UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Monitorovací systém polohy mobilních zařízení s využitím GPS a GSM technologií

Bc. Jan Keller

Diplomová práce
2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan KELLER**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Monitorovací systém polohy mobilních zařízení s využitím GPS a GSM technologií**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je vytvoření komplexního systému pro sledování polohy mobilních zařízení pomocí GPS lokátoru. Systém se bude skládat ze tří základních částí: - GPS lokátoru, který data získaná z družic posílá na server prostřednictvím datové sítě GSM (GPRS), - serveru, který data shromažďuje a uchovává - a klientské aplikaci, která je schopna data ze serveru získat a dále je zpracovat a zobrazit. Teoretická část práce bude obsahovat přehled problematiky sledování polohy a srovnání s již exitujícími systémy. V praktické části bude provedena analýza a návrh celého systému pomocí jazyka UML a následně implementace SW v jazyce JAVA pro GPS lokátor, implementace serverové části včetně příslušné databáze pro uchování dat a implementace klientské aplikace v jazyce C pro zobrazení mapových podkladů a pro zpracování a vizualizaci údajů, získaných ze serveru.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

1. RAPANT, Petr. Družicové polohové systémy. 1. vyd. Ostrava : VŠB - Technická univerzita Ostrava, 2002. 200 s. Dostupný z WWW: . ISBN 80-248-0124-8. 2. KABELOVÁ, Alena, DOSTÁLEK, Libor. Velký průvodce protokoly TCP/IP a systémem DNS. 3. aktualiz. vyd. Praha : Computer Press, 2002. 552 s. ISBN 80-7226-675-6. 3. NAGEL, Christian, et al. C 2008 : Programujeme profesionálně. 1. vyd. Brno : Computer Press, 2009. 2 sv. (1126, 772 s.) . ISBN 978-80-251-2401-7. 4. HUML, Milan. Mapování a kartografie. 1. vyd. Praha : Ediční středisko ČVUT, 2001. 211 s. ISBN 80-01-02383-4.

Vedoucí diplomové práce: **Ing. Petr Veselý**
Katedra informatiky v dopravě

Datum zadání diplomové práce: **30. října 2009**

Termín odevzdání diplomové práce: **21. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 10. listopadu 2009

Prohlášení autora

Tuto práci jsem 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 moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 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.

V Pardubicích dne 18. srpna 2010

Jan Keller

Anotace

Práce pojednává o vytvoření komplexního systému pro sledování polohy mobilních zařízení s hlavním zaměřením na vytvoření univerzální a rozšiřitelné softwarové aplikace pro optimalizované zobrazování mapových podkladů, polohy objektů a dalších souvisejících informací. Kromě této klientské části se dále práce zaměřuje na vytvoření serveru pro sběr a distribuci dat a na výběr a zprovoznění vhodného hardwarového zařízení pro snímání a odesílaní polohy.

Klíčová slova

GPS, APRS, sledování polohy, mapová projekce, OpenStreetMap, vizualizace map

Title

Monitoring system for tracking of mobile devices using GPS and GSM technology.

Annotation

The work is about developing complex system for position tracking of mobile devices with the main focus at creating universal and extensible software application for optimized rendering of maps, position of objects and other related information. Besides the client application the work is about developing the server for collecting a distributing data and then selecting and connecting appropriate hardware solutions for position tracking.

Keywords

GPS, APRS, position tracking, map projection, OpenStreetMap, map visualization

Obsah

Seznam zkratek

API	Application Programming Interface
APRS	Automatic Packet Reporting System
CLR	Common Language Runtime
DLL	Dynamic-link library
GDI	Graphics Device Interface
GNSS	Global Navigation Satellite System
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
IPC	Inter-process communication
MEF	Managed Extensibility Framework
NMEA	National Marine Electronics Association
ORM	Object-relational mapping
OSM	OpenStreetMap
PDA	Personal Digital Assistant
PNG	Portable Network Graphics
RFID	Radio Frequency Identification
TCP	Transmission Control Protocol
TNC	Teminal Node Controller
UTC	Coordinated Universal Time
UTF	Unicode Transformation Format
VHF	Very high frequency
WCF	Windows Communication Foundation
WGS84	World Geodetic System 1984
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

Seznam obrázků

Seznam tabulek

1 Úvod

Tato diplomová práce se zabývá návrhem a implementací systému pro monitorování polohy mobilních objektů v terénu. Jádrem projektu je vytvoření softwarové platformy pro renderování map. Hlavní směr, kterým se práce vydává, je zaměřen na počítačovou 2D grafiku a na možnosti renderování a práce s mapovými podklady. Vedlejším směrem je návrh a výběr vhodného hardwarového vybavení umožňujícího zjišťování a odesílaní polohy mobilních objektů v terénu. Cílem práce není vytvořit plně odladěnou aplikaci připravenou k ostrému nasazení ale provést implementaci základního systému, na němž budou demonstrovány postupy, jimiž se dosáhne požadovaných cílů. S použitím v praxi se však počítá a vývoj aplikace bude pokračovat i po ukončení diplomové práce.

První část se zabývá teorií sledování polohy. Seznámí čtenáře s několika hlavními technologiemi používanými v tomto odvětví a také v této práci. Dále je vysvětlen základní princip vizualizace map a mapová projekce.

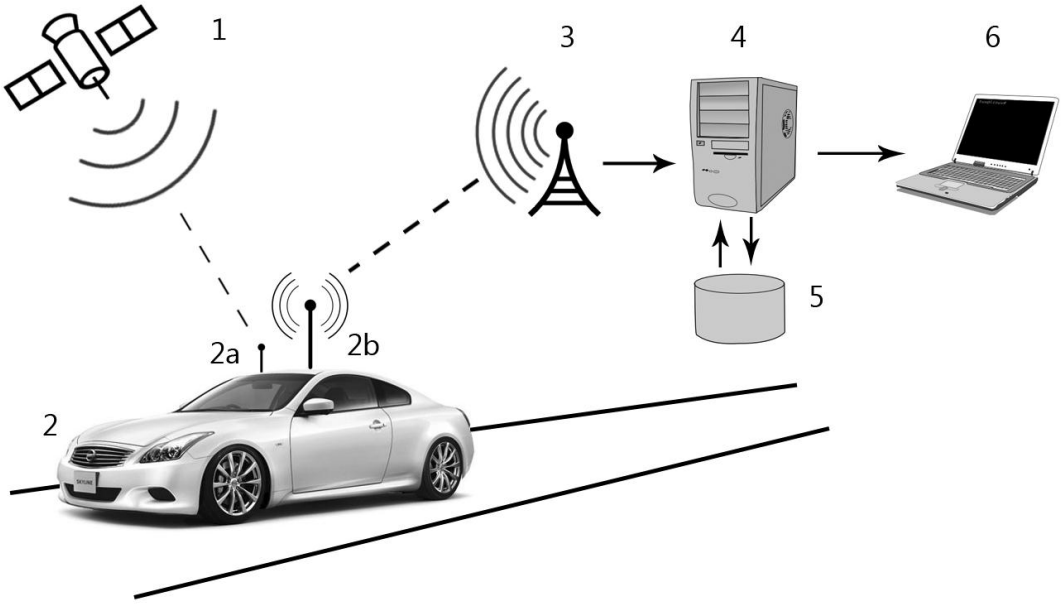
Praktická část se zabývá analýzou a implementací systému, jež byl pracovně nazván Lishka. V analytické části jsou specifikovány požadavky na systém, prezentovány diagramy a provedena analýza pro výběr softwarových řešení. V druhém oddíle je detailně popsána samotná implementace softwarových aplikací klienta a serveru, s detailním popisem jednotlivých jejich částí. Dále jsou rozepsány nejdůležitější algoritmy použité při implementaci. Poslední oddíl praktické části je věnován hardwaru pro sběr a přenos informací o poloze. Zde je popsáno konkrétní hardwarové řešení a způsobem, jakým je integrováno do systému.

V závěru práce jsou shrnuty dosažené výsledky a získané zkušenosti. Je nastíněno, jakým směrem bude směřovat další vývoj a také jaké jsou možnosti budoucího využití.

Při analýze a implementaci docházelo ke změnám oproti původnímu zadání. Tyto změny jsou v práci popsány a zdůvodněny. Většina těchto změn vychází z toho, že podrobná analýza celého systému a jeho nasazení byla provedena až po uzavření zadání.

2 Sledování polohy

Pod pojmem sledování polohy lze chápat určení polohy nějakého objektu v prostoru a předání této informace jinému systému pro zpracování. Objekty, jejichž poloha se sleduje, mohou být osoby, automobily, vlaky, kontejnery nebo letadla. Systémy, které přijímají informace o poloze objektů, pak mohou být databáze, statistické programy či mapové aplikace. Jednou z běžných aplikací je sledování polohy vozidel. Typický scénář je zobrazen na Obrázku 1. Satelity globálního satelitního navigačního systému (GNSS) vysílají signál směrem k Zemi (1). Automobil (2), vybavený přijímačem tohoto signálu (2a), tento příjme a zpracuje tak, aby jej mohl přeposlat dále radiovému vysílači (2b). Radiový vysílač pravidelně odesílá informace pozemnímu přijímači (3), který je napojen na server (4). Přijatá data se na serveru zpracují a uloží do databáze (5). K serveru se mohou připojovat klienti (6) a získávat nasbíraná data k dalšímu zpracování. Celé schéma můžeme rozdělit na subsystém určování polohy (1, 2a) a na subsystém předávání informací a zpracování (2b, 3, 4, 5, 6).



Obrázek 1: Schématické zobrazení systému sledování polohy

Zdroj: autor

Sledování polohy lze rozdělit podle toho, zdali sledujeme objekty lokálně nebo globálně. Lokální sledování se týká většinou hal či rozsáhlejších, ale stále lokálních oblastí, jakými

mohou být nemocnice, průmyslové podniky nebo celá města. V globálním sledování pak nejsou omezení týkající se místa téměř žádná a lze provádět sledování objektů kdekoliv.

Lokální sledování se také někdy nazývá RTLS (Real Time Location System). Pod tímto pojmem si lze představit různé technologie, které se liší především použitelnou maximální vzdáleností mezi vysílačem a přijímačem:

- RFID[1] (desítky metrů),
- Wi-Fi (stovky metrů),
- Bluetooth (až 100 m),
- Ultra-Wideband[2] (desítky metrů).

Všechny tyto technologie slouží nejen jako prostředek pro bezdrátový přenos dat, ale i pro samotný výpočet polohy. Metody výpočtu polohy jsou popsány dále.

Pro globální sledování je nutné využít více technologií najednou. Pro zjištění polohy se používá GNSS (GPS, Glonass…, viz kapitola 2.1). Pro přenos informací o poloze ke zpracování je nutné použít některou z bezdrátových technologií s globálním pokrytím. Takovou technologií může být např. GPRS či UMTS (kapitola 2.4).

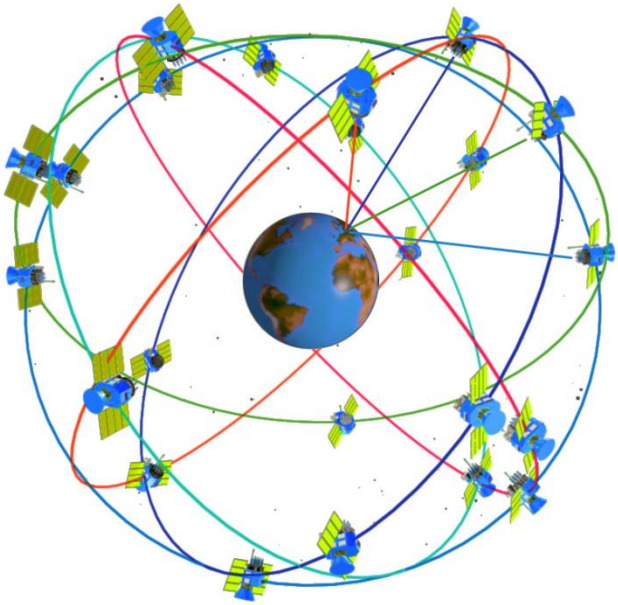
Pro výpočet polohy mobilních objektu existuje několik různých metod. Mnoho metod využívá závislosti vzdálenosti na čase potřebném k překonání vzdálenosti mezi přijímačem a vysílačem. Příkladem takové metody je ToA (Time of Arrival). Její nevýhodou je nutnost synchronizace času vysílače s přijímačem. Sebemenší odchylka má výrazný vliv na přesnost výpočtu polohy. Tuto nevýhodu řeší metoda TDoA (Time Difference of Arrival). Zde se nevychází z absolutních hodnot časů ale z rozdílů mezi sousedními vysílači. Tato metoda je používána u technologie GPS. Metody pracující s časovými rozdíly jsou obecně vhodné pro venkovní prostory s minimem odrazů a s přímou viditelností z vysílače na přijímač. Příkladem metody, která není závislá na čase je RSS (Received Signal Strength). Princip vychází z měření síly signálu radiově viditelných přístupových bodů. Metoda využívá závislosti síly signálu na vzdálenosti od vysílače. Tato metoda se používá například pro zjištění polohy v GSM síti. [20]

[1] RFID (Radio Frequency Identification) je malý čip, který může být buď pasivní (nepotřebuje zdroj energie) nebo aktivní (ke svému provozu vyžaduje napájení), který má větší dosah.
[2] Ultra-Wideband je radiová technologie pro přenos informací pracující na velmi nízkých energetických úrovních.

2.1 GPS

Global Positioning System je globální družicový navigační systém vlastněný vládou Spojených Států Amerických a sloužící uživatelům k poskytování informací o pozici, navigaci a čase. GPS poskytuje přesné informace o poloze a času pro neomezený počet lidí za každého počasí, ve dne i v noci, kdekoliv na světě v exteriéru. Celý systém se skládá ze tří částí: vesmírný segment, kontrolní segment a uživatelský segment. O vývoj a údržbu vesmírného a kontrolního segmentu se stará U. S. Air Force. [6]

Princip, na kterém GPS funguje, je založen na tom, že všechny družice vysílají informace o poloze ve stejný čas. Signál ze satelitů se šíří rychlostí 300 000 km/s. Na Zemi umístěný přijímač poté může vypočítat svoji pozici na základě toho, s jakým zpožděním dorazí signál z jednotlivých satelitů. K určení pozice ve dvourozměrném prostoru je potřeba přijímat signál z minimálně třech satelitů. Pro určení nadmořské výšky jsou pak potřeba minimálně čtyři. S vyšším počtem přijímaných signálů pak stoupá přesnost, s jakou lze vypočítat polohu uživatele.



Obrázek 2: Rozložení GPS satelitů okolo Země

Zdroj: http://www.aero.org/publications/crosslink/summer2002/03.html

Vesmírný segment se skládá celkem z 24 satelitů (Obrázek 2), z nichž 3 slouží jako záložní. Satelity vysílají jednosměrný signál, který nese informace o pozici a času satelitu.

Každý satelit je vybaven manévrovacími motory, kterými lze korigovat jeho dráhu podle příkazů z kontrolních stanovišť na povrchu Země.

Kontrolní segment se skládá z monitorovacích a řídících stanic rozmístěných kolem celého světa, které se starají o to, aby satelity obíhaly po správné dráze, a upravují satelitní atomové hodiny. Tyto stanice sledují GPS satelity, stahují navigační data a udržují správné rozložení satelitů tak, aby bylo zaručeno co nejlepší konstantní pokrytí signálem.

Uživatelský segment, to jsou GPS přijímače, které mohou přijímat signál ze satelitů a tyto údaje posléze využít k výpočtu polohy a času uživatele. [5]

GPS není jediným globálním družicovým polohovým systémem. V současné době funguje také systém Glonass, který je provozován ruskou armádou. Síť se skládá z 24 satelitů, které jsou vypouštěny od roku 1982. Další alternativou k GPS je navigační systém Galileo, jenž se zatím nachází ve fázi plánování a testování. Systém má být tvořen 30 satelity a jeho nejbližší spuštění je plánováno na rok 2014. Čína také pracuje na vybudování svého vlastního navigačního systému. Jeho název je Compass a měl by být plně dokončen do roku 2020, s celkovým počtem 35 satelitů na oběžné dráze. [19]

2.2 Protokol NMEA

NMEA 0183 je standardní protokol používaný a podporovaný většinou zařízení, pracujících s GPS informacemi, k přenosu dat. Komunikace probíhá zasíláním jednotlivých řádků dat nazývaných věty, které obsahují specifická data spadající do určité kategorie a jsou zcela nezávislé na ostatních větách. Všechny věty jsou kódovány v ASCII. Každá věta začíná symbolem dolaru ($) a ukončena je znaky <CR><LF>. Jednotlivé položky jsou ve větě odděleny čárkou. Věta může mít nejvýše 80 znaků. Konkrétní význam jednotlivých položek je jasně definován pro každý typ věty[3].

Protokol NMEA byl vyvinut Mezinárodní námořní elektronickou asociací jako protokol pro různé námořní přístroje: např. sonary, echolokátory, gyrokompasy a přijímače GPS. Proto jsou věty určené pro danou oblast vždy specificky pojmenovány. Věty, které používají GPS přijímače, začínají písmeny GP ve svém názvu. Věta, která obsahuje

[3] Popis jednotlivých vět NMEA: http://www.gpsinformation.org/dale/nmea.htm.

nejnutnější data k určení polohy, se nazývá GPRMC. Z ní lze získat datum a čas, zeměpisnou šířku a délku, vodorovnou rychlost, kurz pohybu a magnetickou deklinaci.

Příklad věty GPRMC včetně vysvětlení jednotlivých prvků v Tabulce 1:

```
$GPRMC,055626.658,A,5002.1574,N,01429.3056,E,0.22,335.14,100610,,*02.
```

Tabulka 1: Význam jednotlivých prvků věty GPRMC z protokolu NMEA

#	Formát	Příklad	Vysvětlení
1	`hhmmss.sss`	055626.658	Čas (UTC)
2	`c`	A	Status (A = OK)
3	`ddmm.mmm`	5002.1574	Zeměpisná šířka
4	`c`	N	Sever / jih
5	`dddmm.mmm`	01429.3056	Zeměpisná délka
6	`c`	E	Východ / západ
7	`d.d`	0.22	Vodorovná rychlost v uzlech
8	`d.d`	335.14	Kurz pohybu ve stupních
9	`ddmmyy`	100610	Datum
10	`d.d`	N/A	Magnetická deklinace ve stupních
11	`*xx`	02	Kontrolní součet

Protokol také definuje nastavení pro datovou vrstvu. Rychlost, kterou může komunikovat je 4800 b/s, počet datových bitů je 8, přičemž sedmý bit (MSB) je vždy nulový. Počet stop bitů je jeden, parita není žádná. [5]

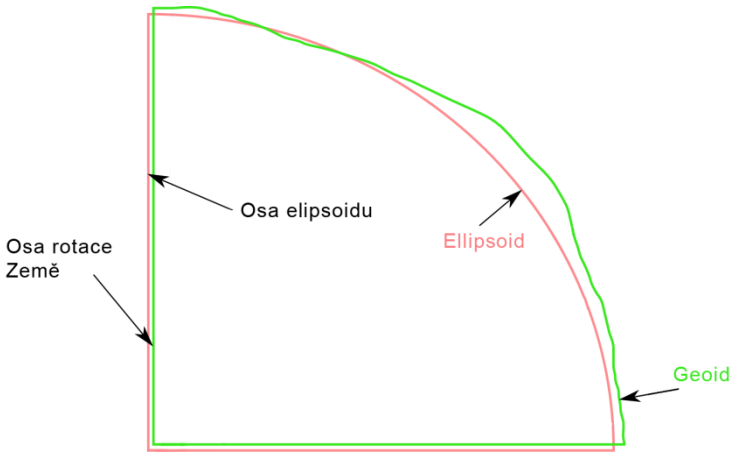
2.3 WGS-84

World Geodetic System 1984 je světový geodetický referenční systém, který roku 1984 vydalo Ministerstvo obrany USA. Definuje souřadný systém a referenční elipsoid[4]. Tento systém je v současné době používán jako standard pro satelitní navigační přístroje GPS.

WGS-84 používá zeměpisné souřadnice a jednotlivé osy jsou definovány zeměpisnou délkou, šířkou a výškou. Zápis zeměpisné délky a šířky může mít různý formát: pouze ve

[4] Elipsoid je matematicky definované těleso, je rotační (rotuje kolem menší poloosy – S-J). Bývá definován tak, aby jeho střed ležel ve středu Země a aby se co nejlépe přimykal geoidu.

stupních (např. *50,0289; 14,4879*), ve stupních a minutách (např. *50° 2,523'; 14° 12,54'*) nebo ve stupních, minutách a sekundách (např. *50° 2' 49,853"; 14° 27' 9,966"*). V zápisu je třeba vždy uvést, zda se jedná o severní či jižní šířku a o východní či západní délku. To se lze provést dvěma způsoby. První z nich je uvedení velkého písmene za jednotlivé souřadnice, které představuje první písmeno anglického názvu světové strany (N, S, E, W). Druhou možností je uvedení znaménka – (mínus) před souřadnice ležící na západní nebo jižní polokouli. Zápis výšky je uveden v metrech. Nepředstavuje však skutečnou nadmořskou výšku, ale elipsoidickou výšku (vzdálenost od referenčního elipsoidu). Na Obrázku 3 je znázorněn rozdíl mezi geoidem[5] a referenčním elipsoidem (nejedná se o konkrétní elipsoid).



Obrázek 3: Poloha referenčního elipsoidu vůči geoidu

Zdroj: autor

V České republice byl donedávna standardem pro vojenské a turistické mapy souřadný systém S-42. Tento souřadný systém používá jiný referenční elipsoid, a tudíž není kompatibilní s WGS84. Souřadný systém S-42 je dnes ve většině případů nahrazen systémem WGS84. [7]

2.4 GSM

GSM je nejrozšířenější standart pro mobilní telefony na světě. Tuto síť využívá vice než 3,8 miliard lidí ve více než 200 zemích světa. Oproti předchozím technologiím je GSM první, která používá plně digitální přenos. Díky tomu je považována za takzvanou síť

[5] Geoid je fyzikální model povrchu Země při střední hladině světových oceánů.

druhé generace (2G). GSM je buňková síť, což znamená, že mobilní telefony se do ní připojují vyhledáním buňky ve svém nebližším okolí. Tyto buňky mohou mít různé velikosti a jsou určeny pro pokrytí různě velkých území. Můžeme je nalézt na samostatných stožárech, na střechách domů nebo ve stanicích metra. Síť GSM pracuje ve čtyřech různých frekvenčních pásmech. Většina GSM sítí pracuje na frekvencích 900 MHz nebo 1800 MHz, ale některé země v Americe (např. Kanada nebo USA) využívají také frekvenčních pásem 850 MHz a 1900 MHz a to kvůli tomu, že pásma 900 MHz a 1800 MHz jsou již zahlcená. [9]

GSM používá různé hlasové kodeky pro kompresy zvuku na frekvenci 3,1 kHz. Takto komprimovaný zvuk je poté přenášen rychlostí mezi 5,6 kb/s a 13 kb/s. Takové rychlosti stačí pro hlasovou komunikaci, ne však pro datovou. Proto bylo zavedeno několik dalších systému založených na síti GSM:

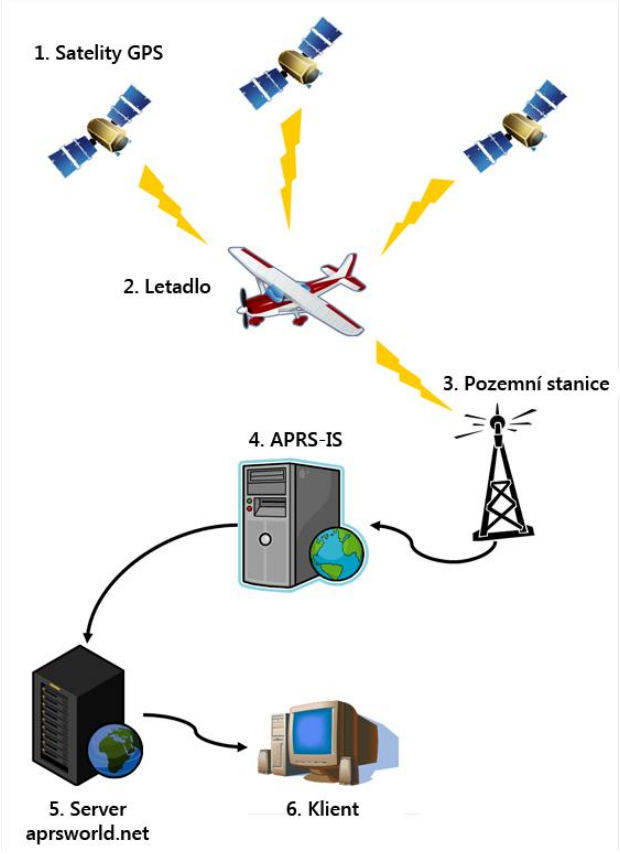
- **GPRS** – Jedná se o datovou službu dostupnou všem uživatelům 2G GSM sítě. Tím, že je dostupná všude tam, kde je pokrytí signálem GSM, se jedná o datovou síť s nejširším pokrytím. Je označována jako technologie 2,5G. Je to paketově orientovaná datová služba a její přenosová rychlost se pohybuje průměrně 56–114 kb/s. Pro přenos dat se používají TDMA kanály a výsledná přenosová rychlost zaleží na počtu právě používaných kanálů. [9]
- **EDGE** – Je dalším stupněm ve vývoji datových služeb. Oproti GPRS nabízí přenosovou rychlost až 236,8 kb/s. EDGE ke svému provozu potřebuje vlastní síť a mobilní telefon s jeho podporou. Pokrytí touto technologií není tak široké jako u GPRS. EDGE je označována jako síť generace 2,75. [9]
- **UMTS** – Je první datová síť 3. generace. Byla koncipována jako nástupce GSM. Pracuje ve frekvenčním pásmu 2100 MHz. Hlavní výhodou pro uživatele je opět navýšení rychlosti a to až na teoretických 42 Mb/s. V současné době se však lze v praxi setkat s rychlostí do 7,2 Mb/s. [9]

2.5 APRS

APRS znamená Automatic Packet Reporting systém. Jedná se v první řadě o komunikační protokol definující, jak data (včetně stanic, poloh mapových objektů, informací o počasí, textových zpráv a telemetrie) mohou komunikovat prostřednictvím paketových radiových

stanic. APRS je zároveň síť, která přenáší tyto informace. Ve Spojených Státech Amerických, Evropě a několika dalších zemích existují sítě digitálních opakovačů, takzvaných digipeaterů, většinou běžících na vyhrazené národní frekvenci. Tyto sítě poskytující transportní infrastrukturu pro všechny uživatele, kteří se chtějí připojit. Národní sítě jsou otevřené a přístupné komukoliv s patřičným vybavením a to zdarma.

APRS protokol, hardware a software muže být používán také nezávisle na národních sítích. Místní nebo dočasné sítě mohou být zřízeny pro speciální události nebo pro naplnění potřeb nějaké organizace. Některé scénáře nasazení ani nevyžadují použití digipeaterů. Např. balóny, operující ve velkých výškách, často používají APRS k vysílání pozice a telemetrie na určené frekvenci přímo k operačnímu týmu.



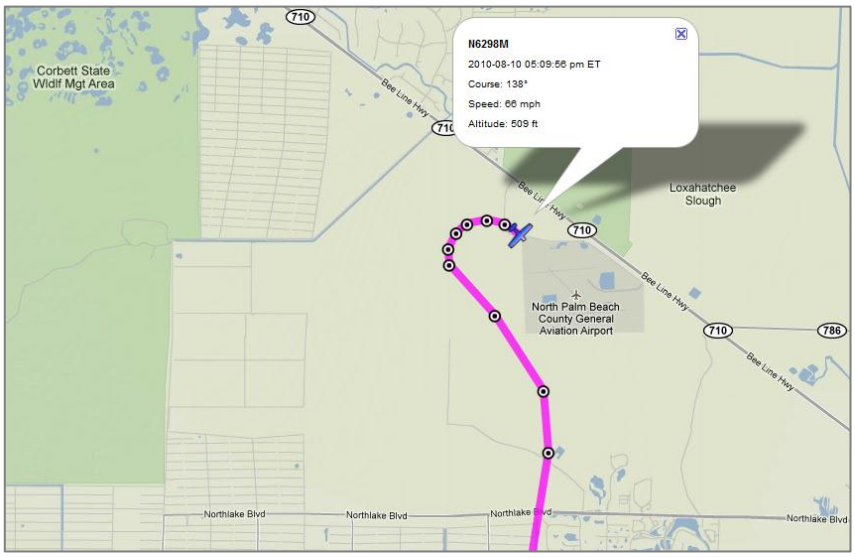
Obrázek 4: Schéma APRS-IS [11]

Stanice APRS mohou být stacionární (např. meteorologické stanice) nebo mobilní (automobily, osoby, letadla…). Použití mobilního APRS může mít několik různých forem. Nejjednodušší z nich je taková, kdy je přítomen tracker, který pouze vysílá, a GPS přijímač. Takové trackery nemají většinou žádnou schopnost přijímat informace kromě ověření, zdali je kanál volný před tím, než začnou vysílat. Dovolují tak mobilním

objektům, aby mohly být sledovány, ne však přijímat zprávy nebo informace o jiných objektech. Další možností je použití běžné vysílačky a TNC[6] ve spojení s notebookem nebo PDA. Takový systém nabízí plnou funkcionalitu APRS včetně zobrazování informací na mapě a posílaní zpráv. [10]

Jako rozšíření k APRS existuje APRS-IS (Automatic Packet Reporting System-Internet Service). APRS-IS je internetová služba, která propojuje různé radiové sítě APRS z celého světa. APRS-IS je budováno a udržováno dobrovolnými radioamatéry, za účelem poskytnutí možnosti využívat sítě APRS jako jeden velký celek. [11]

Na Obrázku 4 je zobrazeno schéma APRS-IS. Pro zjištění polohy letadla (2) se využívá technologie GPS (1). Zjištěná poloha se poté odešle nejbližší pozemní stanici (digipeateru) (3), která je napojena na lokální APRS-IS server (4). Aby mohla být data přístupná uživatelům na celém světě, musí se z lokálního serveru nahrát do globální databáze (5). Jednou z takových databází je server aprsworld.net. Odtud již mají k datům přístup klientské aplikace, které je mohou dále zpracovávat a distribuovat, např. prostřednictvím online map, jakou jsou Google Maps[7] či Bing Maps[8]. Jak taková data na mapě vypadají, je zobrazeno na Obrázku 5. Podobným způsobem lze zobrazovat i informace z klasické sítě APRS, která nemusí být propojena s APRS-IS.



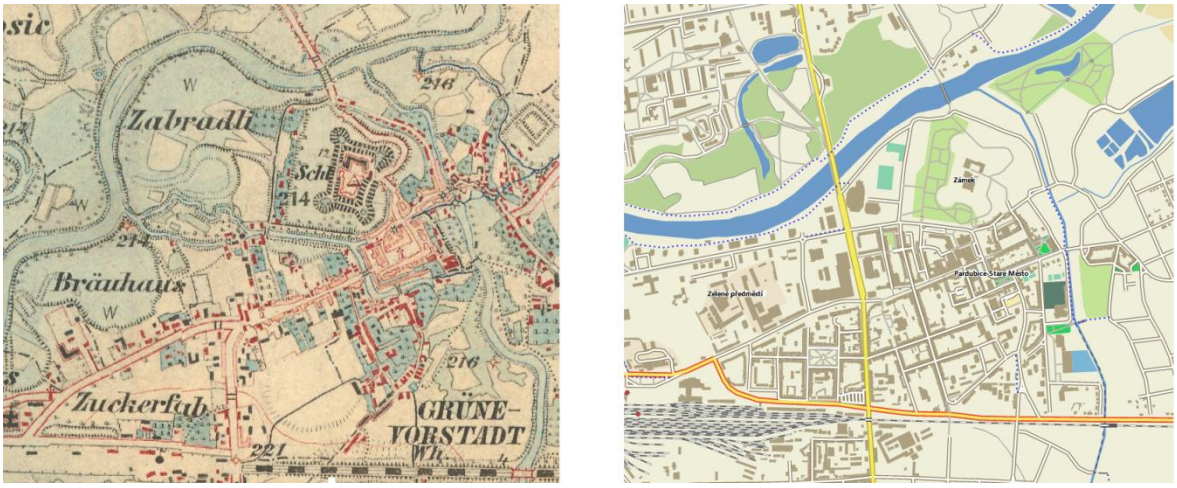
Obrázek 5: Informace z APRS-IS zobrazené na mapě [11]

[6] TNC (Terminal node controller) je zařízení používané radioamatéry k převodu digitálního signálu na analogový pro použití v radiových paketových sítích využívajících protokol AX.25.
[7] http://www.google.com/maps
[8] http://www.bing.com/maps

3 Vizualizace map

K tomu, abychom mohli mapy zobrazit takovým způsobem, aby co nejvěrněji odpovídaly realitě, slouží vizualizace. V dnešní době je většina mapových podkladů uchovávána ve vektorových formátech a proto je musíme před prezentací uživateli převést na odpovídající formu. Na začátku musíme mít k dispozici mapové podklady, jejichž kvalita ovlivňuje přesnost a detailnost výsledné mapy. Informace jsou v mapových podkladech ve formě zeměpisných souřadnic a je potřeba je převést do vhodného tvaru pro zobrazení na rovině. K tomu slouží mapové projekce. Důležité je také definovat styly, jakými se budou vykreslovat čáry a plochy, barvy a popisky.

Dříve se vizualizace map prováděly ručně. S nástupem počítačů se však stále více využívá automatického generování podle naměřených dat. S ručně kreslenými mapami se dnes setkáme jen velmi málo. Příkladem mohou být historické mapy nebo starší katastrální mapy. První mapy kreslené na počítači se začaly objevovat kolem roku 1990. Na Obrázku 6 je porovnán stejný výřez ručně kreslené historické mapy z roku 1877 a tentýž výřez na mapě aktuální, generované na počítači z vektorových dat.



Obrázek 6: Porovnání ručně kreslených a počítačově generovaných map

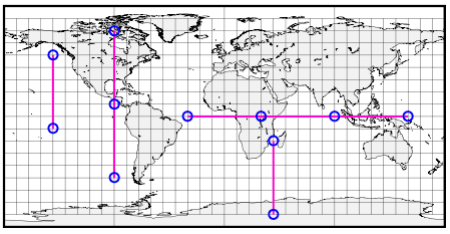
Zdroj: http://oldmaps.geolab.cz a autor

3.1 Mapová projekce

Kartografická mapová projekce je formální proces, kterým se převádějí mapové prvky mezi kulatým nebo elipsoidním povrchem a projekčním povrchem, ve většině případů tvořeným rovinnou plochou. Existuje velké množství projekcí, ale pouze několik z nich je v současnosti široce využíváno. [4]

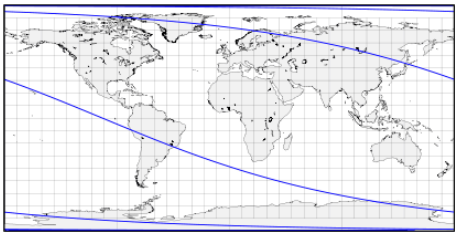
Mapové projekce jsou většinou vytvářeny tak, že se projekční rovina v jednom nebo více regionech dotýká koule představující mapovaný povrch. Z toho vyplývá, že čím blíže se nacházíme u těchto dotýkajících se regionů, tím menší je tvarové zkreslení. Neexistuje promítání, které by bez deformace zachovalo všechny údaje (délky, úhly, plochy). Každá projekce má charakteristický deformační vzor, proto existují takové, které kladou důraz na co nejlepší zachovaní vybraných údajů. Podle toho můžeme projekce dělit:

- **délkojevná** – nezkreslují vzdálenosti podél určitého systému čar,



- **plochojevná** – zachovávají poměry ploch, silně jsou však zkresleny úhly,

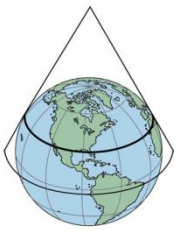
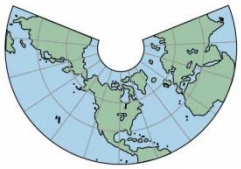
- **úhlojevná** – věrně zachycují úhly, ale silně zkreslují plochy,



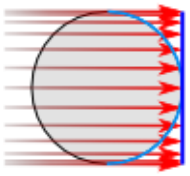
- **kompenzační** – kompromisní, s mírným zkreslením úhlů i ploch; patří sem i mnohá délkojevná zobrazení. [3]

Dále můžeme projekce rozlišovat podle tělesa, na které zobrazujeme mapu. Toto těleso se poté většinou rozvine do roviny a tím vznikne mapa. Existují tyto základní typy projekcí:

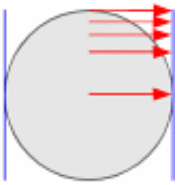
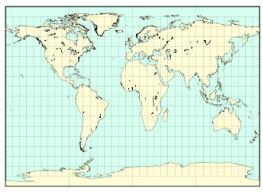
- **kuželová** – promítaní na kužel a jeho následné rozvinutí do roviny,

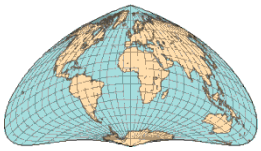
- **azimutální** – promítaní přímo na rovinu,

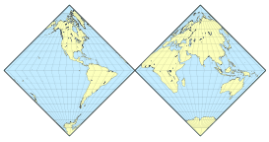



- **válcová** – promítaní na plášť válce, který se poté rozvine do roviny,

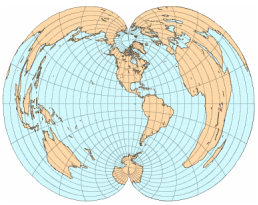



- **pseudozobrazení** – odvozená z výše uvedených,





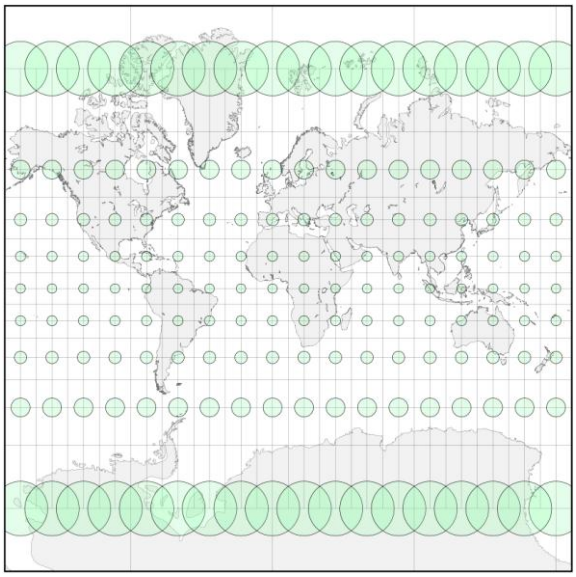
- **polykónická** – pro promítání je použita soustava kuželů. [3]



3.1.1 Mercatorova projekce

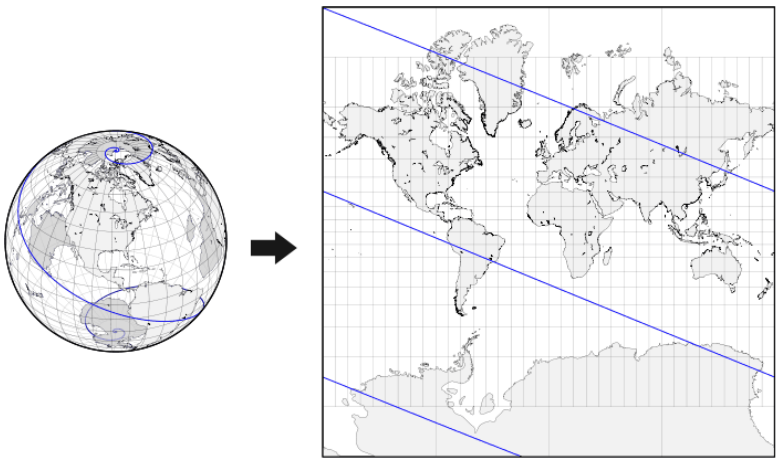
Mercatorova projekce je válcové mapové zobrazení, které vymyslel vlámský geograf Gerardus Mercator již v roce 1569. Jelikož se jedná o válcovou projekci, je výsledná mapa obdélníkové tvaru. Mercatorova projekce je dnes hojně používaná pro online mapy (např. Google Maps, Bing Maps nebo OpenStreetMap) a také u map určených pro satelitní navigace. Je tomu hlavně proto, že lineární měřítko je konstantní v každém směru okolo

jakéhokoliv bodu mapy a díky tomu jsou zachovány úhly a tvary malých objektů. Naproti tomu se velikosti a tvary velkých objektů deformuje s rostoucí vzdáleností od rovníku k pólům, kde rostou do nekonečna. Tato zkreslení jsou graficky znázorněna na Obrázku 7 (kružnice zobrazené na obrázku by měli mít při ideálním zobrazení stejný průměr jako kružnice kolem rovníku). Dobrých výsledků zobrazení tedy dosahuje kolem rovníku a u malých objektů s velkým přiblížením i v ostatních částech světa.



Obrázek 7: Mercatorova projekce s vyznačenými deformacemi [3]

Dalším důvodem, proč je Mercatorova projekce tolik používána je schopnost zobrazovat čáry konstantního kurzu, tzv. loxodromy[9], jako rovné segmenty, což je pro navigaci v určeném směru nezbytné (Obrázek 8). [3]



Obrázek 8: Ukázka loxodromy na Mercatorově projekci [3]

[9] Loxodroma je křivka na glóbu, která protíná všechny poledníky pod stále stejným úhlem (azimutem).

3.2 Mapové podklady

Základem pro vizualizaci map jsou mapové podklady. Mapové podklady jsou data v takové formě, která nám dovolí je graficky prezentovat a pracovat s nimi. Formáty, ve kterých mohou být tyto podklady distribuovány, jsou rastrové nebo vektorové. Rastrové formáty jsou ve většině případů paměťově náročné a práce s nimi je velice omezená. Vektorové formáty nám kromě menších nároků na paměťový prostor dávají také mnohem širší možnosti použití. Vektorová data můžeme vykreslovat a manipulovat s nimi ve 2D i 3D prostoru a zcela přizpůsobit výsledný obraz našim potřebám. Ne vždy je však možnost výběru formátu. V případě, že potřebuje pracovat s ortofoto mapami [10], či satelitními mapami[11], nelze jinak, než použít rastrový formát.

Jedním z problémů při použití mapových podkladů v aplikacích třetích stran jsou licenční omezení, která většinou jakékoliv takové využití zakazují. Existuje jen málo podkladů, které jsou opravdu zdarma. Jejich kvalita není ovšem vždy ta nejlepší. Mezi kvalitní a zdarma dostupné mapové podklady patří např. projekt OpenStreetMap [8].

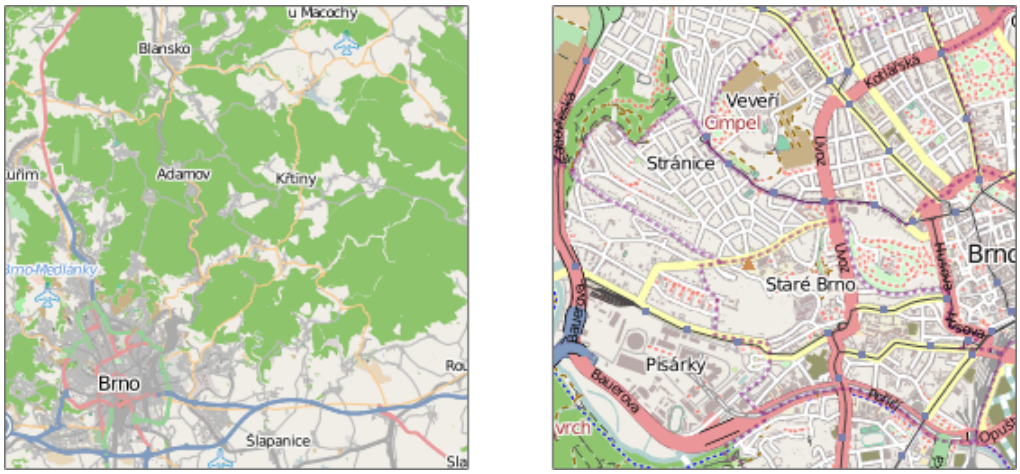
3.2.1 OpenStreetMap

„OpenStreetMap je projekt zaměřený na vytváření svobodných geografických dat. U většiny ostatních volně dostupných map je ale užívání technicky a právně omezeno. Proto vznikl tento projekt, aby umožnil lidem volně nakládat s geografickými daty, používat je neobvyklými způsoby a v neposlední řadě, aby byla data dostupná v aktualizované a platné podobě bez dalších nákladů a omezení.“ [8]

Projekt umožňuje uživatelům získat geografická data celého světa, a to jak v rastrové, tak i ve vektorové podobě. Obě formy dat lze získat buď z online mapy na adrese *www.openstreetmap.org* nebo z databáze jejíž nejnovější revize jsou přístupné na stránce *http://wiki.openstreetmap.org/wiki/Planet.osm*. Rastrové podklady jsou ve formě takzvaných dlaždic, což je obrázek typu PNG o velikosti 256×256 pixelů (Obrázek 9). Z těchto dlaždic se spojováním skládá mapa potřebné oblasti a přiblížení. Data jsou

[10] Ortofoto mapa je mapa složená z leteckých záběrů, jež jsou kalibrovány, a jejich měřítko přesně odpovídá mapě území, které pokrývá. Snímky jsou pořizovány většinou letecky z menších výšek.
[11] Satelitní mapy jsou podobné ortofoto mapám ale snímky, z nichž se skládají mapy, jsou pořizovány ze speciálních satelitů na oběžné dráze země.

nabízena pod licencí Creative Commons Attribution-ShareAlike 2.0 [12], která dovoluje jakékoliv zacházení s materiálem za podmínky uvedení autora.



Obrázek 9: OSM dlaždice 256×256 pixelů [8]

Vektorové podklady jsou distribuovány ve formě souborů XML. Konkrétní podoba tohoto formátu se nazývá JOSM. Díky tomu, že se jedná o XML, jsou data snadno čitelná a zpracovatelná. Na druhou stranu však mohou soubory, obsahující mapové podklady celých států, mít velikost několik GB.

Formát JOSM obsahuje v těle dokumentu několik důležitých elementů. Na začátku se nachází element `bounds`, který určuje hranice výřezu mapy popisované daným dokumentem. Za ní většinou následuje seznam nodů. Element `node` může být jednoduchý a obsahovat jen nejdůležitější informace, kterými jsou `id`, `zeměpisná šířka` a `zeměpisná délka`. Takové nody jsou součástí složitějšího objektu a samy o sobě na mapě neexistují. Složitější nody, definující nějaké místo nebo bod zájmu, obsahují uvnitř elementu `node` navíc seznam elementů `tag`. Element `tag` popisuje jednotlivé vlastnosti prvků. Těmito vlastnostmi může být název, typ, výšková úroveň a jiné. Za seznamem nodů následuje seznam cest. Z cest jsou tvořeny otevřené i uzavřené geometrické prvky. Sám element `way` neobsahuje souřadnice jednotlivých bodů, ale pouze reference na dříve uvedené nody. Kromě referencí jsou zde opět uvedeny elementy `tag` pro popsání vlastností objektu.

[12] Licence Creative Commons je soubor veřejných licencí, které přinášejí nové možnosti v oblasti publikování autorských děl: posilují pozici autora při rozhodování, za jakých podmínek bude dílo veřejně zpřístupněno.

Příklad formátu JOSM:

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.5' generator='JOSM'>
  <bounds minlat='51.5076' minlon='0.12798' maxlat='51.5077' maxlon='0.1277' />
  <node id='265986 visible='true' lat='51.5076' lon='0.12780' />
  <node id='2682110' user='dank' visible='true' lat='51.50772' lon='0.127968'>
    <tag k='created_by' v='Potlatch 0.10f' />
    <tag k='name' v='Nelson&apos;s Column' />
    <tag k='tourism' v='attraction' />
    <tag k='monument' v='statue' />
  </node>
  <way id="22768483" user="vrabcak" uid="8007" visible="true" version="3">
    <nd ref="'265986 "/>
    <nd ref="2444215"/>
    <tag k="highway" v="service"/>
    <tag k="service" v="parking_aisle"/>
  </way>
</osm>
```

Celý projekt funguje podobně jako Wikipedie[13], kdy je obsah vkládán samotnými uživateli a tudíž kvalita i množství vložených dat jsou závislé na nich. Nové mapy lze tvořit buď v terénu, sběrem a následným exportem logu z GPS přijímače, nebo použitím některého z dostupných editovacích programů a vytvářet mapy na základě importovaných satelitních či leteckých snímků. Široká členská základna zaručuje, že se projekt bude v budoucnu dále vyvíjet a nabízet kvalitní geografická data pro široké možnosti použití. [8]

[13] http://www.wikipedia.org/

4 Analýza

Při počátečním zkoumání problému a při specifikaci požadavků se zjistilo, že ne všechny postupy a technologie uvedené v zadání práce budou nejvhodnější pro implementaci systému. Toto platí zejména o spojení pomocí GSM sítě a z toho vyplývající použití konkrétního GPS lokátoru. Jedním ze základních požadavků je nezávislost na sítích třetích stran a možnost provozu v terénu. Signál GSM, konkrétně pak pokrytí technologii GPRS, není 100% na celém území České republiky (pokrytí záleží na konkrétním mobilním operátorovi). Navíc je nutné za přenesená data platit, což je jen další nevýhodou. Proto byly podrobeny zkoumání alternativní druhy spojení a jako nejvhodnější kandidát byla vybrána síť APRS (viz kapitola 2.5). Této technologii byl také přizpůsoben výběr hardwaru. Místo původně uvažovaného trackeru Siemens byl vybrán APRS tracker Argent Data Tracker2 model OT2m. Více o něm v kapitole 7.3.

Protože hardwarová část projektu se pouze integruje do navrhovaného softwarového řešení, je většina analýzy věnována právě softwarové části. Byly specifikovány požadavky na budoucí systém, sestrojeny diagramy užití zachycující nejdůležitější operace a diagram nasazení pro popis architektury systému. Dále byly provedeny dvě analýzy týkající se výběr technologií použitých při implementaci.

4.1 Specifikace požadavků

Při zahájení práce na systému byly jednotlivé jeho plánované části podrobeny zkoumání, aby se zjistilo, jaké budou na systém kladeny požadavky. V první fázi byly požadavky stanoveny autorem projektu. V další fázi byli přizváni potencionální uživatele aplikace a formou diskuze nad daným problémem byly specifikovány další požadavky. Požadavky jsou specifikovány zvlášť pro každou ze tří hlavních částí systému. Požadavky jsou vypsány pouze textově. Diagram požadavků je v příloze A.

4.1.1 Funkční požadavky

Požadavky na klientskou část

- Aplikace bude ovládána pomocí grafického uživatelského rozhraní, které bude možno doplňovat o další funkcionality.

- Grafické uživatelské rozhraní musí umožňovat pohodlné dotykové ovládání.
- Aplikace umožní provádět s mapou běžné úkony jako je zoomování a posouvání.
- Aplikace bude obsahovat pluginovací subsystém s jednoduchou správou pluginů.
- Aplikace bude nastavení ukládat do externí databáze.
- Navrhnou API, které by zpřístupňovalo funkce pro pluginy.
- V jádru aplikace ponechat pouze nejnutnější funkce pro renderování map a API. Ostatní doplnit přes pluginy.
- Aplikace bude mít optimalizovaný proces renderování map.
- Aplikace umožní vytvářet projekty zapouzdřující mapy, styly a další informace.
- Aplikace umožní definovat výsledný grafický vzhled map pomocí stylů.
- Aplikace musí umět pracovat s některým z nekomerčních mapových formátů.

Požadavky na server

- Aplikace poběží nezávisle na klientské aplikaci.
- Aplikace bude implementována jako konzolová aplikace nebo služba systému.
- Aplikace bude schopna komunikovat přes více než jeden protokol, čímž bude zaručena funkčnost na různých systémech.
- Aplikace bude podporovat jak synchronní tak asynchronní komunikaci.
- Aplikace implementuje databázový subsystém pro ukládání nasbíraných dat.
- Aplikace umožní ostatním částem systému přístup k databázovému subsystému.
- Aplikace bude rozšiřitelná pomocí pluginu – použit systém vyvinutý pro klienta.
- Příjem dat ze sítě, ať již od klientské aplikace či trackeru, zajistit pomocí pluginů.

Požadavky na tracker

- Zařízení bude získávat data z GPS přijímače a odesílat je na server.
- Zařízení bude umožňovat komunikace prostřednictvím různých bezdrátových sítí (Upřednostňované jsou sítě, které pracují nezávisle na sítích třetích stran).
- Zařízení umožní v budoucnu přímé propojení s počítačem v terénu.

4.1.2 Nefunkční požadavky

Požadavky na klientskou část

- Aplikace bude napsána v programovacím jazyce C#.
- Aplikace bude podporovat různé operační systémy včetně Microsoft Windows a Linux.
- Aplikace bude pro komunikace se serverem využívat standardních kanálů – na stejném počítači Named Pipes či jiný IPC a na síti TCP binding či Web Services.

Požadavky na server

- Aplikace bude napsán v programovacím jazyce C#.
- Aplikace bude podporovat různé operační systémy.

4.2 Případy užití

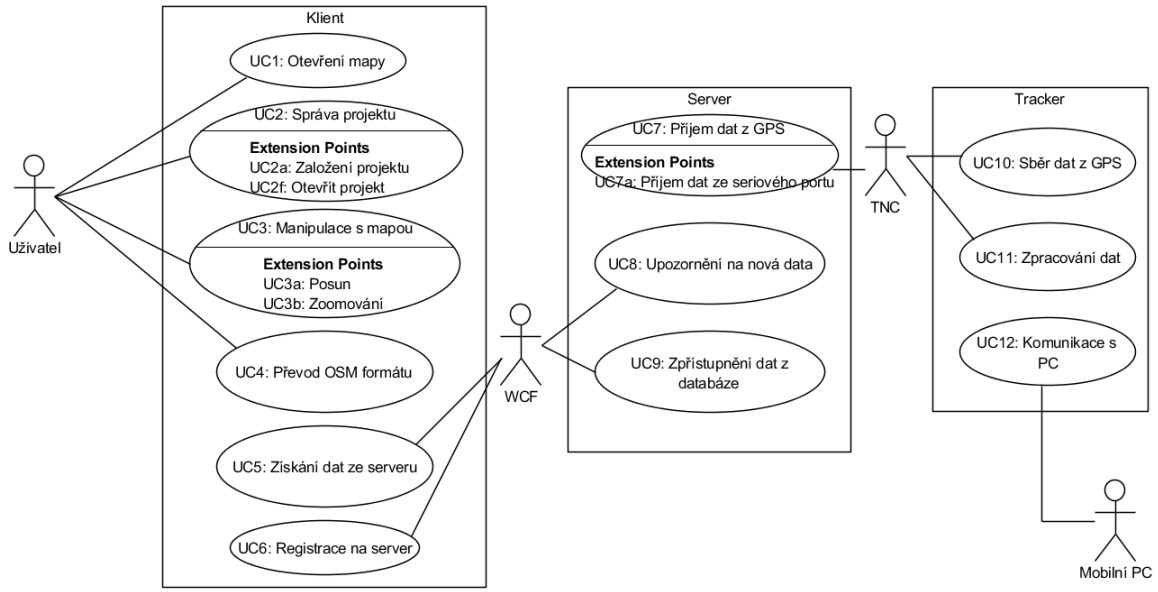
V této kapitole jsou popsány případy užití pro vybrané scénáře. Případy užití jsou popisovány diagramy případů užití a poskytují tak jednoduchý způsob jak nastínit, co se děje v systému a jak s ním přicházejí do interakce různí aktéři. Aktérem může být osoba, jiný systém, hardware, atd. Aktér spouští akci nebo přijímá výsledek nějaké akce (případu užití). Případy užití popisují dynamické chování systému bez bližšího popisu vnitřní implementace jednotlivých funkcí.

V diagramech jsou popsány nejdůležitější postupy ve třech systémech: Klient, Server a Tracker. S těmito systémy přicházejí do styku tito aktéři:

- **Uživatel** – jedná se o osobu, která bude pracovat s klientskou aplikací, vytvářet a otevírat projekty, manipulovat s mapou a převádět mapové formáty. Uživatel se také stará o spuštění klientské aplikace, případně i o spuštění serverové aplikace, pokud je umístěna na stejném počítači jako klientská a dosud není spuštěna. V systému neexistují uživatelské role, proto má jakýkoliv uživatel s přístupem ke klientské aplikaci neomezená práva.
- **WCF** – Windows Communication Foundation je framework (softwarový systém) umožňující spojení a přenos dat mezi jednotlivými systémy. Objevuje se tam, kde je potřeba přenést informace mezi klientskou a serverovou částí systému.

- **TNC** – představuje zařízení pro spojení s mobilní bezdrátovou sítí pro přenos dat. Při příchodu nových dat spouští příslušné akce související se zpracováním dat.
- **Mobilní PC** – tento aktér představuje možné budoucí rozšíření. Pracuje pouze s Trackerem. Získává z něj informace a nové mu posílá. Přes tracker tak komunikuje s klientskými aplikacemi a dalšími Trackery v terénu. Přináší možnost nasazení klientských aplikací i na straně Trackeru a rozšíření informačních a taktických možností přímo v terénu. V reálném světě tento aktér může vystupovat jako Tablet PC, PDA, chytrý telefon nebo notebook.

Na Obrázku 10 je zobrazen diagram popisující aktéry systému a obecnější případy užití, který rozšiřují případy užití uvedené v další části kapitoly. Jsou zde znázorněny všechny tři systémy a jejich provázanost v rámci případů užití. V diagramu jsou zobrazeny jen nejdůležitější případy užití. Některé z nich jsou dále podrobněji popsány. Aktér ***Uživatel*** přichází do interakce pouze se systémem ***Klient*** a může otevírat mapy, spravovat projekty, manipulovat s mapou a převádět mapové formáty. Se zbylými případy užití pracuje aktér ***WCF***, který převádí data mezi ***Serverem***. ***Server*** sám nemá příliš mnoho funkcí. Jedná se hlavně o centralizovanou databázi pro sběr dat z ***Klienta*** a ***Traceru***.

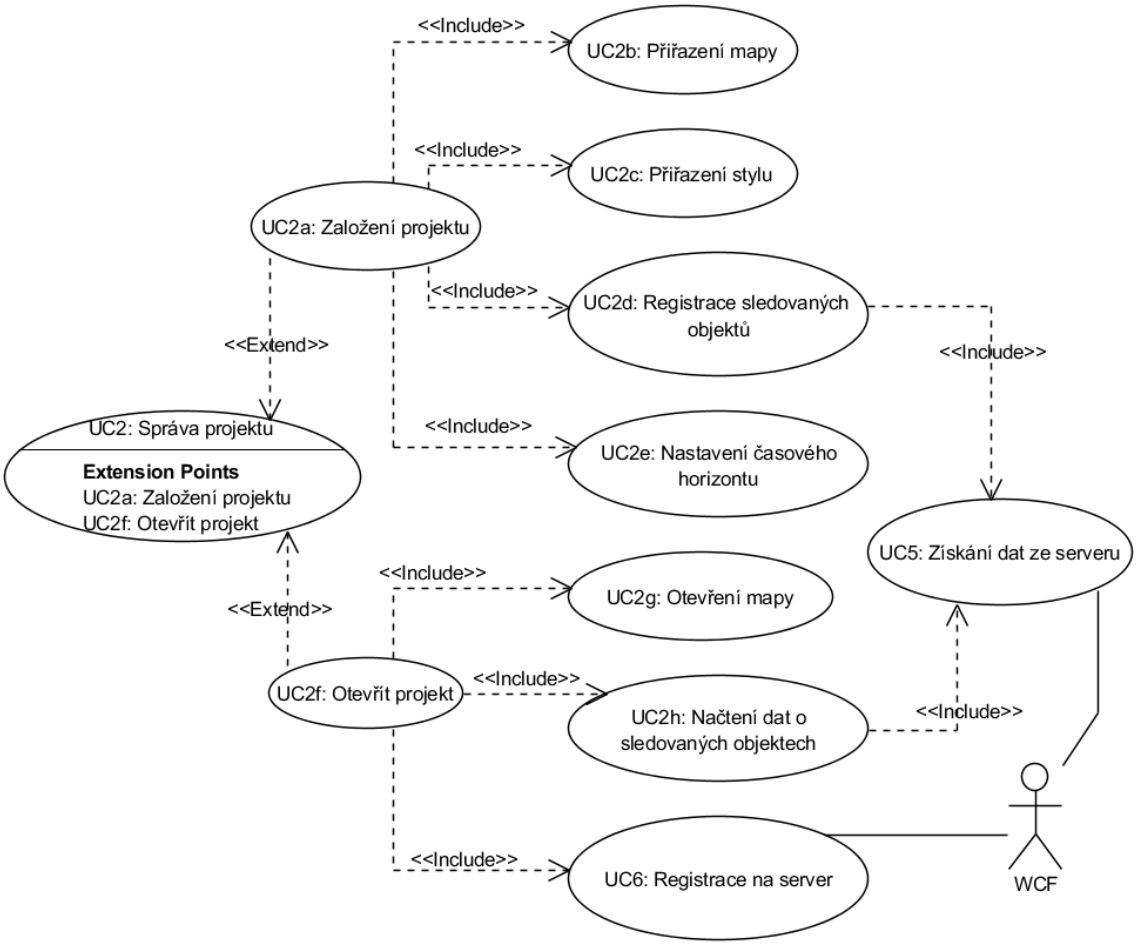


Obrázek 10: Případ užití – obecný přehled

Zdroj: autor

Podrobnější případy užití rozšiřují dříve uvedené o detailnější popis, který lépe zobrazuje jednotlivé funkce systému. Případy užití jsou rozloženy na více malých akcí, které se

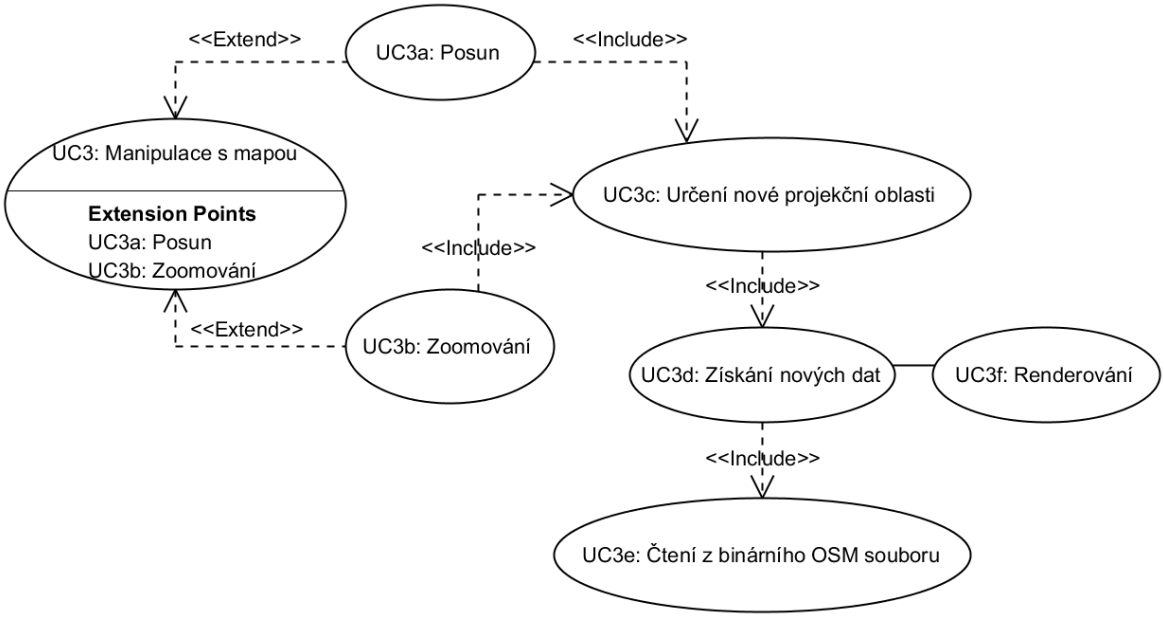
budou při implementaci jednodušeji zpracovávat v samostatných iteracích. Na Obrázku 11 je rozšířen případ užití **Správa projektu** o případy **Založení projektu** a **Otevření projektu**. **Založení projektu** se skládá z činností vedoucích k vytvoření projektu, podle kterých se bude tvořit příslušný modul případně i formát projektu. V případu **Registrace objektů** je nutné získat údaje z databáze ***Serveru***, proto zde přichází do interakce také aktér ***WCF***.



Obrázek 11: Případ užití Správa projektu

Zdroj: autor

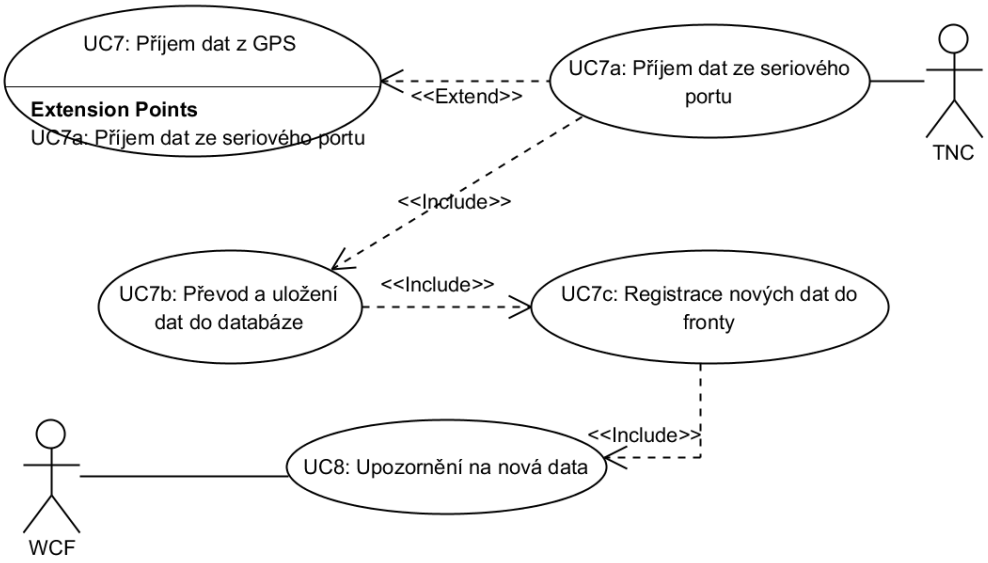
Obrázek 12 popisuje případ užití **Manipulace s mapou**. Případy užití **Posun** a **Zoomování** tento případ rozšiřují. Pokud aktér ***Uživatel*** provede některou z těchto akcí, musí se přepočítat nová projekční oblast. Po jejím vypočítaní se načtou nová data pro tuto oblast, což zahrnuje čtení z binárního souboru, obsahujícího vektorová data pro mapy. Nová data jsou poslána jádru systému, kde probíhá renderování.



Obrázek 12: Případ užití Manipulace s mapou

Zdroj: autor

Posledním popsaným rozšiřujícím případem užití je **Příjem dat z GPS**. Sběr informací o poloze zajišťuje ***tracker***. Ten je připojen do bezdrátové sítě, kterou v modelech představuje aktér ***TNC***. Ten spouští případ užití **Příjem dat ze sériového portu.** Data jsou po přečtení ze sériového portu zpracována a odeslána dále, kde jsou převedena do konečného formátu a uložena do databáze. Při ukládání dat do databáze je zároveň provedena registrace dat do fronty. V pravidelných časových intervalech je tato fronta kontrolována, a pokud obsahuje data, pokračuje se v případu užití **Upozornění na nová data**. Zde vstupuje do hry aktér ***WCF*** přes kterého se nová data posílají klientovi.



Obrázek 13: Případ užití Příjem dat Z GPS

Zdroj: autor

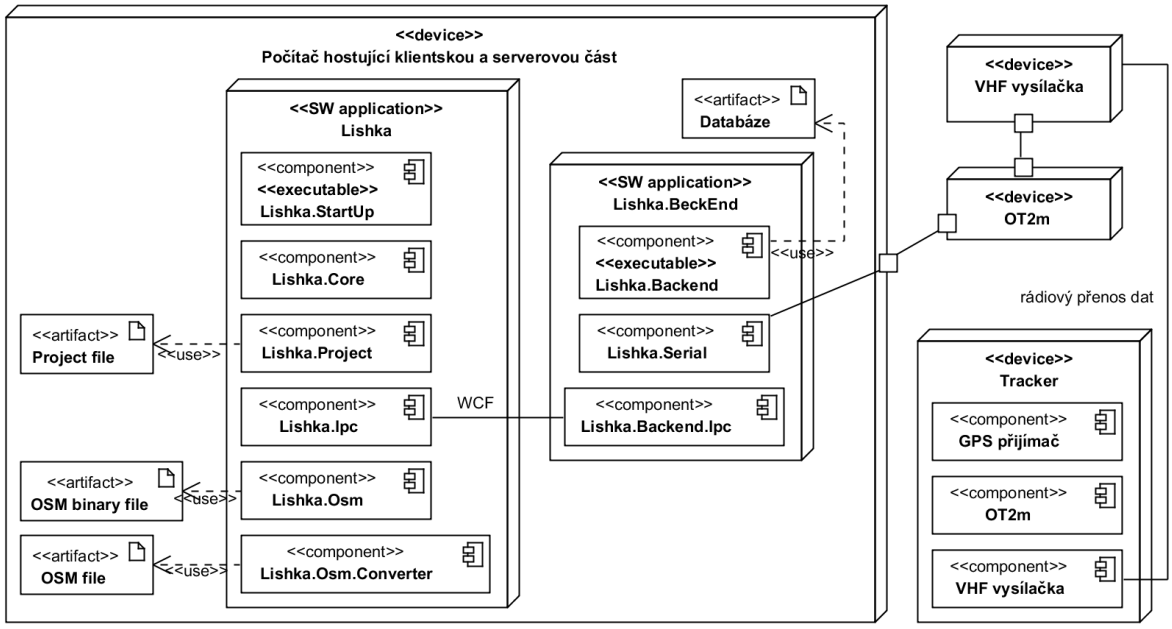
4.3 Architektura systému

Architektura systému vyplývá z dříve stanovených požadavků. Je navrhována tak, aby celý systém mohl fungovat v terénu bez nutnosti připojení k jiné síti než navrhované. V takovém případě by se dosah signálu měl pohybovat ve vzdálenostech 3–5 km mezi jednotlivými uzly.

Na Obrázku 14 je zobrazena jedna z možných konfigurací systému, kdy je klientská i serverová aplikace hostována na jednom počítači. Tento počítač představuje hlavní funkční jádro celého systému. Operační systém, který na tomto počítači poběží, není ve fázi analýzy přesně specifikován a je závislý na dostupnosti vhodných zařízení na trhu. Pro účely nasazení v terénu je vhodné, aby tento počítač tvořil netbook či notebook s co nejvyšší vydrží. Výkon toho zařízení není oproti výdrži již tolik důležitý. Při vývoji a testování však bude nasazen operační systém Microsoft Windows 7 a vhodná Linuxová distribuce podporující nejnovější verzi Mono[14]. Na počítači pak běží aplikace Lishka a Lishk.BeckEnd, jež tvoří klientskou a serverovou část. Tyto aplikace využívají několik externích souborů, jako jsou projekty, mapy či databáze. K hostujícímu počítači musí být také připojena VHF vysílačka či jiné zařízení schopné připojit se do APRS sítě.

[14] Mono je softwarová platforma pro vývoj multiplatformních aplikací pomocí jazyků .NET Framework.

Původní návrh s přenosem dat realizovaným pomocí GSM sítě byl zavrhnut pro jeho nedostatky popsané dříve a byl nahrazen sítí APRS. Do ní se jak na straně serveru, tak na straně trackeru připojují zařízení pomocí VHF vysílaček podporující protokoly APRS. Samotný tracker se skládá celkem ze tří částí. Jádrem je APRS tracker OT2m, k němuž se přes porty RS232 připojují GPS přijímač a VHF vysílačka. K OT2m je rovněž možno připojit počítač a rozšířit tak možnosti trackeru. Tato varianta však není v této fázi plánována.



Obrázek 14: Architektura navrhovaného systému

Zdroj: autor

4.4 Analýza grafických subsystémů

Tato fáze analýzy byla vytvořena až v době implementace klientské části systému. Původně nebyla plánována analýza grafických subsystémů a byl zvolen WPF jako hlavní subsystém pro vykreslování map. WPF mělo být určeno pro operační systémy Microsoft Windows s tím, že pro alternativní systémy bude renderovací jádro vyvíjeno zvlášť. Tento přístup se však projevil jako velmi nepraktický a zároveň výkon WPF pro účely aplikace nebyl takový, jaký se předpokládal. Na základě těchto okolností byla provedena dodatečná analýza, ze které byl vybrán nový grafický subsystém.

Pro renderování map a obecně 2D grafiky v jazyce C# existuje několik grafických subsystému a frameworků, které lze k tomu účelu využít. V Tabulce 2 jsou vypsány subsystémy, které mají svoji implementaci pro jazyk C#. Liší se nejen podporovanými operačními systémy, ale především způsobem, jakým pracují s grafickými objekty. Většina grafických knihoven funguje v *immediate módu*, což znamená, že uživatelská volání způsobují okamžité vykreslení objektů a defaultně se nepoužívá double buffering. Stav grafických objektů není nikde uložen a při změně některého z nich je třeba překreslit celou plochu. Naproti tomu existují systémy pracující v *retained módu* (např. WPF). Systémy pracující v *retained módu* se samy starají o použití double bufferingu a všechny grafické objekty jsou ukládány v paměti včetně jejich stavů. Uživatelská volaní zde nezpůsobují přímé vykreslení, ale pouze změnu stavů objektů, které jsou pak vykresleny pomocí svých vlastních renderovacích metod. Výhodou *retained módu* je jednoduchá práce s grafickými objekty, jejich vytváření a úprava stavů, ovšem za cenu vyšších režie. Každý grafický objekt existuje jako instance třídy v paměti a tím narůstá jak paměťová, tak časová náročnost.

Tabulka 2: Přehled grafických subsystému s podporou jazyka C#

Grafický subsystém	**Podporované OS**	**Poznámka**
GDI+	Windows, Windows Mobile, Windows Phone, Windows CE, Linux, Mac OS	Alternativní OS s využitím Mono
WPF	Windows	Retained mód
Silverlight	Windows, Windows Phone, částečně Linux	
Cairo	Linux, Windows	
XNA	Windows, Windows Phone	HW akcelerace
Direct 2D	Windows	HW akcelerace
SlimDX	Windows	HW akcelerace

Pro aplikaci Lishka byly testovány tři grafické subsystémy: WPF, GDI+ a Cairo. Protože se pro renderování map používají z grafického subsystému pouze jednoduché funkce pro kreslení multičar, polygonů, kružnic a textu, je snadné přizpůsobit kód různým API. Popis všech testovaných systému je uveden v následujících odstavcích. Výsledky testování, se zaměřením na časovou náročnost, jsou uvedeny v Tabulce 3. Pro testování je využito výřezů na několika vybraných úrovních zoomu, vždy se stejným středem mapy (centru a okolí Bratislavy).

Tabulka 3: Porovnání časové náročnosti vykreslování různých grafických systémů

Grafický subsystém	Čas pro vykreslení mapy [ms]			Poznámka
	Zoom 8	Zoom 11	Zoom 14	
WPF	585	815	395	
GDI+ pod Windows	*182*	*445*	*155*	
Cairo pod Windows	245	528	164	
GDI+ pod Linuxem	340	840	265	Virtuální PC
Cairo pod Linuxem	378	912	291	Virtuální PC

Testování subsystémů probíhalo ve fázi, kdy bylo renderovací jádro již několikrát optimalizováno. I přesto zde zůstává mnoho prostoru pro budoucí optimalizace. Optimalizovat se dají jak použité algoritmy, tak práce se samotným grafickým subsystém.

4.4.1 WPF

Prvním testovaným subsystémem je Windows Presentation Foundation. Na rozdíl od dvou dalších subsystému pracuje WPF v *retained módu*. Pro vykreslování složitých mapových podkladů není tento způsob nejlepší, protože mapa je vykreslena jako pozadí, na němž se jednotlivé mapové prvky již nemění. Pokud by se každý prvek (cesta, nod, oblast) tvořil jako samostatný objekt v paměti, bylo by překreslovaní každého z nich při posunu či změně měřítka velice neefektivní. Naštěstí dovoluje WPF pracovat s grafikou i na nižší úrovni, která se mnohem více podobá práci v *immediate módu*.

Pro práci s grafikou na nejnižší úrovni slouží třída `DrawingVisual` ze jmenného prostoru `System.Windows.Media`, což je nejjednodušší prvek, který lze využít pro renderování. Samotné kreslení se neprovádí s třídou `DrawingVisual` ale s jejím `DrawingContext`. Ten získáme voláním metody `RenderOpen`, třídy `DrawingVisual`. `DrawingContext` obsahuje velké množství funkcí pro vykreslování jednoduchých geometrických objektů, ale i složitých geometrií. Právě metodu pro vykreslování geometrií `DrawGeometry` využijeme pro téměř veškeré renderování map. Parametrem, který se musí předat této metodě je objekt reprezentující geometrii. Může se jednat o geometrii kruhu, polygonu či textu. To, že je tato geometrie poměrně komplexním objektem, má dopad na výkonnost. Ve WPF existuje několik tříd pro geometrie, lišící se svoji funkcionalitou a komplexností.

Nejjednodušší a pro účely aplikace dostačující geometrií je *StreamGeometry*. Tímto způsobem lze poměrně rychle vykreslit velké množství grafických objektů.

Samotné vykreslení do *DrawingVisual* však nezpůsobí zobrazení na obrazovce. Stále se nacházíme v *retained módu*, proto musíme tento objekt předat vyšší vrstvě, která se stará o vykreslování. K tomu můžeme využít třídu dědící ze *System.Windows.FrameworkElement*, což je třída představující základní prvek, jež lze použít v kontejnerech pro zobrazení na obrazovce. Z toho vyplývá, že pro práci s jednoduchou grafikou není WPF příliš ideální. Při přihlédnutí k podporovaným operačním systémům je tedy jasné, že výběr tohoto grafického subsystému nebude na místě. Z výsledků v Tabulce 3 je vidět, že WPF se při testování umístilo až na posledním místě.

4.4.2 GDI+

Nejvýhodnějším grafickým subsystém je z hlediska podpory OS a doby vykreslování jednoznačně GDI+. GDI+ pracuje v *immediate módu*. Použití double bufferingu lze zapnout nebo vypnout. Pro jeho efektivnější využití je však nutná vlastní implementace. Toho se dá při zvolení vhodných technik využít ve prospěch zvýšení výkonu. Na rozdíl od WPF odpadá nutnost používat různé „obalové“ třídy. Pro kreslení stačí získat instanci *Graphics* toho prvku, na který chceme kreslit. Třída *System.Drawing.Graphics* poskytuje všechny potřebné metody pro renderování geometrických objektů a textu. Při přímém vykreslovaní na *Graphics* Controlu, který je již zobrazen dochází ke ztrátě výkonu a při překreslování scény k nepříjemnému blikání. Pro výrazné zrychlení renderování stačí implementovat jednoduchý double buffering. Ten se vytvoří např. ze třídy *System.Drawing.Bitmap* a prostřednictvím jejího *Graphics* na ní kreslíme. Výslednou mapu tak máme vyrenderovanou mimo obrazovku, kam jí pak přeneseme najednou předáním metodě *Graphics.DrawImageUnscaled* požadovaného Controlu. Na rozdíl od vestavěného double buffurenigu, lze mít ve vlastní implementaci plnou kontrolu nad mimoobrazovkovým bufferem a rozhodovat o tom, kdy je potřeba znovu načítat grafická data a kdy stačí použít již načtená z bufferu. To vše ve prospěch zvýšení výkonu.

4.4.3 Cairo

Cairo je platformě nezávislá grafická knihovna pro práci s 2D grafikou[15]. Je využívaná hlavně v grafickém frameworku GTK+ a jako backend v prostředí Mono, kde jsou na něj převáděna volání GDI+. Při použití knihoven Mono můžeme využívat třídy Cairo přímo nebo v případě použití operačního systému Linux, psát kód stejně jako při použití GDI+. V Linuxu neexistuje implementace GDI+, proto se vždy ve výsledku všechno vykreslování provádí prostřednictvím Cairo. Výsledky testování GDI+ a Cairo pod Linuxem by tedy měli být shodné. Ve skutečnosti tomu tak není. Použití tříd Cairo přímo je pomalejší, což je nejspíše způsobeno tím, že volání přes GDI+ jsou v Mono lépe optimalizována. Samotná práce s Cairo je velice podobná GDI+. Výsledky testování jsou opět uvedeny v Tabulce 3. Horší výsledky pod operačním systémem Linux jsou částečně způsobeny tím, že testovací systém běžel ve virtuálním PC.

4.5 Analýza možností zpracování rozsáhlých souborů XML

Další analýzou prováděnou při implementaci je analýza zpracování rozsáhlých souborů XML, jež má za cíl nalézt vhodný postup pro práci se soubory, jejichž velikost může dosahovat několika GB. Aby bylo možné v této analýze najít takové řešení, je nutné pracovat s konkrétními komponentami a datovými strukturami.

Jednou z možných cest je použití LINQ to SQL. Zde však narážíme na dva zásadní problémy: rychlost a paměťová náročnost. Při zpracování souboru se celý jeho obsah načítá do paměti a z UTF-8 se konvertuje do UTF-16, což znamená, že v paměti zabírá dvakrát tolik místa.

Pokud je soubor dostatečně malý, lze ho celý načíst. Jeho procházení je však bez použití pomocných struktur neúnosně pomalé a to kvůli způsobu jakým jsou v XML souboru uloženy definice cest a relací[16] Možností, jak zpracování zrychlit je použití pomocné datové struktury `Dictionary`[17] ze jmenného prostoru `System.Collections.Generic` do které se při prvním průchodu souboru načtou všechny nody. Klíčem bude jejich ID a hodnotou samotný nod. Při druhém průchodu se již nody k jednotlivým cestám či relacím

[15] Více informací o Cairo včetně dokumentace na http://www.cairographics.org.
[16] Více informací o jednotlivých elementech lze získat na oficiálních stránkách projektu OpenStreetMap http://wiki.openstreetmap.org/wiki/Data_Primitives nebo v kapitole 3.2.1.
[17] Podrobný popis třídy Dictionary<TKey, TValue> http://msdn.microsoft.com/en-us/library/xfhwa508.aspx.

budou hledat v `Dictionary`, kde jsou uloženo pod jejich ID a odpadá tudíž zdlouhavé vyhledávání XML stromu. Zavedení této pomocné struktury zvýší rychlost zpracování zhruba 800krát. Bohužel stále přetrvává problém s paměťovou náročností.

Možností jak pracovat s rozsáhlými XML soubory je použití třídy `XStreamingElement`[18] ze jmenného prostoru `System.Xml.Linq`, která načítá danou část XML stromu až ve chvíli, kdy jej opravdu zpracovává. Tímto způsobem lze parsovat velice rozsáhle soubory. [15] Pro dosažení větší rychlosti jsou opět všechny nody nejprve načteny do `Dictionary`. V Tabulce 4 jsou porovnány všechny tři přístupy z hlediska rychlosti zpracování. Také je zde vidět, jak velké soubory jdou s daným přístupem zpracovat.

Tabulka 4: Porovnání rychlostí zpracování souborů OSM

Použitý přístup	**Doba zpracování souboru OSM [ms]**		
	Malý (4,59 MB)	**Střední (340 MB)**	**Velký (1,16 GB)**
LINQ to SQL	94 250	nelze	nelze
LINQ to SQL + Dictionary	116	7 590	nelze
XStreamingElement	745	54 270	249 526

[18] http://msdn.microsoft.com/en-us/library/system.xml.linq.xstreamingelement.aspx

5 Technologie a nástroje použité k vypracování

5.1 Technologie

Mono je softwarová platforma dovolující vývojářům snadno vytvářet multiplatformní aplikace. Jedná se o open source implementaci Microsoft .Net Framework podle platných standardů pro jazyk C# a CLR. Díky Mono lze jednoduše vytvářet aplikace pro operační systémy Microsoft Windows a poté téměř bez úprav tyto kódy použít pro sestavení aplikace pod systémy Linux či Apple Mac OS. V současné době je implementována většina funkcí z .NET Framework 3.5 a 4.0, s výjimkou WPF. [17]

Mono je v projektu Lishka využito ke zkompilování zdrojových kódů pod operačním systémem Linux. Pro kompilaci a úpravy bylo použito vývojové prostředí MonoDevelop 2.4 s Mono 2.6 běžících na operačním sytému OpenSuse 11.2. Kromě drobných úprav v rozšířeních pro spojení klientské a serverové části pomoci WCF, kde v Mono nejsou zatím implementovány Named Pipes[19], bylo možné ponechat všechny kódy ve stejné podobě jak pro Windows, tak pro Linux.

Managed Extensibility Framework[20] je komponenta .NET Framework 4.0 pro vytváření rozšiřitelných aplikací. Dává vývojářům možnost jednoduše vyvíjet moduly, které jsou poté za běhu připojovány a to téměř s nulovou potřebou konfigurace. Mono od verze 2.6 také obsahuje plnou podporu MEF. V aplikaci Lishka je MEF použit jako základní stavební kámen pro plugin engine. Kromě samotných pluginů je MEF také použit jako Dependency Injection[21] na uvolnění vazeb mezi objekty.

Pro komunikace mezi klientem a server je využit komunikační framework Windows Communication Foundation. Jedná se o soubor knihoven obsažených v .NET Framework a s částečnou podporou v Mono. WCF zapouzdřuje většinu běžně používaných komunikačních technologie pro síťovou komunikaci, jako jsou běžné Web Services, TCP binding, nebo např. Named Pipes pro IPC.

[19] Named Pipe slouží pro komunikace mezi procesy na stejném počítači nebo mezi procesy na různých počítačích skrze síť.
[20] http://mef.codeplex.com
[21] Dependency Injection je návrhový vzor, ve kterém objekty samy nevytvářejí objekty, které využívají, ale jejich tvorbu a připojení nechávají na třetí straně.

System.Data.SQLite[22] je ADO.NET provider pro databázový engine SQLite. Díky němu lze v projektu Lishka používat open source databáze SQLite. Databáze SQLite jsou použity v klientské části pro ukládání nastavení. Zde je použita implementace s přímým přístupem a zadáváním SQL dotazů bez použití ORM nástrojů, a to kvůli nízkému počtu (2) tabulek v databázi. Dále byla tato databáze použita v serverové části, kde byl pro přístup použit ORM nástroj NHybernate.

NHibernate[23] je všestranným ORM nástrojem pro platformu .NET. Jedná se o port původního Hibernate napsaného v jazyce Java. V serverové části aplikace Lishka je použit pro přístup k databázi SQLite uchovávající informace o mapových objektech a jejich poloze. Protože je vůči databázi prováděno množství různých dotazů od jednoduchých až po složitějších, bylo rozhodnuto o nasazení tohoto ORM pro zjednodušení práce s databází v ostatních vrstvách aplikace.

5.2 Nástroje

Hlavním vývojovým prostředím je Microsoft Visual Studio 2008[24]. Toto vývojové prostředí poskytuje podporu pro mnohé .NET frameworky a formáty souboru. Není problém navrhovat uživatelské rozhrání, psát kód v jazyce C# nebo XML. Díky dostupným pluginům je možné přidat podporu pro databáze SQLite a použít pro jejich správu vestavěný designer.

Pro úpravy a kompilaci v operačním systému Linux je použito vývojové prostředí MonoDevelop 2.4[25] od firmy Novell. Nenabízí takové možnosti jako Visual Studio a práce v něm není tak pohodlná, ale i přesto se jedná o velice dobré vývojové prostředí pro platformu .NET. Je k dispozici pro operační systémy Windows i Linux a je nabízeno zcela zdarma.

Pro snadnější testování na operačním systému Linux bylo využito virtualizace. Jako virtualizační nástroj byl použit VMware Player 3[26]. Díky tomu mohla být aplikace testována současně v několika systémech na jednom počítači. Nevýhodou je menší výkon

[22] http://sqlite.phxsoftware.com
[23] http://www.nhforge.org
[24] Informace o jednotlivých verzích Visual Studio, včetně možnosti stažení zkušebních verzí, jsou k dispozici na adrese http://www.microsoft.com/cze/msdn/vstudio.
[25] http://www.monodevelop.com
[26] http://www.vmware.com/products/player

virtualizovaného systému. Testovací Linuxovou distribucí je OpenSuse 11.2[27]. Jedná se o distribuci vyvíjenou firmou Novell, tudíž obsahuje podporu nejnovějších knihoven platformy Mono.

5.3 Plugin engine

Ve fázi analýzy bylo zvažováno, kterou z dostupných technologií použít pro tvorbu pluginovacího jádra. Hlavním požadavkem bylo, aby vybraná technologie umožňovala připojování knihoven DLL za běhu aplikace. Zvažovanými technologiemi byly **System.Reflection**, **Mono.Addins** a **MEF**. Po prozkoumání všech možností, které tyto technologie nabízejí, byl vybrán MEF pro jeho univerzálnost a jednoduchost použití.

Díky použití MEF odpadla nutnost navrhovat a implementovat samotnou infrastrukturu, která by zajišťovala správu a napojování jednotlivých modulů do páteře systému. Toto dělá MEF sám pomocí několika katalogů. Katalogy jsou třídy, které slouží ke shromažďování a následnému propojení objektů z různých zdrojů. Např. `AssemblyCatalog` hledá objekty v konkrétní knihovně DLL, `DirectoryCatalog` zase ve všech DLL v zadaném adresáři. Do katalogů jsou přidávány všechny objekty, jejichž definice jsou označeny atributem `Export` nebo `Import`. Takovým objektům se říká Composable Part. Jednotlivé části jsou v katalogu identifikovány pomocí kontraktů, což jsou unikátní řetězce, tvořené např. celým názvem třídy včetně jmenného prostoru nebo libovolným vloženým názvem. Načtené části se pak navzájem propojí. [16]

Načtení jednotlivých pluginů je první fází. Aby mohly pluginy nějakým způsobem spolupracovat s ostatními částmi systému, musí jim být k tomu poskytnuto nějaké rozhraní. Druhou částí pluginovacího jádra je Lishka API. Jedná se o soubor služeb, zpřístupňujících různé funkcionality systému, které jsou načteny do katalogu jako části Export. Jednotlivé pluginy si je poté mohou importovat a využívat je. Podrobnější informace o službách API jsou popsány v kapitole 7.1.2.

[27] http://cs.opensuse.org

6 Použité algoritmy

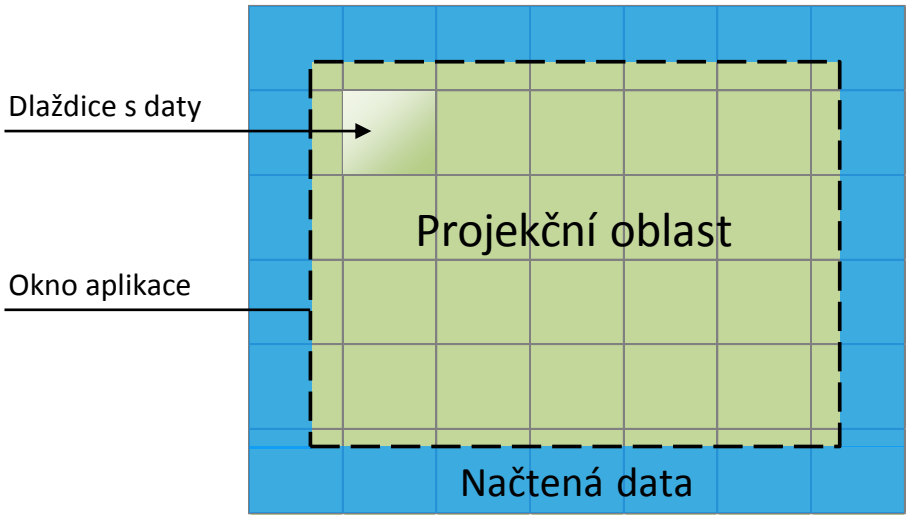
6.1 Renderování mapových podkladů

Renderování je proces, při němž je z dodaných dat vykreslena mapa do výsledné formy tak, jak ji uvidí uživatel v okně aplikace.

Samotný cyklus renderování začíná u pluginu, který chce prezentovat svá data. To, jakým způsobem a jaká data plugin zpracovává, je záležitostí jeho vnitřní implementace. Pokud však chce vykreslovat nějaká z nich do mapy, musí dodržovat standardní formát pro funkce zpřístupněné pomocí aplikačního API.

Konkrétně plugin Lishka.Osm pracuje s binárním formátem OpenStreetMap. Při inicializaci pluginu se importuje `RenderService`, která obsahuje všechny potřebné metody, vlastnosti a události pro renderování map. Po otevření nové mapy se nastaví střed mapy a výchozí úroveň zoomu. Jakmile plugin nastaví tyto vlastnosti, může z `RenderService` získat aktuální projekční oblast, která je určena velikostí okna aplikace, umístěním mapy a zoomem. Projekční oblast si může uživatel měnit sám podle potřeby tím, že posouvá či přibližuje mapu nebo mění velikost okna aplikace. Při každé takové změně je potřeba poskytnout renderovacímu enginu nová data pro renderování. Aby plugin o změně věděl, je nutné v `RenderService` zaregistrovat událost `ProjectionAreaChanged`. Při otevření souboru a při každé změně projekční oblasti se posílají nová data.

Při zahájení renderování se musí volat metoda `RenderOpen`, kde se jako parametry předávají velikost renderované oblasti a její umístění v rámci projekční oblasti. Na konci se pak naopak volá `RenderClose`. Teprve poté jsou data vykreslena na obrazovku. Voláním `RenderOpen` se inicializuje třída `TileRender` k příjmu dat. Ty jsou posílána přes přetížené metody `RenderService.RenderObject`. V binárním souboru jsou data indexována po dlaždicích 256×256 pixelů, proto se musí vybrat takové množství dlaždic, aby byla pokryta celá viditelná část projekční oblasti tak, jak je zobrazeno na Obrázku 15. Při volání `RenderClose` se vytvoří mimoobrazovkový buffer obsahující vyrenderovanou mapu. Tento buffer se poté používá k překreslování mapové oblasti v případech, kdy není nutné načítat nová data (např. minimalizace okna, či změna pozice dynamických objektů).



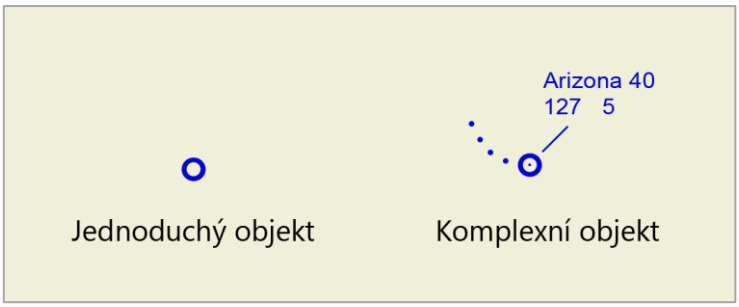
Obrázek 15: Načtená data ve výřezu projekční oblasti

Zdroj: autor

Při renderování dynamických objektů je postup odlišný. Třída `TileRender` zmíněná v předešlém odstavci pracuje v immediate módu, zatímco třída, která se stará o renderování dynamických objektů pracuje v retained módu. Tou třídou je `DynamicRenderer`. Každý dynamický objekt je přidán do rendereru pomocí metody `AddObject`. Všechny takto přidané objekty musí implementovat rozhraní `IDynamicMapObject`. Po vložení jsou uchovávány v kolekci. Každý z objektů sám ví, jak se má vykreslit. Pokud je nutné objekty překreslit, volá se metoda `Refresh` třídy `DynamicRenderer`, ve které se prochází všechny registrované objekty a volá se jejich metoda `Draw`, které se jako parametr předá aktivní `Graphics`. Pokud je potřeba aktualizovat hodnoty objektu, je k dispozici metoda `UpdateObject`, která jako jediný parametr očekává opět rozhraní `IDynamicMapObject`.

Při renderování nepracují pluginy přímo se třídou `DynamicRenderer`, ale se službou API RenderService. Ta práci s dynamickými objekty zjednodušuje – obsahuje pouze dvě přetížené metody `RenderDynamicObject`, které jsou použity jak pro registraci, tak pro aktualizaci objektu. Pokud potřebuje plugin renderovat pouze jednoduchý objekt, jehož výslednou podobu lze definovat mapovými styly, lze využít defaultní implementace rozhraní `IDynamicMapObject`. Pokud však potřebuje plugin renderovat složitější objekty, je nutné, aby sám implementoval vlastní třídu. V ní lze v metodě `Draw` vykreslit libovolně složité tvary nezávisle na stylech. Komplexní objekt je takový objekt, jehož výslednou podobu nelze definovat ve stylech. Pokud se například objekt skládá z více geometrických

útvarů, musí být jeho definice implementována zvlášť ve třídě `Draw`. Rozdíl mezi defaultním objektem a komplexnějším je zobrazen na Obrázku 16. Zobrazený komplexní objekt je tvořen velkým kruhem, který představuje aktuální pozici a malými tečkami, které představují 4 poslední pozice objektů (složí k určení směru pohybu). Dále objekt obsahuje textové a informace: volací znak Arizona 40 a pomocné číselné hodnoty.



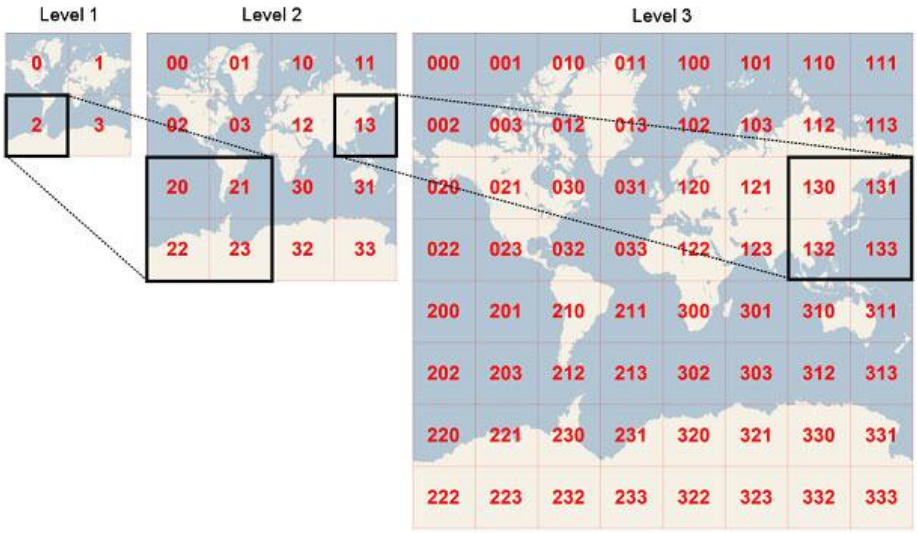
Obrázek 16: Dynamické objekty

Zdroj: autor

6.1.1 Dlaždicový systém

Při práci s informacemi o mapách je potřeba vždy, pokud možno, pracovat pouze s daty pokrývajícími oblast mapy, kterou budeme zobrazovat uživateli. Práce s celým datovým setem by byla příliš paměťově a časově náročná. Pro nalezení těchto dat můžeme procházet celý soubor a vybrat pouze ty, co spadají do námi specifikované oblasti. Tento postup je však hlavně u větších mapových setů velice zdlouhavý a pro práci s mapou v reálném čase nepoužitelný. Pro zefektivnění vyhledávaní je nutno zavést index. Při indexování každé jednotlivé položky (cesty, bodu, oblasti) by však byl takový index příliš velký a jeho implementace a následná práce s ním by byla zbytečně náročná. Proto je vhodné mapu rozdělit na takzvané dlaždice.

Celou oblast mapy rozdělíme na jednotlivých úrovních přiblížení na stejně velké oblasti obdélníkového tvaru – dlaždice. Každá tato oblast ponese informace pouze o mapových objektech procházející právě skrze ni. Po „rozsekání“ celé mapy můžeme bázový soubor jednoduše indexovat právě podle dlaždic. Dělení dlaždic se děje po jednotlivých úrovních zoomu druhou mocninou čtyř, počínaje úrovní 1, která obsahuje 4^1 dlaždic. Toto dělení odpovídá quadstromu. Na této struktuře lze také vystavět index. Každá dlaždice dostane svoje číslo, které jasně identifikuje její pozici a úroveň zoomu ve kterém se nachází. Toto číslo se nazývá quad klíč. Ukázka dělení mapy na dlaždice je zobrazena na Obrázku 17.



Obrázek 17: Dělení mapy na dlaždice

Zdroj: http://msdn.microsoft.com/en-us/library/bb259689.aspx

Tento systém dlaždic je založen na stejném principu jako systém dlaždic u mapových podkladů OpenStreetMap v kapitole 3.2.1.

6.1.2 Mapové styly

Mapové podklady jsou vykreslovány různými barvami a čáry navíc různými tloušťkami a styly. Tyto informace musí být někde definovány, aby podle nich mohl renderer vykreslovat. Pro možnost přizpůsobit zobrazení mapy různým situacím byly zavedeny styly, které popisují všechny důležité grafické informace definující výslednou podobu mapy.

Definice stylů jsou uloženy v XML souboru. Ten začíná kořenovým elementem `styleDefinitions`, který obsahuje atributy jméno stylu, typ mapového formátu pro který je definován a verzi. Dále následují definice stylů pro jednotlivé mapové prvky. Zde je nutné vždy definovat cílový prvek a selektor. Cíl může být `way` neboli cesta – vykresluje se jako lomená čára s dalšími možnými prvky jako např. text; `area` neboli oblast – vykresluje se jako polygon; nebo `node`, který se vykresluje jako symbol či ikona. Selektor určuje konkrétní mapový prvek, např. `highway=motorway`. V další části se definují informace, které přímo ovlivňují vzhled výsledného prvku. Sem patří barvy, šířky, úrovně zoomu, na

které se tento prvek zobrazuje (*minZoom*), barva čar (*color*), barva casingu [28] (*casingColor*), případně výplně, tloušťky čar a další informace. Konkrétní strukturu a všechny použitelné datové typy jsou definovány v XML schématu *mapStyleSchema.xsd*. Příklad jednoduchého souboru definujícího styly pro dva prvky *highway=motorway* a *waterway=riverbank*.

```
<?xml version="1.0" encoding="utf-8"?>
<styleDefinitions name="Tourist style" type="osm" version="0.1">
  <style target="way" selector="highway=motorway">
    <polyline minZoom="6" color="#fff36b" casingColor="#007104" priority="50"
              width="5:1; 19:18" casing="5:2;19:24" />
  </style>
  <style target="area" selector="waterway=riverbank">
    <polygon minZoom="9" color="#6c9ac9" zIndex="1" />
  </style>
</styleDefinitions>
```

Tloušťky čar a casingu jsou definovány ve formě takzvaného zoom faktoru (atribut *width* a *casing*). Jedná se o zápis, který definuje, jakou tloušťku bude mít čára na jednotlivých úrovních zoomu. Například zápis *5:2;19:24* znamená, že na úrovni zoomu 5 bude mít čára tloušťku 2 body a na úrovni 19 to bude 24 bodů. Hodnoty mezi těmito úrovněmi jsou dopočítávány interpolací.

6.1.3 Renderovací vrstvy

Při renderování je důležité, aby objekty byly vykresleny ve správném pořadí a nepřekrývaly se (např. aby řeka netekla přes most). K tomu aby se dalo určit správné pořadí objektů a jejich následné vykreslení, byl zaveden třífázový systém vrstev.

První fáze dělí objekty do mapových vrstev, které říkají, zdali se jedná pouze o objekty na pozadí, infrastrukturu či aktivní objekty. Existuje celkem pět Vrstev:

- **Background** – uzavřené oblasti (pole, vodní nádrže, atd.),
- **Auxiliary background** – dodatečné objekty (vrstevnice, stínování),
- **Infrastructure** – cesty, nody,

[28] Casing je anglický výraz, který znamená ohraničení nebo kryt. V mapových stylech je výraz používán k označení barevného ohraničení cest. Toto ohraničení slouží pro zdánlivé spojení cest na stejné výškové úrovni.

- **Labels** – vykreslování textových informací,
- **Function** – aktivní objekty, většinou pohybující se.

Každá z vrstev má svoje ohodnocení, a to v celých tisících: 1000, 2000, 3000, 4000 a 5000. To jaké ohodnocení daný objekt dostane, závisí na jeho typu. Pokud se jedná o Area, Way či text, mají jasně předurčeno, do jaké vrstvy se zařadí.

Pořadí ve vrstvách lze ovlivňovat pomocí Z-Indexu, což je druhá fáze určení pořadí. Z-Index je definován ve stylech a určuje pořadí objektů v rámci vrstvy tak, jak jsou ve skutečnosti povrchově uspořádány. Příkladem může být umístění mostu vedoucím nad silnící, či vedení vysokého napětí nad všemi ostatními cestami. Ohodnocení Z-Indexu se přidává jako číslo v celých stovkách k předchozímu ohodnocení vrstev: 1100, 2500, 5600… Bylo stanoveno 11 úrovní Z-Indexu, od -4 do 5. Defaultní hodnotou je 0. Při hodnocení vrstev se toto číslování převádí tak, že hodnota -4 je ohodnocena jako 100, hodnota -3 jako 200, atd. Tento převod je z důvodů možnosti zadávat do souboru stylů uživatelsky příjemnější hodnoty.

Poslední fází je určení vykreslovací priority. Ta slouží k dodatečnému určení, která cesta se bude kreslit přes jinou, ve stejné úrovni Z-Indexu. Ve výsledku tak lze jemně doladit výsledný vzhled mapy tím, že se určí vyšší priorita dálnici a nižší priorita vedlejší silnici, a tím se zajistí, aby se vždy dálnice kreslila výše. Ohodnocení priority se děje v rozmezí od 1 do 99. Priorita 0 je rezervována pro casing, což je vrstva, ve které se vykresluje orámování cest. To je vždy vykresleno s prioritou 0, aby bylo zajištěno, že různé typy silnic z různých směrů budou mít jednotné a spojité orámování. Výsledné ohodnocení pak může vypadat takto: 3505, 3510, 3678…

Pokud objekt projde ohodnocovacím procesem, je zařazen do `RenderCollection`, což je třída založená na třídě `List<T>`, tedy generickém seznamu. Před samotným vykreslováním je pak na tento seznam aplikováno řazení QuickSort, které seřadí objekty připravené k vykreslení podle jejich ohodnocení. Když se pak bude vykreslovat jeden objekt za druhým, budou se postupně skládat na sebe přesně tak, jak bylo definováno.

Na Obrázku 18 je vidět, jak je vykresleno pozadí (domy a trávnaté oblasti) a nad ním infrastruktura. Silnice jsou všechny ve stejné výškové úrovni (mají stejný Z-Index), ale žlutá, důležitější silnice, má vyšší prioritu, proto je kreslena vždy přes ostatní. Černá tlustá

čára pak představuje tramvajové koleje a její Z-Index je vyšší než Z-index okolních objektů. Orámování neboli casing všech cest na stejné úrovni Z-indexu vždy navazuje a vytváří tak celistvý a upravený dojem celé mapy.

Obrázek 18: Vykreslování pomocí renderovacích vrstev

Zdroj: autor

6.1.4 Algoritmy pro Mercatorovo promítaní

Při práci s Mercatorovým promítáním je nutné mít na paměti, že čím blíže se dostáváme k pólům, tím rychleji rostou vzdálenosti k nekonečnu. Se zemskou šířkou tedy nelze počítat až do plných 90° ale je nutné určit hranice. Pro výpočty v aplikaci Lishka byly zvoleny hranice jako konstanty -85,05112878 a 85,05112878. Při výpočtech se pak vždy na hodnoty zeměpisné šířky aplikuje ořezávací funkce, jejímž výstupem je hodnota v spadající mezi hranice. Tato funkce se dá jednoduše zapsat v jazyce C# pomocí třídy `Math` (parametry `minValue` a `maxValue` jsou stanovené hranice):

```
return Math.Min(Math.Max(n, minValue), maxValue);
```

Následující rovnice vycházejí z obecných vztahů pro Mercatorovo promítání a jsou převzaty ze stránek MSDN [10]. Při převodu zemské délky a šířky v souřadném systému WGS84 na obrazovkové koordináty v pixelech se vychází z dlaždicového systému popsaného v kapitole 6.1.1. Ve výpočtech se tedy vždy musí určit výsledná velikost projekční plochy, která je závislá na aktuální úrovni zoomu. Ta se vypočítá jako:

$$mapSize = 256 \cdot levelOfZoom^2$$

Samotný přepočet ze stupňů na pixely je pak podle rovnic:

$$sinLatitude = \sin\left(latitude \cdot \frac{\pi}{180}\right)$$

$$pixelX = \left(\frac{longitude + 180}{360}\right) \cdot mapSize$$

$$pixelY = \frac{\frac{0{,}5 - \log(1 + sinLatitude)}{1 - sinLatitude}}{4\pi} \cdot mapSize$$

Při opačném postupu, tedy určení zeměpisné délky a šířky ze známých souřadnic na projekční ploše, se použijí tyto rovnice:

$$longitude = 360 \cdot \frac{pixelX}{mapSize} - 0{,}5$$

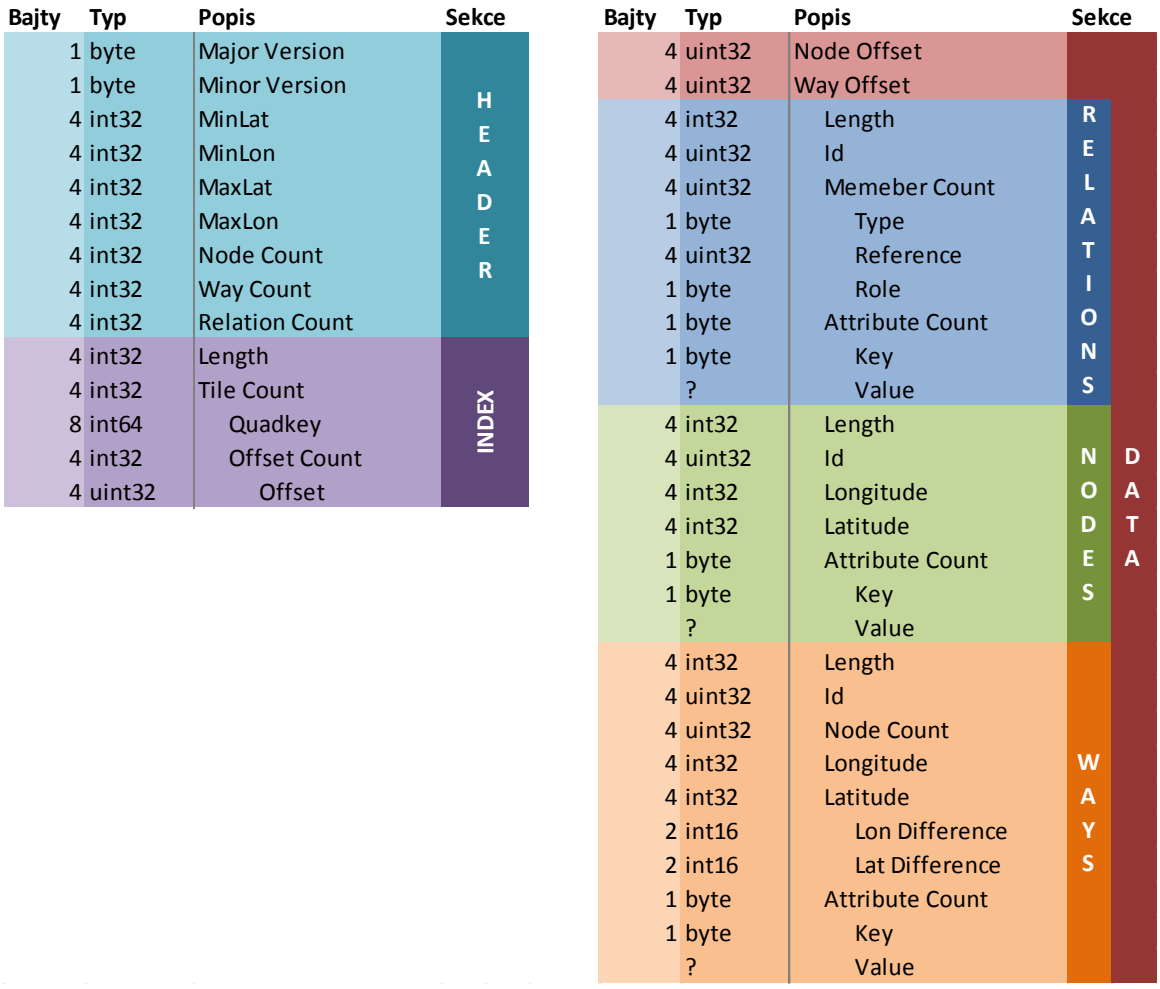
$$latitude = 90 - 360 \cdot \frac{\tan^{-1}\left(e^{-\left(0{,}5 - \frac{pixelY}{mapSize}\right) \cdot 2\pi}\right)}{\pi}$$

6.2 Binární formát OpenStreetMap

Hlavním mapovým formát používaným v aplikaci Lishka je vektorový formát OpenStreetMap. Tento formát je distribuován jako XML a může dosahovat velikosti několika GB. Jeho zpracování je tudíž paměťově a výpočetně velmi náročné. Oba tyto aspekty jsou při vykreslování map důležité, proto je mnohem výhodnější použití binárního formátu. Existuje několik popsaných binárních formátu pro OSM. Žádný z nich však nevyhovuje požadavkům na rychlé čtení, nízkou paměťovou náročnost a snadnou implementaci.

Návrh binárního formátu vychází z dlaždicového systému. Konkrétní implementace dlaždicového systému závisí od konkrétního pluginu, který s tímto systémem pracuje. V našem případě je dlaždicový systém použit v pluginu Lishka.Osm pro práci s vektorovým mapovým formátem OpenStreeMap. Velikost dlaždice je 256×256 bodu. Na první úrovni zoomu je tedy velikost celé mapové projekce 512×512 bodu; na úrovni 18 pak již 67 108 864× 67 108 864 bodů.

Při návrhu vyvstala otázka, jak ukládat data v indexu a v bázovém souboru. První možností bylo ukládat data v bázovém souboru do oddílů podle jednotlivých dlaždic a v indexu mít pouze odkazy na tyto oddíly. Při načítání dat pro vybraný mapový výřez pak stačí vybrat z indexu dlaždice a načíst jejich data. Tím, že ukládáme data v každé dlaždici, dochází k redundanci, která vzniká, pokud například delší silnice vede přes více než jednu dlaždici. V takovém případě musí být informace o ní uloženy ve všech dlaždicích, kterými prochází. Ve výsledku tím vzniká neúměrně velký bázový soubor.



Obrázek 19: Binární formát OSM

Zdroj: autor

Druhou možností, méně paměťově náročnou, je ukládání dat v bázovém souboru netříděně, pouze s jednoduchým dělením po sekcích pro cesty, nody a relace. Celý dlaždicový systém je poté uložen pouze v indexu, který obsahuje dělení podle quad klíčů a odkazy do bázového souboru na všechny mapové prvky obsažené v dané dlaždici. Redundance dat

sice zůstává, ale opakují se nám pouze offsety, které jsou podstatně menší než celá definice mapových prvků.

Struktura binárního souboru je zobrazena na Obrázku 19. Na začátku souboru se nachází hlavička obsahující základní informace o mapě. Její velikost je 30 bajtů. Následuje index, který začíná celkovou velikostí indexu. Po něm je uveden počet indexovaných dlaždic. Další část indexu je definice jednotlivých dlaždic. V nich je na začátku každého opakujícího se bloku uveden quad klíč a počet offsetů v bloku následovaný seznamem konkrétních offsetů odkazujících do datové části souboru. Datová část je dělena na sekce obsahující relace, nody a cesty. Na začátku datové části jsou uvedeny offsety začátku těchto sekcí. Podle nich lze určit nejen, kde začíná jaká sekce, ale i typ mapového prvku, na nějž získáme odkaz z indexu. Díky tomu nemusíme v indexu držet informaci o typu prvku. Určení typu mapového prvku je důležité při jeho převodu z binárního tvaru.

6.2.1 Převod do binárního formátu

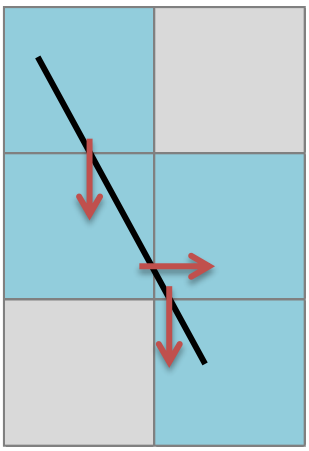
Před samotným zpracováním vektorových dat získaných z databáze OpenStreetMap je nutné tyto převést do binárního formátu. Při převodu z XML souboru je prvním problém právě samotný XML, který obsahuje informace v textové podobě, přičemž mnoho z nich není při vykreslování potřeba. Zpracování takového množství dat je náročné. Například soubor obsahující data pro celou Českou republiku má více než 2 GB.

Prvním krokem je parsování XML souboru. Pro výběr vhodného postupu byla vytvořena analýza, ze které vyplývá, že pro zpracování objemnějších souborů (řádově GB) je nutné v jazyce C# použít třídu `XStreamElement`. Při tomto postupu se nenačítá celý strom XML do paměti a lze tak procházet teoreticky nekonečně dlouhý soubor, viz kapitola 4.5.

Po zpracování XML souboru a převedení informací na příslušné datové struktury `OsmNode` a `OsmWay` je nutné vytvořit index a zapsat data do binárního souboru. Zápis do souboru probíhá ve dvou fázích. Nejprve se zpracují nody a cesty, které se ukládají do dočasného souboru. Toto se děje z důvodu, že na začátku zpracování neznáme velikost výsledného indexu ani jeho obsah, který je tvořen quad klíči a offsety jednotlivých mapových prvků, které zjistíme právě až při jejich zápisu do dočasného souboru. Během zápisu dat do dočasného souboru se v paměti postupně vytváří index. Hotový index je v druhé fázi

zapsán do finálního souboru a za něj je překopírován obsah dočasného souboru, který je po té smazán. Výsledkem je jeden kompletní soubor včetně hlavičky, indexu a dat.

Při tvorbě indexu je nejsložitějším procesem správné zařazení jednotlivých mapových objektů (nodu, cest a ploch) do odpovídajících dlaždic na všech požadovaných úrovních zoomu. U zpracování nodu je situace jednoduchá, jelikož se jedná pouze o bod, a zařadí se do všech dlaždic, ve kterých se vyskytuje. U cest už to tak jednoduché není. Pokud je cesta tvořena z delších rovných úseků nebo se nachází ve vyšších úrovních zoomu je velice pravděpodobné, že tyto rovné úseky prochází přes více než jednu dlaždici. Do indexu nelze tudíž zapsat, že daná cesta se vyskytuje pouze v dlaždicích, kde se nalézají koncové body rovných úseků, ale také ve všech dlaždicích, které protíná. Zde je nutné aplikovat algoritmus, který zjistí, kterými všemi dlaždicemi daná úsečka prochází. Důraz je přitom kladen na jednoduchost a hlavně na rychlost algoritmu. Využívá se toho, že víme, že všechny dlaždice jsou čtvercové a stejně velké. Algoritmus na začátku určí směr, v jakém se budou hledat průsečíky. Směr se určuje od počátečního bodu ke koncovému jako vlevo-dolu, vlevo-nahoru, vpravo-dolu nebo vpravo-nahoru. V určeném směru se pak hledá průsečík úsečky (právě zpracovávaný segment cesty či plochy) se stěnou dlaždice, ve které se právě algoritmus nachází. Pokud je průsečík nalezen, posune se algoritmus na další dlaždici ve směru průsečíku, viz Obrázek 20.

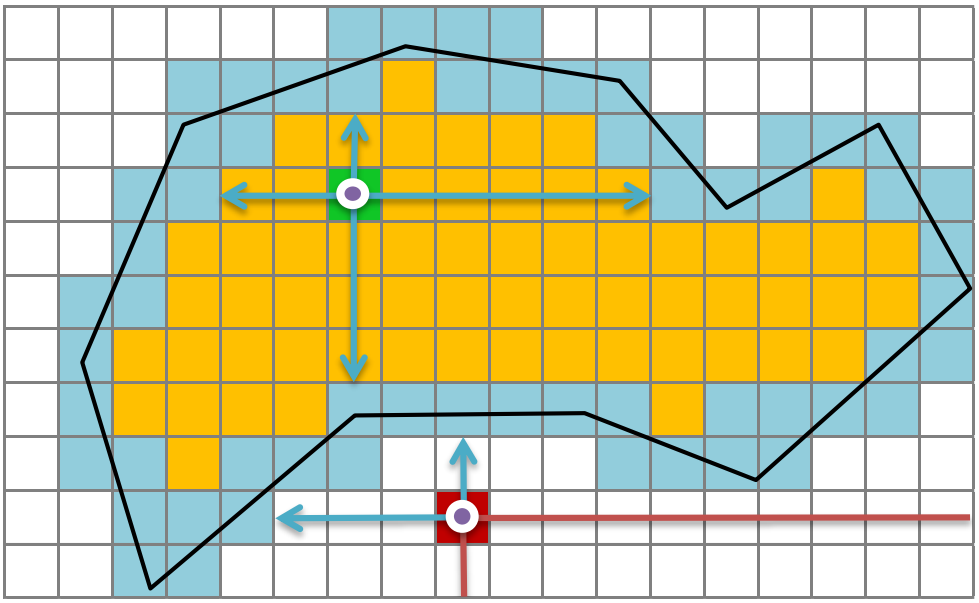


Obrázek 20: Posun indexovacího algoritmu při hledání protínajících dlaždic

Zdroj: autor

Takto se postupuje, dokud algoritmus nedojde do koncového bodu. Všechny dlaždice, kterými v průběhu algoritmus projde, jsou uloženy do indexu. Tento postup se opakuje pro

všechny segmenty cesty ve všech požadovaných úrovních zoomu (ve kterých úrovních se daný mapový objekt nachází je určeno v mapových stylech).



Obrázek 21: Výplň dlaždic uvnitř rozsáhlých oblastí

Zdroj: autor

Při zpracování uzavřených ploch je navíc nutné, kromě dlaždic, kterými procházejí hraniční segmenty určit i hranice, které tvoří výplň oblasti a žádný ze segmentů jimi neprochází. Příklad takové oblastí je zobrazen na Obrázku 21, kde modrou barvou jsou zvýrazněny dlaždice, kterými procházejí hraniční segmenty a oranžově pak dlaždice tvořící výplň. K určení, zdali dlaždice jsou či nejsou výplní uzavřené oblasti, slouží jednoduchý algoritmus. Na začátku se určí obdélník tvořící hranice prostoru, který se bude zkoumat. Všechny dlaždice v tomto obdélníku se budou postupně procházet a testovat. Jelikož dlaždice, přes které procházejí hraniční segmenty, jsou určeny předchozím algoritmem, inicializujeme tyto v poli, reprezentující všechny dlaždice ve vybraném obdélníku, na hodnotu `True`. Nyní u každé testované dlaždice zjišťujeme, jestli z každé její strany lze dojít k některé dlaždici do plochy. Pokud ano, je tato dlaždice zahrnuta také do plochy a přidána do indexu. Na Obrázek 21 je takový případ znázorněn zelenou dlaždicí a červenou dlaždicí naopak případ, kdy jako vnitřní dlaždice plochy není zahrnuta. Dlaždice je vyřazena, pokud se stane, že na minimálně jedné její straně nelze nalézt již označenou dlaždici. V takovém případě je v algoritmu rovnou přeskočeno na další dlaždici. Tímto

způsobem se prověří všechny uzavřené oblasti a zajistí se tak, že při vykreslování větších ploch nebudou v mapách vznikat „díry“.

Místo uvedených algoritmů by mohly být použity již existující algoritmy pro rasterizaci úseček a vyplňování oblastí. Většina existujících algoritmů je však příliš komplexní s důrazem na přesnost výpočtu. Použité algoritmy jsou naopak navrhovány tak, aby byly jednoduché a jejich provádění co nejrychlejší. Pokud při zpracování dojde k zaindexování některých dlaždic navíc, znamená to pouze větší velikost indexu, a samotné vykreslování není ovlivněno. Použití jiných algoritmů je otázkou budoucí optimalizace.

7 Návrh a implementace systému

Tato kapitola se věnuje samotné implementaci navrhovaného systému. V částech věnujících se jednotlivým softwarovým aplikacím je popsána navržená struktura s detaily implementace. Důležité funkce a části, na které jsou aplikace děleny, jsou zde dále nastíněny a vybrané důležité funkce podrobněji popsány. Pro lepší představu o implementovaných řešeních jsou uváděny ukázky zdrojových kódů, včetně jejich stručného popisu. Závěr kapitoly se věnuje popisu a konfiguraci navrženého hardwarovému řešení.

7.1 Klientská část

Klientská část systému je grafická aplikace, jejíž hlavním úkolem je vykreslování mapových podkladů různých formátů a do nich následně dokreslovaní dalších statických či dynamických objektů. Aplikace je snadno rozšiřitelná pomocí pluginovacího podsystému, který jednotlivým rozšířením zpřístupňuje funkce jádra pomoci služeb API.

7.1.1 Jádro

Jádro aplikace je obsaženo v knihovně `Lishka.Core` a jeho základem je optimalizovaný renderovací engine a pluginovací engine. V jádře je také implementován jednoduchý databázový engine pro ukládání nastavení. Pro přístup do databáze a pro správu dotazů slouží třída `DatabaseManager`.

Pro vykreslování statických mapových podkladů slouží třída `TileRenderer` a pro práci s dynamickými objekty, jako jsou pohybující se automobily či za běhu vkládané body zájmu, pak slouží třída `DynamicRenderer`. Tyto a ostatní pomocné třídy pro vykreslování se nachází ve jmenném prostoru `Lishka.Core.Render`. Přístup k těmto třídám je zprostředkován pomocí veřejných služeb API popsaných dále.

Nejdůležitější částí pluginovacího enginu je správce rozšíření `ExtensionManager` a API sestávající se ze služeb, které jádro zpřístupňuje ostatním pluginům a přes které s ním poté komunikují. Správce rozšíření se stará o inicializaci a následně taky o řádné ukončení všech registrovaných pluginů. Implementuje rozhraní `IExtensionManager`.

```
public interface IExtensionManager
{
    List<IExtension> Extensions { get; set; }
    void InitializeExtensions();
    void UnloadExtensions();
}
```

Díky tomu, že jsou všechna důležitá rozhraní uložena v knihovně *Lishka.Contracts*, nemusí žádná z ostatních částí aplikaci mít přímou referenci na jádro, a tím vzniká mnohem volnější vazba mezi jednotlivými moduly.

7.1.2 Lishka API

Hlavním požadavkem na API je, aby zpřístupňovalo všechny potřebné funkce pro ostatní části aplikace. Každá služba obsažená v API musí implementovat rozhraní, přes které k nim budou ostatní třídy přistupovat. Tato rozhraní jsou přístupná nezávisle na jádře z knihovny *Lishka.Contracts*. Všechna rozhraní API pak musí implementovat rozhraní *IServise*:

```
public interface IService
{
    bool IsInitialized { get; }
    void InitializeService();
    void UnloadService();
}
```

ApplicationService

Zpřístupňuje základní systémové funkce a vlastnosti jako je instance hlavního okna aplikace, pracovní adresář aplikace či metodu pro ukončení aplikace.

```
public interface IApplicationService : IService
{
    IMainWindow MainWindow { get; }
    void Shutdown();
    bool IsFullScreen { get; set; }
    string ApplicationRoot { get; }
}
```

DatabaseService

Databázový engine implementovaný v jádře je přístupný přes službu DatabaseService a slouží zejména k ukládání nastavení týkajících se aplikace. K dispozici jsou metody pro obecné parametry a pro práci z nedávno otevřenými soubory.

```
public interface IDatabaseService : IService
{
    void AddOption(string key, string value);
    void AddRecentFile(IRecentFile file);
    string GetOption(string key);
    List<IRecentFile> GetRecentFiles();
    List<IRecentFile> GetRecentFiles(int numberOfRecent);
    IRecentFile GetRecentFile(string fileName);
}
```

DialogService

Zpřístupňuje metody pro použití základních dialogových oken ErrorDialog a FileDialog.

```
public interface IDialogService : IService
{
    void ShowErrorDialog(string text);
    void ShowErrorDialog(string title, string text);
    bool OpenFileDialog(string filter, out string fileName, out int filterIndex);
    bool SaveFileDialog(string filter, out string fileName, out int filterIndex);
}
```

FileService

Pro ostatní části aplikace zpřístupňuje metody pro otevření a zavření souboru. Obsahuje také seznam handlerů *`List<IFileFormatHandler> formatHandlers`* pro všechny registrované formáty souborů. Při otevírání souboru se použije patřičný handler, který bude vědět, jak s daným typem souboru pracovat. Každý otevíraný soubor může také dále otevírat další podsoubory; např. projekt může dále otevřít přiřazenou mapu nebo soubor stylů. Soubor, který se otevře jako první, je nastaven jako hlavní. Při otevření a zavření hlavního souboru se přes *`DatabaseService`* uloží informace o posledním otevřeném souboru. Při inicializaci se do hlavního menu registrují podmenu Recent Files a Open File.

```
public interface IFileService : IService
{
    void OpenFile(string filePath, string format);
    void OpenFileWithDialog();
    void CloseAll();
    void CloseParent();
}
```

Formát handler

Aplikace Lishka v základní verzi neumí otevírat a pracovat s žádným formátem souborů. Nabízí však pluginům možnost rozšířit podporu pro nové formáty pomocí funkcí API. Existuje rozhraní *`IFileFomatHandler`*, které předepisuje, jaké metody a vlastnosti musí implementovat třída pro práci se souborovým formátem. Tyto třídy musí být definovány v rámci pluginu a jsou importovány do kolekce ve službě *`FileService`*. V této třídě bude definováno, jak se bude s daným formátem zacházet. Např. v rozšíření *`Lishka.Osm`* je definována třída *`OsmFormatHandler`*, která obstarává práci s mapovým formátem OSM.

```
public interface IFileFormatHandler
{
    string FormatName { get; set; }
    string[] FormatExtensions { get; set; }
    string CurrentlyOpened { get; set; }
    Action<string> Open { get; set; }
    Action Close { get; set; }
    bool IsMapFormat { get; set; }
}
```

Podle rozhraní *`IFileFormatHandler`* musí každá třída implementovat jméno souboru, pole obsahující všechny známé přípony formátu, cestu k právě otevřenému souboru daného typu, delegát pro otevření souboru, jehož argumentem je cesta k souboru, delegát pro zavření souboru a vlastnost *`IsMapFormat`* pro určení, zdali je formát souboru mapovým formátem. Při otevírání souboru se vždy projde seznam zaregistrovaných handlerů a hledá se takový, který by odpovídal typu souboru. Pokud je takový nalezen volá se delegát *`Open`* k otevření souboru. Jakmile je soubor otevřen, další činnost je již řízena z pluginu, který ví, jak dále s daty v souboru pracovat.

LoggingService

Díky *LoggingService* lze jednoduše logovat události v aplikaci. Logovat lze události typu Debug, Error, Fatal, Info a Warning. Po importu služby LoggingService pak stačí jen volat potřebné metody: *LogDebug(object)*, *LogError(object)*, *LogWarnign(object)*, *LogFatal(object)* nebo *LogInfo(object)*.

MenuService

MenuService slouží k registraci položek do systémového menu aplikace. Registrování položek může probíhat přímo do hlavního menu nebo jako podmenu do již existujících. Každá registrovaná položka menu je definována příkazem který obsahuje delegát, který se má vykonat při spuštění, hlavičku a definici určující pozici registrace mezi existujícími položkami. Každý takový příkaz musí implementovat rozhraní *IMenuCommand*.

```
public interface IMenuService : IService
{
    IMenuItem RegisterMenuItem(MenuType menuType, IMenuCommand menuCommand);
    IMenuItem RegisterSubMenuItem(string parentItem, IMenuCommand menuCommand);
    IMenuItem RegisterSubMenuItem(IMenuItem parentItem, IMenuCommand menuCommand);
    void UnregisterMenuItem(IMenuItem menuItem);
}
```

RenderService

Nejrozsáhlejší a nejdůležitější službou API je RenderService. Kromě metod pro vykreslování mapových objektů zpřístupňuje také vlastnosti uchovávající aktuální stav o úrovni zoomu, středu mapy, pozici myši na mapě a také aktuální projekční oblast.

Celkem pět událostí informuje o změnách týkajících se aktuálně zobrazené mapy. Jsou zde události vyvolané tlačítkem myši a také událost vyvolaná při změně projekční oblasti. Projekční oblast je výřez v Mercatorově projekci viditelný v aplikačním okně. Hodnoty určující tuto oblast jsou závislé na aktuální úrovni zoomu a centru mapy, proto každá změna těchto hodnot vyvolá událost *ProjectionAreaChanged*.

Metody *RenderObject*, *RenderDynamicObject* a jejich přetížení jsou převáděny na příslušná volání tříd *TileRender* a *DynamicRender*.

```
public interface IRenderService : IService
{
    event ProjectionAreaChangedEventHandler ProjectionAreaChanged;
    event MapMouseButtonChangedEventHandler MapLeftMouseButtonDown;
    event MapMouseButtonChangedEventHandler MapLeftMouseButtonUp;
    event MapMouseButtonChangedEventHandler MapRightMouseButtonDown;
    event MapMouseButtonChangedEventHandler MapRightMouseButtonUp;

    byte ZoomLevel{get;set;}
    Rect ProjectionArea { get;}
    Wgs84 MapCenter{get;set;}
    Wgs84 CurrentMapMousePosition {get;}

    void RenderOpen (double left, double top, int sizeX, int sizeY);
    void RenderOpen(double left, double top, int sizeX, int sizeY, long tileKey);
    void RenderClose();
    void RenderObject(uint id, List<Wgs84> nodes, StyleSheet style, string text);
    void RenderObject(uint id, List<Wgs84> nodes, StyleSheet style);
    void RenderObject(uint id, Wgs84 coordinates, StyleSheet style);
    void RenderObject(uint id, Wgs84 coordinates, StyleSheet style, string text);
    void RenderDynamicObject(uint id,Wgs84 coordinates,StyleSheet style, string text);
    void RenderDynamicObject(IDynamicMapObject dynamicObject);
}
```

StatusBarService

Souží pro registraci položek do stavového řádku aplikace. Dovoluje zobrazovat textové, obrazové a další informace podle zvoleného typu *`StatusBarItemType`*.

7.1.3 Knihovna Lishka.Library

Tato knihovna obsahuje obecné třídy, které používají různé části systému, a není závislá na žádných dalších knihovnách projektu. Pokud existuje nějaká třída, struktura či výčet, které využívá více než jeden modul aplikace, jsou tyto umístěný právě v *`Lishka.Library`*.

Knihovna obsahuje události, výjimky, třídy reprezentující geometrické prvky, pomocné třídy, třídy pro práci s mapovým souřadným systémem a třídy pro práci s mapovými styly.

7.1.4 Knihovna Lishka.Contracts

V této knihovně jsou uložena všechna důležitá rozhraní používaná v klientské a serverové části. Jsou zde definice pro příkazy, API služby, grafické prvky a ve jmenném prostoru *Lishka.Contracts.Communication* také datové entity a definice služeb pro WCF.

7.1.5 Rozšíření Lishka.Osm

Toto rozšíření přidává do aplikace podporu binárního formátu OSM. Třída *OsmFormatHandler* představuje implementaci rozhraní *IFormatHandler*, která se registruje do služby *FileService* a umožňuje otevírat soubory binárního formátu. Při otevření se načte hlavička souboru a tzv. částečný index, který bude uložen v paměti, dokud se soubor opět nezavře. Tento částečný index obsahuje odkazy na plný index v souboru. Je to kompromis mezi rychlým vyhledáváním a paměťovou náročností. V případě, že již v databázi existuje záznam o dřívějším otevření tohoto souboru, jsou tyto informace použity k nastavení počáteční projekční oblasti. O to se automaticky stará služba *FileService*. Pokud však takový záznam neexistuje, jsou použity informace získané z hlavičky souboru. Prostřednictvím služby RenderService je zaregistrován delegát, který reaguje na změnu projekční oblasti a podle potřeby posílá nová data do rendereru, kde dochází k překreslení mapy. Ve třídě *MapFeatures* jsou uvedeny všechny existující OSM tagy, definující jednotlivé mapové prvky a k nim defaultní úroveň zoomu, od které se zobrazují na mapě.

7.1.6 Rozšíření Lishka.Osm.Converter

Data z databáze OpenStreetMap jsou distribuována ve formátu XML, který je před použitím v *Lishka.Osm* nutné převést na binární formát. O převod mezi XML a binárním formátem se stará rozšíření *Lishka.Osm.Converter*. Viz kapitola 6.2.1.

Rozšíření se registruje do aplikačního menu, odkud je možné otevřít dialog pro konverzi formátu. O čtení XML souboru se stará třída *OsmXmlParser* a samotnou konverzi poté provádí třída *Converter*. Pro uložení do finálního formátu je využita třída *BinaryFile* z knihovny *Lishka.Osm*.

7.1.7 Rozšíření Lishka.Ipc

Toto rozšíření zajišťuje spojení mezi klientskou a serverovou částí. Hlavní logika se nachází ve třídě *Client*. Tato třída obsahuje dvě proxy pro synchronní a asynchronní volání. Asynchronní volání jsou použita k registraci do vzdálených událostí na serveru a přenosu dat při jejím vyvolání. Synchronní potom pro klasická volaní typu dotaz-odpověď. Ostatní rozšíření mohou tyto proxy využívat prostřednictvím služby *IpcService*. Služba *IpcService* není součásti API a pro její použití je nutná reference na knihovnu Lishka.Ipc.

7.1.8 Rozšíření Lishka.Project

Toto rozšíření přidává možnost používat v aplikaci projekty. Projekt představuje časový rozsah, ve kterém jsou sledovány pohyby dynamických objektů a s nimi spojené události, styl jakým budou tyto objekty vykreslovány. Projekt tedy definuje časový rozsah, soubor obsahující mapové podklady, soubor definující styl zobrazení mapy, hranice mapy, se kterými se bude pracovat a seznam registrovaných dynamických objektů které se budou vykreslovat zároveň s mapovými podklady. Seznam dynamických objektů je veden jako celé číslo odpovídající jejich ID v databázi serveru.

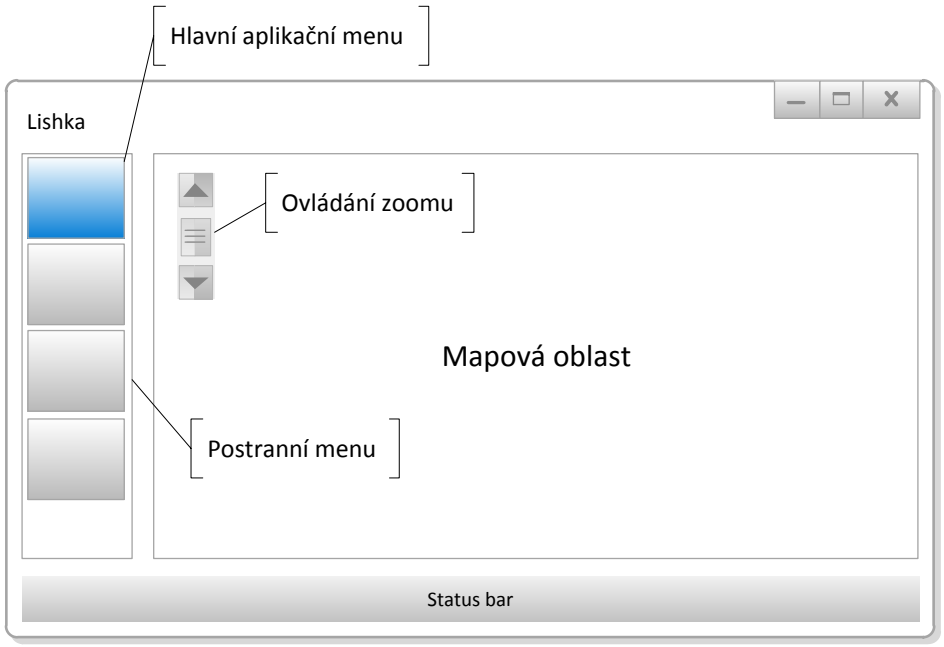
Informace definující projekt jsou uloženy v XML souboru. Tento způsob byl zvolen kvůli snadné distribuci mezi více klientskými aplikacemi. Samotný XML soubor je velice jednoduchý:

```
<?xml version="1.0" encoding="utf-8"?>
<project version="1">
  <properties>
    <beginTime></beginTime>
    <endTime></endTime>
    <mapFile>Maps\polabiny.osmb</mapFile>
    <mapBounds minLat="0" minLon="0" maxLat="0" maxLon="0" />
    <styleFile>Style\touristStyle.xml</styleFile>
  </properties>
  <items>
    <registredItems>1,2,3</registredItems>
    <disabledItems>1</disabledItems>
  </items>
</project>
```

Rozšíření obsahuje také handler pro manipulaci se souborem projektu, který se registruje mezi ostatní handlery ve službě `FileService`. Třída `ProjectManager` funguje jako XML parser obsahující metody, jejichž výstupem je instance třídy `Project`. Pro vykreslování komplexnějších dynamických objektů existuje přetížená metoda `RenderDynamicObject` služby `RenderService`, která jako parametr přijímá třídy implementující rozhraní `IDynamicMapObject`. Toto rozhraní definuje metodu `Draw(Graphics, Func<Wgs84, PointF>)`, která obstarává komplexní vykreslování. Knihovna `Lishka.Project` obsahuje implementace rozhraní `IDynamicMapObject`, které se použijí pro vykreslování objektu při změně jejich polohy. Informace o změně polohy jsou získávána ze služby IpcServis, kterou importuje třída `ProjectExtansion`.

7.1.9 Uživatelské rozhraní

Uživatelské rozhraní bylo dle požadavků navrženo tak, aby bylo jednoduché, intuitivní a dovolovalo dotykové ovládání. Největší část zabírá mapová oblast, což je místo, kde se budou renderovat mapy a uživatelské objekty. V mapové oblasti se nachází ovládání zoomu. Zoomovat lze po jednotlivých úrovních pomocí tlačítek na horní a dolní straně ovládání nebo tažením posuvníků. Pokud je k dispozici myš, lze mapu přibližovat a oddalovat také kolečkem myši.



Obrázek 22: Návrh uživatelského rozhraní

Zdroj: autor

Aplikace má jedno menu na levé straně, které sdružuje většinu ovládacích prvků. Tlačítka jsou velká a umožňují pohodlné ovládání prstem. Pod prvním tlačítkem se nachází aplikační menu, které obsahuje všechny běžné položky. Jednotlivé pluginy mohou své položky registrovat sem nebo jako samostatná menu pod nová tlačítka na postranní lištu. Pro zobrazování dodatečných informací slouží stavový řádek umístněný ve spodní části okna. Graficky je návrh zobrazen na Obrázku 22.

7.2 Serverová část

Server je jednoduchá konzolová aplikace. Jedná se v podstatě o databázový engine a jednoduché API, díky němuž lze přistupovat do databáze. Funkcí serveru je uchovávat data o poloze objektů, které je díky příslušným pluginům mohou zasílat serveru a podobným způsobem také tyto informace poskytovat dalším aplikacím pro zpracování.

7.2.1 Jádro

Jádro se skládá ze tří částí. Hlavní z nich je databázový engine ve jmenném prostoru *`Lishka.BackEnd.Database`*. Databáze v současné verzi obsahuje pouze dvě tabulky (*`Coordinates`*, *`MapElements`*) a pro práci s nimi je využit ORM nástroj NHibernate. Soubory *`Coordinates.hbm.xml`* a *`MapElement.hbm.xml`* mapují jednotlivé tabulky pro NHibernate. Pro zjednodušení práce s databází je využit návrhový vzor Repository[29]. Jeho implementaci představuje generická třída *`Repository<T>`*, s rozhraním *`IRepository<T>`*.

```
public interface IRepository<T>
{
    DetachedCriteria Criteria { get; }
    T Get(object id);
    void SaveOrUpdate(T entity);
    void Delete(T entity);
    IEnumerable<T> FindAll(DetachedCriteria criteria);
    IEnumerable<T> FindAll(DetachedCriteria criteria, params Order[] orders);
    T FindOne(DetachedCriteria criteria);
    T FindFirst(DetachedCriteria criteria);
    long Count(DetachedCriteria criteria);
    bool Exists(DetachedCriteria criteria);
}
```

[29] http://martinfowler.com/eaaCatalog/repository.html

Další části jádra je API pro práci s databází. Součástí je služba DatabaseService, která umožňuje pluginům přistup k tabulkám. Obsahuje všechny potřebné metody pro získání a uložení dat.

Pluginy a služby API jsou navzájem propojené pluginovacím jádrem, stejným jako je použité v klientské části. Pluginy jsou připojeny a inicializovány ve spouštěcí třídě `Program`.

7.2.2 Rozšíření Lishka.Aprs

Toto rozšíření slouží pro příjem dat ze sítě APRS. Tracker, který přijímá a zpracovává data je připojen do sériového portu. Data jsou poté prostřednictvím sériového portu posílaná přes protokol KISS do hostitelského počítače. Protokol KISS zapouzdřuje jednotlivé rámce z protokolu Ax.25.

Celý protokol Ax.25 je poměrně komplikovaný, ale informační rámec, který je používán pro přenos informací v protokolu KISS je jednoduchý. Jeho struktura je zobrazena v Tabulce 5. Flag určuje hranice rámce. `Address` obsahuje volací znak stanice odesílající rámec a volací znak pro kterou je určen. V poli `Control` jsou uloženy kontrolní informace pro nižší vrstvu protokolu. Pole `Info` obsahuje uživatelské informace posílané v rámci. `FCS` je kontrolní součet sloužící pro ověření, zdali není rámec poškozen. [15]

Tabulka 5: Informační rámec protokolu Ax.25

Flag	Address	Control	PID	Info	FCS	Flag

Z rámců, které jsou přijímány v binárním tvaru, jsou vyextrahovány pole `Address` a `Info`. Tyto pole jsou převedeny do použitelného tvaru a uloženy do třídy `Ax25Frame`. Ve třídě `AprsExtension` se importuje služba DatabaseService a prostřednictvím jejich metod se získaná data vkládají do databáze.

Pro účely testování a spuštění bez nutnosti připojení do sítě APRS je k serverové aplikaci přiložen plugin `Lishka.Aprs.Simulation`, který simuluje získávání dat z trackeru. Tento plugin čte dříve nasbíraná data z textových souborů a přes serverové API je posílá do databáze. Díky tomu je možné spustit a otestovat aplikaci uloženou na CD, které je přílohou této práce.

7.2.3 Rozšíření Lishka.BeckEnd.Ipc

Klientská aplikace, která chce data uložená na serveru získat, se musí nějakým způsobem spojit se serverem. K tomuto spojení slouží plugin Lishka.BackEnd.Ipc. Spojení probíhá prostřednictvím WCF. Plugin nabízí možnost synchronního a asynchronního spojení. Pro připojení jsou k dispozici dva typy bindingu: Http a Named Pipe.

Při asynchronní komunikaci musí nejdříve klient zaregistrovat událost pomoci služby UpdateService. Ta je registrována v události jádra, která upozorňuje na příchod nových informací. Každá nová informace se kromě databáze uloží také do kolekce ve třídě *MessageBox*, která funguje jako buffer nových dat. Periodicky se tento buffer kontroluje a obsahuje-li nějaká nová data, jsou tyto odeslány všem zaregistrovaným klientům. Pokud tedy dynamický objekt změní např. polohu, je o tom klientská aplikace informována a může provést jeho překreslení na mapě.

7.3 Tracker

Tracker je zařízení pro určování polohy a její odesílaní prostřednictvím datové sítě k dalšímu zpracování. Na základě předchozí analýzy byla jako síť pro přenos informací o poloze vybrána APRS, viz kapitola 2.5. Na trhu existuje mnoho zařízení pracujících s tímto systémem. Po porovnání několika dostupných byla jako nejlepší varianta zvolena platforma Tracker2 od firmy Argent Data Systems. Při výběru platformy se hledělo na možnosti konfigurace, režimy, ve kterých je možné zařízení provozovat (Tracker2 může pracovat jako tracker a digipeater zároveň). Bylo také zohledněno množství vstupů a výstupů. Další výhodou Tracker2 je možnost volby pro odesílání mezi SmartBeaconing a časovými sloty.

Tato americká společnost vyrábí několik výrobků vhodných pro různé použití. Existují moduly integrující základní tracker s radiovou jednotkou VHF. K tomuto modulu stačí připojit GPS přijímač a po základní konfiguraci je připraven k provozu. Další výhodou je jeho velikost, díky které je možná jeho integrace do široké škály systémů. Ze série Tracker2 je zástupcem takového modulu model T2-301 (Obrázek 1).

Obrázek 23: Tracker2 model T2-301 [13]

Pro účely vývoje a testování byl však vybrán univerzálnější modul. Model OT2m představuje samostatnou TNC jednotku bez radiového modulu a GPS přijímače. Její výhodou je, že disponuje širokými možnostmi připojení a konfigurace. Obsahuje dva sériové porty (přístupné přes jeden konektor) pro připojení GPS přijímače, nebo počítače a jeden port pro připojení radiového zařízení. GPS přijímač může být jakýkoliv přístroj odesílající informace prostřednictvím protokolu NMEA nebo Garmin. Pokud je dané zařízeni GPS opatřené FMI[30], lze na něm také informace z trackeru zobrazovat (např. některé navigační přístroje od firmy Garmin). Do radiového portu lze připojit VHF vysílačku, podporující APRS.

Základní funkce OT2m [13]:

- **APRS tracker** – Pracuje s GPS přijímači používající buď standartní NMEA formát (věty $GPRMC, $GPGGA a $GPGGL) nebo proprietární binární protokol Garmin[31]. Kromě vysílání své vlastní pozice, může také přijímat pozice jiných objektů a zobrazovat je na displejích GPS přijímačů.
- **KISS mód** – KISS protokol definuje rozhraní mezi TNC a hostitelem, typicky PC. Tento mód umožnuje aby Tracker2 mohl být použit s aplikacemi na hostitelském počítači. KISS protokol byl navržen s důrazem na datovou jednoduchost.

[30] FMI (Garmin Fleet Management Interface) je rozhraní umožňující na navigačních přístrojích zobrazovat dodatečné informace.
[31] http://www8.garmin.com/support/commProtocol.html

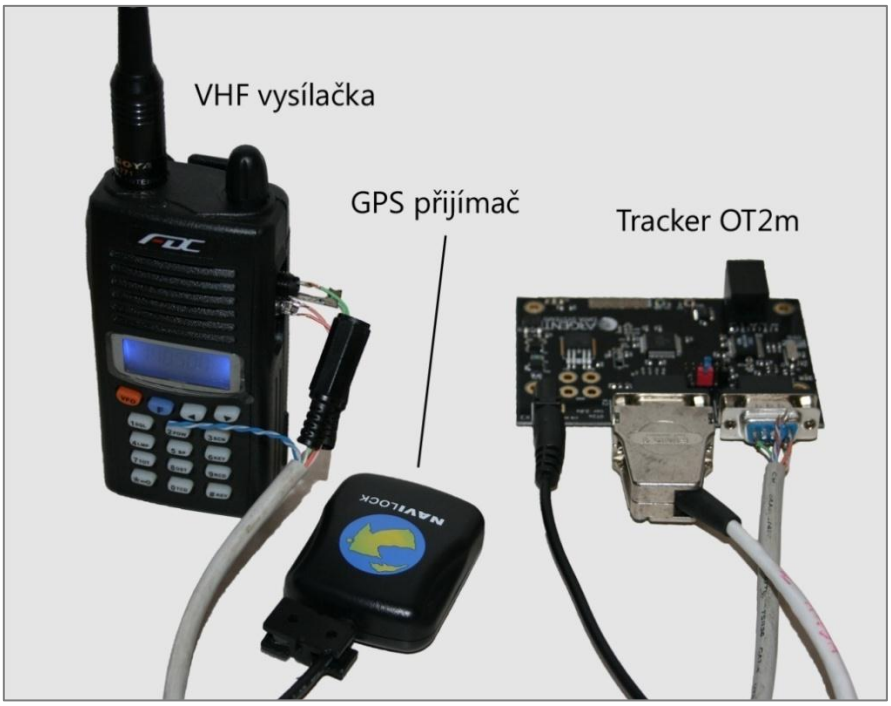
- **Digipeater** – Digipeater se chová jako simplexový digitální opakovač, který přijímá pakety a přeposílá je dál, většinou na stejném radiovém kanálu. Funkce digipeateru v Tracker2 je specifická pro použití v APRS a podporuje pokročilé funkce, jako jsou WIDEn-N, eliminace duplicit, preemptivní opakovaní a několikanásobné aliasy.
- **Meteorologická stanice** – Tracker2 může být připojen k několika modelům meteorologických stanic a poskytovat tak vzdáleně informace o počasí.
- **Příkazová konzole** – Kromě konfigurační aplikace, které je k dispozici pro operační systémy Windows, lze Tracker2 konfigurovat, zapínat a upgradovat prostřednictvím tradiční rozhraní konzole, příkazy podobnými klasickým TNC2.

7.3.1 Integrace do systému

Hlavní funkcí trackeru je sběr dat z GPS a odesílaní na server prostřednictvím APRS sítě. K tomu je potřeba vybudovat infrastrukturu pro přenos a její propojení s počítačem hostujícím softwarové aplikace. Pro testovací scénář postačí velice jednoduchá síť s několika stanicemi. Podle požadavků na provoz celého systému je jasné, že nebude využita existující národní síť, ale vybudována nová. Tato nová síť bude dočasná, nebude obsahovat žádné digipeatery a bude vybudována vždy znovu pro každou testovací akci.

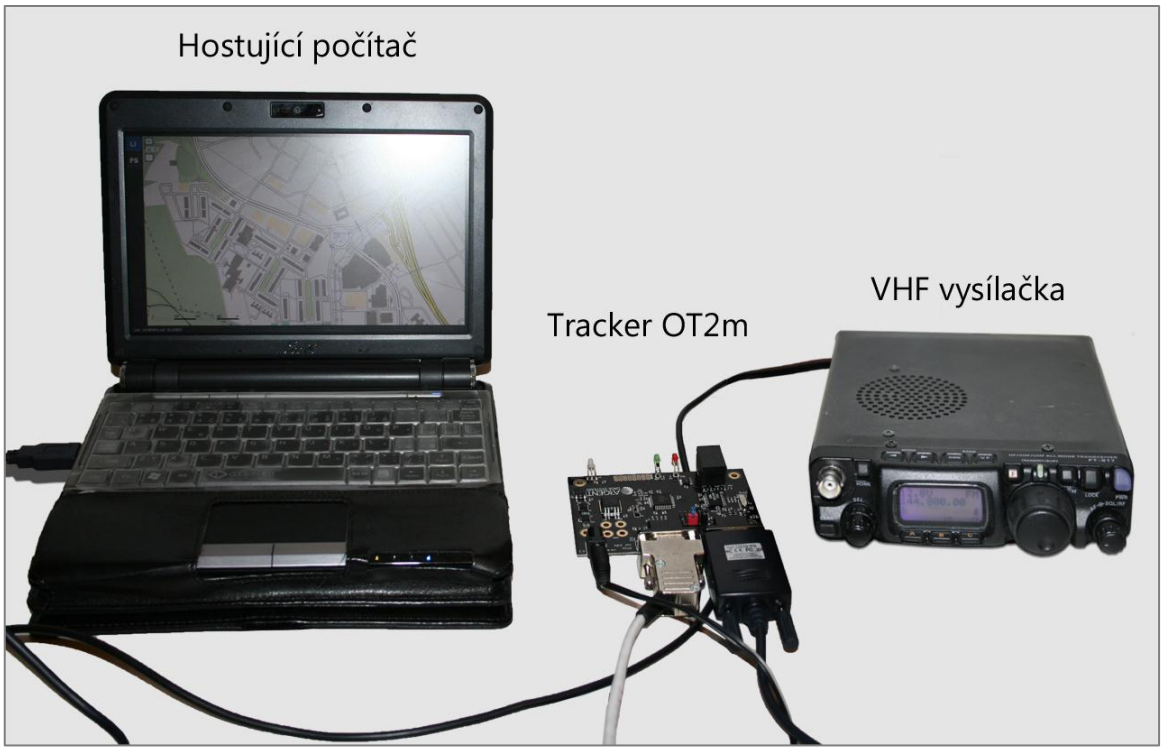
Pro testování jsou k dispozici dva trackery Tracker2 OT2m, dvě VHF vysílačky FDC FD-160A, GPS navigace Garmin Etrex Legeng, GPS přijímač Navilock 202 a notebook. Z těchto komponent jsou sestaveny dvě stanice. Jedna mobilní (Obrázek 24) a jedna stacionární (Obrázek 25). Mobilní stanice odesílá do sítě informace o svojí poloze. Stacionární stanice funguje jako přístupový bod a přijímá data od mobilních objektů. Pro zprovoznění sítě je potřeba propojit všechny komponenty, nakonfigurovat trackery a naladit vysílačky na stejnou frekvenci. Poté již obě stanice vysílají i přijímají.

Přijatá data je potřeba uložit na server. Ten běží na notebooku, který je propojen s trackerem. Pro komunikaci je použito sériové rozhraní (s redukcí USB) a protokol KISS.



Obrázek 24: Testovací sestava – mobilní stanice

Zdroj: autor

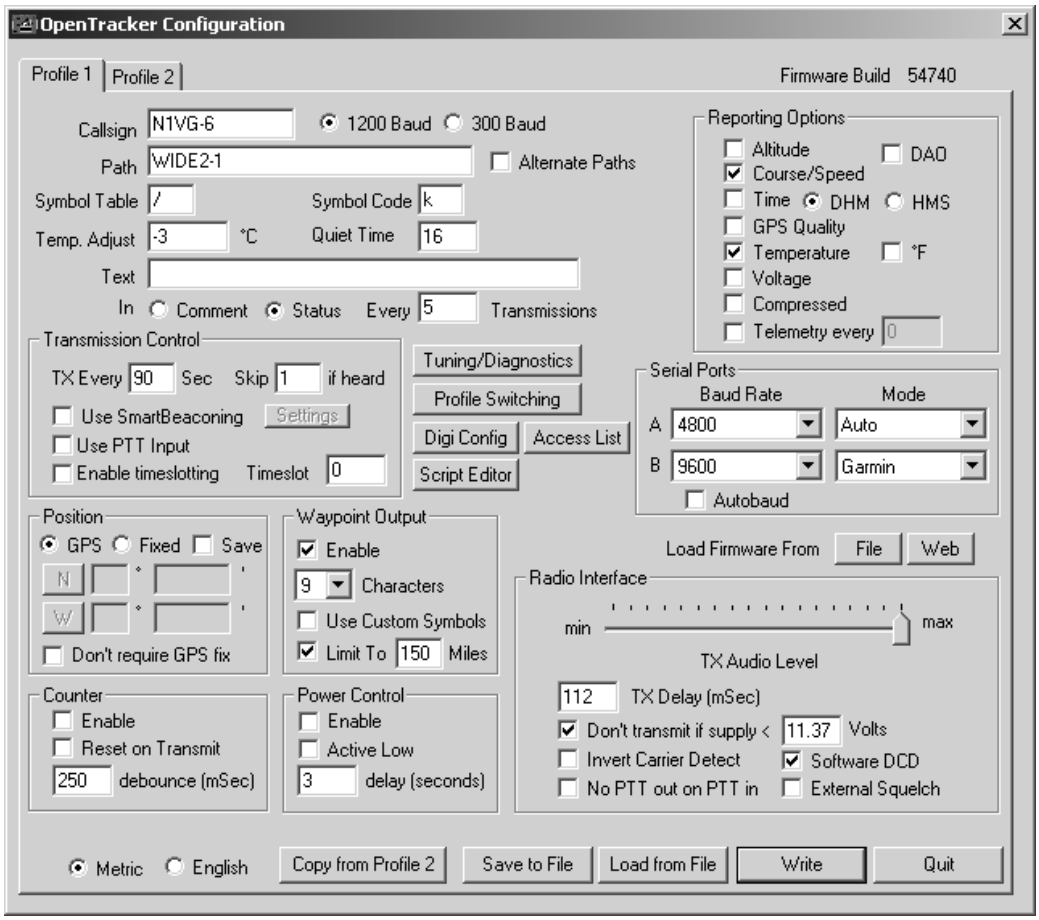


Obrázek 25: Testovací sestava – stacionární stanice

Zdroj: autor

7.3.2 Nastavení

OT2m nabízí široké možnosti nastavení, které dovolují přizpůsobení různým způsobům nasazení. Nastavení může být prováděno prostřednictvím konfiguračního programu (Obrázek 26) nebo pomocí některého z terminálových programů jako je HyperTerminal nebo Minicom. Pokud se jedná o jednoduchý scénář nasazení, jediná potřebná konfigurace je nastavení volacího znaku. Vše ostatní připraveno pro okamžité použití.



Obrázek 26: Konfigurační program pro Tracker2 [13]

Ne vždy lze však vystačit s defaultním nastavením a proto budou ta nejdůležitější podrobněji popsána [14]:

- **Call sign** – Volací znak, pod kterým se bude tracker identifikovat. Může obsahovat nejvíce šest alfanumerických znaků, případně ještě pomlčku následovanou číslem od 1 do 15.
- **Baud** – pro frekvenční pásmo VHF se vždy používá přenosová rychlost 1200 b/s.

- **Path** – Cesta je seznam volacích znaků nebo aliasů, které určují směrování paketů. Nastavení cesty má smysl u složitějších sítí, kde může směrování urychlit přenos nebo zamezit přehlcení sítě.
- **Výběr symbolu** – Tabulka symbolů a kód symbolu určují, jak se bude trakcer zobrazovat na mapě. V tabulce se uvádí označení „/“ pro primární nebo „\“ pro alternativní symbol. Kódy symbolů jsou předepsány pro různá zařízení a účely. Těmito kódy mohou být:
    - **'** – malé letadlo,
    - **-** – dům,
    - **<** – motocykl,
    - **>** – automobil,
    - **R** – obytný přívěs,
    - **[** – osoba,
    - **b** – jízdní kolo,
    - **k** – nákladní automobil,
    - **v** – dodávka.
- **Quiet Time** – určuje, jak dlouho musí být kanál volný před tím, než tracker bude vysílat. Nastavení v jednotkách odpovídajících zhruba 1/56 sekundy. Správné nastavení předchází zahlcení kanálu.
- **Text** – Slouží k doplnění dodatečného komentáře. Jakýkoliv zapsaný text se bude vysílat spolu s ostatními informacemi.
- **Reporting Options** – Nastavení v tomto bloku určují, jaké dodatečné informace se mají posílat. Vždy je vhodné volit pouze ty nejnutnější informace a udržovat co nejmenší datovou zátěž. Seznam možností:
    - **výška**,
    - **DAO** – slouží k přidání extra desetinného místa do posílaných souřadnic zemské šířky a délky za účelem zvýšení přesnosti,
    - **kurz/rychlost** – zasílaní souřadnic pozice a rychlosti,
    - **čas** – zasílaní času v podobě buď den-hodina-minuta (DHM) nebo hodina-minuta-sekunda (HMS),
    - **kvalita GPS** – hlášení kvality signálu a počet viditelných GPS satelitů,
    - **teplota** – hlášení teploty z vnitřního senzoru trackeru,
    - **napětí**,

    - **komprese** – možnost komprimovat poziční data a ušetřit až 50 % datového toku.
- **TX Every** – Nastavení doby vysílaní. Možno nastavit, jak často se budou odesílat informace z trackeru. Možné hodnoty jsou od 0 do 65 535 sekund.
- **Použití SmartBeaconing** – Umožňuje použít algoritmu SmartBeaconing™. Tento algoritmus je založen na tom, že vysílá informace o poloze v závislosti na současné rychlosti a směru pohybu. Pomáhá lépe využívat kanál a vysílat data, pouze je-li to potřeba.
- **Position** – Lze nastavit fixní pozici v případě, že se jedná o stacionární stanici bez připojeného GPS přijímače nebo získávat pozici v reálném čase z GPS.
- **Serial ports** – Nastavení přenosové rychlosti a protokolu ve kterém bude daný sériový port pracovat.

Pro testování navržené sítě bylo provedeno nastavení některých z položek tak, aby co nejlépe vyhovovaly potřebám systému. Volací znaky byly zvoleny jednoduše jako „Test1“ a „Test2“. Kód symbolu pro stacionární jednotku je „dům“ (-) a pro mobilní jednotku „osoba“ ([). Odesílání dat je prováděno každých 90 sekund, bez zapnutého algoritmu SmartBeacon. SmartBeacon se pro toto nasazení nehodí z důvodu toho, že jednotka, která by se nepohybovala, by tím pádem nevysílala informace o poloze a systém by nevěděl, zda se tato jednotka pouze nepohybuje nebo zdali je mimo dosah radiového signálu. Jediný údaje, který chceme posílat do sítě je poloha a rychlost. Žádné další údaje nejsou potřeba. Nastavení sériových portů je u stacionární stanice na rychlost 9600 kb/s a protokol je použit KISS. Jelikož zde není připojena GPS, druhý port může zůstat nastaven na auto. U mobilní jednotky je také využit pouze jeden sériový port, a to pro připojení GPS. Tento je nastaven na rychlost 9600 kb/s a výběr protokolu automaticky (v tom případě bude komunikace probíhat přes protokol NMEA).

8 Závěr

Cílem práce bylo navrhnout a poté implementovat systém pro monitorování polohy. Dle zadání byla vypracována analýza, ve které se podle požadavků sestavily případy užití, a definovala se celá struktura systému. Zde vyšlo najevo, že oproti původnímu zadání bude nutné navrhnout nový způsob bezdrátového přenosu dat. Původně plánované použití technologie GSM by nevyhovovalo budoucímu nasazení a proto bylo nahrazeno technologii APRS. S touto změnou je spojen i další bod zadání a to použitý GPS lokátor. Byl použit takový, aby vyhovoval požadavkům projektu a mohl pracovat v síti APRS. Podle analýzy byl poté navrhnut celý systém, který byl implementován v programovacím jazyku C# pro platformy .NET a Mono. Výsledkem této práce je provozuschopná softwarová aplikace, s možností převádění a vykreslování mapových podkladů a zobrazování pozic mobilních objektů, jejichž informace jsou přijímány prostřednictvím sítě APRS. Výsledná aplikace však není ve stavu vhodném k ostrému nasazení do provozu a nachází se spíše ve fázi testování.

Do budoucna se naskýtá obrovský prostor pro rozšíření funkcionality a optimalizaci stávajícího řešení. Díky plnohodnotnému pluginovacímu jádru a aplikačnímu API, představuje systém Lishka robustní platformu pro další rozšiřování. Jedním z budoucích využití, pro která je systém plánován, je vybudování datové sítě pro účely fleet managementu. Jako další rozšiřující funkce poté přidají v úvahu: zasílání zpráv mezi stanicemi, možnost přidávat dynamicky mapové objekty přes menu aplikace a jejich distribuce prostřednictvím sítě, podpora pro jiné bezdrátové sítě než APRS, podpora pro další mapové formáty (např. Garmin IMG), tvorba a export map podle šablony, nebo statistické funkce pracující s nasbíranými daty. Kromě dodatečných rozšíření se také počítá s podporou více operačních systému, zvláště pak těch mobilních (Android, Meego a Windows CE).

Přínosy této práce spočívají hlavně v získaných zkušenostech při vývoji komplexnějšího systému, při kterém bylo použito velké množství technologií z oblastí softwaru i hardwaru. Zvláště pak práce na softwarové části byla velkým přínosem. Konkrétně nahlédnutí a pochopení základních principů mapové vizualizace a s tím spojené algoritmy, principy a technologie rozšiřitelných aplikací, porovnání a práce s různými grafickými knihovnami a s tím spojené prohloubení znalostí v oblasti 2D grafiky.

9 Použitá literatura

[1] KABELOVÁ, Alena; DOSTÁLEK, Libor. *Velký průvodce protokoly TCP/IP a systémem DNS*. 3. aktualiz. vyd. Praha : Computer Press, 2002. 552 s. ISBN 80-7226-675-6.

[2] CHRISTIAN, Nagel, et al. *C# 2008 : Programujeme profesionálně*. 1. vyd. Brno : Computer Press, 2009. 2 sv. (1126, 772 s.). ISBN 978-80-251-2401-7.

[3] FURUTI, Carlos A. *Map Projections* [online]. 2008 [cit. 2010-07-19]. Dostupné z WWW: <http://www.progonos.com/furuti/MapProj/CartIndex/cartIndex.html>.

[4] Map Projections : From Spherical Earth to Flat Map. *Nationalatlas.gov* [online]. 2009, [cit. 2010-08-05]. Dostupný z WWW: <http://www.nationalatlas.gov/articles/mapping/a_projections.html>

[5] MARTÍNEK, Jan. *GPS a komunikační protokol NMEA*. AbcLinuxu.cz [online]. 2006, [cit. 2010-07-18]. Dostupný z WWW: < http://www.abclinuxu.cz/serialy/gps-a-komunikacni-protokol-nmea>.

[6] *Global Positioning System* [online]. 2010 [cit. 2010-08-05]. Dostupné z WWW: <http://www.gps.gov/>

[7] *Geocaching.cz* [online]. 2010 [cit. 2010-08-16]. WGS-84. Dostupné z WWW: <http://wiki.geocaching.cz/wiki/WGS-84>.

[8] *OpenStreetMap* [online]. 2010 [cit. 2010-07-19]. Dostupné z WWW: <http://wiki.openstreetmap.org>.

[9] *3G Americas* [online]. 2010 [cit. 2010-08-02]. GSM Technology Center. Dostupné z WWW: <http://www.3gamericas.org/index.cfm?fuseaction=page&sectionid=108>.

[10] BRUNINGA, Bob. *Automatic Packet Reporting System* [online]. 2010 [cit. 2010-07-18]. Dostupné z WWW: <http://www.aprs.org/>.

[11] APHAM, John. *JohnLapham* [online]. 2009 [cit. 2010-08-19]. How the Airplane Tracker Works. Dostupné z WWW: <http://www.johnlapham.com/Airplane/Tracker/APRS>.

[12] CHWARTZ, Joe. *Microsoft Developer Network* [online]. 2007 [cit. 2010-07-18]. Bing Maps Tile System. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/bb259689.aspx>.

[13] *Argent Data Systems* [online]. 2010 [cit. 2010-08-06]. Tracker2. Dostupné z WWW: <http://wiki.argentdata.com/index.php?title=Tracker2>.

[14] *Tracker2 OT2m manual* [online]. Argent Data Systems, 2009 [cit. 2010-08-10]. Dostupné z WWW: <http://www.argentdata.com/support/tracker2_manual.pdf>.

[15] CHEPPONIS, Mike; KARN, Phil. *AX.25 Layer 2* [online]. 1997 [cit. 2010-08-06]. KISS Protocol. Dostupné z WWW: <http://www.ax25.net/kiss.aspx>.

[16] *CodePlex* [online]. 2010 [cit. 2010-08-06]. Managed Extensibility Framework . Dostupné z WWW: <http://mef.codeplex.com/>.

[17] *Mono* [online]. 2010 [cit. 2010-08-07]. Dostupné z WWW: <http://www.mono-project.com>.

[18] *Microsoft XML Team's WebLog* [online]. 2007 [cit. 2010-08-06]. Streaming with LINQ to XML. Dostupné z WWW: <http://blogs.msdn.com/b/xmlteam/archive/2007/03/05/streaming-with-linq-to-xml-part-1.aspx>.

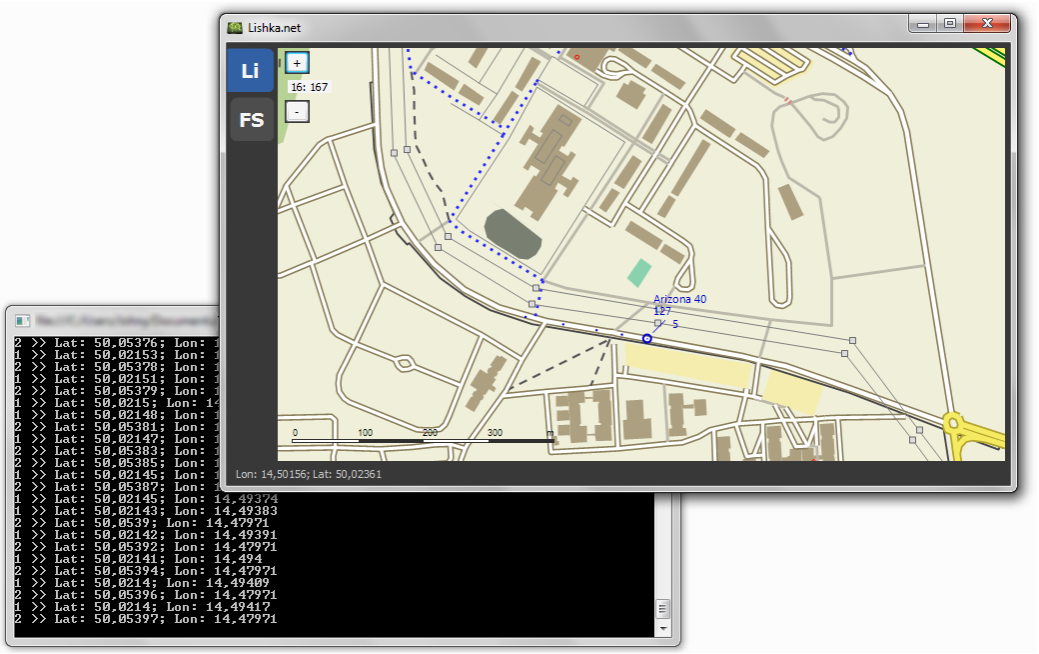
[19] GIBBONS, Glen. GPS, GLONASS, Galileo, Compass : What GNSS Race? What Competition. *Inside GNSS* [online]. 2009, [cit. 2010-08-15]. Dostupný z WWW: <http://www.insidegnss.com/node/1389>.

[20] *Barco* [online]. 2010 [cit. 2010-08-19]. RTLS Systémy (Real-Time Locating System). Dostupné z WWW: <http://www.barco.cz/?id=produkty&sel=rtls-1>.

Příloha A – Obsah přiloženého CD

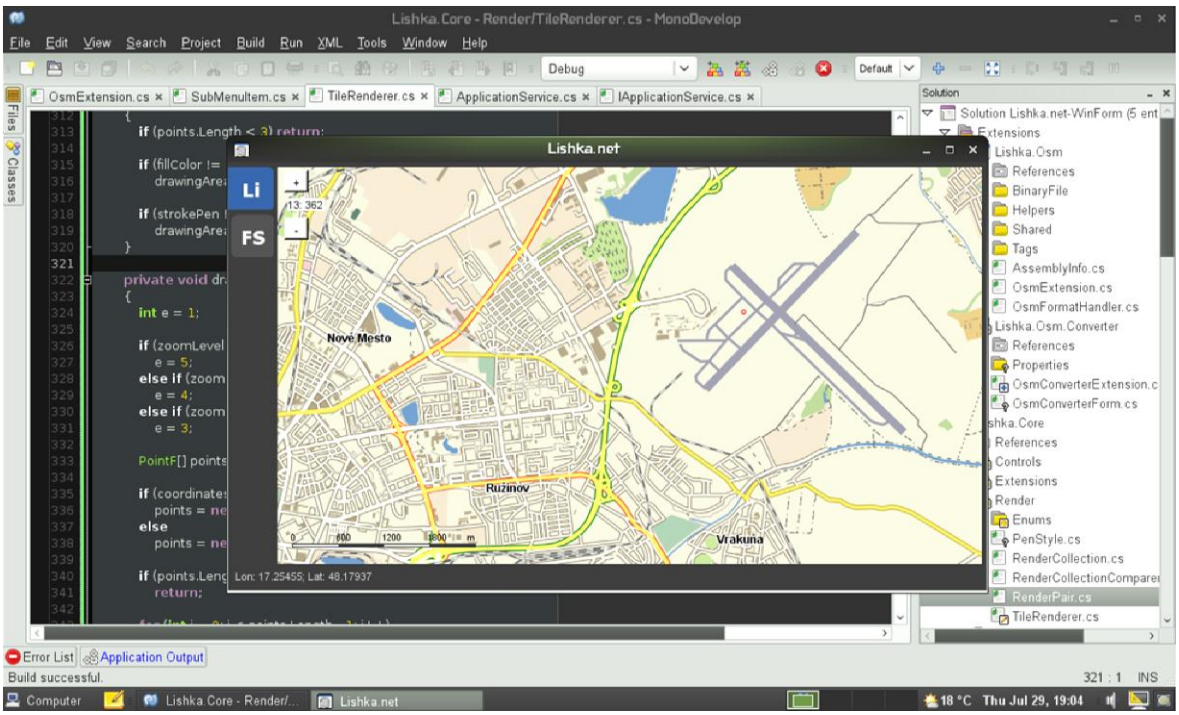
- Zdrojové kódy klientské a serverové části aplikace.
- Zkompilované aplikace, včetně ukázkových souborů map, stylů a projektů, a dat potřebných pro spuštění simulace.
- Elektronická verze diplomové práce ve formátu PDF.
- Diagramy UML vytvořené v nástroji Visual Paradigm for UML.
- Uživatelská dokumentace.

Příloha B – Ukázka aplikace



Obrázek 27: Aplikace Lishka v operačním systému Windows včetně spuštěné simulace

Zdroj: autor



Obrázek 28: Ukázka běžící aplikace v operačním systému Linux

Zdroj: autor

Příloha C – Export větší oblasti mapy