Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Bakalářská práce

Řídící SW pro modelové kolejiště

Autor práce: Vojtěch Lapuník
Vedoucí práce: Ing. Petr Weissar, Ph.D.

Plzeň 2019

Abstrakt

Náplní této bakalářské práce je vývoj aplikace pro snadné řízení modelového kolejiště na FEL ZČU v jeho současné podobě. Práce může sloužit, v první řadě jako uživatelská příručka pro všechny, jež chtějí toto kolejiště provozovat. Ale v neposlední řadě, také jako popis aplikace do takové míry, aby bylo možné na práci navázat a aplikaci stále vyvíjet. V první části je čtenář seznámen s problematikou struktury kolejiště, digitálního řízení a formátu zpráv pro řídící jednotky. Následuje část popisující první verzi aplikace i s jejími chybami a nedokonalostmi. Poslední a zároveň největší část je věnována podrobnému popisu druhé verze aplikace, jež se stala hlavním produktem této práce. V poslední části jsou rozebrány a vysvětleny významy jednotlivých tříd a metod v nich obsažených, včetně popsání myšlenek jednotlivých algoritmů. Zadání práce bylo splněno nad rámec požadovaných bodů. Povedlo se vytvořit aplikaci, jež dokáže zobrazovat a pracovat s obsazeností kolejových úseků, řídit lokomotivy manuálním i autonomním řízením, a to i pomocí vzdáleného přístupu. Navíc je aplikace velmi lehce rozšiřitelná a připravená pro další vývoj. V závěru práce jsou shrnuty všechny důležité klady, zápory a poznatky získané během vytváření aplikace.

Klíčová slova

modelové kolejiště, digitální řízení, DCC, vývoj řídící aplikace, TCP,

Abstract

Lapuník, Vojtěch. *Controlling SW for model railway* [*Řídící SW pro modelové kolejiště*]. Pilsen, 2019. Bachelor thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Applied Electronics and Telecommunications. Supervisor: Petr Weissar

---

Text of the abstract in English...

Keywords

model railway, digital control, DCC, software development, TCP,

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 4. června 2019

Vojtěch Lapuník

.....

Podpis

Obsah

Seznam obrázků

Seznam tabulek

1

Úvod

Modelové kolejiště, nacházející se na pátém patře Fakulty elektrotechnické v areálu Západočeské univerzity v Plzni, realizuje dlouhodobý projekt pro rozvoj studentů. Pro někoho může být modelové kolejiště jen hračka na volný čas, ale zde je využíváno k praktické výuce elektroniky a programování. Projekt zažil své znovuvzkříšení minulý rok, kdy byli vytvořeny dvě bakalářské práce na téma jednotka generující protokol DCC a jednotka řízení úseků. To nastartovalo o vláčky opětovný zájem a už nyní je jisté, že na příští rok budou vypsány další témata týkající se této problematiky.

Jak již bylo zmíněno v minulém roce byl vybudován systém jednotek pro řízení kolejiště tak, aby bylo možné uvést lokomotivy do pohybu. Avšak chyběl velice důležitý dílek skládačky, a sice řídící software pro stolní počítač. Systém tvořený z dílčích jednotek je propojen se stolním počítačem přes konektor USB, tudíž cílem bylo vytvoření aplikace, která krom jiného dokáže nabídnout komunikaci přes sériový port. Zároveň bylo potřeba, aby aplikace disponovala pohodlným uživatelským prostředím a byla do budoucna lehce rozšiřitelná. Rozšiřitelnost aplikace byl vůbec jeden z nejdůležitějších požadavků na ovládací aplikaci. Protože současně s vývojem celého kolejiště a s přidáváním řídících jednotek pro všelijaké účely stoupá také náročnost na řízení. Jelikož jsou řídící jednotky konstruované velmi specializovaně, žádná již dostupná aplikace neodpovídá našim požadavkům a vývoj tím dostává opravdový smysl.

V první části práce je vysvětlována spíše teoretická stránka problematiky. Ať už se jedná o představení struktury kolejiště, popsání jednotlivých bloků, nebo seznámení se stavbou zpráv probíhajících v systému. Přes uživatelskou příručku, popisující první verzi aplikace a především její využití, je čtenář doveden k majoritní náplni práce. Tou je detailní popis konstrukce druhé verze aplikace. A to včetně poukázání na všechny důležité myšlenky, jež byli využity pro její tvorbu.

2

Základní seznámení s modelovým kolejištěm

2.1 Provedení kolejiště

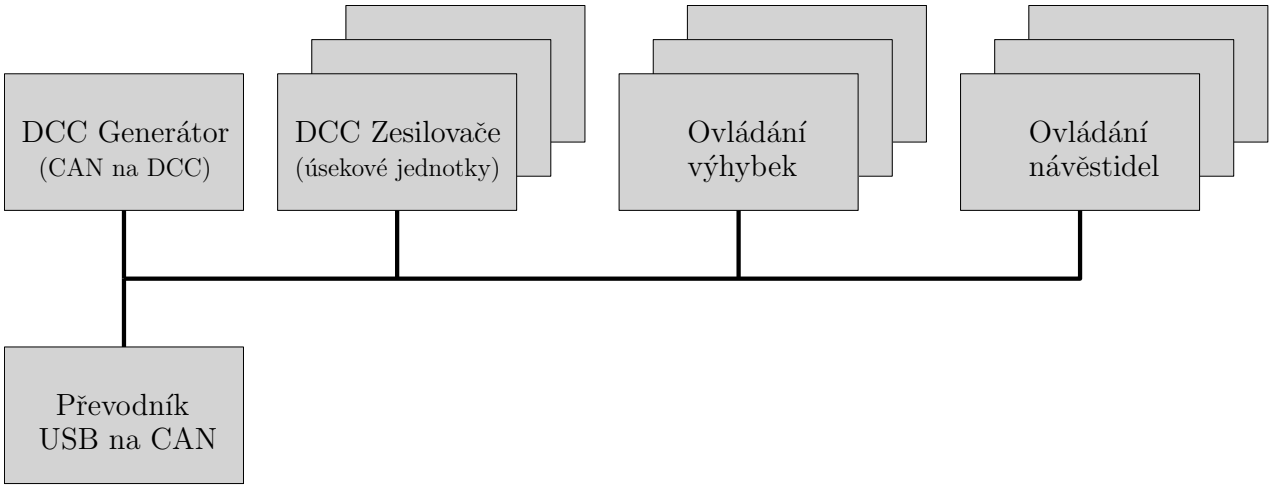
Neznalost provedení struktury modelového kolejiště by mohla vést k nedokonalému pochopení této práce, proto si musíme nejprve ujasnit, jak kolejiště vypadá a jak bylo navrženo.

Modelové kolejiště je v měřítku TT což odpovídá 1:120 a skládá se ze tří zastávek, pracovně nazvaných Beroun, Karlštejn a Lhota. Ve stanici Beroun je XY oddělených kolejových úseků pro manipulaci i s vlaky o několika pevně spojených nápravách. Stanice také disponuje, bohužel zatím stále nepřipojenou, točnou a depem pro lokomotivy. Oproti stanici Beroun je stanice Karlštejn menší, obsahuje pouze YZ kolejových úseků s malou možností manipulací s vlaky. Nejmenší stanicí je Lhota, zatímco propojení stanic Beroun a Karlštejn je provedeno dvoukolejnou tratí, do Lhoty vede pouze jedna kolej ze stanice Beroun, která je rozvětvena do tří nástupišť, koleje jsou zde zakončené zarážkami a není možné ve stanici vlak otočit.

2.2 Systém řízení kolejiště

Systém řízení se skládá z několika úrovní, pro pochopení řízení a celkového fungování je nutno mít alespoň elementární znalost o každé úrovni.

Pro začátek je potřeba zdůraznit že se jedná o digitální způsob řízení. U analogového řízení spočívá princip v prostém přivedení napětí na kolejnice, které mohou být také rozděleny do úseků a toto napětí budí stejnosměrný motor uvnitř každé lokomotivy a rozjíždí tak všechny lokomotivy které se na úseku vyskytují. Se změnou amplitudy napětí, které přivádím na kolejnice stoupá, nebo naopak klesá, rychlost vlaků a pomocí přivedení napětí opačné polarity, lokomotivy mění svůj směr jízdy. Tento systém řízení nedovoluje řízení návěstidel a výhybek přes kolejnice a je tak potřeba přivést k těmto prvkům další vodiče. Celkově vzato systém analogového řízení nenabízí ani z daleka takové možnosti



Obr. 2.1: Koncepce řízení vlaků TT

jako řízení digitální. Jelikož se u našeho kolejiště jedná o digitální řízení, popíšeme si ho důkladněji níže.

Základem řízení kolejiště je stolní počítač, na kterém je spuštěn software pro ovládání celého systému. Tento software je majoritní náplní této práce, proto si nejprve blíže představme zbytek komunikace kolejiště s PC. Do stolního PC je přes port USB připojen převodník USB signálu na signál CAN. Tento převodník je pomocí nestíněné kroucené dvoulinky UTP propojen se zbylými bloky modelového kolejiště, kterými jsou generátor signálu DCC (Digital Command Control), Zesilovač signálu DCC a poté další bloky pro řízení návěstidel a výhybek. Viz obr. 2.1.

2.2.1 Generátor DCC signálu a DCC zesilovač

Do generátoru DCC signálu přichází data určené k vyslání do kolejí, to znamená příkazy k jízdě lokomotiv, přehození výhybek nebo ovládání návěstidel. Generátor tyto data převede do podoby DCC signálu a přes zesilovač vypustí do kolejiště, zároveň se také stará o jejich neustálé opakování. K řízení kolejiště je potřeba pouze jedna jednotka Generátoru DCC, neboť do všech kolejových úseků se posílá ten jistý signál. Naproti tomu DCC zesilovač kromě funkce zesílení signálu z generátoru a jeho vyslání do kolejí, provádí ještě měření proudu na jednotlivých kolejových úsecích, které potřebujeme pro vyhodnocení obsazenosti úseků, tudíž čím více úseků chceme rozlišit, tím více jednotek je potřeba. Důležitou funkcí DCC generátoru je také posílaní watchdog zpráv do řídícího počítače, a naopak hlídání pravidelně příchozích zpráv z řídícího počítače. Nepřichází-li zprávy po sběrnici CAN, je všem vlakům nastavena nulová rychlost, jestliže jednotka generátoru vypadne, přeruší napájení zesilovače tak, aby nedošlo k rozjetí všech vlaků na stejnosměrné napětí.

2.2.2 Propojení kolejnic s DDC zesilovačem

Kolejnice jsou propojeny s jednotkami pro řízení jednotlivých kolejových úseků, kde každá z jednotek budí až osm úseků. Úsekem je chápána část kolejnic, která je od zbytku kolejnic elektricky izolována vzduchovou mezerou a lze tak na základě proudového odběru lokomotivy identifikovat obsazenost tohoto úseku. V současné době je celá trať rozdělena pouze do osmi úseků a těmi jsou krátké úseky ve stanicích, a to jeden úsek ve stanici Lhota, dva úseky ve stanici Karlštejn a tři úseky ve stanici Beroun. Zbylé dva úseky jsou využity pro rozvod takzvané širé trati, tou je myšlen zbytek kolejí kde se mohou vlaky ocitnout, každý z úseků napájí polovinu širé trati. Model také obsahuje místa kolejnic, kde prozatím není připojena žádná úseková jednotka a tudíž dáme-li vlak na takové místo, systém ho nedokáže rozjet. Tyto místa jsou, krátký okruh okolo modelu hradu nedaleko stanice Lhota a také přechod mezi dvěma úseky širé trati.

2.2.3 Provedení spojení systému s lokomotivou

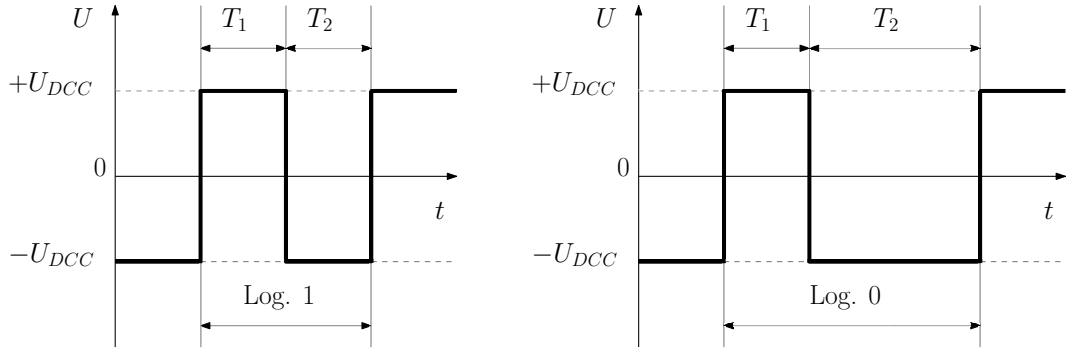
Do jednotlivých lokomotiv je potřeba přivést jak napájení, tak i řídící pokyny. Obě tyto funkce zajišťuje připojení lokomotivy přes její dvojkolí k vodivým kolejnicím, které nesou DCC signál ze zesilovače. Každá lokomotiva má v sobě svůj dekodér, který dokáže vyčíst z paketu DCC signálu adresu a zjistit tak, komu je zpráva adresovaná. Tento způsob komunikace je vhodný zejména protože umožňuje nezávislé řízení více lokomotiv na jednom úseku. Dekodér lokomotivy je důvodem, proč má stojící i jedoucí lokomotiva stálý odběr proudu z kolejnic, a proto lze stroj na kolejnici identifikovat a vyhodnotit obsazenost úseku. Pro identifikaci vagonů je potřeba do jejich dvojkolí přidat zatěžovací rezistor. Stejným systémem lze ovládat návěstidla nebo výhybky.

2.3 Komunikační protokol DCC

Standart DCC je rozhraní pro řízení lokomotiv definované NMRA (National Model Railroad Association). Tento systém dovoluje nejen řízení lokomotiv na kolejišti bez rozdělení kolejiště do více úseků, jelikož se jedná o adresovou komunikaci, ale také o snadné řízení příslušenství kolejiště, jakým mohou být návěstidla, výhybky nebo například osvětlení domečků podél tratě.

2.3.1 Digitální signál v kolejích

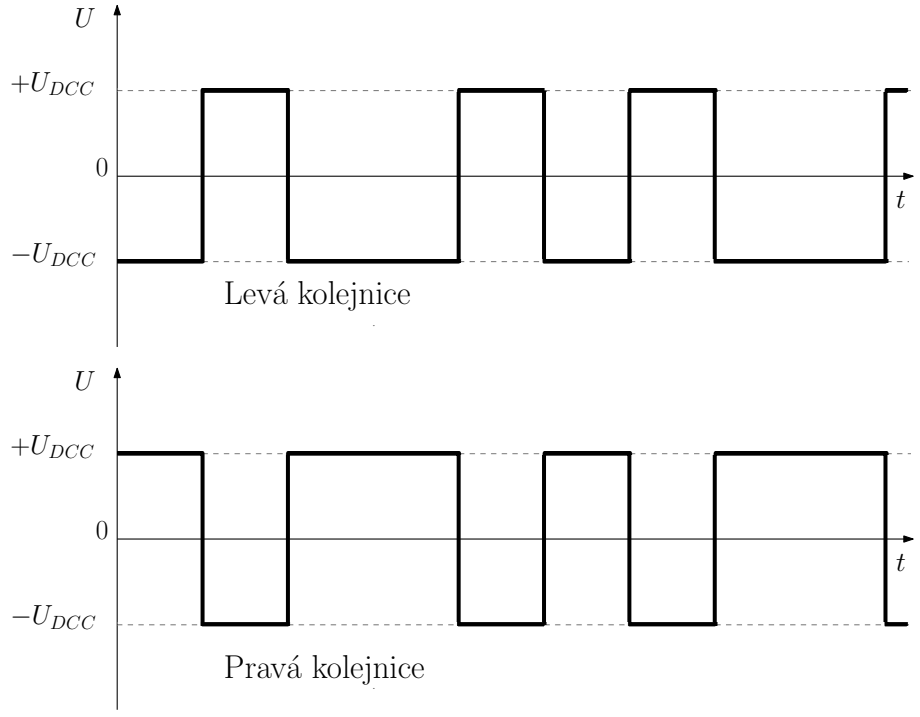
Pro obdélníkový signál v kolejích je použito kódování pomocí šířky impulzu. Logická jednička je reprezentována periodou složenou ze dvou intervalů se shodnou velikostí 52 až 64 μs. Viz Obr. 2.2. Zatímco u logické nuly se části periody T_1 a T_2 mohou lišit, ale i přes to musí být v rozmezí 95 až 9900 μs. Viz obr. 2.2.



(a) DCC signál reprezentující log. 1 (b) DCC signál reprezentující log. 0

Obr. 2.2: Průběhy DCC signálu při logických stavech log. 1 a log. 0

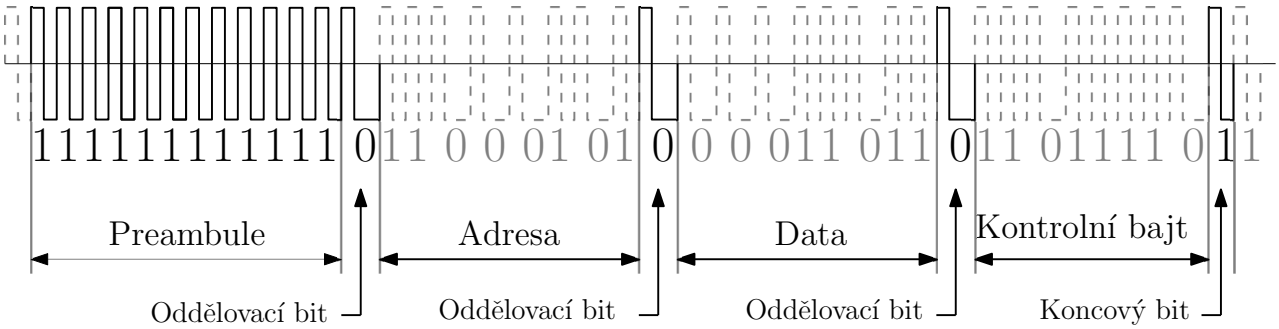
Změnou poměru intervalů pro logickou nulu lze ovlivnit celkovou stejnosměrnou složku signálu a tím řídit lokomotivy které nemají podporu DCC signálu. Zvýšením T_1 oproti T_2 lze dosáhnout rozdílu potenciálu mezi kolejnicemi a rozběhnout tak stejnosměrný motor v analogových lokomotivách, pokud bychom potřebovali změnit směr lokomotivy, postačí pouze snížit část periody T_1 a zvýšit T_2. Signál se v takové podobě nachází pouze v jedné z kolejnic a ve druhé kolejnici je signál zrcadlen, tím je zajištěno že dekodér může plnohodnotně rozpoznat logickou hodnotu stavu. Viz obr. 2.3.



Obr. 2.3: Příklad průběhu signálu DCC v jednotlivých vodičích(kolejnicích)Vyobrazená logická hodnota je: 010

2.3.2 Pakety DCC

Pro bezpečnou komunikaci je použita komunikace pomocí tzv. paketů. Paketem je myšlena zpráva s detekovatelným začátkem a koncem, tak aby byla jasně rozpoznatelná v toku dat. Pakety mají jednotnou strukturu, složenou ze čtyř částí, a to preambule, adresová část, datová část a část kontrolní. Všechny části mají formát délky v podobě násobků 8 bitů, až na preambuli, která obsahuje minimálně 12 po sobě jdoucích jedniček. Jednotlivé části jsou od sebe odděleny bitem hodnoty log. 0 a to z důvodu nezaměnitelnosti s preambulí, pokud by nastala situace, kde datová část by obsahovala dva bajty samých jedniček, dekodér by takovou posloupnost vnímal jako začátek nového paketu. Následující část paketu obsahuje informaci o adrese zařízení, ať už se jedná o lokomotivu, výhybku, nebo návěstidlo. Adresy námi používaných lokomotiv viz tab. 2.3. U běžných paketů je adresová část dlouhá obvykle pouze jeden bajt, ale v případě potřeby lze použít lokomotivy s délkou adresového prostoru dva bajty, anebo u některých zařízení lze použít jako adresu, kromě adresového bajtu, také začátek bajtu datového. Datová část obsahuje informace o příkazu k určité činnosti zařízení, například povel k jízdě, nebo rozsvícení světel lokomotivy, ale také k překlopení výhybky nebo ke změně signalizace návěstidla. Poslední část paketu je kontrolní bajt určený pro detekci chyb. Tento bajt je prostou aplikací logické funkce XOR předchozích bajtů.



Obr. 2.4: Příklad kompletního DCC paketu

Níže jsou uvedeny zásadní pakety pro ovládání modelového kolejiště DCC signálem. Nejsou zde uvedeny úplně všechny pakety, které dekodéry v zařízeních dokáží rozpoznat, ale pouze ty stěžejní pro naše potřeby.

2.3.2.1 Základní pakety

1. IDLE paket

   Tento paket slouží k udržení aktivní sběrnice a také k napájení všech dílů kolejiště, proto je nutné, aby se tento paket posílal v určitých intervalech nepřetržitě po celou dobu provozu kolejiště.

2. RESET paket

   Příkaz pro nastavení DCC dekodérů jednotlivých komponent do výchozího stavu.

praembule	0	1111 1111	0	0000 0000	1111 1111	1

Tab. 2.1: IDLE paket

praembule	0	0000 0000	0	0000 0000	0000 0000	1

Tab. 2.2: RESET paket

Adresa (HEX)	Lokomotiva
0x03	Brejlovec
0x05	Desiro (Kamera)
0x06	Taurus Railion
0x07	DB204 274-5
0x09	Ragulin
0x0A	ICE
0x0B	Taurus DHL
0x0C	Herkules Priessnitz
0x0D	Para 555
0x0E	T334 Rosnicka
0x0F	Taurus EVB
0x10	ES363

Tab. 2.3: Adresy lokomotiv

2.3.2.2 Pakety pro ovládání lokomotiv

1. **Základní paket pro jízdu**

   Paket pro jízdu je ve formátu viz tab. 2.4 resp. tab. 2.5. Datový bajt se skládá z předpony typické pro základní paket pro jízdu 01, směrového bitu D a pěti rychlostních bitů CSSSS. Dříve byl bit C využíván na ovládání světel lokomotivy, ale dnes kvůli zvýšení počtu stupňů rychlostí, slouží pouze jako nejnižší rychlostní bit.

   Čtyři nejnižší kombinace rychlostních bitů jsou vyhrazena pro čtyři typy zastavení, zatímco zbytek možných kombinací těchto pěti bitů reprezentuje 28 stupňů rychlosti. Nejběžnějším typem zastavení je kombinace pěti bitů s hodnotou nula, která vlak plynule zabrzdí podle nastavené rychlostní rampy. Brzdící rampa slouží pouze k realističtějšímu chování lokomotivy a celkově tak přispívá k lepšímu vizuálnímu efektu kolejiště. Nastane-li však situace, při které je potřeba lokomotivy zastavit co možná nejrychleji, je vhodnější použít tzv. Emergency stop neboli E-stop. Tento typ při zastavení ignoruje rychlostní rampu a lokomotivku okamžitě odpojí od napájení. Vyjma ignorování rychlostní rampy, lze také ignorovat směrový bit "D", takové zastavení je v tabulce označeno "I". To vede k zastavení při kterém se nezmění ostatní

směrové funkce lokomotivy. V našem případě tento jev lze sledovat pouze na světlech lokomotiv. Tento typ zastavení může být použit např. pro nouzové zastavení všech lokomotiv s adresou broadcast, při kterém ale světla zůstanou rozsvícená s respektováním směru z posledního paketu. Všechny typy zastavení přehledně znázorněny v tab. 2.6.

praembule	0	0AAA AAAA	0	01DC SSSS	EEEE EEEE	1

Tab. 2.4: Základní paket pro jízdu

praembule	0	11AA AAAA	0	AAAA AAAA	0	01DC SSSS	EEEE EEEE	1

Tab. 2.5: Základní paket pro jízdu s čtrnáct bitů dlouhou adresou

- A. . . . . adresa lokomotivy
- D. . . . . směr jízdy
- C/S. . . .rychlost jízdy
- E. . . . . kontrolní bajt

CSSSS	Typ zastavení
00000	stop
10000	stop I
00001	E-stop
10001	E-stop I

Tab. 2.6: Tabulka kombinací rychlostních bitů pro zastavení lokomotivy

2. **Rozšířený paket pro jízdu**

   Funkční rozdíl oproti základnímu paketu je pouze v počtu rychlostních kombinací, kterých je 128 včetně všech druhů zastavení. Viz tab. 2.7. Pro tyto účely j sou použity dva datové bajty, přičemž jeden obsahuje konstantu 00111111 a druhý kombinaci směrového a rychlostních bitů. Stejně tak, jako u základního paketu pro jízdu, lze odesílat paket na dekodéry s rozšířenou adresou.

   - A. . . . . adresa lokomotivy
   - D. . . . . směr jízdy
   - S. . . . . rychlost jízdy
   - E. . . . . kontrolní bajt

praembule	0	0AAA AAAA	0	0011 1111	0	DSSS SSSS	EEEE EEEE	1

Tab. 2.7: Rozšířený paket pro jízdu

3. **Pakety pro ovládání funkcí lokomotivy**

   Lokomotiva může disponovat až dvaceti osmi funkcemi, které jsou rozděleny do pěti typů paketů. V tab. 2.8 jsou pro přehlednost uvedeny pouze datové bajty těchto paketů, nikoliv však preambule, adresa a kontrolní součet, které mají shodný tvar jako u paketů pro jízdu. Pro lokomotivy s rozšířenou adresou je paket také obdobný jako u paketu pro jízdu, pochopitelně s rozdílem obsahu datových bajtů.

Skunipa funkcí	1. datový bajt	2. datový bajt
f_0 až f_4	100L FFFF	—
f_5 až f_8	1010 FFFF	—
f_9 až f_{12}	1011 FFFF	—
f_{13} až f_{20}	1101 1110	FFFF FFFF
f_{21} až f_{28}	1101 1111	FFFF FFFF

Tab. 2.8: Přehled datových bajtů všech typů paketů pro ovládání funkcí lokomotiv

   - L. . . . . světla
   - F. . . . . ostatní funkce

2.3.2.3 Pakety pro ovládání příslušenství

Řízením modelového kolejiště se rozumí, nejen ovládání samotných lokomotiv, eventuálně vlaků, ale také všelijakého příslušenství, od signalizace návěstidel, ovládání závor až k osvětlení okolních domů, nebo pouličních lamp. Pro všechny tyto eventuální zařízení je zde paket pro ovládání příslušenství.

1. **Základní paket pro příslušenství**

praembule	0	10AA AAAA	0	1AAA CDDD	EEEE EEEE	1

Tab. 2.9: Základní paket pro příslušenství

   - A. . . . . adresa zařízení
   - C. . . . . sepnuto/rozepnuto
   - D. . . . . číslo spínače
   - E. . . . . kontrolní součet

praembule	0	10AA AAAA	0	1AAA 0AA1	000X XXXX	0	EEEE EEEE	1

Tab. 2.10: Rozšířený paket pro příslušenství

2. **Rozšířený paket pro příslušenství**

   - A. . . . . adresa zařízení
   - X. . . . . data
   - E. . . . . kontrolní součet

2.3.3 Elektrická specifikace

Pro správnou činnost zařízení je potřeba dodržovat určité normy, které upřesňuje Standart S-9.1. Podle této normy musí být systém funkční již při výstupním napájení DCC generátoru o velikosti 7 V, avšak doporučená hodnota je 14 až 16 V. Hodnota výstupního napětí DCC zesilovače je závislá na na měřítku modelového kolejiště. Pro námi používané měřítko TT je maximální hodnota 22 V, avšak vhodná hodnota je v rozmezí pouze 14 až 16 V. Zařízení jsou nad dimenzovány tak, aby byli schopné v tomto měřítku zvládnout napěťové špičky až o velikosti 27 V. Další nároky jsou kladeny na strmost hran, která by v oblasti od -4 do 4 V neměla na výstupu z DCC generátoru překročit hranici 2,5 V/μ, zatímco na vstupu dekodéru je hranice 2 V/μ. Další nároky které musí systém splňovat je časování logických hodnot, a to ne pouze pro dobu trvání jednotlivých hodnot viz 2.3.1, ale také na rozdíl mezi signálem v levé resp. pravé kolejnici. Na výstupu DCC generátoru nesmí být tento rozdíl větší než 3μ, avšak dekodér by měl být schopen správně vyhodnotit i signál s rozdílem 6μ a menším.

2.4 Formát zpráv pro řídící jednotky

V podkapitole 2.3.2 jsme si popsali formát DCC paketu, který DCC generátor odesílá nebo naopak přijímá z kolejnic. Nyní se zaměříme na formát zpráv probíhajícími mezi jednotkami připojenými ke kolejišti a řídícím PC. Pro účely komunikace řídícího PC s jednotkou DCC generátoru a DCC zesilovače byl použit formát zprávy viz tab. 2.11 kde jedna buňka odpovídá jednomu bajtu. Jak lze vidět formát zprávy se skládá, z jeden bajt dlouhé hlavičky, dvou bajtů nesoucí adresu jednotky, jednoho nebo několika bajtů dat a na konec kontrolním bajtem CRC.

Hlavička	Adresa jednotky	Adresa jednotky	Data	...	Data	CRC

Tab. 2.11: Formát zprávy pro řídící jednotky

Hlavička musí nabývat hodnot z rozsahu jedna až osm a obsahuje informaci o počtu datových bajtů zprávy. Z toho vyplývá, že počet datových bajtů také bude z tohoto

rozsahu. Díky informaci o délce datových bajtů dokážeme aplikovat následující postup. Nejdříve detekujeme hlavičku, poté z informace v ní odpočítáme délku zprávy a provedeme její kontrolní součet, vyjde-li nám stejná hodnota jaká je obsažena v následujícím bajtu kontrolního součtu, můžeme si být téměř jistí, že se jedná o naši zprávu.

Adresové bajty skrývají informace o typu a pořadovém čísle jednotky ke které je zpráva směřována. Dva bajty nám udávají kapacitu šestnáct bitů, ze kterých my ale využíváme pouze horních jedenáct. Horní čtyři bity obsahují informaci o typu jednotky viz tab. 2.12. Typy jednotek jsou dále rozděleny podle směru přenosu dat, tj. zda jsou odeslány z počítače do jednotek, či z jednotek do počítače. Spodních sedm bitů nese informaci o pořadovém čísle jednotky, tudíž sytém nepočítá s více než 128 jednotkami stejného typu.

Horní bity adresy	Typ jednotky
000	Emergency (z počítače do jednotky)
0001	DCC generátor (z počítače do jednotky)
0010	DCC generátor (z jednotky do počítače)
0011	DCC zesilovač 1 (z jednotky do počítače)
0100	DCC zesilovač 2 (z jednotky do počítače)
0101	DCC zesilovač 1 (z počítače do jednotky)
0110	DCC zesilovač 2 (z počítače do jednotky)
0111	Výhybky (z počítače do jednotky)
1000	Výhybky (z jednotky do počítače)
1001	Návěstidla (z počítače do jednotky)
1010	Točna (z počítače do jednotky)
1011	Točna (z jednotky do počítače)
1100	—
1101	—
1110	—
1111	—

Tab. 2.12: Přiřazení horních čtyř bitů adresových bajtů konkrétním typům jednotek s ohledem na směr šíření zprávy

Jak již bylo zmíněno, počet datových bajtů je definován v hlavičce a tudíž zpráva může obsahovat minimálně jeden a maximálně osm bajtů. V současné době jsou připojeny pouze dvě jednotky a těmi jsou DCC generátor a DCC zesilovač. Z toho důvodu zpráv, které systém zpracovává, je jen několik, ale do budoucna jich bude významně přibývat.

2.4.1 Zprávy probíhající mezi DCC generátorem a řídícím PC

2.4.1.1 Zprávy odeslané DDC generátorem do řídícího PC

Jediná zpráva periodicky odesílaná z DDC generátoru do rozhraní je tzv. watchdog viz tab. 2.13. Tato zpráva obsahuje výhradně jeden datový bajt s hodnotou 0x55 z čehož vyplývá hlavička s hodnotou 0x01. Adresa watchdog zprávy nese dva bajty s hodnotou 0x20, tedy odpovídá typu jednotky "DCC generátor (z jednotky do počítače)" viz tab. 2.12 a pořadovému číslu jednotky jedna. Zpráva je poté zakončena kontrolním součtem, který vychází 0x88. Zpráva nenese žádnou důležitou informaci uvnitř svých datových bajtů, ale slouží k hlídání funkčnosti zařízení. Proto za běžných podmínek řízení kolejiště tuto zprávu nepotřebujeme nijak zobrazovat uživateli. Pro ujištění funkčnosti potřebuje jednotka DCC generátoru pravidelný příjem některých zpráv, jinak vlaky automaticky zastaví. Interval sledující, zda byla přijata zpráva, je ve výchozím nastavení seřízen na 1,5 s.

0x01	0x20	0x20	0x55	0x88

Tab. 2.13: Watchdog zpráva periodicky odesílaná z DCC generátoru do řídícího PC

2.4.1.2 Zprávy odeslané řídícím PC do DCC generátoru

Cílem zpráv odeslaných do DCC generátoru je samotné řízení lokomotiv. Chceme-li ovládat směr pohybu, rychlost nebo snad další funkce lokomotivy jako např. ovládání světel, adresujeme zprávu právě jednotce DCC generátoru. Zpráva pro takové ovládání lokomotiv vždy začíná hlavičkou 0x08, tudíž zpráva obsahuje ten jistý počet datových bajtů. Následují adresové bajty kde, typ jednotky bude opět DCC zesilovač, ale tentokrát chceme adresu pro odesílání z počítače do jednotky, takže první adresový bajt má nyní hodnotu 0x10 a druhý znovu 0x20. Datové bajty zprávy odpovídají adresovým a datovým bajtům DCC paketu viz 2.3.2. To znamená, že pokud používáme pro řízení pouze základní pakety, potom první datový bajt vždy obsahuje adresu lokomotivy viz tab.2.3. A ve druhém bajtu jsou ukryty konkrétní příkazy, které má lokomotiva vykonávat, jako např. rozsvit světla, nebo rozjeď se určitou rychlostí vpřed. Výše jsem uváděl, že datová část zprávy je osm bajtů dlouhá a odpovídá adresovým a datovým bajtům DCC paketu. Ale základní DCC paket pro jízdu lokomotivy, nebo pro ovládání jejího příslušenství, obsahuje pouze jeden bajt adresový a druhý bajt datový, tedy zbytek datových bajtů naší zprávy musíme doplnit nulami. Na závěr nezakončíme zprávu kontrolním součtem, jak bychom předpokládali, ale pouze kombinací 0xFF. To z zůvodu použitého převodníku sériových dat na CAN data. Takto sestavená zpráva pro jízdu vlaku může vypadat např. takto viz tab. 2.14.

0x08	0x10	0x20	0x09	0x7F	0x00	0x00	0x00	0x00	0x00	0x00	0xFF

Tab. 2.14: Příklad zprávy odesílané řídícím PC do generátoru DCC (zde se jedná o příkaz lokomotivě Ragulin plnou rychlostí vpřed)

2.4.2 Zprávy probíhající mezi DCC zesilovačem a řídícím PC

Řídící jednotka kolejových úseků neboli DCC zesilovač má mimo funkce opakování DCC signálu ještě velice důležitou funkci a to měření odebíraného proudu na kolejnicích. Tímto způsobem dostáváme zpětnou vazbu o výskytu lokomotiv na jednotlivých úsecích kolejiště.

2.4.2.1 Zprávy odeslané DCC zesilovači do řídícího PC

Z důvodu měření proudu na kolejových úsecích, vychází první klíčová zpráva, kterou musí DCC zesilovač odesílat. Periodické zasílání naměřených proudů probíhá v pravidelném nastavitelném intervalu. Další zprávou odesílanou periodicky je informace o přetížení H-můstku, ovšem jen v případě, kdy k přetížení skutečně dojde. Restartujeme-li H-můstek, jednotka odešle potvrzovací zprávu, jako signalizaci, že H-můstek je opět v provozu. Posledním typem zpráv odesílané jednotkou jsou zprávy sloužící k potvrzení, že došlo k přijetí nastavovací zprávy viz 2.4.2.2. Nastavovat lze prodlevu odesílání změřených hodnot proudů, nebo vypínat či zapínat 15 V zdroj. Zpráva s obsahem změřených úseků obsahuje osm datových bajtů, takže hlavička má hodnotu 0x08. Každý z datových bajtů reprezentuje změřenou hodnotu na jednom z izolovaných úseků fyzicky připojených k jednotce. Zbytek výše zmíněných zpráv obsahuje pouze jeden datový bajt. Prozatím jediná připojená řídicí jednotka kolejových úseků má pořadové číslo tři a z tab. 2.12 vidíme, že typ jednotky "DCC zesilovač 1 (z jednotky do počítače)" odpovídá decimálně také číslu tři. Takže pro tuto jednotku má první adresový bajt hodnotu 0x30 a druhý bajt hodnotu 0x60. Podoba všech datových bajtů zobrazena přehledně v tab. 2.4.2.2. V tabulce si lze povšimnout, že datové bajty změřeného proudu mohou nabývat hodnot 0 až 255. Zde hodnoty nereprezentují přesnou hodnotu naměřeného proudu, ale pouze poměr v určitém rozsahu měření.

Význam zprávy	Obsah datových bajtů
Změřený proud na úsecích	8krát (0x00 až 0xFF)
Potvrzení změny prodlevy odesílání změřených proudů	0x00
15 V zdroj zapnut	0x01
15 V zdroj vypnut	0x02
H-můstek v provozu	0x03
Chyba H-můstku	0x04

Tab. 2.15: Přehled datových bajtů zpráv odesílaných DCC zesilovači do řídícího PC

2.4.2.2 Zprávy odeslané řídícím PC do DCC zesilovačů

Všechny zprávy odesílané uživatelem z řídícího PC do úsekové jednotky mají za úkol provést konfiguraci této jednotky. V rámci konfigurace lze provést nastavení prodlevy odesílání naměřených proudů, zapnout či vypnout napájecí zdroj napětí a v neposlední řadě, restartování H-můstku nebo mikrokontroléru. Pro zatím jedinou připojenou úsekovou jednotku s pořadovým číslem tři vychází první adresový bajt 0x50 a druhý 0x60. Datové bajty všech zpráv jsou uvedeny v tab. 2.16. Ve výchozím nastavení odesílá jednotka naměřené hodnoty odebíraného proudu každých 100 ms. Tato prodleva je nastavitelná viz první zpráva v tabulce níže. První datový bajt této zprávy nese vždy hodnotu 0x00 a druhý bajt představuje číslo, kterým je vynásoben základní interval 100 ms. Při nastavení druhého bajtu do hodnoty 0x00, jednotka přestane informace o naměřených proudech posílat.

Význam zprávy	První datový bajt	Druhý datový bajt
Změna prodlevy odesílání změřených hodnot proudů	0x00	(0x00 až 0xFF))
Vypnutí 15 V zdroje	0x01	—
Zapnutí 15 V zdroje	0x02	—
Restart H-můstku	0x03	—
Restart mikrokontroléru	0x04	—

Tab. 2.16: Přehled datových bajtů zpráv odesílaných řídícím PC do DDC zesilovačů

3

Software pro řízení modelového kolejiště Train TT 1.0

Veškeré zde popisované okna aplikace si lze prohlédnout v sekci přílohy na konci této práce.

3.1 Základní informace pro obsluhu aplikace

Bezprostředně po spuštění aplikace, je uživatelovi nabídnuto hlavní menu, obsahující čtyři tlačítka. Viz obr. A.1. První tlačítko *Manual control* otevře okno pro ovládání jednotlivých vlaků. Druhé tlačítko *Timetable control*, nejprve vyzve uživatele k nahrání souboru typu csv (comma-separated values) ,který obsahuje jízdní řád ve vhodném formátu a poté zobrazí okno reprezentující tabuli jízdních řádů v nádraží. Třetí tlačítko *Debug* slouží pouze programátorovi k odlaďování všech funkcí aplikace, protože viditelně zobrazuje okno, obsahující detailní informace o dění na kolejišti. Toto okno je u zbylých režimů této aplikace neviditelné, pouze běží na pozadí a vykonává svou funkci. Poslední tlačítko *Exit* slouží k ukončení aplikace.

3.1.1 Manuální řízení

Okno pro manuální řízení obsahuje třináctkrát opakující se motiv řádku s obrázkem lokomotivy, jejím názvem, tlačítkem pro změnu směru jízdy, posuvnou lištou pro změnu rychlosti a nakonec spouštěcím, respektive zastavovacím tlačítkem. K dispozici máme maximálně třináct lokomotiv, proto stejný počet těchto ovládacích slotů. Kliknutím na obrázek lokomotivy se obrázek přiblíží. Za zdůraznění stojí, že stav kdy lokomotiva jede vpřed, je ve chvíli, kdy na tlačítku udávajícím směr je nápis *Backward*. Stejně tak, lokomotiva stojí, je-li na tlačítku udávající pohyb nápis *Start* a naopak jede, je-li na tlačítku nápis *Stop*. V dolní části okna se nachází tlačítko *Pause* pro pozastavení všech lokomotiv a tlačítko *Close* pro odchod zpět do hlavního menu.

3.1.2 Řízení jízdním řádem

Okno jízdního řádu, které se zobrazí po úspěšném nahrání jízdního řádu viz. 3.1 je rozděleno na dvě části, zatímco v levé části se nacházejí analogové hodiny, v části pravé je zobrazena tabule nejbližších odjezdů. První tři sloty v jízdním řádu, označené světle červenou barvou, patří posledním spojům, které se již vykonaly. Zbytek slotů je vyhrazen pro spoje, které se teprve vykonají. Jízdním řádem lze rolovat a zjistit tak odjezd ve kterýkoliv čas právě probíhajícího dne. Pokud uživatel jízdní řád nevypne, provede se znovu načtení jízdního řádu z předešlého dne. Stejně jako okno pro manuální řízení, tak i okno jízdního řádu disponuje tlačítkem pro pozastavení všech lokomotiv a tlačítko pro odchod do hlavního menu.

3.1.2.1 Formát jízdního řádu

Soubor jízdního řádu je typu csv (comma-separated values, čárkami oddělené hodnoty), neboli jednoduchý formát určený pro výměnu tabulkových dat. Vzhledem k omezeným možnostem kolejiště a také této aplikace je řízení jízdním řádem pouze přesouvání vlaku z jedné stanice do druhé stále dokola v nastavených časech. Obsah jízdního řádu se skládá ze dvou typů záznamů, jeden záznam odpovídá jednomu řádku v tabulce. První typ slouží k inicializaci vlaků a stanic mezi kterými bude vlak cestovat, zatímco druhý typ záznamu obsahuje informace o počáteční stanici, cílové stanici a času odjezdu pro konkrétní spoj. To znamená, že první typ záznamu uživatel při používání aplikace neuvidí, tyto informace jsou pouze zpracovávány aplikací. Naproti tomu druhý typ záznamu je v přesně takovém formátu, jak ho uživatel uvidí v aplikaci.

První typ záznamu má formát viz tab.3.1. V první buňce se nachází inicializátor, v podobě tří hvězdiček, který upozorňuje na fakt, že řádek je prvním typem záznamu obsahující inicializaci vlaku. Následující buňka nese informaci o lokomotivě která bude danou trasu vykonávat. Tento název musí odpovídat jednomu z názvů lokomotiv uvedené ve třídě Engine.cs v seznamu listOfEngines. Další dvě buňky obsahují čísla reprezentující úseky mezi kterými bude vlak jezdit. Dále je uveden tzv. pracovní název vlaku, jedná se o název pod kterým budou spoje v aplikaci uvedeny např. R 764, Os 8845 atd. Pracovní názvy stanic mají obdobný význam jako pracovní název vlaku, tedy názvy stanic které uvidí uživatel v aplikaci. V sedmé buňce je informace o rychlosti jízdy, vyjádřená v rozsahu 0 až 1, reprezentující násobitel maximální rychlosti lokomotivy. Nasledují dvě buňky obsahují směr jízdy a mohou nabývat pouze hodnot "Forward"(Dopředu) nebo "Backward"(Dozadu). Je nutné mít možnost nastavit zvlášť směr jízdy z první stanice do druhé a zvlášť směr jízdy z druhé stanice zpět do první. To z důvodu, že se na kolejišti vyskytují stanice, kde není možné vlak otočit ani stanicí projet a tak je potřeba ze stanice vycouvat. Poslední dvě buňky patří informacím nazvaným časy čekání které jsou klíčové při příjezdu vlaku do stanice. Čas čekání je totiž doba od chvíle, kdy cílový úsek zaregistruje přítomnost vlaku, neboli změní svou obsazenost až do momentu kdy je

odeslán zastavovací paket pro tento vlak. Musíme si uvědomit že nástupiště je kratší než úsek který ho reprezentuje a fakt, že je na úseku detekován vlak ještě neznamená že dojel i se všemi vagóny až do stanice. Právě z tohoto důvodu jsou zde nastavitelné doby čekání, díky kterým si může uživatel v jízdním řádu nastavit takový čas čekání, aby vlak dojel do stanice celý a vizuální požitek byl co možná největší.

***	L	ČÚ1	ČÚ2	PNV	PNS1	PNS2	R	S1	S2	ČČ1	ČČ2

Tab. 3.1: Formát záznamu v jízdním řádu pro inicializaci vlaků a stanic.

- L. . . . . lokomotiva
- ČÚ1/2. . . . . číslo prvního/druhého úseku
- PNV. . . . . pracovní název vlaku
- PNS1/2. . . . . pracovní název první/druhé stanice
- R. . . . . rychlost
- S1/2. . . . . směr jízdy
- ČČ1/2. . . . . čas čekání

Druhý typ záznamu je oproti prvnímu kratší a jednoduší viz tab. reftab˙format˙jizdni˙rad˙zaznam. Je složen pouze ze čtyř buněk, kde první buňka obsahuje pracovní název vlaku, druhá počáteční stanici, třetí cílovou stanici a poslední buňka čas odjezdu. Název vlaku slouží, nejen ke zobrazení jízdního řádu v plikaci, ale také ke spárování s příslušnými daty získanými z řádků obsahujících inicializace.

PNV	PS	CS	ČO

Tab. 3.2: Formát záznamu v jízdním řádu obsahující konkrétní informace o spoji

- PNV. . . . . pracovní název vlaku
- PS. . . . . počáteční stanice
- CS. . . . . cílová stanice
- ČO. . . . . čas odjezdu

3.1.3 Odlaďovací řízení

Jak již bylo zmíněno, tento režim řízení není vhodný pro běžného uživatele, ale pouze pro programátora nebo správce kolejiště. Po zobrazení okna je většina ovládacích prvků nepřístupna, je to z důvodu, že na rozdíl od manuálního řízení, nebo řízení jízdním řádem, není programem automaticky vybrán COM port. Ten lze vybrat v seznamu připojených COM porů uvnitř rozbalovací lišty v levé horní části okna. Tento fakt dovoluje programátorovi propojit aplikaci, přes virtuální COM port s jinou aplikací a tím celý systém efektivně odladovat. Po vybrání příslušného portu a kliknutí na tlačítko *Open Port* dojde k otevření komunikačního kanálu a zpřístupnění zbytku ovládacích tlačítek. Pro ukončení komunikace je zde tlačítko *Close Port*, které také opět uvede zbylé tlačítka do stavu znepřístupnění. V levé části ovládacího okna jsou umístěny tři textové pole. Nejmenší z nich leží těsně pod zmíněnými tlačítky a zobrazuje informaci o stavu připojení. Vpravo od tlačítek se nachází pole vypisující všechny data přijaté sériovou komunikací, není-li přijímání zpráv pozastaveno tlačítkem *Stop* níže. Naproti tomu, pod textovým polem pro odesílání dat se nachází tlačítka dvě. Tlačítko *Send* které odešle obsah textového pole pouze jednou a tlačítko *Auto Send* které zahájí opakující se cyklus odesílání zadaných dat. Levou část okna uzavírá vyhrazený prostor pro jízdní řád, spustitelný tlačítkem *Start Timetable*. V prostřední části okna nalezneme prostor pro ovládací sloty jednotlivých lokomotiv. Sloty lze přidávat, nebo naopak odebírat tlačítky *Add Engine* resp. *Remove Engine*. Každný slot obsahuje rozbalovací lištu pro výběr lokomotivy, tlačítko pro výběr směru, posuvnou lištu pro výběr rychlosti a nakonec tlačítko *Start* resp. *Stop*, záleží na tom je-li vlak právě v pohybu, či nikoliv. Nelze vybrat k ovládání ten jistý vlak ve dvou různých slotech najednou, a to z důvodu případné kolize zpráv. Poslední komponenta v okně je sloupcový graf. K jeho vykreslení dochází až s přijatými zprávami o obsazenosti kolejových úseků.

3.1.4 Nedokonalosti verze 1.0

První verze aplikace pro řízení modelového kolejiště vznikala během seznamování se s mechanizmy kolejiště, teorií DCC paketů i samotným jazykem C#, tudíž se aplikace může zdát místy chaotická, nebo nepřehledná a především komplikovaně rozšiřitelná o další funkce. Jako celek je funkční pro současnou podobu kolejiště, avšak ne zcela bezchybná a pro její bezpečné používání je potřeba tyto nedokonalosti znát a brát je na vědomí. Přirozeně lze tyto nedokonalosti odstranit, to však nebylo prioritou, vzhledem ke vzniku nové verze, která tyto nedokonalosti neobsahuje, navíc celý systém vylepšuje a dovoluje jeho snadné rozšíření.

První nedostatkem je pevné nastavení sériového portu v režimu manuálního řízení a řízení jízdním řádem, kde je automaticky vybrán nejvyšší nalezený port v PC. Jinak tomu je v případě odlaďovacího řízení, kde lze vybrat port sériové komunikace podle potřeby. Dalším nedostatkem je špatné vyhodnocení situace, kde vlak stojí ve stanici v moment kdy, je nahrán jízdní řád obsahující instrukci pro příjezd vlaku do této stanice právě v

tento jistý moment. Při této situaci se totiž vlak rozjede na dobu nazývanou čas čekání viz 3.1.2.1.

Těchto nedokonalostí, větších či menších, lze najít více. Však tou největší, která si vyžádala vznik druhé verze aplikace je obtížná rozšířitelnost funkcí aplikace. V současné době probíhá vývoj dalších komponent pro modelové kolejiště, a to konkrétně komponenta pro řízení výhybek, řízení točny a další řídící jednotka kolejových úseků. Z toho důvodu je nutné aby byla aplikace pružná a snadno aplikovatelná i pro nově vytvořené komponenty. Tuto podmínku první verze aplikace nesplňuje, a proto není vhodná pro pozdější použití. Tento klíčový nedostatek řeší vznik aplikace verze 2.0.

4

Software pro řízení modelového kolejiště Train TT 2.0

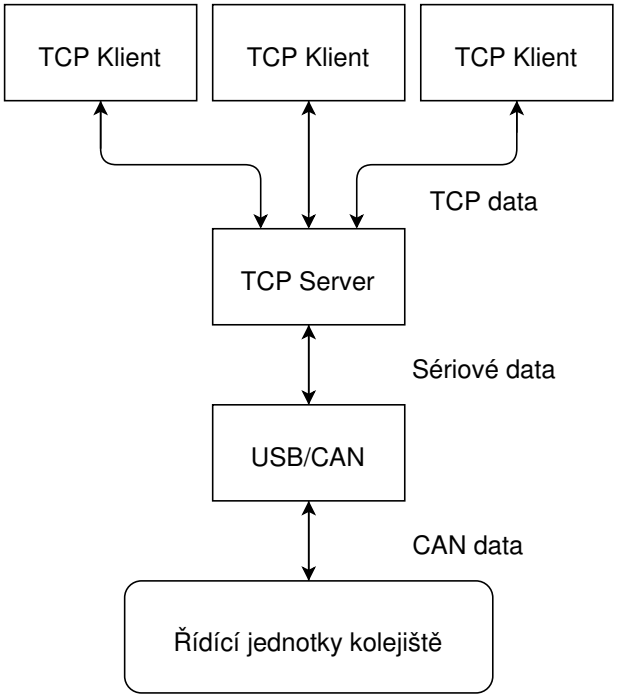
Z důvodů blíže zmíněných v kapitole 3.1.4 byla vytvořena druhá verze aplikace pro řízení modelového kolejiště. Tato verze je založena na TCP (Transmission Control Protocol) komunikaci a to přináší výhody v podobě zcela nového pojetí řízení. Na rozdíl od verze 1.0, není nová verze pouze jedním celistvým spustitelným programem, ale skládá se hned z několika samostatně fungujících programů. Propojení jednotlivých programů je provedeno asynchronní komunikací, přes již zmíněný protokol TCP.

Nejdůležitějším programem z nové aplikace je program s názvem *TCP Server train TT* zprostředkovávající TCP server pro ostatní klienty a také zajišťující komunikaci s jednotkami kolejiště přes sériový port. Server přijímá zprávy od ostatních klientů v podobě přesně definovaného textového řádku a vytváří z nich zprávy pro jednotlivé řídící jednotky. Tímto způsobem můžeme k odesílání řádků použít jakýkoliv TCP terminál v síti, stačí jen znát IP adresu řídícího PC a číslo portu se kterým je server svázán. Textové řádky dávají smysl a je z nich jasně čitelná instrukce i pro osobu, která nezná zásady tvoření DDC paketů nebo zpráv pro řídící jednotky. V tom spočívá klíčová přednost této verze, nejen že můžeme ovládat kolejiště z kteréhokoliv místě v síti a ani k tomu nepotřebujeme nikterak speciální program, ale dokonce nám k ovládání stačí pouze znát několik jednoduchých ovládacích zpráv viz tab. 4.1. Další výhodou takového řízení je, že není potřeba, aby server byl vizuální aplikací, proto je vytvořen jen ve formě konzolové aplikace. To přináší výhody ve formě velice rychlé reakce na zastavení, je-li zastavení vyvoláno přímo serverem. Pokud je zastavení vyvoláno některým z klientů, musíme do reakční doby započítat také čas na odeslání příkazu klientem a následné přijmutí serverem. Velká přednost verze 2.0 spočívá také v možnosti jednoduchého rozšíření aplikace o snadné dodělání klienta pro různé potřeby. Pokud by bylo zapotřebí vytvořit nového klienta, který by byl schopen např. zobrazovat mapu kolejiště a výskyt lokomotiv na ní, nebo řídit kolejiště pomocí hlasového ovládání z mobilu, není to v této verzi žádná překážka.

Jak bylo řečeno, nová aplikace je složena z více jednotlivých programů. Zatím jsme si představili jenom teď nejzákladnější a to server, avšak nyní se zaměříme na zbytek

programů, kterými jsou dva klienti a jeden řízený spouštěč. Klient nazvaný *Visual Debug Control Train TT* umožňuje uživateli sledovat obsazenost všech úseků, provádět konfiguraci jednotek DCC zesilovače a sledovat jejich odezvu, ale v první řadě řídit lokomotivy na úrovni příkazů, která lokomotiva jakou rychlostí a směrem má jet. Další klient se jmenuje *Timetable Control Train TT* a zprostředkovává řízení pomocí nahraného jízdního řádu.

Ústřední částí nejen pro funkčnost serveru, ale také všech klientů je knihovna nazvaná *Train TT Library* a obsahující rovnou několik tříd stěžejních pro chod aplikace.

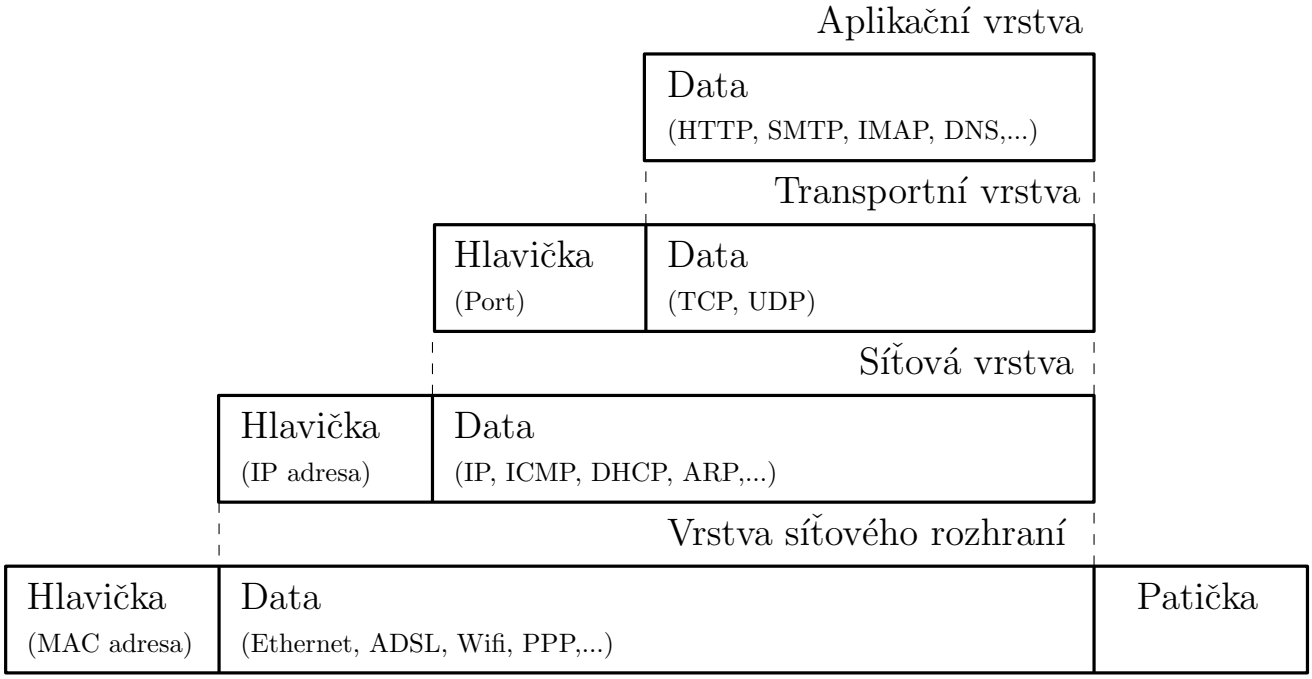


Obr. 4.1: Grafické znázornění jednotlivých úrovní

4.1 Síťový model TCP obecně

Celosvětový fenomén internet si vyžádal příchod celé řady pravidel a přesně dané syntaxe pro síťovou komunikaci. Tyto pravidla pevně definuje model TCP/IP (Transmission Control Protocol/Internet Protocol) obsahující množství protokolů rozdělených do čtyř vrstev. Nejnižší vrstva se nazývá Vrstva síťového rozhraní a umožňuje fyzický přístup k přenosovému médiu. Nejznámější příklad této vrstvy je Ethernet, který se dokázal prosadit ve více než 80 % všech síťových instalací. Nad vrstvou síťového rozhraní stojí Síťová vrstva zajišťující zejména síťovou adresaci, směřování a předávání datagramů. Porotokoly této vrsty jsou např. IP(Internet Protocol), ARP(Address Resolution Protocol), nebo také IGMP(Internet Group Management Protokol). Další vrstvou je Transportní vrstva, obsahující spojový protokol TCP, či nespojový protokol UDP. Poslední a zároveň nejvyšší vrstva je Aplikační vrstva, zastupující procesy využívající přenosu dat po síti ke konkrétním službám. Nejrozšířenějšími příklady jsou protokoly HTTP(Hypertext Transfer Pro-

tocol), FTP(File Transfer Protocol), SMTP(Simple Mail Transfer Protocol) nebo např. DNS(Domain Name System).



Obr. 4.2: Zapouzdření síťového modelu TCP/IP

4.2 Základní informace pro obsluhu aplikace

Pro správnou funkčnost systému je nutné mít vždy spuštěn program *TCP Server train TT*. Tento program zprostředkovává server pro TCP komunikaci a zároveň zajišťuje komunikaci s řídícími jednotkami kolejiště. Pokud je program spuštěn správně, před uživatelem se objeví konzole, která nejdříve vypíše síťový port, na kterém je server připraven naslouchat a následně také sériový port přes jenž je systém kolejiště připojen k řídícímu PC. Pro přehlednost je vypisování informací v konzoli rozděleno barvou fontu. Modrá barva patří přijatým datům od jednotlivých klientů, zelená barva značí odeslané data ke klientům a v neposlední řadě jsou zde zprávy žluté, které informují uživatele o nastavení aplikace a zprávy červené které upozorňují na chyby v systému. Je-li připojení ke kolejišti provedeno bez komplikací, v konzoli se okamžitě začínají objevovat zprávy s obsazeností jednotlivých úseků. Nestane-li se tak, jeden z možných důvodů může být špatné nastavení sériového portu. Toto nastavení lze provést v konfiguračním souboru *Settings.settings* uvnitř adresáře *Properties* pomocí běžného textového editoru. Po spuštění serveru nám nic nebrání zapnout klienta či rovnou více klientů a začít s řízením kolejiště. Veškeré zde popisované programy aplikace si lze prohlédnout v sekci přílohy na konci této práce.

4.2.1 Řízení pomocí programu *Visual Debug Control Train TT*

První variantou je tzv. odlaďovací řízení. To nabízí několik funkcí, kterým by se měl nepoučený uživatel aplikace raději vyhnout, ale také základní řízení lokomotiv a jednoduché zobrazení obsazenosti úseků. Po spuštění programu je uživateli nabídnuto ovládací okno obsahující několik prvků. Prostor na levé straně je určen pro zobrazení vybraných zpráv příchozích z řídících jednotek. Jsou to zejména zprávy oznamující chyby systému. Horní část okna je věnována čtyřem hlavním ovládacím tlačítkům. Tlačítko *Add locomotive* přidá do prostoru pod tlačítky ovládací slot pro jednu lokomotivu, naopak tlačítko *Remove locomotive* odstraní poslední otevřený ovládací slot. Třetí tlačítko *Central stop* odešle všem lokomotivám příkaz k okamžitému zastavení a vymaže nastavený obsah v ovládacích slotech. Poslední tlačítko *Close* slouží pouze k ukončení programu, ale než se tak stane, klient odešle všem lokomotivám příkaz k zastavení, stejně tak jako u předešlého tlačítka. Důvod je zřejmý, zavřeli bychom tento program bez zastavení lokomotiv, lokomotivy by jezdili tak dlouho, dokud by byl zapnutý program serveru. Ovládací sloty jsou velice intuitivní. Nabízejí vybrání lokomotivy pomocí rozbalovací lišty, zapnutí či vypnutí světel, vybrání rychlosti jízdy rolovací lištou a nastavení směru zaškrtávacím políčkem. Slot zakončuje tlačítko *Start*, které zapříčiní odeslání předvoleného nastavení do systému. Pokud klient přijímá informace o obsazenosti úseků, zobrazí se v pravé části okna signalizační diody s názvy jednotlivých úseků, naznačující obsazenost příslušných úseků. Poslední součástí okna je spodní pasáž zahrnující také čtyři tlačítka, jako horní prostor okna, ale k nim jsou přidány dvě lišty pro nastavení číselné hodnoty. Celý tento sektor je určen pro konfiguraci úsekových jednotek. První lišta číselných hodnot označená nápisem *Unit* určuje číslo jednotky, které konfiguraci provádíme. Následují zmíněná tlačítka pro zapnutí či vypnutí 15 V zdroje, restartování H-můstku a restartování mikroprocesoru. Lišta číselných hodnot na konci, popsaná textem *Delay of sending currents*, dokáže nastavit prodlevu odeslání změřených proudů z úseků. Hodnota reprezentuje číslo, jímž je přenásobena základní nastavená prodleva 100 ms. Stane-li se že klient se pokouší odeslat zprávu, ale server není v provozu, vyskočí chybová hláška s možnostmi *Zrušit* nebo *Opakovat*.

4.2.2 Řízení pomocí programu *Timetable Control Train TT*

Také řízení jízdním řádem prodělalo řadu změn a vylepšení od verze 1.0. Po otvření programu uživatel vidí prázdný prostor určený pro zobrazení jízdního řádu a nad ním sedm ovládacích tlačítek.

První je zde tlačítko *Load* pro načtení jízdího řádu. Po jeho stisknutí je uživateli nabídnuto okno pro výběr souboru jízdního řádu ve formátu csv (comma-separated values) viz 4.2.2.2. Načtením jízdního řádu se stává tlačítko *Load* znepřístupněno, zatímco tlačítka *Pause* a *Change train data* jsou nyní k dispozici. Druhé tlačítko *Pause* způsobí okamžité zastavení všech lokomotiv a zároveň ignorování dat jízdního řádu do doby, dokud tlačítko není opět zmáčknuto. Funkce následujícího tlačítka *Change train data* je detailně popsaná

níže v kapitole 4.2.2.1. Tlačítko *Clock* zapne, resp. vypne vizualizaci analogových hodin. Další tlačítko s názvem *Full screen* zapne režim, kde okno s jízdním řádem je roztaženo přes celou obrazovkou a font je zvětšen tak, aby data z obrazovky byli čitelné i z větší vzdálenosti. Při tomto režimu je znepřístupněno tlačítko *Clock*. Předposlední tlačítko nese název *Central stop*, to ukončí okamžitý pohyb lokomotiv na kolejišti a zároveň úplně vypne jízdní řád, tak aby mohl být načtený jiný. Poslední tlačítko slouží k ukončení celého programu.

4.2.2.1 Funkce tlačítka *Change train data*

Oproti první verzi aplikace je zde možnost změnit data jízdního řádu přímo v aplikaci. Nejedná se o změnu časových záznamů v jízdním řádu, ale pouze o data pro inicializaci vlaků a stanic v hlavičce tohoto řádu. Díky tomu jsme schopni změnit lokomotivu, která vykonává dané spoje. Dále název lokomotivy, pod kterým je spoj v jízdním řádu uveden a také zastávky zastávek mezi kterými vlak jezdí. V neposlední řadě lze nastavit rychlost vlaku a také tzv. čekací neboli dojezdové časy. Jak bylo již zmíněno v kapitole 3.1.2.1, čas dojezdu je doba, po kterou je vlak stále v pohybu, potom co daný úsek zaregistruje změnu obsazenosti, tak aby vlak dojel až na požadované místo na úseku. K nastavení výše uvedených dat slouží okno vyvolané kliknutím na tlačítko *Change train data*.

Okno je složeno z několika řádků s nastavovacími prvky, kde každý řádek odpovídá jednomu vlaku který plní spoje vypsané ve zbytku jízdního řádu. Řádek obsahuje tři rozbalovací lišty pro výběr zastávek a lokomotiv, jedno textové pole s pracovním názvem vlaku a tři lišty číselných hodnot. Seznamy lokomotiv a zastávek jsou také načítány z konfiguračních souborech, blíže popsaných v kapitole 4.3. Pracovní název vlaku může být v podstatě jakýkoliv, jediné omezení je, že se nesmí pracovní názvy opakovat u dvou různých spojů. Dojde-li ke snaze zadat ten jistý název dvakrát, program ho automaticky změní. Rychlost vlaku je zde zadávána v procentech vtažených k maximální rychlosti daného vlaku, tudíž nastavení 100% rychlosti u dvou vlaků neznamená, že vlaky jezdí shodnou rychlostí. Dojezdové časy se nastavují v milisekundách a můžou se lišit pro jednotlivé stanice v řádku. Okno je dole zakončeno tlačítkem *OK* pro potvrzení zadaných změn a tlačítkem *Cancel* pro jejich zrušení. Vedle tlačítka pro zrušení změn je zde ještě zaškrtávací políčko s textem *Change data also in file*, které nejen změní zadaná data pro okamžitý provoz jízdního řádu, ale také změní zdrojový soubor, tak že při budoucím načtení tohoto řádu budou námi nastavené data zachována.

4.2.2.2 Formát jízdního řádu

I jízdní řád prošel od první verze aplikace jistými úpravami, však základní myšlenka je stále stejná viz /refjizdi˙rad˙ver1. Proto zde jen krátce k provedeným změnám. Řádky se samotnými časovými údaji spojů nejsou změněny na rozdíl od řádků nesoucí inicializaci vlaků a stanic. Inicializátor pro tyto řádky zůstává stejný a to v podobě tří hvězdiček.

Následuje název lokomotivy, který opět musí odpovídat obsahu konfiguračního souboru viz 4.3. V dalších dvou buňkách nejsou uvedeny žádné číslice označující úsek, jak tomu bylo v první verzi. Místo toho třetí buňka nese pracovní název vlaku a je následována dvěma buňkami s informacemi stanic mezi kterými jsou spoje vykonávány. Následující buňka definující rychlost neprošla žádnou změnou. To již neplatí pro další dvě buňky nesoucí směr. Tak jako je tomu v celé druhé verzi, i zde jsou pro použity pro určení směru klíčové slova *ahead* neboli dopředu a *reverse* neboli zpátečka. Nakonec jsou zde dvě buňky dojezdových časů, také shodné s první verzí jízdního řádu.

4.3 Textové zprávy TCP komunikace

Pro uživatele obsluhujícího aplikaci Train TT 2.0 není znalost těchto několika typů zpráv nutná, nicméně pro ovládání kolejiště z jiného TCP terminálu, je tato znalost klíčová. Textové zprávy se snaží být co možná nejjednodušší a čitelné na první pohled. Na úplném začátku zprávy nalezneme název typu, který zpráva přenáší a za ním jeho obsah viz tab. 4.1. Ve výčtu obsahu pod tabulkou je vždy uveden smysl obsahu a v závorce hodnoty kterých může daný obsah nabývat. Důležité je, že mezi typem zprávy a jejím obsahem se vždy nachází dvojtečka a mezi jednotlivými proměnnými v obsahu zase čárka. Každá zpráva musí být ukončena znakem pro odřádkování, který signalizuje konec zprávy. Dále žádná zpráva nikdy neobsahuje mezeru, ty jsou zde nahrazeny podtržítkem. Prvních pět zpráv v tabulce je primárně určeno pro směr datového toku od klientů k serveru, zatímco následující dvě zprávy odesílá naopak server všem klientům. Zpráva typu *unknow* není serverem nijak zpracovávaná, ale pouze vypsaná na konzoli a slouží tak hlavně pro informování uživatele, který zprávu odeslal. Upozorňuje uživatele, že při rozpoznávání zprávy došlo k chybě z důvodu např. špatně napsaného názvu lokomotivy, nebo třeba špatného rozsahu zadané rychlosti a uživatel může na tuto informaci reagovat opravením své zprávy a znovu odesláním. Ve výčtu pod tabulkou jsou detailněji vysvětleny významy proměnných v obsahu zprávy a v závorce za nimi jsou uvedeny hodnoty, kterých tato daná proměnná může nabývat. Je-li v závorce uvedeno, že musí odpovídat jedné položce ze seznamu, jsou tím myšleny dokumenty typu xml, které slouží jako konfigurační soubory pro aplikaci. Tyto soubory můžeme nalézt v adresáři *TrainTTLibrary* a lze je upravit v běžném textovém editoru.

- NAZEV..... název lokomotivy(musí odpovídat jednomu z seznamu lokomotiv)
- RYCHLOST..... číslo udávající rychlost (0 až 31)
- SMĚR..... směr jízdy (ahead/reverse)
- SVĚTLA..... rozsvícení nebo zhasnutí světel (on/off)
- CÍL..... cílový úsek pro dojetí (musí odpovídat jednomu z seznamu úseků)

Typ	Obsah
train_move	NAZEV,RYCHLOST,SMĚR
train_function	NAZEV,SVĚTLA
train_move_to_place	NAZEV,RYCHLOST,SMĚR,CÍL,ČAS PŘEJEZDU
unit_instruction	INSTRUKCE
unknow	—
unit_info	INFORMACE
occupancy_section	8krát(ÚSEK=HODNOTA,)

Tab. 4.1: Přehled všech typů textových zpráv probíhajících mezi serverem a klienty

- ČAS PŘEJEZDU..... číslo udávající dobu [ms] jak dlouho má lokomotiva pokračovat v jízdě dorazí-li do cílového úseku
- INSTRUKCE ..... název instrukce pro úsekovou jednotku (zapnuti_zdroje, vypnuti_zdroje, restart_H_mustku, restart_mikroprocesoru, prodleva_odesilani_zmerenych_proudu)
- INFORMACE..... informace od změně nastavení od úsekové jednotky (nabývá stejných hodnot jako INSTRUKCE)
- ÚSEK..... název úseku(musí odpovídat jednomu z seznamu úseků)
- HODNOTA..... hodnota naměřená na úseku (0 až 255)

4.4 Podrobný popis vnitřní struktury aplikace

Jak již bylo zmíněno, druhá verze aplikace je složena z několika programů, které dohromady tvoří celek pro různé způsoby ovládání kolejiště. Těmito programy jsou *TCP Server Train TT*, *Visual Debug Control Train TT* a *Timetable Control Train TT*, kde každý z programů čerpá funkcionalitu z knihovny *TrainTTLibrary*. Všechny tyto části jsou podrobně popsány v následujících podkapitolách.

Do aplikace lze také zahrnout program s názvem *Eink32˙Executor*, který slouží jako spouštěč předešlých programů. Přesněji řečeno se jedná o konzolový program, který nejprve spustí program *TCP Server Train TT* a následně *Timetable Control Train TT*. Následně hlídá běh programů a při jejich nenadálém ukončení je spustí znovu. Při spuštění programu, *Timetable Control Train TT* přes tento spouštěč, je také předáván parametr s cestou k jízdnímu řádu, uvedenému v konfiguračním souboru *exec˙config*. Předání tohoto parametru program pro řízení jízdním řádem zaregistruje a neprodleně spouští vybraný jízdní řád. Z toho plyne, že pouhým spuštěním *Eink32˙Executor* je zaručen bezpečný provoz vybraného jízdního řádu. Vzhledem k tomu, že tento spouštěč není prací mojí, nýbrž prací vedoucího práce, a mé úpravy byli pouze velmi malé, nebudu zde program detailněji popisovat.

4.4.1 Knihovna *TrainTTLibrary*

Knihovna *TrainTTLibrary* nese několik tříd společných pro všechny programy v aplikaci. Jedná se o třídy zprostředkovávající TCP komunikaci, třídy pro kódování a dekódování zpráv a třídy reprezentující objekty kolejiště.

Tři třídy s názvy *TCPBase*, *TCPServer* a *TCPClient* slouží k vytvoření serveru na jedné straně a klienta na straně druhé. *TCPBase* je předkem zbylých dvou tříd, zahrnující metody pro zpracování dat, odeslání dat a další úkony společné pro obě třídy. Nutno zdůraznit, že tento systém pro asynchronní TCP komunikaci je dílem vedoucího práce. Z toho důvodu nebude podrobný popis těchto tříd náplní této práce.

Pro pochopení všech programů v aplikaci je klíčová nejen samotná znalost tříd v knihovně, ale také princip jejich využití. Knihovna obsahuje hned několik tříd, které mají v názvu klíčové slovo *Packet*. Podíváme-li se do tab. 4.1 na názvy jednotlivých typů zpráv a porovnáme je s názvy tříd, zjistíme, že každému typu náleží právě jedna třída s koncovkou *Packet* v názvu. Mimo těchto tříd pro konkrétní typy je zde ještě jedna třída nazvaná jen *Packet*. Ta obsahuje pro všechny typy společné vlastnosti a slouží jako předek pro ostatní třídy souvisejícími s pakety. Vlastnostmi třídy jsou hlavička, adresa, data, crc a kompletní paket vše v bajtové podobě. Dále textová podoba TCP paketu a typu zprávy. Třída *Packet* neobsahuje žádný konstruktor, na rozdíl od tříd konkrétních typů paketů, které mají dokonce konstruktory přetíženy. Lze tak vytvořit instanci konkrétního typu paketu, buď pomocí bajtového pole přijatým ze sériového portu, nebo pomocí textové zprávy přijaté z TCP komunikace. Některé třídy mají ještě jedno přetížení a to s několika vstupními argumenty, kde každý argument přímo odpovídá jedné informaci přenášené v paketu.

Správný způsob jak s třídami týkajících se paketové komunikace pracovat je následující. Přijde-li paket po sériovém portu, nejdříve vytvoříme novou instanci třídy *Packet* s prázdným konstruktorem a poté zavoláme její metodu *SetPropetiesFromByte* se vstupním argumentem polem bajtů, přijatým po sériovém portu. Tím získáme objekt typu *Packet*, u kterého lze zavoláním statické metody *RecognizeTCPType* zjistit typ a vytvořit tak už konkrétní typ paketu. Metoda pro rozpoznání typu zprávy vyžaduje vstupní argument ve tvaru textu, který již máme uvnitř vlastnosti *Type* díky předešlému zavolání metody *SetPropetiesFromByte*. Takže např. zjistíme-li, že příchozí typ je occupancy_section, tak vytvoříme novou instanci třídy *OccupancySectionPacket* s polem bajtů přijatého paketu jako vstupním argumentem konstruktoru. Nyní máme paket požadovaného typu a můžeme s ním pracovat podle potřeby.

Přijetí a zpracování textové zprávy z TCP komunikace se jen lehce liší od zpracování paketu přijatého ze sériového portu. Konkrétně zpracování začíná už samotným rozpoznáním typu zprávy prostřednictvím statické metody *RecognizeTCPType*. Jako vstupní argument zde nevyužijeme vlastnosti *Type*, nýbrž celé přijaté textové zprávy. Následné kroky jsou již stejné jako u zpracování paketu po sériové lince.

V knihovně nalezneme, kromě tříd zabývajících se TCP komunikací a tříd reprezentu-

jící pakety, také třídy reprezentující určitý objekt kolejiště a třídy uskutečňující načtení informací z konfiguračních souborů. Třídy reprezentující objekt jsou *Locomotive* a *Section*, zatímco třídy pro načtení konfiguračních souborů jsou *LocomotiveInfo*, *SectionInfo*, *Item* a *ConfigItem*.

Po obecnějším pohledu, na princip fungování několika tříd pracujících jako celek, se nyní podíváme na jednotlivé třídy a pokusíme se popsat veškeré jejich funkce a možnosti.

4.4.1.1 Třída *Packet*

Jak již bylo zmíněno výše, tato třída slouží jako předek pro zbytek tříd týkajících se kódování a dekódování paketů. Obsahuje několik metod a vlastností a také výčtů. které si nyní detailněji popíšeme.

Vlastnosti:

- *BytePacket*: Seznam obsahující všechny bajty paketu
- *TCPPacket*: Textová podoba TCP zprávy
- *Type*: Text s typem zprávy viz tab. 4.1
- *Head*: Bajt s hodnotou hlavičky paketu
- *Adress*: Seznam dvou bajtů s hodnotami adresy paketu
- *Data*: Seznam několika bajtů s hodnotami dat paketu
- *CRC*: Bajt s hodnotou crc paketu
- *NumberOfunit*: Číslo udávající číslo jednotky
- *NumberOfAdress*: Číslo udávající typ jednotky a směru přenosu viz tab. 2.12

Metody:

- *SetPropetiesFromByte*:

  Metoda bez návratové hodnoty, s jedním vstupním argumentem, ve tvaru bajtového pole. Cílem metody je naplnění vlastností daty z vloženého pole bajtů.

- *SetPropetiesFromTCP*:

  Metoda bez návratové hodnoty, s jedním vstupním argumentem, ve tvaru textového řetězce. Cílem metody je naplnění vlastností daty z vložené TCP zprávy.

- *AssingType*:

  Metoda bez návratové hodnoty a žádným vstupním argumentem. Cílem metody je přiřazení typu zprávy na základě ostatních vlastností.

- *RecognizeTCPType*:

  Statická metoda s návratovou hodnotou typu *dataType* a vstupním argumentem ve tvaru textového řetězce. Cílem metody je navrácení typu zprávy ze zadaného textového řetězce. Řetězec může být celá přijatá zpráva, nebo jen název typu.

- *VypocetCRC*:

  Metoda s návratovou hodnotou crc bajtu a vstupním argumentem ve tvaru seznamu bajtů. Cílem metody je výpočet hodnoty crc na základě zadaného bajtového pole.

- *AdressToNumberOfAdressAndUnit*:

  Metoda bez návratové hodnoty a žádným vstupním argumentem. Cílem metody je v daném paketu převést informaci z vlastnosti *Adress* do vlastností *NumberOfAdress* a *NumberOfUnit*.

- *NumberOfAdressAndUnitToAdress*:

  Metoda bez návratové hodnoty a žádným vstupním argumentem. Cílem metody je v daném paketu převést informace z vlastností *NumberOfAdress* a *NumberOfUnit* do vlastnosti *Adress*.

- *SetBytePacket*:

  Metoda bez návratové hodnoty a žádným vstupním argumentem. Cílem metody je poskládání vlastnosti *BytePacket* z vlastností *Head*, *Adress*, *Data* a *CRC*.

- *UnknowPacket*:

  Metoda s návratovou hodnotou typu textový řetězec a vstupním argumentem ve tvaru opět ve tvaru textového řetězce. Cílem metody je nahrazení typu zprávy za typ *unknow*.

- *GapToUnderLine*:

  Metoda s návratovou hodnotou typu textový řetězec a vstupním argumentem ve tvaru opět ve tvaru textového řetězce. Cílem metody je nahrazení všech mezer za podtržítka v zadaném textovém řetězci.

- *UnderLineToGap*:

  Metoda s návratovou hodnotou typu textový řetězec a vstupním argumentem ve tvaru opět ve tvaru textového řetězce. Cílem metody je nahrazení všech podtržítek za mezery v zadaném textovém řetězci.

Výčty:

- *dataType*: Výčet typů zpráv viz 4.1.

- *unitAdress*: Výčet možných adres jednotek viz tab. 2.12.
- *unitInstruction*: Výčet instrukcí pro komunikaci řídícího PC s úsekovou jednotkou viz tab. 2.15 resp. tab. 2.16.

4.4.1.2 Třída *TrainMotionPacket*

Jak je patrné z názvu, primárním cílem tohoto paketu je uvést lokomotivu do pohybu. Přesněji se jedná o jednoduchý příkaz, pro jízdu konkrétní lokomotivy, danou rychlostí a směrem. K tomuto účelu přidává ke svému předkovi vlastnosti *Name*, *Reverse*, *Speed* a *ID*. Stejně jako předchozí třída, i *TrainMotionPacket* obsahuje tři konstruktory. První dva klasicky pro vytvoření instance z bajtového paketu, nebo TCP zprávy a třetí konstruktor pro vytvoření instance ze zadaných informací o lokomotivě, směru a rychlosti jízdy.

Vlastnosti:

- *Name*: Vlastnost typu textový řetězec obsahující název lokomotivy.
- *ID*: Vlastnost typu bez-znaménková celočíselná hodnota obsahující ID lokomotivy.
- *Speed*: Vlastnost typu bajt obsahující hodnotu rychlosti z rozsahu 0 až 31.
- *Reverse*: Vlastnost typu bool obsahující logickou hodnotu *true*, pokud má lokomotiva couvat a logickou hodnotu *false*, má-li lokomotiva jet dopředu.

Metody:

- *ComposeByte*:

  Metoda bez návratové hodnoty a s žádným vstupním argumentem. Cílem metody je seskládaní bajtového paketu a odvození zbytku vlastností ve třídě.

- *SecondByte*:

  Metoda s návratovou hodnotou typu bajt a s žádným vstupním argumentem. Cílem metody je poskládání druhého datového bajtu nesoucí informace o jízdě.

4.4.1.3 Třída *TrainFunctionPacket*

Úkolem této instrukce je rozsvícení nebo zhasnutí světlometů lokomotivy. Z toho důvodu je charakter tohoto paketu téměř shodný s paketem *TrainMotionPacket*. Rozdíl nalezeme ve tvaru druhého datového bajtu, který má tentokrát tvar pro řízení funkcí lokomotiv a nikoliv pro ovládání pohybu lokomotivy. S tím souvisejí také jiné vlastnosti třídy, a mírné odlišnosti v tělech metod. Konstruktory mají opět stejnou povahu, to znamená možnost vytvoření instance pomocí bajtového paketu, TCP zprávy, anebo pomocí přímo zadané lokomotivy a logické hodnoty nesoucí informaci o stavu světlometů.

Vlastnosti:

- *Name*: Vlastnost typu textový řetězec obsahující název lokomotivy.
- *ID*: Vlastnost typu bez-znaménková celočíselná hodnota obsahující ID lokomotivy .
- *Lights*: Vlastnost typu bool obsahující logickou hodnotu *true*, pokud má lokomotiva zapnout světlomety a logickou hodnotu *false*, má-li lokomotiva zhasnout světlomety.

Metody:

- *ComposeByte*:

  Metoda bez návratové hodnoty a s žádným vstupním argumentem. Cílem metody je seskládaní bajtového paketu a odvození zbytku vlastností ve třídě.

- *SecondByte*:

  Metoda s návratovou hodnotou typu bajt a s žádným vstupním argumentem. Cílem metody je poskládání druhého datového bajtu nesoucí informace o stavu světlometů.

4.4.1.4 Třída *TrainMotionInstructionPacket*

Úlohou *TrainMotionInstructionPacket* je dojet s určeným vlakem do konkrétního místa na kolejišti. K tomu na rozdíl od ostatních paketů používá lehce odlišnou myšlenku. Ta spočívá v přidání celého pohybového paketu svému předkovi ve formě vlastnosti, tudíž tu dostáváme paket uvnitř paketu. To znamená, že chceme-li pracovat s vlastnosti jako je rychlost, směr nebo název lokomotivy, musíme k nim přistupovat právě přes tuto vlastnost pohybového paketu. Paket také přidává svému předkovi další dvě vlastnosti, nikoliv však žádnou metodu. I zde jsou tři konstruktory shodné s paketem pro pohyb vlaku viz 4.4.1.2. Jediný rozdíl je, že pro vytvoření instance pomocí zadané lokomotivy, směru a rychlosti jízdy je potřeba doplnit ještě cílovou stanici a dojezdový čas.

Vlastnosti:

- *TrainMoveInfo*: Vlastnost obsahující celý pohybový paket
- *Section*: Stanice kam má lokomotiva dojet.
- *WaitTime*: Dojezdový čas, tedy jak dlouho má lokomotiva pokračovat v jízdě, potom co je zaregistrována cílovým úsekem.

4.4.1.5 Třída *UnitInstructionPacket*

Potomek třídy *Packet* zaštiťující instrukce pro nastavení úsekové jednotky. Třída je velice jednoduchá, k vlastnostem svého předka přidává pouze jednu další, a to *UnitInstruction* typu *unitInstruction*. Jsou zde také tři konstruktory pro vytvoření instance pomocí bajtového paketu, TCP zprávy, a nebo pomocí zadání požadované instrukce typu *unitInstruction* a pořadového čísla a adresy cílové jednotky.

Vlastnosti:

- *UnitInstruction*: Vlastnost typu *unitInstruction* obsahující klíčovou instrukci směřovanou do úsekové jednotky.

Metody:

- *stringToUnitInstruction*:

  Statická metoda s návratovou hodnotou typu *unitInstruction* a vstupním argumentem ve tvaru textového řetězce. Cílem metody je pouhé převedení zadané textové instrukce na typ *unitInstruction*.

4.4.1.6 Třída *UnitInfoPacket*

Zatímco třída *UnitInstructionPacket* slouží k popsání instrukcí tekoucích do úsekových jednotek, tak třída *UnitInfoPacket* je naopak určena pro informace mířené z jednotek do řídícího PC. K tomu účelu obsahuje vlastnost, která je stejně jako v případě třídy *UnitInstructionPacket* typu *unitInstruction*. Může se zdát zvláštní, že vlastnost je stejného typu jako v předchozím případě, ale podíváme-li se na tab. 2.15 a tab. 2.16, zjistíme, že významy zprávy jsou velice podobné a tudíž není problém ve využití pouze jednoho typu proměnné pro oba směry toku dat. Konstruktory jsou také zcela shodné s třídou *UnitInstructionPacket*.

Vlastnosti:

- *UnitInfo*: Vlastnost typu *unitInstruction* obsahující klíčovou informaci z úsekové jednotky.

4.4.1.7 Třída *OccupancySectionPacket*

Prozatím posledním potomkem třídy *Packet* je třída *OccupancySectionPacket* sloužící k zabalení informací o obsazenosti úseků z konkrétní úsekové jednotky. Jedná se tedy o paket, u kterého se nepředpokládá, že by ho chtěl programátor vytvářet ve svém kódu. Proto má třída pouze dva konstruktory a to pro vytvoření instance z bajtového paketu anebo textové TCP zprávy. V současné době je tento paket konstruován tak, že při přiřazení informací o úseku záleží na pořadí vypsaných úseků v konfiguračním souboru.

Vlastnosti:

- *Sections*: Vlastnost v podobě Seznamu proměnných typu *Section*. Pod typem *Section* jsou skryty informace o názvu a obsazenosti úseku.

Metody:

- *RecognizeSection*:

  Metoda bez návratové hodnoty a vstupním argumentem ve tvaru textového řetězce. Cílem metody ze zadané textové TCP zprávy odvodit vlastnost *NumberOfSection* a přiřadit obsah obsazeností jednotlivých úseků do vlastnosti *Sections*.

- *RecognizeUnit*:

  Metoda bez návratové hodnoty a s žádným vstupním argumentem. Cílem metody je přiřazení hodnot do vlastnosti *NumberOfSection* z příchozího bajtového paketu.

4.4.1.8 Třída *Locomotive*

Od popisu tříd reprezentující pakety se dostáváme k třídám představujícím určitý objekt kolejiště. Objektem kolejiště se rozumí v tomto případě lokomotiva. Lokomotiva má pouze dvě vlastnosti a to název a ID. Konstruktory této třídy jsou tři a to pro vytvoření instance pomocí zadaného ID, nebo pomocí zadaného názvu lokomotivy popř. pomocí obou zadaných vlastností. Dále třída obsahuje přepsanou metodu *ToString*, která nyní vrací pouze název lokomotivy.

Vlastnosti:

- *Name*: Vlastnost typu textový řetězec nesoucí pracovní název lokomotivy.
- *ID*: Vlastnost typu bez-znaménková celočíselná hodnota s obsahem identifikačního čísla lokomotivy.

4.4.1.9 Třída *Section*

Třída *Section* zastupuje určitý úsek na kolejišti, ten musí mít hned několik vlastností. Základními vlastnostmi jsou název úseku a pořadové číslo jednotky, který daný úsek snímá. Za povšimnutí stojí, že třídy reprezentující objekt na kolejišti již pochopitelně nedědí ze třídy *Packet*, a proto musíme vlastnost s číslem jednotky vytvořit přímo ve třídě *Section*. Další vlastností je *Current* představující změřený proud na úseku. Změna této vlastnosti okamžitě vyvolá změnu poslední vlastnosti jasně vypovídající o stavu obsazenosti úseku. K tomu účelu stačí logická hodnota, kde pravda signalizuje obsazený úsek a nepravda zase úsek na němž se žádná lokomotiva nevyskytuje. Stejně tak, jako u třídy reprezentující lokomotivu, i zde je přepsaná metoda *ToString* na pouhý výpis názvu úseku. K vytvoření instance této třídy lze využít jeden ze tří konstruktorů. První potřebuje vstupní argumenty název úseku, číslo jednotky a naměřený proud. Druhý pouze název úseku a číslo jednotky, naměřený proud je automaticky nastaven na hodnotu nula. Poslednímu konstruktoru stačí pouze název úseku, sám si totiž zjistí číslo jednotky z seznamu úseků a proud nastaví opět na hodnotu nula.

Vlastnosti:

- *Name*: Vlastnost typu textový řetězec nesoucí pracovní název úseku.
- *NumberOfUnit*: Bez-znaménková celočíselná hodnota představující pořadové číslo jednotky, která daný úsek snímá.
- *Current*: Bez-znaménková celočíselná hodnota reprezentující změřený proud v úseku.

- *Occupancy*: Logická hodnota vypovídající o obsazení kolejového úseku.

Metody:

- *OccupancySection*:

  Metoda bez návratové hodnoty a s žádným vstupním argumentem. Cílem metody je vyhodnocení naměřeného proudu a přiřazení do vlastnosti *Occupancy* logickou hodnotu.

4.4.1.10 Třída *ConfigItem*

Následující čtyři třídy jsou určeny pro práci s daty z konfiguračních souborů. Oba konfigurační soubory mají zápisy ve tvaru textového řetězce a k nim přiřazené celočíselné hodnoty viz 4.4.1.11. Právě k načtení těchto obecných zápisů slouží třída *ConfigItem*. Má jedinou vlastnost a to seznam proměných typu *Item* s obsahem veškerých dat z načteného souboru. Načítání souboru probíhá přímo v konstruktoru třídy, který jako vstupní argument přijímá textový řetězec s cestou k zdrojovému souboru. Třída nadále se soubory nijak nepracuje, po vytvoření její instance s určitou cestou k souboru je její seznam naplněn zápisy s obecným typem *Item* a připraven tak k dalšímu zpracování. Mírnou zajímavostí je metoda *Serialize*, která není nikde volána, ale původně posloužila pro vytvoření konfiguračních souborů.

Vlastnosti:

- *Items*: Seznam proměnných typu *Item* obsahující načtené data z konfiguračního souboru.

Metody:

- *Serielize*:

  Metoda bez návratové hodnoty a s žádným vstupním argumentem. Cílem metody je vytvoření nového souboru s konfiguračními daty.

4.4.1.11 Třída *Item*

Třída bez jakýkoliv metod i konstruktoru. Slouží pouze jako obecná struktura pro načtení konfiguračních souborů viz 4.4.1.10.

Vlastnosti:

- *Str*: Obecný textový řetězec připravený pro další zpracování
- *Num*: Obecná bez-znaménková celočíselná hodnota.

4.4.1.12 Třída *LocomotiveInfo*

Třída slouží pro pohodlnou práci se seznamem všech lokomotiv. K tomu nepotřebuje žádný konstruktor, stačí pouze statická vlastnost s návratovou hodnotou seznamu lokomotiv. Seznam je vracen skrze metodu *Initloco*, která provádí přístup do konfiguračního souboru díky třídě *ConfigItem*. V konfiguračním souboru *locomotives* jsou ukryty názvy lokomotiv a jejich ID, které jsou nahrány zmíněnou třídou do seznamu položek typu *Item*. Následně převedena na příslušný typ *Locomotive* a vráceny uživateli. Kromě metody pro inicializaci lokomotiv jsou tu ještě dvě metody, jedna pro zjištění názvu lokomotivy po zadání příslušného ID a druhá naopak pro zjištění ID po zadání názvu.

Vlastnosti:

- *listOfLocomotives*: Seznam proměnných typu *Locomotive* obsahující všechny lokomotivy načtené z konfiguračního souboru *locomotives*.

Metody:

- *IDToName*:

  Statická metoda s návratovou hodnotou typu textový řetězec a vstupním argumentem typu bez-znaménková celočíselná hodnota. Cílem metody podle seznamu lokomotiv přiřadit zadanému ID název lokomotivy.

- *NameToID*:

  Statická metoda s návratovou hodnotou typu bez-znaménková celočíselná hodnota a vstupním argumentem typu textový řetězec. Cílem metody podle seznamu lokomotiv přiřadit zadanému názvu ID lokomotivy.

4.4.1.13 Třída *SectionInfo*

Charakter této třídy je naprosto totožný s třídou *trida˙LocomotiveInfo*. Rozdíl je pouze v obsahu seznamu, kde se tentokrát místo lokomotiv nachází úseky. Třída také nedisponuje žádnými statickými metodami, je zde pouze metoda pro inicializaci úseků. V konfiguračním souboru *sections* nalezneme názvy úseků a pořadové čísla jednotky, ke které úsek náleží.

Vlastnosti:

- *listOfSections*: Seznam proměnných typu *Section* obsahující všechny úseky načtené z konfiguračního souboru *sections*.

4.4.2 Program *TCP Server Train TT*

Základní obsluhu pro program *TCP Server Train TT* jsme si již vysvětliti v kap. 4.2. Nyní se blíže zaměříme na strukturu kódu a vysvětlení mechanik v něm použitých. Nejdůležitějšími úlohami programu je zprostředkování TCP server pro ostatní klienty a navázání

spojení s kolejištěm zkr-ze sériový port. Důležité také je, že pro optimální rychlost komunikace není *TCP Server Train TT* v podobě klikacího okna jako zbytek programů, ale pouze v podobě konzolové aplikace. Na první pohled je v projektu vidět spousta globálních proměnných, které si však popíšeme až při jejich konkrétním využití, protože vysvětlování bez kontextu v kódu by nebylo přehledné. Celý program je provázen výpisem jednotlivých zpráv pro uživatele na konzoli. Pro přehlednost ve většině případů nebudou v následujícím detailním popisu programu tyto výpisy komentovány, protože jejich úloha je na první pohled zřejmá. Jediná zajímavost na výpisu zpráv do konzole je využití tzv. *NuGet* balíčku s názvem *ColorfulConsole*, který umožňuje vybírat barvu vypisovaného textu. Pro zřejmost významu zpráv byly definovány proměnné *TCPInfo*, *TCPError*, *TCPDataRecived*, *TCPDataSend* nesoucí pevně nastavené barvy pro čtyři různé typy zpráv. Jimi jsou zprávy informující o stavu systému vypisovány barvou žlutou, zprávy signalizující narušení chodu aplikace červenou barvou, zprávy odeslané klientům TCP komunikace barvou zelenou a přijaté zprávy od klientů barvou modrou.

4.4.2.1 Metoda *Main*

V metodě *Main* dochází nejprve k přednastavení funkcí jako je zpřístupnění TCP komunikace, otevření příslušného sériového portu a spuštění všech potřebných časovačů. Pro tyto potřeby jsou postupně volány metody *StartTCPServer* viz 4.4.2.2, *ConnectToSerialPort* viz 4.4.2.3 a metoda *StartTimers* viz 4.4.2.4. Následuje smyčka vytvořená cyklem *while*, kde podmínkou je předem vytvořená logická proměnná ze začátku nastavená na hodnotu true. Tím je zajištěn nepřetržitý cyklus do doby, dokud se nezmění hodnota této logické proměnné. Uvnitř smyčky nalezneme cyklické snímání stisku libovolné klávesy. Je-li stisknuta klávesa *Q*, logická hodnota je změněna na hodnotu *false* a cyklus *while* je tím ukončen. Dále dochází k přerušení TCP komunikace a program je ukončen. Lze z toho odvodit že cyklus while slouží pouze k tomu aby udržení běhu programu a veškerou funkcionalitu se starají časovače, nebo asynchronní procesy.

4.4.2.2 Metoda *StartTCPServer*

Metoda zajišťující vytvoření TCP serveru přes třídu *TCPServer* a přiřazení příslušných metod obslužným událostem. Konkrétně události *OnClientConnected* je přiřazena metoda *TCP_NewClient* viz 4.4.2.5, události *OnClientDisconnected* je přiřazena metoda *TCP_DisconnectClient* viz 4.4.2.6 a nejdůležitější události *DataReceived* je přiřazena metoda *TCP_DataRecv* viz 4.4.2.7. Dále je vybrán typ komunikace, kde je konec zprávy demonstrován znaky pro odřádkování. Nakonec je server uveden do aktivního stavu na zadaném portu.

4.4.2.3 Metoda *ConnectToSerialPort*

Cílem metody je otevření portu a zajištění tak bezchybnou komunikaci s kolejištěm. Celý obsah metody se kvůli ošetření výjimek nachází v bloku *try-catch* a tak v případě nesplnění příkazů v bloku *try* je program ukončen. V opačném případě se nejprve nakonfigurují vlastnosti sériového portu a následně je otevřen a tok dat má volný průchod. Pro zpracování dat je zde události *DataReceived* přiřazena metoda *SerialDataReceived* viz 4.4.2.8.

4.4.2.4 Metoda *StartTimers*

Ve chvíli, kdy je aktivní komunikace s kolejištěm přes sériový port a zpřístupněn server pro síťovou komunikaci, je potřeba vytvořit a spustit určité časovače. Jsou tu dva hlavní časovače běžící neustále při běhu programu. První nese jméno *timerTCPsend* a slouží pro odesílání přijatých dat přes sériovou linku, zatímco druhý, nazvaný *timerSerialSend* zaručuje cyklické odesílání paketů do řídících jednotek. Systém je totiž navržen tak, že data ze sériového portu jsou přijímána asynchronně pomocí metody *SerialDataReceived* viz 4.4.2.8, kde jsou zformována do celých paketů, ale k jejich odesílání všem klientům dochází v pravidelných intervalech až v těle metody *SendTCPData_Tick* viz 4.4.2.9. Naopak data která cestují po sériové lince do řídících jednotek kolejiště se dělí na dvě skupiny. A to sice na zprávy, které stačí poslat pouze jednou, jako např. zpráva o rozsvícení světel lokomotivy, a zprávy, které je potřeba posílat cyklicky, jako např. zpráva určená k pohybu vlaku. Právě k odesílání druhého zmíněného typu zpráv je tady časovač *timerSerialSend*. Více informací o zasílání těchto cyklických zpráv se nachází v popisu metody *TCP_DataRecv* viz 4.4.2.7. Kromě hlavních časovačů je potřeba vytvořit ještě dojezdové časovače sloužící ke správnému plnění pohybových instrukcí viz 4.4.1.4. Na rozdíl od hlavních časovačů, které jsou typu *System.Timers.Timer* jsou časovače pro dojezdové odpočty typu *MyTimer*. Jedná se o typ vytvořen exklusivně pro potřeby těchto speciálních časovačů. Dědí ze třídy *System.Timers.Timer* a doplňuje ji o vlastnosti *Count*, *Counter* a *TrainMotionPacket*. Všem dojezdovým časovačům je nastaven jednotný interval 10ms. Detailněji vysvětleny tyto vlastnosti a princip jejich využití v popisu metody *MyTimer_Elapsed* viz 4.4.2.11.

V těle této metody se tedy nachází nejprve cyklus vytvářející každé lokomotivě její vlastní dojezdový časovač a ukládá ho do seznamu *StopTimers*. Následně jsou vytvořeny a zapnuty dva hlavní časovače s nastaveným intervalem tikání 50 ms a přiřazenými příslušnými metodami viz výše. Tím jsou všechny časovače připraveny na následné použití.

4.4.2.5 Metoda *TCP_NewClient*

Metoda reagující na připojení nového klienta pouhým výpisem jeho portu na konzoli.

4.4.2.6 Metoda *TCP_DisconnectClient*

Stejně jako metoda předešlá, slouží pouze pro informování uživatele o faktu, že jeden z klientů ukončil TCP spojení pomocí výpisu na konzoli.

4.4.2.7 Metoda *TCP_DataRecv*

Metoda zajišťující asynchronií příjem zpráv a zabezpečující jejich zpracování. Princip spočívá ve využití tříd blíže zmíněných v kapitole 4.4.1. Tato metoda je tedy vyvolána pouze při přijetí kompletní textové zprávy od jednoho z klientů. Identifikace klienta a obsah zprávy jsou uloženy ve vstupních argumentech vstupujících do metody. K vyzvednutí informací o odesílateli zprávy je nutné přetypovat sender jako *SocketObject*, ze kterého následně můžeme převzít informace a uložit je do předpřipravené proměnné *ipeClient* typu *IPEndPoint*. Dalším krokem je zpracování samotné textové zprávy. Vstupní argument *e* typu *TCPReceivedEventArgs* obsahuje vlastnost *data*, ve které je tato zpráva uložena ve formátu *object*. Je tedy potřeba testovat je-li možné přetypování tohoto objektu na typ *String* a posléze přetypování provést. Následují příkazy, které mají za cíl rozpoznat typ zprávy a podle toho vykonat příslušné opatření. Rozpoznání paketu je provedeno statickou metodou *RecognizeTCPType* ve třídě *Packet* viz 4.4.1.1. Je potřeba si uvědomit, že zatím máme pouze textový řetězec se zprávou a nikoliv paket příslušného typu, proto je nutné tento příslušný paket vytvořit a až poté s ním pracovat.

Před tím než si vysvětlíme instrukce, které se vykonávají pro jednotlivé typy zprávy po jejím rozpoznání, popíšeme si k tomu využitý princip. Nejsou-li přijatá data cyklicky odesílána viz 4.4.2.4, je jejich zpracování jednoduché. Pouze je vypsána zpráva uživateli na konzoli a je odeslán příslušný paket po sériové lince. Naproti tomu zprávy potřebující cyklické odesílání, jako např. zpráva o pohybu vlaku, je třeba tyto zprávy uložit a dále zpracovávat. Proměnné které k uložení těchto zpráv využíváme jsou seznamy *trainMotionInstructions* a *trainMotionPackets*. Systém je totiž navržen tak, že již zmíněný časovač *timerSerialSend* volá v pravidelných intervalech metodu *SendSerialData_Tick* viz 4.4.2.10, uvnitř které jsou prohledávány právě tyto seznamy a nejsou-li prázdné, jsou provedeny opatření k odeslání příslušných dat.

Nyní si tedy popíšeme instrukce pro jednotlivé typy zpráv po jejich rozpoznání. Pro každý typ je začátek instrukcí podobný. Nejprve se převede textová zpráva na příslušný druh paketu a posléze je kontrolováno, zda není nově vytvořený paket typu *unknow*, protože tento typ indikuje chybně odeslaný formát paketu. Všechny typy zpráv viz tab. 4.1. Jestliže vytvoření paketu proběhlo bez komplikací, je možné přistoupit ke zpracování podle příslušných typů. Jedním z rozpoznaných typů může být zpráva o pohybu vlaku *train_move*. V takovém případě je nejprve prozkoumán seznam *trainMotionInstructions*. V případě nalezení záznamu se shodnou lokomotivu, jako je lokomotiva v novém přijatém paketu, je paket z tohoto seznamu vyhozen. Kdyby k tomu nedošlo, mohla by nastat situace, kde by program odesílal pro jednu lokomotivu dva různé pokyny pro pohyb. V

dalším kroku je podobným principem zkontrolováno, jestli se pro lokomotivu z nového paketu, už nenachází záznam v seznamu *trainMotionPackets*. Pokud ne, je vytvořen nový záznam s obsahem nového paketu, pokud ano, je záznam přepsán novým obsahem paketu. Tím končí zpracování zprávy v této metodě a o zbytek se již musí postarat metoda *SendSerialData_Tick* viz 4.4.2.10. Dalším typem přijaté zprávy může být *train_move_to_place* a pracování tohoto typu je téměř totožný s typem předešlým. Je provedeno prohledání seznamů, a jsou provedeny nutné instrukce viz výše. Jediný rozdíl je v přidání cyklu, který porovná cílovou stanici uvedenou v nově přijatém paketu s obsazeností kolejových úseků a v případě, že lokomotiva se již nachází na místě kam se ji zpráva snaží dovést, zpráva je dále ignorována. Další dva možné typy přijatých zpráv jsou *train_function* a *train_unit˙instruction*. Pro oba typy platí podobný charakter zpracování a to v podobě vytvoření paketu z textové zprávy, vypsání informace na konzoli a odeslání paketu po sériové lince. Posledním příchozím typem, který je očekáván je typ *unknow*. Pro ten neprobíhá žádné zpracování, a je pouze vypsán na konzoli. Podíváme-li se do tab. 4.1 zjistíme, že metoda nepodporuje všechny existující typy zpráv. Je tomu tak, protože jednoduše nepředpokládáme určité typy zpráv od klientů. Např. zpráva o obsazenosti kolejí chodí pravidelně přes sériový port z kolejiště a nedávalo by smysl kdyby obsazenost chodila od některého z klientů.

4.4.2.8 Metoda *SerialDataReceived*

Zatímco v metodě *TCP_DataRecv* viz 4.4.2.7 byl zajištěn asynchronní příjem dat TCP komunikace, v metodě nazvané *SerialDataReceived* je uskutečněn asynchronní příjem dat ze sériového portu. Jedná se tedy o obslužnou metodu pro událost přijmutí dat, kde je metoda volána při přijmutí každého bajtu. Protože data chodí bajt po bajtu, je potřeba jeden po druhém ukládat a následně testovat, zda-li jako celek odpovídají formátu paketu tj. hlavička, příslušná délka podle hlavičky a zakončení v podobě crc. K těmto účelům využijeme globální souběžné fronty nazvané *packet*, do které budeme ukládat přijaté nově vytvořené pakety. Následné zpracování přijatých dat přes sériový port probíhá v metodě *SendTCPData_Tick* viz 4.4.2.9. Fronta musí být souběžná, jelikož přijetí dat zde probíhá v jiném vláknu než metoda pro zpracování. Další klíčovou proměnou je seznam bajtů nazvaný *packetInCheck*. Do něj jsou postupně přidávány příchozí data, pokud je podezření, že by data mohli poskládat paket. To znamená že tento seznam se začne plnit ve chvíli kdy byl detekován bajt který může reprezentovat hlavičku paketu. Posléze je odpočítáno tolik bajtů, kolik je uvedeno v hlavičce viz 2.4 a vyhodnoceno výpočtem crc jestli se opravdu jedná o paket. V případě že ano, je paket vložen do fronty *packet* a seznam *packetInCheck* je vyprázdněn. Ale v opačném případě nemůžeme seznam jednoduše vyprázdnit, protože je možné že jeden z bajtů uvnitř prozkoumávaného paketu byl právě onen začátek skutečného paketu a tímto způsobem bychom data ztráceli. Řešení spočívá v zavedení nového seznamu do kterého jsou data z prvního seznamu převedena a namísto opětovného čtení dat ze sériového portu jsou přednostně čteny data ze seznamu. Seznam

zajišťující přezkoumání těchto bajtů znovu se nazývá *secondCheckPacket*. Úplně poslední globální proměnou sloužící tomuto algoritmu je logická hodnota *chcekPacket*. Ta slouží pouze jako jednoduchá signalizace, zda se již některý paket skládá, nebo jestli přijatá data mohou být hlavičkou nového paketu.

4.4.2.9 Metoda *SendTCPData_Tick*

Metoda periodicky volána časovačem každým 50 ms, která je úzce spjata se zpracováním dat ze sériového portu viz 4.4.2.8. Konkrétně se stará o rozeslání úspěšně přijatých paketů jednotlivým klientům TCP komunikace. Celá metoda je v cyklu while s podmínkou, dokud je co číst z fronty přijatých paketů, tak opakuj. Následuje pokus o vyzvednutí paketu z fronty. V úspěšném případě je za pomocí statické metody *RecognizeTCPType* ve třídě *Packet* viz 4.4.1.1 zjištěn typ přijatého paketu. Nepředpokládá se, že z řídících jednotek přijde jiný typ zprávy než jsou zprávy o obsazenosti kolejových úseků a nebo zprávy potvrzující změnu konfigurace úsekové jednotky viz 4.1. Z kapitoly 2.4.1.1 již víme, že dalším příchozím paketem z řídících jednotek kolejiště, je tzv. watchdog zpráva. Ta je sice prozatím ignorována, ale v budoucím vývoji je možné využití této zprávy ke kontrole běhu kolejiště. Po provedení rozpoznání typu zprávy je vytvořen příslušný paket a odeslán klientům. V případě zprávy o obsazenosti kolejových úseků je, kromě odeslání paketu, ještě volána metoda *SaveOccupancySection* viz 4.4.2.12.

4.4.2.10 Metoda *SendSerialData_Tick*

Obslužná metoda časovače *timerSerialSend* zajišťující cyklické odesílání pokynů pro pohyb lokomotiv. Žádné jiné typy zpráv totiž nejsou potřeba odesílat cyklicky a stačí je odeslat pouze jednou. Pro přehlednost je náplň této metody rozdělena do dvou metod bez návratové hodnoty a sice *SendTrainMotionPacket* viz 4.4.2.13 a *SendTrainMotionInstruction* viz 4.4.2.14.

4.4.2.11 Metoda *MyTimer_Elapsed*

Jak již bylo přiblíženo v kapitole *StartTimers* viz 4.4.2.4, tento typ časovače je vytvořený speciálně pro uskutečnění dojezdu vlaku do stanice. K tomuto účelu dopomáhají vlastnosti *Count*, *Counter* a *TrainMotionPacket*. Událost časovače je volána každých 10ms a časovač je sám ukončen napočítá-li požadované hodnoty. Vlastnost *Count* obsahuje údaj kolikrát má být metoda volána než ukončí činnost časovače. Ke sledování počtu iterací zase slouží vlastnost *Counter*. Poslední vlastnost *TrainMotionPacket* obsahuje samotný paket, s již z nastavenou lokomotivou a rychlostí, který je odesílán aby byl vlak udržen v pohybu. Ve chvíli, kdy *Counter* napočítá hodnoty *Count* je nastavena hodnota rychlosti v paketu na nulu a paket je odeslán.

Na začátku metody je provedeno přetypování senderu na typ našeho časovače *MyTimer* a následně je provedena inkrementace vlastnosti *Counter*. Poté přichází vyhodnocení,

zda-li už *Counter* nepřesáhl hodnotu *Count*. Jestliže ano, proběhne vytvoření nového paketu s rychlostí nula a jeho odeslání pomocí metody *SendSerialData* viz 4.4.2.15 a činnost časovače je ukončena. Pokud však podmínka splněna není, proběhne odeslání paketu o pohybu lokomotivy pouze jednou za čtyřicet opakování volání této metody. Ve výsledku se tedy paket o pohybu odesílá jednou za 400 ms. Důvodem je zahalování USB/CAN převodníku při odesílání dat příliš velkou rychlostí.

4.4.2.12 Metoda *SaveOccupancySection*

Velice jednoduchá metoda se vstupním argumentem ve tvaru seznamu naplněného proměnnými typu *Section* a bez návratové hodnoty. Cíl této metody je v naplnění globální proměnné *occupancySections* daty o obsazenosti kolejových úseků. Tato proměnná má stejný formát jako vstupní argument metody a sice seznam typu *Section*. Neboť je nutné, aby celý program znal aktuální pozice lokomotiv na kolejišti a mohl je tak řídit. Nejprve zde dojde k porovnání nově přijatých dat o obsazenosti s těmi starými. Pokud jsou dané úseky již v proměnné *occupancySections* obsaženy, dochází k přepsání jejich hodnot nesoucích obsazenost. V případě že v proměnné tyto úseky ještě nejsou, je jim vytvořen nový záznam s aktuálními hodnotami.

4.4.2.13 Metoda *SendTrainMotionPacket*

V kapitole s popisem metody *SendSerialData_Tick* viz 4.4.2.10 jsme se dozvěděli, ře tato metoda slouží jen pro větší rozvrstvení a zpřehlednění programu. Konkrétně zajišťuje cyklické odesílání paketů o prostém pohybu vlaku do řídícího systému kolejiště. To je rozdíl oproti metodě *SendTrainMotionInstruction* viz 4.4.2.14, která se zase stará o cyklické odesílání paketů pro dojezd vlaku do určitého úseku. Ještě před důkladnějším probráním těla samotné funkce, je potřeba vysvětlit princip cyklického odesílání. Potíž je v převodníku USB/CAN, který se při nadměrnému množství procházejících dat zahltí a zastaví svou činnost. Na druhou stranu je nutné posílat pakety do systému kolejiště ve vysoké frekvenci, z důvodu zajištění rychlé reakce na změnu rychlosti, nebo zastavení vlaku. My už z kapitoly 2.4.1.1 víme, že DCC generátor potřebuje neustále přijímat některé pakety, jinak zastaví všechny vlaky na kolejišti. Je proto klíčové zajistit odesílání dostatečně pomalé, aby nebyl zahlcen převodník, ale zároveň dostatečně rychlé, aby se vlaky samovolně nezastavovali a k tomu navíc dokázat pružně reagovat na změnu řízení lokomotivy. Pakety připravené k cyklickému odesílání jsou uvnitř seznamů *trainMotionInstructions* resp. *trainMotionPackets*, jak už bylo řečeno, při popisu metody *TCP_DataRecv* viz 4.4.2.7. K řešení uvedené nesnáze nám nebudou stačit pouze tyto dvě globální proměnné, ale zavedeme další čtyři. Přesněji řečeno se jedná o dva seznamy *oldTrainMotionInstructions* resp. *oldTrainMotionPackets*, které budou sloužit jako paměť posledních odeslaných paketů do řídících jednotek. Přítomnost posledních odeslaných dat nám umožní testovat, jsou-li nové pakety, které se snaží metoda odeslat rozdílné či nikoliv. Při detekci rozdílných

paketů, je potřeba pakety okamžitě odeslat a zajistit tak rychlou reakci na změnu řízení. V případě kdy se pakety shodují s předešlými, je zapotřebí odeslat je pouze jednou za čas, aby nedošlo k zahlcení. O tom, kdy se nezměněná data odešlou rozhodují dva seznamy čítačů. Ke každému záznamu v seznamech *trainMotionInstructions* a *trainMotionPackets* je přiřazen jedna bez-znaménková celočíselná hodnota představující čítač. Jedná se tedy o dva globální seznamy s proměnnými typu *uint* nazvané *trainMotionPacketCounters* a *trainMotionInstructionCounters*. Ve zkratce je tedy zavolána metoda *SendTrainMotionPacket*, která nejdříve prochází všechny záznamy v seznamech a porovná hodnoty nových paketů s těmi starými. Pokud se liší jsou nové pakety neprodleně odeslány, jestli jsou stejné jako při posledním odeslání, jsou kontrolovány čítače. Čítače jsou nastaveny na odeslání dat napočítají-li hodnotu větší než osm. S přihlédnutím k intervalu volání této metody (50 ms), je doba mezi nezměněnými pakety 400 ms, což zaručuje plynulou jízdu.

Po průchodu výše rozepsaného algoritmu je následně prohledán seznam *trainMotionPackets* a jsou z něj odstraněny všechny záznamy s rychlostí menší než čtyři, tj. všechny typy zastavovacích rychlostí viz tab. tab˙kombinace˙zastaveni. Následně jsou také odstraněny příslušné záznamy v seznamech *oldTrainMotionPackets* a *trainMotionPacketCounters*.

4.4.2.14 Metoda *SendTrainMotionInstruction*

Účelem metody je zprostředkování cyklického odesílání paketů nesoucích zprávu o dojezdu vlaku do určitého úseku. Základním principem se nijak neliší od metody *SendTrainMotionPacket* viz 4.4.2.13, která zase zajišťuje cyklické odesílání paketů o prostém pohybu vlaku. Proto je zbytečné podrobně popisovat princip znovu. Funkční rozdíl je pouze ve způsobu vyhodnocení, kdy je záznam vyhozen ze seznamu. K tomu nedochází ve chvíli, kdy je nastavena rychlost na hodnotu menší než čtyři, jako u předchozí metody, ale ve chvíli, kdy je na příslušném úseku detekována přítomnost vlaku.

4.4.2.15 Metoda *SendSerialData*

Velice krátká metoda sloužící k prostému odeslání dat, přes sériový port, do řídících jednotek kolejiště. Vstupním argumentem metody je seznam bajtů obsahující data k odeslání. Data jsou následně převedena do pole bajtů a v takovémto formátu odeslána.

4.4.2.16 Metoda *OldSentPacket*

Metoda s dvojnásobným přetížením spravující seznamy obsahující poslední odeslané pakety. Důvod existence těchto seznamů viz 4.4.2.13. Vstupní argument může být buď typu *TrainMotionPacket* nebo typu *TrainMotionInstructionPacket*, podle toho spravuje příslušný seznam. Metoda jednoduše prohledá příslušný seznam a porovnává jeho název s názvem v paketu získaným ze vstupního argumentu. V případě nalezení shody v názvu, je starý paket nahrazen novou hodnotou. Nenastane-li shoda, je pro paket vytvořen nový

záznam v seznamu *oldTrainMotionInstructions* resp. *oldTrainMotionPackets* a také nový záznam v seznamu *trainMotionPacketCounters* resp. *trainMotionInstructionCounters* s nastavenou nulovou počáteční hodnotou.

4.4.3 Program *Visual Debug Control Train TT*

Nejjednodušší řízení kolejiště nabízí program *Visual Debug Control Train TT*. Okrajový popis a především informace pro ovládání programu viz 4.2.2. Jedná se o okenní neboli formulářovou aplikaci s jednoduchým uživatelským rozhraním, která svým principem uživateli nabízí možnost pohybovat vlaky z místa na místo a sledovat při tom obsazenost kolejiště. Další funkcí je odesílání konfiguračních zpráv pro řídící úsekové jednotky. Jelikož se jedná o formulářovou aplikaci, základní kostra je již vytvořená vývojovým prostředím, a proto popis tohoto programu zahrnuje pouze detailnější pohled na třídu *VisualDebugControl*. Konstruktor třídy obsahuje pouze předdefinovanou inicializaci komponent.

U metod obsluhujících události, vyvolané použitím ovládacích prvků aplikace je vždy provedeno bezpečné přetypování senderu na příslušný prvek. Pro přehlednost popisu nebude ta toto přetypování u jednotlivých metod upozorňováno.

4.4.3.1 Metoda *Form1_Load*

První metoda volaná při spuštění programu je metoda *Form1_Load*. Ta zahrnuje jedinou instrukci, a sice zavolání metody *StartTCPClient* viz. 4.4.3.2.

4.4.3.2 Metoda *StartTCPClient*

Zde se zařizuje připojení k TCP serveru. Charakter metody je v podstatě totožný s metodou *StartTCPServer* viz 4.4.2.2. Nejdříve je zjištěna IP adresa počítače, na kterém je program spuštěn a následně je vytvořen nový *TCPClient*. Program tedy zatím nepočítá s možností, že server a klient jsou spuštěny každý na jiném PC. Následuje přiřazení příslušných metod obslužným událostem TCP klienta podobně jako u serveru viz 4.4.2.2. Posledním krokem je pokus o připojení klienta k serveru. Při nezdaru je volána metoda *KlientCleanUp* viz 4.4.3.3.

4.4.3.3 Metoda *KlientCleanUp*

Jednoduchá metoda bez návratové hodnoty a s žádným vstupním argumentem zařizující bezpečné odpojení klienta. Nejdříve je kontrolováno, jestli vůbec nějaký klient je vytvořen. Jestliže ano, tak se ho pokusí odpojit od serveru. Následně smaže metody pro obsluhu příslušných událostí a nakonec klienta zlikviduje.

4.4.3.4 Metoda *KlientConnected*

Program potřebuje mít permanentní informaci o připojení k serveru, aby nedošlo k situaci, kdy se snaží odeslat data přes zavřený přenosový kanál. Z tohoto důvodu je zde zavedena logická globální proměnná *IsConnect*, tak aby se mohl program, za pomoci podmínky, kdykoliv dotázat na spojení se serverem. Proměnná je v výchozím stavu naplněna hodnotou false a k její změně na true může dojít právě v metodě *KlientConnected*. Metoda slouží jako obsluha události vyvolané připojením se k TCP serveru. To však ještě nestačí k tvrzení, že je klient úspěšně připojen a lze vést bezpečnou komunikaci. Abychom mohli klienta prohlásit za připojeného, musíme ještě zkontrolovat nenulovost dat přijatých ve vstupním argumentu. Pokud tato situace nastane, přichází změna proměnné *IsConnect* na hodnotu true a vypsání zprávy o připojení do textového pole v levé části formuláře. Vyvolání této metody s nulovými vstupními argumenty vyvolá reakci v podobě změny hodnoty *IsConnect* na hodnotu false a zavolání metody *KlientCleanUp* viz 4.4.3.3.

4.4.3.5 Metoda *TCPDisconnectClient*

Zatímco v metoda *KlientConnected* viz 4.4.3.4 prováděla obsluhu události připojení se k TCP serveru, metoda *TCPDisconnectClient* je zase volána v okamžik odpojení se od serveru. Jediná instrukce uvnitř této metody je změna hodnoty proměnné *IsConnect* na stav false. Význam proměnné vysvětlen viz 4.4.3.4.

4.4.3.6 Metoda *TCPDataRecv*

Metoda obsluhující událost *DataReceived* vyvolanou přijetím dat vyslaných TCP serverem. Jelikož ale asynchronní příjem dat probíhá v jiném vlákně, než se nachází vykreslování okna formuláře, je potřeba použít metodu *BeginInvoke* pro napojení dat do žádoucího vlákna. To znamená, že jediná funkce této metody je zavolat metodu *DataProcessing* viz 4.4.3.7.

4.4.3.7 Metoda *DataProcessing*

Po přijetí dat v metodě *TCPDataRecv* viz 4.4.3.6 přichází na řadu jejich zpracování. Tomu se děje velice obdobně jako u předchozího programu viz 4.4.2.7. U příchozí zprávy je zjištěn typ pomocí statické metody *RecognizeTCPType* ve třídě *Packet* viz 4.4.1.1 a podle toho přiděleno příslušné zpracování. V případě rozpoznání typu *occupancy_section* je vytvořen paket typu *OccupancySectionPacket* a použit jako vstupní argument pro metodu *GetSectionInformations* viz 4.4.3.8. Je-li však rozeznán jakýkoliv jiný typ zprávy, vyjma typů *watchdog*, nebo *occupancy_section*, proběhne pouhé vypsání zprávy do textového okna v levé části formuláře.

4.4.3.8 Metoda *GetSectionInformations*

Příchozí data o obsazenosti kolejiště musí být zobrazena na panelu diod v pravé části okna. Diody odpovídající příslušným úsekům se automaticky zobrazí poté, co program identifikuje paket obsazenosti dat právě s těmito úseky. To znamená, že pokud připojíme za chodu aplikace novou úsekovou jednotku, která začne odesílat data o obsazenosti z dalších úseků, jsou pro tyto úseky automaticky přidány nové signalizační diody. Vstupním argumentem metody je paket typu *OccupancySectionPacket* obsahující nové data o obsazenosti. Nově příchozí data jsou porovnány s již existujícími diodami a je vyhodnoceno, zda jsou úseky obsaženy mezi diodami či nikoliv. V případě nových úseků je potřeba vykreslit nové diody pomocí metody *AddSections* viz 4.4.3.9. Pokud však úseky už mají přiřazeny diody, dochází pouze k jejich překreslení v metodě *SetSections* viz 4.4.3.10.

4.4.3.9 Metoda *AddSections*

Metoda se vstupním argumentem ve tvaru paketu typu *OccupancySectionPacket* viz 4.4.1.7 a s žádnou návratovou hodnotou. Cílem metody je, přes cyklus for, přidat všechny úseky do panelu signalizačních diod. Diodám jsou nastaveny příslušné atributy získané ze vstupního argumentu.

4.4.3.10 Metoda *SetSections*

Zatímco metoda *AddSections* viz 4.4.3.9 vytváří nové diody pro signalizaci obsazenosti jednotlivých úseků, metoda *SetSections* jen upravuje jejich hodnoty. Nové data o obsazenosti úseků jsou načtena vstupním argumentem ve tvaru paketu typu *OccupancySectionPacket* viz 4.4.1.7. Samotné přiřazení hodnot probíhá jednoduchým prohledáním všech signalizačních dioda a porovnáváním jejich názvů s názvy nově příchozích úseků. Nastane-li shoda, hodnota diody je přepsána a cyklus začne prohledávat diody pro nalezení dalšího úseku.

4.4.3.11 Metoda *StopAll*

Dříve nebo později nastane v provozu kolejiště moment kdy je potřeba co nejrychleji zastavit všechny pohybující se vlaky. Ať už je tento požadavek vyvolán zmáčknutím příslušného tlačítka, či programově např. v rámci vypnutí celého klienta, je vždy zařízen vyvoláním metody *StopAll*. Metoda prohledá celý seznam lokomotiv viz 4.4.1.12 a každé jednotlivé lokomotivě pošle příkaz k zastavení pomocí metody *SendTCPData* viz 4.4.3.12.

4.4.3.12 Metoda *SendTCPData*

Přijetí a zpracování dat z TCP komunikace již bylo vysvětleno v popisu metod výše. Nyní je na ředě zmínit metodu, která naopak zařizuje odesílání dat k serveru. Pro tento účel je opět využita třída *TCPClient* a její metody, avšak samotné odeslání zprávy není jedinou úlohou metody *SendTCPData*. Tato metoda totiž také kontroluje, zda odeslání proběhlo

v pořádku či nikoliv, ať už z důvodu krátkodobého odpojení od serveru, nebo jiného problému. Pokud program detekuje některý z problémů, jsou uživateli nabídnuty možnosti *Opakovat* nebo *Zavřít*. Opakování pokusu znamená volání metody *StartTCPClient* pro znovupřipojení k serveru a následnému dalšímu pokusu o odeslání zprávy.

4.4.3.13 Metoda *SendTrainMotionPacket*

Další metodou asistující v činnosti odesílání dat přes sériovou linku je *SendTrainMotionPacket*. Metoda disponuje dvěma vstupními argumenty, a sice celočíselnou hodnotou *tag* a logickou hodnotou *stop*. Celočíselná hodnota odkazuje na jeden z otevřených ovládacích panelů viz 4.4.3.14, zatímco logická hodnota určuje charakter zprávy. Jednoduše řečeno, proměnná *stop* určuje, jestli se lokomotivě vybrané v panelu s pořadovým číslem *tag* odešle zpráva pro zastavení, nebo data nastavená v ovládacích prvcích.

4.4.3.14 Metoda *buttonAddLocomotive_Click*

Chceme-li lokomotivy ovládat, musíme nejdříve otevřít panel s ovládacími prvky viz základní informace pro obsluhu aplikace **??**debug˙rizeni˙v2. Každý panel obsahuje ovládání pouze pro jednu lokomotivu. Je-li třeba řídit více lokomotiv najednou, je možné přidávat a odebírat panely podle potřeby. Tlačítko *Add locomotive* slouží k zavolání metody *buttonAddLocomotive˙Click* a tím zařizuje ono zmíněné přidání panelu s ovládacími prvky. Tělo metody se skládá z části, kde je připravena plocha pro nový panel a části, kde jsou do nově vzniklé plochy přidány ovládací prvky. Připravení plochy spočívá ve vyhrazení nových míst v tabulkovém rozpoložení okna a zarovnání jejich velikostí na jednotnou hodnotu. Následuje sekvence vytváření jednotlivých ovládacích prvků s nastavenými požadovanými vlastnostmi a poté jejich vložení do předpřipravených prostor.

4.4.3.15 Metoda *buttonRemoveLocomotive_Click*

Pro přehlednost v panelech pro ovládání lokomotiv je možné nepoužívané panely odstranit. Blíže o významu přidávání a odebírání panelů viz 4.4.3.14. Dříve než se však může ovládací panel odstranit, je nutné poslat lokomotivě, jež byla řízena, zprávu o zastavení. Kdyby program tuto zprávu neposlal, hrozilo by, že lokomotiva bude neřízeně jezdit po kolejišti a uživatel by ji musel zastavit jiným způsobem. K zastavení použita metoda *SendTrainMotionPacket* viz 4.4.3.13. Na závěr je odstraněna plocha, ve které se ovládací prvky nacházely.

4.4.3.16 Metoda *comboBoxName_SelectedIndexChanged*

Prvním ovládacím prvkem v panelu pro řízení lokomotivy je rozbalovací lišta s názvem lokomotivy. Vybráním jedné z položek docílíme vyvolání metody s názvem *comboBoxName˙SelectedIndexChanged*. Obsluha vybrání nové lokomotivy je důležitá z několika důvodů. Zaprvé není žádoucí vybrání lokomotivy, která už je vybrána v jiném panelu. K této

kontrole a případné změně z nevhodně vybrané lokomotivy zpět na stav rozbalovací lišty nazvaný *Select locomotive*, slouží cyklus foreach na začátku této metody. Dále se program stará o případné zastavení lokomotivy, která byla ovládána na panelu před vybráním lokomotivy nové. Jelikož ale je tato metoda volána, až jako následek změny lokomotivy, nelze nijak zjistit, která lokomotiva byla nahrazena. Proto zavádíme globální proměnnou *comboBoxOldState* typu seznam textových řetězců, do níž jsou ukládány všechny lokomotivy vybrané v ovládacích panelech. Tudíž není problém, díky porovnání současně vybraných lokomotiv a lokomotiv uložených v tomto seznamu zjistit, jaká lokomotiva byla přepsána a následně ji poslat zprávu o jejím zastavení. Zároveň jsou, při vybrání položky z rozbalovací lišty, také nastaveny počáteční hodnoty ovládacích prvků.

4.4.3.17 Metoda *buttonStart_Click*

Obslužná metoda pro událost vyvolanou kliknutím myši na jedno z tlačítek *Start* uvnitř panelů pro ovládání lokomotiv. Nejen že tímto kliknutím uživatel odešle zprávu se současně nastaveními parametry jízdy, ale také umožní odesílání zpráv od zbytku ovládacích prvků. Přesněji řečeno, při tlačítku v poloze *Stop* bude jakákoliv změna rychlosti či směru okamžitě odeslána. Kdežto během doby, kdy je tlačítko v poloze *Start*, změny rychlosti ani směru odeslány nebudou až do chvíle, dokud není tlačítko opět zmáčknuto. K ovládání lokomotiv použita metoda *SendTrainMotionPacket* viz 4.4.3.13.

4.4.3.18 Metoda *checkBoxLight_CheckedChange*

Jediný ovládací prvek, který není závislý na stavu tlačítka *Start* viz 4.4.3.17, je zaškrtávací políčko umožňující rozsvícení nebo zhasnutí světel. Změna stavu tohoto políčka vyvolá metodu *checkBoxLight˙CheckedChange*, uvnitř které je nejprve provedena kontrola, zda je pro políčko vybrána příslušná lokomotiva a případně je odeslána zpráva o změně stavu světel.

4.4.3.19 Metoda *checkBoxReverse_CheckedChanged*

Událost zaškrtnutí políčka pro vybrání směru jízdy vyvolá metodu *checkBoxReverse˙CheckedChanged*. Metoda zkontroluje přítomnost lokomotivy v příslušné rozbalovací liště a v případě, že je některá z lokomotiv v této liště vybrána, zavolá metodu *SendTrainMotionPacket* viz 4.4.3.13 s příslušnými vstupními argumenty. V opačném případě je stav zaškrtávacího políčka nastaven na stav nezaškrtnuto a metoda je ukončena. To vše z důvodu odeslání nové řídící zprávy ve chvíli, kdy uživatel upraví směr vlaku.

4.4.3.20 Metoda *trackBar_Scroll*

Metoda vyvolaná změnou hodnoty na posuvné liště reprezentující rychlost .Stejně jako u metody *checkBoxReverse_CheckedChanged* viz. 4.4.3.19, je potřeba reagovat na uživatelský impulz v podobě změny rychlosti lokomotivy, odesláním nové řídící zprávy. Na rozdíl

od zmíněné metody však stačí ke správnému fungování pouze samotné zavolání metody *SendTrainMotionPacket* viz 4.4.3.13.

4.4.3.21 Metoda *buttonClose_Click*

Velice jednoduchá obslužná metoda je přiřazena k události kliknutí na tlačítko s názvem *Close*. Jedná se o prosté zavolání předpřipravené metody *Close*, který zavře formulář programu.

4.4.3.22 Metoda *buttonCentralStop_Click*

Stisk tlačítka s nápisem *Central stop* volá metodu *buttonCentralStop_Click*. Nejprve jsou všechny otevřené ovládací prvky v panelech přenastaveny do výchozí pozice a následně je odeslán příkaz k zastavení všech lokomotiv pomocí metody *StopAll* viz 4.4.3.11.

4.4.3.23 Metoda *Form1_FormClosed*

Při snaze ukončit program kterýmkoliv způsobem je vždy těsně před ukončením volána metoda 4.4.3.23. Uvnitř metody je odeslána zastavovací zpráva pro všechny lokomotivy pomocí metody *StopAll* viz 4.4.3.11, ale pouze v případě, že je program stále připojen k TCP serveru. Poté je uvolněn a ukončen klient a následně také celý program.

4.4.3.24 Metody pro řízení úsekových jednotek

Následují čtyři metody mají téměř totožný obsah, kde každá jednotlivá metoda náleží jednomu příslušně pojmenovanému tlačítku. Jedná se o metody *buttonSupllyOn˙Click*, *buttonSupllyOff˙Click*, *buttonResetBridge˙Click* a *buttonResetMikro˙Click*. Uvnitř metody je vždy vytvořen paket typu *UnitInstructionPacket* viz 4.4.1.5. Vstupními argumenty konstruktoru paketu jsou druh zprávy podle stisknutého tlačítka a číslo jednotky vybrané v liště celočíselných hodnot nadepsaných *Unit*. Následně je vytvořená zpráva odeslaná pomocí metody *SendTCPData* viz 4.4.3.12.

4.4.3.25 Metoda *numericUpDelay_ValueChanged*

Poslední metoda zařizuje reakci na změnu obsahu lišty celočíselných hodnot s názvem *Delay of sending currents*. Číslo uvedené v této liště představuje násobitel výchozí doby mezi odesíláním zpráv o obsazenosti jednotek. Tělo metody a princip fungování je naprosto shodný s metodami pro řízení úsekových jednotek viz 4.4.3.24.

4.4.4 Program *Timetable Control Train TT*

Posledním z rodiny programů tvořící aplikaci verze dva je program určený k řízení kolejiště jízdním řádem. Celou řadou konstrukcí a metod je totožný s programem *Visual Debug Control Train TT* viz 4.4.3, proto se zde budu často odkazovat na popis výše zmíněného

programu. Také i zde se jedná o formulářovou neboli okenní aplikaci, kde základní kostra programu nebude blíže popisována, neboť je automaticky vytvořená vývojovým prostředím. V sneposlední řadě nebudu při popisu upozorňovat na bezpečné přetypování senderu na určitý prvek, jak tomu bylo už u popisu programu *Visual Debug Control Train TT*. Přehled správného používání programu viz 4.2.2. Konstruktor třídy opět obsahuje pouze předdefinovanou inicializaci komponent.

4.4.4.1 Třídy *DataForTimetable* a *NoteInTimetable*

Pro jednoduchost práce se záznamy z jízdního řádu jsou informace uloženy do dvou seznamů. Jedním je seznam proměnných typu *NoteInTimetable* nesoucí název *timetable*. Druhý seznam, složen z proměnných typu *DataForTimetable*, je pojmenovaný *dataForTimetable*. Typ *DataForTimetable* reprezentuje inicializační řádek z hlavičky jízdního řádu, zatímco *NoteInTimetable* představuje záznam s konkrétním spojem v jízdním řádu. Oba tyto typy resp. třídy pracují na shodném principu a jsou vytvořeny speciálně pro řízení jízdním řádem. Obsahují řadu vlastností a dva konstruktory. Jeden pro načtení vlastností z textového řádku a druhý pro načtení přímo zadaných jednotlivých vlastností. V konstruktoru pro načítání dat z textového řádku je nejdříve provedeno rozdělení oddělovačem v podobě středníku a následně jsou jednotlivá data naparsována resp. trimována a uložena do příslušných vlastností.

1. *NoteInTimetable*

   Vlastnosti:

   - *Type*: Pracovní název vlaku typu textový řetězec (musí se shodovat s jedním z uvedených názvů v hlavičce jízdního řádu).
   - *StartStation*: Vlastnost typu *Section* viz 4.4.1.9 nesoucí stanici, ze které bude vlak v daném spoji vyjíždět.
   - *FinalStation*: Vlastnost typu *Section* viz 4.4.1.9 nesoucí stanici, do které má vlak v daném spoji dojet.
   - *Departure*: Vlastnost typu *DateTime* obsahující časový údaj spoje.

2. *DataForTimetable*

   Vlastnosti:

   - *Locomotive*: Vlastnost typu *Locomotive* nesoucí informaci o lokomotivě, která plní spoje se stejnojmenným názvem. Název uložen ve vlastnosti *Type*.
   - *Type*: Pracovní název vlaku shodný s názvy vyskytujícími se v jednotlivých časových záznamech jízdního řádu. Právě podle shody vlastnosti *Type* v datech jízdního řádu se stejnojmennou vlastností u časového záznamu indikuje, jaká lokomotiva plní tyto spoje.

- *Station1/2*: Stanice mezi kterými je vlakové spojení realizováno. Vlastnosti jsou typu *Section* viz 4.4.1.9. Hodnota odpovídá jedné položce ze seznamu úseků viz 4.4.1.13.
- *Speed*: Desetinná hodnota reprezentující rychlost spojů dané lokomotivy. Vlastnost může nabývat hodnot nula až jedna a představuje číslo jímž je přenásobena maximální rychlost lokomotivy.
- *Reverse1/2*: Logické hodnoty udávající, jestli na dané trase pojede vlak přímo nebo pozpátku.
- *WaitTime1/2*: Celočíselná bez-znaménková hodnota reprezentující dojezdový čas. Tedy čas jak dlouho má lokomotiva pokračovat v jízdě, potom co je zaregistrována cílovým úsekem.
- *Line*: Vlastnost typu textový řetězec obsahující celý řádek nahraný z jízdního řádu.

4.4.4.2 Metoda *Form1_Load*

Spuštění programu zahajuje metoda *Form1˙Load* zajišťující přípravu mechanizmů pro jejich správné použití. Nejdříve přiřadí globálnímu časovači s názvem *timer* obslužnou metodu *timer˙Tick* viz 4.4.4.5. Globální časovač je nastaven na 1 s a zajišťuje pravidelné nahlížení do jízdního řadu, tak aby mohl program reagovat na na jízdní řád a posílat zprávy určené k pohybu lokomotiv. Dále je volána metoda *TimetableEnabled* se vstupním parametrem false. Význam metody viz 4.4.4.3. Následují úpravy tabulky nesoucí jízdní řád. Konkrétně změna fontu buněk a nastavení formátu výpisu času pro sloupec v tabulce nazvaný *Departure* neboli odjezd. Tabulce je také přidělen zdroj dat, a sice globální seznam s názvem *timetable*, který je v době přidělení prázdný, ale uživatel do něj bude moci později načíst data v podobě nahrání souboru s jízdním řádem. Je-li vše provedeno bez komplikací, program se bude pokoušet připojit k TCP serveru do té doby, dokud se to nepovede. Posledním krokem který je potřeba udělat před zobrazení okna uživateli je kontrola, která určí jestli program spustil uživatel, nebo spouštěč viz 4.4.4. V případě, kde byl program spuštěn spouštěčem, je nutné zavolat načtení jízdního řádu pomocí metody *LoadTimeTable* viz 4.4.4.10 se vstupním parametrem cesty a názvu jízdního řádu. Informaci o cestě a názvu jízdního řádu, který má být spuštěn, dostává program od spouštěče prostřednictvím pole obsahujícím argumenty příkazového řádku. Kontrola je tedy velmi jednoduchá, neboť pole obsahující cestu k žádnému jízdnímu řádu je jistě zavolané spouštěčem a nikoliv uživatelem.

4.4.4.3 Metoda *TimetableEnabled*

Tuto metodu lze vnímat jako takový přepínač mezi stavem, kdy je načtený jízdní řád a stavem, kdy jízdní řád načten není. Metoda nemá návratovou hodnotu, ale obsahuje jeden vstupní parametr s logickou hodnotou. Logická hodnota určuje, zda se má program

překonfigurovat do stavu, kdy je nahrán jízdní řád, nebo naopak kdy nahrán není. Pro oba stavy je program nastaven lehce odlišně. Konkrétně ve stavu nahraného jízdního řádu je zakázáno tlačítko *Load* a je zapnut časovač *timer*. Na rozdíl od stavu, kdy jízdní řád nahrán není, kde jsou zase zakázány tlačítka *Pause*, *Change train data* a časovač *timer* stojí. To však není všechno co metoda dělá.

V případě, že je metoda volána se vstupním argumentem true, což symbolizuje nastavení programu s načteným jízdním řádem, je také provedeno seřazení vlaků, načasování hodin a zapnutí vizualizace analogových hodin. Seřazení vlaků je provedeno pomocí metody *Prepareposition* v jejímž popisu viz 4.4.4.4 je také vysvětlen smysl tohoto seřazení. Načasováním hodin je myšleno pouhé zastavení programu do okamžiku kdy se současný čas nerovná celé vteřině, tj. nula milisekund a následně je okamžitě spuštěn časovač *timer*. Časovač má totiž nastavený interval tikání na 1000 ms, to znamená, že pouze jednou za vteřinu se prohledává jízdní řád. Pochopitelně v zájmu uživatele je, aby byl jízdní řád prohledán co nejdříve v dané vteřině a nenastávalo tak nechtěné zpoždění.

Pokud je metoda volána se vstupním parametrem false, je tím vyjádřena snaha o uvedení programu do stavu bez nahraného jízdního řádu. K tomuto účelu je kromě výše uvedených opatření také nutné vyčistit globální seznam *timetable* od starého jízdního řádu, přepsat tabulku kde se jízdní řád nachází vyobrazen a také pro jistotu zastavit všechny lokomotivy. Zastavení všech lokomotiv proběhne pouze pokud je program připojen na TCP server. K zastavení je využita metoda *StopAll* naprosto totožná se stejnojmennou metodou v programu *Visual Debug Control Train TT* viz 4.4.3.11.

4.4.4.4 Metoda *PreparePosition*

Metoda nazvaná *PreparePosition* neboli příprava do pozic zajišťuje srovnání vlaků do počátečních pozic poté co je nahrán jízdní řád. V jízdním řádu jsou uvedeny startovní i cílové stanice spoje, tudíž je nutné zajistit, aby lokomotivy startovali z příslušných stanic a ne z míst kde se zrovna nachází. Počáteční pozice musí tudíž program najít na základně současného času a záznamů v jízdním řádu. Algoritmus začíná prohledáváním časových záznamů v jízdním řádu zvlášť pro každou lokomotivu uvedenou v datech pro jízdní řád, tj. data v hlavičce jízdního řádu. Ve chvíli, kdy algoritmus nalezne první záznam, který by se měl vykonávat, získá z něho informaci odkud by měla lokomotiva vyjíždět a právě na tento úsek lokomotivu posílá. To má za následek, že až uplyne čas a přijde na řadu zpracování prvního záznamu v jízdním řádu, bude lokomotiva stát na pozici uvedené jako startovní. Jediný případ, kdy se tak nestane, bude za předpokladu, že lokomotiva dostane příkaz k vykonání záznamu v jízdním řádu dřív, než stihne dojet na počáteční místo. K odeslání zprávy o dojetí na určitý úsek je použita metoda *SendTrainInstructionPacket* viz 4.4.4.6. Před zavoláním odesílací metody je ještě nutné vytvořit nový záznam v jízdním řádu, tj. typ *NoteInTimetable* viz **??**. Nový záznam je totožný se záznamem nalezeným pomocí algoritmu, pouze s prohozenou cílovou a počáteční stanicí. Následně je záznam předán výše zmíněné metodě pro odeslání zprávy a tím je příprava do počáteční pozice

pro danou lokomotivu u konce.

4.4.4.5 Metoda *timer_Tick*

Popis metody *Form1˙Load* viz 4.4.4.2 již naznačil, že globální časovač *timer*, potažmo metoda volaná jako důsledek tikání časovače jsou pro běh tohoto programu zcela stěžejní. Předpokládejme úspěšně zapnutý program, připojený k serveru, s načteným jízdním řádem. Již víme, že v tomto stavu, je poleven časovač *timer*, a tak dochází k nepřetržitému volání metody *timer_Tick*. Tělo této metody je na první pohled velmi podobné metodě *PreparePosition* viz 4.4.4.4. Také je zde procházen záznam po záznamu celý seznam *timetable*, ale nyní nehledáme záznamy, které se vykonávat teprve budou, ale záznamy které se mají vykonat právně v současný okamžik. Najde-li se záznam který by se měl v daný moment odeslat, přichází na řadu nalezení odpovídající lokomotivy a posléze odeslání zprávy opět pomocí metody *SendTrainInstructionPacket* viz 4.4.4.6.Nakonec je zde překreslena tabulka vykreslující jízdní řád, tak aby byli na obrazovce uživatele zobrazeny vždy aktuální spoje.

4.4.4.6 Metoda *SendTrainInstructionPacket*

Pro odesílání zpráv na základě vyhodnocení záznamů v jízdním řádu je zde speciální metoda s názvem *SendTrainInstructionPacket*. Vstupními argumenty metody jsou jednak proměnná typu *NoteInTimetable* obsahující informace o samotném, ale také proměnná typu textitDataForTimetable s obsahem dat pro uskutečnění spoje. Uvnitř metody jsou nejprve deklarovány všechny proměnné, nutné k poskládání zprávy typu *TrainMotionInstructionPacket* viz 4.4.1.4. A následně jsou naplněny z informací obsažených ve vstupním argumentu typu textitDataForTimetable. Druhý vstupní argument slouží v podstatě pouze k informaci, do které ze stanic má vlak dojet. Zpráva složená z jednotlivých informací je následně předána k odeslání metodě *SendTCPData* viz 4.4.4.8.

4.4.4.7 Metoda *DataGridViewInvalidate*

Aby dostával uživatel aktuální informace o vykonávaných spojích, musí se tabulka zobrazující tyto spoje pravidelně překreslovat. Překreslení není úplně přesný termín, protože je v zásadě pouze posouván náhled na dané spoje v tabulce a samotná tabulka zůstává stále stejná. Metoda přebírá vstupní argument v podobě čísla odpovídajícího pořadovému číslu v tabulce, na které má být náhled.

4.4.4.8 Metoda *SendTCPData*

Tato metoda nenese stejný obsah jako stejnojmenná metoda v programu *Visual Debug Control Train TT*, i přes fakt, že pro program plní prakticky totožnou práci. Metoda přebírá textovou zprávu která by měla být odeslána jako vstupní argument. Toto odeslání se pokusí vykonat, pokud je globální proměnná *IsConnect* naplněná hodnotou true. Pokud

odeslání proběhlo v pořádku, metoda je ukončena stejně jako tomu bylo u programu *Visual Debug Control Train TT* viz 4.4.3.12. Rozdíl přichází ve chvíli kdy v odeslání nastal problém. Není žádoucí, aby nám zde vyskakovalo okno vyžadující reakci uživatele, protože lze předpokládat, že uživatel při tomto druhu řízení vůbec nebude přítomen u PC. Proto je nastavena varianta, kde se program sám od sebe snaží o připojení k serveru po dobu 15 vteřin. Není-li ani po této době klient připojen, dochází k ukončení programu. Za předpokladu, že klient se odpojil pouze krátkodobě a je tak ve stanovené lhůtě k serveru opět připojen, je volána metoda *PreparePosition* viz 4.4.4.4 pro srovnání případných promeškaných spojů.

4.4.4.9 Metoda *buttonLoadTimetable_Click*

Tlačítko *Load* slouží k nahrání a zprovoznění jízdního řádu. Tomuto tlačítku je přidělena obslužná metoda *buttonLoadTimetable_Click*, která ve svém těle pouze volá metodu *LoadTimeTable* viz 4.4.4.10 se vstupním argumentem prázdného textového řetězce.

4.4.4.10 Metoda *LoadTimeTable*

Hlavním účelem metody *LoadTimeTable* je naplnit globální seznamy *timetable* a *dataForTimetable* daty z požadovaného jízdního řádu. V programu jsou dvě varianty jak jízdní řád do seznamů načíst. Metoda totiž vlastní vstupní argument v podobě textového řetězce, který je buď prázdný, nebo nese cestu a název souboru s jízdním řádem. Jeli obsah vstupní argumentu prázdný, otevírá se uživateli dialogové okno pro výběr souboru typu csv(comma-separated values). Varianta, kde je cesta a název souboru načten přímo ze vstupního argumentu, je určena především pro spouštěč viz 4.4. Ve chvíli, kdy je vybrána cesta a název souboru, metoda přechází do druhé části, kde jsou informace ze souboru získávány a načítány do příslušných seznamů. Dojde-li k úspěšnému nahrání informací, je potřeba provést jejich zobrazení uživateli do tabulky spojů. Pro lepší vizuální zážitek je do tabulky místo seznamu *timetable* nahrán seznam s názvem *onScreen*. Tento seznam je lokální kopií seznamu *timetable*, pouze s rozdílem, že v názvech stanic jsou použity mezery místo podtržítek. Na závěr je zavolána metoda *TimetableEnabled* viz 4.4.4.3 se vstupním argumentem true.

4.4.4.11 Metoda *buttonPause_Click*

Obslužná metoda volaná při stisknutí tlačítka *Pause*. Stisknutím tlačítka uživatel vyvolá pauzu, která trvá do doby, dokud není tlačítko znovu zmáčknuto. Princip pozastavení je velice snadný, a sice tlačítko vyvolá zapnutí resp. vypnutí časovače *timer*. Pokud je časovač vypínán, je zároveň odeslána zpráva o zastavení všech vlaků pomocí metody *StopAll* viz 4.4.3.11. V případě, že se tlačítkem snažíme pauzu vypnout, tudíž je časovač zapínán, je navíc volána metoda *PreparePosition* viz 4.4.4.4.

4.4.4.12 Metoda *buttonCentralStop_Click*

Tlačítko centrálního zastavení nazvané *Central stop*, kromě zastavení všech lokomotiv, také přerušuje chod celého jízdního řadu. Dosahuje toho zavoláním metody *TimetableEnabled* viz 4.4.4.3 se vstupním argumentem false.

4.4.4.13 Metoda *buttonClock˙Click*

Vizualizace analogových hodin není nezbytná, ale zlepšuje vizuální stránku celého programu. Navíc, pokud je jízdní řád strukturován tak, že odjezdy neprobíhají pouze v celých minutách, je žádoucí, aby uživatel mohl sledovat čas i na úrovni vteřin. Jelikož tlačítko *Clock* slouží k zobrazení, ale také skrytí hodin, je nejprve zjištěno, která z těchto akcí bude potřeba vykonat. Následuje přidání nebo naopak odebrání hodin z okna programu. Metoda je zakončena srovnáním prvků v okně, tak aby okno působilo symetricky.

4.4.4.14 Metoda *buttonFullScreen_Click*

Tlačítko nazvané *Full screen* umožňuje zvětšené zobrazení okna, ale také textu, tak aby byli informace čitelné z větší dálky. K tomu účelu se v těle metody nachází sekvence příkazů upravující vzhled, např. maximalizování okna, zvětšení fontu, nebo vypnutí vizualizace analogových hodin. V případě, že se program nachází v režimu full screen a uživatel znovu stiskne příslušné tlačítko, provádí se stejné změny pouze v inverzním smyslu.

4.4.4.15 Metoda *buttonChangeTrainData_Click*

Funkčnost tlačítka s názvem *Change train data* je podrobněji popsána v kapitole 4.2.2.1. Pro takto fungující nastavovací okno je nejprve nutné vytvořit samotný formulář typu *ChangeTrainData* a následně do něj nahrát seznamy *timetable* a *dataForTimetable*. Předáván je také název souboru s jízdním řádem z globální proměnné *fileName*. Uvnitř formuláře viz 4.4.4.16 uživatel nastaví požadovaná data a stisknutím tlačítka *OK* svou volbu potvrdí. Na to metoda reaguje uložením nových dat do výše zmíněných seznamů a překreslení tabulky spojů.

4.4.4.16 Třída *ChangeTrainData*

Princip fungování změny dat za běhu programu *Timetable Control Train TT* je blíže vysvětlen viz 4.2.2.1. Ke správnému fungování potřebuje třída tři globální proměnné. Seznam s obsahem spojů z jízdního řádu, seznam s obsahem dat pro jízdní řád a název souboru jízdního řádu, jež má být upraven. Všechny tyto proměnné jsou nahrány při volání třídy viz 4.4.4.15. Konstruktor třídy obsahuje pouze inicializaci komponent.

Metody:

- *ChangeTrainData_Load*:

Metoda volaná před načtením okna. Jelikož je počet nastavovacích prvků závislý na počtu lokomotiv, které vykonávají spoje, je okno vizuálně velmi proměnlivé. Uvnitř metody je tedy pro každou lokomotivu přidán jeden ovládací řádek. Zpočátku je vyhrazen prostor pro řádky a následně je zaplněn ovládacími prvky s původními hodnotami získanými ze seznamu *dataForTimetable*. Teprve poté má uživatel možnost měnit měnit příslušná data.

- *comboBox_SelectedIndexChanged*:

  Tato metoda hlídá uživatele, aby nebyla zadána jedna lokomotiva pro dvě různá spojení. Při vybrání již obsazené lokomotivy je vybraná lokomotiva jednoduše změněna na jinou, zatím nevybranou lokomotivu.

- *textBox_TextChanged*:

  Podobně jako metoda *comboBox_SelectedIndexChanged*, tak i tato metoda slouží jako kontrola uživatele. Tentokrát však nehlídá vybranou lokomotivu, ale vybraný pracovní název vlaku. V okamžik, kdy uživatel zadá pracovní název který byl již použit u jiného spoje, je automaticky doplněn o velké písmeno "X".

- *buttonOK_Click*:

  Stiskem tlačítka *OK* uživatel dává najevo, že chce změněná data uložit. K tomuto záměru je využita sekvence přepisování dat nejprve v seznamu *timetable* a poté také v seznamu *dataForTimetable*. Pokud je zaškrtnuto políčko popsané textem *Change data also in file*, je nakonec ještě zavolána metoda *ChangeDataInFile*, která se postará o přepsání dat přímo v souboru s jízdním řádem.

- *ChangeDataInFile*:

  Přepsání dat ve zdrojovém souboru s jízdním řádem probíhá, nejdříve vytvořením pole typu textového řetězce o velikosti počtu řádků v původním jízdním řádu. Následně jsou do pole nahrány všechny data ze nově vzniklých seznamů *timetable* a *dataForTimetable*. Na závěr je pole nahráno do původního souboru a je tak dokončeno přepsání dat.

4.4.4.17 Metody společné s *Visual Debug Control Train TT*

Řada tříd a mechanizmů jsou pro programy *Visual Debug Control Train TT* a *Timetable Control Train TT* zcela shodné, tudíž jejich opětovné popsání nedává smysl. Společnými metodami jsou *StopAll* viz 4.4.3.11, *StartTCPClient* viz 4.4.3.2, textitKlientConnected viz 4.4.3.4, *KlientCleanUp* viz 4.4.3.3, *TCPDisconnectClient* viz 4.4.3.5, *Form1_FormClosed* viz 4.4.3.23 a *buttonClose_Click* viz 4.4.3.21.

5

Závěr

Konečný výsledek snahy o vývoj komplexní a dobře fungující aplikace je detailně popsán v kapitole 4. Řeší spoustu nedostatků a problému, které sužovali první verzi aplikace viz 3.1.4, ale některé z komplikací přetrvávají. Jedná se převážně o problémy nevyřešitelné, nebo jen z části řešitelné na úrovni aplikace.

Např. detekce obsazenosti úseků sice funguje dobře, ale aplikace nedostává od řídících jednotek informaci, kde se nachází daná lokomotiva, pouze informaci, že úsek je některou z lokomotiv obsazen. To považuji za nejhrubší nedostatek celého systému. Aplikace by sice mohla obsahovat algoritmus, který by při spuštění aplikace, zkoušel posílat popořadě všem lokomotivám zprávu k pohybu a ze sledování obsazenosti by se tak dalo vyhodnotit, na kterém úseku se daná lokomotiva nachází. Ale tento systém by zřejmě přinesl víc komplikací, než užitku.

Velké omezení v provozu kolejiště zapříčinila absence jednotky, pro řízení výhybek. Z toho důvodu, je zatím kolejiště, pouze struktura třech pevně nastavených okruhů. S ohledem na tento fakt, by bylo nesmyslné, snažit se vyvíjet algoritmus na ochranu před konflikty. Tato oblast problematiky bude aktuální v době dokončení a nainstalování alespoň jedné jednotky pro řízení výhybek.

První dva výše zmíněné problémy, by nepůsobili tak markantně, kdyby kolejiště disponovalo větším počtem úsekových jednotek. Ale v současném stavu, kde je k dispozici pouze jediná úseková jednotka, není prostor na žádné složité konstrukce.

Navíc, čas od času dojde ke spadnutí řídících jednotek, většinou na podmět zkratování kolejiště lokomotivou projíždějící přes výhybky. Zde by se nenacházel žádný závratný problém, kdyby však v řídících jednotkách fungoval restart pomocí zprávy z PC. Takhle je ale uživatel nucen provést restart pomocí fyzického tlačítka a je tím narušen autonomní průběh řízení.

Napříč všem problémům v tomto projektu, je potřeba dívat se na odvedenou práci kladně. Po dlouhé době je provoz kolejiště alespoň v nějakém chodu a může tak těšit náhodné kolemjdoucí i všechny, kteří se na projektu podíleli. Body zadání této práce tedy byli úspěšně splněny a lze říci, že v určitých oblastech výsledná aplikace dokonce převyšuje rámec zadání.

Literatura

[1] Weissar P., Žahour J., Lufinka O.:*Projekty FEL, Vlaky TT*[online]. Poslední změna 9. 10. 2018 v 15:10 [Cit. 9. 3. 2019]. Dostupné z: http://projekty.fel.zcu.cz/index.php/Vlaky˙TT

[2] Zvonař F.:*Generátor DCC signálu pro modelovou železnici.* 2018. Dostupné z: http://projekty.fel.zcu.cz/images/f/f8/BP-zvonar.pdf

[3] Malena O.:*Řídící jednotka kolejových úseků pro modelovou železnici.* 2018. Dostupné z: http://projekty.fel.zcu.cz/images/d/df/BP˙Malena˙2018˙DCC˙Repeater.pdf

[4] Zavavov:*DCC pro laiky a začátečníky*[online]. Poslední změna 4. 8. 2016 [Cit. 4. 6. 2019]. Dostupné z: http://www.zavavov.cz/cz/modelova-zeleznice/elektronika/89-dcc-pro-laiky-a-zacatecniky/

[5] Fulda:*Digitální řízení modelové železnice – DCC*[online]. Poslední změna 28. 3. 2016 [Cit. 4. 6. 2019]. Dostupné z: http://robodoupe.cz/2016/digitalni-rizeni-modelove-zeleznice-dcc/

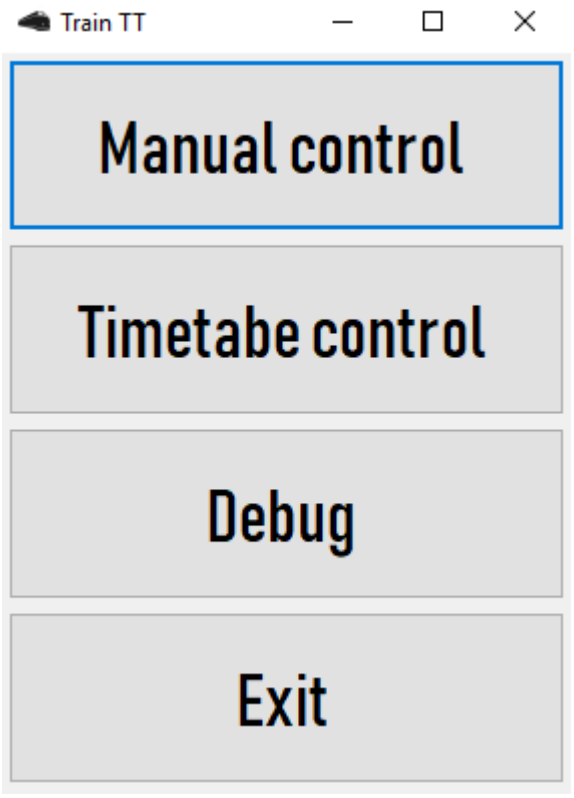
[6] Fulda:*Digitální řízení modelové železnice – DCC, 2. část*[online]. Poslední změna 30. 3. 2016 [Cit. 4. 6. 2019]. Dostupné z: http://robodoupe.cz/2016/digitalni-rizeni-modelove-zeleznice-dcc-2-cast/

[7] DCCWiki:*Digital packet*[online]. Poslední změna 23. 2. 2019 v 18:17 [Cit. 4. 6. 2019].https://dccwiki.com/Digital˙packet

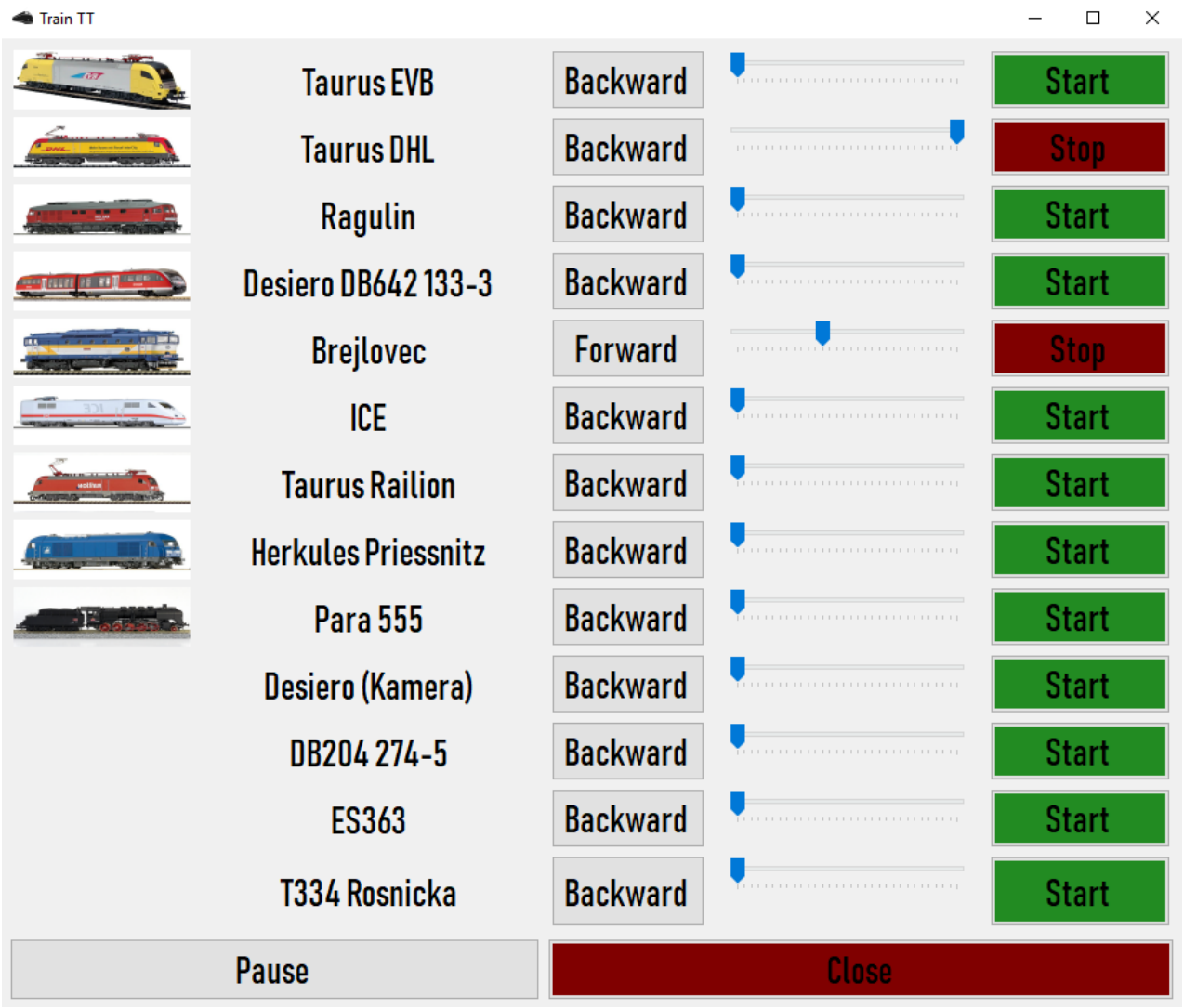
[8] NMRA:*Standards and Recommended Practices. National Model Railroad Association*[online]. Poslední změna 7. 1. 2014 [Cit. 4. 6. 2019].https://www.nmra.org/index-nmra-standards-and-recommended-practices

Příloha A

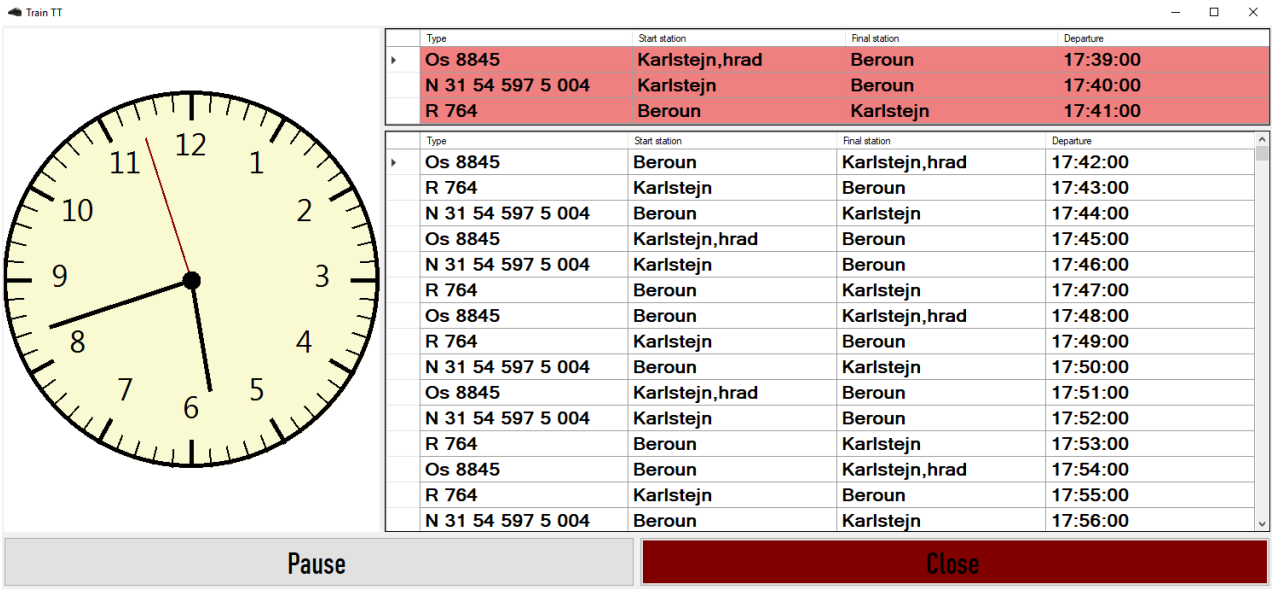
Aplikace



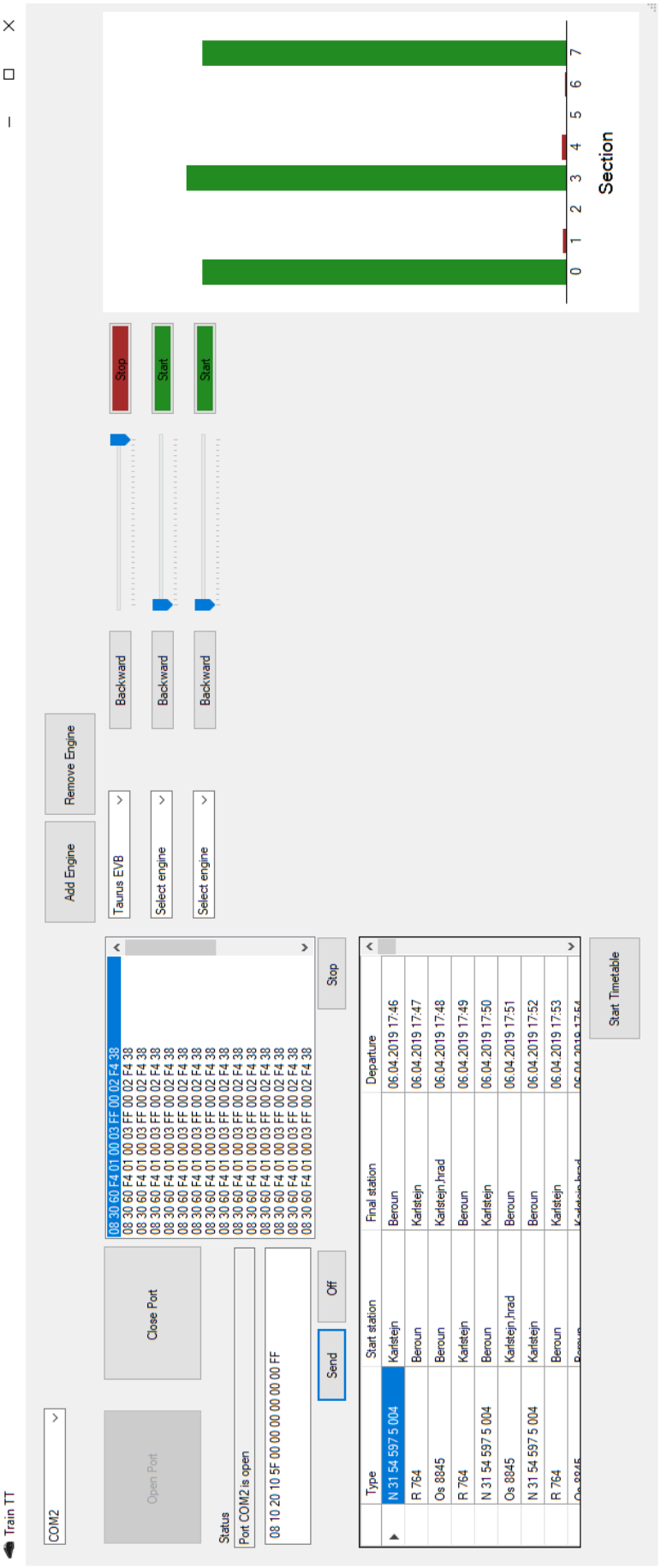
Obr. A.1: Hlavní menu aplikace



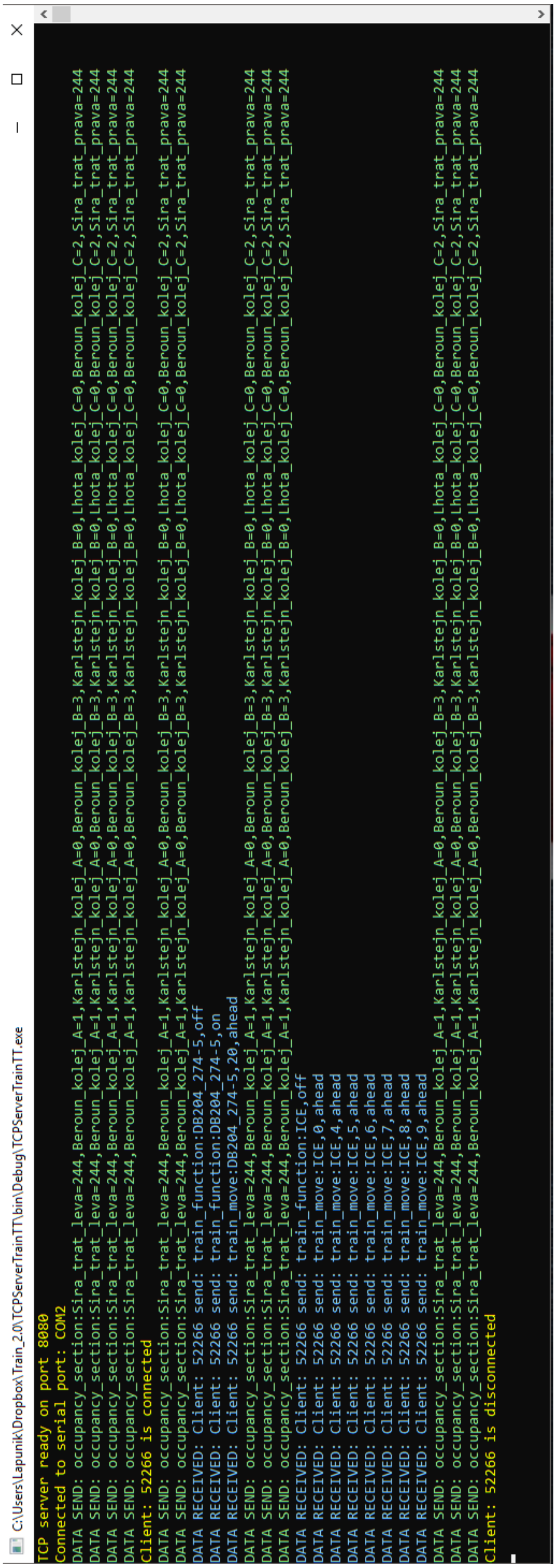
Obr. A.2: Okno aplikace pro manuální řízení první verze aplikace.



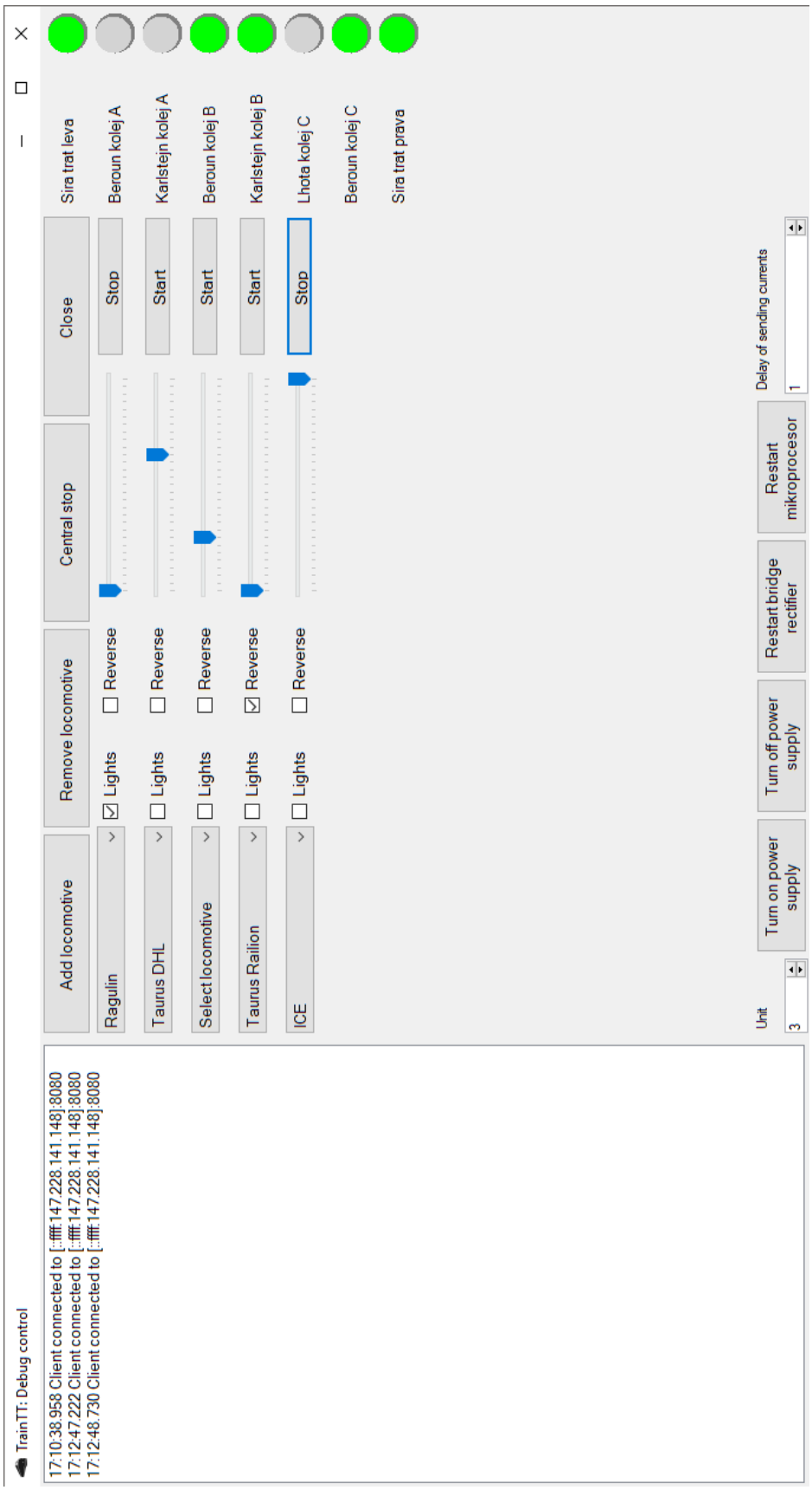
Obr. A.3: Okno aplikace pro řízení jízdním řádem první verze aplikace.



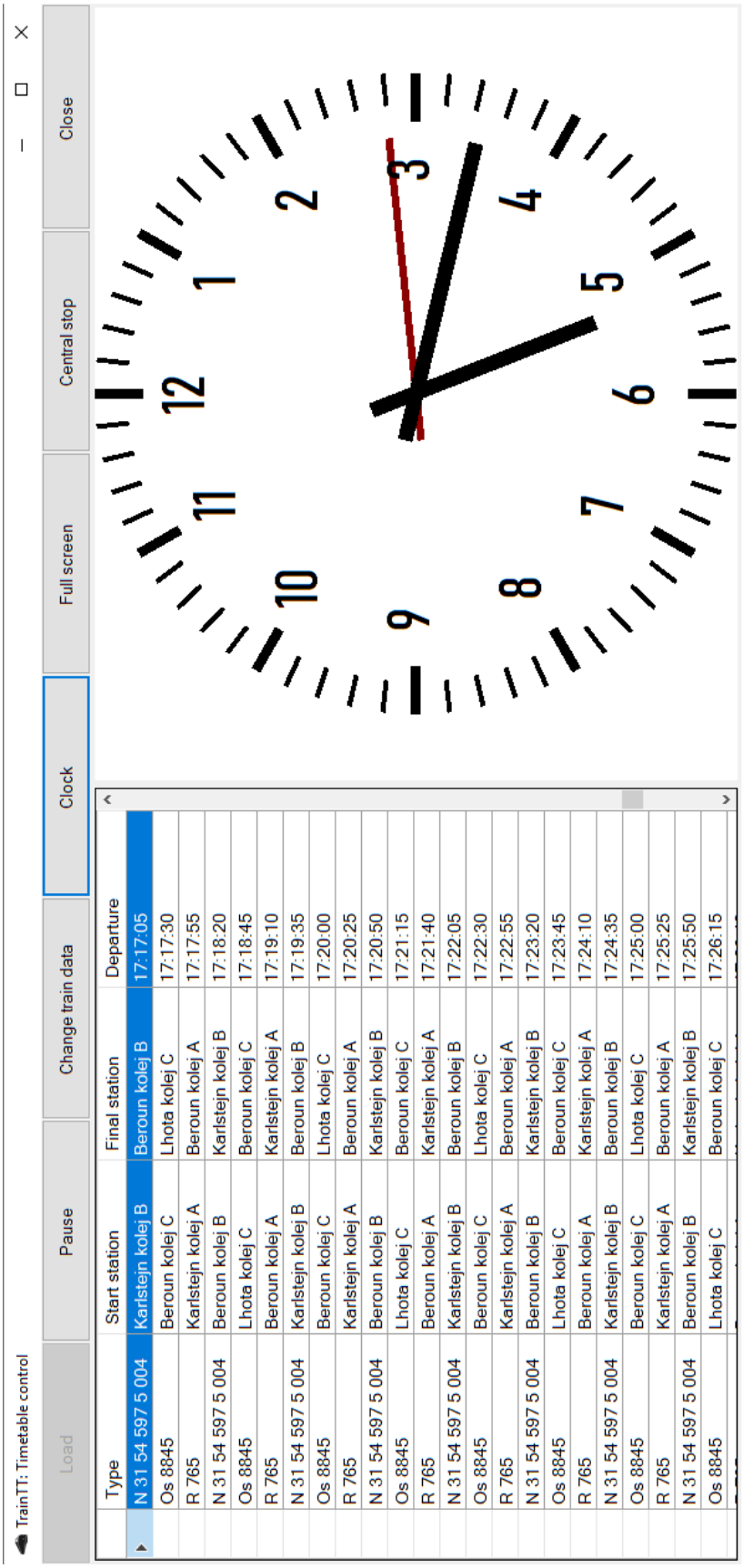
Obr. A.4: Okno aplikace pro odladovací řízení první verze aplikace.



Obr. A.5: Konzole zobrazující běh programu *TCP Server Train TT*



Obr. A.6: Program pro odlaďovací řízení druhé verze aplikace



Obr. A.7: Program pro řízení jízdním řádem druhé verze aplikace