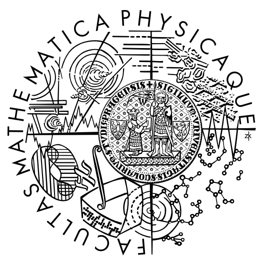


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Filip Havel

Systém pro vyhledávání a aktualizace jízdních řádů

Katedra softwarového inženýrství

Vedoucí bakalářské práce:	RNDr. Filip Zavoral, Ph.D.
Studijní program:	Informatika
Studijní obor:	Programování a softwarové systémy

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) 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

Děkuji panu RNDr. Filipu Zavoralovi, Ph.D. za odborné a trpělivé vedení mé práce a za čas, který mi během jejího vypracování věnoval. Rovněž děkuji lidem, kteří mě po dobu vypracovávání této práce podporovali.

Název práce: Systém pro vyhledávání a aktualizace jízdních řádů

Autor: Filip Havel

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D., Katedra softwarového inženýrství

Abstrakt: Většina lidí využívající veřejnou dopravu má specifické oblasti zájmů, kterými je možné omezit dopravní síť na určité části. U těchto lidí lze předpokládat, že budou chtít být informováni v případě, že nastane změna ve vymezené části dopravní sítě. Z tohoto důvodu jsme vytvořili aplikaci pro informování uživatele o změnách veřejné dopravy, které jej zajímají. Předpokládáme, že hlavní oblastí zájmu uživatele jsou spojení mezi stanicemi, a tak se práce zabývá především možnostmi vyhledání spojení v jízdních řádech. Kromě zmíněného vyhledávání jsme se zaměřili na přizpůsobitelnost aplikace a implementovali příklady možných přizpůsobení pro různé výstupy výsledků, uživatelská rozhraní a zdroje dat. Pro pohodlí uživatele jsme vytvořili mobilní aplikaci, která komunikuje s hlavní aplikací a zobrazuje aktuální data jízdních řádů dle zájmů uživatele.

Klíčová slova: jízdní řády, časově závislý graf, Dijsktrův algoritmus

Title: Searching and Updating Public Transport Timetables

Author: Filip Havel

Department: Department of Software Engineering

Supervisor: RNDr. Filip Zavoral, Ph.D., Department of Software Engineering

Abstract: Most of the people who use public transportation have specific areas of interests which can be used to cut down the transport network to several smaller parts. It is reasonable to assume that these people will want to be informed in case a change occurs in the restricted part of transport network they are interested in. For this reason, we created an application which informs its users about the changes in public transportation that concern them. We assume that for users, the most important area of interest are the connections between individual stations, which is why this thesis mainly focuses on the possibility of search for connection in timetables. In addition to this search, we also deal with the application's adaptability and we have implemented examples of expansions for various result outputs, user interfaces and data sources. To improve general user experience, we have developed a mobile application that communicates with the main application and displays updated timetables according to the user's area of interest.

Keywords: timetable, time-dependent graph, Dijkstra's algorithm

Obsah

Úvod **5**
Cíle . . . 5
Struktura práce . . . 6

1 Existující řešení **7**
1.1 Vyhledávací aplikace . . . 7
1.2 Google . . . 7
1.3 Tato aplikace . . . 8
1.4 Zdroj dat . . . 8

2 Návrh a architektura aplikace **9**
2.1 Základní data aplikace . . . 9
2.2 Požadavky . . . 9
2.3 Jádro aplikace a ImplicitModules . . . 10
2.4 Jádro . . . 10
2.5 Uživatelské rozhraní . . . 12
2.6 Notifikátor . . . 12
2.7 Zdroj dat . . . 12
2.8 Tvorba formátovaného výstupu . . . 13
2.9 Filtrování výsledků . . . 13
2.10 Zdroj dat CIS JŘ . . . 14
2.11 Konzole . . . 15
2.12 Mobilní aplikace . . . 15
2.13 Napojení mobilní aplikace . . . 16
2.13.1 Výsledné propojení mobilní a desktopové aplikace . . . 16
2.13.2 Identifikace požadavku . . . 17
2.13.3 Synchronizace . . . 17
2.13.4 Zprávy . . . 18

3 Vyhledávání **21**
3.1 Formalizace problému . . . 21
3.1.1 Problém nejdřívějšího příjezdu . . . 21
3.2 Reprezentace dat grafem . . . 22
3.2.1 Příklad jízdního řádu . . . 23
3.2.2 Time-expanded model . . . 24
3.2.3 Time-dependent model . . . 25
3.2.4 Reálné modely . . . 26
3.2.5 Přestupy . . . 27
3.2.6 Denní periodicita . . . 29
3.2.7 Implementovaný model . . . 30
3.3 Algoritmus vyhledávání . . . 31
3.3.1 Prioritní párovací fronta . . . 31
3.3.2 Navržený algoritmus . . . 31
3.3.3 Průchod týdnem . . . 35

4 Experimentální srovnání uvedeného řešení **37**
4.1 Spuštění experimentu 37
4.2 Prostředí experimentu 37
4.3 Výsledky experimentu 37
4.3.1 Vyhodnocení 38

5 Implementace **39**
5.1 Desktopová aplikace 39
5.1.1 K čemu je aplikace určena 39
5.1.2 Jak vypadá hromadná doprava z pohledu systému 39
5.1.3 Co se stane po spuštění 39
5.1.4 Práce s aplikací 40
5.1.5 Základní rozšíření aplikace 42
5.1.6 Rozšíření formátovače výstupu 45
5.2 Mobilní aplikace 47
5.2.1 Rozdělení aplikace 47
5.2.2 Synchronizace 47
5.2.3 Hlavní aktivita 47
5.2.4 Vytvoření požadavku 49

Závěr **51**

Reference **53**

I Přílohy **55**

A Reference výsledků **57**

B Gramatika automatu **59**

C Instalace **61**
C.1 Desktopové aplikace 61
C.1.1 Předpoklady pro běh aplikace 61
C.1.2 Bez instalačního balíčku 61
Hierarchie souborů aplikace 61
C.1.3 Přidání rozšíření 62
C.2 Mobilní aplikace 62
C.2.1 MyTransportation.apk 62
C.2.2 MobileInterface.dll 62
C.2.3 První spuštění 62

D Implementační detaily jádra aplikace a ImplicitModules **65**
D.1 Základní data aplikace 65
D.2 Výroba vlastního rozšíření 65
D.3 Jádro aplikace 65
D.4 Model databáze 66
D.5 Kontrola požadavků na spojení 68
D.6 ImplicitModules 68
D.6.1 Výstup do tabulky 68

D.6.2 Notifikace emailem 69
D.6.3 Konzolové rozhraní 69
D.6.4 Protokol komunikace skrze rouru 69
D.6.5 Zdroj dat CIS JŘ 71
D.7 Knihovna ResourcesForPlugins 72

E Implementační detaily mobilní aplikace **73**
E.1 Použité knihovny 73
E.2 Mobilní aplikace 73
E.3 Rozšíření hlavní aplikace 74
E.3.1 Periodický kontrolní cyklus (*Looper*) 74
E.3.2 Lokální uložení dat 74
E.3.3 Odesílání zpráv 74
E.3.4 Zpracování výsledku 75
E.3.5 MobileInterface.dll.config 75
E.3.6 Projekty 75
E.4 Prostředník 75
E.4.1 API 77

F Kompilace jádra a základního rozšíření **79**
F.1 Části programu 79
F.2 Preprocesorové direktivy 80

G Obsah elektronické přílohy **81**

Úvod

Veřejná hromadná doprava je nepostradatelnou součástí velkých měst. Velké množství lidí ji využívá pravidelně například pro cestu do (resp. z) práce. Vyhledávat každý den totožnou trasu se stejnými kritérii může být náročné a zbytečné, jelikož trasa se často nezmění a není těžké si ji zapamatovat. Následně však může dojít ke změně v hromadné dopravě, které si člověk nevšimne a přijede do cíle pozdě nebo vůbec. A právě řešením tohoto problému se tato práce zabývá.

V práci je postupně popsána analýza a implementace desktopové aplikace (resp. jejích částí), která umožní uživateli nastavit oblíbená spojení, odjezdy ze stanic, či jízdy linek. Tyto nastavené oblasti zájmů (dále jen požadavky) aplikace pravidelně kontroluje a v případě změny ve výsledku některého požadavku informuje uživatele o nastalé změně (např. zasláním emailu).

Jelikož se předpokládá, že nejčastěji bude uživatel zadávat požadavek na spojení, je součástí aplikace, a také velkou části této práce, časově závislá verze Dijkstrova algoritmu pro vyhledání spojení mezi stanicemi. V textu práce jsou uvedené již existující způsoby modelování a vyhledávání v jízdních řádech. Následně je navržena vlastní úprava jedné z možností.

Součástí této práce je také analýza a implementace mobilní aplikace, která se spojí s desktopovou aplikací, čímž vytvoří přívětivé uživatelské rozhraní s možností mít výsledky vždy po ruce.

Cíle

Cílem práce je navrhnout a implementovat desktopovou aplikaci, která:

- Informuje uživatele o změnách v jízdních řádech, které ovlivnily výsledky dat sledovaných uživatelem.
- Umožňuje uživateli zadat požadavky na sledování dat linek, stanice a spojení ve veřejné dopravě.
- Je modulární tak, aby ji bylo možno rozšířit o různé zdroje dat jízdních řádů, uživatelská rozhraní, výstupy výsledných dat, způsoby informování uživatele o změně.
- Umí vyhledávat spojení v jízdních řádech.
- Je schopna vytvářet Excel tabulky s výsledkem.
- Má konzolové rozhraní.
- Informuje uživatele o změnách emailem.
- Pracuje s daty jízdních řádů Pražské integrované dopravy.

Dalším cílem je navrhnout a implementovat mobilní aplikaci pro prohlížení výsledků, vytváření požadavků a informování uživatele o změnách v jízdních řádech, která bude sloužit jako zjednodušené uživatelsky přívětivé rozhraní pro desktopovou aplikaci.

Struktura práce

V první kapitole jsou uvedeny existující aplikace týkající se jízdních řádů z pohledu běžného člověka. Dále v druhé kapitole je uveden návrh a architektura aplikace jako celku. Následně se v třetí kapitole práce zaměřuje na vyhledávání v jízdních řádech, kde je uvedena vlastní úprava časově závislé verze Dijkstrova algoritmu a její očekávané přínosy. V následující kapitole jsou tato očekávání experimentálně otestována. Nakonec jsou v páté kapitole popsány vyvinuté aplikace.

1. Existující řešení

V této kapitole jsou popsána aktuálně známá řešení pro vyhledávání a práci s jízdními řády. Zabýváme se funkcemi vybraných a otázkou, jaké funkce by měla mít naše aplikace.

1.1 Vyhledávací aplikace

První skupinou jsou aplikace založené na vyhledávání v aktuálních datech. Mezi ně patří jak mobilní aplikace (např. Jízdní řády IDOS[1], PID info[2]), tak weby (např. IDOS [3], DPP[4]).

Pomocí těchto aplikací může uživatel vyhledávat spojení, odjezdy ze stanice nebo jednotlivé jízdní řády linek. Všechny tyto aplikace nabízí v rámci vyhledávání různá filtrování dat jako:

- Čas odjezdu (resp. příjezdu) včetně data
- Omezení počtu přestupů shora
- Nastavení rychlosti, maximální doby, nebo minimální doby přestupu
- Povolené (resp. zakázané) dopravní prostředky
- Nutnost bezbariérového (resp. nízkopodlažního) spojení
- Průjezd nebo přestup v zastávkách

Některé aplikace umožňují zvolit oblast vyhledávání (nejčastěji město nebo stát).

Jiné mobilní aplikace (např. CG Transit[5], Praha Transit[6], ezRide[7]) umožňují vyhledávaní offline. Při použití offline vyhledávání aplikace kladou vyšší nároky na výkon a úložiště zařízení, zároveň vzniká problém s aktualizací stažených dat.

1.2 Google

Mapy Google umožňují zajímavou funkci, která se nejvíce přibližuje zadání této práce. Je možné si nastavit obvyklé trasy a dostávat notifikace, že je čas vyrazit v závislosti na aktuální dopravě[8].

Kromě automatického plánování trasy s ohledem na aktuální jízdní řády má výhodu podkladových map, které umožňují počítat dobu chůze na stanici odjezdu. Tím se objeví možnost více počátečních a cílových stanic.

[1] https://play.google.com/store/apps/details?id=cz.mafra.jizdnirady
[2] https://play.google.com/store/apps/details?id=cz.dpp.praguepublictransport
[3] https://jizdnirady.idnes.cz/vlakyautobusymhdvse/spojeni/
[4] http://spojeni.dpp.cz/
[5] https://www.circlegate.com/cs/cgt
[6] https://play.google.com/store/apps/details?id=com.swash.transitworld.praha
[7] https://www.circlegate.com/ezride
[8] Ukázka nastavení je vidět https://www.androidpolice.com/2017/09/28/google-maps-v9-63-beta-gets-ready-launch-new-commuting-features-hints-increased-points-long-reviews-new-badges-apk-teardown/

1.3 Tato aplikace

Hlavním smyslem vyvíjené aplikace je kontrola aktuálnosti předem definovaných požadavků.

Jak lze vidět u existujících řešení, mohou být požadavky tří typů (data linky, odjezdy ze stanice, spojení mezi stanicemi). K požadavkům by následně měly existovat různorodé filtry. Navíc je požadováno, aby byly požadavky více obecné a zahrnovaly více dílčích požadavků, tak můžeme vytvořit například požadavek obsahující více stanic odjezdu.

Vyvinutá mobilní aplikace by následně měla mít k dispozici výsledky těchto požadavků bez nutnosti připojení k internetu (připojení bude nutné pouze pro založení nového požadavku nebo pro příjem nových dat). Zároveň je požadováno, aby aplikace neukládala celý graf dopravní sítě (a nezabírala tak příliš mnoho paměti zařízení).

Dále by bylo výhodou, aby aplikace byla co nejvíce přizpůsobitelná a bylo možné přidávat nové zdroje dat či nové filtry.

1.4 Zdroj dat

V době počátků práce byl volně přístupný jediný zdroj dat CIS JŘ[9], který má zákonnou povinnost poskytovat data o jízdních řádech veřejné dopravy strojově zpracovatelnou formou. Jak bylo zjištěno v průběhu práce, tento zdroj není úplně ideální, především vzhledem k jeho neustálým změnám.

Aktuálně existuje pravděpodobně lepší zdroj [10], který se zdá být více stabilní a snáze zpracovatelný (především díky referenčnímu formátu GTFS, k němuž jsou dostupné různé knihovny).

Výstup CIS JŘ byl během práce měněn. Autobusová doprava byla poskytována ve formátu JDF (postupně ve verzích 1.9,1.10 a 1.11) a drážní data (včetně tramvají a metra) byla poskytována ve formě PDF a nyní[11] jsou některá drážní data také ve formátu JDF.

[9] https://www.chaps.cz/cs/products/CIS
[10] https://pid.cz/o-systemu/opendata/
[11] 3. 5. 2018

2. Návrh a architektura aplikace

Prvotní myšlenkou aplikace byla co největší přizpůsobitelnost. Dále měla být vytvořena desktopová aplikace a následně mobilní aplikace, sloužící jako UI pro hlavní aplikaci. Naše hlavní aplikace by měla splňovat tyto funkce:

- Získávat data z potenciálně libovolného zdroje
- Kontrolovat aktuálnost dat
- V případě změny upozornit uživatele
- Zobrazit výsledek vyhledávání uživateli
- Přijímat požadavky od uživatele
- Umožnit filtraci dat

Podle těchto funkcí je aplikace rozdělena na části *zdroj dat*, *notifikátor*, *uživatelské rozhraní* a *tvorba formátovaného výstupu*, které obalují jádro aplikace, kontrolující změny v požadavcích. V rámci přizpůsobitelnosti jsou tyto části přidávány k jádru jako rozšíření ve formě *dll* knihoven, které bude jádro využívat dle požadavků uživatele. Management těchto rozšíření si jádro zajišťuje samo.

2.1 Základní data aplikace

Z pohledu aplikace je systém veřejné dopravy definován stanicemi a linkami. Linky jsou definovány množinou jízd s podobnou trasou. Trasu tvoří posloupnost uzlů trasy, což jsou stanice s dalšími informacemi popisujícími trasu.

2.2 Požadavky

V rámci aplikace používáme definici požadavků, což jsou specifikované oblasti zájmu uživatele, které zadá do systému. Jeden požadavek je tvořen množinou dílčích požadavků (což umožní vytvářet komplexnější požadavky), identifikací zdroje dat a způsoby upozornění uživatele. Požadavky (resp. dílčí požadavky) mohou být tří typů.

1. Požadavek na linku
    - Je požadavek na data linky.
    - Data mohou být filtrována na časový interval, v němž hledaná linka projíždí určitou stanici.

2. Požadavek na odjezdy ze zastávky
    - Je požadavek na data linek, které odjíždějí z této zastávky.

- Data mohou být filtrována na odjezdy v určitém časovém intervalu, na linky, které pokračují přes alespoň jednu ze zastávek, kterou uživatel k požadavku dodá, a na typ linky (autobus, vlak, tramvaj, trolejbus, metro).

3. Požadavek na spoj
   - Je požadavek pro data o všech spojích mezi počáteční a cílovou zastávkou.
   - Data mohou být filtrována na časový interval odjezdu, nebo na časový interval dojezdu spoje a použití linek určitého typu (autobus, vlak, tramvaj, trolejbus, metro).

Aplikace s požadavky pracuje tak, aby umožnila co nejvyšší rozšiřitelnost. Jednotlivé pluginy následně mohou definovat vlastní filtry a přidávat další parametry požadavku.

2.3 Jádro aplikace a ImplicitModules

V základní verzi je systém tvořen jádrem (*TransportSystemKernel*) a základním pluginem (*ImplicitModules*), který poskytuje zdroj dat z CIS JŘ, notifikaci emailem, výstup do excelové tabulky a konzolové rozhraní. K *ImplicitModules* patří také spustitelný soubor *SimpleConsole.exe*, který zajišťuje konzolové UI. UI komunikuje s aplikací skrze pojmenovanou rouru, kdy rouru vytvoří komunikační cyklus v *ImplicitModules* a *SimpleConsole* se k ní připojí. Na obrázku 2.1 jsou zobrazeny tyto soubory a jejich důležité části. V příloze D jsou sepsány další detaily pro lepší orientaci ve zdrojových kódech.

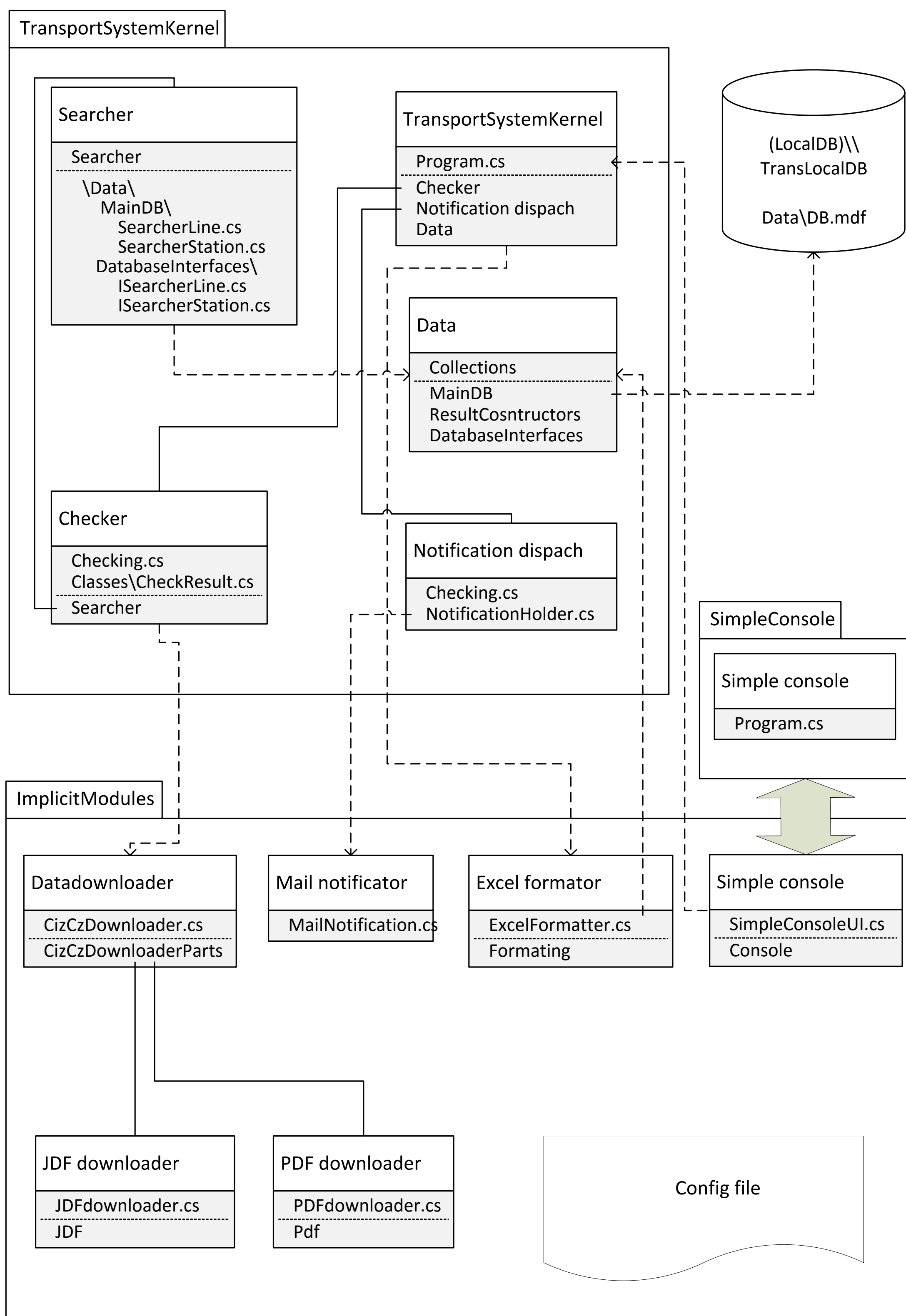
2.4 Jádro

Od jádra požadujeme schopnost uchovávat požadavky a aktuální výsledky požadavků. Musí být schopné najít výsledek pro požadavky i v omezených zdrojích dat a musí rozpoznat změny ve výsledcích.

Získání výsledku pro požadavek je závislé na typu požadavku a možnostech poskytnutých zdrojem dat. Vyhledání správného výsledku tak může být zajištěno zdrojem dat nebo jádrem aplikace. Například při vyhledávání spojení mezi stanicemi může zdroj dat poskytovat přímo výsledek, nebo výsledek nalezne jádro za pomocí zdroje dat, který poskytuje seznam stanic (resp. linek). Aby bylo jádro schopné vyhledat spojení mezi stanicemi, je v jádru implementován základní vyhledávač spojení (viz kapitola 3).

Detekce změn výsledků spočívá v porovnání předchozího (uloženého) výsledku s aktuálním. Pokud nejsou výsledky stejné, pak nastala změna, na níž je upozorněn uživatel, a nový výsledek se uloží pro další zpracování.

Dále je potřeba, aby jádro aplikace bylo schopno perzistentně ukládat data. Mezi daty lze nalézt řadu vazeb, proto je použita relační databáze. Protože aplikace musí být schopná běžet na počítači uživatele, vybírali jsme z neplacených



Obrázek 2.1: Model aplikace spolu se základním rozšířením (konzolové rozhraní, notifikace emailem a zdroj dat CIS JŘ)

verzí známých relačních databází. V úvahu pak připadají z nejčastěji používaných databází Oracle, MS SQL, nebo MySQL. Ve vyvíjené aplikaci nezáleží tolik na výkonu a nepředpokládá se ani uložení velkého množství dat. Vybrali jsme MS SQL Express LocalDB, protože desktopová aplikace cílí na OS Windows a tato kombinace by měla mít velmi nízký dopad na uživatelův počítač a je možné, že již bude nainstalována z balíčku jiné aplikace. Na obrázku D.1 je UML schéma tabulek databáze.

Poslední funkcí jádra je řízení práce s rozšířeními. Toto zahrnuje detekci rozšíření a přiřazení jednoznačného identifikátoru, který je pak používán v uživatelském rozhraní při psaní příkazů.

2.5 Uživatelské rozhraní

Uživatelské rozhraní umožňuje uživateli ovládat jádro aplikace. Skrze toto rozhraní se vytváří požadavky. Další funkce poskytované uživatelským rozhraním jsou funkce pro nastavení programu, výpis všech kontrolovaných požadavků, výpis všech pluginů a jejich identifikátorů (rozhraní, notifikace, formátovač) a vytvoření formátovaného výstupu dat.

Mezi nastavení aplikace patří zejména interval, v kterém se má kontrolovat aktuálnost dat.

2.6 Notifikátor

Notifikátor obdrží instanci požadavku, v němž došlo ke změně, a specifickým způsobem (např. emailem u pluginu *ImplicitModules*) upozorní uživatele na změnu.

Vždy je spuštěn jako nová úloha, což mu umožní čekat na připojení cíle notifikace (např. mobilu).

Po skončení musí jádro informovat o tom, zda notifikace dopadla úspěšně. Pokud notifikace skončí a nepředá informaci o úspěchu, bude jádro vyvolávat notifikaci znovu, dokud neproběhne úspěšně.

2.7 Zdroj dat

Zdroj dat je část programu, která se stará o získání dat ze zdrojů. Musí implementovat rozhraní pro předání požadavku a dále může implementovat následující rozšíření tohoto rozhraní:

- Seznam linek
  - Rozhraní schopné vrátit seznam názvů všech linek.
- Seznam stanice
  - Rozhraní schopné vrátit seznam stanic v dopravní síti.
- Data linky
  - Rozhraní schopné vrátit data o lince.

    - Pokud je implementováno toto rozhraní spolu s rozhraním seznamu linek, je jádro schopno získat data na požadavky rozhraní Data zastávky i Data spoje.

- Data zastávky
    - Rozhraní schopné vrátit data o všech linkách projíždějící danou zastávkou.
    - Pokud je implementováno toto rozhraní spolu s rozhraním seznamu stanic, je jádro schopno zajistit vyhledání spoje.

- Data spoje
    - Rozhraní schopné vrátit spojení mezi dvěma zastávkami.

Jádro této části vždy při inicializaci předá celou instanci požadavku, což umožní, aby uživatelský požadavek obsahoval mimo jiné také nastavení pro data downloader. Posléze jsou na něm postupně volány metody získávající data dle implementovaného rozhraní.

Zdroj dat může sám vyřešit celý požadavek i filtraci dat, a následně v metodách vracet jen relevantní data. Ta jsou ovšem vždy v jádru znovu filtrována dle požadavku, čímž je zajištěna jejich správnost.

Detailní popis rozhraní je k nalezení ve zdrojových kódech *ResourceForPlugins.*

Během získávání výsledků ze zdroje je třeba aplikaci předat jen relevantní data, tudíž není například nutné pro každou stanici, skrz níž projíždí hledaná jízda, předat informace o ostatních linkách, které skrz tuto stanici projíždějí. Data vrácená jádru pro dotaz nesmí obsahovat duplicity, pokud mají tedy dva elementy výsledku stejný význam, musí se jednat o tutéž instanci objektu na stejné adrese. Tato podmínka platí jen pro aktuální předání výsledku, ne pro všechny výsledky od spuštění aplikace. Vrácená data by měla být referenčně propojena podle schématu zobrazenému v příloze A, pokud se jedná o linku (viz obr. A.1) a pokud se jedná o stanici (viz obr. A.2).

2.8 Tvorba formátovaného výstupu

V případě potřeby vytvořit formátovaný výstup výsledku je možné využít pluginu formátovače. Při volání formátovače je předána instance požadavku, která může obsahovat rozšířené parametry určené pro vybraný formátovač. Všechny formátovače ovšem musí obsahovat výchozí nastavení, aby bylo možné vytvořit výstup pro požadavek bez rozšiřujících dat. Spolu s instancí je formátovači předáno rozhraní pro přístup do databáze a stejně tak jsou předána další data o výsledku.

2.9 Filtrování výsledků

Systém nabízí širokou možnost filtrování jednotlivých částí dopravní sítě. Z dat mohou být odfiltrovány jednotlivé stanice, linky, trasy a uzly trasy.

Speciálními případy jsou filtry generující počátek vyhledávání spojení.

Základními filtry, jež lze nalézt v knihovně *ResourcesForPlugins*, umožňují:

- odfiltrovat jízdy, které nepokračují do jedné z uvedených stanic.
- odfiltrovat jízdy, které nefungují v alespoň jednom z uvedených dnů.
- filtrovat dle typu požadavku, stanice a časového intervalu výsledné jízdy (resp. spojení). V případě požadavku na linku, nebo stanice vyfiltruje pouze jízdy projíždějící stanicí v časovém intervalu. V případě požadavku na spojení definuje časový interval v němž spojení odjíždí z první stanice (resp. přijíždí do poslední stanice).
- odfiltruje jízdy, které nepoužívají jeden z uvedených typů dopravních prostředků.

Filtry mohou být rozšířeny implementací zmíněných rozhraní a zajištěním jejich specifikace uživatelským rozhraním.

2.10 Zdroj dat CIS JŘ

Jako zdroj dat jsme implementovali knihovnu stahující a zpracovávající data z CIS JŘ. Vzhledem k rozdělení a častým změnám dat zpracovává knihovna formát JDF (ve více verzích) a dokumenty PDF. Následně jsou tato data spojena v jeden výsledek. Stažená (předzpracovaná) data jsou lokálně cachována uložením v dočasných souborech.

Pro zpracování PDF je využita knihovna *iTextSharp* [1], která umožňuje rozšířené zpracování PDF a je dostupná pod licencí *Affero General Public License* [2]. V úvahu také připadala knihovna *PDFsharp* [3], která ovšem nemá dobrou dokumentaci a není schopná poskytnout poziční informace o textu, které jsou důležité pro zpracování tabulky jízdního řádu.

Při testování aplikace bylo zjištěno, že poskytnutá data nejsou dostatečná. Respektive, pokud je linka zrušena, nelze tuto skutečnost z poskytovaných dat poznat.

Problém spočívá v tom, že data nesou informaci o platnosti *od-do*, ovšem tato platnost je nastavena při vytvoření dat a není dále aktualizována. V případě změny linky je vytvořena nová složka dat, která obsahuje nová data. Stará data ovšem nejsou odebrána z výstupního souboru, a tak je běžné, že data pro jednu linku existují v mnoha verzích, kdy jen jedna verze je aktuálně platná.

Problém tak nastává i při změně linky. Zde však existuje řešení: považovat za aktuální data, která jsou platná a mají nejzazší počátek platnosti.

Příklad. V datech použitých pro experiment lze nalézt data linky 1 ve 13 verzích s různou platností (jejich seznam je vypsán v tabulce 2.1). Dne 29. 4. 2018 by platila data ze složky 577. Zároveň lze nalézt v datech ve složce 2066 linku 239 s platností 10. 12. 2017 až 08. 12. 2018. Tato linka přitom 3. 1. 2018 byla zrušena[1].

Pro ověření zrušení linky bylo nutné přidat do zdroje dat kontrolu existence linky, která vytvoří dotaz na webovou stránku dpp.cz a z vrácené stránky se snaží rozpoznat, zda linka stále existuje.

[1] Pro upřesnění časových souvislostí byl tento příklad napsán 8. 6. 2018. Může se stát, že linka bude znovu zavedena

Složka	Platné od		Platné do
504	02. 09. 2017	–	08. 12. 2018
627	25. 11. 2017	–	08. 12. 2018
565	04. 12. 2017	–	08. 12. 2018
168	24. 12. 2017	–	08. 12. 2018
245	09. 12. 2017	–	08. 12. 2018
624	03. 01. 2018	–	08. 12. 2018
642	15. 01. 2018	–	08. 12. 2018
266	22. 01. 2018	–	08. 12. 2018
572	22. 01. 2018	–	08. 12. 2018
108	21. 02. 2018	–	08. 12. 2018
675	03. 03. 2018	–	08. 12. 2018
577	28. 04. 2018	–	08. 12. 2018
332	15. 05. 2018	–	08. 12. 2018

Tabulka 2.1: Výpis různých verzí dat linky 1 v datech pro experiment

2.11 Konzole

Jádru aplikace je nastaven příznak aplikace Windows, čímž jsou poskytnuty přídavným modulům rozšířené možnosti pro práci s UI. Tímto příznakem je umožněno libovolnému rozšíření v případě potřeby zobrazovat vlastní formuláře. Zároveň ovšem není možné použít přímo konzolové rozhraní, které se tak musí připojit jako externí proces.

Pro propojení konzole a jádra existuje velké množství způsobů. Protože se připojuje konzole, je takovému přenosu dat nejbližší pojmenovaná roura, nebo síťový localhost callback. Pojmenovaná roura je z těchto dvou výhodnější, protože má menší režijní náklady a nemáme v plánu toto rozhraní využívat po síti.

Pro detekci chybových příkazů je vytvořen stavový automat, který je využit pro zpracování a validaci vstupu jak v rozšíření vytvářejícím rouru, tak v konzolovém procesu. Automat takto poskytne striktní definici příkazu, převod na vytvoření objektů a volání funkcí. Dále umožní vytvořit nápovědu pro uživatele a zabrání napsaní nevalidního příkazu. Přesná gramatika automatu je popsána v příloze B a v příloze A.3 je zobrazena část automat pomocí schématu.

Konzolové příkazy lze nalézt v sekci 5.1.5.

2.12 Mobilní aplikace

Mobilní aplikace je vytvořena především za účelem vizualizace výsledků uživateli a přívětivějšího zadávání požadavku. V aplikaci je možné zobrazit seznam požadavků nebo seznam výsledků. Požadavky je možné editovat, mazat nebo kopírovat. Výsledky je možné zobrazit všechny dohromady nebo pro jedem požadavek pomocí výběru ze seznamu požadavků.

Jednotlivé výsledky mají více úrovní detailu. Výchozí úrovní je seskupený seznam, kdy některé podobné výsledky jsou seskupeny v jeden výsledek. Následuje úroveň výběru jednoho ze seskupených výsledků. Další úrovní je zobrazení více jízd tvořících spojení, kdy pro jízdy je pouze relevantní část trasy. Poslední úrovní je detail jízdy, který zobrazuje všechny zastávky trasy jízdy. Bližší popis viz 5.2.

2.13 Napojení mobilní aplikace

Pro propojení mobilní aplikace s hlavní aplikací byla zvažována možnost přímého síťového připojení nebo využití nějakého serveru jako prostředníka. Ačkoliv se přímé připojení zdá být dobrým řešením, má následující problémy:

- Nelze se spolehnout na neustálou dostupnost obou zařízení, tudíž je potřeba řešit problémy s čekáním na druhé zařízení, což by u mobilní aplikace mohlo vézt k nadměrné spotřebě baterie.

- Musel by se vyřešit problém s neveřejnou IP adresou. Na to existují různé protokoly jako UPnP atd. Ty ale mohou být blokovány sítěmi.

Proto jsme se rozhodli využít nějakého serveru jako prostředníka. Prostředník uchovává zprávy, dokud nejsou smazány příjemcem. Tím je propojení zbaveno uvedených problémů. Toto řešení má jedinou nevýhodu, a to, že musí existovat nějaký prostředník, který bude předávat data.

Nejkritičtější částí je stahování dat na straně mobilu, protože typicky je mobil více omezen než stolní počítač. Naším záměrem bylo využít služby Firebase Cloud Messaging (dále jen FCM), díky níž je vyřešeno automatické stahování dat odeslaných do cloudu, které je optimalizováno pro využití dat, výkon a spotřebu energie. Zprávu do této služby lze předat pomocí HTTP volání. Tak je vyřešeno předání zprávy (resp. dat) z rozšíření hlavní aplikace do mobilní aplikace.

Služba FCM umožňuje i posílání zpráv druhým směrem, kdy ale musí existovat pouze jeden odesílatel (resp. může být odesílatelů více, ale všichni dostanou všechna data jdoucí tímto směrem). Navržená aplikace potřebujeme párování pouze jednoho serveru s jedním mobilem. Druhý směr je pak řešen tak, že se odešle zpráva přes HTTP na server (resp. cloud) a hlavní aplikace se periodicky přes HTTP dotazuje serveru na nové zprávy. Naším záměrem bylo vše implementovat službami poskytnutými Firebase, aby bylo vše na jednom místě. Zde by pro předání zpráv ze serveru do hlavní aplikace bylo vhodnější použití WebSocket, které je duplexní a hlavní aplikace by tak nemusela periodicky dotazovat server. Použití WebSocket pro přístup k uložišti Firestore ovšem není možné vzhledem k vlastní implementaci řízení přístupu.

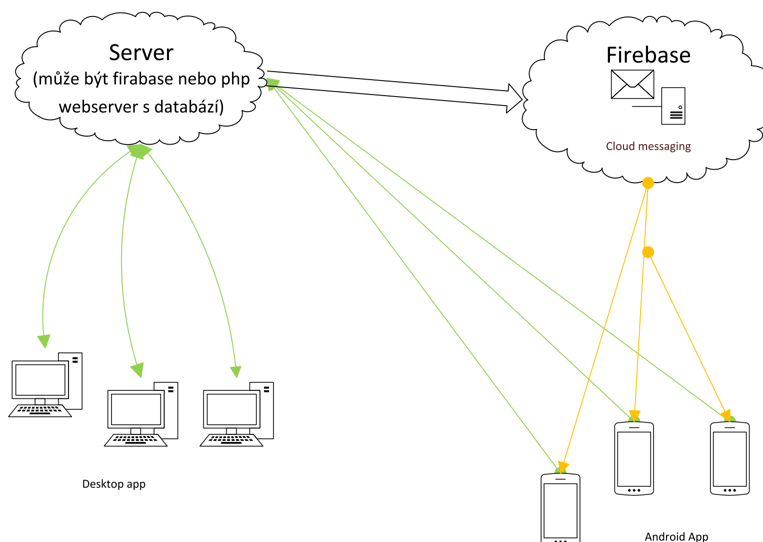
Při tomto návrhu lze snadno vyčerpat limitaci služeb, které poskytuje Firebase zdarma. Jedná se o limity: počet HTTP dotazů, objem uložených dat a velikost zprávy FCM. Proto bylo přesunuto ukládání dat na samostatný server (zaplacený web hosting), který tato omezení pomůže překonat a my budeme využívat pouze FCM a Autentikace Firebase. Pokud bude velikost zprávy přesahovat limit FCM, uloží se data na serveru a aplikace si je stáhne při obdržení zprávy o uložení.

2.13.1 Výsledné propojení mobilní a desktopové aplikace

Výsledné propojení mobilní aplikace s hlavní aplikací Transport Checking System lze rozdělit na tři části (viz obr. 2.2). Za prvé samotná mobilní aplikace zobrazující výsledky a vytvářející požadavky. Za druhé rozšíření k hlavní aplikaci, které odesílá nová data do mobilní aplikace. A za třetí skript (pro server s veřejnou doménou), který tvoří prostředníka pro přenos dat mezi mobilní a hlavní aplikací.

Pro přenos zprávy z hlavní aplikace do mobilní aplikace je odeslána zpráva z hlavní aplikace do prostředníka, který přepošle zprávu do FCM, z nějž si mobilní aplikace ve vhodné chvíli zprávu stáhne. Pro přenos zprávy z mobilní aplikace do hlavní aplikace je zpráva z mobilu odeslána prostředníkovi a hlavní aplikace si zprávu stáhne při nejbližším průchodu periodického cyklu. Odůvodnění tohoto návrhu je v sekci 2.13.

Na obrázku 2.3 jsou jednotlivé části více popsány. Rozšíření hlavní aplikace implementuje rozhraní pluginů pro UI, notifikace a formátovaný výstup. V prostředníkovi jsou zprávy ukládány do databáze a přeposílány do FCM. Mobilní aplikace přijímá zprávy z FCM a ukládá data do lokální databáze SQLite. Mobilní aplikace zobrazuje data z databáze, odesílá do prostředníka nové požadavky a spravuje párování s rozšířením hlavní aplikace. Podrobnější informace jsou k nalezení v příloze E.



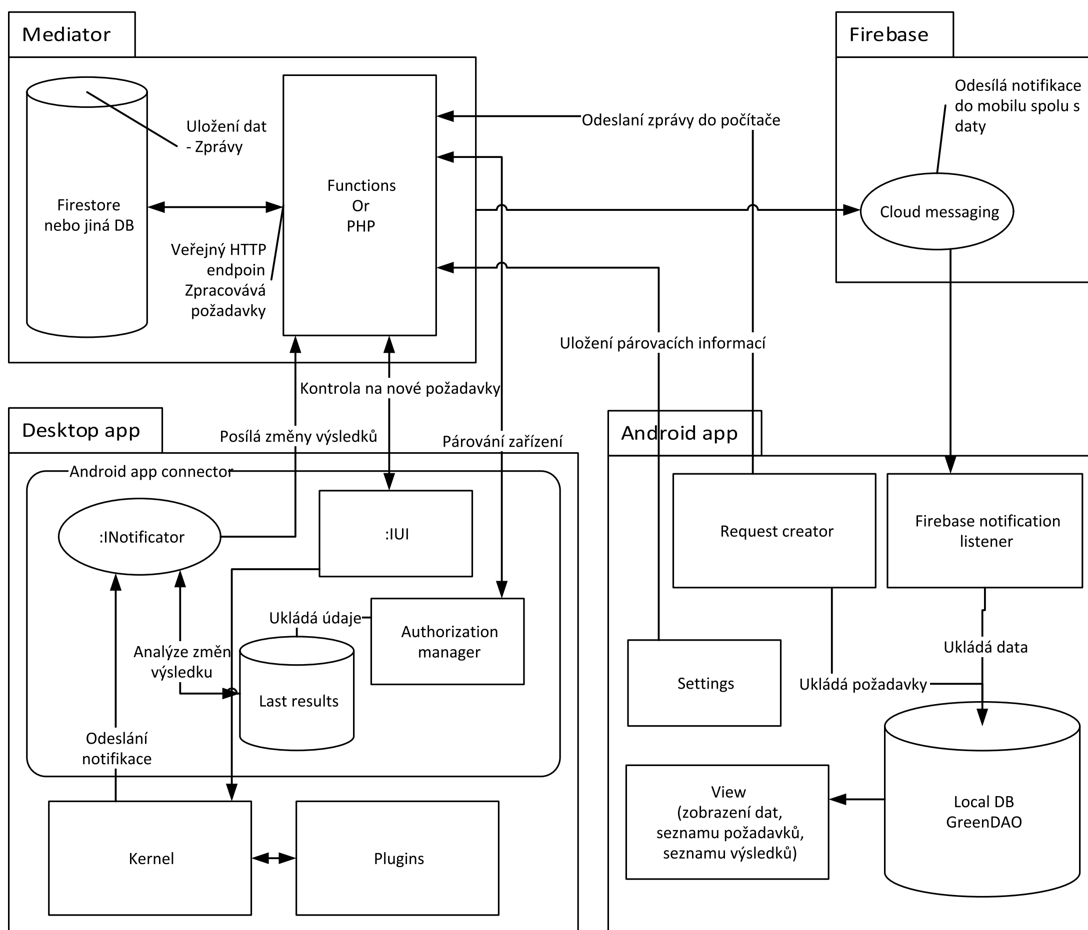
Obrázek 2.2: Rozčlenění aplikace - desktopové aplikace komunikují s prostředníkem, který přeposílá zprávy skrz Firebase cloud messeging do mobilu, a přijímá zprávy z mobilu

2.13.2 Identifikace požadavku

Jádro desktopové aplikace přiřazuje identifikátor ke každému založenému požadavku. Tyto požadavky ovšem nelze po založení upravovat. Z toho důvodu je v mobilní aplikaci a příslušném rozšíření zaveden nový identifikátor požadavku, který přiřazuje mobilní aplikace a je spárován vždy s jedním požadavkem. Při změně požadavku se po zpracování zprávy v rozšíření upraví párování s identifikátorem nově založeného požadavku. Nový identifikátor se odešle do mobilní aplikace až se změnou výsledků.

2.13.3 Synchronizace

Příjem a odeslání zpráv v mobilní aplikaci probíhá dvou fázově.



Obrázek 2.3: Podrobné rozčlenění aplikace

Při přijetí zprávy se příchozí zpráva vloží do fronty *IncomingMessageQueue*. Služba *Importer* následně v případě potřeby stáhne obsah zpráv z prostředníka a zpracuje samostatné zprávy.

Při odeslání se zpráva zařadí do fronty *PostQueue*, kterou zpracuje služba *SenderService*.

2.13.4 Zprávy

Zprávy jsou objekty serializované do JSON. Většina zpráv popisuje změnu (tzn. seznam nových resp. upravených dat a seznam identifikátorů ke smazání), kde se předpokládá, že změna se vztahuje k aktuálnímu stavu lokálních dat. Pokud dojde k inkonzistenci dat v mobilní aplikaci (např. není doručena nějaká zpráva), odešle mobilní aplikace zprávu *reset* do hlavní aplikace. Pokud dojde k inkonzistenci dat v hlavní aplikaci (např. neodpovídají id požadavků), nebo hlavní aplikace přijme zprávu *reset*, odešle novou zprávu *reset* mobilní aplikaci a považuje mobil za čistý (bez dat výsledků, požadavku a základních dat). Hlavní aplikace následně začne odesílat všechna data do mobilní aplikace. Mobilní aplikace při přijetí zprávy *reset* smaže všechna uložená data.

Zprávy jsou čtyř typů:

- Základní data (*BaseDataMessage*)

  - Obsahuje popis změn v seznamu stanic, linek, zdrojů dopravních dat a způsobů notifikace.
- Definice požadavku (*RequestDefinitionMessage*)
  - Obsahuje novou definici požadavku.
- Smazání požadavku (*RequestIdPairDto*)
  - Obsahuje pouze identifikátor požadavku ke smazání
- Změna výsledku požadavku (*ResultChangeMessage*)
  - Obsahuje změny ve výsledcích

3. Vyhledávání

Jednou z hlavních částí jádra je vyhledávání v jízdních řádech, které umožní vyhledávat spoje i v případě nedostupnosti přímého zdroje dat. Díky vlastnímu vyhledávání bude aplikace také schopna najít spoj komplexnějšího zadání.

V této kapitole si uvedeme definice problémů, způsoby modelování jízdních řádů, které jsou převzaty z prací [4, 5].

3.1 Formalizace problému

Základem vyhledávání jsou jízdní řády, z kterých se sestaví graf, nad nímž je provedeno vyhledávání. Jízdní řád se skládá ze *stanic*, *vlaků* [1], časů *příjezdů* a *odjezdů* vlaků ze stanic.

Formálně je jízdní řád tvořen množinou vlaků $\mathcal{Z}$ a množinou stanic $\mathcal{B}$. Dle definice 2 je vlak tvořen posloupností elementárních spojení. Množinou všech *elementárních spojení* dle definice 1 budeme označovat $\mathcal{C}$.

Definice 1. ***Elementárním spojením*** *c se rozumí pětice $c := (Z,S_1,S_2,t_d,t_a)$ reprezentující vlak $Z \in \mathcal{Z}$ odjíždějící ze stanice $S_1 \in \mathcal{S}$ v čase t_d a jeho následující stanice je $S_2 \in \mathcal{S}$ do níž přijede v čase t_a.*

Definice 2. ***Vlak*** *z je konečná posloupnost elementárních spojení $c_1, c_2, ..., c_n$, kdy $\forall i \in [1,n] : c_i = (z, S_i, S_{i+1}, d_i, a_i) \wedge (i = n \vee c_{i+1} = (z, S_{i+1}, S_{i+2}, d_{i+1}, a_{i+1}))$*

Čas odjezdu $t_d(c)$ a příjezdu $t_a(c)$ elementárního spojení $c \in \mathcal{C}$ v rámci dne jsou celá čísla z intervalu [0,1439] reprezentující čas v minutách od půlnoci. *Délka* elementárního spojení označována jako $length(c) = day-diff(t_d(c),t_a(c))$ odpovídá době mezi odjezdem ze stanice a příjezdem do stanice.

3.1.1 Problém nejdřívějšího příjezdu

V naší práci se zabýváme problémem nejdřívějšího příjezdu uvedeném dle definice 4, tento popis problému lze nalézt v [6]. Pro úplnost je v definici 5 uvedena trochu jiná verze problému nejdřívějšího příjezdu z práce [5], kterou označíme jako problém nejrychlejšího spojení. Pro definici obou problémů je potřeba uvést definici 3, která zavádí význam *konzistentního spojení*.

Definice 3. *Nechť $P = (c_1, ..., c_k)$ je posloupnost elementárních spojení spolu s časy odjezdu $dep_i(P)$ a příjezdu $arr_i(P)$ pro všechna elementární spojení $c_i, 1 \leq i \leq k$. Předpokládejme, že časy $dep_i(P)$ a $arr_i(P)$ obsahují také složku dne příjezdu (resp. objezdu), takže čas je definován $t = a \cdot 1440 + b; a \in [0,364], b \in [0,1439]$, kdy a je den a b jsou minuty tohoto dne.*

Posloupnost P je označována ***konzistentním spojením*** *ze stanice $A = S_1(c_1)$ do stanice $B = S_2(c_k)$ pokud splňuje následující:*

- $S_2(c_i) = S_1(c_{i+1})$

[1] Vlaky se rozumí libovolné dopravní prostředky (autobusy, přívozy, metro atd.). Toto označení se používá ve většině textů zabývajících se problémem vyhledáváním cest v jízdních řádech, proto se bude tento text držet stejného označení.

- $dep_i(P) \equiv t_d(c_i) \ (mod 1440)$
- $arr_i(P) = length(c_i) + dep_i(P)$
- $dep_{i+1}(P) - arr_i(P) \geq 0$

Poznamenejme, že v definici 3 předpokládáme možnost přestupu v jakékoliv stanici do jiného vlaku, který odjíždí ze stanice ve stejný čas nebo později, než do stanice přijel. Dále se také předpokládá, že všechny vlaky jsou v provozu každý den. Díky těmto zjednodušením lze uvést v definici 4 *zjednodušený problém nejdřívějšího příjezdu.*

Definice 4. *Mějme dotaz* (A, B, t_0)*, kdy* A *je stanice odjezdu,* B *je cílová stanice a* t_0 *je čas odjezdu. Dále nechť* S *je množina obsahující všechna možná konzistentní spojení z* A *do* B *odjíždějící nejdříve v* t_0*. Označme* $arr(x)$ *čas příjezdu konzistentního spojení do* B*.*

Problémem nejdřívějšího příjezdu *rozumíme nalezení* $c \in S$*, takového že* $\forall o \in S : arr(c) \leq arr(o)$*.*

Definice 5. *Mějme dotaz* (A, B, t_0)*, kdy* A *je stanice odjezdu,* B *je cílová stanice a* t_0 *je čas odjezdu. Dále nechť* S *je množina obsahující všechna možná konzistentní spojení z* A *do* B *odjíždějící nejdříve v* t_0*. Označme* $arr(x)$ *čas příjezdu konzistentního spojení do* B*.*

Problémem nejrychlejšího spojení *rozumíme nalezení* $c \in S$*, takového že* $\forall o \in S : arr(c) \leq arr(o) \wedge dep(c) \geq dep(o)$*.*

Definici 4 si můžeme představit jako problém osoby, která stojí ve stanici A v čase t_0 a hledá spojení, kterým se dostane nejdříve do stanice B. Definici 5 si můžeme představit jako problém osoby, která plánuje trasu z A do B a požaduje, aby trasa byla časově nejkratší.

Implementace vyhledávání výsledku pro problém dle definice 5 se liší použitím Dijkstrova algoritmu oběma směry. Nejprve provede vyhledání času, kdy je možné nejdříve dosáhnout cílové stanice, a následně se provede vyhledání nejpozdějšího odjezdu z počáteční stanice. Oproti získání výsledku pro problém dle definice 4 proběhne pouze vyhledání času dosažení cílové stanice.

My jsme se rozhodli pracovat s problémem dle definice 4, protože budeme provádět sérii vyhledávání pro pokrytí celého časového intervalu a chceme poskytnout uživateli více možností k výběru. Tím nám může vzniknout více spojení, která dorazí do cílové stanice ve stejný čas, ale jsou různě dlouhá. Uživatel tím získá možnost vybrat si spojení, které trvá déle, ale je vhodnější pro aktuální situaci.

Například může uživatel vyžadovat delší přestup v některé ze stanic přestupu, aby si ve stanici stihl nakoupit jídlo. Důvodů pro výběr spojení, které trvá déle, ale končí ve stejný čas může být mnoho. Pokud bude uživatel vyžadovat nejkratší spojení, tak jím je vždy poslední ze spojení se stejným časem příjezdu do cíle.

3.2 Reprezentace dat grafem

Pro modelování jízdních řádů grafem se nejčastěji využívá nějaké varianty ze způsobů *time-dependent* (viz 3.2.3) a *time-expanded* (viz 3.2.2) grafu.

Tyto způsoby se liší především způsobem vytvoření časových závislostí v grafu, kdy časově závislý model přiřazuje hodnoty hranám grafu pomocí funkce závislé na dočasném výsledku, oproti tomu časově expandovaný graf má hrany modelující čekání ve stanicích.

Pro oba způsoby se následně používá pro vyhledávání Dijkstrův algoritmus nebo jeho optimalizované nadstavby.

3.2.1 Příklad jízdního řádu

Pro další použití si uvedeme příklad jízdních řádů definovaný v tabulce 3.1, který nám bude sloužit pro uvedení příkladů *time-expanded* a *time-depended* grafů.

Jízdní řád je tvořen linkami *1, 2, 3, 4*. Linka *1* má dvě jízdy jedoucí po stejné trase. První jízda linky *1* vyjíždí v 0:00 a druhá jízda linky *1* vyjíždí v 0:10.
Linka *1* začíná ve stanici A, jede do stanice B, kam dorazí o pět minut později. Ve stanice B čeká dvě minuty. Následně pokračuje do konečné stanice C, kam dojede po deseti minutách od počáteční stanice.
Linka *2* odjíždí ze stanice D v 0:02, pokračuje do B, v které zastaví 0:06, a končí ve stanici E v 0:08.
Linka *3* odjíždí ze stanice A v 0:01, pokračuje do B, ve které se v 0:05 na minutu zastaví, a končí ve stanici E v 0:08.
Linka *4* odjíždí ze stanice E v 0:08 a přijede do C v 0:09, kde končí. Dále definujme minimální dobu přestupu pro stanice A, B, C, D 2 minuty a pro stanici E 0 minut.

Vlak	**Stanice**	**Událost**	**Čas**
Vlak 1-1	A	odjezd	0:00
Vlak 1-1	B	příjezd	0:05
Vlak 1-1	B	odjezd	0:07
Vlak 1-1	C	příjezd	0:10
Vlak 1-2	A	odjezd	0:10
Vlak 1-2	B	příjezd	0:15
Vlak 1-2	B	odjezd	0:17
Vlak 1-2	C	příjezd	0:20
Vlak 2	D	odjezd	0:02
Vlak 2	B	příjezd	0:06
Vlak 2	B	odjezd	0:06
Vlak 2	E	příjezd	0:08
Vlak 3	A	odjezd	0:01
Vlak 3	B	příjezd	0:05
Vlak 3	B	odjezd	0:06
Vlak 3	E	příjezd	0:08
Vlak 4	E	odjezd	0:08
Vlak 4	C	příjezd	0:09

Pozn: Vlak 1-1 a vlak 1-2 reprezentují stejnou linku, která jede po sobě na stejné trase.

Tabulka 3.1: Příklad jízdního řádu.

3.2.2 Time-expanded model

Časově expandovaný model je založen na orientovaném ohodnoceném časově expandovaném grafu, který je vytvořen dle definice 6. Pro každou událost (příjezd nebo odjezd) ve stanici je vytvořen vrchol grafu, pro každé elementární spojení je přidána hrana mezi vrcholy a jsou vytvořeny hrany mezi vrcholy stejné stanice seřazenými podle časové posloupnosti událostí. Ohodnocení hran odpovídá časové vzdálenosti mezi vrcholy.

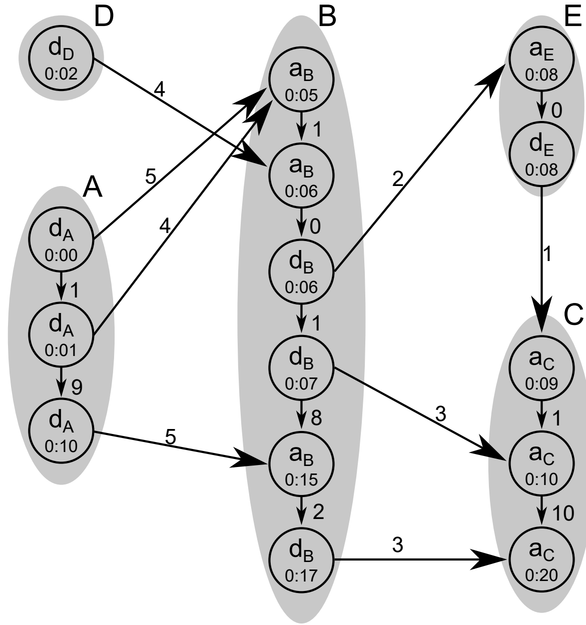
Definice 6. *Mějme množinu stanic S a množinu elementárních spojení C. Vrchol je definován trojicí (ev, s, t), kdy $ev \in$ ‘a‘,‘d‘ je typ události na vrcholu, s je stanice a t je čas události. Dále definujme částečné uspořádání vrcholů $x = (ev_x, s_x, t_x), y = (ev_y, s_y, t_y) \in V : x < y \Leftrightarrow t_x < t_y \vee (t_x = t_y \wedge ev_x = \text{‘}a\text{‘} \wedge ev_y = \text{‘}d\text{‘})$* ***Time-expanded graf*** *$G = (V, E)$ s ohodnocením hran $\omega : E \to \mathbb{R}$ vytvoříme dle pravidel:*

- $\forall (z, S_1, S_2, t_d, t_a) \in C : \begin{cases} \exists s_d = (\text{‘}d\text{‘}, S_1, t_d) \in V \\ \exists s_a = (\text{‘}a\text{‘}, S_2, t_a) \in V \\ \exists e = (s_d, s_a) \in E \\ \omega(e) = ((t_a - t_d) \mod 1440) \end{cases}$
- $\forall s \in S \forall x = (ev_x, s, t_x), y = (ev_y, s, t_y) \in V : x < y \implies ((\exists e = (x,y) \in E \Rightarrow \omega(e) = t_y - t_x) \Leftrightarrow \nexists z = (ev_z, s, t_z) \in V \text{ t.ž. } x < z < y)$

Příklad. Příkladem časově expandovaného grafu dle jízdního řádu z podkapitoly 3.2.1 a definice 6 je graf znázorněný na obrázku 3.1. Vrcholy spadající pod jednotlivé stanice jsou vyznačeny šedým obalem.

Stanice A je reprezentována vrcholy odjezdu $d_A^{0:00}, d_A^{0:01}, d_A^{0:10}$ a hranami čekání $(d_A^{0:00}, d_A^{0:01})$, $(d_A^{0:01}, d_A^{0:10})$ s ohodnocením $\omega(d_A^{0:00}, d_A^{0:01}) = 1$, $\omega(d_A^{0:01}, d_A^{0:10}) = 9$. Stanice B je reprezentována vrcholy příjezdu $a_B^{0:05}, a_B^{0:06}, a_B^{0:15}$, vrcholy odjezdu $d_B^{0:06}, d_B^{0:07}, d_B^{0:17}$ a hranami čekání ve stanici $(a_B^{0:05}, a_B^{0:06})$, $(a_B^{0:06}, d_B^{0:06})$, $(d_B^{0:06}, d_B^{0:07})$, $(d_B^{0:07}, a_B^{0:15})$, $(a_B^{0:15}, d_B^{0:17})$ s ohodnocením $\omega(a_B^{0:05}, a_B^{0:06}) = 1$, $\omega(a_B^{0:06}, d_B^{0:06}) = 0$, $\omega(d_B^{0:06}, d_B^{0:07}) = 1$, $\omega(d_B^{0:07}, a_B^{0:15}) = 8$, $\omega(a_B^{0:15}, d_B^{0:17}) = 2$. Stanice C je reprezentována vrcholy příjezdu $a_C^{0:09}, a_C^{0:10}, a_C^{0:20}$ a hranami čekání $(a_C^{0:09}, a_C^{0:10})$, $(a_C^{0:10}, a_C^{0:20})$ s ohodnocením $\omega(a_C^{0:09}, a_C^{0:10}) = 1$, $\omega(a_C^{0:10}, a_C^{0:20}) = 10$. Stanice D je reprezentována vrcholem odjezdu $d_D^{0:02}$. Stanice E je reprezentována vrcholem příjezdu $a_E^{0:08}$ spojeným s vrcholem odjezdu $d_E^{0:08}$ hranou $(a_E^{0:08}, d_E^{0:08})$ s ohodnocením $\omega(a_E^{0:08}, d_E^{0:08}) = 0$.

Vlak 1-1 je reprezentován hranami $(d_A^{0:00}, a_B^{0:05})$, $(d_B^{0:07}, a_C^{0:10})$ s ohodnocením $\omega(d_A^{0:00}, a_B^{0:05}) = 5$, $\omega(d_B^{0:07}, a_C^{0:10}) = 3$. Vlak 1-2 reprezentují hrany $(d_A^{0:10}, a_B^{0:15})$, $(d_B^{0:17}, a_C^{0:20})$ s ohodnocením $\omega(d_A^{0:10}, a_B^{0:15}) = 5$, $\omega(d_B^{0:17}, a_C^{0:20}) = 3$. Vlak 2 je reprezentován hranami $(d_D^{0:02}, a_B^{0:06})$, $(d_B^{0:06}, a_E^{0:08})$ s ohodnocením $\omega(d_D^{0:02}, a_B^{0:06}) = 4$, $\omega(d_B^{0:06}, a_E^{0:08}) = 2$. Vlak 3 je reprezentován hranami $(d_A^{0:01}, a_B^{0:05})$, $(d_B^{0:06}, a_E^{0:08})$ s ohodnocením $\omega(d_A^{0:01}, a_B^{0:05}) = 4$, $\omega(d_B^{0:06}, a_E^{0:08}) = 2$. Vlak 4 je reprezentován hranou $(d_E^{0:08}, a_C^{0:09})$ s ohodnocením $\omega(d_E^{0:08}, a_C^{0:09}) = 1$.



Obrázek 3.1: Vizualizace příkladu časově expandovaného grafu.

3.2.3 Time-dependent model

Časově závislý model je založen na orientovaném ohodnoceném časově závislém grafu. V tomto grafu je pro každou stanici pouze jeden vrchol a hrany jsou vytvořeny na základě elementárních spojení. Váha hrany je definována funkcí závislou na aktuálním čase s ošetřením přelomu půlnoci. Přesný popis grafu je specifikován v definici 7.

Definice 7. *Mějme množinu stanic S, množinu elementárních spojení C a množinu časů $\mathbb{Z}_{1440}$. Dále mějme částečné uspořádání množiny C definované $x = (z_1,u_1,v_1,t_{dx},t_{ax}), y = (z_2,u_2,v_2,t_{dy},t_{ay}) \in C : x < y \Leftrightarrow t_{dx} < t_{dy} \vee (t_{dx} = t_{dy} \wedge t_{ax} < t_{ay})$.*

Time-dependent graf *$G = (V, E)$ s ohodnocením hran $\omega : E \times \mathbb{Z}_{1440} \to \mathbb{R}$ a funkcí $f : E \to \mathbb{R}$ vytvoříme dle pravidel:*

- $\forall s \in S : s \in V$
- $\forall A,B \in S : \exists e = (A,B) \in E \Leftrightarrow \exists c = (Z, A, B, t_d, t_a) \in C$
- $f(t_d,t_a) = t_a - t_d + \begin{cases} 1440 & t_a < t_d \\ 0 & t_d \leq t_a \end{cases}$
- $\forall (u,v) \in E \; \forall t \in \mathbb{Z}_{1440}$:

$$\omega(u,v,t) = \begin{cases} t_d - t + f(t_d,t_a) & \textit{pokud } \exists (Z,u,v,t_d,t_a) = \\ & \min(\{(Z,u,v,t_1,t_2) \in C : t_1 \geq t\}) \\ f(t,t_d) + f(t_d,t_a) & \textit{jinak } \exists (Z,u,v,t_d,t_a) = \\ & \min(\{(Z,u,v,t_1,t_2) \in C\}) \end{cases}$$

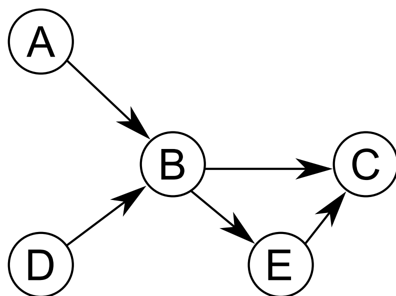
Podmínkou časově závislého modelu je, že pro libovolné stanice A, B neexistují žádné dva vlaky odjíždějící po sobě ze stanice A, takové že přijedou do stanice B v opačném pořadí, než opustily A.

Příklad. Příkladem časově expandovaného grafu dle jízdního řádu z podkapitoly 3.2.1 a definice 7 je graf znázorněný na obrázku 3.2.

Tento graf je tvořen vrcholy A, B, C, D, E, hranami (A,B), (B,C), (B,E), (D,B), (E,C) a ohodnocení hran $\omega(A,B,t) = \begin{cases} 5 & t \in [0,0] \\ 5-t & t \in (0,1] \\ 15-t & t \in (1,10] \\ 1445-t & t \in (10,1440) \end{cases}$,

$\omega(B,C,t) = \begin{cases} 10-t & t \in [0,7] \\ 20-t & t \in (7,17] \\ 1450-t & t \in (17,1440) \end{cases}$, $\omega(B,E,t) = \begin{cases} 8-t & t \in [0,6] \\ 1448-t & t \in (6,1440) \end{cases}$,

$\omega(D,B,t) = \begin{cases} 6-t & t \in [0,2] \\ 1446-t & t \in (2,1440) \end{cases}$, $\omega(E,C,t) = \begin{cases} 9-t & t \in [0,8] \\ 1449-t & t \in (8,1440) \end{cases}$



Obrázek 3.2: Vizualizace příkladu časově závislého grafu.

3.2.4 Reálné modely

V reálném světe vstupují do problému další proměnné, kterými jsou přestupy v rámci stanice (pěší přechody napříč stanicemi), závislost jízdního řádu na dnech, či další kritéria vyhledávání (např. minimální počet přestupů) atd.

Tyto nové proměnné lze dle potřebné úpravy rozdělit na:

- Přidání hran (resp. vrcholů)
  - Do této kategorie spadá zavedení přestupů.
- Odebrání (skrytí) hran
  - Tímto způsobem se vytváří závislost na dnech, nebo filtrace typu dopravního prostředku atd.
- Úpravu ohodnocení hran
  - Pomocí níž je modelována závislost na dnech v časově závislém modelu, nebo změna dotazu na hledání minimálního počtu přestupů.
- Úpravu vyhledávacího algoritmu
  - Slouží například pro dotazování na nejdřívější příjezd s nejpozdějším odjezdem, nebo přidání požadavku na průjezd jednou ze stanic.

Dále se u reálných modelů řeší vyhledávání s více kritérii, kdy hledané spojení musí být optimální dle více kritérií zároveň.

V implementované aplikaci jsme zahrnuli pouze přidání přestupů ve stanicích, úpravu ohodnocení na základě dne a odebírání hran (resp. vrcholů) na základě filtrace. Přidání zbylých úprav může být předmětem budoucího rozšíření systému.

3.2.5 Přestupy

Vzhledem k tomu, že přidání přestupů obnáší zásadní úpravu grafu, tak podrobněji popíšeme úpravy modelů vytvářející přestupy ve stanici.

Při přidání přestupů se dále rozlišují dvě verze složitosti. Konkrétně se zavádí minimální doba pro přestup ve stanici nebo přestupy mezi jednotlivými vlaky (vrcholy různých tras).

Přidání přestupu mezi jednotlivými vlaky je pro implementaci problematické a to ze dvou důvodů. Prvním důvodem je nedostupnost dat pro přestupy mezi jednotlivými zastávkami stanice. Druhým důvodem je ohromné zvýšení počtu hran a vrcholů ve výsledném grafu. Proto tato možnost není v implementaci zohledněna a její implementace je ponechána na rozšíření zdroje dat, které může stanici rozdělit na více stanic a tak simulovat různé časy přestupů.

Time-expanded model

Časově expandovaný graf popsaný v kapitole 3.2.2 lze rozšířit o modelování přestupů dle definice 8.

Pro každý vrchol odjezdu ze stanice je vytvořena kopie, která se označuje jako přestupní vrchol. Hrany čekání nyní budou mezi přestupními vrcholy. Pro každý vrchol příjezdu nyní existuje hrana do vrcholu odjezdu stejného vlaku a hrana do přestupního vrcholu s hodnotou větší nebo rovnou součtu času příjezdu s minimální časem potřebným pro přestup.

Příklad časově expandovaného grafu s přestupy dle jízdního řádu z podkapitoly 3.2.1 a definice 8 je možné si prohlédnout na obrázku 3.3. Vrcholy spadající pod jednotlivé stanice jsou vyznačeny šedým obalem. Ovály jsou označeny vrcholy příjezdu, čekání a odjezdu jednotlivých stanic.

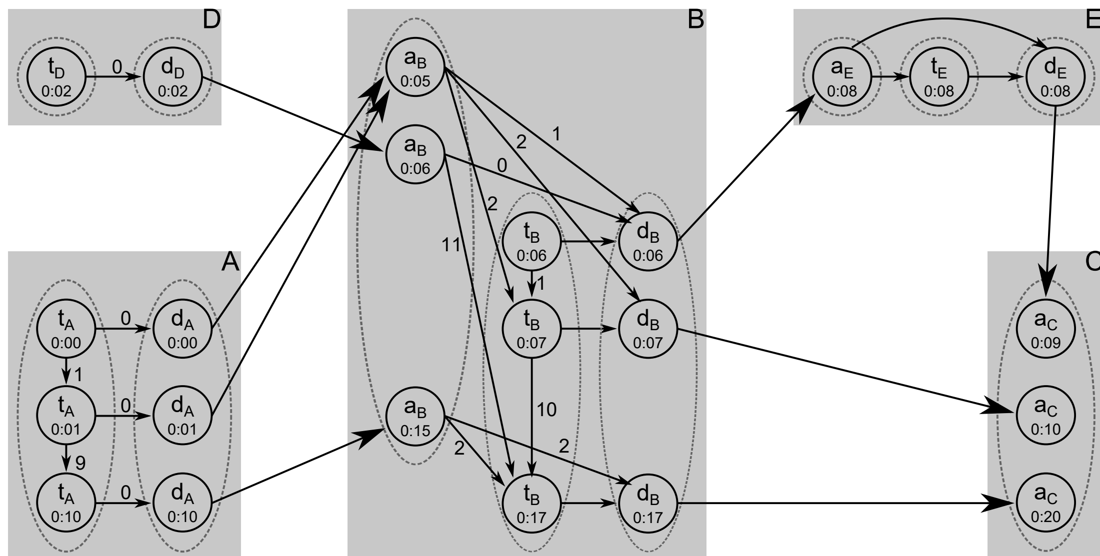
Definice 8. *Nechť G=(V,E) je time-expanded graf s ohodnocením hran ω, množinou stanic S a množinou vlaků Z. Dále mějme částečné uspořádání vrcholů definované $u = (c_u, s_u, t_u), v = (c_v, s_v, t_v) : u < v \Leftrightarrow t_u < t_v$ a funkci $r : S \to R$ definující minimální dobu přestupu ve stanici. Pak graf $G' = (V',E')$ je time-expanded graf rozšířený o přestupy s ohodnocením hran ω', pokud platí:*

- $\forall v = (c,u,t) \in V : \begin{cases} v \in V' \\ v' = (\text{‘}t\text{‘},u,t) \in V' \quad \textit{pokud } c = \text{‘}d\text{‘} \end{cases}$
- $\forall u = (\text{‘}d\text{‘}, s_u, t_u), v = (\text{‘}a\text{‘}, s_v, t_v) \in V\ s_u \neq s_v\ (u,v) \in E : (u,v) \in E'$
- $\forall u = (\text{‘}t\text{‘}, s, t), v = (\text{‘}d\text{‘}, s, t) \in V' : (u,v) \in E'$
- $\forall u = (\text{‘}a\text{‘}, s, t_a) \in V'\ \exists v = (\text{‘}t\text{‘}, s, t_d) = \min(\{(\text{‘}t\text{‘}, s, t_d) : t_a + r(s) < t_d\}) \in V' : (u,v) \in E'$

- $\forall z = (c_1,...,c_n) \in Z \; \forall i \in [1,n-1] \; c_i = (z,s_i,s_{i+1},d_i,a_i), c_{i+1} = (z, s_{i+1}, s_{i+2}, d_{i+1}, a_{i+1}) : \exists u = (‘a‘,s_{i+1},a_i), v = (‘d‘,s_{i+1}, d_{i+1}) \in V' \wedge \exists (u,v) \in E'$

- $\forall u = (‘t‘, s, t_u), v = (‘t‘, s, t_v) \in V'(t_u < t_v) : (u,v) \in E' \Leftrightarrow \nexists w = (‘t‘,s,t_w) \in V' \, (t_u < t_w < t_v)$

- $\forall u = (c_u, s_u, t_u), v = (c_v, s_v, t_v) \in V'(u,v) \in E' : \omega'(u,v) = t_v - t_u \mod 1440$



Obrázek 3.3: Vizualizace příkladu časově expandovaného grafu s přestupy.

Time-dependent model

Časově závislý graf s přestupy se vytvoří dle definice 10 následovně. Vytvoříme vrchol pro všechny stanice a zastávky všech tras. Následně spojíme zastávky jednotlivých tras hranami s časově závislou funkcí. Přidáme hrany vedoucí z vrcholů zastávek do vrcholů korespondující stanice s nulovým ohodnocením a hrany s opačným směrem s ohodnocením odpovídajícím minimální době přestupu ve stanici.

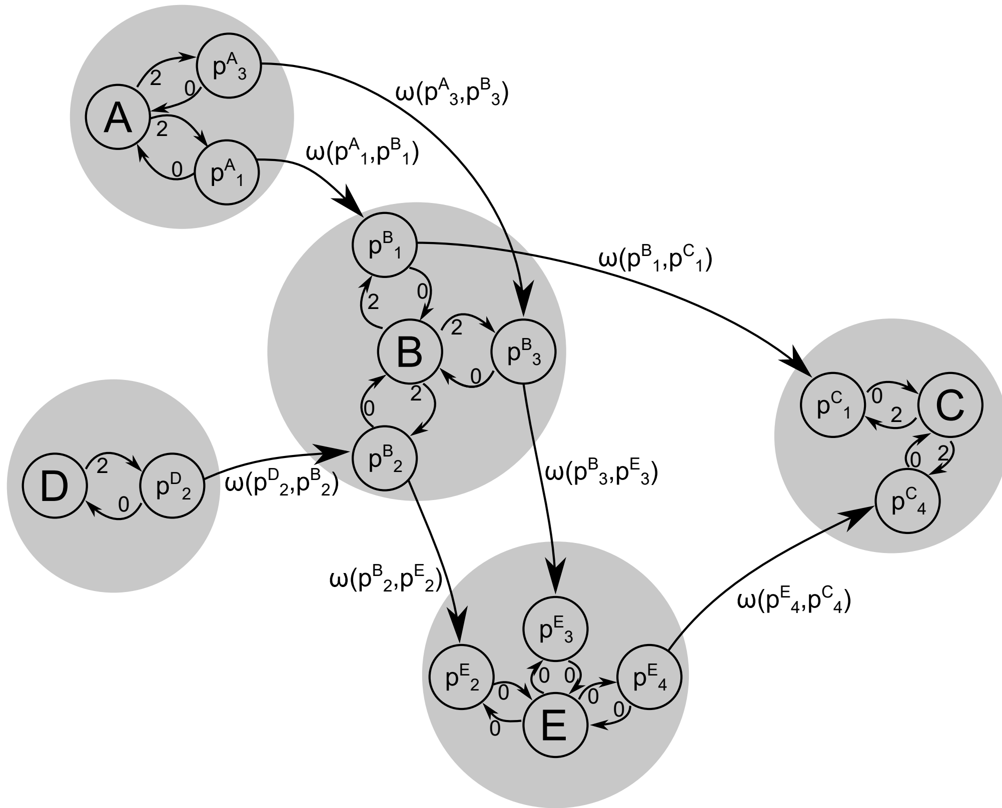
Příklad takového grafu dle jízdního řádu z podkapitoly 3.2.1 a definice 10 je zobrazen na obrázku 3.4. Vrcholy spadající pod jednotlivé stanice jsou vyznačeny šedým obalem.

Definice 9. *Nechť Z je množina všech vlaků.* ***Trasa*** *r je konečná posloupnost stanice $s_1, s_2, ..., s_n$, pro kterou platí: $\exists r = (s_1, ..., s_n) \Leftrightarrow \exists z = (c_1, ..., c_n) \in Z \wedge \forall i \in [1,n] : c_i = (z, s_i, s_{i+1}, d_i, a_i)$. Dva vlaky u, v mají stejnou trasu r pokud $r_u = (s_1, ..., s_n) = r_v = r$.*

Definice 10. *Nechť S je množina stanic, R množina cest a funkce $g : S \to R$ definuje minimální dobu přestupu ve stanici. Pak graf $G = (V,E)$ je time-dependent graf rozšířený o přestupy s ohodnocením hran ω, pokud platí:*

- $\forall s \in S : \exists v = (s,\emptyset) \in V$

- $\forall r = (s_1,...,s_n) \in R \forall i \in [1,n] : \exists v = (s_i, r) \in V$
- $\forall r = (s_1,...,s_n) \in R \forall i \in [1,n-1] : \exists e = ((s_i, r), (s_{i+1}, r)) \in E$
- $\forall r = (s_1,...,s_n) \in R \forall i \in [1,n] : \begin{cases} \exists e = ((s_i, r), (s_i, \emptyset)) \in E & \omega(e,t) = g(s_i) \\ \exists e = ((s_i,\emptyset), (s_i, r)) \in E & \omega(e,t) = 0 \end{cases}$
- $f(t_d,t_a) = t_a - t_d + \begin{cases} 1440 & t_a < t_d \\ 0 & t_d \leq t_a \end{cases}$
- $\forall e = ((u, r_1), (v, r_2)) \in E \; u \neq v :$
$$\omega(e,t) = \begin{cases} t_d - t + f(t_d, t_a) & \textit{pokud } \exists(Z, u, v, t_d, t_a) = \\ & \min(\{(Z, u, v, t_1, t_2) \in C : t_1 \geq t\}) \\ f(t,t_d) + f(t_d, t_a) & \textit{jinak } \exists(Z, u, v, t_d, t_a) = \\ & \min(\{(Z, u, v, t_1, t_2) \in C\}) \end{cases}$$



Obrázek 3.4: Vizualizace příkladu časově závislého grafu s přestupy.

3.2.6 Denní periodicita

Linky (resp. jejich jízdy) mají obvykle specifikováno, které dny v týdnu fungují. Toto omezení se v časově expandovaném modelu implementuje skrytím hran a lehkou modifikací algoritmu, která skrývá a odkrývá hrany při překročení půlnoci. V časově závislém modelu je pro toto omezení potřeba upravit výpočet ohodnocující funkce. Pokud jsou implementovány úpravy pro denní periodicitu, může být potřeba úpravy zohlednit při aplikaci optimalizací algoritmu.

3.2.7 Implementovaný model

Při volbě mezi časově expandovaným a závislým přístupem jsme se řídili podle prací [4, 7, 8], které se shodují na tom, že ačkoliv jsou oba přístupy na sebe převeditelné, ukazuje se experimentálně, že časově závislý přístup je rychlejší, ačkoliv se tato skutečnost těžko dokazuje (např. protože není stanoveno, jak má být implementován výpočet časově závislé funkce). Z tohoto důvodu jsme zvolili časově závislý přístup. Při implementaci jsme ovšem tento model upravili, abychom dosáhli některých zlepšení a vyhnuli se jistým problémům.

Trasa

První z úprav, kterou jsme udělali, byla definice trasy. V podkapitole 3.2.3 jsme uvedli podmínku časově závislého modelu, která zakazuje existenci dvou po sobě odjíždějících vlaků ze stanice A přijíždějících do stanice B v opačném pořadí. Tato podmínka je v případě časově závislého grafu s přestupy omezena na neexistenci dvou vlaků stejné trasy. Abychom předešli problémům, upravili jsme význam trasy definicí 11. Ta nám zároveň umožní sjednocení výpočtu časové funkce pro všechny hrany trasy na výpočet uvedený v definici 12.

Definice 11. *Nechť Z je množina všech vlaků.* ***Trasa*** *r je konečná posloupnost stanice $s_1, s_2, ..., s_n$, pro kterou platí: $\exists r = (s_1, ..., s_n) \Leftrightarrow \exists z = (c_1, ..., c_n) \in Z \wedge \forall i \in [1,n] : c_i = (z, s_i, s_{i+1}, d_i, a_i)$.*

Nechť pro elementární spojení $c = (v, s_1, s_2, t_d, t_v)$ jsou definovány funkce $d(c) = t_d$, $a(c) = t_a$, $s_d(c) = s_1$ a $s_a(c) = s_2$. Dva vlaky u, v mají stejnou trasu r pokud $u = (c_1^u, ..., c_n^u), v = (c_1^v, ..., c_n^v)$ platí $\forall i \in [1,n] : a(c_i^u) - d(c_i^u) = a(c_i^v) - d(c_i^v) \wedge s_d(c_i^v) = s_d(c_i^u) \wedge s_a(c_i^v) = s_a(c_i^u)$ a $\forall i \in [1,n-1] : d(c_{i+1}^u) - a(c_i^u) = d(c_{i+1}^v) - a(c_i^v)$.

Definice 12. *Nechť Z_r je množina vlaků trasy r, $Z_{c_1} = \{c_1 | (c_1, ..., c_n) \in Z_r\}$ a funkce $l(r,v)$ je konstantní vzdálenost vrcholu v na trase r od počátku trasy, pak pro hranu (u,v) na trase je $\omega(u,v,t) = \min(\{t_d | (l,s_1,s_2,t_d,t_a) \in Z_{c_1}\, t_d \geq t - l(r,u)\}) + l(r,v)$*

Tvrzení 1. *Definice 11 zachovává korektnost algoritmu vyhledávání.*

Důkaz. Nechť Z je množina vlaků, S je množina stanic a G je time-dependent graf pro Z, S. Na základě definice 11 mějme množinu R různých tras. Definujme bijektivní zobrazení $i : R \to [1, |R|]$ přiřazující identifikátor jednotlivým trasám. Vytvořme množinu $Z' = \{(c_1,...,c_n,(z,s_d(c_n), i(r),t_a(c_n), 0)) \,|\, c = (c_1,...,c_n) \in Z$ a r je trasa $c\}$ vlaku a množinu $S' = S \cup \{1, ..., |R|\}$. Pro time-dependent graf jízdního řádu Z', S' s definicí trasy 9 je algoritmus korektní na základě [4]. Takový graf bude po odebrání vrcholů $\{1, ... |R|\}$ ekvivalentní grafu G. Tudíž úprava definice nemá vliv na korektnost algoritmu.

□

Přidání hran na trasách

Pro optimalizaci řešení jsme přidali do modelu heuristiku, která by nám měla pomoci vyhnout se zbytečným přestupům. Zároveň nám umožní odstranit přestupní hrany a vrcholy, čímž sníží prohledávaný prostor a celkový čas vyhledávání.

Heuristika je vytvořena přidáním hran pro cestu $r = (s_1,...,s_n)$ tak, že $\forall i,j : 0 \leq i < j \leq n \implies (s_i, s_j) \in E$. Tyto nové hrany ovšem do grafu nepřidáme, ale spočítáme (projdeme) je za běhu. Tato úprava nám vytváří asymptoticky kvadratickou složitost výpočtu v nejhorším případě. My ovšem očekáváme lepší výsledky, než při implementaci normálního modelu, z důvodu odebrání přestupních hran (resp. vrcholů) a skutečnosti, že zpracování nových hran je méně náročné, než zpracování skutečné hrany (náročná část výpočtu ohodnocující funkce se provede pouze jednou). Zároveň očekáváme, že tato heuristika nalezne lepší výsledky v rámci kritéria přestupů[2] pro případy podobné příkladu 3.2.7.

Příklad. Předpokládejme existenci stanic D, A, T, A_d, A_t a předpokládejme vlaky $z_1 = ((z,D,s_1,t_d,t_1), ..., (z, ,s_2, a_d,t_2, t_{A_d}), (z,A_d,A,t_3, t))$, $z_2 = ((z,D,s_3,t + p,t_5),..., (z,s_4,T, t_6, t_T))$, $z_3 = ((z,T,s_5,t_7,t_8), ...(z,s_6,A_t,t_9,t_{A_t}),(z, A_t, A, t_{10},t))$. V takovém jízdním řádu existují dvě možná spojení mezi stanicemi D, A. Jedno spojení je tvořeno vlakem z_1. Druhé spojení je tvořeno vlaky z_2, z_3 s přestupem v T. Stanice A_d je předposlední stanice vlaku z_1 a stanice A_t je předposlední stanice vlaku z_3. Pokud navíc $t_{A_d} > t_{A_t}$ pak Dijkstrův algoritmus nalezne spojení (z_2,z_3) s více přestupy.

3.3 Algoritmus vyhledávání

Nyní máme definován problém jako hledání nejkratší cesty v kladně ohodnoceném grafu. Tento problém řeší Dijkstrův algoritmus nebo jeho rozšířené verze (např. A*). My jsme se rozhodli použít Dijkstrův algoritmus, který jsme doplnili o výpočet ohodnocující funkce a simulaci hran uvedené v 3.2.7.

3.3.1 Prioritní párovací fronta

Jak zmiňuje [4], pro uchování nezpracovaných vrcholů v algoritmu je vhodné využít prioritní frontu. V pracích [4, 6] je zmíněno pro uložení nezpracovaných vrcholů využití implementace Fibonacciho haldy, která zlepšuje asymptotickou složitost z $\mathcal{O}(|V|^2)$ na $\mathcal{O}(|E| + |V| \cdot \log(|V|))$ (viz [9]).

My jsme se rozhodli pro tuto funkci implementovat párovací haldu, která je snazší na implementaci a při experimentálním srovnání v práci [10] se ukazuje být pro Dijkstrův algoritmus rychlejší přesto, že jejím použitím se asymptotická složitost Dijkstrova algoritmu zhorší na $\mathcal{O}((|E| + |V|) \cdot \log(|V|))$.

3.3.2 Navržený algoritmus

Nejprve si uvedeme počátek algoritmu (viz kód 3.2), funkci *vyhledej* (viz kód 3.3) a funkci *uprav* (viz kód 3.4), které odpovídají standardní implementaci Dijkstrova algoritmu s prioritní frontou. Následuje funkce *relax* (viz kód 3.5), v níž se počítá ohodnocení hrany a případně volá funkce *uprav*.

Ve funkcích budeme využívat globálních proměnných $h(v), Q, F, B(v)$, kdy $h(v)$ je ohodnocení vrcholu (čas dosažení), Q je prioritní fronta ke zpracování, která udržuje na svém začátku vždy vrchol s nejmenším *h(v)*, F je množina všech

[2]Poznamenejme, že při použití vyhledávání s více kritérii se hovoří o asymptoticky exponenciální složitosti

zpracovaných vrcholů a $B(v)$ uchovává pro vrchol v informace o dosažení vrcholu ve struktuře *záznam*.

```
záznam
{
  předchozíVrchol
  aktuálníVrchol
  trasa
  vzdálenost
}
```

Kód 3.1: Struktura záznamu

Struktura *záznam* (viz kód 3.1) pro vrchol v se skládá z identifikace vrcholu u ($záznam.předchozíVrchol$), z kterého byl vrchol v dosažen v čase $h(v)$, reference na trasu $záznam.trasa$, jejíž částí je hrana (v,u), identifikace vrcholu v ($záznam.aktuálníVrchol$) a vzdálenosti v ($záznam.vzdálenost \approx h(v)$).

```
  Vstup: Graf G, počáteční vrcholy S, konečné vrcholy T
  Výstup: Spojení z S do T
1   Pro všechny vrcholy x z S:
2     h(x) ← 0
3     vložíme vrchol x do Q
4   vyhledej(G)
5   Sestav výsledek podle B
```

Kód 3.2: Počátek algoritmu

Při spuštění algoritmu (viz kód 3.2) se provede inicializace počátečními vrcholy, následně je provedeno samotné vyhledávání a nakonec je sestaven výsledek podle záznamu o dosažení vrcholů v B.

```
  vyhledej()
1   Dokud Q není prázdná:
2     x ← odeber z fronty Q
3     přidej x do F
4     pokud x je mezi T ukonči vyhledávání
5     pro všechny následníky w vrcholu x:
6       pokud w není v F:
7         relax(x, w)
```

Kód 3.3: Funkce vyhledej

Vyhledávání (viz kód 3.3) tvoří cyklus, který vždy odebere z fronty ke zpracování vrchol s nejmenším časem dosažení, který přidá mezi zpracované vrcholy, a pro všechny nezpracované sousedy provede *relax*.

```
  Parametry: f je zdrojovým vrcholem, t je cílovým vrcholem hrany (f, t),
  r je cesta, na níž se hrana nachází, d je vzdálenost od počáteční stanice
  uprav(f, t, r, d)
1   pokud d < h(t) nebo h(t) není definováno
2     přidej t do Q, nebo uprav jeho zatřídění, pokud již v Q existuje
3     uprav B(t):
4       B(t).předchozíVrchol ← f
5       B(t).aktuálníVrchol ← t
6       B(t).trasa ← r
7       B(t).vzdálenost ← d
8     h(t) ← d
```

Kód 3.4: Funkce uprav

Funkce *uprav* (viz kód 3.4) v případě menšího času dosažení upraví záznam o dosažení vrcholu a vzdálenost vrcholu od počátku tak, aby hodnoty popisovaly časově nejkratší cestu pro dosažení vrcholu.

```
    Parametry: f je zdrojovým vrcholem a t je cílovým vrcholem hrany (f, t)
    relax(f, t)
1     d ← h(f)
2     r ← trasa na níž je (f, t)
3     opusť funkci, pokud B(f).trasa == r
4     pokud f není počáteční stanicí:
5       d += minimální čas přestupu ve stanici f
6     x ← čas počátku vyhledávání + d
7     y ← první vlak jedoucí po r odjíždějící z f po čase x
8     opusť funkci, pokud byl během vyhledávání použit vlak y, nebo vlak jedoucí po stejné
          trase dříve s přestupem ve stanici předcházející f na trase r
9     w ← (čas odjezdu vlaku y ze stanice f) − x
10    d += w
11    pro všechny body trasy n na trase r následující po f → t
12      pokud n není v F:
13        nd = d + vzdálenost n od f
14        uprav(f, n, r, nd)
```

Kód 3.5: Funkce relax

Funkce *relax* (viz kód 3.5) pro hranu (f, t) spočítá čas možného nástupu na vlak trasy jejíž je hrana součástí. Následně je nalezen první vlak jedoucí na této trase skrze stanici f po čase možného nástupu. Pro všechny stanice n na trase tohoto vlaku po stanici f je spočítán čas, kdy vlak do stanice dorazí, a ověřeno, zda tento čas není vylepšujícím dosažením stanice n. Zároveň jsou ve funkci dvě kontroly, které přeruší výpočet v případě, že nemůže být použitím hrany (f, t) spočítán vylepšující výsledek.

Lemma 2. *Časově závislý výpočet času dosažení vrcholu je nezáporná neklesající funkce.*

Důkaz. Z algoritmu můžeme vyčíst výpočet $f(u,v,h(u)) = h(u)+k+w(u,h(u),v,s)$ $+c(u,v,s)$, kdy $h(u)$ je čas dosažení vrcholu u (neklesající funkce), $c(u,v,s)$ je konstantní nezáporná vzdálenost mezi vrcholy na trase, k je nezáporná konstantní doba přestupu a $w(u,h(u),v,s)$ je doba čekání na vlak jedoucí po trase s.

Vzhledem k tomu, že funkce w nemůže nabýt záporných hodnot, je funkce f neklesající.

□

Tvrzení 3. *Uvedený upravený Dijkstrův algoritmus nalezne nejkratší cestu v grafu s ohodnocením časově závislou funkcí.*

Důkaz. Na základě lemma 2 můžeme využít důkazu použitého v [9, str. 151-152]. □

Lemma 4. *Relax řeší korektně problém s přestupy*

Důkaz. Mějme stanici a, kterou zpracováváme, a stanici c do níž vede hrana z a. Do a jsme se dostali z b v čase t po trase r.

Rozeberme všechny možné případy, a ukažme, že budou algoritmem správně zpracovány.

1. Do c vede nejrychlejší cesta s přestupem v a po trase p.

   - Při zpracování není funkce *relax* opuštěna v kroku 3, protože pokud by byla trasa stejná, jedná se o případ 2.
   - Při zpracování není funkce *relax* opuštěna v kroku 8. Pokud by funkce byla opuštěna, tak by to znamenalo, že existuje cesta s přestupem ve stanici d na trase p předcházející a, která dosáhne c ve stejném, nebo dřívějším časem. Což je stejně rychlá nebo rychlejší cesta, což odporuje definici případu.
   - Volání funkce *update* nebude vyloučeno v kroku 12, protože jinak by existovala trasa, která dosáhne c s časem $t'<t$, což odporuje definici případu, protože funkce přestupu a čekání nemůže být záporná, tudíž čas dosažení c musí být větší než t.
   - Je zřejmé, že časově závislá funkce je správně spočítána ve funkci *relax*.
   - Tudíž je tato cesta uložena ve funkci *update*

2. Do c vede nejrychlejší cesta po trase r.

   - V tomto případě byla volána funkce *update*, při zpracování přestupu ve stanici b, tudíž je již cesta zaznamenána a není nutné znovu počítat. Je tedy opuštění v kroku 3 v pořádku.

3. Do c vede nejrychlejší cesta projíždějící a po trase s taková, že vlak jedoucí po této trase přijede do a v čase $u > t$ (jinak by se jednalo o případ 1) a odjede z a v čase $v < t +$ *čas přestupu*.

   - To by znamenalo, že existuje stanice x předcházející a na trase s. Při zpracování stanice x, které může zpracovaní stanice a předcházet, nebo jej následovat, bude zpracována v rámci cyklu v kroku 11 dvojce *(x,c)*. Tudíž, bude dosažení stanice upraveno funkcí *update*.

Vzhledem k tomu, že toto jsou všechny možnosti, které mohou nastat, můžeme říci, že funkce *relax* korektně zpracovává problém s přestupy.

□

Tvrzení 5. *Uvedený upravený Dijkstrův algoritmus řeší problém nejdřívějšího příjezdu v časově závislém modelu s přestupy.*

Důkaz. Vzhledem k tvrzení 3 a lemma 4 je zřejmá pravdivost tohoto tvrzení.

□

3.3.3 Průchod týdnem

Uvedený algoritmus nám vyhledá nejrychlejší spojení v daný den a s odjezdem (resp. příjezdem) v zadaný čas. Požadavek na spojení ve smyslu naší aplikace vyžaduje všechna spojení odjíždějící z počáteční stanice (resp. přijíždějící do cílové stanice) ve zvoleném časovém intervalu a vybraných dnech. Aplikace musí provést více vyhledávání pro průchod časového intervalu a vybraných dnů.

4. Experimentální srovnání uvedeného řešení

V sekci 3.3 byl uveden návrh úpravy algoritmu, který by měl být při porovnání s původním algoritmem v běžném případě rychlejší, méně náročný na paměť a který by měl nalézt z lidského hlediska lepší výsledky vzhledem k přestupům. Pro ověření očekávání je navržen následující experiment.

4.1 Spuštění experimentu

Pro experiment je v elektronické příloze připraven projekt *AlgorithmPerformenceExperiement*. V experimentu je náhodně vytvořeno 120 požadavků na spojení s časovým intervalem odjezdu v rozmezí jedné hodiny. Časové intervaly jsou generovány napříč celým dnem. Startovní a cílové stanice jsou vybrány náhodně z dostupných stanic.

Data pro experiment jsou publikované JDF[1] soubory z 4. června 2018. Pro experiment je následně nutné vytvořit lokální ftp server, tam soubory přidat a v konfiguračním souboru v experimentálním projektu změnit url zdroje dat na lokální ftp server. Zároveň je nutné zrušit kontrolu na existenci linky nastavením hodnoty *LineExistanceCheckerUrl* v konfiguračním souboru na prázdnou hodnotu.

V projektu je porovnávána navržená implementace s implementací originálního časově závislého grafu s konstantními přestupy. Formát dat a výpočet ohodnocující funkce jsou pro obě implementace v rámci možností ekvivalentní.

Při spuštění experimentu je postupně spuštěno vyhledávání vytvořených požadavků. Pro jeden požadavek proběhne vyhledávání několikrát, tak aby prošlo vybraný časový interval pro celý týden. Vzhledem k tomu, že data jsou načítána do vyhledávače až v případě potřeby, je rozlišováno první vyhledávání pro požadavek a ostatní vyhledávání, protože transformace a načtení většiny potřebných dat proběhne při prvním vyhledání.

4.2 Prostředí experimentu

Experiment proběhl na počítači s OS Windows 10 Pro, procesorem Intel Core i7-4700MQ 2,4 GHz, operační pamětí 16 GB a SSD diskem Samsung 850 EVO.

4.3 Výsledky experimentu

V experimentu byly sledovány tyto hodnoty:

- Doba, za kterou byl nalezen výsledek pro první vyhledávání v rámci požadavku a doba zbylých vyhledávání výsledku pro požadavek.
- Počet přestupů v nalezených spojeních.

[1]Přiložené k práci

- Počet uzlů grafu načtených ve vyhledávací třídě.
- Počet volání funkce *uprav*, což by mělo odpovídat počtu využitých hran grafu.

Výsledné hodnoty lze nalézt v tabulce 4.1. Hodnoty uvedené v tabulce jsou zaokrouhleny na dvě desetinná místa nebo celé číslo.

Podrobný seznam naměřených hodnot se nachází v elektronické příloze.

4.3.1 Vyhodnocení

Jak lze pozorovat z naměřených hodnot, první běh vyhledávání je téměř totožný a bude záviset především na načítání dat. U ostatních vyhledávání, která v běžné situaci načítají řádově menší množství dat, lze pozorovat, že upravený algoritmus je ve většině případů až dvakrát rychlejší. Očekávané důsledky navržené úpravy byly naplněny výsledky s menším počtem přestupů a menším počtem uzlů grafu, čímž je sníženo využití paměti. To je ovšem kompenzováno výrazně vyšším počtem prošlých hran. Zde je nutné si uvědomit, že náročnost zpracování jedné hrany u upraveného algoritmu je nižší, jelikož náročný výpočet čekání ve stanici proběhne jednou pro několik hran.

Do budoucna by bylo vhodné aplikovat některé optimalizace časově závislého modelu na upravený model a tyto dva optimalizované modely znovu porovnat.

Zároveň je z tabulky 4.1 vidět, že velké množství času je v průběhu algoritmu stráveno načítáním dat, což indikuje místo vhodné pro vylepšení. Zde je ovšem problém, že doba strávená načítáním se skládá ze získání dat z přidaného rozšíření, které potencionálně není možné ovlivnit, kontroly správnosti vrácených dat a transformace dat (především filtrace). Výpočetně nejnáročnější je z těchto částí získání dat v přidaném rozšířením, které může implementovat neovlivnitelná třetí strana.

	Střední hodnota	**Směrodatná odchylka**	**Min**	**Medián**	**Max**
Upravený algoritmus					
První vyhledání	6,67s	2,13s	0,23s	7,6s	9,98s
Ostatní vyhledání	37,81ms	145,22ms	1,79ms	26,68ms	7,23s
Počet přestupů	3,45	1,16	1	3	8
Počet uzlů	2703	608	567	2938	3245
Volání update	382462	104397	22901	424327	485667
Originální algoritmus					
První vyhledání	7,05s	1,87s	0,53s	7,84s	8,42s
Ostatní vyhledání	118,14ms	140,47ms	3,89ms	117,17ms	7,24s
Počet přestupů	4,73	1,96	1	4	12
Počet uzlů	86147	27121	3837	95356	112887
Volání update	123081	39525	4906	136534	162541

Tabulka 4.1: Naměřené hodnoty experimentu

5. Implementace

5.1 Desktopová aplikace

5.1.1 K čemu je aplikace určena

Hlavním úkolem aplikace je včas upozornit uživatele na změnu hromadné dopravy, kterou často využívá. Příkladem může být každodenní trasa z domova do práce a zpět, nebo odjezdy ze stanice poblíž domova atd. Specifikace oblastí zájmu uživatele jsou zadány do aplikace ve formě požadavků. Aplikace v uživatelem zadaném intervalu kontroluje, zda se nezměnil výsledek některého požadavku. Pokud dojde v hromadné dopravě ke změně, která ovlivní výsledek pro některý z požadavků, oznámí aplikace uživateli tuto skutečnost způsobem, který si zvolil.

5.1.2 Jak vypadá hromadná doprava z pohledu systému

Pro správné využití systému (především vytváření kontrolovaných požadavků) je třeba specifikovat model hromadné dopravy.

Základem dopravního systému jsou stanice, linky, jízdy a trasy.

Stanice jsou místa, kde je možné nastoupit či vystoupit z dopravního prostředku.

Linka je označení několika jízd, které spolu blízce souvisejí.

Jízda představuje jeden průjezd určitou trasou s určitým počátečním časem.

Trasa je posloupnost stanic, po které jede jízda. Stanice mohou mít v rámci trasy rozšiřující parametry (např. ve stanici staví jízda jen po znamení řidiči).

5.1.3 Co se stane po spuštění

Po spuštění souboru *TransportSystemKernel.exe* se vytvoří proces běžící na pozadí. Tento proces se nalézá v systému pod jménem *TransportSystemKernel.*

Proces následně zkontroluje, zda je na lokálním počítači dostupná databáze *(LocalDB)\MSSQLLocalDB.* Pokračuje načtením pluginů ze složky *Plugins.*

Poté se spustí uživatelské rozhraní, které je specifikováno prvním parametrem předaným při startu *TransportSystemKernel*, nebo se vybere první načtené rozšíření, které poskytuje uživatelské rozhraní. Navíc se vždy spustí všechna uživatelská rozhraní specifikována v konfiguračním souboru. Pokud není nenalezeno žádné uživatelské rozhraní, aplikace poběží kompletně na pozadí bez možnosti jejího ovládání.

Nakonec se spustí smyčka, která vždy zkontroluje všechny požadavky a na uživatelem zadaný čas (viz 5.1.4) se uspí. Tato smyčka se přeruší příkazem z uživatelského rozhraní (viz 5.1.4).

Důležitá upozornění

Rozšíření aplikace se načítají pouze při startu aplikace, tudíž je třeba po přidání rozšíření do složky *Plugins* aplikaci ukončit (viz 5.1.4) a následně ji znovu spustit. Pro změnu uživatelského rozhraní není restartování aplikace nutné. Další

uživatelské rozhraní lze spustit z již existujícího. Po přidání požadavku ke kontrole se spustí prvotní kontrola pro získání výsledku. V momentě, kdy aplikace získá prvotní výsledek, sdělí tuto skutečnost uživateli skrze všechna spuštěná UI. Může se stát, že některá rozšíření třetích stran budou způsobovat chyby systému. V takových případech nemusí správně proběhnout kontrola požadavku, který toto rozšíření používá. V krajních případech může taková chyba způsobit selhání celé aplikace.

5.1.4 Práce s aplikací

Aplikaci lze ovládat libovolným uživatelským rozhraním, které je dodáno ve formě pluginu. Základním rozšířením distribuovaným spolu s aplikací je konzolové rozhraní *SimpleConsole*.

Pro ovládání aplikace může být spuštěno zároveň několik uživatelských rozhraní.

Spuštění aplikace

Aplikace se spouští skrze soubor *TransportSystemKernel.exe*. Tuto aplikaci je možné spustit s parametrem, který vybere, jaké uživatelské rozhraní se má spustit po startu. Tímto parametrem je číslo pluginu, které poskytuje uživatelské rozhraní. Pokud je aplikace spuštěna bez parametru, vybere se první načtený plugin poskytující uživatelské rozhraní (tedy pokud je nainstalován základní balíček aplikace, spustí se rozšíření *SimpleConsole*). Nakonec se spustí všechna UI vyjmenovaná v konfiguračním souboru.

Ukončení aplikace

Libovolné uživatelské rozhraní by mělo poskytovat funkci bezpečného ukončení aplikace, která nastaví příznak hlavní aplikaci. Hlavní aplikace se následně ukončí v první chvíli, kdy je stav aplikace a dat bezpečný pro ukončení.

Ovládání aplikace

Aplikace je ovládána skrze zvolené uživatelské rozhraní. Možnosti ovládání aplikace jsou závislé na konkrétním rozšíření (viz dokumentaci používaného rozšíření).

Aplikace samotná poskytuje uživatelskému rozhraní tyto možnosti:

1. Zjištění všech kontrolovaných požadavků
2. Získání kopie kontrolovaného požadavku
3. Přidání a odebrání požadavku
4. Zjištění všech dostupných zdrojů dat o veřejné dopravě
5. Zjištění všech dostupných způsobů upozornění uživatele o změně ve výsledku požadavku
6. Zjištění všech dostupných způsobů získání výsledků požadavku z aplikace

7. Vytvoření výstupu výsledku za použití jednoho způsobu z bodu 6
8. Zjištění všech dostupných uživatelských rozhraní
9. Spuštění libovolného uživatelského rozhraní z bodu 8
10. Nastavení intervalu kontroly (Po jeho nastavení se ihned spustí kontrola požadavků)

Kontrolovaný požadavek
Požadavek jako takový je tvořen jedním nebo více dílčími požadavky. Dílčí požadavky mohou být tří typů, kdy v jednom požadavku musí být všechny jeho dílčí požadavky stejného typu. Každý dílčí požadavek může být omezený filtry.

Filtry mohou být přidány spolu s rozšířením. Následně mohou být použity rozšířeními, které jsou připravené na nové filtry. Například nepůjde vytvořit požadavek s filtrem, který nejde nastavit v uživatelském rozhraní, nebo pokud filtr předpokládá, že zdroj dat bude poskytovat rozšířená data, nepůjde použít se zdrojem dat, který tato data neposkytuje. Systém samotný poskytuje základní filtry, které by měly být kompatibilní se všemi rozšířeními.

Typy požadavků

1. Požadavek na jízdní řády linky
    - Kontroluje jízdní řád linky. Je možné jej omezit na jízdy projíždějící určitou stanici v určitém časovém intervalu.
2. Požadavek na jízdní řády stanice
    - Kontroluje jízdy projíždějící skrze stanici. Lze jej omezit na určitý časový interval a stanice, do nichž jízdy směřují.
3. Požadavek na spojení
    - Kontroluje spojení mezi počáteční a koncovou stanicí. Musí být specifikován časový interval odjezdu (resp. příjezdu)

TransportationSystemKernel.config

Konfigurační soubor je ve formátu XML a obsahuje položky s nastavením:

- LoggerLevel
    - Specifikuje úroveň logování (Debug, Info, Warning, Fatal).
- ConnectionString
    - Connection string specifikující připojení k DB. „{0}“ je nahrazeno cestou do složky <data>.
- StartingUI

  - Seznam id pluginů uživatelských rozhraní, která se spustí po startu aplikace, oddělený středníkem.

- CheckPeriode
  - Perioda kontroly požadavků v minutách. Do aplikace se načítá pouze při spuštění. Pokud je změněna perioda skrze příkaz v konzoli, je nastavení uloženo v *%AppData%*.

- AppDataFolder
  - Obsahuje cestu ke složce <data> (viz C.1.2).

5.1.5 Základní rozšíření aplikace

Základní rozšíření aplikace implementuje konzolové uživatelské rozhraní, způsob získávání dat z CIS JŘ, notifikaci emailem a vytvoření Excel tabulky s výsledkem požadavku.

Uživatelské rozhraní (SimpleConsole)

Jedná se o rozhraní konzolového typu, které je v jistých směrech specifické.

Konzolové okno je rozděleno na dvě části. První část je vyhrazena pro uživatelem zadávané znaky, druhá část vypisuje odpovědi aplikace (nápovědu, výsledek, informační hlášení).

Konzole pracuje celkem ve třech režimech. První a zároveň počáteční je režim, kdy okno konzole čeká na připojení k hlavní aplikaci a načtení inicializačních dat. Po dokončení inicializace připojené konzole se přechází do režimu zadávání příkazu. Z režimu zadávání příkazů lze přejít do režimu procházení historie příkazů.

Samotné okno konzole lze ukončit klávesovou kombinací **Alt+F4**.

Režim zadávání příkazu

Zadání příkazu funguje jako jednoduchý textový editor. Je možné psát příkaz s více řádky zároveň a pohybovat se mezi znaky (resp. řádky) pomocí šipek a kláves **Home/End**.

Zadání příkazu je ukončeno v momentě, kdy je příkaz kompletní, ukazatel pozice je na konci posledního řádku příkazu a uživatel stiskne **Enter**. Příkaz lze také ukončit, pokud je kompletní, stiskem kláves **Ctrl+Enter**.
Pro urychlení zadávání příkazu je vyhrazena klávesa **Tab**. Po jejím stisknutí je příkaz doplněn o všechny jednoznačně možné následující znaky.

Při napsání libovolného znaku je kontrolováno, zda znak může nabýt libovolného z očekávaných významů. Lze tudíž psát jen očekávané příkazy. Pokud je napsán neočekávaný znak, konzole vypíše seznam očekávaných znaků a nápovědu (viz obr. 5.1).

Nápověda v některých případech, kdy je jasně definován následující text příkazu, umožňuje výběr z nabídky v seznamu nápovědy. Tento seznam lze poznat podle textu, který má na počátku řádky tvar „x ->“ (kdy x je volný znak). Pro výběr ze seznamu stačí stisknout tento znak a text příkazu se doplní vybranou možností.

```
C:\Users\Bubyx\Documents\Programing\VisualStudio2013\Projects\RocnikovyProjectFro...
create request{connection from "Ma_
------------------------------ SPECIÁLNÍ FUNKCE ------------------------------
<TAB> doplní všechny následující jednoznačné znaky. <CTRL+ENTER> zadá příkaz do
 aplikace, pokud je v dokončené formě
---------------------------------- NÁPOVĚDA ----------------------------------
Očekáván jeden ze znaků: 's','l','r','n','d','š','c'
"<název počáteční stanice spojení>"
3 -> Masná - Praha,,Masná
4 -> Maniny - Praha,,Maniny
5 -> Mandava - Sulice,Želivec,Mandava
6 -> Madlina - Praha,,Madlina
7 -> Masečín, - Štěchovice,Masečín,
8 -> Marvánek - Říčany,,Marvánek
9 -> Markvart - Pohoří,,Markvart
a -> Marjánka - Praha,,Marjánka
b -> Malovanka - Praha,,Malovanka
e -> Malešická - Praha,,Malešická
f -> Martinov, - Záryby,Martinov,
g -> Masojedy,, - Masojedy,,
h -> Maskovice, - Netvořice,Maskovice,
i -> Malotice,, - Malotice,,
j -> Malvazinky - Praha,,Malvazinky
k -> Maskuv mlyn - Praha,,Maškův mlýn
m -> Maslovice,, - Máslovice,,
0 -> pro zobrazení další stránky
Nápověda je příliš velká (obsahuje 51 možností). Stránka 1/3
------------------------------ ZPRÁVY Z APLIKACE ------------------------------
```

Obrázek 5.1: Zobrazení nápovědy v konzolovém rozhraní

Pokud je příliš mnoho možností v seznamu, je seznam stránkován a pro posun je vybrán volný znak. Tento znak je uveden jako poslední položka v seznamu nápovědy.

Jako volný znak považujeme znak z množiny $('0' - '9') \cup ('a' - 'z')$, který nemůže být na dané pozici příkazu jeho pokračováním.

Režim procházení historie příkazů

Od spuštění konzolového rozhraní až po jeho vypnutí se každý smazaný nebo spuštěný příkaz zaznamenává do historie. V tomto režimu si příkazy uživatel může prohlédnout, případně s nimi inicializovat režim zadávání příkazů.

V režimu se přechází mezi příkazy pomocí šipek nahoru a dolů.

Pokud je příkaz vybrán stisknutím klávesy **Enter**, je text příkazu kopírován do režimu zadávání příkazu.

Režim lze opustit stisknutím klávesy **Esc**.

Uživatelské příkazy

- get requests
  - Vypíše seznam všech požadavků, které jsou kontrolovány, ve tvaru „identifikátor požadavku - text, který ho popisuje".
- get downloaders
  - Vypíše seznam všech pluginů pro stahování dat ve tvaru „identifikátor – popis".

- get notificators
  - Vypíše seznam veškerých pluginů, které zajišťují notifikace, ve tvaru „identifikátor – popis“.
- get formatters
  - Vypíše seznam všech pluginů poskytující formátovaný výpis požadavku ve tvaru „identifikátor – popis“.
- get ui
  - Vypíše seznam všech dostupných uživatelských rozhraní ve tvaru „identifikátor – popis“.
- get history **x**
  - Získá posledních **x** zpráv z aplikace.
- print request **x** with formator **y** to addres "**z**"
  - Vyvolá výpis požadavku **x** skrz formátovač **y** na adresu **z**. Kdy **x** je identifikátor požadavku, **y** je identifikátor formátovacího pluginu a **z** je adresa, na kterou se má výstup uložit. Formát adresy by měl být uveden v popisu formátovače (může se jednat o adresu v souborovém systému, e-mail, atd.).
    - **x** a **y** musí být celá čísla
    - **z** je libovolný řetězec
- set interval **x**
  - Nastaví interval kontroly aktuálnosti dat na **x** hodin, přičemž **x** je přirozené číslo.
  - Po nastavení intervalu se spustí kontrola požadavků.
- delete request **x**
  - Odebere ze seznamu kontrolovaných požadavků požadavek **x**, kdy **x** je celé číslo a identifikátor požadavku získaný příkazem get requests.
- start ui **x**
  - Spustí další uživatelské rozhraní, jehož identifikátor je **x**.
  - Pokud je přidán parametr **-s** (*start ui -s **x***), bude nastaveno automatické spuštění zvoleného uživatelského rozhraní po startu aplikace.
- close app
  - Informuje aplikaci o tom, že by se měla zavřít, až to bude možné.
  - Po ukončení hlavní aplikace bude konzolové okno stále otevřené a bude hlásit problémy s připojením k aplikaci. Pro zavření okna lze použít zkratku **Alt+F4**.

- from request **x**
  - Vytvoří v konzoli tělo pro nový požadavek na základě již existujícího požadavku **x**.

- exec on **x** with "**y**"
  - Předá příkaz **y** zvolenému rozhraní **x**. Slouží pro dodatečné nastavení, nebo vyvolání akce v rozšíření.

- create request ...
  - Tento příkaz slouží pro přidání kontrolovaného požadavku.
  - Dotaz by měl být ve formátu, který odpovídá grafu (viz obr. A.3) nebo gramatice (viz B). Modré jsou objekty dotazu a oranžové jsou poznámky k proměnným. Kruhové objekty značí text, který se má napsat, čtvercové objekty značí proměnnou a deltoidy slouží pouze jako propojení objektů. Celkový text dotazu musí být v pořadí, které určují šipky v grafu. Mezi každými dvěma objekty by měla být mezera.
  - V příkazu se nachází místa pro vyplnění jména stanice (resp. linky). Tato pole jsou dynamicky omezena na jména poskytnuta zdrojem dat vybraným na počátku příkazu („with downloader **x**“). Pokud daný zdroj neposkytuje jména stanic (resp. linek), je do těchto míst možno vyplnit libovolný text. Nesprávný název způsobí chybu až v momentě kontroly požadavku.

Zdroj dat

Rozšíření stahuje data z CIS JŘ (ftp://ftp.cisjr.cz/) a filtruje v nich data pro Prahu a okolí.

Emailová notifikace

Toto rozšíření odešle email na adresu uživatele, která byla nastavena během instalace.

Nastavení cílové emailové adresy a další nastavení lze editovat v konfiguračním souboru *ImplicitModules.dll.config*.

5.1.6 Rozšíření formátovače výstupu

Formátovač výstupu vytváří pro dotazy výstup do tabulky Excel. Vytvoření výstupu lze spustit z konzole příkazem **print**. Jako parametr je očekávána cesta ke složce, v níž se má soubor vytvořit, nebo cesta k souboru, který má být vytvořen. Cesta může být absolutní nebo relativní k cestě specifikované v konfiguračním souboru.

Např.: print request 1 with formatter 2 to address "*C:\Users\Test\Documents\TestingData\TSPrints\pozadavek1.xlsx*"

ImplicitModules.dll.config

Konfigurační soubor je ve formátu XML a obsahuje položky s nastavením:

- smtpServer
  - SMTP server pro odesílání notifikačních emailů
- smtpPort
  - Port k **smtpServer**
- fromAdress
  - Adresa, z níž se odesílají emaily
- fromMailUser
  - Uživatelské jméno pro přihlášení na email, z něhož se odesílají notifikace
- fromMailPassword
  - Heslo k **fromMailUser**
- toAdress
  - Adresa, na níž se odesílají emaily
- simpleconsolepipename
  - Jméno roury skrze níž se komunikuje s *SimpleConsole.exe*
- loggerLevel
  - Specifikuje úroveň logovaní (Debug, Info, Warning, Fatal)
- formatterRootPath
  - Výchozí cesta souborovým systémem pro formátovaný výstup
- JDFSource
  - Seznam (*ArrayOfString*) url adres souborů obsahujících data v JDF formátu
- PDFSourceFolder
  - Url adresa složky obsahující PDF jízdních řádu tramvají a metra PID
- LineExistanceCheckerUrl
  - Url adresa použitá pro ověření funkčnosti linky, parametrizovaná číslem linky („{0}“)
- LineNotexistPattern
  - Text, který je obsažen na stránce specifikované LineExistanceCheckerUrl, pokud linka neexistuje, nebo není nadále v provozu
  - Parametrizováno číslem linky („{0}“)

SimpleConsole.exe.config

Konfigurační soubor je ve formátu XML a obsahuje položky s nastavením:

- AppDataFolder
  - Obsahuje cestu ke složce <data> (viz C.1.2).

5.2 Mobilní aplikace

Mobilní aplikace slouží pro správu a zobrazení výsledků požadavků na veřejnou dopravu. V mobilu můžete obdobně jako v desktopové aplikaci vytvářet požadavky a zobrazovat výsledky požadavků. Mobil je automaticky synchronizován s počítačem.

Po spuštění aplikace zkontroluje, zda je uživatel přihlášen (pokud není, je vyzván k přihlášení) a zda má aplikace dostupná základní data (pokud ne, zobrazí se vyčkávací smyčka), následně je zobrazena hlavní aktivita aplikace.

5.2.1 Rozdělení aplikace

Aplikace se skládá ze tří částí: *Nastavení*, *Zobrazení výsledku* a *Založení požadavku*. V *Nastavení* najdeme pouze založení (resp. přihlášení) k účtu.

Výsledky se zobrazují ve dvou módech. Zobrazení všech výsledků nebo zobrazení výsledků pro jeden požadavek. V obou případech se zobrazí výsledky rozdělené podle typu požadavku (jízdy linky, průjezdy stanicí a spojení).

Při založení požadavku je nejprve nutné zvolit typ. Na typu následně závisí vytváření jednotlivých částí požadavků.

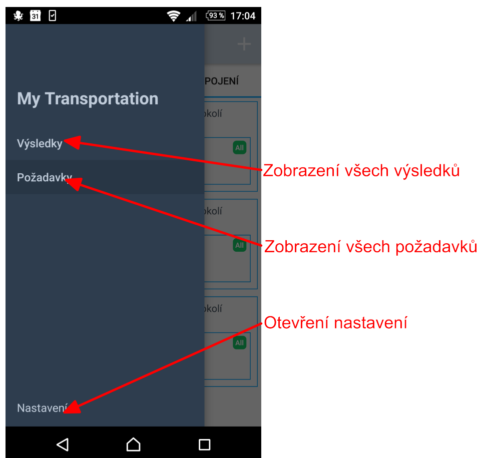
5.2.2 Synchronizace

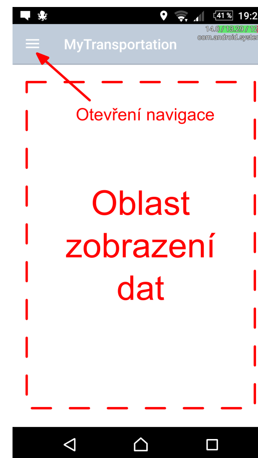
Po přihlášení uživatele aplikace pošle požadavek desktopové aplikaci na obnovení dat. Pokud aplikace nemá k dispozici základní data (tj. seznam stanic, seznam linek, seznam rozšíření desktopové aplikace) zobrazí uživateli vyčkávací smyčku dokud data neobdrží. Další synchronizace probíhá automaticky na pozadí celého systému. Mobilní aplikace čeká na změny poslané desktovovou aplikací a odesílá změny v požadavcích.

5.2.3 Hlavní aktivita

Hlavní aktivitu tvoří navigační lišta (viz obr. 5.2) a prostor pro zobrazení dat (viz obr. 5.3). Podle výběru z navigace je v oblasti dat zobrazen seznam všech výsledků (viz obr. 5.4) nebo seznam všech požadavků (viz obr. 5.5). Prostřednictvím navigace je možné přejít do nastavení, v kterém může uživatel změnit přihlašovaný účet. Pro vytvoření nového požadavku slouží znak plus v pravém horním rohu.

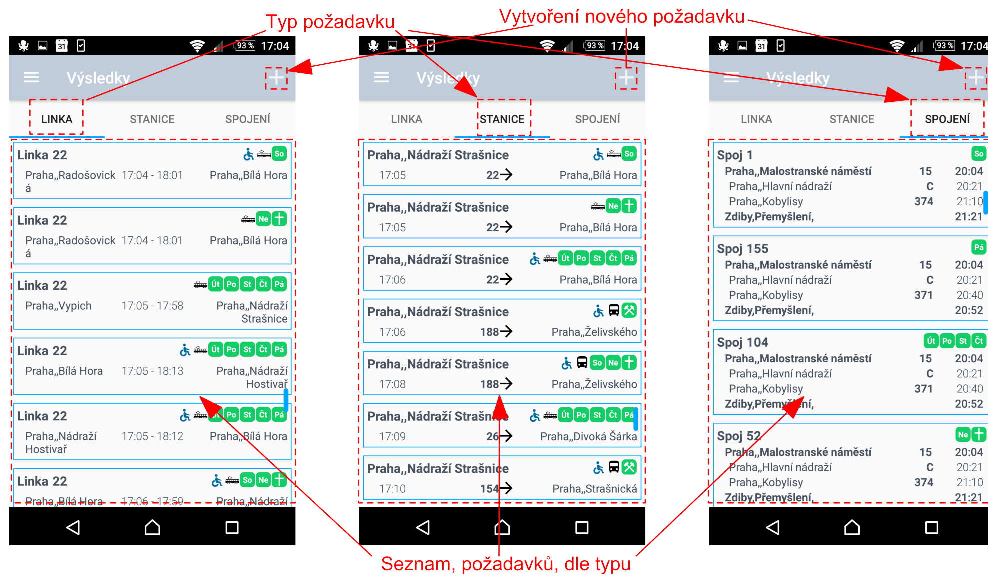
Při výběru seznamu výsledků (resp. požadavků) jsou zobrazeny vedle sebe tři seznamy podle typu požadavku (viz obr. 5.4 a obr. 5.5). Pro přechod mezi seznamy lze kliknout na hlavičku seznamu nebo použít gesto přetažení ze strany na stranu.



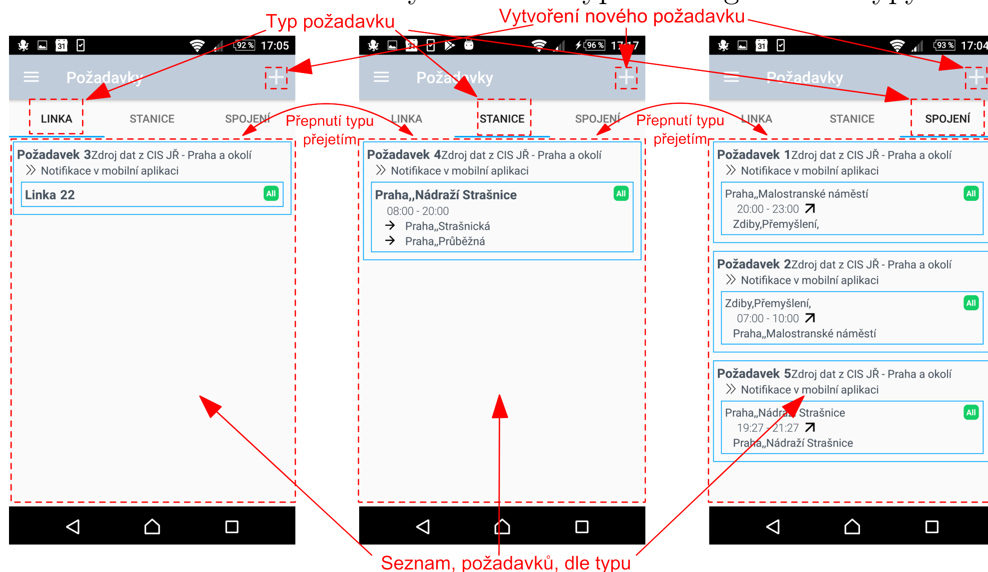


Obrázek 5.2: Navigační lišta s výběrem z výsledků, požadavků a nastavení

Obrázek 5.3: Oblast zobrazení dat



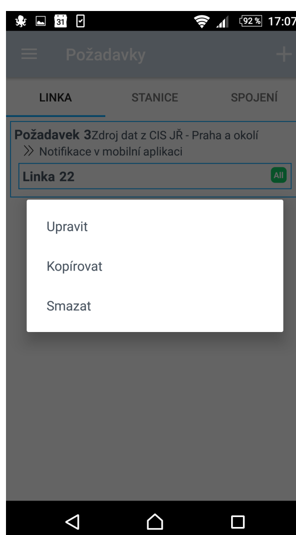
Obrázek 5.4: Seznam výsledků dle typu s navigací mezi typy



Obrázek 5.5: Seznam požadavků dle typu s navigací mezi typy

Akce s položkami seznamů

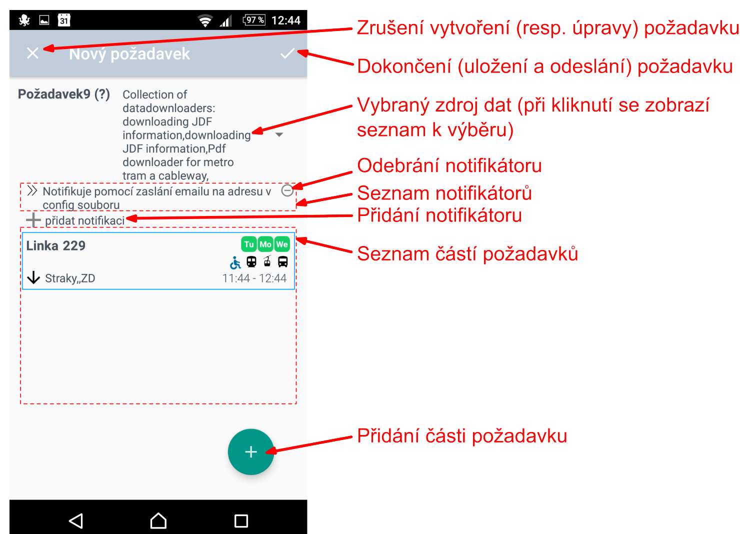
- Při kliknutí na položku požadavku se zobrazí výsledky pro daný požadavek.
- Při dlouhém kliknutí na položku požadavku se zobrazí kontextová nabídka (viz obr. 5.6) s možnostmi požadavek editovat, smazat nebo kopírovat.
- Při kliknutí na položku výsledku se zobrazí detail jízdy linky.
- Při kliknutí na položku výsledku spojení se zobrazí částečný detail posloupnosti jízd. Kliknutím na jízdu se zobrazí detail jízdy.



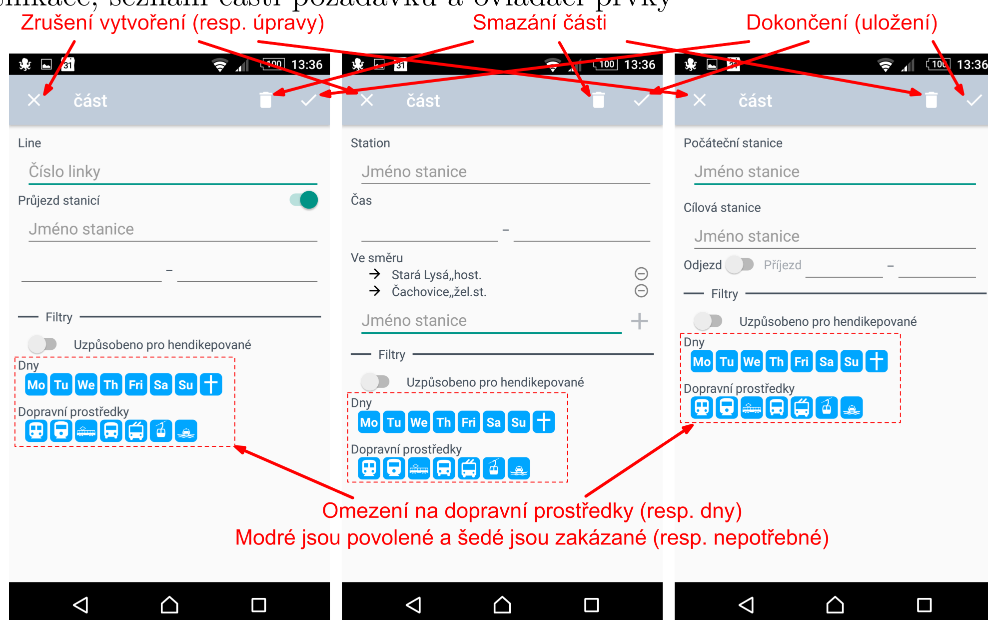
Obrázek 5.6: Kontextová nabídka

5.2.4 Vytvoření požadavku

Vytváření (resp. úprava) požadavku se skládá ze dvou částí, vytváření požadavků (viz obr. 5.7) a vytváření částí požadavku (viz obr. 5.8). Části požadavku se skládají z části závislé na zvoleném typu požadavku a obecných filtrů. V nastavení požadavku uživatel zvolí zdroj dat, a způsob notifikace. Následně založí požadovaný počet částí požadavku.



Obrázek 5.7: Úprava požadavku - obsahuje výběr zdroje dat, seznam způsobů notifikace, seznam částí požadavku a ovládací prvky



Obrázek 5.8: Úprava části požadavku dle typu - obsahuje nastavení filtrů dnů, typů dopravních prostředků a na typu závislá pole

Závěr

Hlavním cílem práce bylo vytvořit systém pro kontrolu změn v jízdních řádech, který informuje uživatele o změnách ve veřejné dopravě ovlivňující výsledky na uživatelem zadané požadavky. Dílčími cíli práce bylo systém napojit na data PID, upozornit uživatele na změny, vizualizovat výsledky a umožnit uživateli vytvářet požadavky. Na základě těchto cílů vznikla desktopová aplikace na platformě .NET, která se pomocí přídavných modulů rozšiřuje o zdroje dat jízdních řádů, způsoby notifikace uživatele, vytváření výstupů výsledků a uživatelská rozhraní pro práci s aplikací. Pomocí modulů jsme implementovali zpracování dat PID z CIS JŘ, odesílání emailů informujících o změně, formátování výsledků do tabulky Excel, uživatelské rozhraní konzolového typu a komunikaci s mobilní aplikací.

Mobilní aplikace umožňuje uživateli spravovat požadavky a zároveň uchovává (resp. zobrazuje) aktuální výsledky i bez přístupu na síť. Mobilní aplikace je s hlavní aplikací synchronizována tak, aby byl minimalizován objem přenesených dat, optimalizována spotřeba energie a celkové vytížení mobilního zařízení. Zároveň je synchronizace navržena s ohledem na síťová omezení běžného desktopového počítače mezi něž patří především proměnlivá neveřejná IP adresa, restrikce síťových prvků na trase spojení a omezená dostupnost samotného zařízení.

V rámci funkce zpracování požadavků na vyhledání spojení mezi stanicemi je součástí hlavní aplikace implementace základního vyhledávání spojení v jízdních řádech.

Aplikace byla celá navržena tak, aby bylo možné ji v budoucnu dále rozšiřovat v rámci jádra i modulů. Aktuálně vytvořená aplikace vyhovuje stanoveným požadavkům, ovšem chtěli bychom ji v budoucnu dále rozšiřovat o nové funkce a nástroje. Jedním z možných rozšíření je úprava dynamického vytváření filtrů v uživatelském rozhraní, pomocí níž bychom chtěli v budoucnu dosáhnout automatické úpravy uživatelského rozhraní na základě nových filtrů přidaných cizím zásuvným modulem. Dalším rozšířením aplikace může být implementace optimalizací algoritmu pro vyhledávání spojení nebo vytvoření nového modulu zdroje dat pro data ve formátu GTFS.

Reference

[1] Knihovna iTextSharp. https://sourceforge.net/projects/itextsharp/.

[2] Affero general public license. https://www.gnu.org/licenses/agpl-3.0.en.html.

[3] Knihovna PDFsharp. http://www.pdfsharp.net/.

[4] Frank Schulz. *Timetable information and shortest paths.* PhD thesis, Universität Fridericiana zu Karlsruhe (TH), 2005.

[5] Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable information: Models and algorithms. 1988.

[6] Robert Geisberger. *Advanced route planning in transportation networks.* PhD thesis, Universität Karlsruhe, 2011.

[7] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics (JEA)*, 2008.

[8] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos D Zaroliagis. Experimental comparison of shortest path approaches for timetable information. In *ALENEX/ANALC*, pages 88–99. Citeseer, 2004.

[9] Martin Mareš and Tomáš Valla. *Průvodce labyrintem algoritmů.* CZ. NIC, zspo, 2017.

[10] Jakub Melka. Fibonacciho haldy-jejich varianty a alternativní datové struktury. 2012.

[11] PID opendata. https://pid.cz/o-systemu/opendata/.

[12] Epplus. https://github.com/JanKallman/EPPlus.

[13] Ministerstvo dopravy – Odbor veřejné dopravy a kombinované dopravy. Popis formátu a struktury dat pro elektronické zpracování jízdních řádů platných od 1. ledna 2003 (jednotný datový formát – verze 1.9). http://www.chaps.cz/files/cis/jdf-1.9.pdf.

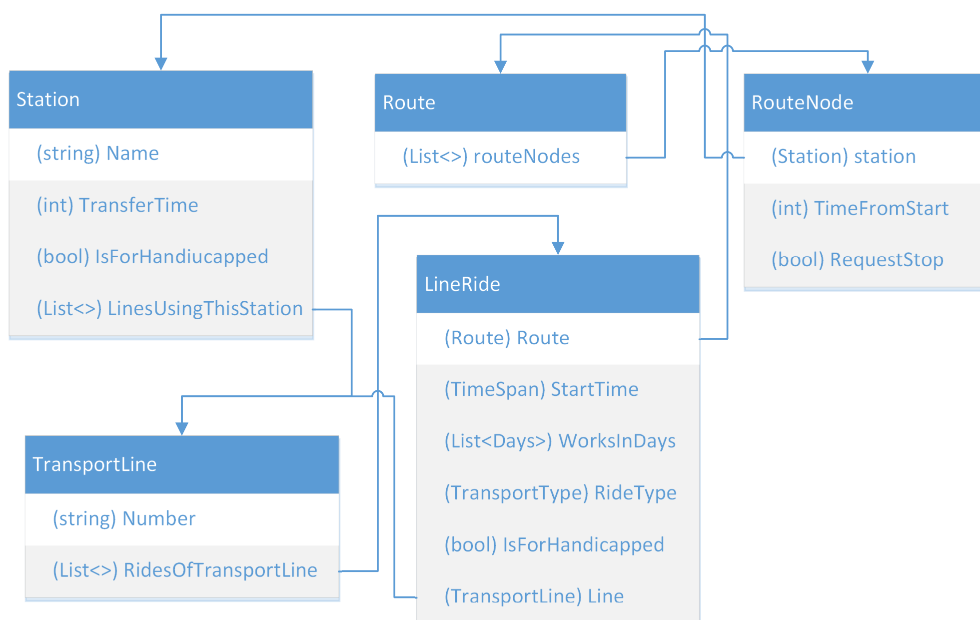
[14] Ministerstvo dopravy – Odbor veřejné dopravy. Popis formátu a struktury dat pro elektronické zpracování jízdních řádů (jednotný datový formát – verze 1.10). https://chaps.cz/files/cis/jdf-1.10.pdf.

[15] Ministerstvo dopravy – Odbor veřejné dopravy. Metodický pokyn č.5 k organizaci celostátního informačního systému o jízdních řádech. https://www.mdcr.cz/getattachment/Dokumenty/Silnicni-doprava/Mezinarodni-autobusova-doprava-(MAD)/Povolovaci-rizeni-mezinarodnich-linek-v-rezimu-(2)/Povolovaci-rizeni-mezinarodnich-linek-v-rezimu-se/Jednotny-format-dat.pdf.aspx.

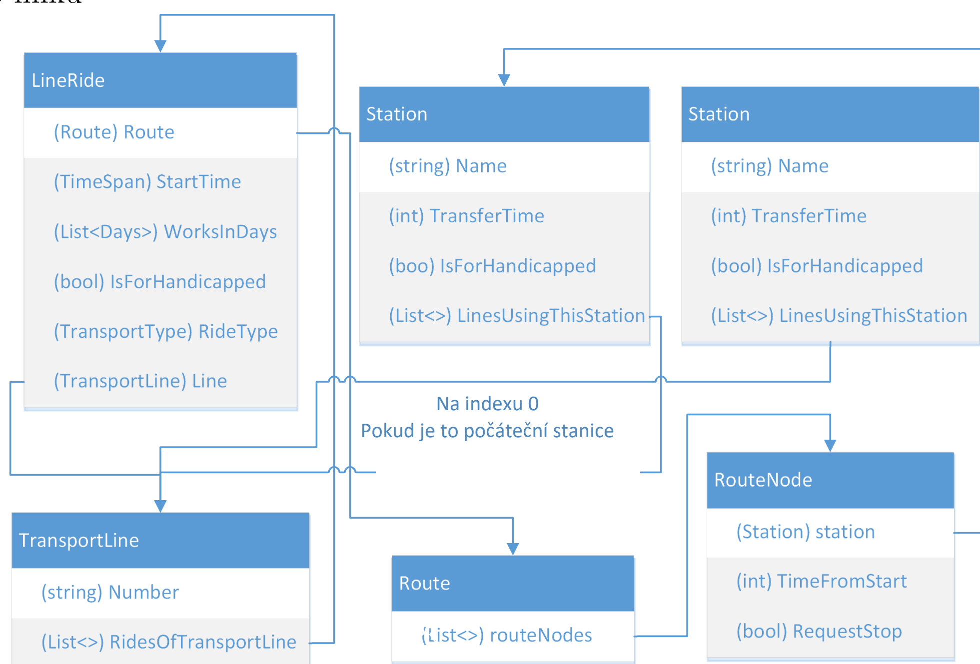
Část I

Přílohy

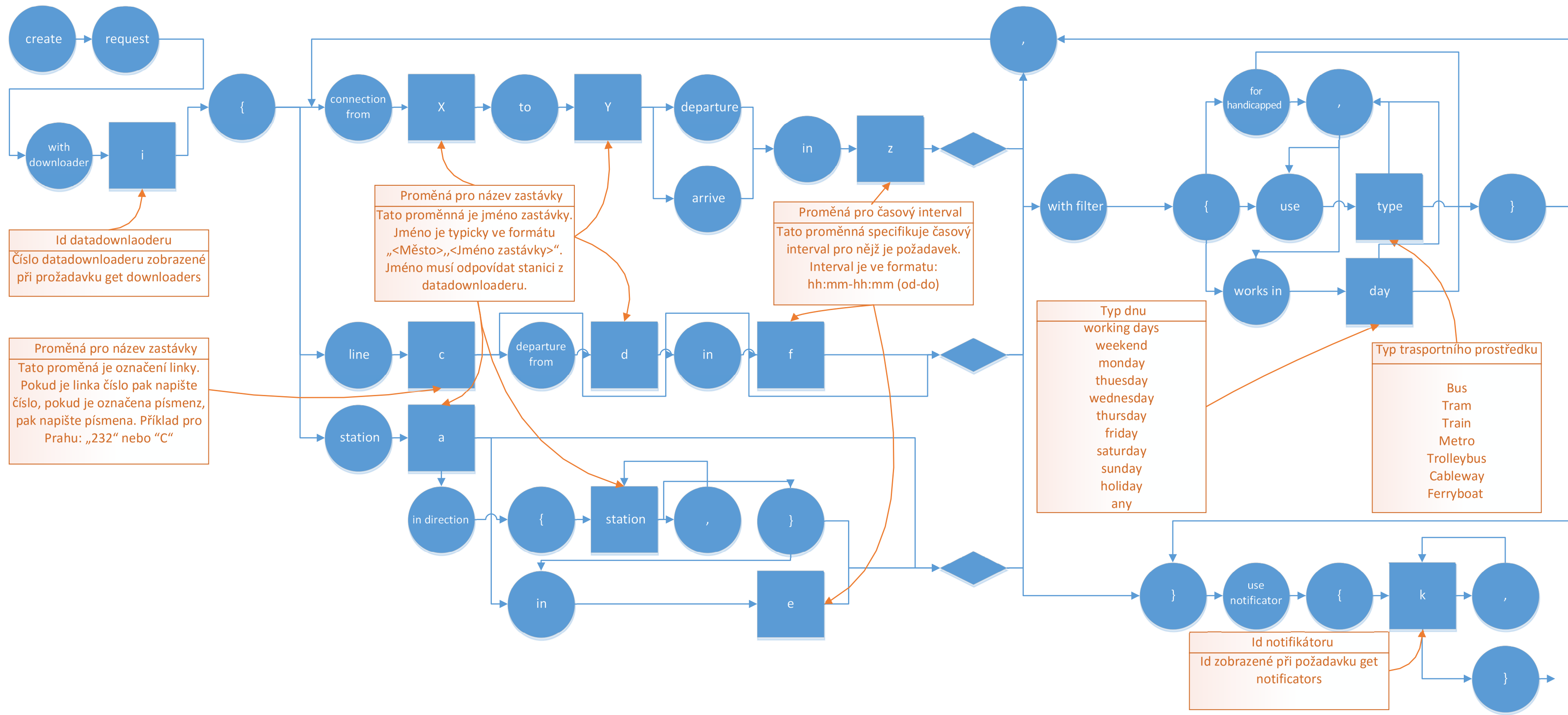
A. Reference výsledků



Obrázek A.1: Model vyžadovaného referenčního propojení dat vrácených ze zdroje pro linku



Obrázek A.2: Model vyžadovaného referenčního propojení dat vrácených ze zdroje pro stanici



Obrázek A.3: Vizualizace části automatu pro vytvoření požadavku

B. Gramatika automatu

Gramatika automatu je specifikována následujícími pravidly, kdy pravidla pro neterminály **A1**, **D1**, **F1**, **S1**, **U**, **R**, **N1** a **L2** jsou dynamicky generovaná s proměnnou gramatikou. Počáteční neterminál je **S**.

A1 -> "{{ jakákoliv písmena }}"
D1 -> {{ data downaloader id }}
F1 -> {{ id formátovače }}
P3 -> {{ id rozšíření }}
S1 -> "{{ station name }}"
U -> {{ ui id }}
R -> {{ request id }}
N1 -> {{ id notifikátoru }}
L2 -> "{{ číslo linky }}"

W0 →' ' | \t | \r | \n |W|λ
W -> ' 'W0 | \tW0 | \rW0 | \nW0
T0 -> metro | train | tram | bus | trolleybus | cableway | ferry | boat
D0 -> working day | weekend | monday | tuesday | wednesday | thursday | friday | saturday | sunday | holiday | all

–time interval–
H2 -> T1-T1
T1 -> G2N3:M2N3
G2 -> 0|1|2|λ
M2 -> 0|1|2|3|4|5
N3 -> 0|1|2|3|4|5|6|7|8|9

S -> get requests | get downloaders | get notificators | get formatters | get ui | close app | from requestWR | delete requestWR

S -> start uiWU1
U1 -> -sWU|U

S -> set intervalWI1
I1 -> 0|1|2|3|4|5|6|7|8|9|I1I1

S -> print requestWRWwith formatterWF1Wto addressWA1

S -> exec onWP3WP4
P4 -> withWA1

S -> create requestWZ
Z -> with downloaderWD1L
L -> {P | W {P
P -> connection fromQ | WP | lineT | stationV

–spojení–
Q -> WS1WtoWS1WW23
W23 -> departureK1 | arriveK1
K1 -> WinWH2H3
H3 -> ,X | WW27 | }Y
W27 -> ,X | with filterO1 | }Y

–linka–
T -> WL2L3
L3 -> ,X | WW29 | }Y
W29 -> ,X | with filterO1 | }Y | departure fromP1
P1 -> WS1WinWH2H4
H4 -> ,X | WW33 | }Y
W33 -> ,X | with filterO1 | }Y

–stanice–
V -> WS1S5
S5 -> ,X | WW35 | }Y
W35 -> ,X | with filterO1 | }Y | inV1 | in directionX1
X1 -> {Y1 | W {Y1
Y1 -> S1S6 | WY1
S6 -> ,A2 | WS6 | }B2
A2 -> WS1S6
B2 -> ,X | WW40 | }Y | inV1
W40 -> ,X | with filterO1 | }Y | inV1
V1 -> WH2H5
H5 -> ,X | WW42 | }Y
W42 -> ,X | with filterO1 | }Y

–multi request–
X -> WW12
W12 -> connection fromQ | lineT | stationV

–filtr–
O1 -> {C2 | W {C2
C2 -> useE2 | WC2 | works inJ2 | for handicappedK2
K2 -> }O2 | WK2 | ,P2
O2 -> ,X | WO2 | }Y
P2 -> for handicappedK2 | WP2 | useE2 | works inJ2
E2 -> WT0Q2
Q2 -> }O2 | WQ2 | ,P2
J2 -> WD0T2
T2 -> ,P2 | WT2 | }O2

–notifikátor–
Y -> use notificatorC1 | WY
C1 -> {E1 | WC1
E1 -> N1N2 | WE1
N2 -> ,WN1N2 | WN2 | }

C. Instalace

C.1 Desktopové aplikace

Aplikace se instaluje pomocí instalačního balíčku, který ji nainstaluje se základními rozšířeními. V případě nedostupnosti Microsoft .NET framework 4.5 a Microsoft SQL Express LocalDB 2014 jsou tyto aplikace zahrnuty v instalaci.

Po spuštění instalačního balíčku se zobrazí standardní průvodce instalací.

Instalační balíček umožní uživateli vybrat, zda chce nainstalovat základní rozšíření, vytvořit odkazy na aplikaci v nabídce Start či automaticky spustit aplikaci po spuštění počítače. Pokud jsou instalována základní rozšíření, je uživatel požádán o emailovou adresu. Emailovou adresu je možné dodatečně nastavit v konfiguračním souboru (viz 5.1.6).

C.1.1 Předpoklady pro běh aplikace

Tato aplikace pro svůj běh vyžaduje Microsoft .NET framework 4.5 nebo vyšší a Microsoft SQL Express LocalDB 2014.

V případě použití instalačního souboru budou požadované programy nainstalovány spolu s aplikací.

C.1.2 Bez instalačního balíčku

Systém je možné distribuovat bez nutnosti instalačního balíčku. V tomto případě je nutné zajistit, aby počítač, na němž aplikace poběží, měl nainstalované výše zmíněné programy a soubory systému byly na stroj zkopírovány s dodržením následující hierarchie souborů. Následně je nutné upravit konfiguraci souboru *ImplicitModules.dll.config* a *TransportSystemKernel.exe.config* (viz níže).

Hierarchie souborů aplikace

Soubory jsou rozdělené do dvou složek. První <složka aplikace> obsahuje spustitelné soubory, knihovny a konfigurační soubory.

Druhá <data> obsahuje databázi a další datové soubory. Aplikace musí mít po spuštění práva tuto složku upravovat. Odkaz na druhou složku musí být správně uveden v konfiguračních souborech *ImplicitModules.dll.config* a *TransportSystemKernel.exe.config* (viz níže).

```
<složka aplikace>\ResourcesForPlugins.dll
<složka aplikace>\TransportSystemKernel.exe
<složka aplikace>\TransportSystemKernel.exe.config
<složka aplikace>\Plugins\EPPlus.dll
<složka aplikace>\Plugins\ImplicitModules.dll
<složka aplikace>\Plugins\ImplicitModules.dll.config
<složka aplikace>\Plugins\ResourcesForPlugins.dll
<složka aplikace>\Plugins\SimpleConsole.exe
<složka aplikace>\Plugins\SimpleConsole.exe.config
<složka aplikace>\Plugins\itextsharp.dll
```

```
<data>\Data\DB.mdf
<data>\Data\DB_log.ldf
```

C.1.3 Přidání rozšíření

V případě přidání rozšíření je třeba přidat soubory rozšíření (resp. nainstalovat) do složky *Plugins* ve složce s aplikací a restartovat aplikaci.

C.2 Mobilní aplikace

Aby bylo možné použít rozšíření, je podmínkou existence funkční instalace aplikace Transportation Checker.

C.2.1 MyTransportation.apk

Do mobilního telefonu lze aplikaci nainstalovat pomocí přiloženého souboru MyTransportation.apk.

C.2.2 MobileInterface.dll

Toto rozšíření je součástí základního instalačního balíčku.

C.2.3 První spuštění

Po splnění předchozích instrukcí je možné spustit aplikaci.

Nejprve spusťte mobilní aplikaci. Aplikace Vás vyzve k vytvoření nebo přihlášení se k účtu (viz obr. C.1). Tento účet je nutné založit z důvodu přenosu dat z hlavní aplikace do mobilní a naopak. Je vyžadována validní emailová adresa a heslo. Email není dále nijak ověřován, tudíž je možné zadat i neexistující email. Po přihlášení se zobrazí informace pro připojení počítače (viz obr. C.2). Po dokončení přihlášení bude mobilní aplikace čekat na základní data z hlavní aplikace (seznam stanic, linek a rozšíření).

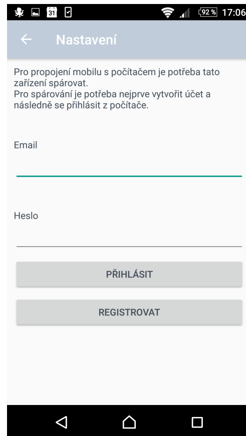
Nyní můžete spustit hlavní aplikaci na Vašem počítači. (Pokud Vám již aplikace běží, je potřeba ji restartovat, aby se do aplikace načetlo nové rozšíření.)

V hlavní aplikaci následně spusťte uživatelské rozhraní dle dokumentace aktuálně používaného uživatelského rozhraní. Při spuštění UI tohoto rozšíření se zobrazí dialogové okno pro přihlášení k uživatelskému účtu vytvořenému v mobilní aplikaci (viz C.3).

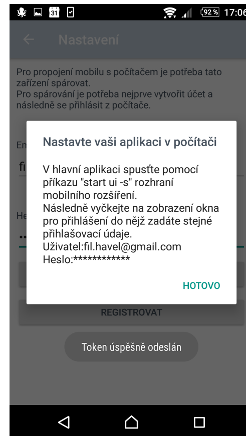
Spuštění uživatelského rozhraní

Pokud používáte aplikaci se základním rozšířením, zobrazte si seznam dostupných UI pomocí příkazu *get ui*. V seznamu najděte identifikátor rozhraní s označením *Interface for mobile application*.

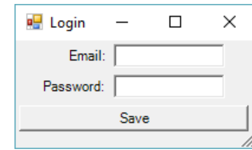
Nalezený identifikátor použijte v příkazu *start ui*, nebo *start ui -s* pokud chcete, aby se rozhraní spouštělo automaticky po startu hlavní aplikace. Uživatelské rozhraní je také možné nastavit v konfiguračním souboru (viz dokumentace hlavní aplikace - TransportationSystemKernel.exe.config).



Obrázek C.1: Přihlášení v mobilu



Obrázek C.2: Přihlášení v mobilu hotovo



Obrázek C.3: Přihlášení v počítači

V případě potřeby může být vyvolána změna přihlašovacích údajů z konzolového rozhraní příkazem *exec on* ***u*** *with ”relogin”* (**u** je identifikátor tohoto rozšíření).

D. Implementační detaily jádra aplikace a ImplicitModules

D.1 Základní data aplikace

Pro práci aplikace s jízdními řady se používají objekty *Station*, *TranportLine*, *LineRide*, *Route* a *RouteNode*. Tyto objekty a jejich propojení jsou v aplikaci reprezentovány dvěma možnostmi podle potřeby aplikace.

První možnost je propojení objektů referencí. Pro toto použití jsou určeny třídy v *ResourcesForPlugins.Results*. Druhou možností je propojení objektů, které je specifikováno identifikátory. Tato možnost je implementována třídami *ResourcesForPlugins.Data*. Při této možnosti nemusí být všechna data zároveň v paměti počítače, je však možné s nimi možné postupně pracovat.

Objekt *Station* reprezentuje stanici. Obsahuje linky (*TransportLine*), které stanici projíždějí, a další vlastnosti stanice.

Objekt *TrasportLine* reprezentuje linku. Obsahuje jednotlivé jízdy (*LineRide*) a další vlastnosti linky.

Objekt *LineRide* reprezentuje jízdu linky. Obsahuje informaci o čase, kdy vyjíždí z první stanice trasy, trasu (*Route*) a další informace.

Objekt *Route* obsahuje pouze seřazený seznam stanic na trase (*RouteNode*) za sebou.

Objekt *RouteNode* reprezentuje stanici na trase. Obsahuje informace o použití stanice (zda je na znamení atd.) a referenci na stanici.

D.2 Výroba vlastního rozšíření

Aplikace se rozšiřuje o pluginy formou .NET knihovny vložené do složky *Plugins* (viz C.1.3)), knihovna musí být označena koncovkou „.dll“. Za jeden plugin je považována každá veřejná třída implementující jedno z následujících rozhraní. Tato třída musí mít bezparametrický konstruktor. Pro spolupráci s hlavní aplikací by měly pluginy využívat pouze tříd a rozhraní z knihovny *ResourcesForPlugins*. V této knihovně jsou především třídy datových modelů a rozhraní pro komunikaci s hlavní aplikací. Dále obsahuje nástroje, které může rozšíření používat (více v kapitole D.7)).

D.3 Jádro aplikace

Jádro aplikace zajišťuje kontrolu aktuálnosti dat, obsluhu požadavků z uživatelského rozhraní, uložení aktuálních relevantních dat o jízdních řádech a ukládání instancí požadavků.

Data o jízdních řádech jsou uložena v lokální databázi běžící na SQL Serveru společně s daty spojujícími je s požadavkem.

Požadavky se ukládají pomocí serializačních nástrojů v .NETu do jednotlivých souborů, jejichž jména jsou uložena v databázi. Pro serializaci je využit BinaryFormatter.

Jádro aplikace je tvořeno hlavním ovládáním aplikace (*Program.cs*), *Checkerem*, který kontroluje aktuálnost požadavků, dispečerem notifikací a databázovým rozhraním (*Data**).

Program.cs obsahuje metodu Main, která startuje aplikaci. Při startu se načtou dostupné pluginy, provede se základní inicializace souborů a ověří se připojení do databáze. Následně se spustí uživatelské rozhraní a nakonec se přejde do smyčky checkeru. Dále tento soubor implementuje rozhraní (*IInputable*), které je poskytnuto spuštěnému UI.

Checker funguje na principu nekonečné smyčky, která se před opakováním uspí na uživatelem nastavenou dobu.

Ve smyčce se vždy získají požadavky z databáze a zkontrolují se požadavky v pořadí: požadavky na linku, požadavky na stanici a požadavky na spojení. Kontrola je závislá na typu požadavku. V principu se vždy získá výsledek uložený v databázi z minulé kontroly a spočítá se aktuální výsledek. Aktuální výsledek a minulý výsledek se porovnají a naleznou se části, které se liší. Pokud se data liší, upraví se výsledek v databázi a vytvoří se *NotificationHolder*, který se pokouší o notifikaci, dokud neuspěje.

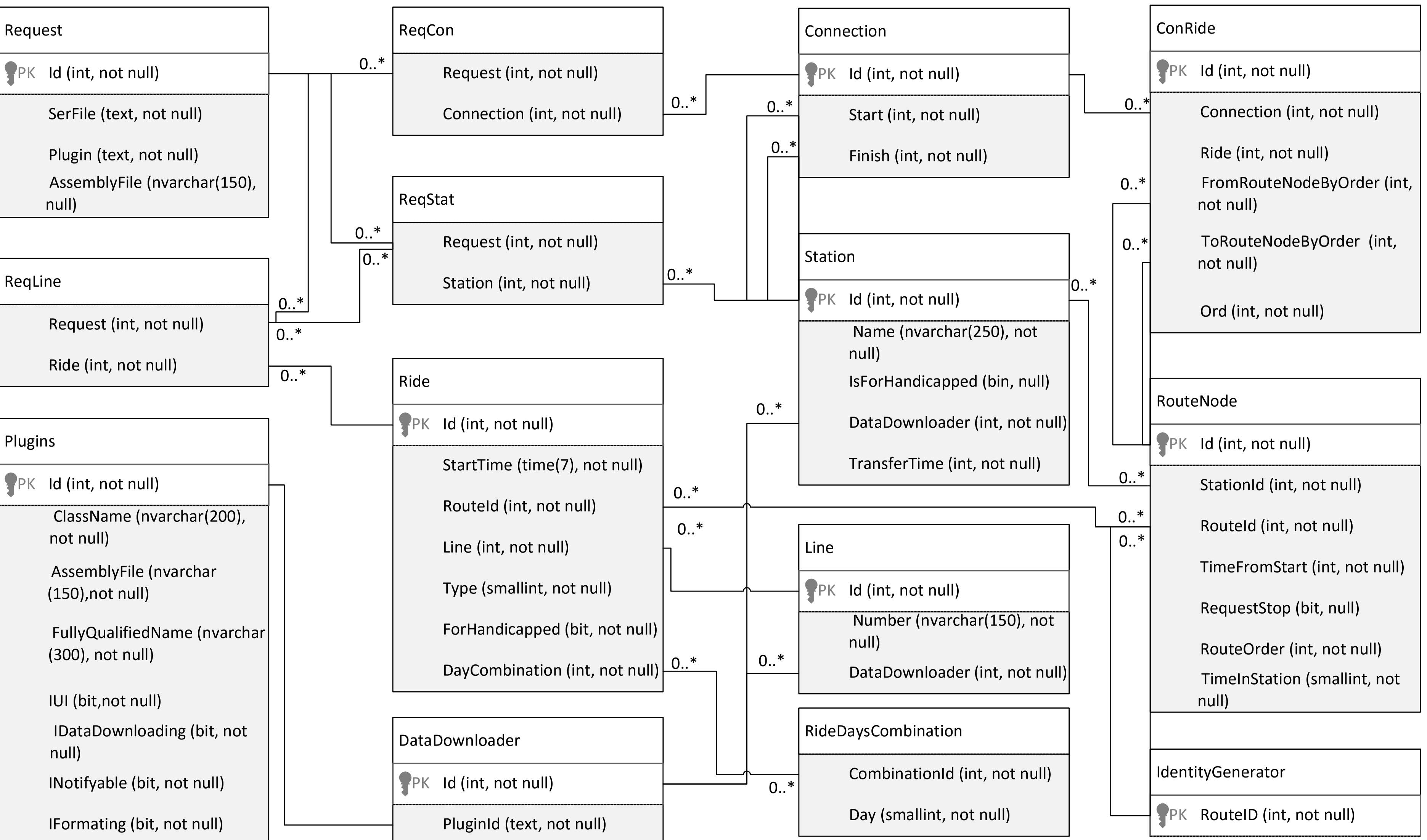
Součástí *Checkeru* je vyhledávač spojení, který je použit, pokud zdroj dat neposkytuje rozhraní *IConnection*. V tomto případě se v závislosti na tom, jaká rozhraní zdroj dat implementuje, vybere správný vyhledávač. Vyhledávače se liší způsobem zpracování dat ze zdroje. Buď vyhledávač konstruuje graf přímo z dat ze zdroje, nebo nejprve uloží data v databázi a následně konstruuje data z databáze.

V jádru je vyčleněná část zpracovávající veškerou komunikaci s databází, deserializaci a rozšiřující nástroje. Definuje rozhraní poskytovaná různým částem systému, především rozhraní pro formátovací rozšíření, a konstruuje výsledky uložené v databázi. Pro unifikaci trasy je zde intenzivně využívána stromová struktura, která umožní efektivně vyhledávat stejnou posloupnost uzlů a jejich parametrů na trase.

D.4 Model databáze

Data v databázi jsou uložena, lze rozlišit na data jízdních řádů, informace o pluginech a přiřazení dat jednotlivým požadavkům. UML schéma je možné vidět na obrázku D.1. Využití tabulek je následující:

- Tabulky *Station,Line,Ride,RouteNode* odpovídají definici dopravní sítě.
- Tabulka *Request* obsahuje požadavky uživatele.
- Tabulka *ReqLine* definuje párování jízd, které tvoří výsledek požadavku.
- Tabulka *ReqStat* definuje párování stanic, které tvoří výsledek pro požadavek.
- Tabulka *ReqCon* definuje párování spojení, která jsou výsledkem pro požadavek.



Obrázek D.1: UML Model databáze

- Tabulka *Connection* obsahuje hlavičky jednotlivých spojení.
- Tabulka *ConRide* obsahuje jízdy jednotlivých spojení.
- Tabulka *DataDownloader* obsahuje zdroje dat, použitých v DB.
- Tabulka *Plugins* obsahuje informace o všech načtených pluginech.
- Tabulka *RideDaysCombination* definuje množinu dnů.
- Tabulka *IdentityGenerator* slouží pro vytváření nových identifikátorů tras.

Výsledek pro požadavek na linku je definován v tabulce *ReqLine*. Výsledek pro požadavek na stanici je definován tabulkou *ReqStat* a zároveň *ReqLine*. Výsledek pro požadavek na spojení je definován v tabulce *ReqCon*.

D.5 Kontrola požadavků na spojení

Má-li požadavek nastavený jako zdroj dat rozšíření, které neimplementuje vyhledávání spojení, pokusí se aplikace sama vyhledat výsledky k požadavku jedním z následujících způsobů.

V případě, kdy zdroj poskytuje seznam a detaily linek, se stáhnou všechny linky ze zdroje do databáze a následně se data z databáze používají pro vyhledávání.

V případě, kdy zdroj dat poskytuje seznam a detaily stanic, jsou pro vyhledávání použita přímo data z rozšíření.

Vyhledávat lze podle času odjezdu nebo příjezdu. Pro obojí je použit stejný algoritmus (viz 3.3) se záměnou počáteční a koncové stanice, počátečního času vyhledávání a znaménka pro posun v čase na trase (vzdálenost mezi uzly). Dále je uveden pouze případ vyhledání s časem odjezdu.

Podle časového filtru pro požadavek se generují počáteční časy vyhledávání. Pro každý čas se vyhledá spojení. Výsledkem pro požadavek je množina spojení všech provedených vyhledávání podle časového filtru.

D.6 ImplicitModules

Plugin *ImplicitModules* (včetně *SimpleConsle*) poskytuje aplikaci formátovaný výstup do tabulky, data z CIS JŘ, notifikaci emailem a konzolové rozhraní.

D.6.1 Výstup do tabulky

Za použití knihovny EPPlus [12] vytváří soubor 'XLSX', který obsahuje výsledky pro jednotlivé požadavky. Vytvoření souboru musí uživatel spustit z uživatelského rozhraní. Vždy je vytvořen jeden soubor pro jeden požadavek. Tvar výsledné tabulky je závislý na typu požadavku.

D.6.2 Notifikace emailem

Notifikace emailem odesílá při vyvolání z jádra prostý text na email uvedený v konfiguračním souboru. V konfiguračním souboru (viz 5.1.6) jsou další nastavení pro připojení k SMTP serveru. Odesílá se prostý text vyjadřující, že byl vytvořen první výsledek nebo nastala změna ve výsledku.

D.6.3 Konzolové rozhraní

Pluigin *ImplicitModules* poskytuje aplikaci uživatelské rozhraní konzolového typu. Napojení konzole je tvořeno částí v knihovně *ImplicitModules* (dále jen příjemce) a spustitelným souborem *SimpleConsole.exe* (dále jen klient). Při spouštění uživatelského rozhraní z jádra, je knihovnou *ImplicitModules* vytvořena pojmenovaná roura a spustí se process *SimpleConsole.exe*, který se připojí k rouře. Jméno roury je definované v konfiguračním souboru (viz 5.1.6).

Příjemce čeká na připojení roury jinou aplikací a následně čte textové příkazy, které zpracovává stavovým automatem. V případě přijetí nevalidních dat příjemce odešle klientovi chybovou zprávu. Pokud se příjemce dostane čtením podle automatu do ukončitelného stavu, spustí se zpracování odpovídajícího kódu. Klient čte příkazy z konzole a přeposílá je příjemci. Čtení konzole je zpracováno stavovým automatem, který umožňuje doplňování znaků a výběr ze jmen stanic (resp. linek). Klient přepošle data příjemci, pokud je ukazatel do automatu v přijímajícím stavu.

Vzhledem ke snaze omezit zadání neplatných požadavků se při spuštění aplikace načítají skrze zdroj dat jména stanic a linek, tedy při prvním spuštění je nutné stáhnout všechna data.

Automat zpracování příkazu konzole

Pro kontrolu uživatelského vstupu je v aplikaci vytvořen automat.

Automat je tvořen jednotlivými uzly a propojením mezi uzly, kdy každý uzel má svůj seznam významů. Samotné uzly obvykle reprezentují jeden přečtený znak a významy definují smysl posloupnosti přečtených uzlů. Při přechodu mezi uzly jsou vždy na dvojici uzlů volány akce pro opuštění a vstup do uzlu. Přechodové akce standardně volají ekvivalentní akce na instancích přiřazených významů. Přechodové akce významů nastavují parametry na builderu, který definuje akci při dokončení čtení (je odeslání dat z klienta do příjemce, nebo konstrukce parametru a vyvolání metody jádra).

D.6.4 Protokol komunikace skrze rouru

Komunikace funguje na principu serveru a klienta, kdy serverem je část *ImplicitModules.dll* a klientem je spuštěná *SimpleConsole.exe*.

Rozlišujeme dva typy komunikace. Prvním typem jsou informační zprávy, které vždy odesílá server a neočekává žádnou odpověď. Druhým typem jsou požadavky, kdy klient odešle požadavek a očekává od serveru odpověď. Pro implementaci zpráv protokolu odeslaných ze serveru slouží třídy ve složce *Console/Protocol*. Zprávy odeslané ze serveru se zpracovávají po řádkách, pokud tedy nějaká část zprávy obsahuje nový řádek, je převeden do řetězce „\n“.

Informační zprávy
Zprávy mají jeden z následujících formátů:

```
APP_INFO:text zprávy z aplikace
```

- Používá se pro informační zprávy uživateli (např. započala kontrola).

```
APP_ERROR:text zprávy z aplikace
```

- Používá se pro informování uživatele o problému (např. nepodařilo se vyhledat výsledek pro kontrolovaný požadavek).

```
APP_FATAL:text zprávy z aplikace
```

- Používá se pro upozornění uživatele o selhání aplikace.

```
APP_PROGRESS:aktuální stav (číslo od 0-1, kdy 1~100 %):název
       aktivity
```

- Požívá se pro informování o průběhu operace, na kterou se čeká (např. prvotní stažení dat při spuštění konzole).

Odeslání požadavku
Požadavek odeslaný z klienta na server se řídí pravidly automatu (viz B). Požadavek se vyhodnotí v momentě, kdy uživatel ukončí zadání příkazu (viz 5.1.5). Pro většinu požadavků odeslaných z klienta na server odpoví server výsledkem, záleží ovšem na implementaci jednotlivých požadavků.

Jako výsledek může být poslán seznam hodnot.

```
RES_:{seznam hodnot}
```

- Hodnoty v seznamu jsou oddělené středníkem. Pokud nějaká hodnota obsahuje tento znak, je před znak přidáno „\“. Pokud měla v původním zápisu znak „\“, pak je tento znak zdvojen.

Další variantou výsledku je více seznamů hodnot.

```
RES_BEGIN:počet seznamů hodnot
RES_:seznam hodnot
 . . .
RES_END
```

Poslední variantou je chybový výsledek.

```
RES_ERROR:popis chyby
```

D.6.5 Zdroj dat CIS JŘ

ImplicitModules poskytuje aplikaci data z CIS JŘ. Poskytnutí těchto dat je rozděleno na data ze souborů PDF a data ze souborů JDF a data jsou sjednocena do jednoho výsledku.

Takto bylo nutné implementovat rozhraní, jelikož data exportována na `ftp://ftp.cisjr.cz/` jsou v různých formátech a formáty se postupně mění. Ve `ftp://ftp.cisjr.cz/draha/mestske/` jsou data o tramvajích a metru městských hromadných doprav (v souborech PDF, nebo aktuálně i JDF) a ve `ftp://ftp.cisjr.cz/JDF/JDF.zip` jsou data o autobusech (ve formátu JDF).

Při stahování a zpracování dat oba způsoby ověřují, zda jsou soubory na ftp serveru novější než již stažené soubory a zda již nemají jejich data mezi dočasnými soubory, aby se omezilo opakované stahování a práce se stejnými daty. Celá tato část je navržena tak, aby bylo možné využít paralelního stahování a zpracovávání souborů. Paralelní zpracování však bylo potřeba omezit z bezpečnostních důvodů cílového serveru.

PDF

Pro získání dat z PDF se používá knihovna iTextSharp [1], které je předána definice způsobu extrakce dat. Data jsou následně uložena ve formě textových souborů. Pro jeden pdf soubor je vytvořen jeden hlavičkový soubor (*x.txt*) a jeden hlavní soubor (*x.txt.body*). Soubory obsahují kontrolní součet. Pokud je soubor poškozený, stáhnou se a získají data znovu z pdf souboru.

Data jsou uložena v souborech, protože extrakce dat z PDF je velmi náročná, ale je možné ji provést pouze jednou. Data jsou rozčleněna do dvou souborů, které jsou uzpůsobeny pro dotazy jádra aplikace. Hlavičkový soubor stačí pluginu pro vyhledání správných souborů a druhý slouží pro uložení detailních informací.

Hlavičkový soubor (.txt) obsahuje číslo linky, platnost dat v souboru, stanice na lince a kontrolní součet.

Soubor obsahuje řádky tří typů:

```
L{číslo linky};{datum začátku platnosti};{datum konce platnosti}
S{id stanice};{pro hendikepované};{doba přestupu};{jméno stanice}
H{SHA1};{MD5}
```

Kdy *id stanice* je identifikátor v rámci dat k jednomu pdf souboru. *Pro hendikepované* je 1, pokud je stanice uzpůsobena pro hendikepované, jinak 0. *Doba přestupu* je minimální doba potřebná pro přestup v rámci stanice, udávaná v minutách.

Hlavní soubor (.txt.body) obsahuje řádky tří typů:

```
N{id trasy};{číslo bodu};{na znamení};{vzdálenost};{id stanice}
R{id trasy};{typ jízdy};{pro hendikepované};{id jízdy};{dny};{odjezd}
H{SHA1};{MD5}
```

Kdy *id trasy* je identifikátor v rámci dat k jednomu pdf souboru. *Vzdálenost* je vzdálenost od počáteční stanice v minutách (celé číslo). *Id stanice* je identifikátor v rámci dat k jednomu pdf souboru. *Číslo bodu* označuje pozici stanice na trase. *Typ jízdy* je celé číslo korespondující s hodnotou enum *TransportType*. *Pro*

hendikepované je 1, pokud je jízda uzpůsobena pro hendikepované, jinak 0. *Id jízdy* je identifikátor převzatý z PDF. *Dny* je seznam celých čísel korespondující s hodnotou enum *Days*, kdy je jízda v provozu oddělené '-'. *Odjezd* je čas odjezdu z první stanice trasy jízdy ve formátu *hhmmss*.

JDF

Pro zpracování JDF je vytvořen celý systém extrakce dat. Formát se používá v různých verzích a jeho specifikace je popsána v [13, 14, 15].

Celý soubor JDF se skládá ze složek, které obsahují soubory popsané JDF formátem. Při extrakci se rozbalí komprimovaný soubor. Následně se pro jednotlivé složky v souboru načtou základní informace, tj. jaké linky a stanice jsou v složce popsány a platnost těchto dat. Všechna data se načtou až ve chvíli, kdy se vytváří výsledek žádosti jádra aplikace. Pro čtení jednotlivých složek se použije dle verze JDF strategie extra dat.

D.7 Knihovna ResourcesForPlugins

Knihovna *ResourcesForPlugins* definuje rozhraní mezi jádrem aplikace a pluginy. Knihovna také poskytuje další nástroje pro porovnání a validaci dat nebo logování. Logování poskytuje správu souborů (včetně mazání) a různé úrovně logování.

E. Implementační detaily mobilní aplikace

E.1 Použité knihovny

Pro správu uživatelů (především jejich identifikaci) a odesílání zpráv z desktopu na mobil je použit cloud *Firebase* a jeho podpůrné knihovny (*auth* a *cloud messaging*).

E.2 Mobilní aplikace

Aplikace je rozdělena dle vzoru MVC. Data jsou uložena v lokální *SQLite* databázi (součást systému Android), ke které přistupujeme pomocí knihovny *GreenDao*. Pro HTTP volání je využita knihovna *Volley*.

Zdrojové kódy jsou rozděleny do balíčků takto:

- **activities** - Aktivity
- **base** - Základní abstraktní třídy
- **notification** - Vyváření mobilních notifikací
- **fragments** - Části aktivit
- **model.db** - Model lokální databáze a adaptéry
  - Definice entit
  - **adapters** - Adaptéry seznamů pro zobrazení
    - **base** - Třídy, z nichž se odvozují adaptéry
    - **data** - Třídy s předpracovanými daty z entit
  - **asyncTask** - Asynchronní práce s databází
  - **cache** - Cache pro načítání z databáze
  - **helpers** - Pomocné třídy pro práci s databází (převody, dotazy)
- **synchronization** - Zpracování příchozích a odchozích zpráv
  - **dto** - Definice zpráv
    - **base** - Kontexty pro zpracování a jejich rozhraní
- **types** - Základní pomocné třídy
- **views** - Struktury pro zobrazení dat

E.3 Rozšíření hlavní aplikace

Do hlavní aplikace je přidáno rozšíření *MobileInterface*, které spravuje komunikaci s mobilní aplikací. *MobileInterface* se skládá z 5 částí: periodického kontrolního cyklu, uložení dat odeslaných do mobilu, komunikace, zpracování výsledku a implementace rozhraní *IUI*,*INotifyable* a *IFormatting*.

E.3.1 Periodický kontrolní cyklus (*Looper*)

Po spuštění *MobileInterface* z jádra, je spuštěn jako *Task* periodický cyklus. Tento cyklus periodicky s časovou periodou nastavenou v konfiguračním souboru kontroluje, zda nenastaly změny v základních datech (linky, stanice, rozšíření), či v požadavcích. V případě detekce změny odešle zprávy obsahující změny do mobilní aplikace. Na konci cyklu jsou staženy a zpracovány příchozí zprávy z prostředníka.

E.3.2 Lokální uložení dat

Storage ukládá data lokálně na disk. Uložená data kopírují aktuální stav dat v mobilní aplikaci nebo uchovávají párování (resp. generování) identifikátorů v kontextu mobilní aplikace.

Data jsou uložena do lokálních souborů. Většina souborů obsahuje JSON serializovaný objekt. Výjimkou je soubor s párovanými identifikátory požadavku jádra a mobilu.

Druhou výjimkou je soubor *auth.ser*, který obsahuje přihlašovací údaje pro Firebase, zadané uživatelem. Tento soubor je šifrovaný pomocí Windows podle přihlášeného uživatele.

Soubor s párovanými identifikátory požadavku

Soubor (*requests.my*) obsahuje identifikátory požadavku z jádra párované s identifikátory požadavku pro mobil (viz 2.13.2).

Soubor je rozčleněn do řádků, kdy řádek je dále dělen znakem ';' na pole. První pole v řádku je vždy identifikátor v kontextu mobilu. Další pole jsou identifikátory jádra párované s tímto identifikátorem, kdy druhé pole je nejstarší párování a poslední pole je nejaktuálnější párování.

V souboru může existovat více řádků pro jeden identifikátor v kontextu mobilu. V tomto případě je nejstarším párováním první řádek s identifikátorem a druhé pole tohoto řádku. Nejnovějším je poslední řádek s identifikátorem a poslední pole tohoto řádku.

Mohou také existovat řádky, kde je první pole prázdné, to jsou řádky, kdy ještě nebyl identifikátor přiřazen.

E.3.3 Odesílání zpráv

Dispatcher serializuje a odesílá zprávy na příslušnou adresu, zvolenou dle typu volání a nastavení v konfiguračním souboru, přes HTTP. *Dispatcher* v případě pokusu odeslat větší zprávu než povoluje limit FCM, nejprve uloží zprávu v prostředníkovi a poté odešle zprávu o uložení zprávy přes FCM. Pokud odeslání

selže, funkce této třídy vrací zápornou hodnotu a volající se musí vypořádat se selháním.

Třída zároveň zpracovává a odesílá notifikace z jádra aplikace. Pokud se jedná o notifikaci změny výsledku, vyvolá zpracování změny a výsledek odešle.

E.3.4 Zpracování výsledku

ChangeProcessor porovná nový výsledek se starým (lokálně uloženým) a vytvoří zprávu se změnou. Zároveň převede identifikátor z jádra na identifikátor v kontextu mobilní aplikace (tj. stanice, linky, požadavek).

Při spuštění zpracování není předán aktuální výsledek. Výsledek z jádra je získán pomocí zpracování callbacku.

E.3.5 MobileInterface.dll.config

Ke knihovně *MobileInterface* může být vytvořen konfigurační soubor. Tento soubor musí mít strukturu definovanou E.1. Libovolné elementy *settings* mohou být vynechány.

Elementy s atributem *name* a s hodnotou *targetMessages*, *targetNotification*, *targetToken*, *targetPayload*, *targetSignIn* a *targetRefreshToken* slouží pro nastavení URL jednotlivých HTTP koncových bodů prostředníka.

Element s atributem *name* a s hodnotou *LoopDelay* slouží pro nastavení periodicity kontrolní smyčky, které mimo jiné stahuje zprávy z prostředníka.

E.3.6 Projekty

Zdrojové kódy pro toto rozšíření se skládají ze tří projektů.

- *MobileInterface*
  - Je hlavním projektem zajišťujícím veškerou funkcionalitu.
- *MobileInterfaceTest*
  - Obsahuje unit testy pro *MobileInterface*
- *KernelMock*
  - Projekt pro lokální testování a vývoj, který mockuje jádro aplikace.

E.4 Prostředník

Prostředník slouží pro propojení mobilní aplikace s desktopem. Jeho hlavní funkcí je předání zprávy, tedy uchování v databázi, poskytnutí možnosti stažení zprávy a následné smazání, a restrikci přístupu.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<configSections>
 <sectionGroup name="applicationSettings"
type="System.Configuration.ApplicationSettingsGroup, System,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
   <section name="MobileInterface.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
 </sectionGroup>
</configSections>
<applicationSettings>
 <MobileInterface.Properties.Settings>
   <setting name="targetMessages" serializeAs="String">
     <value></value>
   </setting>
   <setting name="targetNotification" serializeAs="String">
     <value></value>
   </setting>
   <setting name="targetToken" serializeAs="String">
     <value></value>
   </setting>
   <setting name="LoopDelay" serializeAs="String">
     <value></value>
   </setting>
   <setting name="targetPayload" serializeAs="String">
     <value></value>
   </setting>
   <setting name="targetSignIn" serializeAs="String">
     <value></value>
   </setting>
   <setting name="targetRefreshToken" serializeAs="String">
     <value></value>
   </setting>
 </MobileInterface.Properties.Settings>
</applicationSettings>
</configuration>
```

Kód E.1: Šablona souboru MobileInterface.dll.config

E.4.1 API

Prostředník má vystavené HTTP endpointy, které očekávají tělo ve tvaru JSON. V hlavičce X-Authorized je v případě potřeby UID vyžadován platný validní OAuth token. Tyto endpointy můžeme rozčlenit dle funkcionality:

- Předání zprávy - $\{serverdomain\}\backslash messages$
  - GET - Vrátí všechny uložené zprávy pro UID.
  - POST - Uloží zprávu z těla pro dané UID.
  - DELETE - Smaže uložené zprávy podle seznamu id předaných v těle.
- Uložení obsahu - $\{serverdomain\}\backslash fcm\backslash load\backslash\{messageid\}$
  - GET - Vrátí obsah zprávy dle parametru *message id*.
  - PUT - Vloží obsah zprávy.
    - Tělo požadavku

```
{
  "message_id": string,
  "data": string
}
```

  - DELETE - Smaže obsah zprávy specifikovaný parametrem *message id*.
- Odeslání zprávy - - $\{serverdomain\}\backslash fcm$
  - POST - Odešle zprávu do *firebase cloud messaging (FCM)*.
- Uložení tokenů - $\{serverdomain\}\backslash token$
  - POST - Uloží uživatelův identifikátor zařízení.
    - Tělo požadavku

```
{
  "idtoken": string
}
```

F. Kompilace jádra a základního rozšíření

F.1 Části programu

Všechny zdrojové kódy k uvedeným částem aplikace (včetně rozšíření) jsou v přiložených v projektech. Každý projekt lze přeložit zvlášť.

- *TransportSystemKernel* – hlavní aplikace systému, spouští celý systém
  - Tato aplikace musí být spouštěna ze složky, v níž se nachází. V této složce musí být složka *Plugins*.
  - Složka *Plugins* slouží pro rozšíření aplikace. Při základní distribuci bude obsahovat knihovny rozšíření *ImplicitModules*.
  - Složka *Data* (k níž vede cesta v konfiguračním souboru) musí obsahovat databázový soubor aplikace a log soubor (*DB.mdf* a *DB_log.ldf*). Do této složky by uživatel neměl zasahovat.
- *ImplicitModules* – DLL, které obsahuje základní pluginy (Excel formátovač výstupu, zdroj dat z CIS JŘ, a komunikační část ke konzoli)
  - Pro svůj běh vyžaduje knihovny *EPPlus.dll*, *itextsharp.dll* a konfigurační soubor *ImplicitModules.dll.config*.
- *SimpleConsole.exe* – konzole, kterou se ovládá aplikace
  - Pro svůj běh vyžaduje konfigurační soubor *SimpleConsole.exe.config*.
  - Konzole komunikuje s hlavní aplikací skrze rozhraní v *ImplicitModules*, pokud hlavní aplikace nemá načtený plugin *ImplicitModules* a nebude spuštěno komunikační rozhraní v *ImplicitModules*, nezdaří se *SimpleConsole* připojit k hlavní aplikaci.
  - V případě, že se konzole zavře, hlavní aplikace zůstane běžet na pozadí a okno konzole je možně připojit k aplikaci přímým spuštěním *SimpleConsole.exe*.

Pro lepší práci jsou projekty přidány do jednotlivých řešení (.sln):

- *TransportSystem.sln*
  - Obsahuje všechny základní projekty a projekty s unit testy.
- *SimpleConsoleIntegrationTest.sln*
  - Je řešení sloužící pro integrační test napojení *SimpleConsole.exe* na jádro aplikace.
- *Installer.sln*
  - Je řešení pro tvorbu instalačního souboru. Obsahuje všechny projekty pro běh aplikace včetně rozšíření pro komunikaci s mobilem.

F.2 Preprocesorové direktivy

Překlad je možné ovlivnit následujícími direktivy:

- U překladu *ImplicitModules*
  - NODOWNLOAD
    - ∗ Zajistí, že se nestahují data z CIS, ale načtou se z počítače.
      - · Slouží pro ladění, kdy se data načtou do PC jednou a následně se používají znovu tatáž data.
- U překladu *TransportSystemKernel*
  - DBLOADED
    - ∗ V případě vyhledávání přeskočí načítání dat do DB.
      - · Slouží pro ladění, kdy se data načtou do DB jednou a následně se používají znovu tatáž data.
  - NOGUI
    - ∗ Zajistí, že se při startu aplikace nespustí uživatelské rozhraní.
  - SHORTLOOP
    - ∗ Zajistí, že se mezi kontrolami čeká pouze 3 sekundy.

G. Obsah elektronické přílohy

- AllInOne.exe
  - Instalační soubor desktopové aplikace
- app-release.apk
  - Soubor mobilní aplikace
- Data/Experiment/Data/bus/JDF.zip
  - Data autobusů stažená z CIS JŘ 4. 6. 2018
- Data/Experiment/Data/draha/JDF.zip
  - Drážní data stažená z CIS JŘ 4. 6. 2018
- Data/Experiment/Results/output.txt
  - Výstup spuštění experimentu (na konci jsou vypsány výsledky uvedené v tabulce 4.1)
- Data/Experiment/Results/TransportSystemKernel.Searcher
  - Záznam jednotlivých hodnot (a výsledky) vlastního algoritmu
- Data/Experiment/Results/TransportSystemKernel.TDOriginalSearcher
  - Záznam jednotlivých hodnot (a výsledky) originálního časově závislého algoritmu
- TransportSystem/*
  - Obsahuje všechny zdrojové kódy a projekty hlavní aplikace
- MobileInterface/*
  - Obsahuje zdrojové kódy rozšíření pro hlavní aplikaci o propojení s mobilem
- AndroidApp/MyTransportation/*
  - Obsahuje zdrojové kódy a projekt mobilní aplikace
- PhpExchange/*
  - Obsahuje skripty pro PHP prostředníka
- Firebase/*
  - Obsahuje skripty pro Firebase prostředníka