě silnic. Každá křižovatka by měla mít informace o svém středu a o silnicích, které jsou její součástí.

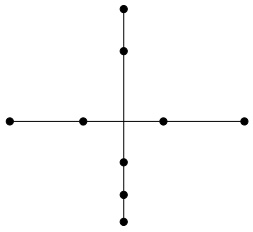
Jednou z možností, jak tyto informace získat, je použít analytickou geometrii – chápat každou silnici jako na sebe navazující úsečky a testovat, zda se některá úsečka neprotíná s jinou. Tato metoda je sice přesná, ale velice zdlouhavá.

Další možností je využít toho, že s vysokou pravděpodobností mají křížící se silnice společný bod a pokud ne přímo společný, mají body, které leží velice blízko sebe. Můžeme tedy určit silnice, které mají body blízko sebe (jak blízko určí konstanta, kterou si uživatel bude moci regulovat přesnost). O takových silnicích lze prohlásit, že přiléhají ke křižovatce, jejíž střed je bod ležící ve středu obrazce, vzniklého spojením blízkých bodů (viz obrázek 4.3). Touto metodou získáme křižovatky poměrně snadno, ale má to háček.

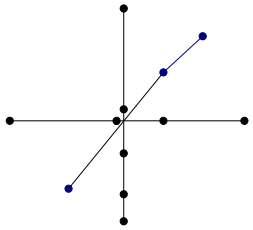


Obrázek 4.3: Ukázka křižovatky, která bude detekována.

Určité procento křižovatek nebude detekováno vůbec – takové, které se sice kříží, ale nemají body blízko sebe (viz obrázek 4.4). Rovněž v případě, kdy některá silnice nebude mít bod blízko středu křižovatky, nebude brána



Obrázek 4.4: Ukázka křižovatky, která nebude detekována.



Obrázek 4.5: Ukázka křižovatky, kde jedna silnice nebude členem křižovatky.

jako člen křižovatky (viz obrázek 4.5). Pravděpodobnost výskytu těchto případů je ale velice malá.

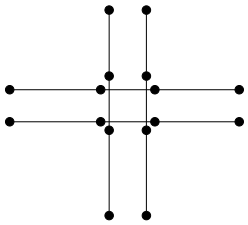
Na druhou stranu je velice pravděpodobné, že bude jako křižovatka označeno něco, co křižovatkou není. Jedním z takových případů jsou dvě na sebe navazující silnice (např. přechází-li silnice v most). O křižovatku se nejedná, ale obě dvě silnice mají společný bod. Tomuto případu se vyhneme následovně:

1. po vygenerování křižovatky zjistíme, zda křižovatka nemá přesně dvě silnice;
2. pokud má, zjistíme, zda mají společný bod, který je na konci jedné a zároveň na začátku druhé silnice;
3. pokud tento společný bod existuje, jedná se o uvedený případ a tedy se nejedná o křižovatku.

Tím jsme tedy vyloučili první problém. Další problém, který může nastat, je označení mimoúrovňové křižovatky, či mostu a podjezdu za křižovatku. Tento problém neřeší ani analytická geometrie. Vyřešíme ho tak, že se podíváme, zda je vyplněný atribut `layer` (relativní vertikální poloha). U všech silnic

v křižovatce musí mít tento atribut stejnou hodnotu. To tedy platí pokud je vyplněný. Pokud není, nemůžeme tento případ detekovat a bude chybně označen za křižovatku.

Také je třeba si uvědomit, že existují složité křižovatky, do kterých vede několik souběžných silnic, které se protínají s jinými souběžnými silnicemi (viz obrázek 4.6). Podle algoritmu by takováto křižovatka byla vyhodnocena



Obrázek 4.6: Ukázka složitější křižovatky se souběžnými silnicemi.

jako čtyři různé křižovatky a ne jako jedna velká. To napravíme tím, že na konci algoritmu, když máme vygenerovány všechny křižovatky, zjistíme, jaká je vzdálenost mezi všemi středy křižovatek a pokud bude menší než hodnota zadaná uživatelem, sloučíme tyto křižovatky do jedné.

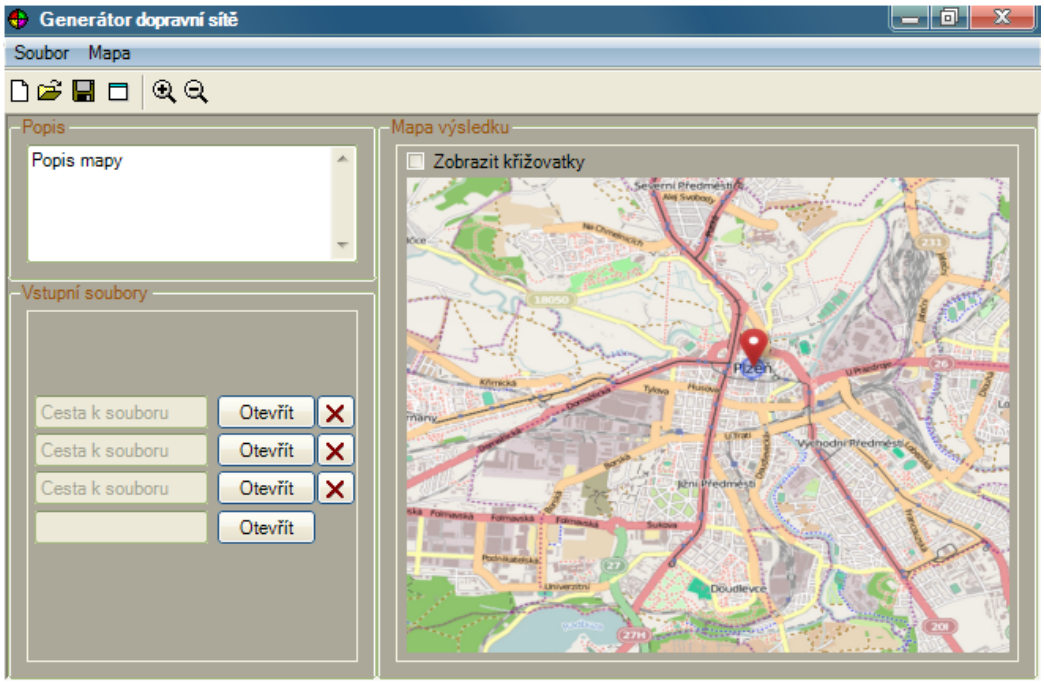
Jak je vidět, celý algoritmus je závislý na dvou konstantách, jejichž nastavení má zásadní vliv na přesnost výsledku. První z nich je maximální vzdálenost bodů různých silnic, aby bylo možné uvažovat, že se kříží. Pokud se tato konstanta zvolí příliš malá, může se stát, že křižovatka nebude detekována. Pokud bude ale příliš velká, mohou být jako křižovatky označovnány všechny body souběžných silnic.

Druhou konstantou je maximální vzdálenost středů křižovatek, které mají být sloučeny do jedné. Pokud bude tato konstanta příliš malá, může se stát, že velké křižovatky nebudou sloučené do jedné a pokud bude příliš veliká, sloučí se do jedné i dvě různé křižovatky. Problémem zůstává, že některé křižovatky jsou větší než soustava malých různých křižovatek. Je na uživateli, jaké menší zlo zvolí nastavením konstanty. Jestli obětuje malé křižovatky a dostane jednu naprosto nesmyslnou, ale bude mít velké křižovatky označené jako jednu. Nebo zachová malé křižovatky, ale velké se mu rozpadnou na více menších křižovatek.

Analýza křižovatek dopravní sítě je jistě nejsložitější částí programu a je tedy vhodným kandidátem pro paralelizování, a tím i urychlení běhu aplikace.

4.4 Návrh grafického prostředí

Grafické prostředí by mělo umožňovat především načíst vstupní data a zobrazit výstupní data. Grafické uživatelské rozhraní by mohlo vypadat jako na obrázku 4.7. V lévé části se zadává popis mapy a načítají se vstupní



Obrázek 4.7: Návrh grafického prostředí.

soubory. Vždy, když se přidá nový vstupní soubor, přibude řádek, pro zadání dalšího. Každý vstupní soubor lze také odstranit tlačítkem s křížkem. V pravé části je prostor pro zobrazení mapy, kterou lze pomocí tlačítek v tool baru zvětšovat a zmenšovat. V mapě bude možné zobrazit křižovatky pomocí zaškrtávacího pole. Výstup se vygeneruje pomocí tlačítka tool baru.

5 Realizace programu

Celý program je rozdělen do několika částí:

- import vstupních dat do vlastních struktur;
- transformace dat a jejich následný export do xml;
- konzolová aplikace;
- grafické rozhraní postavené nad konzolovou aplikací.

5.1 Programovací jazyk a sestavení

Jako programovací jazyk byla zvolena *Java* ve verzi 1.7 s grafickým frameworkem *Swing* [14].

Sestavení programu probíhá pomocí technologie ANT [2] spuštěním defaultního cíle *distjar*. Vstupním bodem aplikace je třída `ApplicationStart` umístěná v balíku `cz.zcu.fav.kiv.stulc.dp.run`.

5.2 Struktura programu

Aplikace je rozdělena do následujících balíků:

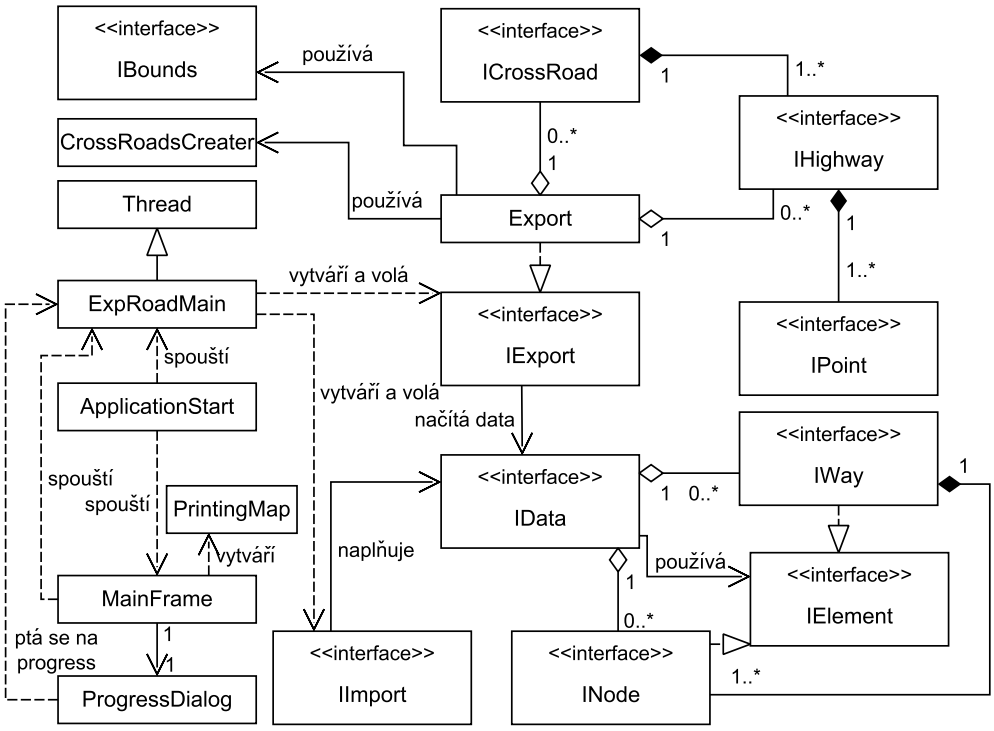
- `cz.zcu.fav.kiv.stulc.dp.console`
- `cz.zcu.fav.kiv.stulc.dp.exported`
  - `cz.zcu.fav.kiv.stulc.dp.exported.data`
    - `cz.zcu.fav.kiv.stulc.dp.exported.data.impl`
  - `cz.zcu.fav.kiv.stulc.dp.exported.app`
    - `cz.zcu.fav.kiv.stulc.dp.imported.app.impl`
- `cz.zcu.fav.kiv.stulc.dp.gui`

  - `cz.zcu.fav.kiv.stulc.dp.gui.config`
  - `cz.zcu.fav.kiv.stulc.dp.gui.filefilters`
  - `cz.zcu.fav.kiv.stulc.dp.gui.map`
- `cz.zcu.fav.kiv.stulc.dp.imported`
  - `cz.zcu.fav.kiv.stulc.dp.imported.data`
    - `cz.zcu.fav.kiv.stulc.dp.imported.data.impl`
  - `cz.zcu.fav.kiv.stulc.dp.imported.app`
    - `cz.zcu.fav.kiv.stulc.dp.imported.app.impl`
- `cz.zcu.fav.kiv.stulc.dp.run`
- `cz.zcu.fav.kiv.stulc.dp.setting`

Zjednodušený diagram tříd lze vidět na obrázku 5.1, kompletní pak v příloze A. Vstupním bodem aplikace je třída `ApplicationStart` umístěná v balíku `cz.zcu.fav.kiv.stulc.dp.run`. Ta určí, zda aplikace má argumenty. Pokud ano, vytvoří instanci třídy `ExpRoadMain` z balíku `cz.zcu.fav.kiv.stulc.dp.console` a argumenty jí předá. Pokud ne, vytvoří instanci třídy `MainFrame` z balíku `cz.zcu.fav.kiv.stulc.dp.gui` a spustí tak grafické prostředí.

V grafickém uživatelském prostředí si uživatel „nakliká“ vstupy a výstupy a spustí generování. `MainFrame` vytvoří argumenty, vytvoří instanci `ExpRoadMain`, argumenty jí předá a spustí ji jako nové vlákno. Dále vytvoří `ProgressDialog` z balíku `cz.zcu.fav.kiv.stulc.dp.gui` a předá mu instanci na vytvořené vlákno. `ProgressDialog` se pak pomocí třídy `Timer` z balíku `javax.swing` každých 500 ms ptá na progress vlákna a upravuje podle toho progress bar.

Spuštěné vlákno `ExpRoadMain` mezi tím vytvoří instanci implementace `IImport`, která provede import dat vtupních souborů do vlastních struktur. Následně vytvoří instanci implementace `IExport`, která za pomoci třídy `CrossRoadsCreater` vytvoří křižovatky a provede export dat do XML souboru.



Obrázek 5.1: Zjednodušený UML diagram tříd aplikace.

5.3 Výjimky

Veškeré neodchycené výjimky jsou zachyceny a zpracovány třídou `ExceptionHandler` umístěnou v balíku `cz.zcu.fav.kiv.stulc.dp.setting`. Tato třída určí, zda se jedná o očekávanou, či neočekávanou výjimku a podle toho nastaví úroveň závažnosti. Zapíše výjimku do logu *log4j*. V případě nastavení zobrazování na konzoli v konfiguračním souboru `log4j.properties` se chybová hláška zobrazí na konzoli. Pokud máme spuštěné uživatelské rozhraní, je zobrazena i chybová hláška v dialogu.

5.4 Logování

V programu je použitý systém logování *log4j* [1], který načítá buď standardní konfigurační soubor z cesty `resources/log4j.properties` nebo uživatelsky definovaný, umístěný vedle JAR aplikace. Výpis defaultního konfiguračního

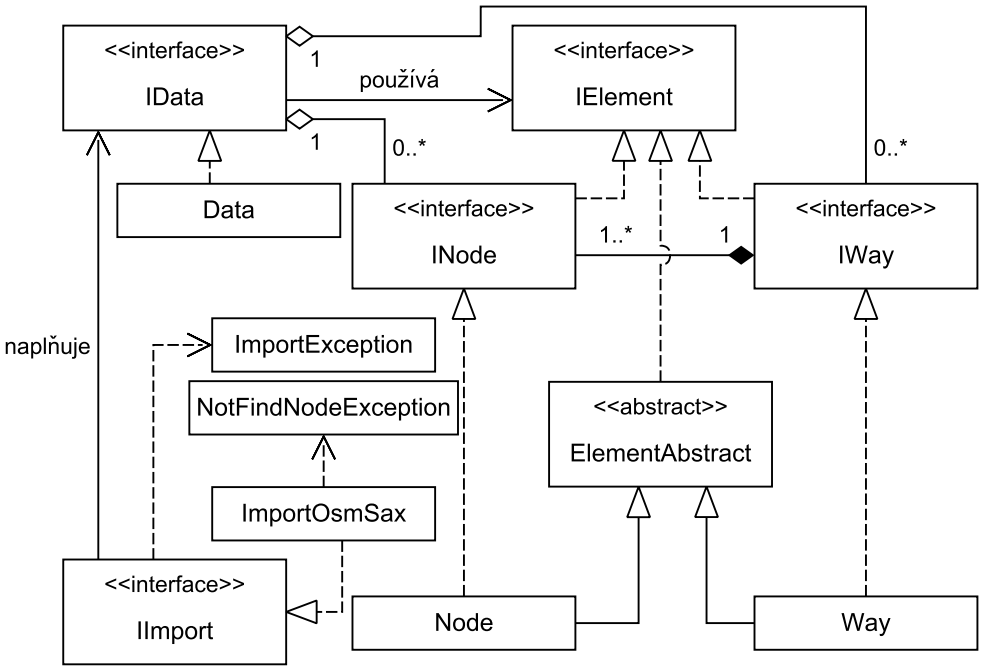
souboru nalezneme v příloze C.6.

5.5 Import dat

V této kapitole je vysvětleno, jak probíhá import vstupních dat získaných ze serveru `http://www.openstreetmap.org` (viz kapitola 3.1) do vlastních datových struktur.

5.5.1 Struktura importu

Datovou strukturu importu dat ukazuje obrázek 5.2. Import je rozdělen na datovou a aplikační vrstvu.



Obrázek 5.2: Diagram tříd importu dat.

Třídy jsou obsaženy v balících s následující strukturou:

- **cz.zcu.fav.kiv.stulc.dp.imported**

- **cz.zcu.fav.kiv.stulc.dp.imported.data**
  - IData
  - IElement
  - INode
  - IWay
  - **cz.zcu.fav.kiv.stulc.dp.imported.data.impl**
    - Data
    - ElementAbstract
    - Node
    - Way
- **cz.zcu.fav.kiv.stulc.dp.imported.app**
  - IImport
  - ImportException
  - NotFindNodeException
  - **cz.zcu.fav.kiv.stulc.dp.imported.app.impl**
    - ImportOsmSax

5.5.2 Implementace importu

Import probíhá v několika fázích:

1. Nejprve se nastaví, kolik vstupních souborů importu máme (metoda `setCountImportDataFiles()`). To je důležité pro správný odhad postupu operace.
2. Dále je potřeba nastavit cestu k importovanému souboru (metoda `setDataFileName()`).
3. Poté vytvoříme instanci implementace `IData` a předáme ji jako parametr metodě `startImport()`, která naplní `IData` daty ze souboru.
4. Pokud máme více souborů, vrátíme se k bodu 2. `IData` jsou tak postupně plněna všemi soubory.

Je zajištěna integrita dat – pokud se nějaké části objevují ve více souborech, jsou vloženy pouze jednou. O tom, zda jsou části stejné, rozhoduje jejich id. Poslední vložený prvek přepíše předchozí se stejným id.

V rámci importu jsou importovány veškeré elementy typu `node` a `way`. Vznikne-li během importu nějaká chyba, je vyhozena výjimka `ImportException` s popisem této chyby, je zobrazena uživateli, zapsána do logu a import je přerušen.

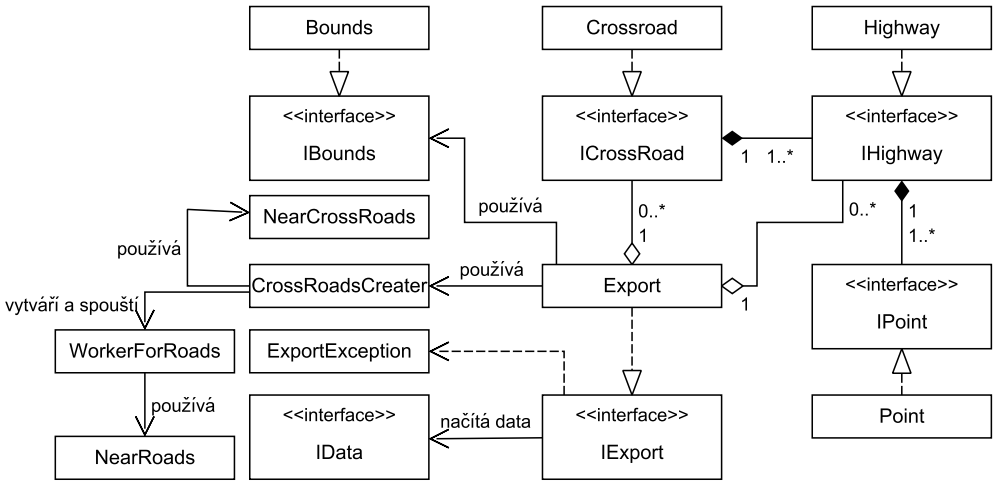
Samotné parsování dokumentu se prováděno technologií *SAX* [12] a výsledkem importu je instance implementace `IData`, která je naplněna daty ze souboru či souborů.

5.6 Export dat

Nyní, když jsou již data ve vlastních strukturách, je třeba vybrat ta, která se týkají dopravní sítě, vytvořit křižovatky a to vše exportovat do XML. Právě tím se zabývá tato kapitola.

5.6.1 Struktura exportu

Datovou strukturu exportu dat ukazuje obrázek 5.3. Export je rozdělen na datovou a aplikační vrstvu.



Obrázek 5.3: Diagram tříd exportu dat.

Třídy jsou obsaženy v balících s následující strukturou:

- **cz.zcu.fav.kiv.stulc.dp.exported**
  - **cz.zcu.fav.kiv.stulc.dp.exported.data**
    - ICrossRoad
    - IBounds
    - IHighway
    - IPoint
    - **cz.zcu.fav.kiv.stulc.dp.exported.data.impl**
      - Bounds
      - CrossRoad
      - Highway
      - Point
  - **cz.zcu.fav.kiv.stulc.dp.exported.app**
    - IExport
    - ExportException
    - **cz.zcu.fav.kiv.stulc.dp.imported.app.impl**
      - CrossRoadsCreater
      - Export
      - NearCrossRoads
      - NearRoads
      - WorkerForRoads

5.6.2 Implementace exportu

Export probíhá v několika krocích:

1. Je načten konfigurační soubor pro výstup `outformat.ini`, který určuje názvy jednotlivých elementů a atributů. Pokud existuje na cestě vedle souboru JAR, jedná se o uživatelsky upravený konfigurační soubor. Pokud tento neexistuje, je použit konfigurační soubor se standardním nastavením uložený uvnitř JAR.

2. Dalším krokem je vytvoření silnic a kolejí. Prohledává se datová struktura `IData`, ze které jsou vybrány pouze silnice a koleje a podle nich vždy upravené okraje mapy (*bounds*). Každá silnice nebo kolej bude nyní reprezentována implementací rozhraní `IRoad`. Silnice, resp. koleje

můžeme identifikovat tak, že `Way` obsahuje tag s klíčem *highway*, resp. *railway*. Tento krok zabírá 20 % práce exportu při zobrazení *progress baru*.

3. Když nyní máme strukturu všech silnic a kolejí, můžeme pomocí nich vytvořit křižovatky reprezentované implementací rozhraní `ICrossRoad`. Tímto vytvářením křižovatek se podrobněji zabývá kapitola 5.6.3. Tento krok zabírá 70 % práce exportu při zobrazení *progress baru*.

4. Dále pak začneme vytvářet dokument. Dokument je postupně vytvářen do `StringBufferu` a následně zapsán na disk. Do dokumentu je postupně zapsána hlavička, popis (`description`), okraje mapy (`bounds`), následují silnice s kolejemi (`road`) a nakonec jsou zapsány křižovatky (`crossroad`).

   Následně je dokument zformátován, je ověřena validita XML a nakonec zapsán na disk. Celý tento krok zabírá 10 % práce exportu při zobrazení *progress baru*.

U exportovaných silnic a kolejí mohou být následující vlastnosti (pokud jsou vyplněny ve zdrojových datech):

- třída silnice – jedná-li se o tramvaj je třída silnice `tram`, pokud se jedná o vlak, je třída silnice `rail`;
- jméno ulice;
- jestli se jedná o jednosměrnou ulici;
- počet pruhů;
- české značení silnic;
- mezinárodní značení silnic;
- maximální povolená rychlost;
- vertikální úroveň silnice.

5.6.3 Vytvoření křižovatek

Algoritmus pro vytváření křižovatek je následující (předávání indexů ukazuje obrázek 5.4):

1. Třída `CrossRoadsCreater` vytvoří vlákna pro vytváření křižovatek `WorkerForRoads`. Jejich počet závisí na konstantě *number_of_workers* z konfiguračního souboru `crossroads.ini`.

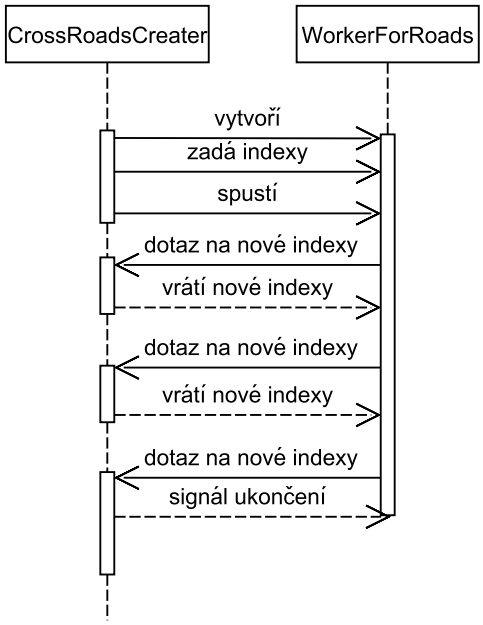
2. Každému vytvořenému vláknu je předáno pole všech silnic, první indexy pro zpracování (jejich počet závisí na konstantě *load_balanced* taktéž z v souboru `crossroads.ini`) a jednotlivá vlákna jsou spuštěna.

3. Hlavní program poté čeká na dokončení jednotlivých vláken, které vytvářejí křižovatky (jejich popis nalezneme dále).

4. Od každého vlákna si následně vyžádá jeho vytvořené křižovatky a uloží si je.

5. Po dokončení všech vláken třída `CrossRoadsCreater` projede všechny křižovatky a na základě konstanty *distance_one_crossoads* z konfiguračního souboru `crossroads.ini` zjistí, které z nich jdou sloučit do jedné a sloučí je.

6. Křižovatky jsou vytvořené a připravené pro uložení do souboru.



Obrázek 5.4: Sekvenční diagram předávání indexů.

O tom, zda jsou body "blízké" (použito v následujícím textu) rozhoduje konstanta *distance_near_points* uložená v souboru `crossroads.ini`. Každé z vláken reprezentovaných třídou `WorkerForRoads` pracuje podle následujícího algoritmu:

1. Je-li příznak ukončení aktivní, vlákno ukončí svou činnost.
2. Projíždíme silnice s přiřazenými indexy a každou takovouto silnici označíme jako referenční.
3. Pro každou referenční silnici testujeme všechny následující, zda nějaký bod neleží blízko nějakého bodu referenční silnice. Pokud se tak stane, označíme silnice jako křižovatku a geometrický střed mezi těmito body silnic jako současný střed křižovatky.
4. Zapamatujeme si index silnice, která byla takto nalezena a pro další testujeme jen jestli mají bod poblíž tohoto středu křižovatky. Pokud mají, přidáme je k současným a nový střed křižovatky bude geometrický střed všech účastnících se bodů. Tím zjistíme další účastníky nově vzniklé křižovatky.
5. Po prozkoumání všech silnic se podíváme na vytvořenou křižovatku a zjistíme, jestli se nejedná pouze o dvě navazující silnice. Pokud ano, křižovatku zahodíme, pokud ne, křižovatku uložíme.
6. Vrátíme se k bodu 3, ovšem až od zapamatovaného indexu. Tím zjišťujeme, zda referenční silnice není součástí další křižovatky.
7. Po nalezení veškerých křižovatek pro referenční silnici, přejdeme k další referenčí silnici a k bodu 3. Po vyčerpání všech přiřazených indexů pro referenční silnice přejdeme k bodu 8.
8. Řekneme si o nové indexy třídě `CrossRoadsCreater`, která nám nové indexy přidělí a zvedne *progress* podle počtu zpracovaných dat. Pokud již nejsou žádné indexy k dispozici, nastaví příznak ukončení.
9. Přejdeme k bodu 1.

5.7 Konzole

Tato kapitola se zabývá parsováním argumentů konzolové aplikace.

5.7.1 Struktura konzole

Třídy jsou obsaženy v balících s následující strukturou:

- **cz.zcu.fav.kiv.stulc.dp.console**
  - ExpRoadMain
  - Switchers

5.7.2 Implementace konzole

Parsování konzole probíhá v následujících krocích:

1. je počet parametrů roven nule – výjimka `ImportException` – nejsou žádné parametry;
2. pokud jsou nějaké parametry, tak je začneme procházet;
3. je-li nalezen nějaký přepínač definovaný ve třídě `Switchers`, zjistíme, zda není signalizován jiný přepínač a případně ho uzavřeme; dále signalizujeme nalezený přepínač;
4. není-li v aktuálním kroku nalezen přepínač, zjistíme zda není nějaký signalizovaný, a pokud ano, vezmeme hodnotu argumentu jako hodnotu vlastnosti, kterou definuje signalizovaný přepínač; pokud nebyl signalizován žádný přepínač je vyhozena výjimka `ImportException` – neznámý argument;
5. byl-li signalizovaný přepínač jednohodnotový, po načtení hodnoty ho uzavřeme, pokud ne, necháme ho otevřený a v příštím kroku očekáváme další hodnotu, či jiný přepínač, který ho uzavře;
6. vracíme se k bodu 3, dokud nevyčerpáme všechny argumenty.

Z algoritmu je vidět, že nezáleží na pořadí argumentů. Třída `Switchers` definuje tři přepínače `-i, -o, -d`, které definují vstupní soubory, výstupní soubor a popis exportované mapy.

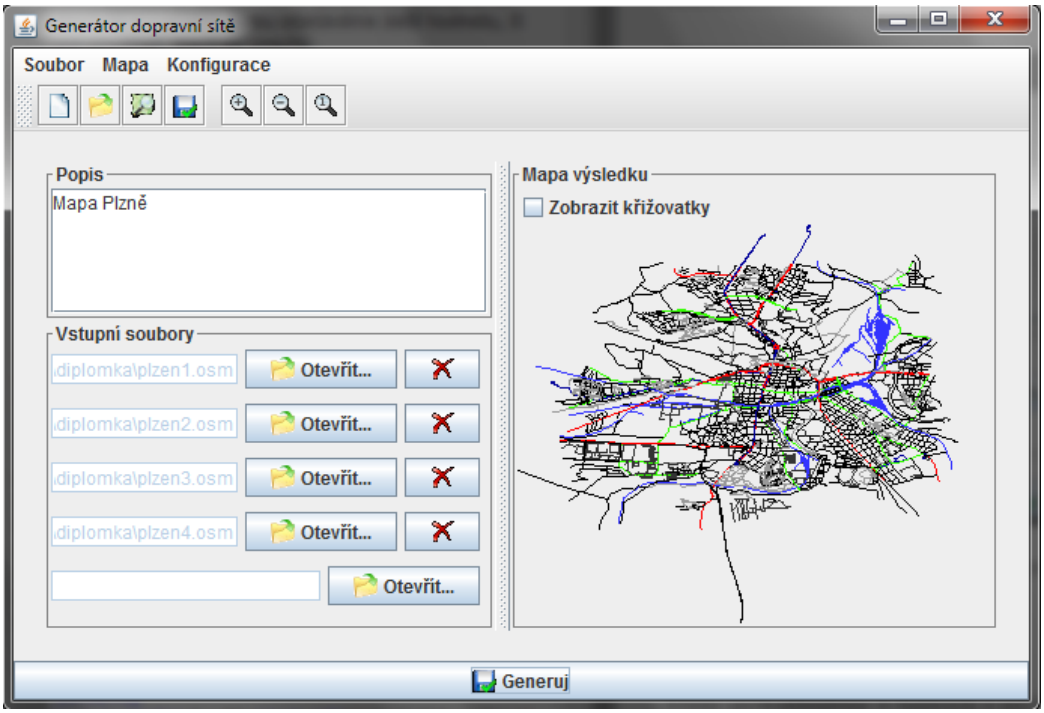
Všechny získané hodnoty jsou následně předány *importu* a *exportu* dat, které se postarají o aplikační část programu.

5.8 GUI

V této kapitole je popsáno rozvržení a implementace grafického uživatelského rozhraní.

5.8.1 Rozvržení GUI

Grafické uživatelské rozhraní hlavního okna je na obrázku 5.5. V levé části



Obrázek 5.5: Hlavní okno grafického uživatelského rozhraní.

okna vybíráme vstupní soubory a definujeme popis mapy. Při každém přidání souboru se přidá i okénko pro přidání dalšího souboru. Každý vstupní soubor jde také změnit nebo smazat. V pravé části je prostor pro zobrazení mapy. Veškeré akce je možné provádět pomocí menu nahoře. Akce mimo konfigurace pak i pomocí ikon v tool baru.

Pro konfiguraci jsou volány dialogy, které umožňují měnit uživateli konfigurační soubory přímo z GUI. Mapa je načítána buď automaticky, ihned

po generování výstupního souboru, nebo může být načtena z již vygenerovaného souboru pomocí třetí ikony v toolbaru. Pomocí zaškrtávacího políčka `zobrazit křižovatky` můžeme zvolit, zda chceme, aby se do mapy zobrazovaly i vypočítané křižovatky.

5.8.2 Struktura GUI

Třídy jsou obsaženy v balících s následující strukturou:

- **cz.zcu.fav.kiv.stulc.dp.gui**
  - MainFrame
  - OpenField
  - ProgressDialog
  - **cz.zcu.fav.kiv.stulc.dp.gui.config**
    - CrossroadsPropertiesDialog
    - ColorPropertiesDialog
    - MemoryButton
    - OutputPropertiesDialog
    - PropertyComponent
    - RemoveButton
  - **cz.zcu.fav.kiv.stulc.dp.gui.filefilters**
    - OsmFileFiletr
    - XmlFileFilter
  - **cz.zcu.fav.kiv.stulc.dp.gui.map**
    - LoadXMLResultException
    - PrintingMap

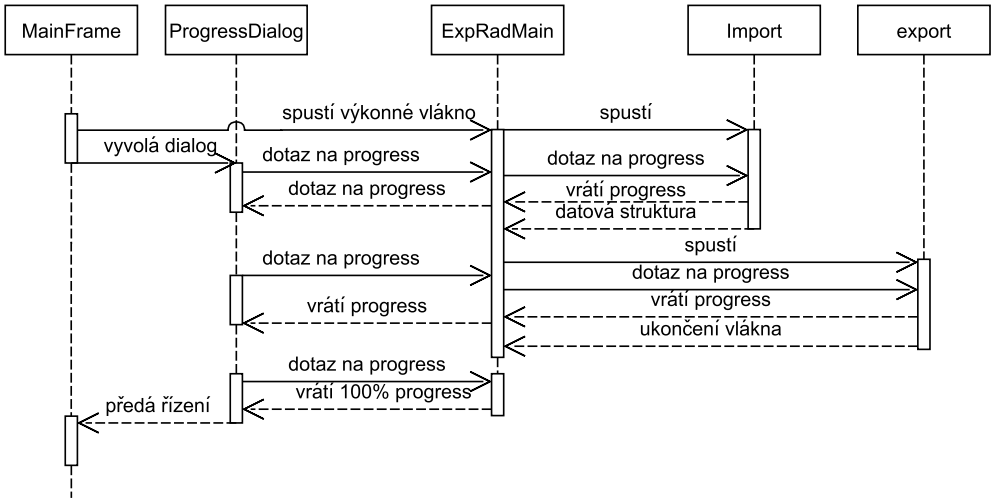
5.8.3 Implementace GUI

Grafické uživatelské prostředí umožňuje několik různých akcí:

- zobrazení již vygenerované mapy a její následné zvětšování/zmenšování;

- úprava konfiguračních souborů;
- načtení vstupních souborů;
- vygenerování výstupního souboru z načtených vstupních souborů s následným zobrazením mapy výsledku.

Při spuštění generování výstupu se vytvoří výkonné vlákno, které provádí samotný import a export. Hlavní vlákno vyvolá dialog s *progressBary* pro import i export. Ten se pak táže výkonného vlákna kolik procent již má hotovo. Podle toho se upraví *progressBary*. Po dokončení generování a potvrzení dialogu se zobrazí mapa. Předávání řízení lépe ukazuje obrázek 5.6.



Obrázek 5.6: Sekvenční diagram generování výstupu.

5.8.4 Zobrazení mapy

Mapa je vykreslována do komponenty děděné od `JPanel`.

1. Komponentě je předán XML soubor s exportovanou dopravní sítí.
2. Je načten konfigurační soubor výstupu a pomocí něho a technologie SAX je soubor parsován a je uložena dopravní síť jako seznam silnic.

3. *Koeficient velikosti* (viz dále) je nastaven na 1.
4. Před samotným vykreslením se pomocí *bounds* a velikosti komponenty spočítá měřítko, ve kterém bude mapa zobrazena.
5. Dále je načten konfigurační soubor s barvami mapy. U každé silnice se podle její třídy určí barva, pokud není obsažena v konfiguračním souboru vybere se černá.
6. Spojováním jednotlivých bodů se vykreslí silnice.
7. Takto se vykreslí všechny silnice.
8. Je-li zaškrtnuto zobrazování křižovatek, vykreslí se kružnice na každý střed křižovatky.
9. Pro zvětšování a zmenšování mapy slouží *koeficient velikosti*. Při každé jeho změně, je upravena velikost vykreslovací komponenty, tím se upraví i měřítka a v případě nutnosti se objeví posuvníky.
10. Pokud chceme vidět opět celou mapu, stačí změnit koeficient velikosti zpět na 1.

5.9 Konfigurační soubory

Výpis lokalizačních a konfiguračních souborů `outformat.ini`, `colors.ini` a `crossroads.ini` se nachází v příloze C. Všechny tyto konfigurační soubory je možné upravovat z grafického prostředí programu.

5.9.1 Lokalizace

Program je lokalizovaný v češtině a angličtině s možností snadného rozšíření o další jazyky.

Veškeré texty a chybová hlášení jsou uloženy v properties souborech umístěných v `resources\localization`. Properties soubory jsou v kódování UTF-8. Pro lokalizaci je použita třída `Bundle` z balíku `cz.zcu.fav.kiv.stulc.dp.setting`, která využívá třídu Javy `ResourceBundle` [11] z balíku `java.util`.

Lokalizace je automaticky vybrána programem podle národního prostředí. Není-li pro dané národní prostředí dostupná, je použita angličtina. Pro vytvoření nové lokalizace stačí vytvořit properties soubor pro daný jazyk a uložit je do výše zmíněné složky.

5.9.2 outformat.ini

Tento konfigurační soubor obsahuje informace o pojmenování elementu ve výstupním XML souboru. Je také důležitý pro zobrazení mapy. Aby byla mapa správně zobrazena je důležité, aby výstupní soubor byl vygenerován se stejným konfiguračním souborem `outputformat.ini` jaký je použit pro její načítání.

Konfigurační soubor má strukturu `klíč=hodnota`, kde klíč je určené jméno elementu a hodnota je jméno, které bude použito ve výstupním souboru.

5.9.3 colors.ini

Tento konfigurační soubor obsahuje informace o barvách zobrazených na mapě jednotlivých tříd silnic či kolejí.

Konfigurační soubor má strukturu `klíč=hodnota`, kde klíč je třída silnice či kolejí a hodnota je barva uložená jako integer, jehož byty obsahují RGB informaci. Touto barvou bude daná třída silnice zobrazena při vykreslení mapy. Není předem znám výčet všech tříd silnic a je na uživateli, které do konfigurační souboru zadá.

5.9.4 crossroads.ini

Tento konfigurační soubor obsahuje konstanty potřebné pro vytváření křižovatek. Popis konstant je následující:

- `number_of_workers` – počet vláken (workerů), které budou vytvářet křižovatky;
- `load_balanced` – počet indexů, které budou přiřazeny jednomu vláknu najednou;

- `distance_one_crossoads` – maximální vzdálenost (v GPS souřadnicích) středů křižovatek, aby byly sloučeny do jedné;
- `distance_near_points` – maximální vzdálenost (v GPS souřadnicích) bodů, aby silnice, které jimi procházejí, byly brány za členy křižovatky.

6 Testování

V této kapitole je aplikace otestována, zda splňuje požadavky zadavatele a zda generuje správný výstup. Testování probíhalo na dvoujádrovém procesoru Intel Core Duo pod operačním systémem Windows 7.

6.1 Testování rychlosti

První testování proběhlo pro čtyři vstupní soubory zahrnující oblast Plzně s následujícími koeficienty:
load_balanced=100
distance_one_crossoads=0.00028
distance_near_points=0.00003
Vygenerováno bylo 3283 silnic a 2470 křižovatek.

Tabulka 6.1: Tabulka časů pro generování Plzně.

Měření	**Počet vláken**	**Čas**
1	1	26,886 s
2	1	27,224 s
3	1	28,015 s
4	1	27,693 s
5	1	27,886 s
Průměr	1	27,541 s
1	2	23,993 s
2	2	18,989 s
3	2	19,044 s
4	2	18,631 s
5	2	19,115 s
Průměr	2	19,954 s

Druhé testování proběhlo pro čtrnáct vstupních souborů zahrnující oblast Brna s následujícími koeficienty:
load_balanced=100
distance_one_crossoads=0.00028
distance_near_points=0.00003

Vygenerováno bylo 15113 silnic a 9291 křižovatek.

Tabulka 6.2: Tabulka časů pro generování Brna.

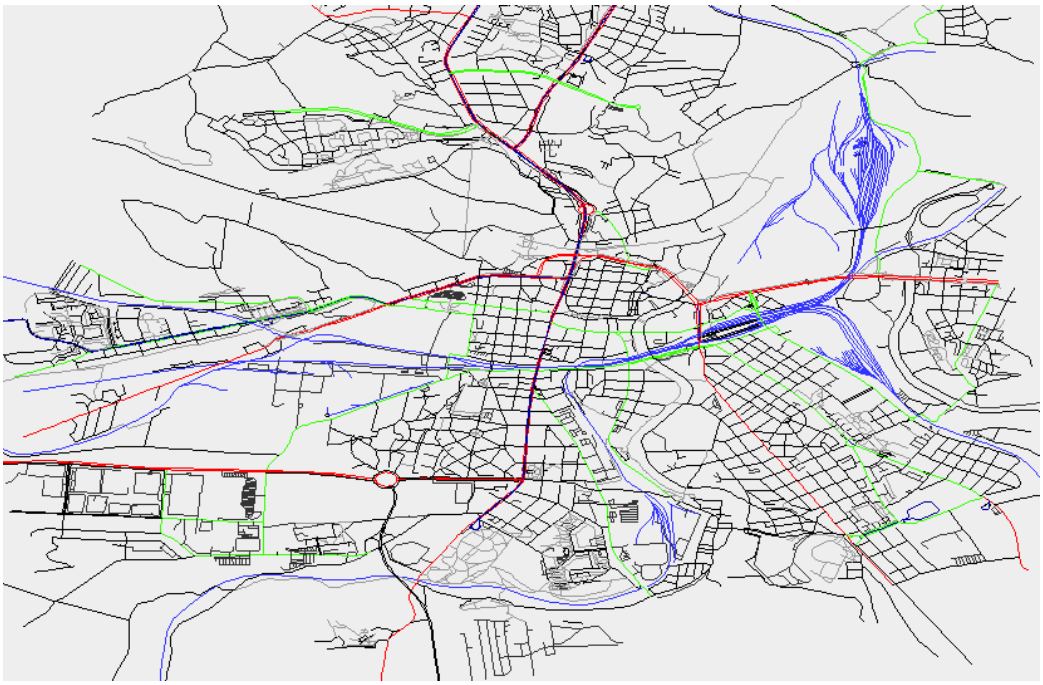
Měření	**Počet vláken**	**Čas**
1	1	7 min 28,532 s
2	1	7 min 29,034 s
3	1	7 min 32,150 s
4	1	7 min 36,749 s
5	1	7 min 37,060 s
Průměr	1	7 min 26,275 s
1	2	5 min 1,906 s
2	2	4 min 57,446 s
3	2	5 min 25,599 s
4	2	5 min 9,795 s
5	2	5 min 11,036 s
Průměr	2	5 min 9,156 s

6.2 Testování generování silnic

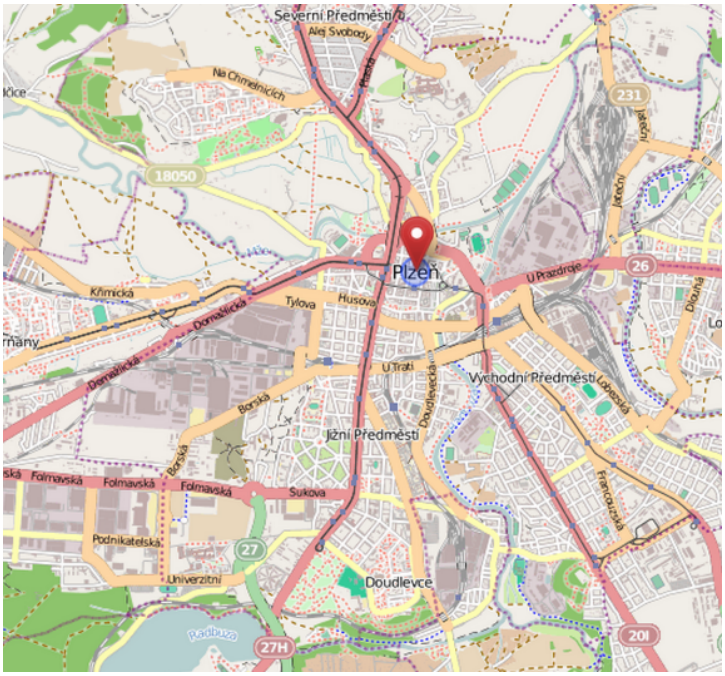
Správné fungování generování silnic se nejlépe ověří porovnáním zobrazené dopravní sítě s mapou. Mapy pro Plzeň jsou vidět na obrázcích 6.1 a 6.2. Plzeň obsahuje 3283 silnic a kolejí. Z podobnosti obrázků je vidět, že algoritmus funguje správně.

6.3 Testování algoritmu křižovatek

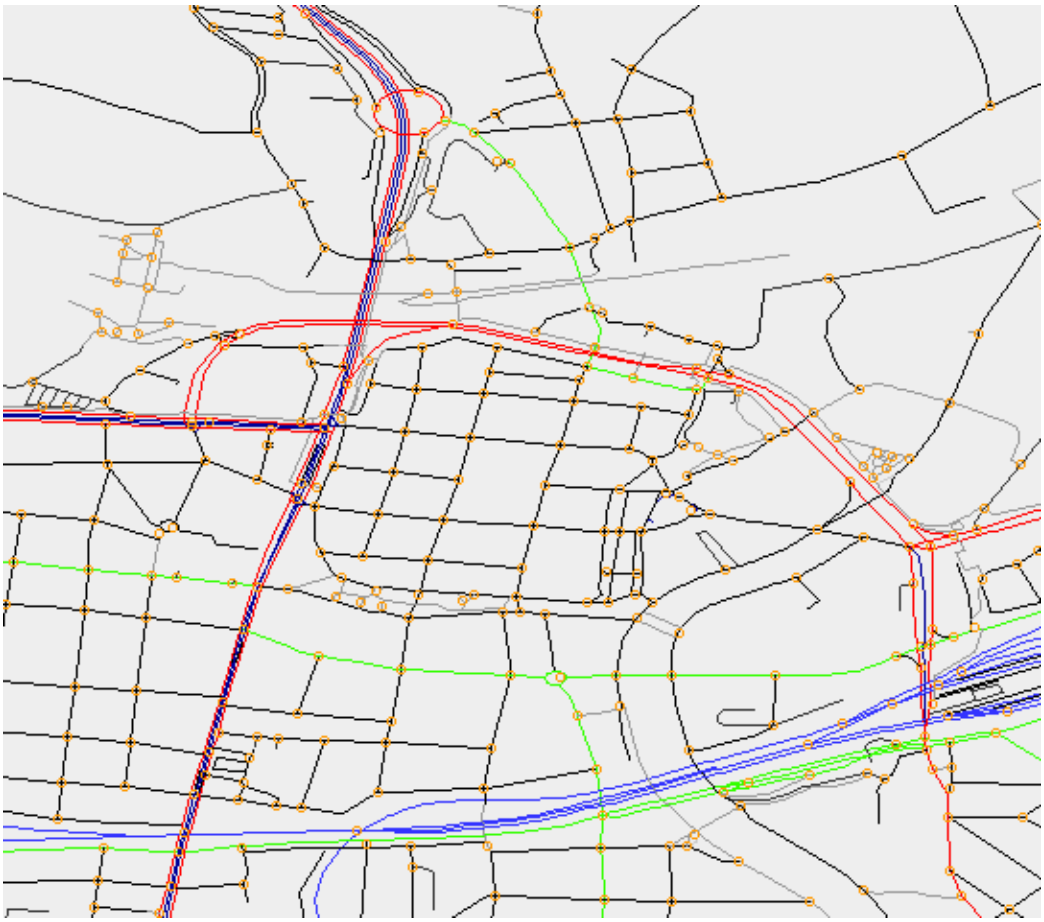
Správné fungování algoritmu pro generování křižovatek ověříme zakreslením křižovatek na mapu silnic. Pokud jsou skutečně na křižovatkách, tak jsou vygenerovány správně (viz obrázek 6.3). Je vidět že naprostá většina křižovatek je na správném místě, ale existuje pár křižovatek, které jsou označeny jako křižovatky, přestože by neměly. Například kruhový objezd je chybně označen jako několik křižovatek. Na druhou stranu silnice pod ním, která je mimoúrovňová správně jako křižovatka označena nebyla, přestože kruhový objezd protíná. Jak již bylo řečeno v kapitole 4.3, algoritmus je závislý na dvou kon-



Obrázek 6.1: Mapa Plzně z vygenerovaného souboru.



Obrázek 6.2: Mapa Plzně z openstreetmap.org.



Obrázek 6.3: Výřez mapy Plzně se zobrazením křižovatek.

stantách, jejichž změnou ovlivňujeme přesnost algoritmu. Závislost na konstantách je otestována v následujících kapitolách.

6.3.1 Závislost algoritmu na konstantě distance_near_points

Jak již bylo uvedeno v kapitole 4.3, konstanta `distance_near_points` určuje maximální vzdálenost bodů od sebe, aby tyto body byly brány jako křižovatka. Tabulka 6.3 ukazuje, jak se změna této konstanty promítá do počtu vygenerovaných křižovatek v Plzni. Všechna měření byla provedena s konstantní hodnotou `distance_one_crossoads` = 0,00028.

Tabulka 6.3: Tabulka počtu křižovatek v závislosti na konstantě distance_near_points.

Měření	Konstanta	Počet křižovatek	Čas běhu
1	0	2462	18,678 s
2	0.00001	2463	18,899 s
3	0.0001	2562	20,364 s
4	0.001	2813	1 min 9,482 s
5	0.002	3061	3 min 39,202 s

Měření 1 ukazuje kolik křižovatek má jeden společný bod. Další pak vzrůstající vzdálenost mezi body, při které jsou stále chápány jako křižovatka. V měření 4 a 5 vidíme velký nárůst času i počtu křižovatek. To je způsobeno tím, že konstanta je již tak velká, že u některých (blízko sebe ležících) silnicích jsou všechny body označeny za křižovatku. Nárůst křižovatek je navíc zmírněný tím, že blízké křižovatky jsou slučované do jedné. Optimální hodnotu konstanty je nutné zjistit experimentálně. Optimální hodnota konstanty `distance_near_points` byla určena na *0,00003*.

6.3.2 Závislost algoritmu na konstantě distance_one_crossoads

Jak již bylo uvedeno v kapitole 4.3, konstanta `distance_one_crossoads` určuje maximální vzdálenost středů křižovatek od sebe, aby byly tyto křižovatky sloučeny do jedné. Tabulka 6.4 ukazuje, jak se změna této konstanty promítá do počtu vygenerovaných křižovatek v Plzni. Všechna měření byla provedena s konstantní hodnotou `distance_near_points` = 0,00003.

Tabulka 6.4: Tabulka počtu křižovatek v závislosti na konstantě distance_one_crossoads.

Měření	Konstanta	Počet křižovatek	Čas běhu
1	0	3524	20,450 s
2	0.00001	3488	20,678 s
3	0.0001	3239	21,392 s
4	0.001	1040	16,897 s
5	0.01	38	15,327 s
6	0.1	1	15,846 s

Měření 1 ukazuje vygenerovaný počet křižovatek bez jakéhokoli slučování. U dalších měření roste maximální možná vzdálenost pro sloučení křižovatek, a proto počet křižovatek klesá. Extrémem je pak měření 6, kde je vzdálenost natolik veliká, že všechny plzeňské křižovatky byly sloučeny do jedné jediné ve středu Plzně. Optimální hodnotu konstanty je nutné zjistit experimentálně. Optimální hodnota konstanty `distance_one_crossoads` byla určena na *0,00028*.

6.3.3 Shrnutí

Z naměřených hodnot je vidět, že čím je konstanta `distance_near_points` větší, tím roste počet vygenerovaných křižovatek a také čas potřebný ke zpracování. Na druhou stranu, čím je větší konstanta `distance_one_crossoads`, tím počet křižovatek klesá a mírně klesá i čas.

Určení obou konstant je nutné povádět experimentálně a to tak, že hledáme optimální kombinaci, při které budou zobrazeny všechny křižovatky, ale velké a složité křižovatky budou sloučeny do jedné. Úplně optimální řešení nalézt nelze, jelikož soustava malých křižovatek může mít středy křižovatek blíže než jsou středy pseudo křižovatek ve složité křižovatce. Je tedy pouze na uživateli, zda raději obětuje přesnost malých křižovatek, ale bude mít velké křižovatky vedené jako jednu křižovatku, či zda zachová přesnost malých křižovatek, ale velké se mu rozpadnou na více malých.

6.4 Vzorový příklad

Abychom se nezabývali pouze Plzní, ukážeme si vzorový případ generování pro malé město na severu Čech zvané Litvínov.

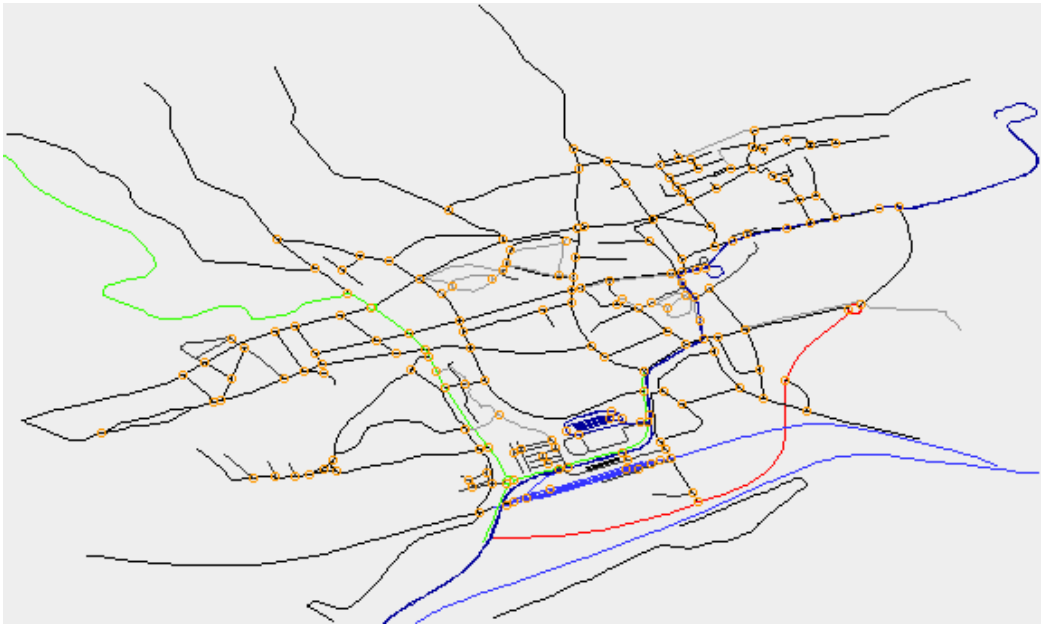
1. Nejprve si uživatel stáhne zdrojový soubor ze serveru `openstreetmap.org`, město je malé, postačí jeden.
2. Spustí aplikaci a stažený soubor nahraje jako vstupní soubor.
3. Vyplní popis mapy.
4. Parametry pro generování křižovatek nechá standardní.

5. Nechá generovat výstup. Generování trvalo 3,222 s a bylo vygenerováno 238 silnic a 175 křižovatek pro standardní parametry pro generování křižovatek.

6. Zobrazí se mu mapa vygenerované dopravní sítě.

7. Nechá si zobrazit křižovatky a ujistí se, že jsou na správných místech (viz obrázek 6.4).

8. V případě, že by uživatel nebyl spokojen, upraví parametry pro generování křižovatek a přejde k bodu 5.



Obrázek 6.4: Dopravní síť Litvínova se zobrazením křižovatek.

7 Závěr

Cílem této práce bylo nalézt veřejně dostupný zdroj informací o dopravní síti. Tento zdroj analyzovat a zjistit, jaké informace poskytuje, popřípadě vymyslet, jak získat informace, které přímo neposkytuje.

Mezi několika možnými zdroji byl vybrán server `openstreetmap.org`, který umožňuje export mapy do XML souboru s příponou *osm*. Bylo zjištěno, že obsahuje téměř všechny potřebné informace, včetně mnoha informací navíc. Jedinou informací, kterou neobsahuje, jsou údaje o polohách křižovatek silnic. Tento nedostatek byl vyřešen vytvořením algoritmu, který, na základě dostupných dat o polohách pozemních komunikací, vypočítal polohu křižovatek včetně komunikací, které do této křižovatky vstupují, resp. z ní vystupují.

Pro zpracování dat ze serveru `openstreetmap.org` byla vytvořena aplikace, která umožňuje načíst data ze souborů *osm* a transformovat je do XML souboru a to jak pomocí konzole (pro případ dávkového zpracování dat), tak s využitím grafického uživatelského prostředí.

Práce zcela splňuje zadání. Vytvořená aplikace umožňuje načtení několika vstupních souborů získaných ze serveru `openstreetmap.org`, vybere data týkající se silnic, pomocí nich určí středy křižovatek a seznam jejich účastníků. Všechna takto získaná data následně uloží do XML souboru. V případě grafického rozhraní, je takto vytvořený soubor možné načíst a zobrazit mapu dopravní sítě s možností zobrazení vypočtených pozic křižovatek. Vytvořená aplikace byla důkladně otestována.

Literatura

[1] The Apache Software Foundation: *log4j – manual*. 2010, 27. 4. 2011.
URL <http://logging.apache.org/log4j/1.2/manual.html>

[2] The Apache Software Foundation: *Apache Ant – Manual*. 2013, 20. 4. 2013.
URL <http://ant.apache.org/manual/index.html>

[3] Břehovský, M.; Jedlička, K.: *Úvod do GIS – přednáškové texty*. 1. 12. 2012.
URL <http://gis.zcu.cz/studium/ugi/e-skripta/ugi.pdf>

[4] Google: *Dodatečné smluvní podmínky Map Google a Google Earth*. 2013, 28. 4. 2013.
URL <http://www.google.com/intl/cs_cz/help/terms_maps.html>

[5] Macháček, J.: *Analýza obrazu I*. 1. 12. 2012.
URL <http://www.vscht.cz/sil/download/Prednaska-OA-08-I.pdf>

[6] Mapy.cz, s.r.o.: *Licenční podmínky mapových podkladů – Mapy.cz*. 2013, 28. 4. 2013.
URL <http://napoveda.seznam.cz/cz/mapy-licencni-podminky.html>

[7] Mapy.cz, s.r.o.: *Mapy API verze 4.8 – Hanzelka a Zikmund*. 2013, 24. 4. 2013.
URL <http://api4.mapy.cz/>

[8] Matoušek, V.: *Rozpoznávání a klasifikace*. 1. 12. 2012.
URL <http://www.kiv.zcu.cz/studies/predmety/tks/Texty2/FThema1.pdf>

[9] OpenStreetMap.org: *Autorská práva a licence – openstreetmap.org*. 2013, 28. 4. 2013.
URL <http://www.openstreetmap.org/copyrightl>

[10] OpenStreetMap.org: *OpenStreetMap Wiki*. 2013, 18. 3. 2013.
URL <http://wiki.openstreetmap.org/wiki/Main_Page>

[11] Oracle: *Class ResourceBundle*. 2011, 16. 4. 2011.
URL <http://download.oracle.com/javase/1.4.2/docs/api/java/util/ResourceBundle.html>

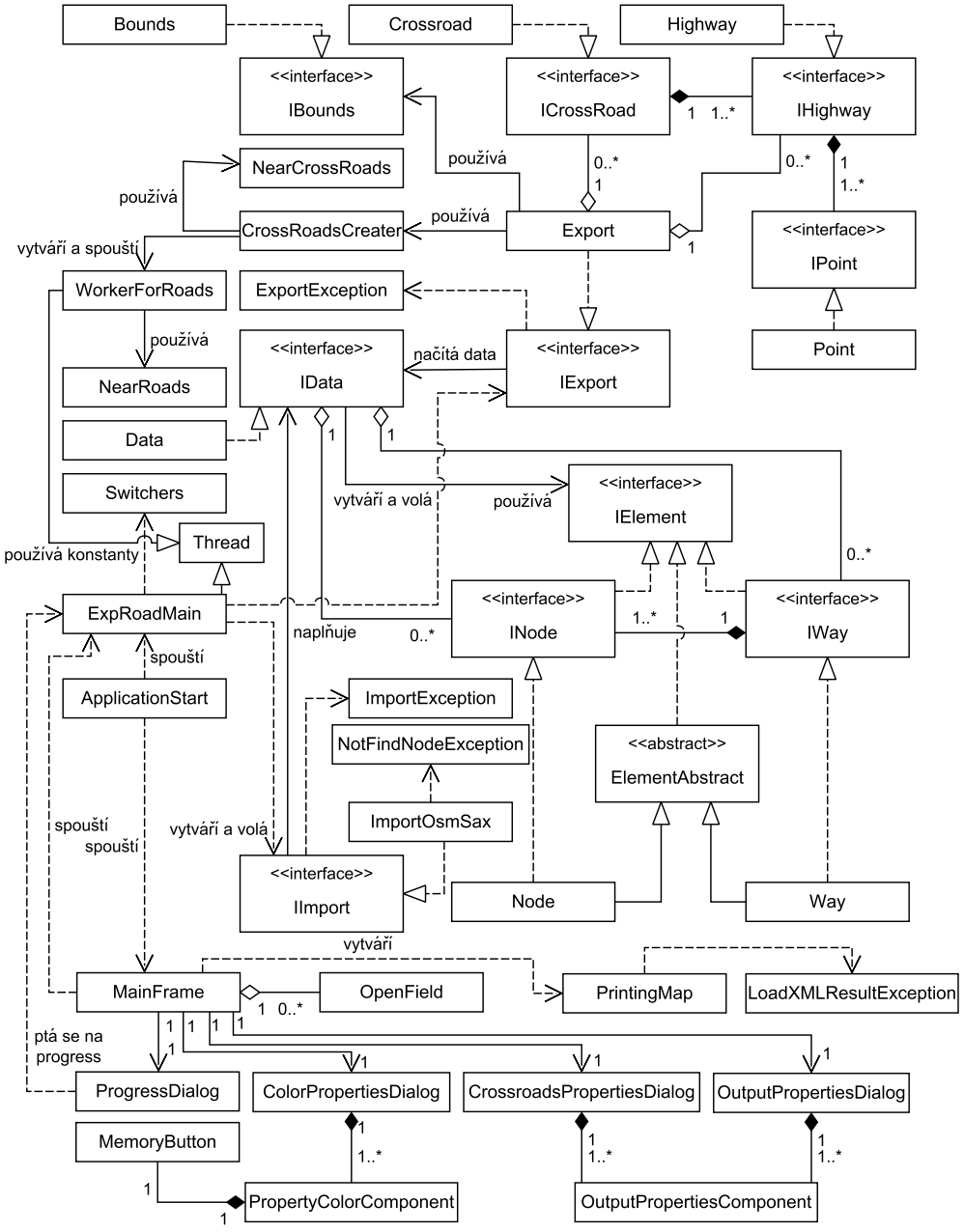
[12] Oracle: *Parsing an XML File Using SAX*. 2013, 10. 2. 2013.
URL <http://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html>

[13] Oracle: *Reading XML Data into a DOM*. 2013, 10. 2. 2013.
URL <http://docs.oracle.com/javase/tutorial/jaxp/dom/readingXML.html>

[14] Spell, B.: *Java Programujeme profesionálně, ISBN:80–7226–667–5*. Computer Press, 2002.

Přílohy

A Komletní UML diagram tříd



B Uživatelská příručka

Tato kapitola obsahuje návody pro jednotlivé akce uživatele.

B.1 Získání dat

1. V internetovém prohlížeči zadáme internetovou adresu `http://www.openstreetmap.org`.
2. Do okénka `hledat` (

), které je umístěné vlevo nahoře, napíšeme, jakou část mapy hledáme (např. Plzeň).
3. Objeví se nám výsledky vyhledávání (obrázek B.1). Klikneme na požadovaný výsledek a na mapě se nám zobrazí požadované místo.
4. Upravíme mapu tak, aby zahrnovala výřez, který chceme použít.
5. Stiskneme tlačítko `export` (

) v menu nahoře.
6. Pomocí tlačítka `Ručně vybrat jinou oblast` (viz obrázek B.2) vybereme oblast, kterou chceme exportovat. Ale pozor, oblast nesmí být moc velká. Případem, že se tak stane, se budem zabývat později. (Pozn. pro export celé Plzně je třeba rozdělit mapu v průměru do čtyř částí.)
7. Dále jako formát exportu vybereme možnost `Data OpenStreetMap XML` (viz obrázek B.2).
8. Stiskneme tlačítko `Export`.
9. V případě, že jsme zvolili velikost dat rozumné velikosti, dojde ke stažení požadovaných zdrojových dat.

   V případě, že je oblast moc velká, zobrazí se nám chybová zpráva: "`You requested too many nodes (limit is 50000). Either request a smaller area, or use planet.osm`". V tom případě stiskneme v prohlížeči tlačítko zpět a vrátíme se k bodu 5. Při výběru oblasti, ale zvolíme oblast menší.

Výběr oblastí je vhodné volit tak, aby se navzájem mírně překrývaly a nevznikla tak ztráta dat. Výsledkem je *XML* soubor s příponou *.OSM*.

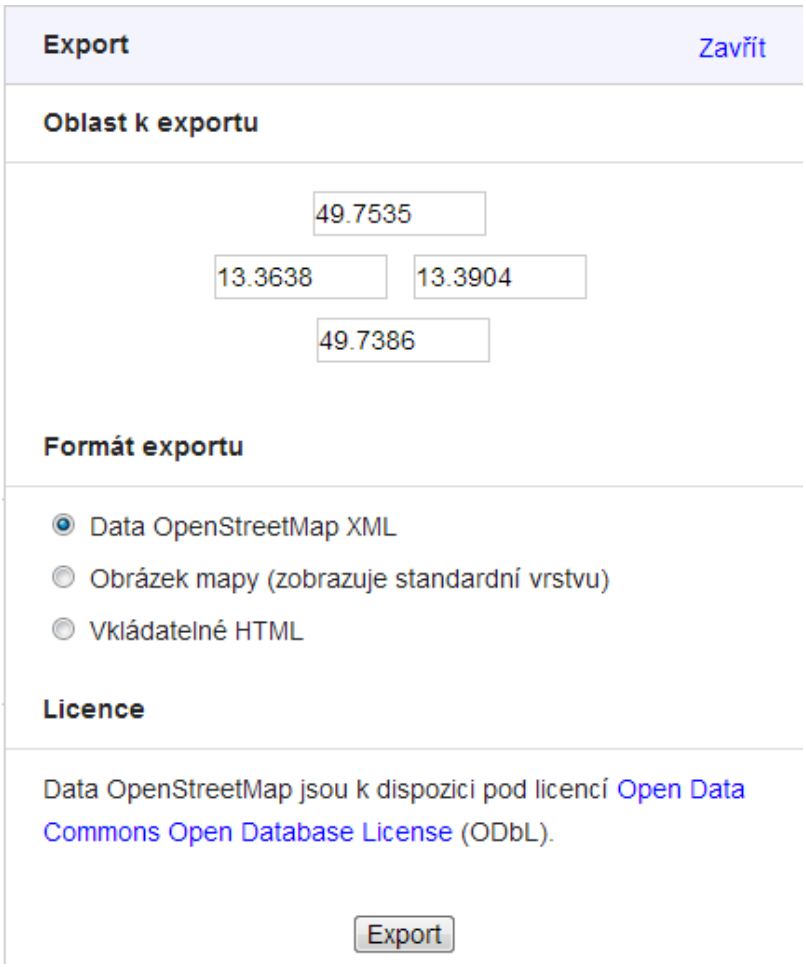
Obrázek B.1: Výsledky vyhledávání.

B.2 Překlad a sestavení programu

Program je sestavován pomocí technologie ANT [2], který je nutný mít nainstalovaný. Dále je nutné mít nainstalovánu Javu JDK 1.7. Překlad a sestavení provedeme následovně:

1. spustíme příkazový řádek;
2. změníme aktuální adresář na `dp_stulc_fav_zcu_cz`;
3. zadáme příkaz `ant`;
4. sestavení proběhne a objeví se zpráva `BUILD SUCCESSFUL`.

Výsedný JAR soubor spolu potřebnými knihovnami nalezneme ve složce `dp_stulc_fav_zcu_cz/jar`.



Obrázek B.2: Výsledky vyhledávání.

B.3 Konzole

V této kapitole je uvedeno jak spustit konzolovou aplikaci a jak pomocí ní generovat výstupní soubor.

B.3.1 Spuštění

Pokud spustíme JAR soubor spolu s argumenty, bude spuštěn jako konzolová aplikace. V těchto argumentech použijeme následující přepínače:

- `-o` za nímž následuje relativní cesta k výstupnímu souboru. Pokud soubor již existuje, bude přepsán;
- `-i` za nímž následuje seznam relativních cest ke vstupním souborům;
- `-d` za nímž následuje popis vytvořené dopravní sítě. Tento přepínač není povinný.

Nezáleží na pořadí jednotlivých přepínačů. Pokud je výstup či popis použit opakovaně, je brána v úvahu pouze poslední udaná hodnota. V případě opakovaného použití `-i` jsou brány v úvahu všechny vstupní soubory.

Např.:
`java -jar dopravni_sit_console.jar -i vstup1.osm "vstup2.osm" -o vystup.xml -d "popis mapy" -i vstup3.osm`

Kde vstupy jsou: `vstup1.osm, vstup2.osm, vstup3.osm`; výstup je `vystup.xml` a popis mapy má hodnotu `popis mapy`.

Po spuštění proběhne import jednotlivých vstupních souborů následovaný exportem do výstupního souboru. O každém provedeném importu vstupního souboru a exportu výstupního je proveden záznam do logu a dle nastavení `log4j.properties` i zobrazen záznam na standardní výstup.

B.3.2 Příklad

Máme čtyři vstupní soubory `plzen1.osm`, `plzen2.osm`, `plzen3.osm`, `plzen4.osm` a chceme vytvořit dopravní síť do souboru `plzen.xml` s popisem mapy "město Plzeň".

Spusíme příkazový řádek a zadáme:
`java -jar dopravni_sit.jar -i plzen1.osm plzen2.osm plzen3.osm plzen4.osm -o plzen.xml -d "město Plzeň"`
a potvrdíme. Program bude pracovat a zobrazí se nám následující výsledky, které nalezneme i v souboru `dopravni_sit.log`:

```
1 2013-04-30 15:29:12,877 INFO  ImportOsmSax Import from 'C:\
    Diplomka\plzen1.osm' done
2 2013-04-30 15:29:13,339 INFO  ImportOsmSax Import from 'C:\
    Diplomka\plzen2.osm' done
3 2013-04-30 15:29:14,237 INFO  ImportOsmSax Import from 'C:\
    Diplomka\plzen3.osm' done
```

```
4 | 2013-04-30 15:29:15,541 INFO  ImportOsmSax Import from 'C:\
      Diplomka\plzen4.osm' done
5 | 2013-04-30 15:29:23,847 INFO  Export Export to 'C:\Diplomka\
      plzen.xml' done
6 | 2013-04-30 15:29:23,847 INFO  Export 3283 highways were created
7 | 2013-04-30 15:29:23,847 INFO  Export 2470 crossroads were
      created
8 | 2013-04-30 15:29:23,853 INFO  ExpRoadMain Time of generate was:
      0h 0min 12s 422ms
```

V aktuální složce pak máme vytvořen soubor `plzen.xml`, který obsahuje dopravní síť města Plzeň spolu s vypočtenými křižovatkami.

B.4 GUI

V této kapitole je popsáno, jak pomocí grafického uživatelského rozhraní načteme vstupní soubory, z nichž vygenerujeme soubor výstupní a jak tento výstupní soubor zobrazíme jako mapu.

B.4.1 Spuštění

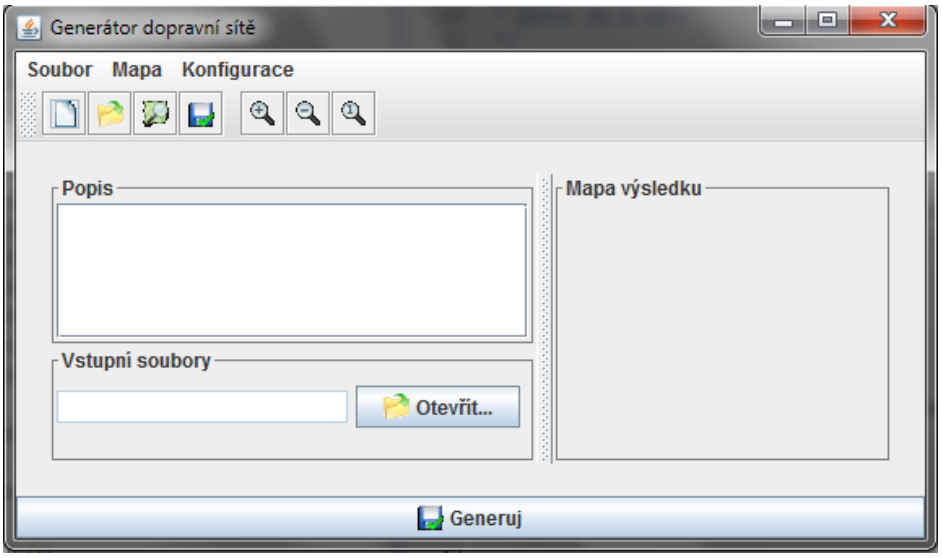
Aplikaci s grafickým uživatelským prostředím spustíme tak, že spustíme program `dopravni_sit.jar` bez argumentů. To můžeme udělat buď poklepáním, v případě, že máme asociovánu příponu JAR s javou, nebo pomocí příkazové řádky příkazem:
`java -jar <cesta k jar souboru>`

Po spuštění se objeví hlavní okno (viz obrázek B.3).

B.4.2 Načtení vstupních souborů a vygenerování výstupního

Vstupní soubory můžeme načíst několika různými způsoby, které spustí otevírací dialog v němž vybereme jeden nebo více vstupních souborů (více vstupních souborů se vybírá přes klávesu `CTRL`):

- kliknutím na tlačítko `otevřít`;



Obrázek B.3: Hlavní okno po spuštění programu.

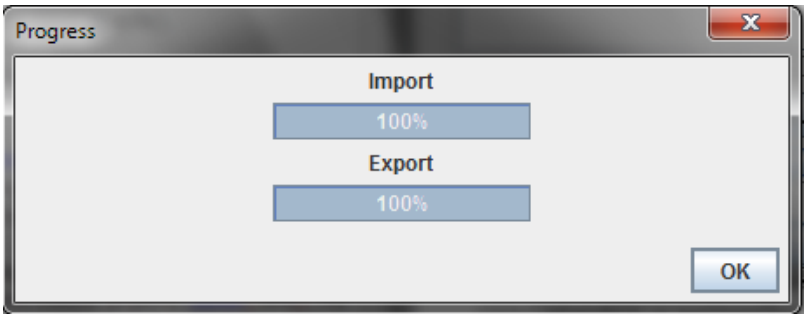
- kliknutím na druhou ikonu v tool baru;
- vybráním `Soubor - Otevřít - Otevřít OSM soubor` v menu.

Načtené soubory můžeme mazat (křížkem u každého souboru), měnit (tlačítko `otevřít` u načteného souboru) nebo přidávat (popsáno výše).

Máme-li vybrány všechny vstupní soubory, můžeme přejít k vygenerování výstupního:

- stiskem tlačítka `Generuj`;
- stiskem čtvrté ikony v tool baru;
- vybráním `Soubor - Generuj výstup` v menu.

Zobrazí se ukládací dialog, kde vybereme cestu a zadáme jméno výstupního souboru včetně přípony *.xml*. Po potvrzení se zobrazí dialog s progress bary pro import a export (viz obrázek B.4) a po dokončení operace se zpřístupní tlačítko `OK`. Výstupní soubor je nyní vygenerován a po potvrzení se zobrazí mapa.



Obrázek B.4: Dialog s progress bary.

B.4.3 Práce s mapou

Mapu můžeme načíst několika způsoby:

- po vygenerování výstupního souboru se automaticky zobrazí;
- stiskem třetí ikony v toolbaru;
- vybráním `Soubor - Otevřít - Otevřít XML mapu` v menu.

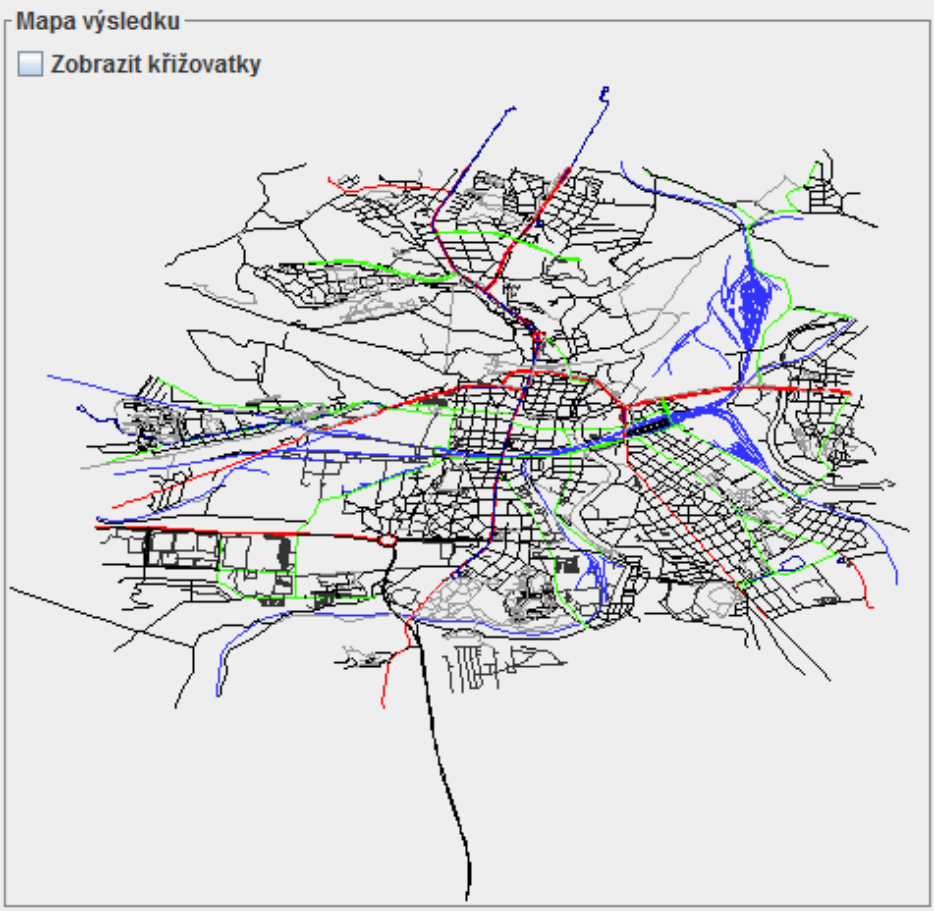
Po načtení se mapa zobrazí (viz obrázek B.5). Mapu můžeme dále zvětšovat, či zmenšovat (pokud již není vidět celá), pomocí tlačítek s lupou v tool baru, nebo vybráním možnosti v nabídce `Mapa` v menu.

Pomocí zaškrtávacího tlačítka `Zobrazit křižovatky`, můžeme zobrazit na mapě křižovatky (viz obrázek B.6).

B.4.4 Úprava konfiguračních souborů

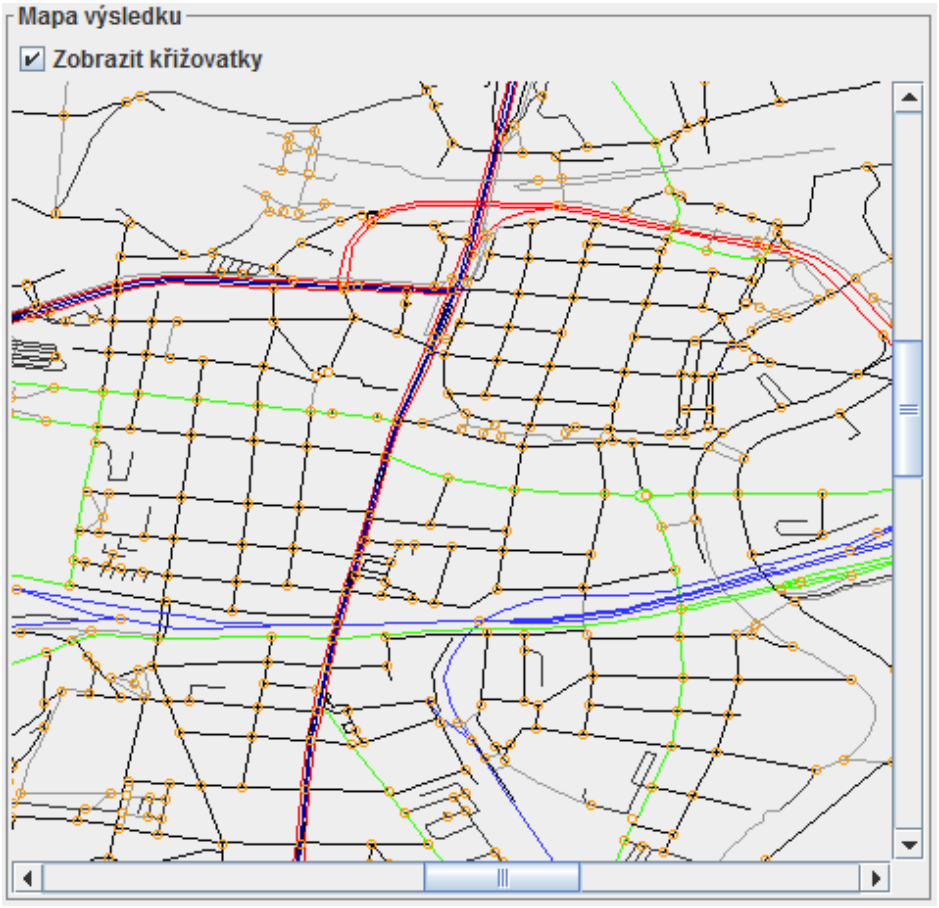
Přímo z GUI je možné měnit konfigurační soubor s barvami silnic pro vykreslování mapy (`colors.ini`), soubor se jmény elementů výstupního souboru (`outformat.ini`) a soubor s definicí konstant pro generování křižovatek (`crossroads.ini`).

Úpravu konfiguračních souborů provedeme tak, že v menu vybereme nabídku `Konfigurace` a zvolíme, co chceme konfigurovat. Zobrazí se dialog s možností změn jednotlivých proměnných. V případě formátu výstupu a konstant křižovatek je počet proměnných daný (viz obrázek B.7). V případě

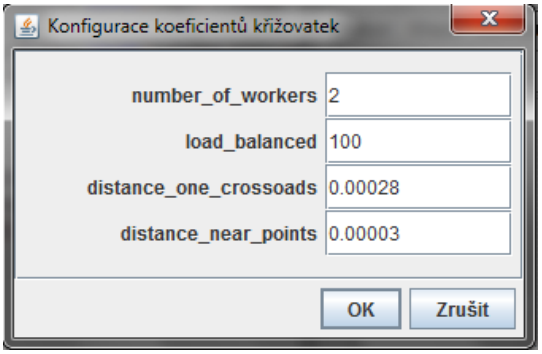


Obrázek B.5: Zobrazení mapy bez křižovatek.

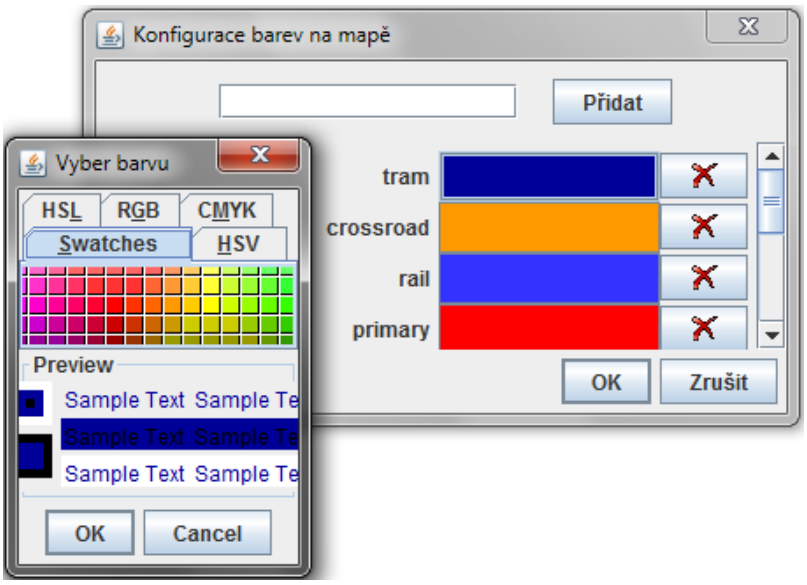
barev pro zobrazení mapy, jdou jednotlivé třídy silnic přidávat a mazat s tím, že každá nedefinovaná třída silnice bude mít černou barvu. U tohoto dialogu je také možné vybírat barvu přímo pomocí palety barev (viz obrázek B.8).



Obrázek B.6: Zobrazení mapy s křižovatkami.



Obrázek B.7: Dialog pro úpravu konstant pro vytváření křižovatek.



Obrázek B.8: Dialog pro úpravu barev tříd silnic na mapě.

C Výpis konfiguračních souborů

C.1 localization_cs.properties

```
1  titleMainFrame = Generátor dopravní sítě
2  descriptionMainFrame = Popis
3  mapMainFrame = Mapa výsledku
4  inputFilesMainFrame = Vstupní soubory
5  btnGenerateMainFrame = Generuj
6  btnOpenMainFrame = Otevřít...
7
8  file = Soubor
9  new = Nový
10 open = Otevřít
11 openOSM = Otevřít OSM soubor
12 openMap = Otevřít XML mapu
13 generateOut = Generuj výstup
14 exit = Konec
15 map = Mapa
16 zoomIn = Zvětši mapu
17 zoomOut = Zmenši mapu
18 fullMap = Zobraz celou mapu
19 configuration = Konfigurace
20 outputConf = Konfigurace výstupního souboru
21 mapConf = Konfigurace barev na mapě
22 crossroadsConf = Konfigurace koeficientů křižovatek
23
24 import = Import
25 export = Export
26 progressDialogCaption = Progress
27
28 outputPropertiesCaption = Konfigurace výstupního souboru
29 colorPropertiesCaption = Konfigurace barev na mapě
30 crossroadsPropertiesCaption = Konfigurace koeficientů křižovatek
31
32 btnAdd = Přidat
33 chooseColor = Vyber barvu
34
35 showCrossRoads = Zobrazit křižovatky
36
37 noInputFileEx = Nejsou zadány žádné vstupní soubory
38 noOutputFileEx = Není zadán výstupní soubor
39 noParameters = Nejsou zadány žádné parametry
40 notFindNodeEx = Nenalezen node s ID=
41 unkrownParameter = Neznámý parametr
42 mustBeNumber = musí být číslo!
```

```
43 |
44 | exportSucces = Export proběhl úspěšně.
45 | existSource = byl přidán již dříve.
46 |
47 | yes = Ano
48 | no = Ne
49 |
50 | ok = OK
51 | cancel = Zrušit
```

C.2 localization_en.properties

```
1  | titleMainFrame = Generator of traffic network
2  | descriptionMainFrame = Description
3  | mapMainFrame = Map result
4  | inputFilesMainFrame = Input files
5  | btnGenerateMainFrame = Generate
6  | btnOpenMainFrame = Open...
7  |
8  | file = File
9  | new = New
10 | open = Open
11 | openOSM = Open OSM File
12 | openMap = Open XML Map
13 | generateOut = Generate Output
14 | exit = Exit
15 | map = Map
16 | zoomIn = Zoom In Map
17 | zoomOut = Zoom Out Map
18 | fullMap = View Full Map
19 | configuration = Configuration
20 | outputConf = Configure Output
21 | mapConf = Configure map colors
22 | crossroadsConf = Configure coefficients of crossroads
23 |
24 | import = Import
25 | export = Export
26 | progressDialogCaption = Progress
27 |
28 | outputPropertiesCaption = Configuration of output file
29 | colorPropertiesCaption = Configuration color of map
30 | crossroadsPropertiesCaption = Configure coefficients of
       crossroads
31 |
32 | btnAdd = Add
33 | chooseColor = Choose color
34 |
```

```
35 showCrossRoads = Show crossroads
36
37 noInputFileEx = Must fill input files
38 noOutputFileEx = Must fill output file
39 noParameters = Must fill parameters
40 notFindNodeEx = Not found node with ID=
41 unkrownParameter = Unknown parameter
42 mustBeNumber = must be a number!
43
44 exportSucces = Export was successful.
45 existSource = is already exist.
46
47 yes = Yes
48 no = No
49
50 ok = OK
51 cancel = Cancel
```

C.3 colors.ini

```
1 tram=-16777063
2 rail=-13421569
3 crossroad=-26368
4 primary=-65536
5 footway=-6710887
6 service=-13421773
7 steps=-3355444
8 residental=-16777216
9 secondary=-13369600
```

C.4 outformat.ini

```
1 map=map
2 description=description
3 bounds=bounds
4 road=road
5 crossroad=crossroad
6 position=position
7 roads=roads
8 ref=ref
9 nodes=nodes
10 node=node
11 latitude=latitude
12 longitude=longitude
13 id=id
14 name=name
```

```
15 roadNumber=roadNumber
16 roadInternationalNumber=roadInternationalNumber
17 roadClass=roadClass
18 lanes=lanes
19 layer=layer
20 maxspeed=maxspeed
```

C.5 crossroads.ini

```
1 number_of_workers=2
2 load_balanced=100
3 distance_one_crossoads=0.00028
4 distance_near_points=0.00003
```

C.6 log4j.properties

```
1  # Set root category priority to WARN and its only appender to A1
       .
2  log4j.debug=true
3  log4j.rootCategory=INFO, CONSOLE, FILE
4
5  #### CONSOLE ####
6  # CONSOLE is set to be a ConsoleAppender.
7  log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
8  # A1 uses PatternLayout.
9  log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
10 log4j.appender.CONSOLE.layout.ConversionPattern=%d{ISO8601} %-5p
       %c{1} %m%n
11
12 #### FILE ####
13 #### Second appender writes to a file
14 log4j.appender.FILE=org.apache.log4j.RollingFileAppender
15 log4j.appender.FILE.File=dopravni_sit.log
16 # Control the maximum log file size
17 log4j.appender.FILE.MaxFileSize=25MB
18 # Archive log files (five backup files here)
19 log4j.appender.FILE.MaxBackupIndex=5
20 log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
21 log4j.appender.FILE.layout.ConversionPattern=%d{ISO8601} %-5p %c
       {1} %m%n
```