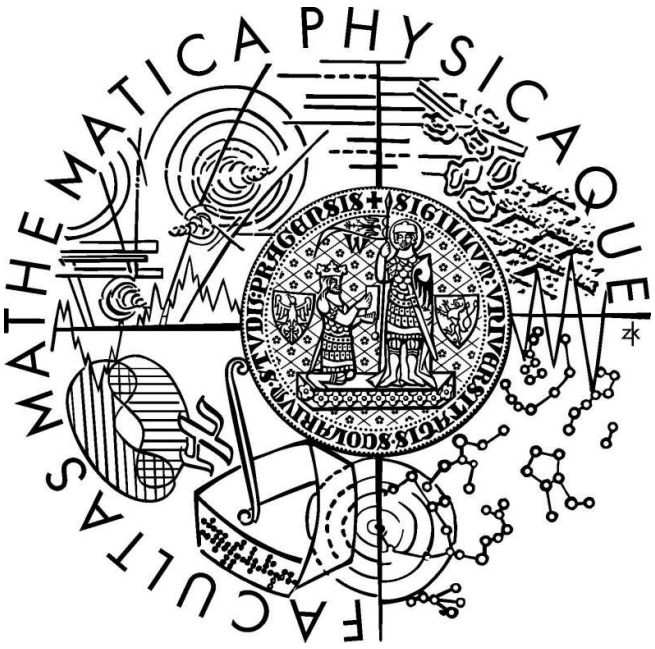
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Petr Lasák

Monitoring procesních dat Národní Vlakové Linky

Katedra softwarového inženýrství

Vedoucí práce: RNDr. Michal Žemlička, Ph.D.
Studijní program: Informatika, softwarové systémy

2010

Na tomto místě bych chtěl poděkovat především RNDr. Michalu Žemličkovi, Ph.D. za téma a vedení diplomové práce, cenné rady a připomínky hlavně při zpracování dokumentace.

Dále bych chtěl poděkovat Ing. Stanislavu Markovi a spol. za poskytnutí hardwaru, testovacího prostředí a dokumentace k Národní Vlakové Lince.

V neposlední řadě bych chtěl poděkovat Bc. Milanu Burdovi za pevné nervy při mých všetečných dotazech ohledně frameworku Qt 4 a Evě Lasákové za jazykovou korekturu celé práce.

Bez Vás by tato práce nevznikla.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Ostravě dne 3. srpna 2010 Petr Lasák

OBSAH:

Název práce: Monitoring procesních dat Národní vlakové linky

Autor: Bc. Petr Lasák

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Michal Žemlička, Ph.D.

e-mail vedoucího: zemlicka@ksi.mff.cuni.cz

Abstrakt: Jedním z protokolů používaných pro komunikaci mezi vozy ve vlacích, je Národní vlaková linka (NVL). Výsledkem této práce je podpora diagnostiky NVL - zejména sledování toku procesních dat a přehledné monitorování vybraných hodnot distribuovaných po celém vlaku, zohledňující jejich charakter a platnost.

Klíčová slova: vlaková sběrnice, procesní data, Národní Vlaková Linka

Title: Monitoring of process data of National Train Bus

Author: Bc. Petr Lasák

Department: Department of Software Engineering

Supervisor: RNDr. Michal Žemlička, Ph.D.

Supervisor’s e-mail address: zemlicka@ksi.mff.cuni.cz

Abstract: One of the protocols used for communication between coaches inside a train is the National Train Bus (NTB). The result of this work is support for diagnostics on the NTB – mainly watching the process data-flow and monitoring of chosen variables distributed along the train, with consideration to their character and validity.

Keywords: train bus, process data, National Train Bus

1 Úvod

Ve světě existuje řada výrobců a provozovatelů kolejových vozidel, které slouží různým účelům. Tato vozidla se většinou spojují do souprav, kde mezi sebou komunikují pomocí elektrických signálů. Cílem této práce je vytvořit prototyp aplikace, která bude moci, díky připojení ke speciálnímu zařízení, zobrazit tuto komunikaci.

1.1 Pozadí projektu

Kolejové soupravy, ať už vlakové, tramvajové nebo metra, obsahují velké množství zařízení, která potřebují elektronicky komunikovat se svým okolím. Tato komunikace probíhá zpravidla pomocí datové sběrnice, která je zabudovaná uvnitř jednotlivých vozů. Takových sběrnic bývá v soupravě několik a jsou rozděleny dle svého dosahu a účelu.

Aby bylo možné propojovat vozy různých typů a od různých výrobců, existuje řada mezinárodních standardů popisujících způsob, formát a obsah posílaných dat.

1.2 Národní Vlaková Linka

Jednou z těchto sběrnic je i *Národní Vlaková Linka* (dále jen NVL), která je poměrně mladá (její návrh byl podán na schválení v roce 2009) a proto se zde otevírá velké pole působnosti, kde lze vytvořit řadu nových a užitečných nástrojů pro práci s ní.

Díky této situaci a také především kvůli dostupnosti hardwaru pro komunikaci a testování byla vybrána jako cílová sběrnice prototypu. Vzhledem k existenci i jiných vlakových sběrnic, musel návrh uvažovat s budoucí kompatibilitou s těmito sběrnicemi.

1.3 Hlavní vize projektu

Graphics Diagnostics System, neboli GDS, je samostatná aplikace, která má uživatelům umožnit pohodlné zobrazení datové komunikace mezi zařízeními, která spolu komunikují pomocí NVL. Hlavní předností softwaru je grafické rozhraní usnadňující orientaci v zobrazovaných datech, neboť se snáze hlídá pár analogových ukazatelů než několik číselných hodnot.

Samostatnou součástí je i aplikace *Gateway Emulator*, neboli GE, která slouží pro simulaci rozhraní komunikace s NVL a umožňuje sestavit různé konfigurace celé vlakové soupravy, jakož i nastavit data jednotlivých vozů. Propojení obou aplikací probíhá prostředky operačního systému, pakliže jsou spuštěny na stejném stroji, anebo pomocí rozhraní RS-232 spojujícího dva oddělené počítače.

1.4 Členění práce

Kapitola 1 obsahuje kontext, motivaci a popis zadání diplomové práce.

Kapitola 2 připomene terminologii použitou v rámci celé práce a technické normy pro komunikaci na vlakových sběrnicích jako jsou IEC 61375, UIC 556 nebo Pre TNŽ 28 1500. Její součástí je i popis použitého frameworku Qt 4.

Kapitola 3 obsahuje analýzu a návrh celého systému, jsou zde definovány vstupní podmínky a posuzována omezení jednotlivých řešení. Při návrhu se zohledňuje charakter a platnost uložených dat, jakož i plánování jejich získávání. Dále pak kapitola popisuje strukturu aplikace, způsob a formu komunikace se sběrnicí NVL.

Kapitola 4 se věnuje implementaci výše popsaného návrhu, jeho vývoji a finální podobě. Jsou zde popsány jednotlivé vrstvy aplikace, implementovaný protokol komunikace a popis ukázkového dynamického modulu zobrazení dat.

Kapitola 5 popisuje obecné principy ladění v celé aplikaci. Dále pak testování aplikace proti emulátoru (GE je samostatnou součástí GDS), ale také proti reálným řídícím prvkům vlakových vozů řady 754.1 a 843.0 [1]. Jsou zde shrnuty výsledky těchto testů včetně změn, které vyvolaly. V závěru je popsán textový protokol aplikace GDS pro komunikaci s NVL.

Kapitola 6 rozebírá jednotlivé části rozhraní obou aplikací včetně jejich použití. Seznámíme se s postupem práce s oběma aplikacemi – naučíme se zjišťovat konfiguraci vlakové soupravy, ovládat plánovače řídícího vozidla a zobrazit si např. rychlost vozidla.

Kapitola 7 popisuje stále otevřené problémy, které zůstaly nevyřešeny v této práci, a jsou zde nastíněna možná budoucí rozšíření aplikace.

Kapitola 8 shrnuje výsledky celé práce.

2 Terminologie a technologie

V této kapitole je připomenuta terminologie a technologie použitá v celé práci. Především jsou zde vysvětleny pojmy a principy fungování komunikace na vlakových sběrnicích a zkratky, které jsou hojně používány v oblasti železniční dopravy. V neposlední řadě jsou vysvětleny principy fungování frameworku Qt 4 v němž je celá aplikace napsána.

2.1 Základní pojmy vlakové terminologie

Základní terminologie železniční dopravy v České Republice se řídí Vyhláškou č.173/1995 Sb. [2]

2.1.1 Vlak, vlaková souprava

Dle §1 písmena k) je *vlak* definován jako …

„… sestavená a svěšená skupina drážních vozidel, tvořená alespoň jedním hnacím a jedním taženým drážním vozidlem, označená stanovenými návěstími, s doprovodem vlaku a jedoucí podle jízdního řádu nebo podle pokynu odborně způsobilé osoby řídící drážní dopravu, anebo též samostatné hnací drážní vozidlo nebo speciální vozidlo s vlastním pohonem …“

Návěstími se rozumí označení začátku a konce vlaku, vlakovým doprovodem pak výpravčí nebo dispečer.

V železniční dopravě se vlakem rozumí též *spoj* uvedený v jízdním řádu, tedy jízda vlaku v určené trase a plánovaných časech pod stanoveným označením (druh, číslo, případně název).

2.1.2 Hnací vozidlo

Dle §1 písmena f) se *hnacím vozidlem* (též nazývaným *trakčním*) rozumí …

„… drážní vozidlo, schopné vyvíjet tažnou, případně brzdící sílu pro pohyb a brzdění vlastní a zpravidla i jiných drážních vozidel.“

Mezi hnací vozidla patří lokomotivy a motorové vozy.

Lokomotiva [3] je …

„… hnací kolejové vozidlo neboli trakční vozidlo, ve kterém se energie dodávaná z vnějšího prostředí prostřednictvím paliva nebo elektrické energie mění v mechanickou energii, přenášenou trakčními mechanismy na nápravu resp. kola lokomotivy (železniční dvojkolí).“

Motorový vůz [4] je …

„… železniční kolejové vozidlo, které je hnací a zároveň slouží pro přepravu osob či zavazadel. Pojem motorový vůz je v železniční terminologii zpravidla používán pouze pro motorové vozy poháněné spalovacím motorem, pro vozy s elektrickým pohonem se používá pojem elektrický vůz, pro historické vozy s parním kotlem pojem parní vůz.“

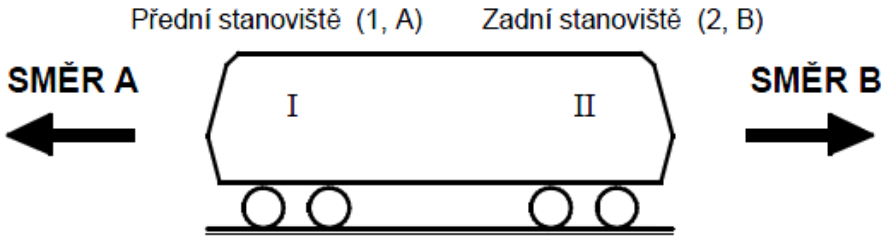
2.1.3 Vlakové vozy

Základní jednotkou vlakové soupravy je *železniční kolejové vozidlo,* což je společný název pro hnací i tažená vozidla. Všechna vozidla v České republice mají dle normy TNŽ 28 0080 [5] označení skládající se ze 7 číslic:

„XXX YYY – Z, kde XXX je označení řady vozidla, YYY je inventární číslo vozidla a Z je kontrolní číslice. Jednotlivé řady pak mají ještě skrytou číslici označující verzi vozidla dané řady.“

Každý vůz má označené oba své konce dle normy Pre TNŽ 28 1500 (norma NVL) takto:

- u vozidel se dvěma stanovišti strojvedoucího (platí i pro vozidla s jednou kabinou) odpovídá směr A směru pohledu ze stanoviště označeného číslem 1 nebo písmenem A.
- u vozidel s jedním stanovištěm strojvedoucího odpovídá směr A směru pohledu z tohoto stanoviště.
- u vozidel bez stanoviště strojvedoucího odpovídá směr A směru pohledu ze středu vozidla k jeho přednímu konci. Směr B je směr opačný ke směru A.



Obrázek 1: Schéma označení konců vozu dle normy Pre TNŽ 28 1500

Základní typy vozidel jsou definovány dle předpisu SŽDC (ČD) D2 [6] takto:

- *Řídící vozidlo* - vozidlo bez vlastního pohonu, které je vybaveno technickým zařízením k dálkovému ovládání určených typů hnacích vozidel.
- *Hnací vozidlo* – vozidlo vyvíjející tah soupravy, viz výše.
- *Tažené vozidlo* (vůz) - je vozidlo, které není hnací nebo speciální, bez ohledu na to, zda je uváděno do pohybu tažením nebo sunutím. Je to společný název pro vozy osobní dopravy, nákladní vozy a vozy speciálního určení.

2.1.4 Konsist

Vyšším celkem než vozidlo jsou tzv. *konsisty* (z angl. consist), což jsou nedělitelné celky (skupiny), čítající jeden nebo i více vozů. Dle definice v normě IEC 61375-1 [7] je to:

„Jeden vůz nebo skupina vozů, které nejsou během normálního provozu děleny.“

Z toho vyplývá, že sestavení těchto vozů probíhá v depu a nelze je během cesty nikterak měnit. Dle americké terminologie pro vlakovou dopravu [8] je konsist:

„… skupina železničních kolejových vozidel, která spolu tvoří vlak. Pokud se používá v kontextu tažné síly, je konsist chápán jako skupina lokomotiv, které tvoří hnací jednotku celé soupravy.“

Z těchto definic tedy vyplývá, že v mnoha případech je konsist jeden vůz (např. u řady vozidel s označením ČD 84x, 94x, 85x, 95x), avšak někdy je tvořen i více

vozy (např. u jednotky řady ČD 680 – Pendolino, kde vlak tvoří 2 spojené konsisty nebo u soupravy M1 pražského metra, kde tvoří konsist celý vlak).

Vlak lze tedy z tohoto pohledu také definovat jako celek tvořený jedním nebo více konsisty, kde konsist tvoří jedno nebo více pevně spojených železničních kolejových vozidel.

2.1.5 Sběrnice

Jak již bylo zmíněno v úvodu, jednotlivé vozy obsahují elektronické zařízení komunikující mezi sebou po různých typech sběrnic. Řízením provozu na nich je pověřen v jednu chvíli právě jeden uzel – *Master*, ostatní uzly jsou podřízené – *Slave*. Výjimku tvoří pouze uzly, které nejsou zařazené do komunikace – *Independent*.

Každý uzel v rámci jedné sběrnice má svoje jedinečné identifikační číslo. To je přiděleno buď při instalaci sběrnice (např. konsistová sběrnice) anebo ji přiděluje linkový Master během procesu tzv. *Inaugurace* (viz níže).

Jednou ze základních typů sběrnic je *konzistová sběrnice*, sloužící zejména pro komunikaci mezi jednotlivými zařízeními uvnitř celého konzistu (tvořeného i více vozy) – např. trakčními měniči napájejícími motory. Obvyklá délka konsistové sběrnice bývá v řádu jednotek až desítek metrů.

Dalším type je *vlaková sběrnice*, která spojuje jednotlivé konsisty podél celé soupravy a zprostředkovává komunikaci mezi nimi. Zároveň slouží k odstínění komunikace probíhající uvnitř konsistů, neboť slouží především pro získání celkového pohledu na dění v soupravě, nikoliv pro získání dat z každého pístu vozu. Typická délka této sběrnice bývá v řádu desítek až stovek metrů.

2.2 Normy IEC 61375 a UIC 556

Vzhledem k tomu, že elektronika proniká i do železničního průmyslu již od 70. let minulého století, není tedy divu, že po určité době vyvstala potřeba komunikace mezi jednotlivými zařízeními uvnitř vlakové soupravy. Díky absenci mezinárodního standardu však jednotliví výrobci řešili způsob, formu a obsah přenášených dat značným množstvím proprietárních a neproprietárních řešení, která nebyla kompatibilní s ostatními.

2.2.1 Historie

Snaha sjednotit tyto rozdíly novým mezinárodním standardem vyústila v roce 1988, kdy *International Electrotechnical Commission (IEC)* ve spolupráci s železničními provozovateli, jako Čína, Francie, Japonsko nebo Německo, sdruženými v *Union Internationale des Chemins de Fer (UIC)* vytvořili pracovní skupinu 22 (*WorkGroup 22* – WG22), která měla tento návrh vytvořit.

Výsledkem je norma *IEC 61375* [7], která definuje komunikační architekturu a nezbytné protokoly pro životně nedůležitou komunikaci na úrovni vlaku a jednotlivých vozů. Skládá se ze dvouvrstvé hierarchické architektury, která vyhovuje potřebám vnitřní i vnější komunikace jednotlivých vozů.

Norma *UIC 556* [7]a její přidružené dodatky definují řídící pohled na vlakovou soupravu, framework pro koordinaci a kompatibilitu různých aplikací a především způsob zajištění dorozumívání mezi jednotlivými vozy vyráběnými různými dodavateli.

Tato sada protokolů se stala základem komunikace v mnoha vyráběných vozech všech typů, včetně tramvajových vozů. V dnešní době se používá v nově vyráběných nebo rekonstruovaných starších vozech.

2.2.2 Práce skupiny 22

WG22 měla dva hlavní úkoly – definovat rozhraní mezi programovatelným vybavením při komunikaci mezi a uvnitř jednotlivých vozů (resp. konsistů), aby bylo dosaženo vzájemné kompatibility. Svou práci založili především na studiu stávajících řešení, aplikaci ověřených postupů a implementačním testováním v praxi před samotným vydáním standardu.

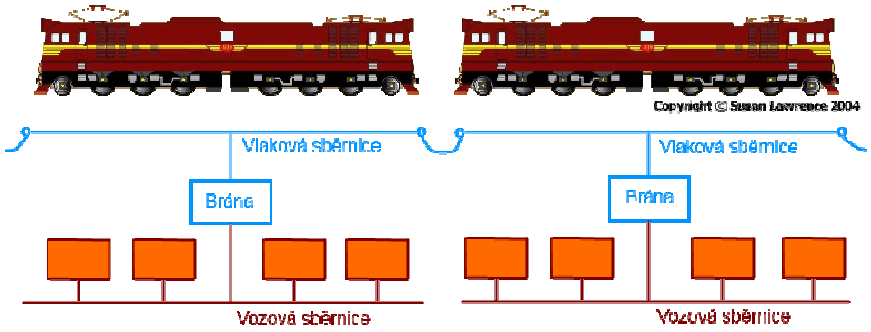
Výsledkem jejich práce je dvouvrstvá architektura, kde:

- *Vlaková sběrnice (TrainBus)* – slouží k propojení jednotlivých vozů mezi sebou. Tato sběrnice má obvykle dosah až 1000 m a může obsluhovat 32 uzlů (celkově až 63 na každou stranu od Mastera)

- *Vozidlová sběrnice (Vehicle Bus)* – slouží pro komunikaci vybavení uvnitř vozu. Tato sběrnice má typický dosah až 200 m a může obsluhovat celkově až 32 zařízení včetně Brány (Gatewaye).

Vlaková sběrnice přistupuje k jednotlivým vozům bez znalosti jejich vnitřní hierarchie jako k souboru funkcí, nikoliv souboru zařízení. Každá taková funkce může být implementována jedním nebo více zařízeními na konsistové sběrnici, případně samotným propojovacím uzlem.

V každém voze jsou obě sběrnice realizovány kabelem (metalickým nebo optickým) a oba segmenty jsou propojeny v jednom bodě, nazvaném Brána, jak znázorňuje následující Obrázek 2: Schéma sběrnic ve vlakové soupravě:



Obrázek 2: Schéma sběrnic ve vlakové soupravě

Zároveň definovali dva typy vlaků:

- *Otevřené* – kde jsou jednotlivé vozy připojovány a odpojovány dle potřeby. Po každé změně složení soupravy nastane re-konfigurace vlakové sběrnice (Inaugurace).
- *Uzavřené vlakové sady* – konsisty (viz výše)

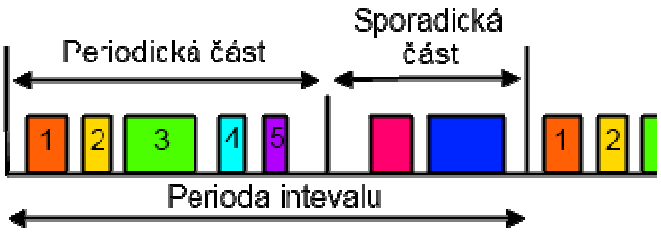
2.3 Norma IEC 61375-2: RTP – Real Time Protocol

Část normy IEC 61375-2 [9] se zabývá protokoly přenosů po vlakových sběrnicích. Po nich totiž proudí nejrůznější typy dat, které lze rozdělit podle důležitosti takto:

- *Řídící* – na vlakové úrovni je to např. řízení tahu a brzd všech hnacích vozidel, na vozidlové úrovni např. stav dveří, světla nebo vytápění
- *Diagnostické* – zprávy o selhání zařízení nebo o jejich stavu
- *Informační* – určeny pro pasažéry vlaku, tedy např. informace o následující stanici nebo přípojích či obsazenosti sedadel

Z těchto tři skupin lze snadno odvodit, že taková data mají buď charakteristiku krátkých (dle normy maximálně 256bitů), ale pravidelně zasílaných zpráv – nazývaných *proměnné* či *procesní data*, nebo dlouhých a nepravidelných *zpráv*. Základní perioda přenášených dat je stanovena tak, aby se nejdříve přenesly všechny proměnné. Poté je vyhrazen čas na částečný přenos zpráv.

Řízením přenosu je pověřen jeden Master, který sestavuje *přenosový plán*, obsahující seznam všech pravidelně zasílaných dat (*periodická* část intervalu) a těch, která byla vyžádána k nepravidelnému zaslání (*sporadická* část intervalu). Master poté začne dle plánu dotazovat jednotlivá vozidla v pořadí daném identifikací uzlu (viz Obrázek 3: Periodická komunikace na sběrnici).



Obrázek 3: Periodická komunikace na sběrnici

2.3.1 Proměnné a porty

Proměnné jsou kriticky důležitá data pro řízení soupravy, avšak vzhledem k faktu, že jsou zasílána pravidelně v krátkých časových intervalech, není nutné při nahodilém výpadku tato data zaslat znova, neboť budou brzy zaslána nová. Hodnoty jsou zasílány v každé periodě, ať už se jejich stav změnil či nikoliv.

Vzhledem k tomu, že jednotlivá zařízení mohou obsahovat velké množství procesních dat, uplatňuje se při jejich přenosu tzv. *princip odběru.* To znamená, že jsou přenášena pouze procesní data vyžádaná, nikoliv veškerá.

Celá linka se chová podobně jako distribuovaná databáze, kdy se data ze zdroje přenášejí pomocí replikací (broadcastem) do pamětí ostatních uzlů. Aplikace tak mohou k těmto datům přistupovat i na jiném uzlu, než zdrojovém.

Přenos proměnných probíhá tedy ve dvou jednoduchých krocích:

- Master pošle identifikace dat, která budou přenesena z jejich zdroje
- Zdroj dat vyšle informaci po sběrnici a všechny uzly, které tyto informace požadují, si je zapamatují.

Získaná data jsou uložena v tzv. *portech* v indexované asociativní paměti (Port Index Table). Pro všechny sběrnice řídící se normou IEC 61375 obsahuje každé zařízení malé množství portů (např. 256), maximálně však 4096. Tyto porty jsou kvůli lepšímu využití paměti formovány do bloků o velikosti 32 portů, kde každý port je velký maximálně 1024 bitů. *Status register* navíc určuje, jestli jde o port zdrojový nebo replikovaný.

Platnost dat uložených v portech není neomezená, aby nebyla data v případě poruchy přenosu a zastarání považována za platná. Stáří dat určuje *počítadlo stáří*, které se přičte při každé přenosové periodě a při příjmu nových dat resetuje. Pakliže nastane chyba při příjmu, není tato chyba propagována přímo aplikacím, ale lze tento stav detekovat právě zastaráním zobrazovaných dat.

Norma dále uvádí některé druhy ochrany proti příchozím poškozeným datům, jako je *Cyclic Redundancy Check* (CRC), případně *Hammingovy kódy* (spolehlivější CRC) nebo redundantně zasílané *kontrolní bity* určující stav zasílaných proměnných.

2.3.2 Zprávy

Oproti proměnným jsou *zprávy* neurgentní necyklicky zasílané, často velmi dlouhé, soubory dat. Používají se především pro diagnostické účely, zasílání zpráv informačních systémů pro pasažéry nebo pro stahování souborů z jednotlivých vozů.

Vzhledem k jejich velikosti jsou rozděleny na jednotlivé *balíky* (packets), které se odesílají v mezeře mezi koncem odesílání pravidelných portů a začátkem dalšího přenosu portů. Díky bezstavovosti komunikačního protokolu a necyklickému charakteru zpráv je nutné zavést mechanismus kontroly doručení jednotlivých balíků. To je implementováno pomocí potvrzovacích zpráv, které odesílatele

informují, že daná část dorazila na místo určení resp. po vypršení časového limitu je část poslána znova.

Komunikace probíhá jako spojový přenos mezi dvěma uzly, přičemž Brána funguje jako router na vlakové úrovni. Jednotlivé balíky jsou vkládány do *datagramů*, které obsahují krom dat také zdrojovou a cílovou adresu určení, potvrzení předchozích dat (resp. doručení potvrzení) a kontrolní součty zajišťující integritu dat.

Routování probíhá na základě dvou tabulek – *station directory*, sloužící pro určení zdrojové resp. cílové sítě nebo subsítě, a *device directory* obsahující jednoznačné identifikátory zařízení v rámci jedné sítě nebo subsítě.

Pakliže dojde ke změně topologie vlakové soupravy (přidání/odebrání vozu nebo změna rolí uzlů), proběhne znovu inaugurace a jsou zrušeny všechny platné konverzace a to i přes to, že změna nemusí ovlivnit žádné probíhající přenosy.

2.4 Norma IEC 61375-3: MVB – Multifunctional Vehicle Bus

Standard IEC 61375-3 [10] definuje komunikační protokol na sběrnici uvnitř vozu (resp. konsistu). Přenosová rychlost se pohybuje okolo 1,5 Mbit/s se zpožděním 1ms. Dokáže obsloužit až 255 programovatelných zařízení a/nebo 4095 jednoduchých senzorů, jimž je přiřazena pevná identifikace během kompletace konsistu.

Sběrnice MVB se dělí podle složitosti na třídy 1-5. Třída 1 (nejnižší) obsahuje pouze senzory, čidla a žádnou řídící jednotku. Třída 5 (nejvyšší) obsahuje programovatelné Brány, které dokážou propojovat více vzájemných MVB nebo WTB linek. Samotná sběrnice je od aplikací oddělena dvoubránovou asociativní pamětí, která umožňuje zápis a čtení najednou.

Komunikace probíhá pomocí zasílání datagramů, které jsou dvojího druhu – *Master message*, které vysílá pouze Master linky a obsahuje příkaz pro zařízení a *Slave message* obsahující pouze datovou část jako odpověď.

Master má na starost následující 4 úkoly:

- Periodické vyžádání portů dle přenosového plánu,
- umožnit přenos neperiodických dat v čase mezi přenosem portů,
- skenování sběrnice, za účelem zjistit případně poruchy či změny chování,
- předání řízení sběrnice.

Vzhledem k tomu, že se tato sběrnice používá v kritických systémech, jako je řídicí systém hnacích vozidel, je vypracován koncept tolerance chyb (cyklické předávání řízení linky mezi více Mastery, redundance komunikačního media). Koncept deklaruje, že MVB raději přestane vysílat data, než by poskytla chybné údaje.

Díky systému ochran a zabezpečení je sběrnice schopna fungovat navzdory chybě některého připojeného zařízení nebo i v případě, že vysílá poškozená data. Vzniklá chyba ovlivní pouze samotné zařízení, nikoliv zařízení závislé na poskytovaných datech.

2.5 Norma IEC 61375-4: WTB – Wire Train Bus

Část normy IEC 61375-4 [11] obsahuje definici sběrnice WTB, která slouží pro komunikaci mezi vozidly vlakové soupravy. Přenosová rychlost se pohybuje okolo 1 Mbit/s se zpožděním 25 ms. Dokáže obsloužit až 32 uzlů na vzdálenost 860 m, kterým je dynamicky přiděleno při *Inauguraci* unikátní identifikační číslo.

Podobně jako u MVB i zde je soubor pravidel, která garantují spolehlivost přenosu – redundance přenosového média, automatické předání řízení linky při selhání Mastera či zdvojení úlohy Master (primární a sekundární Master).

Komunikace probíhá tak, že Master sestaví přenosový plán a pak se postupně dotazuje jednotlivých uzlů (Slave) na přenášené porty. Během sporadické části intervalu se Master dotazuje na data uzlů, které indikovaly přenos zpráv během periodické části intervalu. Přenášené datagramy jsou trojího druhu – *porty*, *zprávy* a *řídicí zprávy*, jejichž obsah je definován v normě IEC 61375-2.

2.5.1 Inaugurace

Inaugurace je proces, při němž je všem uzlům připojeným k vlakové sběrnici přidělena *adresa* a *orientace* relativně vůči linkovému Masteru. Uzly mohou odpovídat vozům, ale také může mít vůz více uzlů nebo nemít žádný. Tento proces se opakuje při každé změně (zkrácení, prodloužení nebo změně Master role uzlu) vlakové soupravy.

Adresy jsou přidělovány od 1, kterou dostane Master. Směrem „za“ (tedy ve směru orientace B) jsou přidělovány čísla od 2 výše, směrem „před“ (tedy ve směru A) jsou přidělována čísla od 63 níže. Každý uzel si navíc zapamatuje směr, ze kterého jej Master oslovil. Tím zjistí svojí orientaci „po“ (souhlasná orientace s orientaci Mastera) nebo „proti“ (opačná orientace vůči Masteru) směru jízdy soupravy.

Inaugurované uzly se dělí do dvou skupin – *vnitřní*, sloužící pro přenos informace oběma směry a *koncové*, které na nespojeném konci detekují změnu, což neznamená nic jiného, než že je připojen další vůz nebo souprava.

Uzly se na vlakové sběrnici rovněž dělí podle toho, jsou-li schopny vykonávat roli Master uzlu, na tři kategorie:

- *Silné uzly* – ty jsou předurčeny k tomu, aby byly vždy Master uzly. Pakliže jsou na lince více než dva silné uzly, rozpadne se linka na jednotlivé segmenty, které jsou samostatné.
- *Slabé uzly* – jsou pro záložní účely, tedy stanou se Master uzlem pouze pokud není přítomen žádný silný uzel.
- *Slave uzly* – nikdy nebudou mít roli Master.

Jakmile je připojen ke koncovému uzlu nový uzel (tzv. *nepojmenovaný*), vyšle koncový uzel tomuto novému signál, jehož prostřednictvím ho informuje o tom, že byl připojen na linku obsahující Master uzel. Nepojmenovaný uzel vyšle požadavek Masterovi o přidělení identifikace a Master odpoví přidělenou adresou, čímž se z něj stane *pojmenovaný* uzel.

Pakliže se uzel připojí na linku, kde není žádný Master uzel, automaticky přebírá jeho roli (jestliže se jedná o silný nebo slabý uzel). Tento stav nastává, pokud je připojen k již existující lince, ale nikdo zatím neaktivoval sběrnici anebo je-li tento

vůz jako samostatná jednotka. Jakmile získá roli Master uzlu, označí koncové uzly a další připojené (aktivované) uzly už si od něj vyžádají svou adresu.

Pokud jsou spojeny dva segmenty, které v sobě obsahují Master uzel (a neobsahují dva silné uzly), je tomu kratšímu segmentu (resp. při rovnosti počtu uzlů tomu, kdo je pomalejší) odebrána role a je začleněn do druhé soupravy.

2.6 Norma IEC 61375-2-5 ETB a IEC 61375-3-4 ECN

Podobně, jako vývoj v oblasti počítačů, tak i v oblasti železniční techniky pokročil vývoj od dob, kdy WG22 vydali normu IEC 61375. Nejnovější snahy výrobců mít stále větší počet jednotek připojených na sběrnice, jejich rychlejší odpověď a hlavně větší množství přenášených dat si vyžádaly změnu přenosových protokolů.

Ethernet Train Backbone (ETB) [12] a *Ethernet Consist Network* [13] (ECN) jsou moderní železniční sběrnice, které při komunikaci používají jako přenosové médium ethernet a rodinu IP protokolů. Již z povahy přenášených dat definované v normě IEC 61375-2 je jasné, že porty se přenášejí podobně jako UDP packety a zprávy se přenášejí jako spolehlivé TCP spojení.

V současnosti existují tři zkušební implementace těchto sběrnic, jejichž vzájemná kompatibilita je velmi omezená. Podobně jako na konci 80. let minulého století, i dnes je snaha sjednotit tyto implementace v mezinárodní standard, který bude platit další desetiletí.

2.7 Návrh normy TNŽ 28 1500 – Národní Vlaková Linka

Podobnou problematiku jako norma IEC 61375 řeší i česká verze vlakové sběrnice Národní Vlaková Linka [14] (NVL). Dle normy byla NVL vyvinuta …

„… v roce 1995 ve VÚŽ na základě požadavku Českých Drah a Moravskoslezské vagónky Studénka pro tehdy vyráběné motorové, řídicí a vložené vozy řad 843, 943 a 043 a tehdy uvažovaný elektrický vůz řady 443.“

Sběrnice je určena pro násobné řízení a přenos provozních a diagnostických údajů mezi vozy vlakové soupravy. Je rozdělena na řídící a diagnostickou linku, aby od sebe striktně oddělila data dle podle jejich účelu.

Řídicí linka propojuje hnací nebo řídící vozidla soupravy a může mít až 8 uzlů. Diagnostická linka propojuje všechna vozidla a může mít až 63 uzlů – Master uzel a 31 uzlů na každé straně (směr A i B).

Linka umožňuje přenos jak procesních dat (porty), tak od roku 2007 také obousměrný přenos zpráv.

2.7.1 Řídící linka

Řídicí linka je …

„… určena pouze k přenosu signálů nezbytně nutných k řízení pohybu vlaku mezi vozidlem s aktivním stanovištěm strojvedoucího a ostatními hnacími vozidly, a to s co nejmenším přenosovým zpožděním a co nejvyšší spolehlivostí. Protože v převážné většině probíhá tento přenos mezi nejvýše dvěma vozidly, je přenosový protokol navržen jako střídavé vysílání rámců neproměnného formátu mezi dvěma vozidly.“

Rámec, který se odesílá po řídící lince, má velikost 12 bytů a to při přenosové rychlosti 9600 Bd dává dobu odeslání délky 15ms. Pro zabezpečení integrity dat se používá v těle rámce Hammingových kódů s velikostí 3, což zajišťuje detekci případné chyby.

2.7.2 Diagnostická linka

Úkolem diagnostické linky je …

„… zajistit přenos provozních hodnot a diagnostických a jim podobných dat mezi jednotlivými vozidly vlaku. Protože se zdroj dat i místo jejich určení může nacházet na libovolném vozidle v soupravě, je účelné zahrnout všechna vozidla do jediného segmentu linky.

Všechny vnitřní uzly segmentu jsou proto přepnuty do T-spojení (kdy jsou propojeny směry A a B do průchozího spojení), kdežto krajní uzly mají odděleně připojené směry a oddělují tak segment od okolí. Provoz na takové lince již musí být řízen linkovým Masterem, ostatní vozy jsou označované jako podřízené (Slave).“

Na rozdíl od WTB jsou jednotlivé uzly číslovány takto: Master má podle normy číslo 1, uzly ve směru A od Mastera mají sudá čísla začínající 2 (2,4,6,…) a ve

směru B od Mastera lichá, počínaje 3 (3,5,7,…). Číslo 0 je vyhrazeno pro dosud nepojmenovaný uzel.

Každý uzel má uložena procesní data v 32 portech, každý port je velký 32 bytů (30 datových bytů a 2 byty určující platnost dat).Účel jednotlivých portů:

- Port 0 je pevně definován a obsahuje identifikaci vozu (řadu, číslo verze, katalogové číslo a kontrolní číslici), zpětné signály vyžadující zvláštní zacházení a seznam obecných poruch,
- port 31 je naopak vyhrazen pro přenos souborů ze Slave na Master uzel (přenos je realizován jako zasílání zpráv dle IEC 61375-2),
- obsah ostatních portů závisí na typu vozidla, všechny porty nemusí být ani využity.

Vzhledem k tomu, že uzel 0 se neúčastní komunikace (čeká na přidělení platné adresy a teprve poté se účastní komunikace), mohou uzly nebo Brány uchovávat data vlastních portů v prostoru pro uzel 0.

Před samotným přenosem dat musí Master sestavit plán přenosu dat, který obsahuje:

- zjištění složení soupravy,
- přenos trvale zobrazovaných provozních hodnot trakčních agregátů,
- přenos dat vyžádaných aplikacemi (např. pro potřeby servisních snímků),
- zprostředkování přenosu dat vyžádaných Slave uzly,
- provoz přenosového kanálu, je-li v době plánování aktivní.

Na každém uzlu připojeném ke směru, pro nějž je relace plánována, je proto povinně naplánován port 0. Tím je zajištěno, že všechny uzly budou znát identitu ostatních.

Rámec odesílaný po diagnostické lince je velký 10-50 bytů podle délky dat. To při přenosové rychlosti 19200 Bd znamená dobu odeslání 6.25-31.25 ms.

2.7.3 GTBA – Gateway Train Bus Adapter

Tento modul slouží pro připojení jedné nebo obou vlakových linek s PC pomocí rozhraní RS-232. Modul se chová jako další uzel na obou linkách a může být, pro

obě linky nezávisle, ve třech stavech – Master, Slave a v módu Příposlechu, kdy neovlivňuje nijak komunikaci na lince, pouze ji zprostředkovává.

S modulem lze komunikovat pomocí speciálního textového protokolu vhodným terminálem, případně přímo psaním a čtením příkazů ze sériového portu PC. V aktivním módu se GTBA chová jako asociativní paměť obsahující data a informaci o platnosti všech portů uzlů připojených k diagnostické lince. Svá vlastní data uchovává dle specifikace NVL v uzlu číslo 0.

2.8 Qt 4 – Cross-platform application and UI toolkit

Qt 4 [15] je jedním ze dvou nejpopulárnějších multiplatformích toolkitů pro výrobu grafického rozhraní aplikací. Byl vytvořen v roce 1999 ve společnosti TrollTech, roku 2008 ho koupila firma Nokia, která jej používá pro vývoj mobilních aplikací. Jeho předností je především možnost vývoje aplikací v různých jazycích a pro rozmanité cílové platformy – od Microsoft Windows, přes Linux až po Symbian S60.

Toolkit poskytuje jednotné multiplatformní API pro přístup k mnoha věcem a definuje proto nové datové typy a třídy, aby tím mohlo dojít ke sjednocení paměťové reprezentace. Ve snaze odlišit nativní a jimi definované typy, přidali před jméno zavedeného typu písmeno Q, a proto se můžeme setkat s typem qint, QString nebo taky QChar či QObject.

2.8.1 QObject a jeho dědičnost

Složitější datové typy jsou reprezentovány pomocí tříd, které mají jednoho společného předka – *QObject* [16]. Díky této dědičnosti lze velmi efektivně využít mechanismus *signálů* a *slotů* (viz níže). Zároveň se tím umožní vytvořit vztah předek – potomek a tím i celou hierarchii objektů, které jsou vytvářeny za běhu aplikace.

Toho je možné velmi dobře využít při správě paměti, kdy v destruktoru objektu se nejdříve odstraní jeho potomci a až pak samotný objekt. Tímto způsobem lze vytvořit v metodě objektu instanci nějaké třídy, jako svého potomka, a nezapamatovat si ukazatel na něj. Jakmile se zavolá na onen objekt destruktor, automaticky bude tato instance odstraněna z paměti.

2.8.2 QWidget a GUI

Podobně jako všechny třídy mají společného předka QObject, tak každý prvek GUI má za společného předka *QWidget* [17] (odtud společný název widgety). Widgety mají stejnou hierarchickou strukturu, jako vytvářejí potomci QObjectu (jsou také poděděny od něj), díky níž se zavoláním metody překreslení jednoho objektu automaticky překreslí všichni jeho potomci.

Hlavní třídou aplikace je *QMainWindow*, které zajišťuje základní okno celé aplikace. V něm je vložen jeden centrální widget, který spolu s oblastmi pro menu, nástrojovými lištami a stavovým řádkem tvoří GUI aplikace. Do centrálního widgetu lze vkládat další widgety a různé jiné grafické prvky.

2.8.3 Signály a sloty

Důležitou vlastností Qt 4 toolkitu je přítomnost signálů a slotů [18] pro komunikaci mezi objekty např. pokud se ve widgetu uskutečnila akce, která změnila jeho stav, tak o tom může být informován widget umístěný v jiném okně aplikace. Tvoří tak velmi silný programátorský nástroj, který nahradil zastaralé Callbacky.

Sloty a signály mohou být využity ve všech objektech, které jsou přímo nebo nepřímo zděděny ze třídy QObject. Při propojování signálů a slotů může být s jedním slotem spojeno několik různých signálů a stejně tak na jeden signál napojeno několik slotů. Sloty mohou být použity pro přijímání signálů a zároveň mohou být použity jako standardní metoda objektu.

Pakliže je signál vyvolán, jsou synchronně obslouženy všechny sloty ve stejném vlákně aplikace, jako je zdroj signálu, a asynchronně všechny v jiných vláknech. Na procesoru i586-500 lze během sekundy vyvolat 2 miliony signálů pro jednoho příjemce nebo 1,2 milionů pro dva. Rychlost je nicméně pomalejší než Callbacky, avšak to je vyhrazeno možností při signálu jako argument předat libovolné datové typy, včetně uživatelem definovaných.

3 Analýza a návrh systému

V této kapitole se seznámíme s návrhem celé aplikace a se všemi omezeními kladenými na tento návrh. Postupně projdeme celý vrstevnatý model, přičemž zdůrazníme jeho zajímavé části.

3.1 Implementační prostředí, platforma a jazyk

Jedno ze zásadních rozhodnutí, které nakonec ovlivnilo i celkový vzhled aplikace, byla volba implementačního jazyka. V úvahu připadalo několik vývojových prostředí, avšak zadání stanovilo omezující podmínky, které neumožnily volbu některých možností použití.

Mezi základní omezení, která ovlivnila volbu jazyka, patřila tato:

- Cílová platforma je Microsoft Windows, avšak měla být brána v úvahu možnost přenositelnosti na jiné platformy
- Komunikace s GTBA probíhá pomocí rozhraní RS-232, do budoucna se mělo počítat s možnou podporou sběrnice CAN nebo pomocí kabelu s koncovkou RJ-45 (LAN kabely).
- Rychlost komunikace na sběrnicích je v řádu 20 ms na zprávu, GTBA obnovuje informace v portech v řádu stovek ms. Je nutné zpracovávat rychle se opakující malé množství dat.
- Grafické rozhraní musí obnovovat zobrazování dat rychlostí alespoň blížící se rozlišovací hranici lidského oka = 0,1s.
- Důraz byl kladen i na ergonomii celé aplikace, její ovládání musí být intuitivní a snadno rozšiřitelné o další prvky

Především kvůli rychlostem na jednotlivých sběrnicích bylo jasné, že nelze pro komunikační rozhraní s nimi použít žádný jazyk, který nemá přímou kontrolu nad správou paměti (jazyky obsahující Garbage Collector). Volba jazyka jádra aplikace byla tedy zřejmá – C/C++.

U uživatelského rozhraní už to nebylo tak jednoznačné. Vzhledem k problémům, které má jazyk Java s integrací C/C++ modulů, zbyly ve výběru dvě platformy – kompletní C/C++ program s využitím některého grafického frameworku nebo platforma C# .NET spolu s C++/CLI pro přístup ke komunikačnímu rozhraní.

Nakonec díky propracovanosti frameworku Qt 4, dobrým osobním zkušenostem s ním a v duchu zachování přenositelnosti i na jiné platformy zvítězila monolitická aplikace s verzí Qt 4.6.2. Drobnou nevýhodou oproti .NET se ukázala nutnost napsat vlastní ovladač na seriový port.

3.2 Omezení kladená na návrh aplikace

Kromě volby jazyka je potřeba během návrhu celé aplikace dbát i na další omezující kritéria. Mezi nejdůležitější patří tato:

- Celou aplikaci mají tvořit dva samostatné celky, přičemž jeden má sloužit jako prototyp aplikace pro monitorování dění na NVL a druhý má simulovat modul GTBA, který obsahuje data o složení vlakové soupravy a porty jednotlivých vozů. Provozování obou aplikací bude umožněno pomocí prostředků operačního sytému na jednom počítači, případně pomocí sériové linky na dvou počítačích.

- Návrh musí počítat s dvojím druhem dat ze sběrnic – porty, zprávy. V návrhu NVL jsou zprávy realizovány pomocí čtení a zápisu do vyhrazeného portu 31 (viz výše).

- Data uložená v portech NVL mají různých charakter – od textových řetězců (data informačního systému), BCD kódu (např. identifikace vozu) až po fyzikální veličiny (tlak, rychlost, teplota …). V návrhu je potřeba zohlednit zobrazení dat v portech dle jejich charakteru.

- Prvky grafického rozhraní aplikace je potřeba volit tak, aby se pokud možno co nejvíce přiblížily reálnému vzhledu. V úvahu se musí vzít také fakt, že lze seskupovat více jednoduchých informací do logických celků – např. hodnotu rychlosti vozu s faktem, že zobrazovaná data mohou být *neznámá*, *aktuální* nebo *zastaralá*.

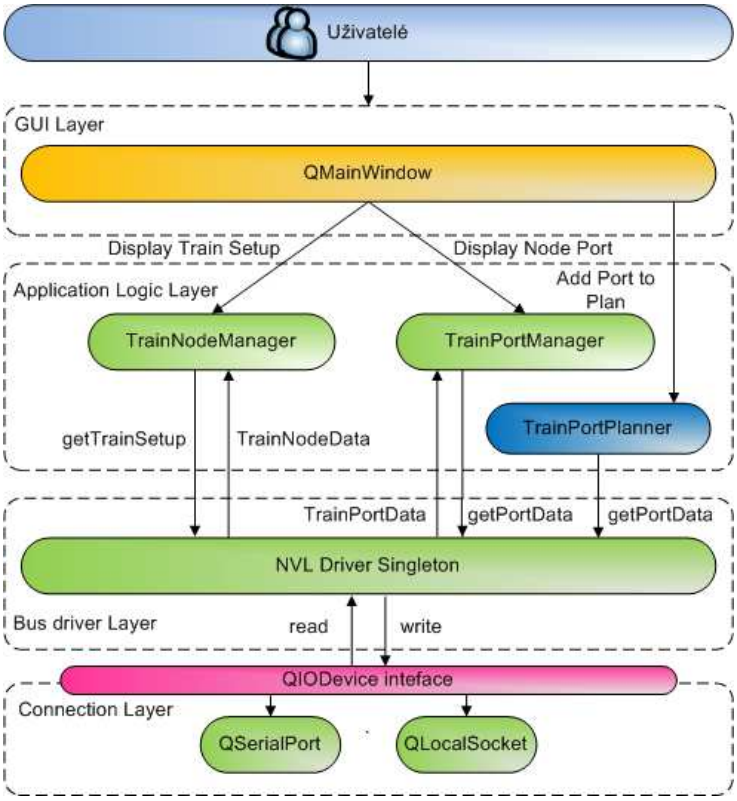
- Je potřeba definovat systém, kterým lze zobrazit libovolně uložená data, přičemž základní jednotka informace byl nibble (resp. horní nebo dolní 4 bity jednotlivých bytů portů).

- Vzhledem k rozdílným rychlostem operací v PC a komunikace pomocí sériové linky s GTBA je nutné umožnit asynchronní přístup k rozhraní RS - 232, zahrnující čtení i zápis. Implementace musí splňovat standardní Qt rozhraní pro práci se vstupně-výstupními zařízeními QIODevice.

- Je nutné zohlednit roli GTBA jako uzlu připojeného k NVL. Jednotlivé role (Master, Slave, Příposlech) totiž ovlivňují funkcionality, které GTBA umožňuje, a lze je tedy v aplikaci využít.

3.3 GDS – Graphics Diagnostics Systém

Díky volbě jazyka C/C++ a frameworku Qt, bude celý systém napsán jako monolitická aplikace s vícevrstvou strukturou. Návrh je přehledně zobrazen na následujícím diagramu:



Obrázek 4: Návrh architektury aplikace

Každá z těchto vrstev bude v samostatném vlákně, které tak umožní např. ovládání aplikace, zatímco se jiné vlákno bude starat o posílání a interpretaci výsledků jednotlivých příkazů.

Díky této struktuře lze v budoucnu velmi jednoduše rozšiřovat funkčnost aplikace. Pakliže bude potřeba změnit přístup k některé sběrnici, např. prostřednictvím UTP kabelu, stačí doplnit modul do vrstvy Connection Layer, který splní rozhraní QIODevice a vrchní vrstvy aplikace nebudou muset být nijak měněny.

3.3.1 Connection Layer

Základem celé aplikace je vrstva umožňující spojení s okolím. V rámci zachování jednotného přístupu, musí každý modul této vrstvy dodržet rozhraní QIODevice, aby tak odstínil implementační a platformní rozdíly. Vyšší vrstvy aplikace tak nepoznají, zda komunikují pomocí sériové linky, sběrnice CAN anebo s emulátorem.

Vzhledem k volbě frameworku Qt 4 je pro připojení k GTBA nutné napsat vlastní modul pro komunikaci po sériové lince. Návrh bude muset počítat s budoucím rozšířením o podporu OS Linux, v rámci prototypu však stačí ovládání sériového portu pod platformou Microsoft Windows.

S aplikací Gateway Emulator se GDS spojí pomocí QLocalSocket, což je standardní Qt 4 třída pro komunikaci pomocí Named Pipe (Windows), případně BSD Sockets (ostatní OS).

3.3.2 Bus Driver Layer

Nad rozhraním QIODevice bude vrstva sloužící pro převod komunikačního protokolu s jednotlivými sběrnicemi do podoby volání příslušných funkcí. Tím lze nejen jednoduše oddělit horní vrstvy od sběrnice, ale zároveň rozšířit protokol o sofistikovanější funkce, které vzniknou např. spojením některých základních příkazů dohromady.

Volání metod bude rozděleno podle charakteru příkazu. Jestliže nemám dostupný výsledek a tudíž je zbytečné pokračovat v práci, pak volání bude synchronní. Ostatní volání budou asynchronně, místo Callbacků se použijí signály, díky kterým lze v případě potřeby převést asynchronní volání na synchronní a data lze předávat i jiným vláknům, než těm, ve kterých budou jednotlivé drivery pracovat.

Odpovědi na příkazy budou rozeznávány podle hlavičky, která obsahuje zopakování daného příkazu. Tím lze spárovat příkaz a jeho odpověď a lze se tak vyvarovat chybám, které mohou nastat, pokud bychom odpovědi rozeznávali pomocí regulárního výrazu. Běžně totiž nelze počítat s tím, že některé odpovědi na příkaz nejsou nebo nebudou velmi podobné – na úrovni regulárních výrazů nerozeznatelné.

Výhoda tohoto přístupu je oddělení vlastní komunikace od logické vrstvy aplikace. Horní vrstvy tak nemusí znát protokol, kterým se na vybrané sběrnici komunikuje, avšak jakékoliv rozšíření horní vrstvy o novou funkcionalitu vyžaduje změnu i této vrstvy.

3.3.3 Application Logic Layer

Tato vrstva tvoří logiku celé aplikace. Základem jsou dvě samostatné třídy – TrainNodeManager a TrainPortManager, starající se o správu uzlů a portů, včetně jejich obnovy pomocí vstupů, které jim předá NVLDriver. Součástí bude i TrainPortPlanner, který se bude starat o pravidelnou obnovu data portů a zajistí tím aktuální data pro GUI vrstvu.

Každá ze tříd, které se starají o správu dat, bude obsahovat pole o maximální možné velikosti, čímž se zajistí rychlý a jednoduchý přístup do paměti. Pro sběrnici NVL to znamená pole o velikosti 64 uzlů a 32*64 portů. Každý uzel bude obsahovat vlastní identifikaci dle normy ČD, stav a roli na lince. Každý port bude mít počet přijetí, periodu obnovy a 30 bytů dat (resp. 15 wordů).

TrainPortPlanner, běžící v samostatném vlákně, bude sloužit pro pravidelné získávání vybraných portů. Jeho konfiguraci bude měnit uživatel pomocí grafického rozhraní, proces pak automaticky v cyklech bude žádat obnovu vybraných portů po NVLDriveru. Za zpracování výsledků bude ale zodpovědný TrainPortManager, aby se tak sjednotil způsob zpracování updatu portu, o který požádal uživatel a nebo TrainPortPlanner.

3.3.4 GUI Layer

Základem uživatelského rozhraní je centrální widget QMainWindow. Do něj jsou vkládány dynamicky další widgety dle potřeb uživatele (např. QDialBox). Slouží také k propojení signálů a slotů jednotlivých widgetu s nižšími vrstvami aplikace, především Application Logic Layer, ale také Bus Driver Layer.

U jednotlivých částí centrálního widgetu se propojení s nižšími vrstvami vytváří při jeho instanciaci, u dynamických částí však tento mechanismus nejde použít. U nich je potřeba předat během vytváření ukazatel na příslušný port, který je zdrojem jejich dat. Díky bitové masce a posunutí lze takto dostat z libovolného bytu (resp. wordu) požadovanou hodnotu.

Problém je pouze s platností dat. U přijatých portů NVL není uvedena doba zbývající platnosti, pouze počet přijetí a doba za kterou budou data znovu obnovena. Pakliže při další periodě nejsou přijata nová data, počet přijetí zůstane stejný. Přidáním QTimeru, který mění stav dat z aktuálních na zastaralá, přičemž se resetuje právě změnou počtu přijetí, lze upozornit uživatele na zastaralost zobrazovaných dat.

3.4 GE – Gateway Emulator

Gateway Emulator je jednovláknová aplikace sloužící pro simulaci dat modulu GTBA linky NVL. Veškeré operace budou prováděny v GUI vlákně, neboť jejich délka není příliš velká a není tedy potřeba aplikaci jinak výrazně členit.

Komunikace s GDS bude probíhat pomocí stejného rozhraní jako v GDS s tím rozdílem, že GE bude navíc obsahovat QLocalServer, který zprostředkovává spojení mezi dvěma aplikacemi pomocí QLocalSocket (čeká na dvě příchozí spojení, aby je mohl propojit a tím udělal dvoustranný tunel pro komunikaci).

Vrstva pro správu jednotlivých uzlů sběrnice NVL a jejich portů bude nahrazena jednoduchou strukturou, avšak vzhledem k tomu, že úkolem GE je pouze číst a zapisovat data, přičemž s nimi nijak nemanipulují, je tato struktura dostatečná.

Veškeré statické součásti grafického rozhraní budou sloučeny do jednoho centrálního widgetu, neboť není potřeba složitého členění ani žádných dynamicky přidávaných části.

GE bude podporovat veškerou funkcionalitu, kterou implementuje NVL Driver v GDS a bude tak schopen mu dodat relevantní data, včetně identifikace GTBA uzlu a reakce na změnu jeho role v rámci sběrnice NVL.

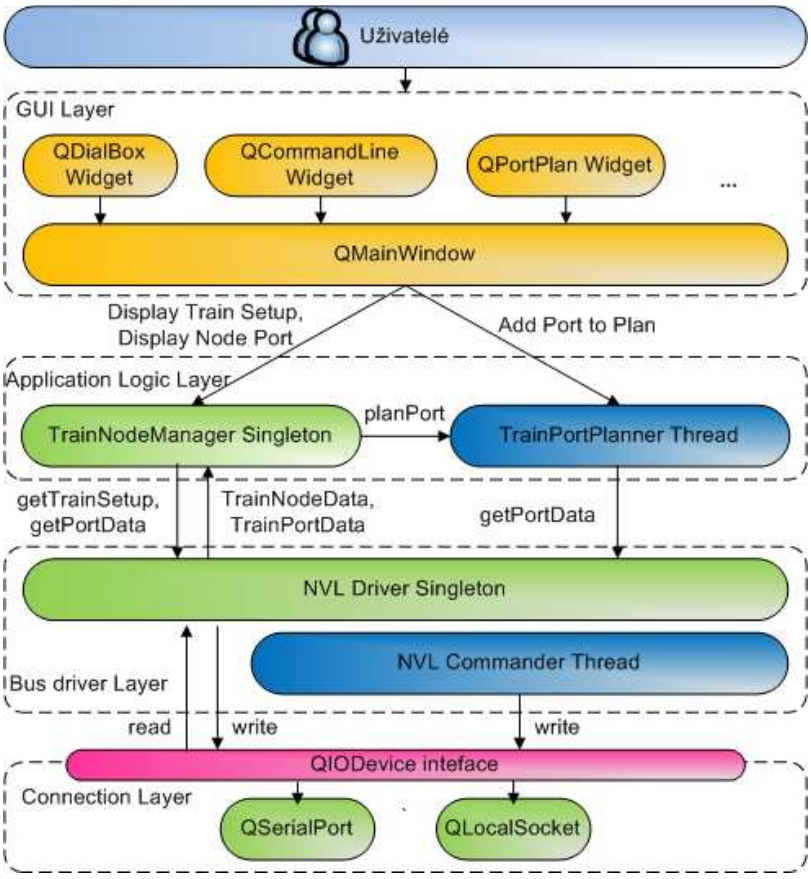
4 Implementace

Obsahem této kapitoly je popis finální implementace jednotlivých návrhů popsaných výše, včetně rozebrání ukázkového dynamického modulu QDialBox. Ten se od návrhu mírně liší, což je způsobeno hlavně odhalením jeho nedostatku v době testování.

Popis zahrnuje pouze zajímavé aspekty jednotlivých částí aplikace, do kompletní implementace lze nahlédnout v přiložených zdrojových souborech. V závěru kapitoly je popsán diagram datových toků v aplikaci.

4.1 Struktura aplikace

Návrh celé aplikace GDS doznal během fáze implementace drobných změn. Rozdělení na vícevrstvou strukturu zůstalo, avšak došlo k některým sloučením nebo rozpadu tříd. Konečný model vypadá takto:



Obrázek 5: Finální model aplikace

Aplikace Gateway Emulator je napsána podle prvotního návrhu, její účel je jednoduchý a návrh se ukázal jako dostatečný. Obě dvě aplikace byly ještě finálně opatřeny ochranou, aby nedošlo k nesprávnému používání (některé prvky GUI jsou dostupné až po té, co jsou k dispozici relevantní data).

4.2 Connection Layer

Základem této vrstvy jsou dvě třídy a jeden interface – QSerialPort, QLocalSocket a QIODevice. Zatímco poslední dvě jmenované třídy jsou standardní součástí frameworku Qt 4.6.2, QSerialPort bylo potřeba naprogramovat.

Vzorem implementace se stala předloha QExtSerialPort [19], která však postrádala několik nezbytných vlastností. Především její implementace se chovala pasivně, což znamená, že nijak nedávala vědět o tom, že jsou k dispozici nová příchozí data. Tomu byl přizpůsoben i charakter volání metody read – její volání bylo neblokující a pokud nebyla k dispozici nová data, vrátila metoda prázdný výsledek.

Operační systém Microsoft Windows umožňuje otevřít sériový port ve dvou módech – *normal* a *overlapped* [20]. První možnost umožňuje čtení a zápis z/do otevřeného portu neblokujícími funkcemi, které se ihned vrátí a pokud nemá žádná nová data ke čtení/pro zápis, pak vrátí prázdný výsledek.

Druhá možnost zavolá asynchronní čtení nebo zápis a vrátí deskriptor, na kterém lze zablokovat průběh metody do doby, než je operace čtení dokončena přečtením alespoň jednoho příchozího znaku. Nebo provede zápis odesláním všech odchozích dat po sériové lince.

Právě druhý mód se stál základem třídy QSerialPort. Ta se chová jako buffer pro příchozí data, která ji dodává vlákno QPortReader, jež se blokuje operací čtení nad otevřeným portem. Čtení z objektu QSerialPort je tedy pouze čtením z tohoto bufferu, který vysílá signál, pokud jsou do něj přidána nová data.

Zápis je pomocí vlákna QPortWriter, který má frontu dat pro zápis do portu a ty zapisuje postupně podle toho, jak přicházejí z QSerialPortu a jakmile nemá co psát, je uspán do doby, než budou nová data k dispozici.

4.3 Bus driver Layer

Tato vrstva slouží jedinému účelu – převodu textového protokolu jednotlivých sběrnic na volání funkcí. Pro NVL a její Bránu (modul GTBA) slouží objekt NVLDriver. Jeho implementace se však oproti původnímu návrhu liší, neboť během testování byly zjištěny v jeho návrhu nedostatky.

Dolní vrstvy aplikace komunikují řádově menší rychlostí, než vrchní vrstvy, a proto se stávalo, že horní vrstvy aplikace se zbytečně blokovaly na volání některých synchronních funkcí. Mezitím totiž mohly přijímat další požadavky uživatele a zajišťovat data pro uživatelské rozhraní.

Dalším problémem bylo párování příkazů a odpovědí, neboť pokud byl některý příkaz opakován, mohlo se stát díky rozdílnosti rychlosti, že odpověď byla spárována s některým z předchozích volání stejného příkazu.

Bylo tedy potřeba vytvořit buffer, který by skladoval požadavky a až po přijetí odpovědi by poslal následující příkaz pomocí komunikační vrstvy. Takto lze jednoduše smazat rozdíl v rychlosti a snadno spolu spárovat příkaz a jeho odpověď. Bylo proto vytvořeno vlákno NVLCommander.

Horní vrstvy tak mají k dispozici asynchronní volání metod, které vloží do sdílené fronty příkazy, jež následně NVLCommnader vykoná. Odpovědi na tyto příkazy jsou propagovány vyšším vrstvám pomocí systému signálů, které emituje NVLDriver.

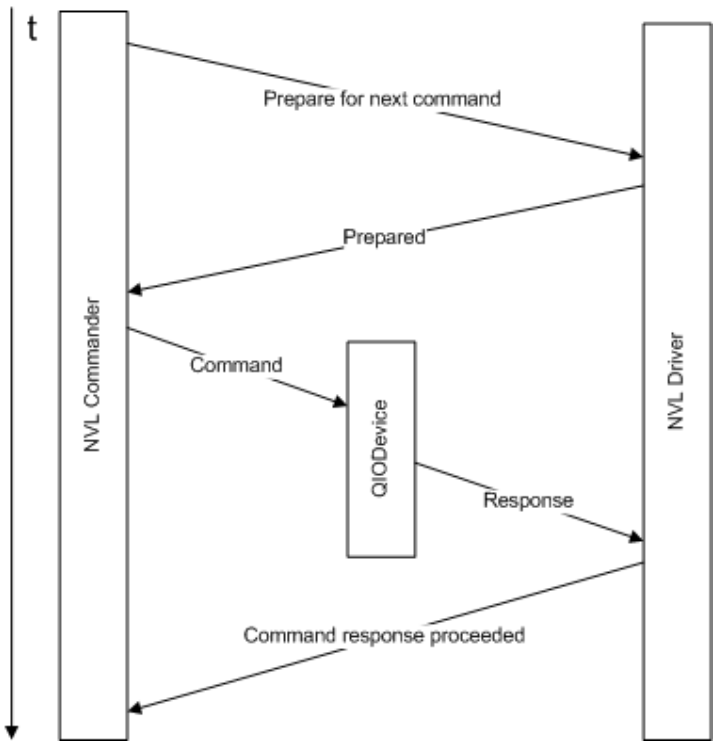
Statický objekt NVLDriver tedy začal sloužit spíše jako rozhraní k tomuto vláknu, NVLCommander pak jako consumer nad společnou frontou, do které vkládá příkazy producer – NVLDriver.

Existují však výjimky z tohoto protokolu, jako jsou potvrzovací odpovědi a příkazová řádka uživatelského rozhraní (viz Kapitola 4.3.2 Výjimky komunikace).

4.3.1 Komunikace s GTBA

Komunikace s modulem GTBA probíhá pomocí textového protokolu přes rozhraní RS-232. Každý příkaz má přesně danou syntaxi. Odpověď se skládá ze tří částí – hlavičky (zopakování příkazu a znaku konce řádku), těla s odpovědí a koncovky (skládající se z řetězce „NVL>“).

Celá komunikace probíhá stavově, podle následujícího schématu:



Obrázek 6: Schéma komunikace se sběrnicí NVL

Jakmile dojde ke zpracování odpovědi na příkaz, je poslán tento stav pomocí signálu vláknu NVLCommander. Ten zjistí, zda-li má ještě nějaký příkaz, který je třeba vykonat, a případně se uspí do doby, než je znovu probuzen přidáním nového příkazu na zpracování.

Poté, co je připraveno odeslání dalšího příkazu, je vyvolán signál, aby NVLDriver zachytil odpověď. Připravenost je opět potvrzena signálem a teprve poté je zapsán příkaz na rozhraní QIODevice. Jakmile je v modulu GTBA vygenerována odpověď, je po příjmu nižšími vrstvami zpracována v NVLDriveru a po příjmu koncovky opět vyslán signál pro zpracování následujícího příkazu.

Tento model komunikace je však v určitých případech porušen a to v takových, kde se během odpovědi na určitý příkaz čeká interakce uživatele (většinou jde o potvrzení některé změny – např. změny role modulu GTBA na lince NVL).

4.3.2 Výjimky komunikace

Výjimku z tohoto protokolu komunikace uvnitř GDS má také příkazová řádka, reprezentovaná widgetem QCommandLineWidget. Ten posílá modulu GTBA zprávy ve třech režimech.

Normal – je režim, ve kterém se celá komunikace obalí hlavičkou „textu“ a je zařazena do fronty zpráv. Příkaz není nijak interpretován aplikací GDS a tedy i jeho odpověď není zpracovávána, přestože by mohla obsahovat relevantní data.

Interactive – je režim, kdy je příkaz rovnou poslán modulu GTBA a tím obejde čekací frontu. Odpověď na něj není opět nijak zpracovávána.

Command – v tomto režimu se GDS snaží odeslaný řetězec ztotožnit s jedním z podporovaných textových příkazů a pokud tak uspěje, provede odeslání jako by byl tento příkaz spuštěn některým prvkem GUI. Pakliže se mu tato unifikace nepodaří, je poslán text jako v režimu Normal.

4.4 Application logic Layer

Jádro aplikace je, oproti návrhu, tvořeno jedním objektem TrainNodeManager. Ten se stará o zpracování veškerých příchozích dat ze sběrnice, zprostředkovaných pomocí NVLDriveru.

Snaha mít uzly a porty jako jednoduché struktury, se během implementace ukázala jako zbytečně komplikovaná. Především proto, že obě entity musí dávat svému okolí nějakým způsobem vědět, pokud jsou změněny. Úlohu upozornit na změnu přebraly nově vzniklé třídy na sebe a tím se stal TrainPortManager nadbytečným.

Zapouzdřením do tříd lze velmi jednoduše např. vytvořit prvek GUI, předat mu ukazatel na jeho zdrojový port a napojit vlastní slot pro obnovu na signál, který port emituje, když je aktualizován.

Uzly jsou nyní třídy (NVLNode), které v sobě obsahují vlastní identifikaci a také pole portů (NVLPort), které mu patří. Vzhledem k faktu, že uzly jsou předávány jako parametr signálu, dochází k jejich kopírování a po obsloužení signálu i k destrukci. Proto jsou porty umístěny do speciální struktury a v samotném uzlu je pouze reference-counted ukazatel na tuto strukturu, čímž se zajistí, že data nebudou

zbytečně kopírována a při změně se projeví v celém systému, nikoliv pouze v dané kopii.

K pravidelné obnově portů slouží vlákno dle návrhu TrainPortPlanner, které postupně zpracovává jednotlivé požadavky na obnovu. Jakmile je port obnoven, je požadavek na jeho znovuobnovení opět vložen do fronty požadavků vlákna TrainPortPlanner. Pakliže nejsou žádné požadavky na obnovu, je vlákno uspáno do doby, než bude nějaký požadavek opět vložen.

4.5 GUI Layer

Tato vrstva v sobě obsahuje dva základní typy objektu – statický (vytvářen při startu aplikace) a dynamický (vytvářen dle potřeby). Statické widgety jsou jako součást centrálního QMainWindow a slouží pouze pro to, aby byl tento widget rozdělen na logické celky a jeho kód byl přehledný. Centrální widget se rovněž stará o propojení jejich signálů se sloty nižších vrstev.

Zajímavou částí uživatelského rozhraní jsou dynamicky vytvářené widgety. Ty jsou rozděleny na dva druhy – QAbstractPullWidget, který má metodu start a v ní je specifikován pravidelný interval, po kterém dojde k znovupřečtení dat ze zdrojového portu a QAbstractPushWidget, který má svůj slot, provádějící update, napojený na update signál zdrojového portu.

Zatímco druhý typ objektů je pouze obdobou staticky vytvářených widgetů, s tím rozdílem, že jsou vytvářeny až při běhu aplikace na žádost uživatele, první druh je o něco zajímavější a stojí za to, si jej rozebrat podrobně.

4.5.1 QDialBox a QDialWidget

QDialWidget je jednoduchý widget, který slouží k zobrazení hodnoty rychlosti na nakresleném tachometru (nebo jiného údaje, který lze zaznamenat pomocí ručičkového budíku). Vstupní hodnota je převedena na cílový úhel a poté je zahájena animace, kdy dochází k přechodu od původního k cílovému úhlu. Jestliže je během této animace změněn cílový úhel, je z aktuální pozice vytvořen počáteční úhel a celá animace je restartována.

Tento widget je vložen do jiného - QDialBoxu, který je právě potomkem třídy QAbstractPullWidget. QDialBox v sobě obsahuje odkaz na zdrojový port a dva QTimery, které slouží pro zajištění jeho běhu. Jeden z nich je přednastaven na

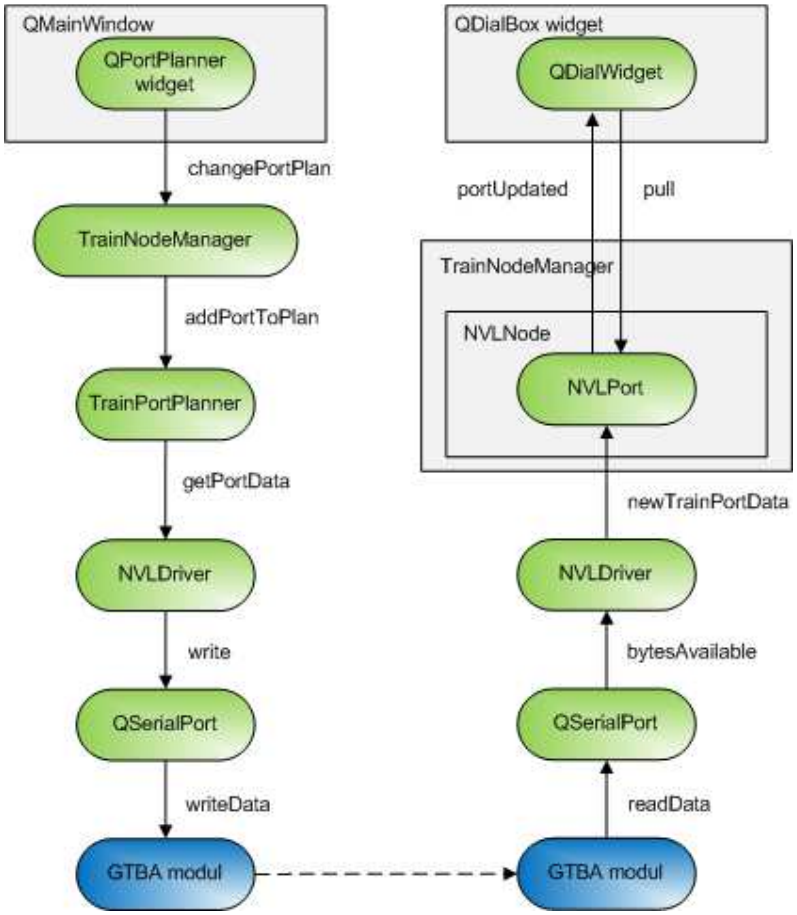
pravidelných 100 ms, kdy přečte hodnotu ze zdrojového portu a nechá ji zobrazit na QDialWidgetu.

Druhý z nich pak je přednastaven na 2 sec a je restartován pokaždé, když dojde k updatu portu. Pakliže dojde k vypršení časovače, znamená to, že zobrazovaná data už nejsou aktuální, ale zastarala a je o tomto uživatel patřičně informován.

4.6 DataFlow

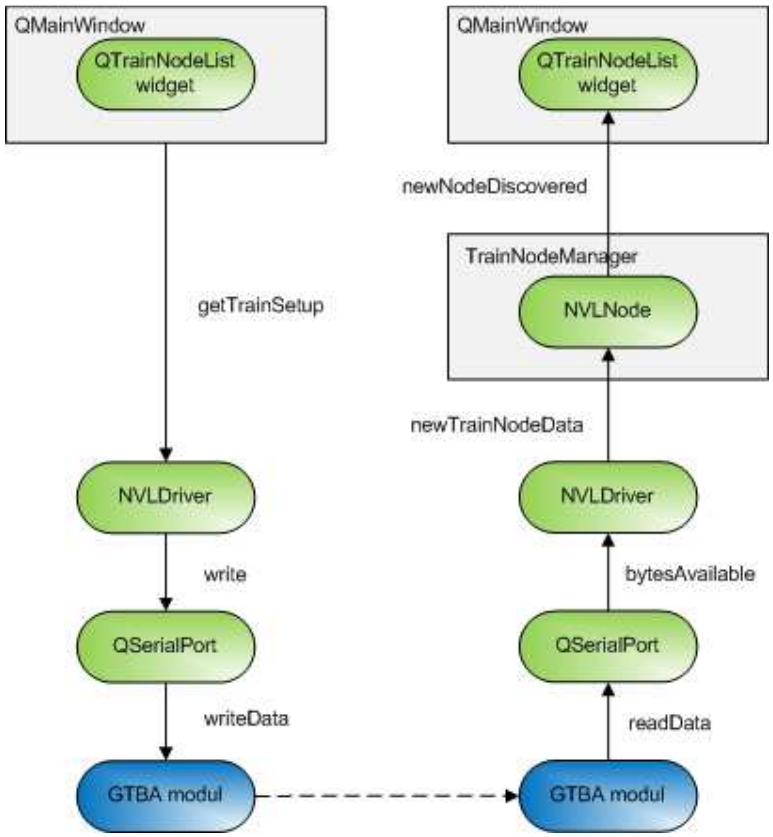
Následující diagramy ukazují nejběžnější toky dat – schéma automatické obnovy dat vybraného portu a zjištění složení vlakové soupravy.

Schéma obnovy portu obsahuje postupný popis průběhu obnovy – od zařazení portu do plánu automatické obnovy, přes průchod ovladačem sběrnice NVL až po zobrazení nových dat pomocí QDialWidgetu.



Obrázek 7: Diagram obnovy dat v portu

Druhý diagram zachycuje schéma zpracování požadavku na získání nového složení vlakové soupravy. QTrainNodeWidget před zavoláním metody getTrainSetup ještě vyvolá signál, kterým vymaže aktuální seznam vlakových uzlů včetně jejich portů a znepřístupní některé prvky grafického rozhraní, aby tak nezobrazovaly neaktuální data.



Obrázek 8: Diagram obnovy složení vlakové soupravy

5 Testování

Nedílnou součástí implementační fáze projektu bylo i testování. V této kapitole jsou popsány jeho hlavní fáze a změny návrhu, které z výsledků vyplynuly. Součástí kapitoly je i popis části textového protokolu komunikace, který Graphics Diagnostics System (GDS) používá ve svém rozhraní s Bránou (GTBA).

5.1 Globální přístup k testování

Celá aplikace je rozdělena do několika vláken, které mezi sebou komunikují pomocí signálů a slotů, občas dokonce sdílejí některé fronty dat. Bylo tedy potřeba zvolit nějaký univerzální přístup v testování aplikace, který pomůže během celé fáze implementace.

Základem se stala funkce qDebug [21], která umožňuje vypisovat cokoliv převoditelného na QString do ladící konzole – v prostředí Microsoft Visual Studia je to součást jeho rozhraní, při normálním spuštění je to konzole (příkazová řádka).

Dalším důležitým pomocníkem při ladění bylo makro Q_ASSERT [21], které umožňuje testovat invariant na jeho správnost. Takto lze jednoduše ověřit vstupní podmínky ve volání funkce a případně tak odladit nějaký problém, který nastal již dávno a programátor jej zapomněl ošetřit.

V neposlední řadě se podařilo pomocí návodu [22] zapnout i kontrolu správy paměti a tím zajistit odstranění některých zapomenutých objektů. Qt 4 používá při odstraňování prvků grafického rozhraní principu tzv. *odloženého odstranění*. To znamená, že objekty jsou zničeny až teprve v následujícím cyklu překreslení grafického rozhraní, přestože ukazatele na ně jsou již neplatné. Při vypnutí se nemusí rozhraní překreslit, a tedy tyto objekty jsou označeny jako memory-leak, ačkoliv nejsou.

Díky těmto obecným postupům se dařilo odhalovat nejrůznější záhadné stavy nebo opakované volání metod, případně i některé špatné posloupnosti volání či předání vadných parametrů signálům.

5.2 Logování

Nepostradatelným, především pro studii protokolu komunikace s NVL, se stalo logování. Zobrazení komunikačního logu bylo jedním z cílů, které měla aplikace

splnit, aby tak plně nahradila hyperterminál. Brzy se však ukázalo, že ruční kopírování logů z rozhraní aplikace a jejich ukládání je značně nepohodlné a proto byla implementována třída QLogger.

Statické rozhraní NVLDriver vysílá dva signály, které oznamují příchozí a odchozí komunikaci se sběrnicí. Na tyto signály jsou napojeny sloty instance QLoggeru a díky tomu zachytávají automaticky každou komunikaci.

Umístění samotných logů je pak v adresáři %APP_DATA%\GDS\, kam se v operačních systémech zpravidla ukládají podobné pomocné soubory aplikací. Poslední log je zaznamenán v souboru *commlog.txt*, starší pak mají za jménem ještě datum poslední změny.

5.3 Sériový port

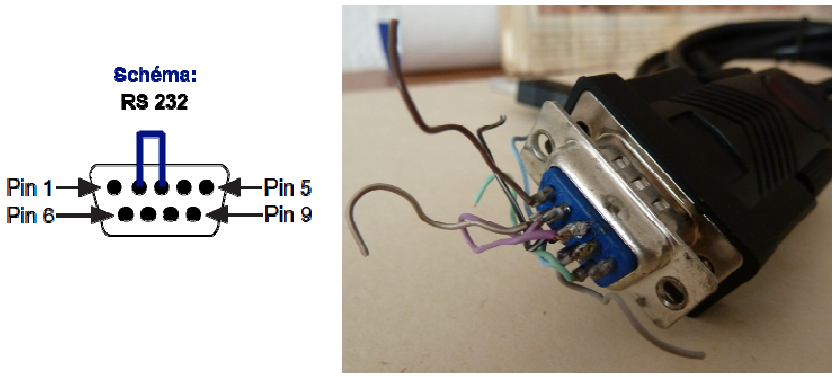
Prvním a zásadním milníkem ve vývoji bylo naprogramování rozhraní pro komunikaci pomocí sériové linky. Součástí frameworku Qt 4.6.2 nebyla žádná základní třída umožňující alespoň synchronní komunikaci po takovéto lince, pouze doporučení použít pro případnou implementaci standardní rozhraní QIODevice.

Jak již bylo napsáno v předchozí kapitole, jako základ byla použita implementace QExtSerialPort. Ta však obsahovala několik „známých“ chyb a také její provedení neodpovídalo tomu, co bylo potřeba.

Problém s testováním nastal ve chvíli, kdy byla hotová první verze třídy QSerialPort i všech dalších podpůrných. Sériová linka (otevřená jako textový souboru) totiž komunikuje v kódování ASCII, QIODevice podporuje přijímání dat pouze ve formátu QByteArray a v aplikaci se pracuje s QString řetězci v kódování UTF-16.

Bylo také potřeba fyzicky vyzkoušet schopnost komunikovat pomocí sériového portu. Jelikož na základní desce vývojového počítače nebylo rozhraní RS-232, musel být nahrazen převodníkem USB-to-Serial.

Problém byl však s nalezením druhého počítače, který by měl vhodný port pro připojení převodníku. Pomocí návodu na internetu [23] se však podařilo vytvořit alespoň loopback, díky kterému šlo alespoň přijímat zpět odeslaný text.



Obrázek 9: Hardwarový loopback na USB-to-Serial convertoru

Poslední zajímavá věc se ukázala až během testování v reálném prostředí a to, že pro porty s číslem vyšším než 9 je ve WinAPI zavedeno zvláštní pojmenování.

Sériový port se totiž normálně chápe jako „COMx“, kde se nahradí místo „x“ pouze jedna číslice a nikoliv dvě. Porty COM10 a výše je tedy nutné zapisovat pomocí tzv. *adresní notace* – „\\.\COMxx”, kde „xx“ je nahrazeno číslem portu.

Díky tomu, že tato část byla z celého projektu programována jako první, a díky fyzickému testování sériové linky, byly odhaleny všechny chyby již v této fázi. Horní vrstvy aplikace se tak mohly spolehnout na bezproblémovou komunikaci se sběrnicí NVL.

5.4 Gateway Emulator a ladění driveru sběrnice

Účelem této aplikace je umožnit testování celé aplikace ve chvíli, kdy není možnost testovat na skutečné NVL. Svému účelu posloužila jak při vývoji driveru pro tuto sběrnici, tak při testování grafického rozhraní, kdy dodávala potřebná data pro zkušební provoz.

5.4.1 NVLDriver

Za pomoci GE se ukázala chyba v návrhu samotného rozhraní pro komunikaci vyšších vrstev. Díky spojení GDS s GE pomocí QLocalSocket, trvala komunikace mezi nimi výrazně kratší dobu, než tomu je ve skutečnosti.

Tím se ukázal původní návrh mechanismu odesílání a přijímání příkazů jako nedostatečný a nepružný (vrchní vrstvy zbytečně čekaly na dokončení operací).

Tak vznikl QNVLCommander, který tento nedostatek odstranil a umožnil tak spolu s QSerialPort bezproblémovou komunikaci se sběrnicí NVL.

5.4.2 GUI

Nejenom odladění komunikace, ale také správné fungování grafického rozhraní, pomohl GE otestovat. Po celou dobu implementace dodával data složení celé vlakové soupravy, módu linky a hodnoty portů jednotlivých uzlů.

Především díky poslednímu jmenovanému se podařilo otestovat dynamický prvek QDialBox, který obsahuje ručičkový tachometr (GE posílal náhodné hodnoty vybraného portu, aby tak otestoval přerušení animace ručičky a její restart s novou hodnotou).

5.4.3 Možnosti Gateway Emulátoru

Gateway Emulator tedy obsahuje následující testovací prvky:

- Připojení ke GDS pomocí rozhraní QLocalSocket nebo QSerialPort (RS-232).
- Simulace role modulu GTBA na sběrnici NVL, včetně protokolu přepnutí mezi jednotlivými stavy.
- Simulace libovolného složení vlakové soupravy, včetně podpory identifikace v šestnáctkové soustavě.
- Možnost zapsat do libovolného portu zvoleného uzlu uživatelem zadaný platný údaj v šestnáctkovém zápise.
- Logování veškeré komunikace v grafickém rozhraní – obsahující čas a identifikaci směru komunikace (příchozí/odchozí).

5.5 Testování na reálné sběrnici

V poslední fázi aplikace přišlo na řadu testování na reálné sběrnici NVL, které sloužilo především ke kontrolním účelům. Ukázalo se, že simulace pomocí GE pomohla odstranit řadu problémů, které by se ukázaly právě až při tomto testování.

Díky němu se podařilo odhalit a doladit problémy, které vznikaly při připojení aplikace k modulu GTBA. Ten je totiž při připojení v neznámém stavu, na rozdíl

od GE, který nemá stavovou logiku. Například, pokud se někdo odpojí během nějakého příkazu, který vyžaduje potvrzení. Když se pak napojí aplikace GDS, stávalo se, že linka NVL se dostávala do nedefinovaného stavu.

Proto byla před samotným prvním příkazem přidána tzv. *preambule*. Ta má právě za úkol se zotavit z takové situace a dostat modul GTBA do stavu, kdy je připraven na příjem prvního příkazu od aplikace GDS.

Během testů bylo rovněž nutné řešit situace, které nebyly dostatečně pokryty předanou originální dokumentací.

5.6 Použitý protokol

Nyní se seznámíme s textovým protokolem, který byl používán při finálním testování a na nějž je tedy aplikace optimalizována. Protokol používá pouze příkazy diagnostické linky, neboť v prototypu jsou zobrazována jen diagnostická data a není tedy nutné používat příkazy řídící linky.

Abychom výpisy odpovědí zkrátili, neuvádíme v nich hlavičku (zopakování příkazu) a koncovku (řetězec “NVL>”).

5.6.1 Seznam příkazů

Seznam příkazů, ovlivňujících nastavení a stav modulu GTBA:

Příkaz	Funkce
DM	Změna role modulu GTBA na linkového mastera
DS	Změna role modulu GTBA na slave uzel
DP	Nastavení módu modulu GTBA na případech sběrnice
IR	Změna identifikace modulu GTBA – zadání nové řady vozu
IC	Změna identifikace modulu GTBA – zadání nového čísla vozu

Seznam příkazů, slouží k získání informací ze sběrnice NVL:

Příkaz	Funkce
U	Výpis konfigurace (seznamu uzlů, jejich role, stav) linky a periody obnovy portů uzlů.
P<u>:<p>	Zobrazení obsahu portu **<p>** uzlu **<u>**
P<u>:<p>+	Zobrazení obsahu portu **<p>** uzlu **<u>** a přidání portu do pravidelného plánu

5.6.2 Příkaz DM

Odpověď na tento příkaz může mít různé podoby, závislé od stavu GTBA před příkazem a konfigurace uzlů na vlakové lince.

Pakliže je GTBA linkovým masterem již před příkazem DM, vypadá odpověď takto:

```
Diagnosticka = Master
```

Pokud je GTBA v módu Slave, ale na lince není žádný jiný master, je aplikace GDS dotázána, zda chce opravdu převzít řízení linky. GDS vyšle kladnou odpověď (písmeno A) a celé to vypadá takto:

```
Opravdu chces prevzit rizeni [A/N] ? A
Diagnosticka = Master
```

Je-li v roli Slave, ale na lince už nějaký existuje, je GDS dotázána, zda chce vynutit převzetí, zrušit požadavek nebo počkat, než se aktuální master sám vzdá své role. GDS na takovýto dotaz reaguje tím, že vyšle vynucovací odpověď (písmeno V). Odpověď je poté:

```
Linku ridi jiny Master !
Cekat / Vnutit se / Nemenit [C/V/N] ? V
Diagnosticka = Master
```

Pokud byl modul GTBA v pasivním režimu a je přepnut do master módu, je situace podobná, jako při přepnutí ze Slave módu:

```
Opravdu chces vlozit Master uzel do linky [A/N] ? A
Diagnosticka = Master
```

5.6.3 Příkaz DS

Příkaz DS je na tom s reakcemi podobně, jako příkaz DM, jenom s tím rozdílem, že není problém, pokud jsou na lince dva a více Slave uzlů.

Pakliže je modul GTBA v módu Slave, je jeho odpověď:

```
Diagnosticka = Slave
```

Pokud je modul v režimu Master, je opět aplikace dotázána na to, zda chce opravdu přepnout do režimu Slave. GDS odpoví kladně (písmeno A):

```
Opravdu chces odevzdat rizeni [A/N] ? A
Diagnosticka = Slave
```

Pokud byl modul v pasivním režimu a je přepnut do Slave módu, je reakce obdobná jako u příkazu DM:

```
Opravdu chces vlozit Slave uzel do linky [A/N] ? A
Diagnosticka = Slave
```

5.6.4 Příkaz DP

Při přepnutí do Pasivního módu (příposlech) z libovolného jiného je odpověď vždy stejná:

```
Diagnosticka = Priposlech
```

5.6.5 Příkazy IR a IC

Na tyto příkazy modul GTBA neodpoví žádnou odpovědí – tedy kromě standardní hlavičky (opakování příkazu) a koncovky (řetězce „NVL>”).

5.6.6 Příkaz U

Odpovědí na tento příkaz je seznam uzlů na lince ve tvaru, jako je vidět na výpisu níže, zakončený výpisem periody přenosu portů (délku přenosového plánu v ms).

```
[uu] = rrr(v).ccc-k RR SS
[uu] = rrr(v).ccc-k RR SS
…
Perioda X: ttt.0 ms
```

Vysvětlivky:

Zkratka	Popis
uu	je číslo uzlu
rrr	je řada vozu
v	je verze řady vozu
ccc	je identifikační číslo
k	je kontrolní číslice
RR	je role uzlu na lince: • ** - je nezávislý uzel • M - je master uzel • Sa - je slave uzel oslovený ze strany A • Sb - je slave uzel oslovený ze strany B
SS	je upřesnění stavu uzlu: • * - je master • T - je v T spojení • Pm - je přítomnost master uzlu • Ps - je přítomnost slave uzlu • nebo nic z toho
X	je písmeno **A** nebo **B** podle toho, jestli je GTBA osloven ze strany A nebo B. Pokud je GTBA v roli master, tak je automaticky A
ttt	je perioda obnovy portů linky

5.6.7 Příkaz P

Odpověď na tento příkaz je stejný pro obě varianty (ať už normální nebo pro zařazení do plánu obnovy portů).

```
UU:PP:00 bb bb bb bb bb  bb bb bb bb bb zzzzz zzzzz
UU:PP:10 bb bb bb bb bb  bb bb bb bb bb zzzzz zzzzz
UU:PP:20 bb bb bb bb bb  bb bb bb bb bb zzzzz zzzzz
UU:PP[dd] prijat =     n x, perioda =         T s
```

Vysvětlivky:

Zkratka	Popis
UU	je číslo uzlu
PP	je číslo portu
bb	Byte
Z	znak (odpovídající příslušnému bytu) nebo tečka pro nezobrazitelný
dd	počet platných bajtů v portu (měřeno v trojbajtech)
n	je počet přijetí portu
T	je perioda příjmu portu

6 Praktické využití aplikace

Obsahem této kapitoly je popis praktického využití výsledků této práce. Přestože se jedná pouze o prototyp, GDS je plně funkční a lze jej použít v reálném provozu. V závěru kapitoly jsou podrobněji popsána rozhraní obou aplikací spolu s ukázkami.

6.1 Zamýšlené využití aplikace v praxi

Nyní se seznámíme s možnostmi a prostředími, kde lze vytvořené aplikace využít. Obecně lze aplikaci využít k libovolnému účelu, neboť díky vestavěné příkazové řádce umožňuje odeslat i příkazy, které nemají k tomu určený grafický prvek v uživatelském rozhraní GDS.

6.1.1 Depo provozovatele vozidel

Provozovatel vozidel potřebuje občas zjistit, co se na vlakové sběrnici děje - například v případě, že vůz hlásí chybový stav. Tehdy je vozidlo odtaženo do depa a tam je v tzv. single módu (pouze jedno vozidlo, nikoliv celá souprava) chyba odstraněna. Pro správnou diagnostiku a následné testování je potřeba vidět obsah portů, což je jedna ze základních funkcionalit aplikace GDS.

Pokud se vozy nemohou mezi sebou domluvit, je potřeba zachytit jejich komunikaci po vlakové lince. Aby bylo možné zachytit komunikaci a následně ji podrobit arbitráži, uvede se GTBA do módu příposlechu a díky přidání implementace QLoggeru lze vše automaticky zachytit a uložit do souboru.

6.1.2 Továrna výrobce vozidel

Podobně jako provozovatelé, chtějí i výrobci vědět, zda-li všechny systémy jimi vyráběných vozů fungují správně. Díky připojení modulu GTBA k vlakové lince a připojení k PC lze nejen zjistit správnost fungování zařízení ve voze, ale také otestovat komunikaci s jiným vozem, emulovaným modulem.

Pomocí grafického rozhraní aplikace je možné zobrazit např. tachograf vozu (tedy zobrazit údaj v příslušném portu) a lze tak detekovat nesoulad mezi hodnotami zobrazenými na skutečném tachografu i v aplikaci. Podobné je to i s ostatními prvky ovládacího panelu.

6.1.3 Laboratoř výrobce vozidel

Vzhledem k tomu, jakým tempem se vyvíjejí technologie kupředu, je nutné občas přizpůsobit chování jednotlivých vozidel a jejich komunikaci po vlakové sběrnici. Taková změna musí být v souladu s ostatními vozy a zároveň zpětně kompatibilní. Poté je potřeba změny otestovat alespoň na simulovaných datech, aby se předešlo některým nechtěným efektům v reálném provozu.

K tomu lze využít modulu GTBA připojeného k několika řídícím jednotkám, neboť pomocí konzole jde změnit obsah portů modulu a lze tak simulovat připojení vozu s novým rozšířením do vlakové soupravy.

6.2 Ukázka práce s aplikaci GDS a GE

V této části kapitoly se budeme zabývat popisem grafického uživatelského rozhraní obou aplikací. Pro co nejpřesnější popis zavedeme následující konvenci:

- *Brána* – budeme chápat jako modul GTBA, ke kterému se připojuje aplikace GDS
- *Modul* – statická část nebo oblast grafického rozhraní aplikace
- *Dialog* – modální okno, které je součástí rozhraní aplikace
- *Dynamický widget* – dynamicky vkládaná část nebo oblast grafického rozhraní aplikace

6.2.1 Spojení aplikace GDS se sběrnicí NVL a nebo s GE

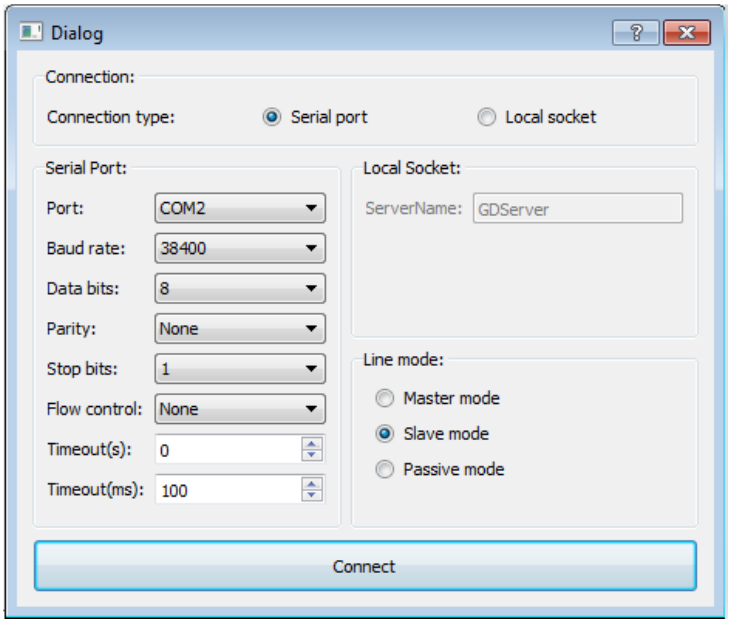
Obě aplikace obsahují podobné rozhraní pro zahájení komunikace, přesto však mají podstatné rozdíly. Pakliže se spojují spolu pomocí QLocalSocket, je nutné nejprve nastartovat server QLocalServer v aplikaci GE a až poté se připojit k němu v aplikaci GDS.

Zatímco modul pro připojení je v aplikaci GE součástí grafického rozhraní, v aplikaci GDS je zobrazen jako modální dialog. Ten lze vyvolat pomocí nabídky v menu Program, případně pomocí klávesových zkratek Alt+C (připojení) a Alt+D (odpojení).

Obrázek 10: Ukázka modulu připojení v aplikaci GE

V aplikaci GDS je modul připojení obohacen ještě o možnost zvolit roli Brány na diagnostické lince, která se nastaví po úspěšném provedení. Pro připojení aplikace GDS ke sběrnici NVL se používá sériový port s následující konfigurací (která je navíc v dialogu přednastavena):

- Baud rate: 38400
- Data bits: 8
- Parity: none
- Stop bits: 1
- Flow control: none



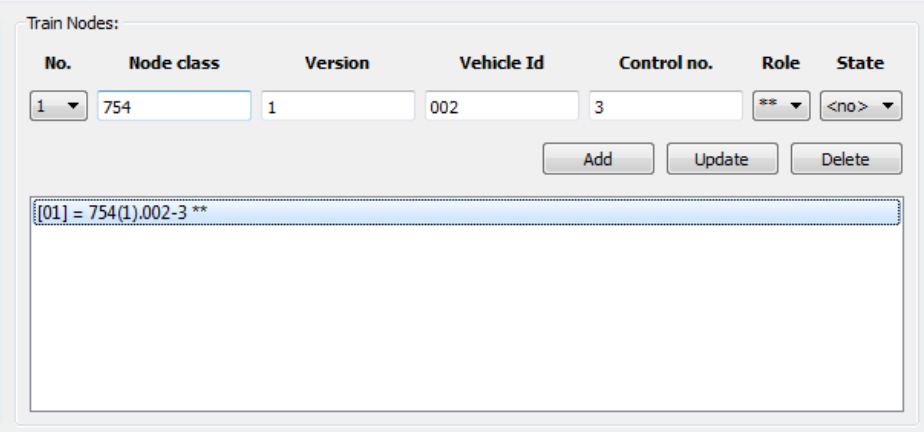
Obrázek 11: Dialog pro připojení aplikace GDS

Aplikace GE má ve svém rozhraní modul pro zobrazení role diagnostické linky, aby tak bylo možné sledovat nastavení GTBA, pokud je emulován pomocí této aplikace.

6.2.2 Úprava složení vlakové soupravy v aplikaci GE

Jedna ze dvou základních operací, které umožňuje aplikace Gateway Emulator je úprava složení vlakové soupravy. Tu lze provádět v modulu pro editaci vlakových uzlů, zobrazeného na následujícím obrázku.

Ve vrchní části je oblast pro zadání identifikace nového popř. editovaného uzlu sběrnice, pod ní je okno obsahující seznam dosud přidaných uzlů. Jestliže je vybrán některý z uzlů, je jeho identifikace nahrána do vrchní části a lze ji upravit a uložit (pomocí tlačítka Update) nebo celý uzel smazat (stisknutím tlačítka Smazat). Jestliže není vybrán žádný uzel, jsou obě tlačítka nedostupná.



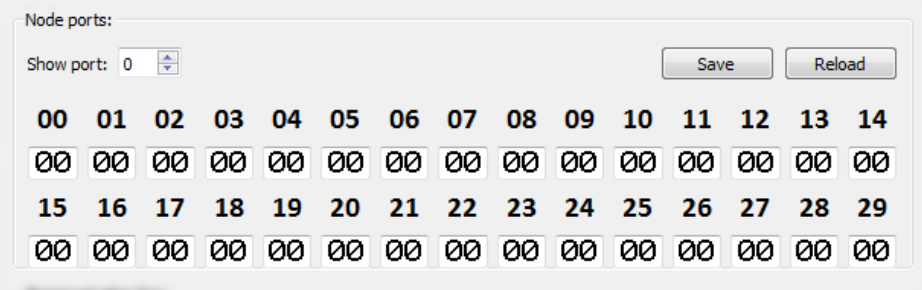
Obrázek 12: Modul pro editaci uzlů sběrnice v GE

6.2.3 Obsah portů vybraného uzlu

Druhou základní funkcí aplikace GE je úprava dat portů vybraného uzlu sběrnice. Tuto funkci zastává modul, jehož rozhraní je zobrazeno na obrázku níže. Je-li vybrán některý z uzlů v seznamu nad ním, je tato část zpřístupněna a automaticky nahrán zvolený port daného uzlu.

Zobrazený port lze změnit pomocí zadání čísla požadovaného portu v horní části tohoto modulu (hodnoty jsou omezeny na rozsah 0 – 31 dle specifikace sběrnice NVL).

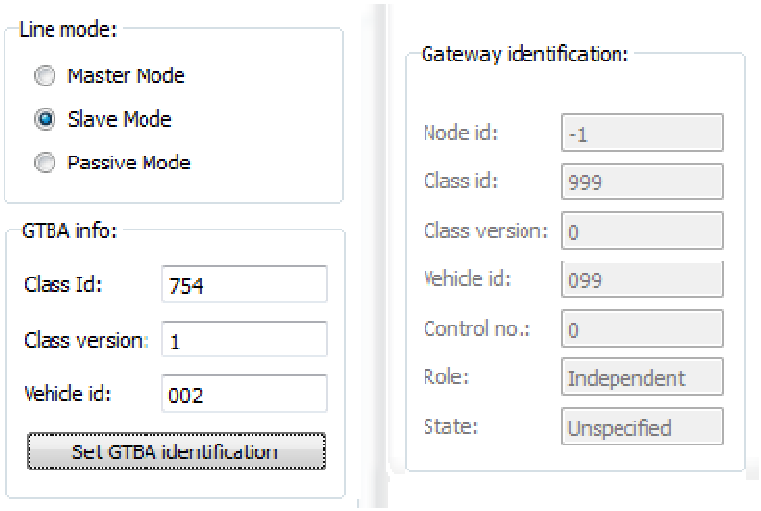
Pod touto částí je zobrazen obsah jednotlivých bytů daného portu a lze jej libovolně upravovat (omezení je samozřejmě na hodnoty od 0 – 255, zapsané v šestnáctkové soustavě). Upravená data lze uložit (pomocí tlačítka Save) nebo obnovit původní obsah portu (stisknutím tlačítka Reload).



Obrázek 13: Modul pro úpravu portů v aplikaci GE

6.2.4 Ovládání modulu GTBA a jeho identifikace v aplikaci GDS

Aplikace GDS je rozdělena do tří záložek – jednou z nich je i záložka Gateway. Ta slouží pro nastavení vlastností GTBA linky NVL a to především role, ve které Brána vystupuje na lince jako uzel, a také nastavení její identifikace (včetně zobrazení současné identifikace).



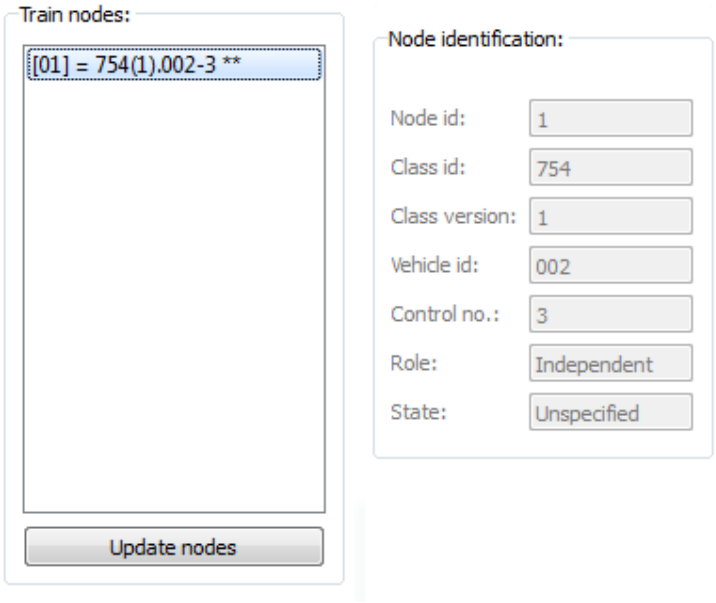
Obrázek 14: Modul pro správu vlastností modulu GTBA

Kromě výše zmíněných modulů obsahuje ještě okno pro logování komunikace mezi aplikací a sběrnicí resp. emulátorem a také příkazovou řádku, do které lze zapisovat textové příkazy pro modul GTBA.

6.2.5 Získání informací o složení linky a jednotlivých uzlech

Další záložka, jménem Train Nodes je určena pro správu uzlů na lince NVL. Na ní je modul pro získání složení linky, skládající se ze seznamu aktuálních uzlů a tlačítka pro znovuzaslání požadavku na získání aktuální konfigurace.

Dále záložka obsahuje modul pro podrobný výpis identifikace uzlu, podobný jako na záložce Gateway. Pokud je některý uzel aktivní, tento modul zobrazuje podrobné informace o něm včetně jeho role a stavu na lince.

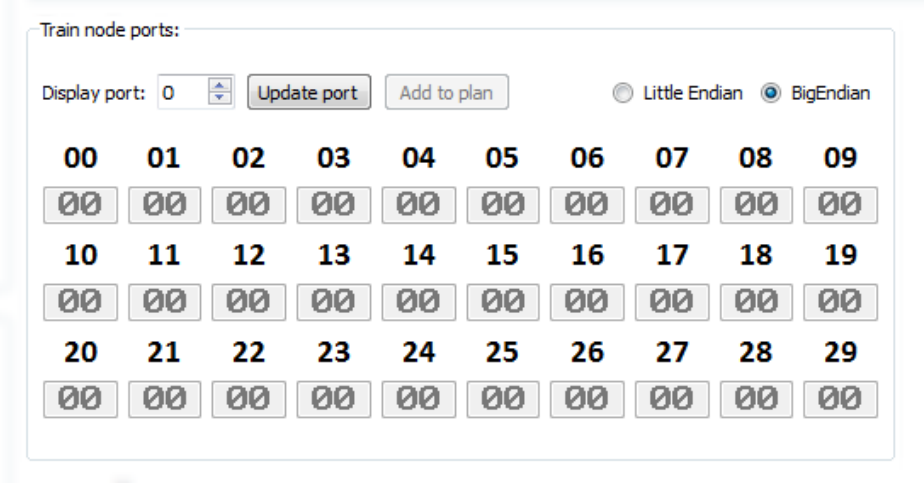


Obrázek 15: Modul určený pro získání složení linky a modul identifikace aktivního uzlu

6.2.6 Zobrazení obsahu portu a jeho obnovování

Podobně jako v aplikaci GE, také v GDS existuje modul pro zobrazení obsahu portů aktuálně zvoleného uzlu. Hlavní rozdíl je v tom, že nelze obsah editovat – slouží pouze pro informaci. Navíc je zde přidána možnost zvolit zobrazení zdrojových dat v Little Endianu nebo Big Endianu (standardně jsou data přijímána v Little Endianu).

Modul obsahuje také dvě tlačítka – pro zaslání požadavku na okamžitou obnovu dat (tlačítko Update port) a tlačítko pro zařazení portu do pravidelného plánu přenosu (tlačítko Add to plan). Jak již bylo popsáno výše, zařadit port do plánu lze pouze, pokud modul GTBA je v roli Master.



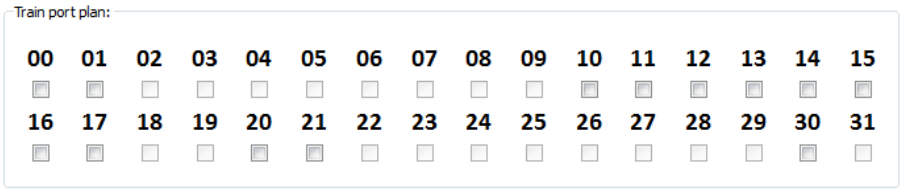
Obrázek 16: Modul pro zobrazení portu v aplikaci GDS

6.2.7 Pravidelný update portu

Aby nemusel uživatel programu stále manuálně obnovovat obsah portu, je zde přidán modul pro správu automatického přenosu portů uzlu z Brány do aplikace GDS. Na rozdíl od přidání portu do plánu (tedy přenosu portu z uzlu při pravidelné cyklické fázi přenosu), tento přenos nijak nezasahuje do plánu GTBA a lze jej provozovat i ve Slave roli.

Jelikož nemusí být všechny porty v každém voze využity, je pro známé řady vozů výběr omezen pouze na využité porty. Obecně lze definovat libovolné nastavení portů vozů, aktuálně jsou k dispozici data o těchto vozech:

- Lokomotiva - řada 754, verze 1
- Motorový vůz - řada 843, verze 0
- Řídící vůz - řada 943, verze 0
- Vložený vůz - řada 043, verze 0
- Vložený vůz - řada 052, verze 2



Obrázek 17: Oblast pro zadání automatické obnovy portu pro lokomotivu ř. 754

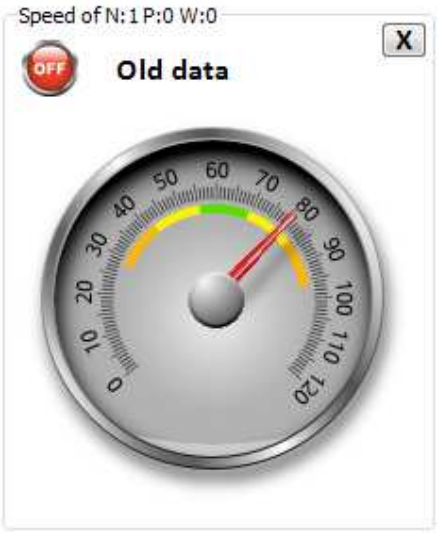
6.2.8 Přidání dynamického widgetu QDialBox

Další neméně zajímavou částí je oblast pro vložení dynamicky vytvářeného widgetu QDialBox, který je vložen na samostatnou třetí záložku aplikace.

Pro získání zdroje dat je nutné vyplnit číslo portu aktuálně zvoleného uzlu a číslo wordu, ze kterého se budou data brát. Zároveň lze tato data ještě modifikovat a to nejdříve bitovým posunem a poté pomocí bitového AND masky, kterou lze rovněž zadat v tomto modulu.

Obrázek 18: Oblast pro vytvoření dynamického widgetu QDialBox

Třetí záložka je dimenzována tak, aby byla schopna najednou zobrazit až deset vytvořených dialogů. Pokud už nejsou některé z nich potřeba, lze je jednoduše zavřít pomocí křížku v pravém horním rohu widgetu.



Obrázek 19: Ukázka dynamicky vkládaného widgetu QDialBox

7 Otevřené problémy a budoucí vývoj aplikace

Obsahem této kapitoly je popis otevřených problémů, které v aplikaci po jejím dokončení zůstávají. Zároveň jsou i nastíněna možná budoucí rozšíření, která by jistě přispěla k lepšímu a snadnějšímu používání aplikace.

7.1 Otevřené problémy

Jedním ze zásadních nedořešených problémů je re-inaugurace sběrnice NVL. Pakliže k ní dojde (např. z důvodu změny složení vlakové sběrnice nebo převzetí Master role), modul GTBA nedá o této změně aplikaci nikterak vědět. Modul totiž sám od sebe neposílá žádná data, pouze odpovídá na dotazy, které mu aplikace posílá.

Jsou-li v aplikaci aktivovány dynamické moduly, které mají jako zdroj dat nějaký cílový port, po re-inauguraci mohou nastat tři případy:

- Uzel má stejné číslo a tedy modul ukazuje na správná data,
- použité číslo uzlu nebo portu už není platné a tedy po vypršení platnosti dat bude aplikace indikovat zastaralá data,
- uzel i port bude platný, avšak bude se jednat fyzicky o jinou řadu vozu, která může mít v tomto portu např. tlak vzduchu v brzdách. Údaj, který bude zobrazen, tedy nebude odpovídat původnímu úmyslu.

Re-inaugurace by mohla na vlakové sběrnici nastat prakticky kdykoliv např. v zastávce, pokud se změní složení vozů a během jízdy v případě, že některý vůz na chvíli vypadne z komunikace a pak se znovu „probudí“. Otázka tedy je, jak detekovat a jak se zachovat, jestliže tuto změnu zachytíme.

Jiným otevřeným problémem jsou provozní proměnné ve vlaku. Když budou dostupné specifikace všech vozů, které danou sběrnici podporují, bylo by lepší při zobrazení adresovat proměnnou (např. rychlost) než určitou paměťovou oblast (port nebo jeho část). Takto bychom mohli např. najednou zobrazit nějakou charakteristiku všech vozů zapojených do vlakové soupravy pomocí jednoho tlačítka.

Dalším problémem je podpora sofistikovaných komunikačních protokolů. Sériová linka, stejně jako Ethernet, používají textový protokol, avšak sběrnice jako např. CAN používají slovníkové příkazy pro řízení komunikace. Při současném návrhu aplikace není proto možné jednoduše zakomponovat ovladač na sběrnici CAN, neboť bychom museli udělat mezi nejnižšími dvěma vrstvami další mezivrstvu, starající se o slovník příkazů.

Tyto a další problémy by měly být v budoucnu vyřešeny, ovšem jejich složitost si žádá rozsah řešení přesahující tuto práci. Některé z postupů se mohou dokonce promítnout do samotné implementace modulu GTBA, případně až do návrhu sběrnice NVL.

7.2 Možnosti rozšíření

Samotné zadání již na začátku počítalo s tím, že aplikace může být v budoucnu rozšířena a tomu byl přizpůsoben i její návrh. Ten je rozdělen do několika vrstev, jež se dají poměrně snadno a efektivně doplnit o nové součásti, bez nutnosti měnit stávající kusy aplikace.

Jak již bylo popsáno výše, jedním z možných rozšíření bude implementace správy provozních proměnných ve vlakové sběrnici. Lze také rozšířit podporu stávající sběrnice NVL o ovládání řídící linky nebo o příkazy diagnostické linky, které zatím nevyužívá.

Pro zobrazení jiného typu dat lze implementovat další dynamicky vkládané prvky rozhraní, které splňují jeden ze dvou navržených interfaců. Díky novým možnostem frameworku Qt 4.7 by mohlo jít např. vytvořit skriptovací prostředí, kde by se dle potřeby daly tyto prvky upravit bez potřeby zkompilovat znovu celou aplikaci.

Lze rovněž rozšířit sadu rozhraní, kterými je aplikace schopná komunikovat se sběrnicí NVL nebo i jinou. Jestliže bude v plánu přidat k současné sériové lince i jiné sofistikované komunikační rozhraní, bude muset být řešen problém mezivrstvy komunikace, nastíněný výše.

Díky své koncepci je možné tato a mnohá další rozšíření poměrně lehce přidat do příslušné vrstvy stávající aplikace a zvětšit tak i její možnosti a pole působnosti.

8 Závěr

Výsledek této práce je aplikace Graphics Diagnostics System, která umožňuje zobrazovat diagnostická data na sběrnici Narodní Vlakové Linky, na kterou se napojuje pomocí hardwarového modulu GTBA. Součástí řešení je i pomocná aplikace Gateway Emulator, která umožňuje zjednodušeně simulovat modul GTBA a uzly na vlakové sběrnici, včetně jejich dat.

Komunikace s modulem GTBA je umožněna pomocí rozhraní sériové linky s rozhraním RS-232, případně pomocí převodníku s rozhraním USB. Pakliže se aplikace spojuje s Gateway Emulatorem, lze pro jejich spojení použít i Named Pipe v prostředí Microsoft Windows resp. BSD Sockets (Linux, BSD, …) a tedy provozovat obě aplikace současně na jednom počítači.

Aplikace GDS slouží především pro zobrazení složení uzlů sběrnice NVL, obsahu jejich portů, které dokáže přehledně zobrazit pomocí grafických prvků. Součástí je i zobrazení protokolu komunikace mezi GDS a sběrnicí NVL, který se automaticky zároveň ukládá do textového souboru na lokálním disku.

Ve svém zobrazení aplikace zohledňuje především platnost zobrazovaných dat, aby tak uživatel dal věrohodný obraz dění na sběrnici v souladu s platnou normou. V aplikaci je proto navržen systém, který umožňuje zdrojová data pro zobrazení automaticky obnovovat a to až v řádu desítek milisekund.

Aplikace využívá nejmodernější technologie Qt 4.6.2, díky čemuž ji lze provozovat na libovolném operačním systému Microsoft Windows 2000 a novější. Jedinou překážkou v provozování na jiných platformách je implementace ovladače sériové linky, který je prozatím pouze pro platformu Microsoft Windows.

Přestože se jedná o prototyp aplikace, je plně funkční a lze jej využít v běžném provozu, což bylo ověřeno testy v reálném prostředí NVL. Ty splnily naše očekávání a umožnily sladit testovací prostředí Gateway Emulatoru s tím skutečným.

9 Seznam zdrojů

[1] PhDr. Zbyněk Zlinský, *Motorové vozy na našich kolejích: řada 843*, 28. prosinec 2007, http://www.vlaky.net/zeleznice/spravy/002216-Motorove-vozy-na-nasich-kolejich-rada-843.asp

[2] *Vyhláška Ministerstva Dopravy České Republiky č. 173/1995 Sb.*, http://www.mdcr.cz/NR/rdonlyres/D18D24A3-C24F-4663-9AA2-E8057E9C34FC/0/17395uplzn.rtf

[3] *Lokomotiva*, 14. srpen 2010, http://cs.wikipedia.org/wiki/Lokomotiva

[4] *Motorový vůz*, 16. březen 2010, http://cs.wikipedia.org/wiki/Motorový_vůz

[5] Vladimír Menšík, *TNŽ 28 0080 - Kolejová vozidla železniční. Vnější označení hnacích vozidel a vozů osobní dopravy*, 1. červen 2000

[6] Josef Balek, *SŽDC (ČD) D 2 - Předpis pro organizování a provozování drážní dopravy*, 13. březen 1997

[7] *IEC 61375 and UIC 556-international standards for train communication.* In *Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st*, pages 1581 - 1585 vol. 2., January 2000

[8] "*Glossary Of Common Railroad Terms: C*". Kalmbach Publishing. Retrieved 2008-01-29

[9] *Electronic railway equipment - Train Communication Network - IEC 61375 – Part 2: Real Time Protocols*, December 1999, IEC

[10] *Electronic railway equipment - Train Communication Network - IEC 61375 – Part 3: Multifunction Vehicle Bus*, September 1999, IEC

[11] *Electronic railway equipment - Train Communication Network - IEC 61375 – Part 4: Wire Train Bus*, September 1999, IEC

[12] *Electronic railway equipment - Train Communication Network - IEC 61375 – Part 2-5: ETB - Ethernet Train Backbone Ed. 1.0*, January 2010, IEC

[13] *Electronic railway equipment - Train Communication Network - IEC 61375 – Part 3-4: ECN - Ethernet Consist Network Ed. 1.0*, January 2010, IEC

[14] MSV elektronika s.r.o. a spolupracující kolektiv, *Pre TNŽ 28 1500 - Národní vlaková linka ČD - Násobné řízení a přenos informací po vlaku*, 6. únor 2009

[15] *Nokia Qt 4 – Cross-platform application and UI toolkit*, TrollTech, 1999, http://qt.nokia.com/

[16] *QObject Class Reference*, Qt 4.6 Documentation, Nokia Corporation 2010, http://doc.qt.nokia.com/4.6/qobject.html

[17] *QWidget Class Reference*, Qt 4.6 Documentation, Nokia Corporation 2010, http://doc.qt.nokia.com/4.6/qwidget.html

[18] *Signals and slots*, Qt 4.6 Documentation, Nokia Corporation 2010, http://doc.trolltech.com/4.6/signalsandslots.html

[19] Stefan Sander, Michal Policht, Brandon Fosdick, *QextSerialPort - cross-platform serial port class*, http://qextserialport.sourceforge.net/

[20] *Serial Communications in Win32*, Microsoft Corporation, 2010, http://msdn.microsoft.com/en-us/library/ms810467.aspx

[21] *Global Qt Declarations*, Qt 4.6 Documentation, Nokia Corporation 2010, http://doc.qt.nokia.com/4.6/qtglobal.html

[22] Memory Leak Detection in VS, Qt Center, March 2008, http://www.qtcentre.org/wiki/index.php?title=Memory_Leak_Detection_in_VS

[23] *How to Do a Serial Loopback Test*, National Instrumental, May 2010, http://zone.ni.com/devzone/cda/tut/p/id/3450