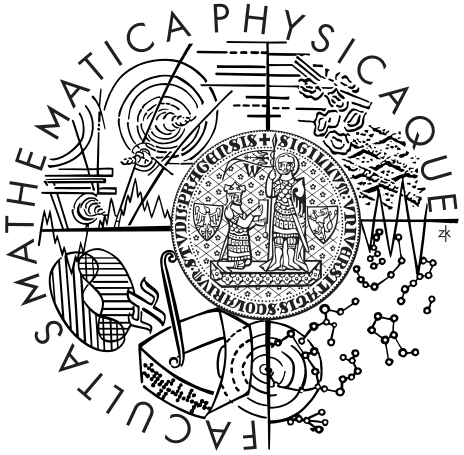
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Pavel Hryzlík

Plánovač tras

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Michal Malohlava
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2011

Zde bych rád poděkoval mému vedoucímu RNDr. Michalu Malohlavovi za jeho příkladné vedení, neocenitelné rady a nápady, bez kterých by tato práce nemohla vzniknout. Stejně tak obrovské díky patří Mgr. Vladislavovi Martínkovi za jeho nesčetné konzultace, které mi byly obrovským přínosem.

Dále bych rád poděkoval přítelkyni za její podporu a pochopení, stejně tak mé rodině.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

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

V ..... dne ..... Podpis autora

Název práce: Plánovač tras

Autor: Pavel Hryzlík

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Michal Malohlava, Katedra distribuovaných a spolehlivých systémů

Abstrakt: Systémy navigace se stávají nedílnou součástí našeho života. Ve větších městech, pokud je třeba se dopravit z místa A do místa B, je často výhodné využít kombinaci sítě veřejné hromadné dopravy a chůze. Stávající navigační systémy však tuto funkcionalitu plně nepodporují. Proto cílem této práce je navrhnout a implementovat plánovač tras založený na vyhledávací knihovně JRGPS. Výsledná aplikace Route Finder System umožňuje efektivní a uživatelsky přívětivé vyhledávání tras kombinující sítě veřejné hromadné dopravy a chůze.

Klíčová slova: plánovač, navigace, trasa, cesta, MHD

Title: Route Finder System

Author: Pavel Hryzlík

Department: Department of Distributed and Dependable Systems

Supervisor: RNDr. Michal Malohlava, Department of Distributed and Dependable Systems

Abstract: The navigation systems have become an integral part of our lives. However, contemporary solutions are not able to effectively combine navigation information based on public transport and walking. Nevertheless, in large cities such functionality is beneficial. Therefore, the objective of the thesis is to design and implement a navigation system utilizing the route find library JRGPS. The resulting application Route Finder System provides an effective and user friendly way to find routes combining public transport and walking.

Keywords: route, path, finder, navigation system, public transport

Obsah

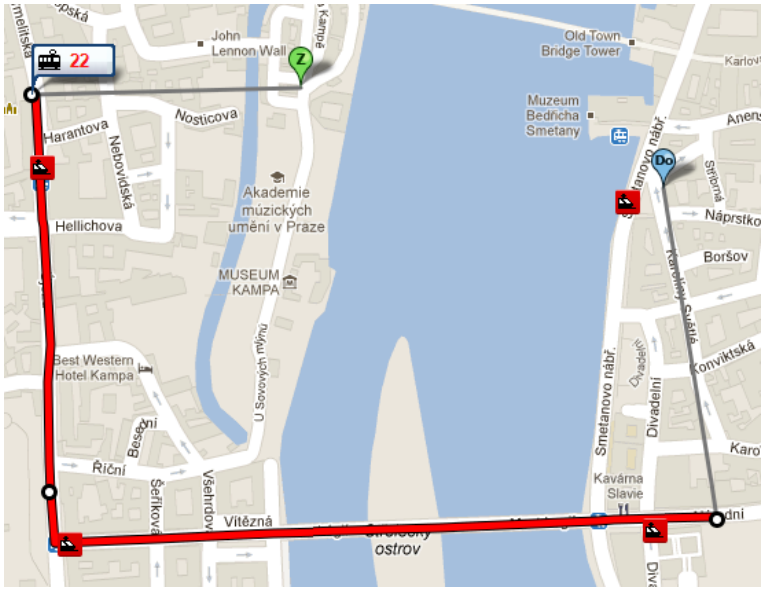
1. Úvod

Systémy navigace se stávají nedílnou součástí našeho života. V dnešní hektické době se klade čím dál větší důraz na plánování a efektivitu. Obzvláště ve velkých městech se složitou dopravní infrastrukturou.

Ve větších městech, pokud se potřebujeme dopravit z místa A do místa B, je často výhodnější využití sítě veřejné hromadné dopravy, než dopravy automobilové. Má to ale jistá omezení. Zástávky nejsou všude, je tu limitace jízdními řády. Otázka tedy zní, zda bychom si ve větším městě nemohli efektivně naplánovat cestu z místa A do místa B nejlépe kombinující pěší chůzi a sítě hromadné dopravy tak, abychom se do cíle dostali co možná nejrychleji.

Taková aplikace by měla být přístupná téměř vždy a všude. Ideálně by se tedy mělo jednat o webovou aplikaci. V prostředí českého internetu se na plánování cest pomocí sítí hromadné dopravy asi nejvíce používá aplikace IDOS. Na plánování pěších cest můžeme využít jakékoli globální mapy (Google/Seznam/Bing mapy). Efektivní kombinace však neexistuje. Jediné, co se této myšlence blíží je nová aplikace IDOS beta. Nutno však podotknout, že je určená jen pro Prahu a že kombinace s plánováním pěší chůze stále chybí. V případě, že se dostaneme z místa A do místa B rychleji pěšky tato aplikace selhává.

Například:



Obrázek 1.1: vizualizace IDOSu na mapovém podkladu - IDOS beta

Na obrázku 1.1 jasně vidíme, že když se potřebujeme dostat z Kampy na Smetanovo nábřeží, tak nás to vede velkou oklikou a přitom by stačilo pěšky přejít Karlův most a byli bychom tam.

Nicméně na Matematicko-fyzikální fakultě vznikl projekt JRGPS. Tento projekt řeší osobní navigaci pro kombinaci chůze a veřejné hromadné dopravy. Cílovou platformou jsou však zařízení PDA se systémem Windows Mobile. Řešení se tedy přímo nabízí. Využít vyhledávací knihovnu projektu JRGPS a vhodně jí uzpůsobit a případně rozšířit pro potřeby webové služby tak, aby umožnila implementaci multi-platformních klientů.

1.1 Cíl práce

Cílem práce je navrhnout a implementovat plánovač tras, který by na základě informací o městské hromadné dopravě a pěších informací, nabízel optimální trasu spojující daná místa. Výsledná aplikace by měla mít serverovou část, která by poskytovala rozhranní k vyhledávací knihovně projektu JRGPS starající se o plánování tras, a klientskou část, která by se starala o interakci s uživatelem a umožnovala vizualizovat výslednou cestu na mapovém podkladu. Jelikož knihovna projektu JRPGS je primárně určená pro PDA zařízení, očekává se, že bude upravena tak, aby ji bylo možno nasadit na server, kde by měla umožnit paralelní obsluhu více klientů. Serverová část by měla poskytovat webovou službu, kterou bude možno využít z různých platforem.

1.2 Struktura práce

V druhé kapitole jsou zadefinovány základní pojmy a vysvětleny všechny důležité použité technologie. Třetí kapitola se zabývá komplexní analýzou problému. Ve čtvrté kapitole je projekt rozebrán z pohledu programátorského, v páté z pohledu uživatelského. V šesté kapitole jsou znázorněny výsledky práce. Následuje závěr shrnující práci jako celek. Nedílnou součástí je také seznam použité literatury a příloha popisující strukturu přiloženého CD-ROM.

2. Úvod do technologií

2.1 Platforma .NET

.NET je název platformy od firmy Microsoft nesoucí si za cíl zjednodušit a sjednotit vývoj aplikací primárně určených pro systém Windows. Platforma nabízí různorodé knihovny a podporuje řadu programovacích jazyků. (viz [2])

2.1.1 Windows Communication foundation

WCF je rozhraní pro programování servisně orientovaných aplikací (SOA). Vzniklo kvůli potřebě sjednotit technologie firmy Microsoft pro tvorbu distribuovaných aplikací. Zjednodušuje jak vývoj, nasazení, tak i údržbu aplikací.

Základní programový model je WCF služba a WCF klient. Služba publikuje metody a komunikuje pomocí koncových bodů. Koncový bod slouží pro příjem a odesílání zpráv. Klient umožňuje komunikaci se službnou WCF pomocí stejně definovaných koncových bodů. (viz [2],[11])

2.1.2 Silverlight

Microsoft Silverlight je aplikační patforma na výtváření interaktivního obsahu bežícího v rámci internetových prohlížečů, nebo i ve vlastním okně mimo prohlížeč. (viz [2],[7])

2.1.3 ASP.NET

ASP.NET je platforma, která poskytuje veškeré služby nezbytné k vytváření webových aplikací. Je součástí .NET frameworku, jehož všechny funkce jsou k dispozici i v APS.NET. (viz [2])

2.1.4 Bing maps

Bing maps je internetová mapová služba, která je součástí internetového vyhledávače Bing od firmy Microsoft. Pro vývojáře je dostupné API (programové rozhraní) pro práci s Bing mapami, jednoduše zakomponovatelné do prostředí .NET. (viz [2],[8])

2.2 REST - Representational State Transfer

REST je architektura rozhraní pro distribuované architektury. Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). REST implementuje čtyři základní metody, které jsou známé pod označením CRUD, tedy vytvoření dat (Create), získání požadovaných dat (Retrieve), změnu (Update) a smazání (Delete). Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu. Pro datovou výměnu se používá několik jednoduchých standartizovaných formátů (XML, JSON). (viz [12],[13])

2.3 Knihovna JRGPS

Vyhledávací knihovna JRGPS se stará o vyhledávání nejkratších spojení mezi zadanými místy. Místem zde může být konkrétní zastávka MHD nebo pozice daná GPS souřadnicí. Spojení je realizováno prostředky MHD (nebo i jinými prostředky, podle zvolené sady dat) a pěší chůzí. Knihovna byla implementována v C++ a je primárně určená pro prostředí PDA, je to tedy single aplikace. (viz [1])

2.3.1 Mapové podklady

K naší aplikaci potřebujeme mapové podklady vzorového města a jeho jízdní řády. Pro náš případ bychom rádi zmíněná data pro město Praha. Jejich shánění ovšem stálo mnoho úsilí a výsledek přesto nebyl uspokojivý. Mapové podklady města Prahy vlastní vícero firem. Žádná z oslovených firem je však nehodlala poskytnout pro studijní účely. Jediná instituce, kde daná možnost existuje, je Český úřad zeměměřický a katastrální (ČUZK). Problém však byl, že ČUZK poskytuje pro studijní účely pouze 10 mapových listů. Jeden mapový list je cca $1km^2$. Pro nás to má ten nepříjemný důsledek, že s 10ti mapovými listy nepokryjeme více než čtvrtinu Prahy. Což je pro nás značně nedostačující.

Použita byla nakonec interní data z projektu JRGPS. Jedná se o starší verze kartografických dat a jízdních řádů hlavního města Prahy.

2.4 Ostatní použité technologie a koncepty

AJAX - zkratka znamená asynchronní JavaScript a XML. Jedná se o soubor technologií pro tvorbu interaktivních webových aplikací. (viz [2],[10])
AutoCompleteBox - je kontrolní prvek, součást rozšiřujícího balíčku Silverlight Toolkit. Nejdůležitější vlastností je, že obsahuje našeptávací funkci.
CLR - Common Language Runtime, je vrituální stroj, který je součástí .NET frameworku. Stará se správu a zpracování .NET programů. (viz [2],[10])
HTTP metody POST/GET - jedná se o základní metody protokolu HTTP. Metoda GET předává formulářová data v URL dotazu. U metody POST to je v těle dotazu.
HttpWebRequest - jedná se o třídu, která je součástí .NET Frameworku. Umožňuje uživateli pracovat se servery pomocí protokolu HTTP. (viz [2])
KD tree - je vícedimenzionální binární strom určený pro reprezentaci prostorových dat vyšších dimenzí. Organizace dat spočívá v tom, že na každé hladině stromu se porovnává jiný klíč dané dimenze. Např. pro prostorová data se bude na každé úrovni stromu porovnávat jiná souřadnice. Např. pro 2D souřadnice v hloubce 0 se bude porovnávat podle x, v hloubce 1 podle y, posléze se opakuje x atd. (viz [5])
Kontrakty ve WCF - umožňují vytváření různorodých webových služev s minimálním počtem vazeb. (viz [2])
IIS - Internetová informační služba je webový server od firmy Microsoft. (viz [2])
Managed code - Managed, neboli řízený kód je termín firmy Microsoft. Jedná se o kód, jehož zpracování řídí CLR. (viz [2],[10])
P/Invoke - neboli Platform Invocation Services, jedná se o vlastnost CLR umožňující řízenému kódu volat kód nativní. (viz [2])
Region Quad Tree - jedná se o stromovou datovou strukturu, kde každý uzel má čtyři, nebo žádné syny. Často se používá na rekurzivní rozdělení roviny na stejné menší kavdranty. To má využití např. v práci s bodovými daty. (viz [10])
SOAP - jedná se o rozšířený rotokol určený pro posílání zpráv převážně na protokolu HTTP. Umožňuje posílání zpráv pomocí XML. (viz [10])
Únik paměti - je jev, který nastane v situaci, kdy jsme v programu alokovali paměť a nejsme ji schopni dealokovat.
XAML - jedná se o značkovací jazyk určený pro tvorbu uživatelských rozhraní v aplikacích pod .NETem.
XML - standartizovaný značkovací jazyk určený pro výměnu dat.

3. Analýza úlohy

3.1 Volba implementační platformy

Jeden z prvních problémů, co bylo třeba vyřešit je volba prostředí, kde naši aplikaci budeme vyvíjet. Musíme brát ohled na to, že JRGPS knihovna, kterou budeme rozšiřovat, je naprogramovaná v C++. V tomto jazyce je naimplementována a optimalizována pro výkon. Nejefektivnější tedy bude při upravování zůstat u tohoto jazyka.

Dále bylo třeba naimplementovat webovou službu. Tato služba by měla poskytovat vyhledávací API knihovny, nejlépe tak, aby ho uživatel mohl využít napříč patformami. Určitě by k službě měl být přístup z webu, ale i z desktopových aplikací. Závěrečný krok je tvorba klienta. Klientská aplikace by měla být nejlépe aplikací webovou. Musí umět komunikovat podle protokolu služby a musí umět pracovat s mapou.

U tvorby webové služby se nám naskýtají hlavní dvě možnosti. Rozšířit knihovnu JRGPS podstatně výrazněji a udělat z ní rovnou webovou službu, nebo jí rozšířit pouze o to nutné a udělat z ní dynamickou knihovnu. Tuto knihovnu pak bude načítat zvlášť naimplementovaná webová služba. V rámci přenositelnosti, abychom upravenou JRGPS knihovnu mohli použít i v jiných aplikacích, byla zvolena možnost dynamické knihovny.

Volba prostředí padla na platformu .NET. Hlavním důvodem je to, že je potřeba vyvíjet více aplikací a tato platforma nám jako jediná umožňuje zastřešit vývoj, jak C++ knihovny, tak webové služby, tak i klientské aplikace. Pro tvorbu webových služeb je přímo pod .NETem vyvinuta technlogie Windows Communication Founfation, ke které je možno přistupovat z jakéhokoli prostředí, záleží jen na protokolu komunikace. U klientské aplikace si jako hlavní dva body klademe jednoduchou komunikaci se službou WCF a efektivní práci s mapou. Firma Microsoft ke svému internetovému vyhledávači Bing vyvinula interaktivní mapy Bing maps. Což by nebylo tak zajímavé, kdyby k nim neposkytla API pro AJAX a Silverlight. Pro naše účely je ideální Silverlight. Webovou službu i klienta budeme implementovat v jednom jazyce (C#.Net), mezi těmito aplikacemi je relativně snadná komunikace, máme podporu pro práci s mapou rovnou z kódu a to celé pod .NETem.

Omezení vyplývá z nasazení webové služby, která se omezuje na Windows Server. Toto omezení rádi akceptujeme, důležité je, že klientká aplikace může být psána v jakémkoli jazyce, pod jakoukoli platformou. Výsledkem tedy budou 3 aplikace: Přenositelná knihovna v C++, webová služba WCF a Silverlight klientská aplikace.

3.2 Rozhraní pro vyhledávání tras

V JRGPS knihovně jsme schopni vyhledávat trasy třemi způsoby:

1. pouze pešky
2. pomocí MHD
3. kombinací MHD a pěší chůze

Tyto módy jsou k dispozici hlavně kvůli tomu, že jsme někdy schopní dosáhnout cíle rychleji pěšky, někdy zase chceme cestovat pouze ze zastávky do zastávky pomocí MHD.

3.2.1 Hledání pěších cest

Ve zpracovaných datech máme k dispozici pěší graf, který obsahuje ulice, chodníky a pěšiny. Výhoda těchto dat spočívá hlavně v tom, že se nás aplikace nebude snažit navigovat po dálnicích a silnicích bez chodníků. Základní rychlost chůze je zvolena na 5 km/h. Nevýhoda je, že v mapových podkladech máme pouze dvojrozměrná data. Což má za následek, že nejdou poznat nadjezdy a různá převýšení. Nadjezd/podjezd se proto bere jako klasická křižovatka. (viz [1])

3.2.2 Hledání tras pomocí MHD

Pro vyhledávání tras MHD máme k dispozici graf městské hromadné dopravy. I během cestování pomocí MHD jsme při přestupech na jiný spoj nuceni využívat pěší chůze. Graf tedy obsahuje přidané pěší hrany. Zpravidla se jedná o hrany značící přestupy mezi jednotlivými zastávkovými ostrůvky. Z každého ostrůvku se tedy dá dostat ke všem nejbližším zastávkovým ostrůvkům. Omezení spočívá pouze v uživatelem nastavené maximální době strávenou chůzí. V grafu se nachází ještě menší množství speciálních hran, které byly zjištěny v terénu. Jedná se o hrany typu výstup z metra resp. jízda po eskalátorech apod. U takových hran totiž nelze uplatňovat základní rychlost chůze 5 km/h. (viz [1])

3.2.3 Kombinace MHD a pěší chůze

Kombinace vyhledání probíhá ve třech fázích. Nejprve se z daného vstupu najde pomocí pěší chůze množina nejbližších zastávek. Z cílové pozice se také najde množina nejbližších zastávek. Nad těmito množinami pak probíhá hledání pomocí MHD, kde se vrátí nejkratší možná kombinace z daných množin. Výsledky se pak sloučí a vydají nejkratší cestu. Toto řešení je zvoleno hlavně proto, že se typicky předpokládá, že doprava pomocí MHD je řádově rychlejší, než pěší chůze. Absolutní kombinace tras MHD a pěší chůze, kdy by se musel v každém kroku vyhledávání kombinovat pěší graf se zastávkovým, by bylo značně neefektivní. Díky vyhledávání MHD tras nad množinama startů/cílů a díky přidaným pěším hranám do grafu MHD jsou výsledky ve valné většině případů uspokojivé. (viz [1])

3.3 Rozšíření vyhledávací knihovny

3.3.1 Výhody/nevýhody JRGPS

Vyhledávací JRGPS knihovna je převážně určena pro zařízení PDA. Znamená to, že je součástí aplikace, kterou si uživatel stáhne a spustí. Knihovna tedy nemusí řešit přístup více uživatelů najednou a má jen jednu interní instanci hlavního jádra knihovny, která se inicializuje při spuštění aplikace a ukončí při jejím vypnutí. Pro naše účely je však tento způsob nedostačující. Pro potřebu webové služby potřebujeme, aby knihovna zvládala velký počet přístupů.

Základní věc, kterou musíme zkontrolovat, je interní práce se zpracovanými vstupními daty. Je důležité, aby knihovna vstupní data nezničila. To by znamenalo, že bychom při každém požadavku museli znovu načítat vstupní data. Což je velmi nežádoucí, protože načítání dat a tvorba grafů je časově náročná operace. S tím souvisí potřeba zařídit, aby si knihovna neudržovala žádné vnitřní stavy.

Dalším problémem je, že součástní projektu JRGPS byla i mapová knihovna. Tato knihovna se starala o převod vstupních GPS souřadnic na identifikátory vrcholů v peším grafu. Vyhledávací knihovně tedy pro funkce pracující s pěším grafem stačilo brát na vstupu identifikátory vrcholů v pěším grafu. Mapovou knihovnu však k dispozici nemáme, navíc je určena pro PDA. Potřebujeme tedy do knihovny dodělat funkce na převod GPS souřadnic.

Samotné veřejné rozhraní knihovny funguje tak, že se publikují vyhledávací funkce. O práci s interními datovými strukturami, volání funkcí a formátování se starala klientská aplikace. Toto je běžná praxe využívání dynamických knihoven. Práce s knihovnou je však značně složitá, budeme se tedy snažit co nejvíce operací do knihovny zakomponovat. Výsledek bude pro uživatele knihovny rychlejší a přehlednější.

3.3.2 Jádro JRPGS

Při inicializaci knihovny se vytvoří interní instance jádra knihovny obsahující i načtená a zpracovaná vstupní data. Knihovna si však v hlavní paměti udržuje pouze odkazy na data. Při požadavku na zmíněná data teprve alokuje potřebné údaje. Důležité ovšem je, že původní data neničí, zachovává je nezměněná. Tuto nainicializovanou verzi potřebujeme jako výchozí pro každý požadavek od uživatele. Stačí nám tedy pro každého uživatele vytvořit novou instanci základní verze jádra knihovny a nad ní volat knihovní funkce. Díky tomu, že vstupní data se neničí, nemusíme je v každé instanci přenášet a ušetříme tak paměť. Výsledná úprava, která umožňuje přístup více požadavkům, tedy spočívá ve třech krocích:

1. Vytvoření konstruktoru/destruktoru a hlavně kopírovacího konstruktoru hlavní třídy knihovny. Konstruktor se bude volat při úvodní inicializaci knihovny. Tato vytvořená instance se bude udržovat jako výchozí. S každým dalším požadavkem se zavolá kopírovací konstruktor na výchozí instanci.

2. Přidání odkazů na instanci jádra knihovny do všech potřebných metod knihovny. Dříve měla knihovna jen jednu interní instanci, tak všechny metody volaly jen jí. Každou metodu je třeba upravit, aby volala tu "svojí"instanci jádra.

3. Vytvoření metod pro úvodní nainicializování/deinicializování knihovny. Obsahuje také načtení a zpracování vstupních dat.

3.3.3 Nearest Neighbour Search

Problém převodu GPS souřadnic na identifikátory pěšího grafu převedeme na klasický problém hledání nejbližšího souseda. K dispozici máme vstupní GPS souřadnici a pole indexů pěšího grafu. Potřebujeme pole indexů reprezentovat tak, abychom v něm mohli rychle a snadno vyhledávat. Naše požadavky tedy jsou:

1. Vhodně reprezentovat indexy pěšího grafu.

2. Implementovat rychlý algoritmus na najití nejbližšího souseda, očekává se totiž mnoho dotazů na vyhledání.

Pro reprezentaci indexů můžeme zvolit řadu datových struktur. Pro bodová data se asi nejlépe jeví `Region Quad Tree` a `KD tree`. S přihlednutím na to, že pro algoritmus hledání nejbližšího souseda se běžněji používá reprezentace pomocí `KD tree`, byla zvolena tato možnost. Vybudování tohoto stromu z pěších dat je však časově náročná operace, tak ji budeme provádět pouze při inicializaci knihovny. V programu si pak budeme předávat pouze odkaz na vybudovaný strom. Pěší data nepotřebujeme upravovat, tedy strom zůstane konzistentní. Samotný algoritmus pro hledání nejbližšího souseda dokáže nad těmito daty hledat průměrně v logaritmickém čase, což je přesně to, co potřebujeme.(viz [5], algoritmus je popsaný v programové dokumentaci)

3.3.4 Požadavky na vyhledávací API

V základu máme k dispozici sadu vyhledávacích funkcí a řadu funkcí pomocných. Průběh vyhodnocení požadavku typicky probíhá tak, že se vytvoří datové struktury knihovny, naplní se vstupními daty, nad kterými se postupně volají pomocné i vyhledávací funkce. V následujícím seznamu je ukázáno, co za kroky je třeba naimplementovat např. pro vyhledávání nejkratší kombinující chůzi a MHD:

1. Vyhledání ke vstupním pozicím nejbližší body v pěším grafu.

2. Uložení těchto pozic do vnitřních struktur jako startovní a cílové pozice.

3. Nad danými strukturami pro startovní/cílové pozice zavolat metody pro najití nejbližší zastávky pomocí chůze.

4. Vrácená data sjednotit.

5. Nastavit dodatečné nastavení jako čas, počet přestupů apod.

6. Zavolat se nad těmito daty hlavní vyhledávací funkce (v našem případě vyhledání pomocí MHD).

7. Na vrácenými daty zavolat odpovídající formátovací funkce, výsledek naformátovat do výstupní podoby.

V původní aplikaci JRGPS se o podobné kroky starala klientská aplikace. Což skýtá vcelku netriviální požadavek na obsluhu a využití knihovny. Chtěli bychom tedy, aby práce s knihovnou byla co nejjednodušší, případně se knihovna dala snadno využít i v jiných aplikacích. Nejlépe aby zadání požadavku a vrácený výsledek byly v jednom formátu a obsahovaly všechny potřebné údaje. K tomu se hodí standartizovaný formát XML. Do vstupního XML tedy zapíšeme co požadujeme a zavoláme knihovnu. Výsledná knihovna bude kromě inicializační a deinicializační funkce publikovat pouze jednu funkci `Execute`. Tato funkce rozparsuje vstupní XML do datových struktur, zpracuje požadavek a vrátí zpět výstupní XML. Má to značnou výhodu, že se z venku vůbec nemusíme starat o režiji a funkci knihovny.(viz Programová dokumentace)

Pro samotné zpracování požadavku je vhodné jednotlivé operace oddělit a zapouzdřit. Vytvoříme tedy následující třídy:

1. Search - zapozdření hlavních vyhledávacích funkcí a pomocných operací.
2. Lists - obsluha práce s vnitřími datovými strukturami.
3. Time - práce s časem.
4. Format - naformátování výsledných dat.
5. XML - převod vstupního XML do datových struktur knihovny a zpět.
6. Exceptions - třídy dědící od třídy `Exception` přidávající nové výjimky.

Pro komunikaci s knihovnou budou k dispozici následující publikované funkce:

1. **InicializeRFS** - inicializace jádra knihovny, načtení vstupních dat, tvorba `KD tree`
2. **ReleaseRFS** - deinicializace výše zmíněného
3. **Execute** - funkce na obluhu požadavků

3.4 Webová služba

3.4.1 Komunikace s knihovnou

Načítání dynamických knihoven se pod .NETem může zdát snadné. Musí se ale jednat o `managed *.dll`. To ale není náš případ. Upravená vyhledávací C++ knihovna je pochopitelně `unmanaged`. Problém přistupování z jazyku C# k nativní knihovně se dá řešit minimálně dvěma způsoby. Pomocí P/Invoke, nebo pomocí C++/CLI wrapperu. Pomocí C++/CLI by řešení bylo komplexnější, zvolili jsme však P/Invoke. Pro naše účely totiž dostačuje.

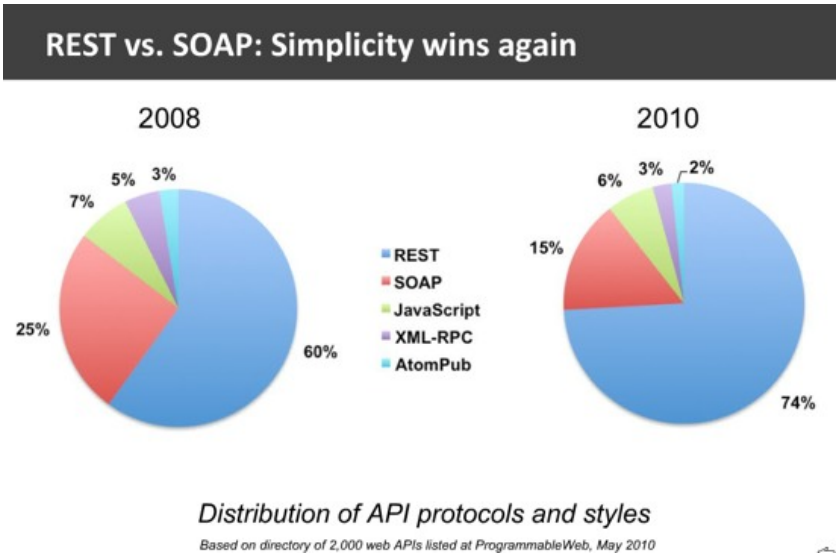
Publikované knihovní funkce musíme definovat pomocí atribudu `[dllimport]`. Definované funkce však musí být statické, přesněji `static extern`. Vytvoříme tedy pro definice knihovních metod zapouzdřující statickou třídu `PInvoke`. K této třídě pak můžeme přistupovat z jakékoli části programu.

Při startu/ukončení služby je třeba inicializovat/deinicializovat knihovnu. Ve WCF službě k tomu slouží speciální soubor `global.asax`, který poskytuje základní funkce životního cyklu služby.

3.4.2 REST - komunikace s klientem

Jedním z hlavních požadavků je, aby službu mohli využívat i webové aplikace. Základem webových klientů bývá protokol `HTTP`. Dále by bylo dobré si předávat nějaký standartizovaný formát, nejlépe XML. Pro tyto požadavky se nabízí hlavně dvě nejrozšířenější technologie: `SOAP` a `REST`.

Využití technologií za rok 2008 a 2010:



Obrázek 3.1: REST vs. SOAP (viz [9])

Výhody SOAP:

1. řada vývojových nástrojů
2. komplexnost
3. rozšiřitelnost

Výhody REST:

1. jednoduchost, snadné k naučení
2. blíže k filosofii vývoje webových aplikací

Vyvíjíme webovou službu a očekáváme webové klienty. Vhodnější pro nás tedy bude technologie REST. Je rozšířenější, modernější a jednodušší k ipmlementaci.

3.5 Webový klient - Silverlight

3.5.1 Prostředí Silverlightu

Pro vyhledávání cest potřebujeme, aby uživatel mohl nastavit buď startovní pozici pomocí GPS souřadnic, nebo jako zastávky MHD. Na získání potřebných souřadnic bude sloužit Bing mapa. Zastávky bude moc uživatel zadat do připravených vstupních polí. Bylo by však pěkné, kdy se uživateli automaticky podbízely zaskátvky, které má na mysli. Přirovnal bych to k internetovému `Google` vyhledávači, kde se uživateli automaticky nabízejí podobné výrazy. Pro prostředí .NET byl k těmto účelům vyvinutý ovládací prvek `AutoCompleteBox`. Potřebujeme však někde získat data (názvy zastávek) z kterých bude filtrovat a podbízet je. Dotazovat se s každým dotazem na zastávku webové služby se ukázalo jako ne zcela efektivní, převážně proto, že bychom museli neustále měnit zdrojové data pro výše zmíněné `AutoCompleteBoxy`. Tuto funkcionalitu tedy přenecháme na klientské aplikaci. Při spuštění klienta načteme soubor se seznamem názvů zastávek a těmito daty naplníme výše zmíňené `AutoCompleteBoxy`. Výsledek je rychlejší.

3.5.2 Vizualizace výsledků

Od služby získaná data je vhodné uživateli zobrazit dvěma způsoby. Jednak jako seznam přestupů a časů, podruhé jako grafickou křivku na mapě. Přestupy naformátujeme tak, aby uživatel přesně viděl odkud kam se má vydat, jakým typem dopravy a jak dlouho mu to bude trvat.

Mapa

Vytvoření mapy v Silverlightu díky Bing maps API není obtížné. V našem případě však nechceme, aby se uživatel mohl po mapě pohybovat volně. Je pro nás žádoucí, aby velikost mapy byla ohraničená do takové míry v jakém rozsahu máme vstupní data pro vyhledávací knihovnu. Přesněji chceme definovat velikost mapy podle datové sady pro vyhledávání. Tyto údaje by měli být konfigurovatelné. Budeme mít tedy soubor `config.xml`, který bude aplikace při svém spuštění načítat. V daném souboru budou údaje:

1. centrální bod - při spuštění bude tento bod zobrazován ve středu okna mapy
2. rozmezí zeměpisné šířky/délky - ohraniční mapy, za dané meze se uživatel nebude moct pohybovat
3. rozpětí příblížení/oddálení a počáteční přiblížení - nastavení omezení přiblížení resp. oddálení, aby se uživatel nemohl oddálit např. na plochu celé země

Všechny zmíněné body můžeme nastavit přetížením volaného konstruktoru mapového módu. Zbývá pouze ošetřit pohyb po mapě tak, aby se nedalo pohybovat za dané meze. Při každé změně zobrazení mapy se zavolá metoda `ConstrainView`. Stačí ji tedy vhodně modifikovat.

Pro nás nejdůležitější vlastnost mapy je `Map.Children`. Lze do ní přidávat grafické prvky, které chceme na mapě zobrazit.

Polylines

Díky Bing API máme ke kreslení křivek k dispozici třídy `MapPolyLine` a `Location`. První zmíněná funguje jako kolekce bodů `Location`. Rádi bychom výstupní data zobrazily jako více křivek podle typu používané dopravy. Vytvoříme si tedy nějaký kontejner, který bude typu `MapPolyLine`. Pro dané úseky vytvoříme odpovídající křivky a přidáme je do kontejneru. Na závěr pro zobrazení výsledku přidáme dané křivky do `Map.Children`.

Points of Interests

Pro přidání význačných bodů do mapy můžeme použít buď tvarově předdefinovaný prvek `PushPin`, nebo vlastně definováný prvek třídy `UIElement`. Význačné body se hodí kvůli zvýraznění např. přestupů na jiný typ dopravy, nebo také na počáteční definování startovním/cílových bodů. Pro dané prvky však neexistuje speciální třída v Bing maps API. Vytvoříme si tedy novou mapovou vrstvu (třída `MapLayer`), kam budeme vkládat naše body.

3.6 Ostatní požadavky

Autopointers

V C++ knihovně pracujeme s větším počtem naalokovaných datových struktur najednou. Zároveň ale v programu chceme odchytávat vyjímky. Aplikace v C++ nemá automatickou správu paměti, tak často může dojít situaci, že pokud vznikne vyjímka uvnitř nějakého `try` bloku, tak vznikne u naolokovaných struktur únik paměti. Tento problém se dá v C++ řešit pomocí takzvaných `autopointerů`. Pokud je ukazatel na danou strukturu `autopointer`, tak jakmile dojde k jakémukoli opuštění bloku kódu, tak se daná datová struktura zničí.

Parsování XML

Pod .NETem máme k parsování XML k dispozici vícero způsobů. Silverlight má však omezený .NET framework, nemůže tedy využít každý způsob. To samé platí i u načítání souborů. Např. nelze použít žádné běžně používané I/O proudy. Řešení spočívá v použití třídy `WebClient` a vstupní soubory asynchronně stahovat.

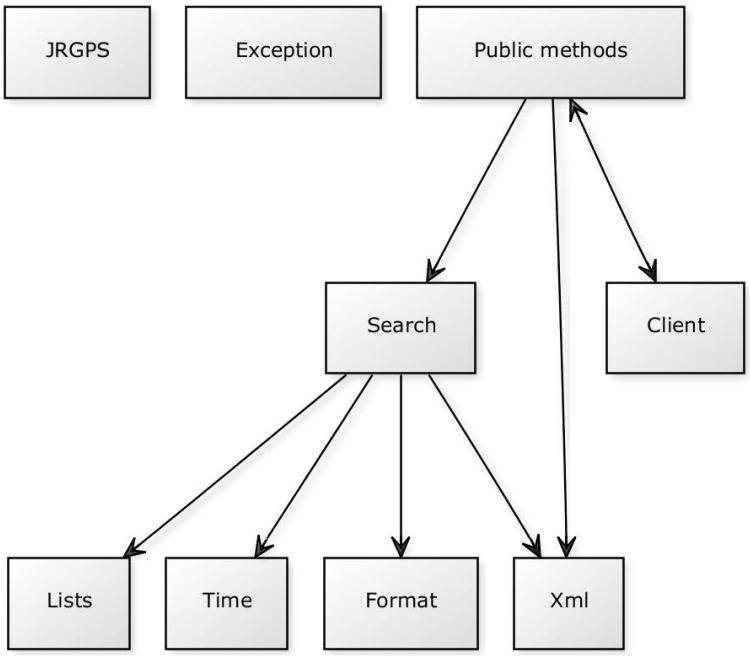
4. Programová dokumentace

Tento projekt je rozdělen do tří základních částí. Skládá se z rozšíření vyhledávací knihovny JRGPS psané v C++, z webové služby WCF a z Silverlight klienta.

Obrázek 4.1: Základní rozdělení projektu (dvojitá šipka znamená vzájemnou komunikaci)

4.1 C++ DLL

Knihovna se skládá ze dvou základních částí. Ze zdrojových souborů vyhledávací knihovny JRGPS a ze souborů plánovače tras, jež ji rozšiřují. Na obrázku 4.2) se nachází základní rozdělení hlavních částí knihovny.



Obrázek 4.2: Diagram hlavních částí knihovny (dvojitá šipka znamená vzájemnou komunikaci)

4.1.1 Knihovna JRGPS

Aby knihovna mohla fungovat, tak se musí načíst mapové/datové podklady. Typicky se zavolají inicializační data *.dat a *.wlk, které obsahují informace o zastávkách, jízdních řádech, časových tabulkách a také obsahují zastávkový resp. pěší graf.

Knihovna pracuje s vlastně definovanými datovými strukturami. Nejdůležitější je FastList.

FastList

„Zvláštní strukturou pro uchování vrcholů dosažitelných během vyhledávání je asociativní pole. Každá položka tohoto pole je seznam prvků, jejichž odhad odpovídá pozici v poli. Tyto seznamy jsou alokovány dle potřeby. Struktura si pamatuje nejmenší odhad, který byl vložen, v proměnné min.“ (viz [1])

Používá se tak, že nejprve je potřeba vytvořit prázdný list, naplnit ho seznamy vstupních dat, což jsou typicky ID, názvy zastávek, nebo vrcholy pěšího grafu. Posléze se nad těmito listy volají vyhledávací funkce. Každá knihovní funkce, pokud vrací seznam prvků, tak jej vrátí v tomto listu.

4.1.2 Komunikační rozhraní

S knihovnou se komunikuje pomocí standartizovaného formátu XML. Z prostředí .Net se knihovně doporučuje předávat datový typ `StringBuilder` a z C++ typ `BSTR`.

Příklad vstupu

```
<RequestData>
    <query>MHD</query>
        <starts>
            <start>Anděl</start>
        </starts>
        <goals>
            <goal>Hlušičkova</goal>
        </goals>
    <gpsStart>
        <lon>14.404591867</lon>
        <lat>50.088048666</lat>
    </gpsStart>
    <gpsGoal>
        <lon>14.428495714</lon>
        <lat>50.088296479</lat>
    </gpsGoal>
    <time>
        <day>8</day>
        <dayOfWeek>1</dayOfWeek>
        <hour>14</hour>
        <minute>50</minute>
        <month>11</month>
        <year>2011</year>
    </time>
    <set>
        <arrival_time>5</arrival_time>
        <changes_limit>3</changes_limit>
```

```
        <line_num>9</line_num>
        <stop_name>Anděl</stop_name>
        <wait_limit>5</wait_limit>
        <walk_limit>15</walk_limit>
    </set>
</RequestData>
```

Nejdůležitější prvek vstupu je položka `<query>`, kde se nachází jednoznačná identifikace požadavku, který se má vykonat. Musí být vždy vyplněná, jinak knihovna vrátí vyjímku `Input Error`. Knihovna přijímá 6 základních druhů požadavků:

1. Line - vyžaduje pouze vyplněnou položku `<line_num>`.
2. TimeTable - vyžaduje položky `<starts>`,`<goals>`,`<time>`,`<line_num>`.
3. Walk - vyžaduje položky `<gpsStart>`,`<gpsGoal>`,`<time>`, volitelně pak `<set>`.
4. MHD - vyžaduje položky `<starts>`,`<goals>`,`<time>`, volitelně pak `<set>`.
5. General - vyžaduje položky `<gpsStart>`,`<gpsGoal>`,`<time>`, volitelně pak `<set>`.
6. Departures - vyžaduje položky `<time>`,`<stop_name>`,`<walk_limit>`.

Příklad výstupu

```
<ResponseData>
    <Answer>MHD</Answer>
    <StartTime>14:50</StartTime>
    <EndTime>15:18</EndTime>
    <Changes>
        <Transfer>
            <StopName>Anděl</StopName>
            <LineName>pěšky</LineName>
            <LineNum></LineNum>
            <Time>10</Time>
        </Transfer>
        <Transfer>
            <StopName>Hlušičkova</StopName>
            <LineName>Tram</LineName>
            <LineNum>9</LineNum>
            <Time>28</Time>
        </Transfer>
    </Changes>
    <Points>
        <GPS>
            <Lon>14.4035396575928</Lon>
            <Lat>50.0695114135742</Lat>
        </GPS>
        <GPS>
            <Lon>14.404369354248</Lon>
```

```
            <Lat>50.0714378356934</Lat>
        </GPS>
        <GPS>
            <Lon>14.3935298919678</Lon>
            <Lat>50.0723686218262</Lat>
        </GPS>
        <GPS>
            <Lon>14.3872900009155</Lon>
            <Lat>50.0715675354004</Lat>
        </GPS>
        <GPS>
            <Lon>14.3798894882202</Lon>
            <Lat>50.0705108642578</Lat>
        </GPS>
        <GPS>
            <Lon>14.3702096939087</Lon>
            <Lat>50.0699310302734</Lat>
        </GPS>
        <GPS>
            <Lon>14.3623809814453</Lon>
            <Lat>50.0699920654297</Lat>
        </GPS>
        <GPS>
            <Lon>14.3542699813843</Lon>
            <Lat>50.0684394836426</Lat>
        </GPS>
        <GPS>
            <Lon>14.3468399047852</Lon>
            <Lat>50.0680923461914</Lat>
        </GPS>
        <GPS>
            <Lon>14.3409004211426</Lon>
            <Lat>50.06787109375</Lat>
        </GPS>
        <GPS>
            <Lon>14.3363094329834</Lon>
            <Lat>50.0676918029785</Lat>
        </GPS>
        <GPS>
            <Lon>14.3260202407837</Lon>
            <Lat>50.0660018920898</Lat>
        </GPS>
        <GPS>
            <Lon>14.316969871521</Lon>
            <Lat>50.064395904541</Lat>
        </GPS>
    </Points>
    <TimeTable></TimeTable>
```

```
    <LineInfo></LineInfo>
</ResponseData>
```

Formát výstupu je pro všechny požadavky jednotný. Požadavky `Walk`, `MHD`, `General`, `Departures` vracejí startovní, cílový čas, seznam přestupů (u Departures je to seznam nejbližších odjezdů) a seznam GPS souřadnic hledané trasy. Požadavky `TimeTable` a `Line` vrátí naformátovaný výstup v odpovídajících položkách výstupu.

4.1.3 Implementace

Veřejné funkce

```
RFS_API int InicializeRFS();
RFS_API int ReleaseRFS();
RFS_API BSTR Execute(BSTR input);
```

Funkce které knihovna publikuje jsou tři. Zavoláním funkce `InicializeRFS` se vytvoří nová instance jádra knihovny JRGPS. Dále se ze vstupních dat načte zastávkový a pěší graf do vnitřních datových struktur. Následuje vytvoření instance třídy `KD_Tree` a vybuduje se KD strom z pěšího grafu. Jedná se o časově náročné operace, které se typicky zavolají jen jednou při spuštění programu.

Funkce `RekeaseRFS` se naopak obvykle volá při ukončování programu, kdy se zavolají destruktory na instanci jádra knihovny JRGPS a třídu `KD_Tree`.

Njedůležitější publikovatelnou funkcí je `Execute`. Tato funkce pomocí kopírovacího konstruktoru vytvoří z nainicializovaného hlavního jádra knihovny (které se vytvořilo funkcí `InicializeRFS`) novou instanci. Dále inicializuje vyhledávací třídu `Search` a parsovací třídu `XMLconvert`. Vstupní XML rozparsuje do datových struktur a zavolá na ně odpovídající vyhledávací funkce. Z výsledných dat vytvoří finální XML, které vrátí.

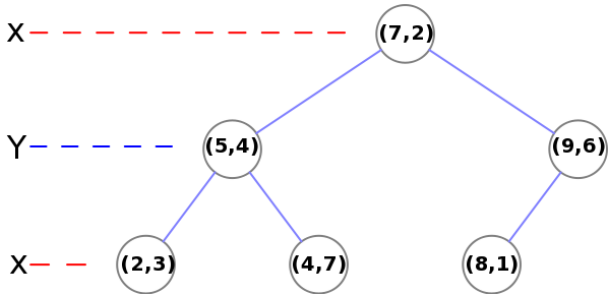
Třída KDTree

```
void build_KDtree(Core *p_core);
Point * findNearestNeighbour(KD_Node* t, Point *p);
```

Třída `KD_Tree` vznikla kvůli potřebě vyhledat nejbližší bod v pěším grafu pomocí zadané GPS souřadnice. Má dvě hlavní funkce. Vytvoření 2-dimenzionálního KD stromu z grafu pěších cest a vyhledání nejbližšího souseda k danému bodu.

Reprezentace stromu je řešená pomocí třídy `KD_Node`, která má svoje souřadnice a odkaz na levého a pravého syna. Tvorba stromu probíhá rekurzivně zatřiďováním vrcholů pěšího grafu, kdy se na každé hladině porovnává jiná souřadnice. Pokud je daná souřadnice menší, než aktuální bod ve stromě, jde se doleva, jinak doprava. Poté co se dostaneme na dno stromu, tak tam daný vrchol uložíme. Pro předpokládaná dostatečně náhodně uspořádaná data vyjde relativně vybalancovaný strom. Během celé operace musíme projít všechny prvky pěšího grafu a v každém kroku zatřídit daný vrchol do stromu. Výsledná časová složitost je tedy $\mathcal{O}(NlogN)$, kde N je počet vrcholů pěšího grafu. Jedná se o časově náročnou operaci, provádět se tedy bude jen jednou při inicializaci. (viz [4],[5])

Samotné vyhledání nejbližšího souseda spočívá v implementaci algoritmu Nearest Neighbor Search.



Obrázek 4.3: KD strom [4]

Algoritmus Nearest Neighbor Search

Kroky algoritmu: (viz [4],[5],[6])

I. Jako vstup algoritmu máme kořenový uzel a vstupní bod, ke kterému hledáme nejbližšího souseda. Začínáme s kořenovým uzlem. Spočítáme čtverec vzdálenosti vstupního bodu postupně s levým a pravým synem daného uzlu. Rekurzivně se vydáme do té části podstromu, kde je spočítaná vzdálenost menší. Čtverec vzdálenosti počítáme proto, abychom nemuseli dělat časově náročnou operaci odmocňování.

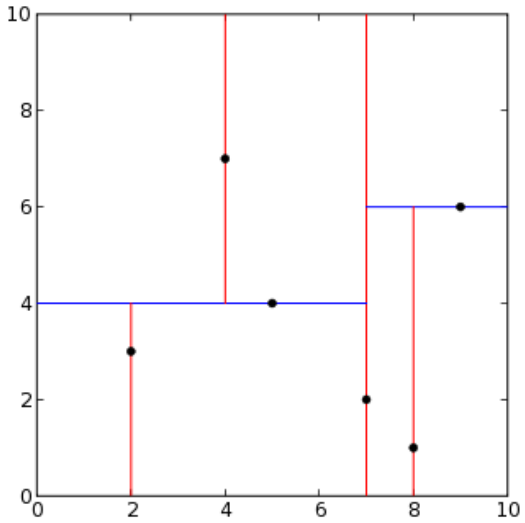
II. Když algoritmus dorazí na dno, uložíme si vrchol v listu jako aktuálně nejbližší vrchol ke vstupnímu bodu a postupně se vracíme nahoru.

III. Při vracení kotrolujeme základní dva jevy:

   1. Rodič uzlu z kterého se vracíme je bližší k vstupnímu bodu než aktuálně nejbližší vrchol. Aktualizujeme tedy nebližší vrchol rodičem.
   2. Když si jednotlivé body ve stromě představníme v rovině, tak si uvědomíme, že nám ji vlastně rozdělují na jednotlivé podoblasti. (Viz obrázek 4.4) V každé hladině vrcholy rozdělují rovinu střídavě vodorovně a svisle. Celou rovinu nám to tedy rozdělí na obdélníky. Vytvoříme si takový hraniční bod dané podoblasti (odpovídající druhému podstromu rodiče, než z kterého se vracíme), který je co možná nejblíže vstupnímu bodu, ale nepřekračuje hranice daného obdélníku. Následně pouze stačí zkontrolovat, jestli vzdálenost hraničního bodu je od vstupního bodu blíže, než aktuálně nejbližší vrchol. Pokud ano, tak to znamená, že v druhém podstromě, než z kterého se vracíme, se nachází alespoň jeden bod, který je vstupnímu bodu blíže, než aktuální nejbližší vrchol. Za tohoto předpokladu se tedy do daného podstromu vydáme a postupujeme opět od kroku I.

IV. Když algoritmus ukončí výše popsané kroky algoritmu pro kořenový uzel, tak je algoritmus u konce a vydá nejbližší možný vrchol ke vstupnímu bodu.

Za předpokladu dostatečně náhodně uspořádaných dat má algoritmus průměrnou časovou složitost $\mathcal{O}(logN)$. Což je pro náš případ ideálně rychlé a efektivní. (viz [4],[6])



Obrázek 4.4: Body grafu rozdělující rovinu na podoblasti [4]

Třída XMLconvert

```
RequestData * XMLToData(std::wstring xml);
std::wstring wDataToXML(ResponseData *responseData);
```

Třída `XMLconvert` se stará o rozparsování vstupních XML dat do datových struktur a zpět.

Třída Search

Třída `Search` obsahuje hlavní vyhledávací funkce. Typicky bývají potřebné funkce volány z veřejné metody `Execute`.

Funkce:

```
void matchName(Core *p_core);
```

- Funkce vyhledá jména zastávek, které by mohly odpovídat vstupu.

```
void searchLine(Core *p_core);
```

- Funkce nejprve vyhledá linku zadanou na vstupu a poté vrátí jakými zastávkami daná linka projíždí a kolik druhů linek se stejným označením jezdí a v jakém směru.

```
void searchTimeTable(Core *p_core);
```

- Funkce nejprve najde odpovídající ID zastávek, posléze vyhledá opovídající zastávky a pak zavolá funkci na vyhledání časové tabulky.

```
void searchPathWalk(Core *p_core, KD_Tree *kd);
```

- Funkce se stará o vyhledání nejkratší pěší cesty mezi dvěma GPS souřadnicemi.

```
void searchPaths(Core *p_core);
```

- Funkce slouží k vyhledání nejkratší cesty pomocí MHD.

```
void searchGeneralPaths(Core *p_core, KD_Tree *kd);
```

- Funkce vyhledá nejbližší zastávku jak pro startovní pozici, tak pro cílovou pozici a pak vyhledá nejkratší cestu pomocí MHD z daných zastávek.

```
void searchDepartures(Core *p_core);
```

- Funkce vyhledá nejbližší odjezdy z dané zastávky.

Třídy Lists/Time/Format

Jedná se o pomocné třídy sloužící k třídě `Search`. Třída `Lists` slouží k práci s vnitřními datovými strukturami JRGPS knihovny. Převážně operuje s výše popsanými fast-listy (vytváří, ukládá, načítá, plní). Třída `Time` je velmi prostá. Stará se o načítání vstupního času do vnitřích datových struktur. Třída `Format` parsuje výsledná data do výstupních struktur.

Výjimky

V programu vzniklá výjimka se odchytí a knihovna ji uživateli vrátí místo výstupního XML.

Seznam vyjímek:

Nothing Found - vyhodí, když knihovna pro zadaný vstup nenajde žádný výstup
Empty List - vyhodí, když se snažíme pracovat s prázdným listem (viz. Fast Lists výše)
Input Error - vyhodí se, když je chybné vstupní XML
JRGPS exception - vyhodí, když nastane chyba v kódu JRGPS knihovny. Vrátí název vnitřní chyby. V následující tabulce je uveden seznam nejdůležitějších hlášených událostí i s identifikačním číslem: (viz [1])

JRGPS výjimka	**Kód**	**Popis**
OK	0	vše je v pořádku, JRGPS výjimka se nevyhodí
NA	9	nedefinovaná hodnota proměnné pro uchování chyby
NOT_DONE	10	požadovaná operace se nezdařila
NOT_FOUND	11	hledaný prvek nenalezen
FOUND_MANY	12	hledaný prvek nalezen vícekrát
ERR_EMPTY_LIST	-2	pokud se snažím získat něco, co v dané struktuře není prohledávání prázdného seznamu
ERR_NON_EMPTY	-3	pokud čekám, že bude něco prázdné a ono není
ERR_ARG	-10	špatně zadané argumenty
ERR_ARG_NULL	-11	null reference v argumentu
ERR_ARG_FILE	-12	zadaný soubor neexistuje
ERR_ARG_RANGE	-20	hodnota mimo meze
ERR_INIT	-50	inicializace se nezdařila
ERR_FILE_NULL	-51	chyba souboru
ERR_LOOP	-100	pravděpodobné zacyklení
ERR_LOW_MEMORY	-110	nedostatek paměti
ERR_NULL	-200	neočekávaný NULL
ERR_UNKNOWN	-300	neznámá chyba

4.2 WCF služba

Služba se skládá ze čtyř základních částí. První resp. druhá část je rozhraní resp. implementace kontraktů. Třetí částí je statická třída `PInvoke` starající se o komunikace s C++.dll. Konečně poslední částí je soubor `Global.asax`, kde se definuje, jaké operace se vykonají při spouštění, nebo naopak při ukončování služby.

4.2.1 Kontrakty

Kontrakty služby

```
[ServiceContract]
public interface IRestServiceImpl
{
    [OperationContract]
    [WebInvoke(
        Method = "POST",
        UriTemplate = "/search",
        RequestFormat = WebMessageFormat.Xml,
        ResponseFormat = WebMessageFormat.Xml)]
    ResponseData Search(RequestData rData);
}
```

Kontrakt služby definuje operace, které má služba vykonat. V našem případě tu máme definovaný pouze jeden kontrakt, který komunikuje pomocí REST. Tento kontrakt přijímá metodu `HTML POST` ve formátu XML. Přijímaná data musí odpovídat třídě `RequestData`. Ty se vyhodnotí ve funkci Search, která vrací data ve třídě `ResponseData`, které se převedou do výstupní XML a odešlou zpět klientovi.

Datové kontrakty

Datové kontrakty jsou třídy reprezentující typy posílané mezi službou a klientem. Jak již bylo řečeno výše, prvky datového kontraktu musí odpovídat prvkům ve vstupním XML. Formát přijímaného XML je stejný jako u C++ knihovny s tím rozdílem, že jako jmenný prostor musí být uvedeno url `http://www.ms.mff.cuni.cz/~hryzlikp/`. (viz 4.1.2)

Příklad datového kontraktu:

```
[DataContract(Namespace = "http://www.ms.mff.cuni.cz/~hryzlikp/")]
public class RequestData
{
    [DataMember]
    public string Query { get; set; }
    [DataMember]
    public string Start { get; set; }
    [DataMember]
    public string Goal { get; set; }
```

```
    [DataMember]
    public GPS StartGPS { get; set; }
    [DataMember]
    public GPS GoalGPS { get; set; }
    [DataMember]
    public Time Time { get; set; }
    [DataMember]
    public Settings Settings { get; set; }
}
```

4.2.2 Komunikační rozhraní

PInvoke

```
[DllImport(DLL_CORE_NAME, EntryPoint = "InicializeRFS")]
public static extern Int32 InicializeRFS();

[DllImport(DLL_CORE_NAME, EntryPoint = "ReleaseRFS")]
public static extern Int32 ReleaseRFS();

[DllImport(DLL_CORE_NAME, EntryPoint = "Execute",
CharSet = CharSet.Unicode,
CallingConvention = CallingConvention.Cdecl)]
public static extern StringBuilder Execute(StringBuilder input);
```

PInvoke je statická třída sloužící ke komunikaci s knihovnou C++. Má nadefinované 3 funkce s pomocným atributem `DllImport`, který má dva hlavní parametry. První je název knihovny, které se budeme dotazovat a druhý je přesný název knihovnou publikované funkce. Typické použití je takové, že po spuštění služby se zavolá funkce InicializeRFS, která nainicializuje knihovnu. Když služba běží, je s každým požadavkem volána metoda Execute, která daný požadavek vyhodnotí. Konečně při ukončení knihovny se zavolá metoda ReleaseRFS, která zavolá destruktory na knihovní struktury.

REST

Ve službě se komunikuje pomocí technologie REST. Základem REST je protokol HTTP. Potřebujeme tedy na zprovoznění HTTP server. Tento požadavek nám splňuje naše WCF služba. Ke komunikaci se službou díky této technologii stačí pouze běžný webový prohlížeč.

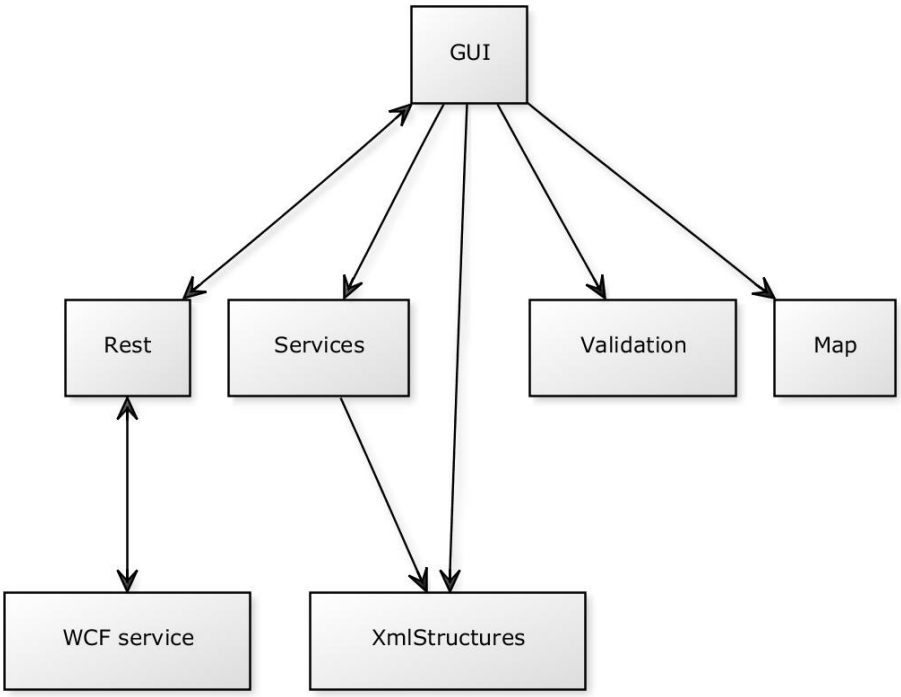
4.2.3 Global.asax

„Soubor Global.asax, označovaný také aplikační soubor technologie ASP.NET, je nepovinný soubor, který obsahuje kód pro reagování na události na úrovni aplikace a úrovni relace aktivované technologií ASP.NET nebo moduly HTTP.“ (viz [2])

V našem případě potřebujeme při startu resp. ukončení služby naalokovat resp. odalokovat C++ knihovnu. K tomuto účelu v souboru `Global.asax` přímo slouží funkce `Application_Start` a `Application_End`.

4.3 Silverlight klient

Klientská aplikace se skládá z šesti částí. (viz obrázek 4.5) Hlavním prvkem je třída `GUI` ve které je definováno uživatelské rozhraní. Toto rozhraní volá všechny ostatní části. Pro práci s mapou slouží prvek `Map`, pro komunikaci s WCF službou je tu prvek `Rest`. Načítání a parsování souborů/XML obsluhuje část `Services`. Prvek `Validation`, jak název napovídá, obsluhuje validování uživatelského vstupu. Zbývá část `XmlStructures`, kde jsou definované datové struktury odpovídající vstupnímu a výstupnímu XML pomocí kterých klient komunikuje.



Obrázek 4.5: Základní rozdělení klientské aplikace (dvojitá šipka znamená vzájemnou komunikaci)

4.3.1 GUI

Spuštění klienta

Při spuštění klienta se provede konstruktor třídy `GUI`. Nainicializují se uživatelské prvky, načte se konfigurační soubor a soubor se seznamem názvů zastávek. Dále se nainicializuje zobrazovaná mapa, třída `Draw` a třída `Validation`

Konfigurační soubor (config.xml):

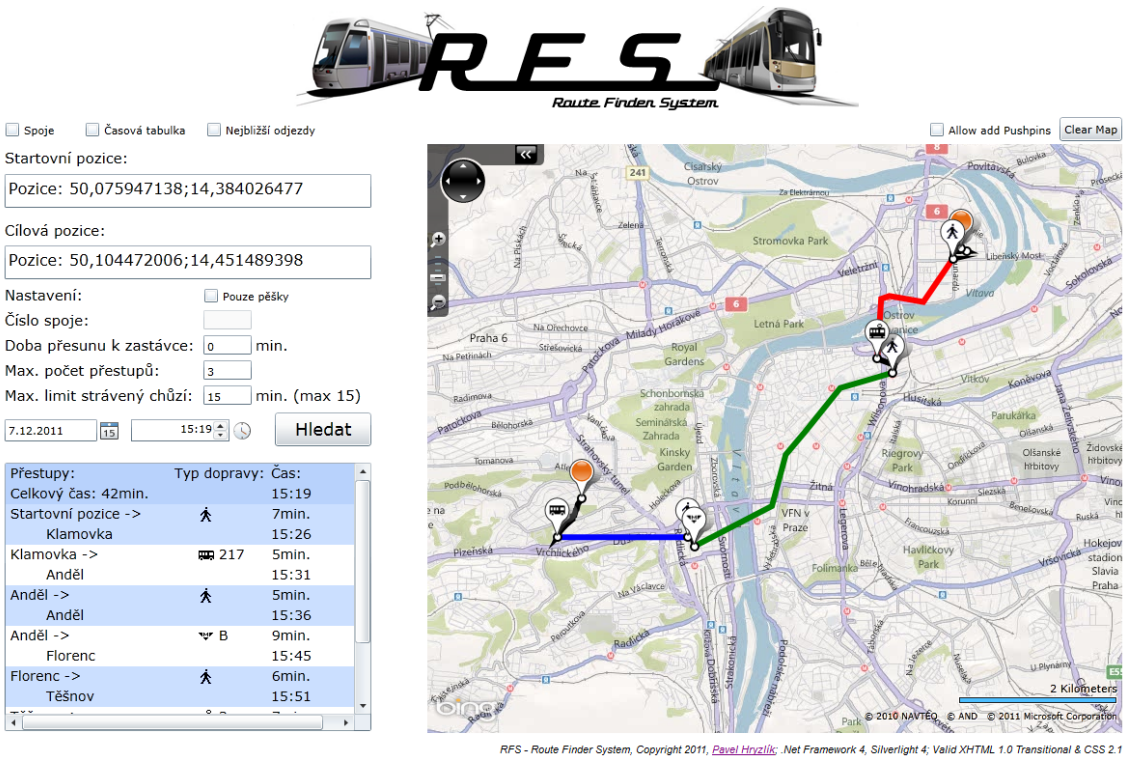
```
<?xml version="1.0"  encoding="utf-8" ?>
<config>
  <rest>
    <uri_address>
        http://localhost:59554/RestServiceImpl.svc/
    </uri_address>
```

```
    <contentType>application/xml; charset=utf-8</contentType>
  </rest>
  <map>
    <central_point>50,0506133739091|14,4058193478645</central_point>
    <latitude_range>49,98|50,15</latitude_range>
    <longitude_range>14,25|14,58</longitude_range>
    <zoom_level>12</zoom_level>
    <zoom_range>12,0|16,0</zoom_range>
  </map>
</config>
```

Konfigurační soubor obsahuje dva druhy informací. Prvním je definování adresy WCF služby a druh komunikačního média. Druhou informací je úvodní nastavení zobrazované mapy. Je to proto, abychom si mapu mohli vycentrovat pro oblast, kterou zrovna potřebujeme. Kvůli vstupním datům je úvodní konfigujrace nastavena na hlavní město Praha.

Seznam zastávek načítáme ze souboru kvůli tomu, abychom se s každým požadavkem na název zastávky nemuseli ptát WCF služby.

Uživatelské rozhraní



Obrázek 4.6: Základní rozdělení klientské aplikace

Uživatelské rozhraní je definováno pomocí jazyku XAML. Jak je vidět na obrázku 4.6, obsahuje okno s Bing mapami a ovládacími prvky určenými k interakci s uživatelem. Důležité jsou především dva vstupní `AutoCompleteBoxy`, kam se zadávají vstupní zastávky, případně GPS pozice. Uváděná Bing mapa podporuje funkce přidávání bodů do mapy a také vyčistění mapy.

Funkcionalita

Nejdůležitější funkce v třídě `GUI` jsou:

```
void PostReceived(Rest client)
private void btnSearch_Click(object sender, RoutedEventArgs e)
```

Typický průběh spočívá v základních třech krocích:

1. Uživatel nastaví v grafickém uživatelském rozhraní vstupní požadavky.
2. Klikne na tlačítko `Search`, které zavolá metodu `btnSearch_Click`, která zařídí převedení vstupních dat do XML. Poté pošle požadavek pomocí REST (metodou HTML POST) službě WCF.
3. Klient obdrží výsledné XML od WCF služby (v metodě `PostReceived`), které rozparsuje a vykreslí/vypíše do grafického rozhraní.

4.3.2 Komunikační rozhraní

```
public void PostData(string _input)
void RequestProceed(IAsyncResult asuncResult)
public void ResponceProceed(IAsyncResult asuncResult)
```

Klient komunikuje s WCF službou pomocí REST (metodou HTML POST). V projektu se o logiku přenosu požadavků stará prvek `Rest` obsahující jedinou třídu `Rest`. Samotné posílání/přijímání se provádí pomocí asynchronního volání služby WCF. Zavoláním metody `PostData` ve třídě `Rest` se nainicializuje HttpWebRequest, a poté se data asynchronně pošlou WCF službě metodou `RequestProceed`. Po odeslání dat se spustí metoda `ResponceProceed` na asynchornní přijímání. Přijmutá data se pošlou do metody `PostReceived` ve třídě `GUI`, kde se vyhodnotí. Formát posílaných/přijímaných souborů XML je stejný jako u WCF služby/C++.dll. (viz 4.1.2)

4.3.3 Práce s mapou

Prvek `Map` obsahuje důležité dvě třídy. První je třída `NewMapMode`. Druhá se nazývá `Draw`.

Třída NewMapMode

```
public NewMapMode(string latitudeRange,
string longitudeRange,string zoomRange)

protected override Range<double> GetZoomRange(Location center)

public override bool ConstrainView(Location center,
ref double zoomLevel, ref double heading, ref double pitch)
```

Třída `NewMapMode` dědí od třídy `RoadMode` z `Bing maps API`. Přetěžováním funkcí třídy `RoadMode` si definujeme vlastní mód mapy. Tato třída se inicializuje při inicializaci mapy ve třídě `GUI`. Pro náš případ uvádíme možnost nastavení vymezení mapy, rozsah přiblížení a centrální bod.

Třída Draw

```
public void Clear()

public MapLayer AddPushpin(Location point)

public MapPolyline DrawPolyline(Location startPoint,
                                List<Location> points)
```

Tato třída se stará o vykreslování do mapy. Má základní 3 funkce:

1. Metoda `Clear`, jak název napovídá, vyčistí mapu.
2. Funkce `AddPushpin` umí přidávat do mapy význačné body.
3. Konečně metoda `DrawPolyLine` vykreslí vstupní seznam bodů jako trasu do mapy.

4.3.4 Pomocné prvky

Services

Nejdůležitější třídou prvku `Services` je třída `XMLParser`. Tato třída obsahuje metody na načítání a parsování XML dokumentů.

Validation

Prvek `Validation` obsahuje 2 třídy. První je třída `Validation`, která kontroluje správnost vyplněných polí v grafické uživatelském rozhraní. Druhou třídou je třída `Errors`, která obsahuje chybová hlášení.

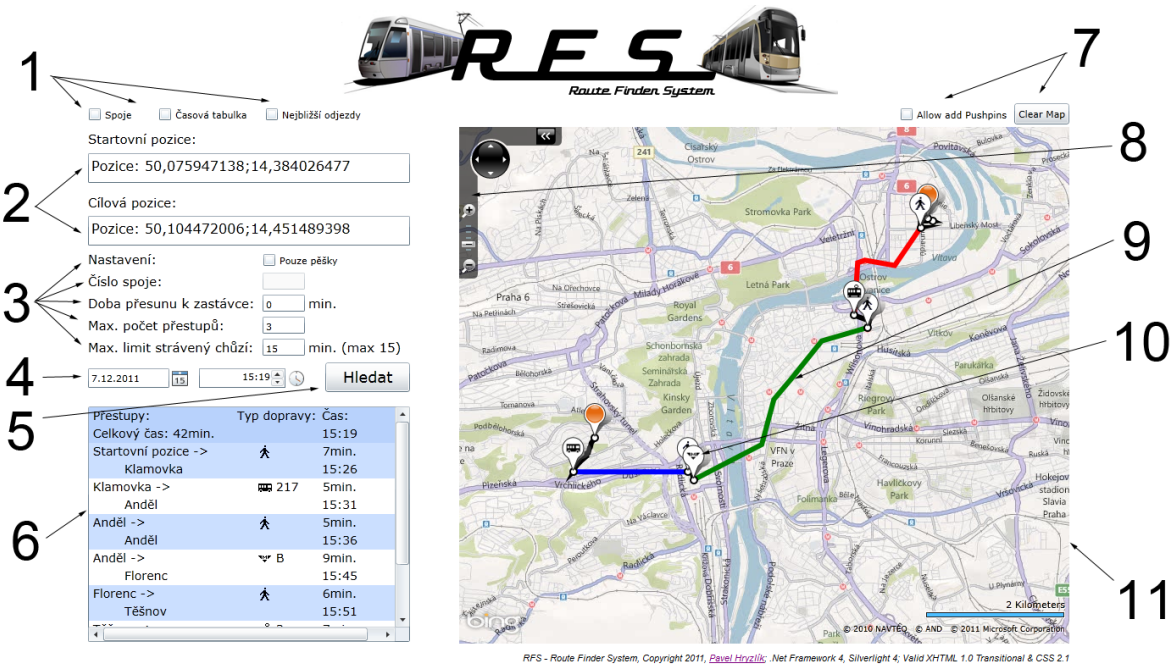
Seznam chybových hlášení:
OkValue - nevypíše nic, vše je v pořádku
NonValue - není zadána žádná vstupní hodnota
WrongValue - je zadána špatná hodnota

4.3.5 Zveřejnění

Po zkompilování Silverlight aplikace získáme výstupní soubor *.xap. Pokud z naší aplikace chceme udělat např. webovou aplikaci, musíme soubor *.xap vložit jako interaktivní obsah do nějaké webové stránky. Pro naše účely jsme k tomuto využili ASP.NET, kde jsme vytvořili webovou prezentaci a přidali do ní zkompilovanou Silverlight aplikaci.

5. Uživatelská dokumentace

5.1 Popis uživatelského rozhraní



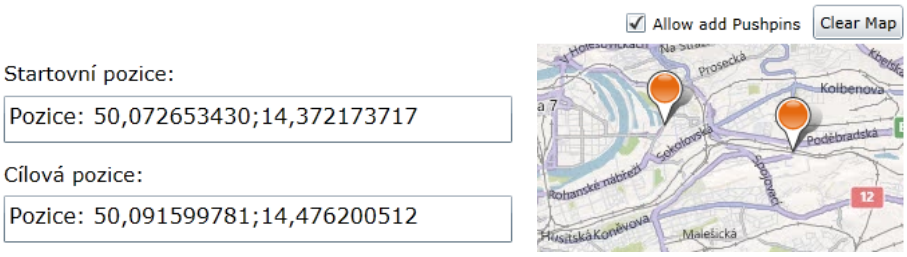
Obrázek 5.1: Uživatelské rozhraní

1. Dodatečné vyhledávací možnosti - možnost vyhledat informace o daném spoji, časovou tabulku a nejbližší odjezdy.
2. Pole na zadávání startovní/cílové zastávky, nebo GPS pozice.
3. Možnosti podrobného nastavení.
4. Nastavení datumu a času.
5. Hlavní vyhledávácí tlačítko, kterým se spustí vyhledávání.
6. Výstupní okno, kam se vypisují všechny důležité informace o vyhledávání - přestupy, časy apod.
7. Tlačítka sloužící k práci s mapou. Tlačítko `Allow add Pushpins` umožňuje přidávat do mapy startovní a cílové body. Tlačítko `Clear map` kompletně vymaže mapu.
8. V levém horním rohu mapy je ovládací panel mapy. Umožňuje přibližování/oddalování a pohyb po mapě.
9. Po skončení vyhledávání se zobrazí výsledná trasa na mapě formou křivky.
10. Na výsledné křivce se zobrazují význačné body, typicky znamenající přestupy.
11. Microsoft Bing mapa.

5.2 Vyhledání trasy

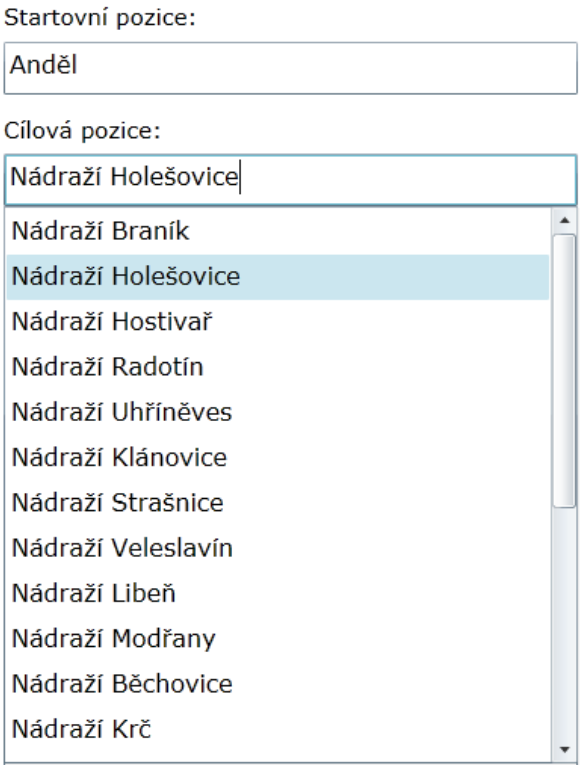
5.2.1 Krok první - výběr startovních bodů

Vybrat startovní a cílový bod můžeme dvěma způsoby. První možnost je označit v mapě dva body, kdy jeden bude startovní a druhý cílový. Musíme nad mapou povolit tlačítko `Allow add Pushpins`, abyhom do mapy mohli přidávat body. Ve vstupních polích se nám poté objeví GPS souřadnice označených bodů. (viz. obrázek 5.2)



Obrázek 5.2: Přidávání bodů

Druhou možností je zadat přímo do vstupních polí požadované zastávky. Vstupní pole nám po napsání pár znaků samo podbízí zastávky, které odpovídají vstupním znakům. (viz. obrázek 5.3)

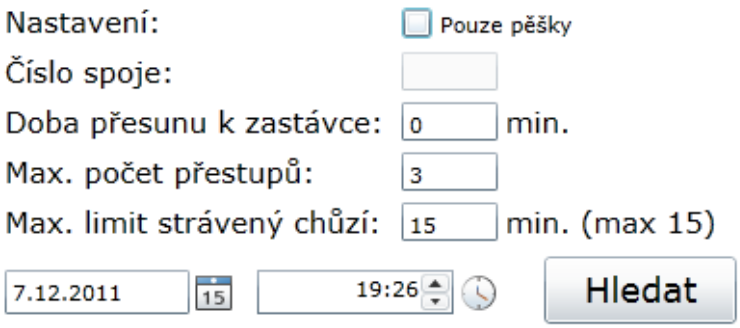


Obrázek 5.3: Výběr zastávky

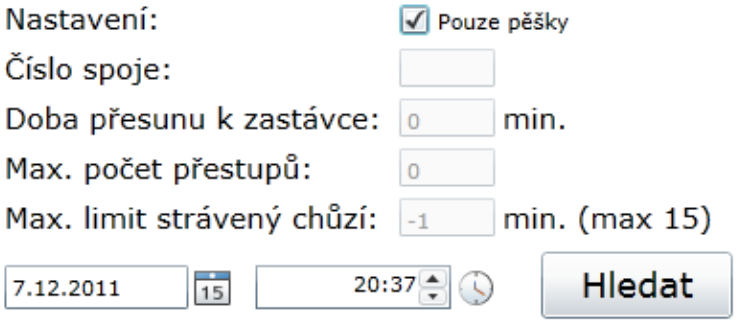
5.2.2 Krok druhý - volba nastavení

Pokud chceme vyhledávat trasy pouze pomocí MHD můžeme nastavit dobu přesunu k nejbližší zastávce, max. počet přestupů a max. limit strávený chůzí. V případě kombinovaného vyhledávání máme stejné nastavení, jen doba přesunu k nejbližší zastávce nemá žádný efekt. V neposlední řadě se tu nachází možnost nastavit datum a čas.

Máme tu také k dispozici tlačítko `Pouze pěšky`. Toto nastavení (viz obrázek 5.5) nám umožňuje vyhledávast trasy pouze pěší chůzí. Nepodporuje tedy dodatečné nastavení.



Obrázek 5.4: Nastavení

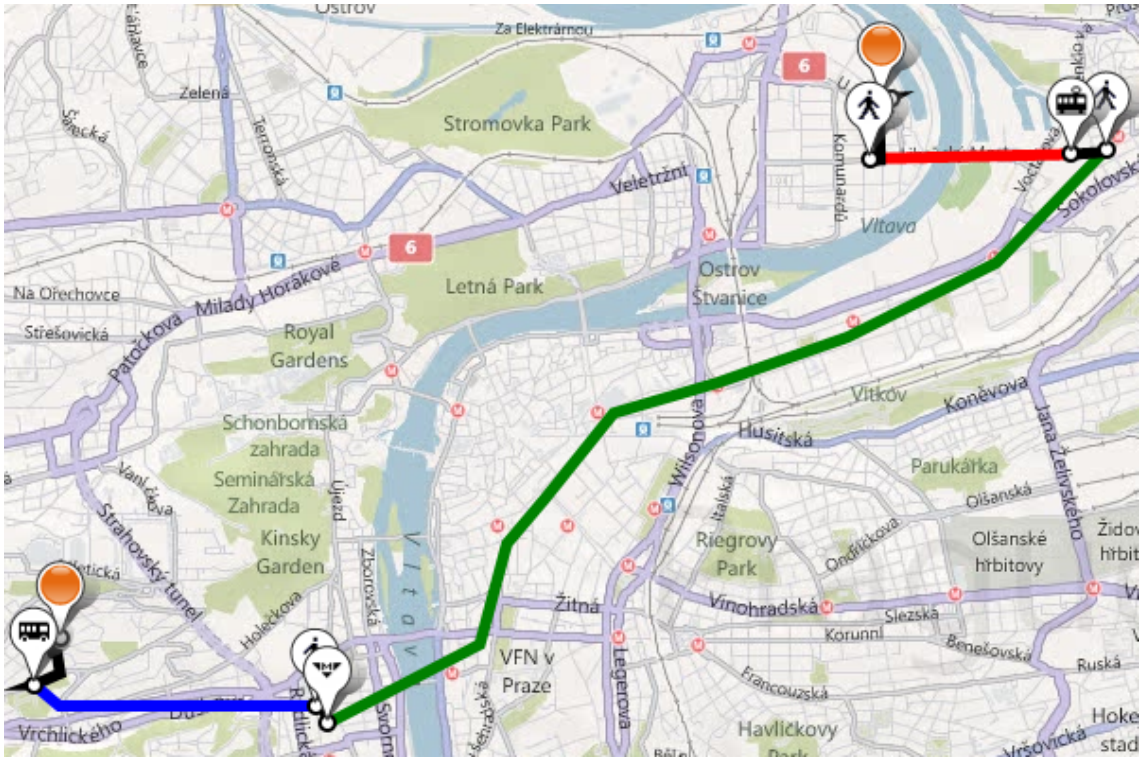


Obrázek 5.5: Nastavení pěší chůze

5.2.3 Krok třetí - vyhledání a zobrazení trasy

Po kliknutí na tlačítko Hledat se odešle požadavek k webové službě. Po vyhodnocení požadavku se výsledek zobrazí do výstupního okénka a trasa se vykreslí do mapy.

Popis zobrazené trasy



Obrázek 5.6: Zobrazovaná trasa

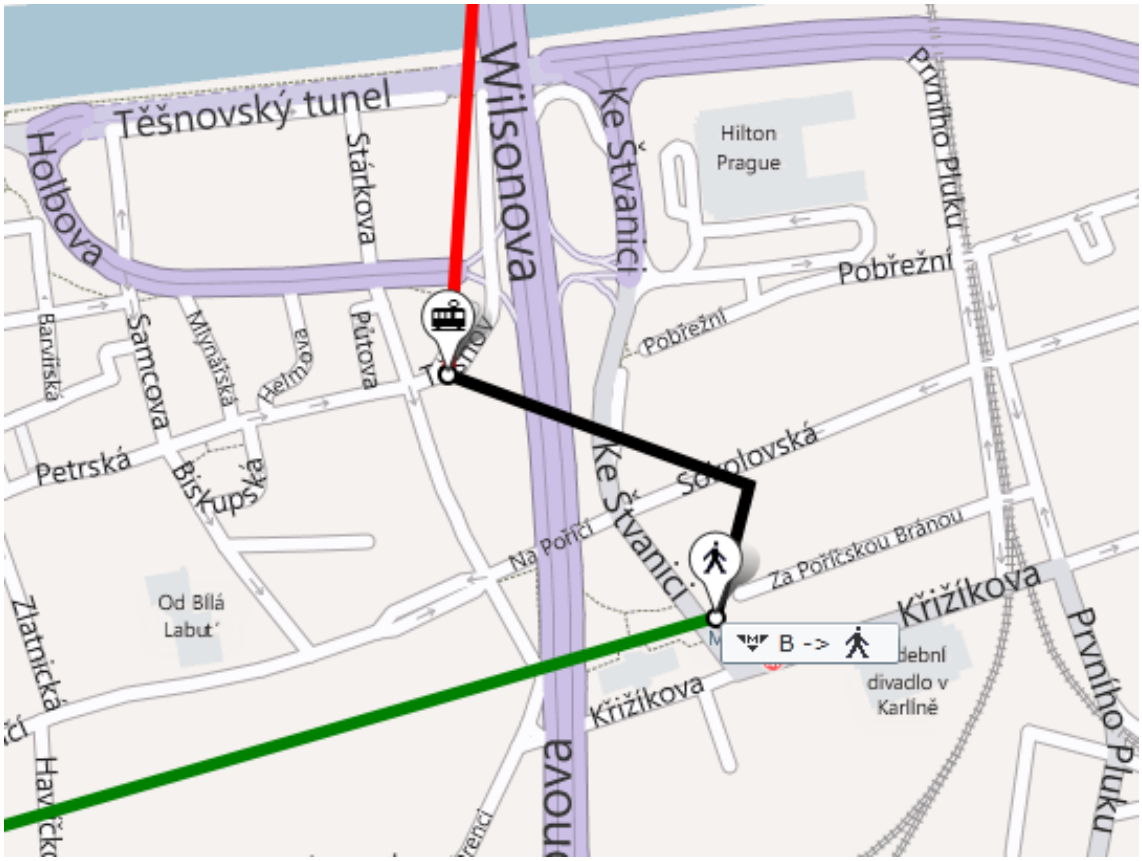
V zobrazené trase se objevují křivky 4 barevných typů:

1. černá - značí chůzi pěšky
2. zelená - značí jízdu metrem
3. modrá - značí jízdu autobusem
4. červená - značí jízdu tramvají

Jak můžeme vidět na obrázku 5.6, jedlotlivé cesty daným typem dopravy jsou rozděleny, jak barevně, tak pomocí význačných bodů, které nám oznamují na který typ dopravy se v daném místě přestupuje.

Detail přestupu

Obrázek 5.6 nám znázorňuje, jak vypadají přestupy. Každý přestup je označen význačným bodem, který nám obrázkem znázorňuje typ dopravy na který se přestupuje. Pokud myší najedeme na nějaký význačný bod, tak se nám zobrazí detaily přestupu. Na obrázku výše je to pro tenhle případ zobrazen přestup z metra B na pěší chůzi.



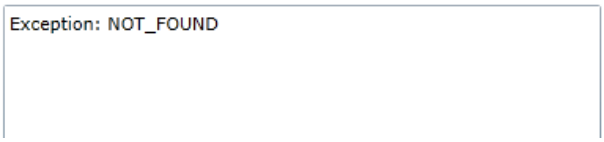
Obrázek 5.7: Detail zobrazované trasy

Popis výstupního okna

Výstupní informace se skládají z nadpisu, kde je vypsán popis sloupečků, celkový a startovní čas. Každý další řádek značí jednotlivé úseky cesty. Je tam odkud kam se daným typem dopravy přepravujeme, jak dlouho nám to bude trvat a v kolik hodin a minut tam budem. Za předpokladu, že aplikace žádnou trasu nenajde, nebo nastane nějaká interní chyba, vypíše do výstupního okénka název této chyby. Typicky se jedná příznak `NOT FOUND` (viz obrázek 5.9), což znamená, že žádná trasa splňující vstupní požadavky nebyla nalezena.

Přestupy:	Typ dopravy:	Čas:
Celkový čas: 26min.		17:18
Startovní pozice ->		4min.
Hřebenka		17:22
Hřebenka ->	176	10min.
Karlovo náměstí		17:32
Karlovo náměstí ->		1min.
Karlovo náměstí		17:33
Karlovo náměstí ->	14	4min.
Lazarská		17:37
Lazarská ->		6min.
Václavské náměstí		17:43

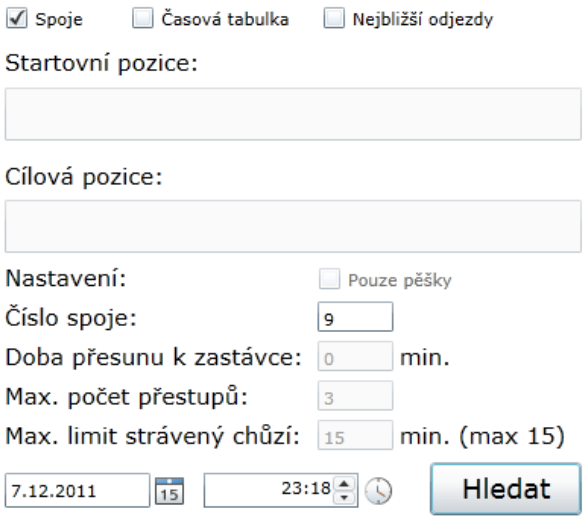
Obrázek 5.8: Výstupní okénko



Obrázek 5.9: Chybová zpráva

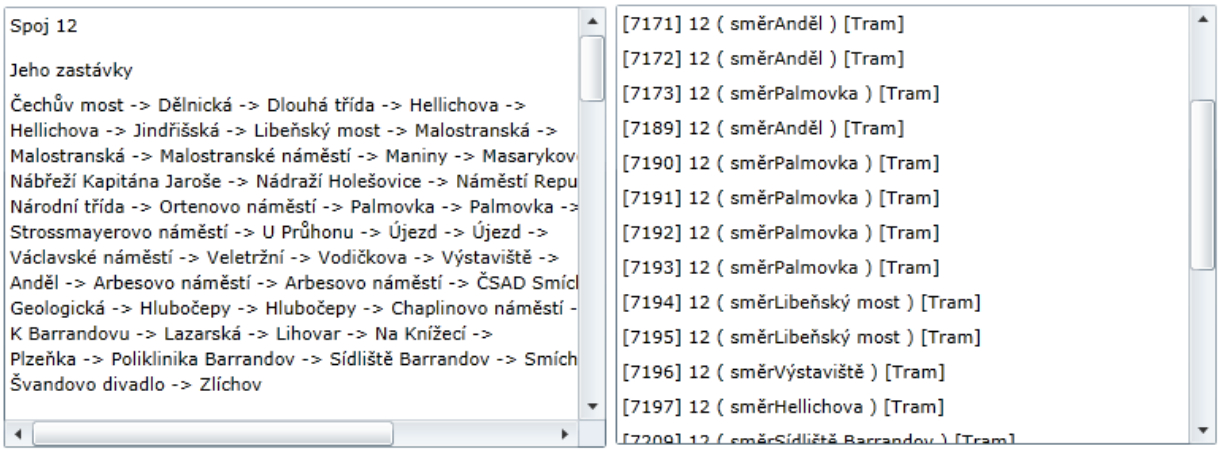
5.3 Dodatečné funkce

5.3.1 Informace o lince



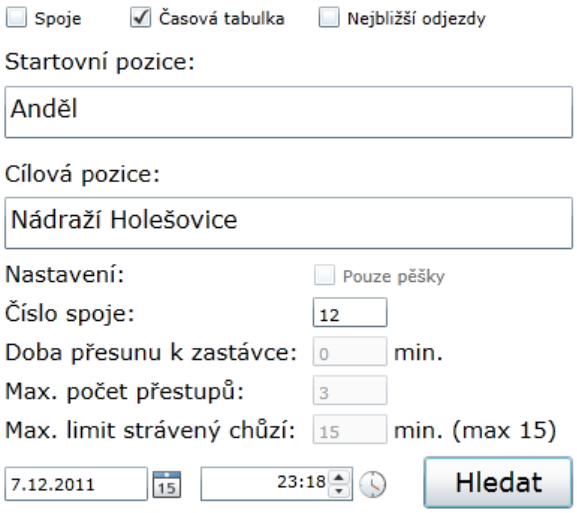
Obrázek 5.10: Zadání názvu linky

Dodatečné informace o některé z linek veřejné hromadné dopravy získáme tak, že zaškrtneme tlačítko spoje. Pak napíšeme do jediného volného políčka číslo linky (viz obrázek 5.10), kterou požadujeme. Pro metro zadáme písmeno A,B, nebo C. Výstup je pak seznam všech zastávek dané linky a seznam všech fyzických spojů, co danou linku jezní. (viz obrázek 5.11)



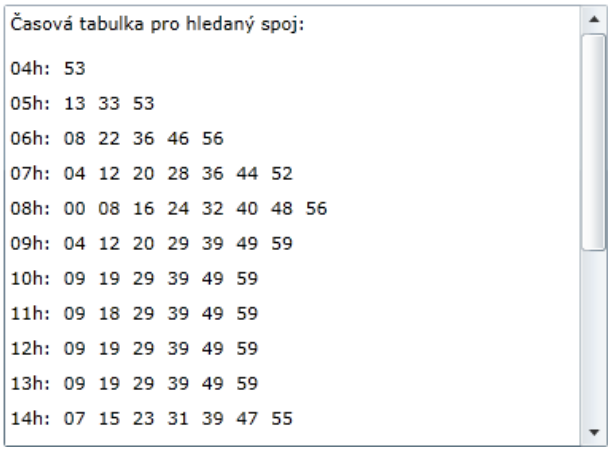
Obrázek 5.11: Informace o lince

5.3.2 Časové tabulky



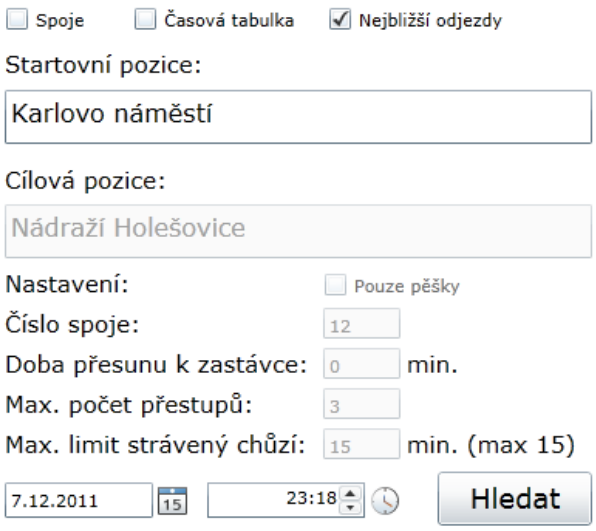
Obrázek 5.12: Požadavek o časovou tabulku

Pro vyhledání časových tabulek je potřeba zaškrtnout tlačítko Časová tabulka. K vyhledání potřebujeme tři informace. První musíme vyplnit zastávku z které chceme Časovou tabulku. Jako druhou zastávku vyplníme libovolnou zastávku na trase daného spoje. Tento údaj je tu proto, abychom mohli rozlišit směrovou orientaci časové tabulky. Poslední údaj, co je třeba vyplnit je požadovaný spoj. (viz obrázek 5.12) Výstupem je naformátovaná časová tabulka ve výstupním okénku. (viz obrázek 5.13)



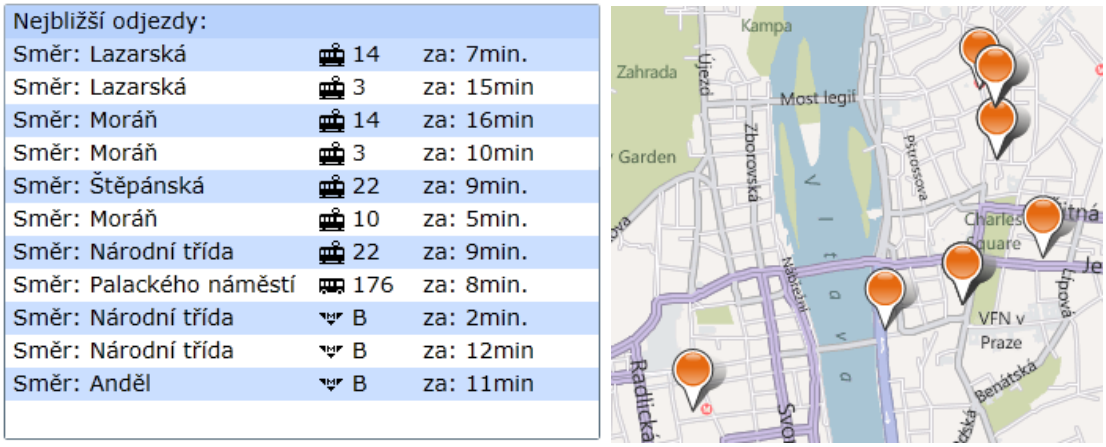
Obrázek 5.13: Výstupní časová tabulka

5.3.3 Nejbližší odjezdy



Obrázek 5.14: Požadavek o nejbližší odjezdy

Po zaškrtnutí tlačítka Nejbližší odjezdy a vyplnění zastávky lze vyhledat nejbližší odjezdy spojů z dané zastávky. (viz obrázek 5.14) Výstupem je seznam spojů s informacemi o tom, jakým směrem a za jak dlouho jedou. Cílové zastávky směrů jsou pro názornost ukázany na mapě pomocí význačných bodů.



Obrázek 5.15: Výstup nejbližších odjezdů

5.4 Tipy triky

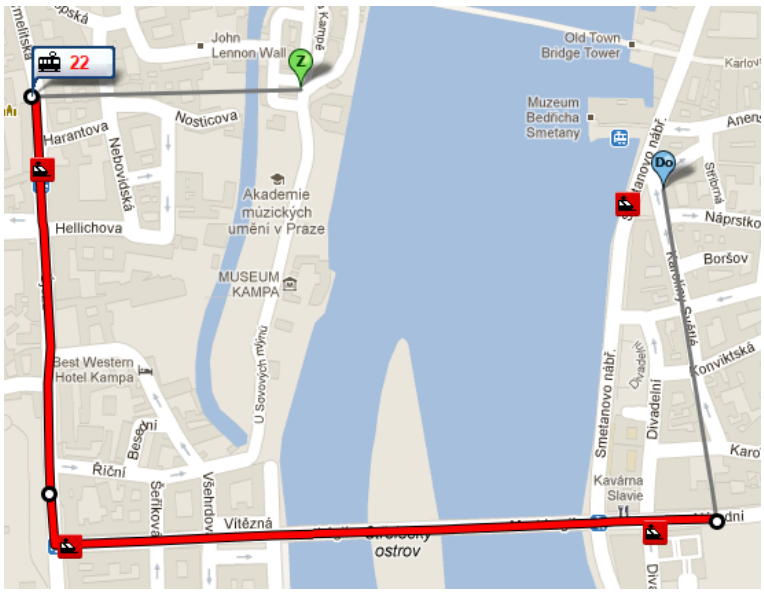
Začít vyhledávaní se doporučuje s dvěma zvolenými body bez žádného dodatečného nastavení. Za předpokladu, že nejsme s výslednou trasou, nebo s časem spokojeni, přidáváme počet přestupů, nebo maximální dobu strávenou chůzí. Za předpokladu, že je startovní a cílová pozice dostatečně blízko od sebe pak zkusíme přepnout jen do vyhledání pěších cest.

5.5 Aktualizace dat

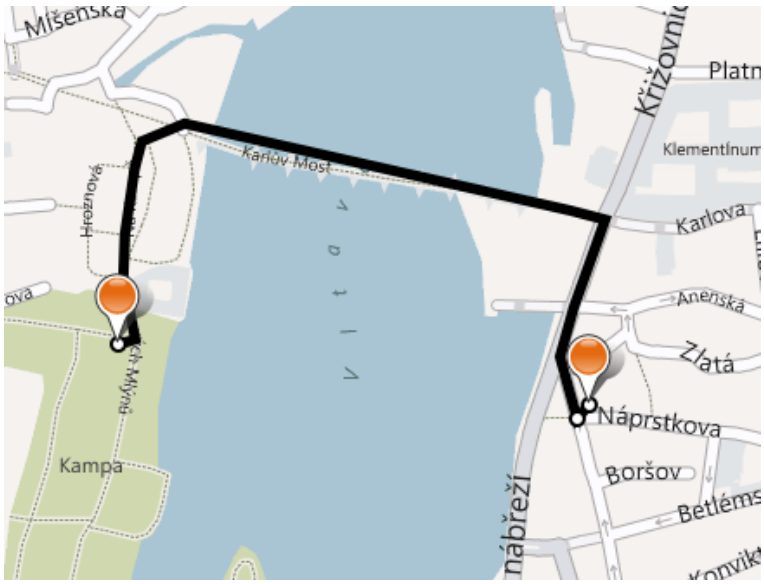
Hlavními zdrojovými daty jsou soubory *.dat a *.wlk. Jedná se o binární data obsahující informace o mapových podkladech a jízdní řády. Pro aktualizaci tedy stačí jednoduše nahrát nové tyto dva souboru. V projektu JRGPS existují pro tento účel speciální generátory. Jsou to programy, které jsou schopny vytvářet a aktualizovat výše popsané binární soubory.

6. Výsledky

Vrátíme se k příkladu z úvodu. Potřebujeme se dostat co nejrychleji z Kampy (levý břeh) na Smetonovo nábřeží (pravý břeh). Na následujících obrázcích je uvedeno srovnání s aplikací IDOS beta jakožto jediného konkurenta. Je jasně vidět, že na kratší vzdálenosti nemá smysl jezdit pomocí MHD a vyplatí se jít pěšky.

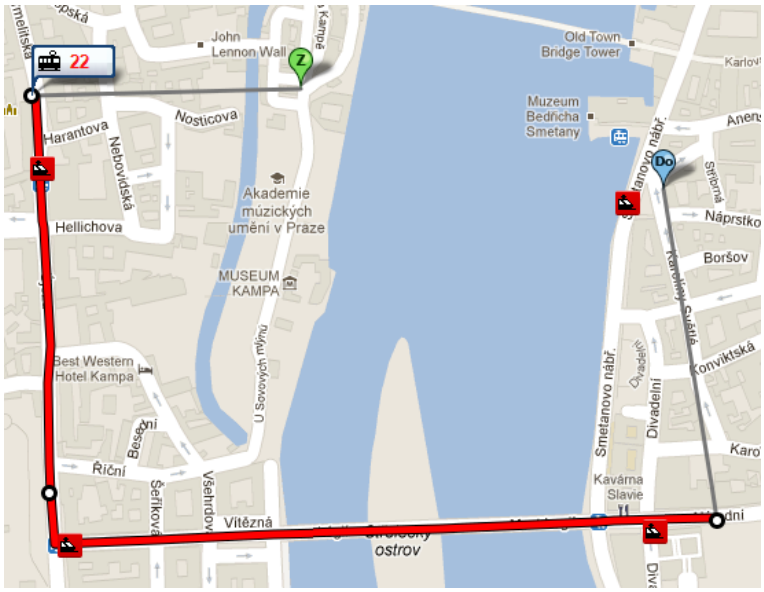


Obrázek 6.1: IDOS beta

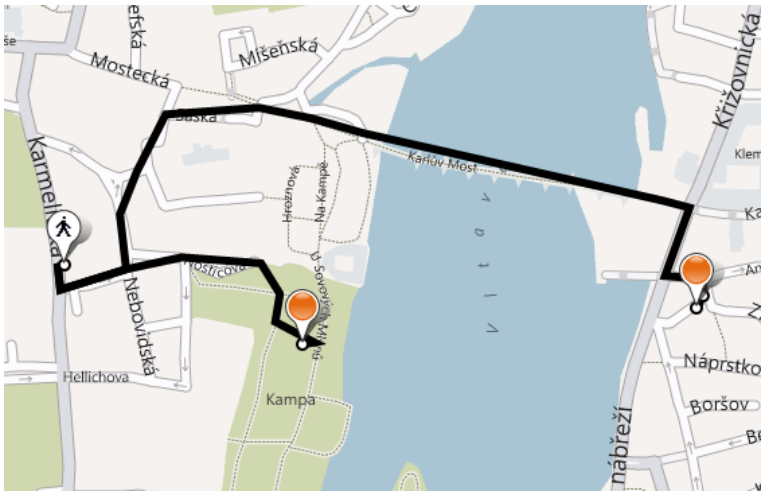


Obrázek 6.2: RFS - se zaškrtnou volbou pouze pěšky

Pokud ve stejném případě nechceme jít pouze pěšky a necháme vyhledat trasu s kombinací MHD, tak s dostatečným limitem maximální doby strávenou chůzí dostaneme řešení níže. Můžeme si všimnout, že program ze startovního i cílového bodu najde cestu k zastávce. Pokud je však vzdálenost dostatečně malá, tak už se necestuje pomocí MHD.

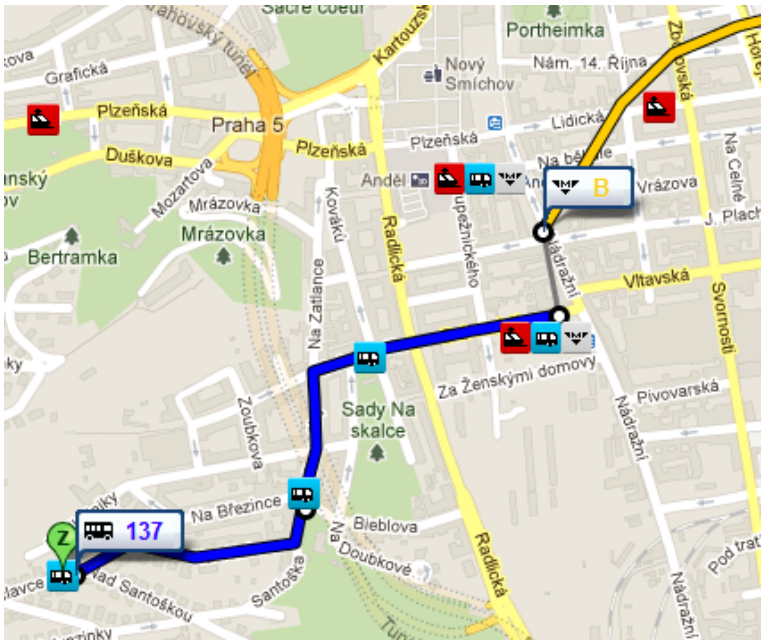


Obrázek 6.3: IDOS beta



Obrázek 6.4: RFS - hlavní vyhledávání s dostatečně velkým limitem maximální doby strávenou chůzí

Další zajímavou výhodou je, že díky přidání některých pěších hran do grafu MHD můžeme uštřit čas. Např. v situaci, kdy při vyhledávání tras pomocí MHD je výhodnější z nějaké zastávky na další dojít pěšky, díky čemuž stihneme dřívější navazující spoj. Na obrázcích níže vidíme, že když jedeme na zastávku Anděl pomocí autobusu 137, vystoupíme o zastávku dřív a dojdeme na Anděl pěšky, tak můžeme stihnout dřívější metro.



Obrázek 6.5: IDOS beta

Obrázek 6.6: RFS

Další výhodou je, že v pěších datech je uvedené ulice, pěšiny, stezky. Program nás se tedy nebude snažit navigovat např. po magistrále, či posilnici, kde nejsou chodníky. Je pěkně vidět na následujícím obrázku 6.7.

Obrázek 6.7: Bezpečnější cesta

Závěr

V rámci implementace této práce se podařily splnit všechny plánované kroky. Knihovna JRGPS byla rozšířena tak, aby umožňovala paralelní obsluhu více klientů najednou, poskytovala jednoduché rozhraní ke komunikaci a umožňovala efektivní plánování tras. Nad touto knihovnou vznikla webová služba, která se stará o komunikaci s knihovnou a umožňuje připojení multi-platforních klientů. Posledním prvkem této práce je klientská aplikace komunikující s webovou službou a umožňující skrze webové uživatelské rozhraní plánovat a zobrazovat trasy na mapovém podkladu.

Výsledná aplikace poskytuje efektivní plánování tras využívající kombinaci veřejné hromadné dopravy a chůze. Program ve srovnání s konkurencí vychází více, než dobře. Efektivní kombinaci pěší chůze a MHD totiž žádný konkurenční program plně nenabízí. Jako vzorové město bylo použito město Praha. Aplikace však může fungovat pro jakékoli další město, záleší pouze na vstupních datech.

Hlavní uvažovanou možností rozšíření je zavést podporu více měst najednou. Vhodným vylepšením by také bylo dodat do mapových podkladů výškové souřadnice, které by umožňily rozlišovat nadjezdy a ovlivňovat rychlost chůze v závislosti, zda-li jdeme do kopce, či z kopce. V neposlední řadě by bylo pěkné propojit aplikaci se sociálními sítěmi.

Seznam použité literatury

[1] JRGPS project documentation September 3, 2009.

[2] Microsoft Developer Network *http://msdn.microsoft.com*

[3] .NET Framework 4.0 Reference *http://msdn.microsoft.com/en-us/library/w0x726c2.aspx*

[4] Nearest Neighbor Search Using KD-tree *http://nixu.wordpress.com/2010/01/04/nearest-neighbor-search-using-kd-tree/*

[5] Bezděk Josef *Problém hledání nejbližšího souseda a indexační algoritmy* Západočeská univerzita v Plzni, 2009.

[6] Rina Panigrahy *An Improved Algorithm Finding Nearest Neighbor Using Kd-trees* Západočeská univerzita v Plzni, 2009. Brazil, April 2008. ISBN 978-3-540-78772-3

[7] Frequently Asked Questions § System requirements *Microsoft Silverlight product page. Microsoft Corporation.* November 4, 2010.

[8] Maps Silverlight Control Interactiove SDK *http://www.microsoft.com/maps/isdk/silverlight.*

[9] New Job Requirement: Experience Building RESTful APIs *http://blog.programmableweb.com/2010/06/09/new-job-requirement-experience-building-restful-apis/.*

[10] *http://www.wikipedia.org/.*

[11] FPablo Cibraro, Kurt Claeys, Fabio Cozzolino, Johann Grabner *Professional WCF 4: Windows Communication Foundation with .NET 4.* Wrox, June 15, 2010. ISBN 0-470-56314-1.

[12] Fielding, Roy T.; Taylor, Richard N. *Principled Design of the Modern Web Architecture, ACM Transactions on Internet Technology (TOIT).* New York: Association for Computing Machinery, 2002-05. ISSN 1533-5399.

[13] Richardson, Leonard; Ruby, Sam. *RESTful Web Services.* O'Reilly, 2005-07. ISBN 978-0-596-52926-0.

Příloha: Obsah přiloženého CD-ROM

V této příloze je popsána adresářová struktura přiloženého CD-ROM.

/RFS_DLL/ - zdrojové soubory rozšířené vyhledávací knihovny

/RFS_WCF_Service/ - zdrojové soubory WCF služby

/RFS_SilverLight_Client/ - zdrojové soubory klientské Silverlight aplikace

/README.txt - soubor popisující práci s jednotlivými části aplikace

/BP.pfd - elektronická verze bakalářské práce