FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	ETCS - Modul pro komunikaci mezi EVC a RBC
Student:	Daria Roshchupkina
Vedoucí:	Ing. Jan Matoušek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

ETCS (European Train Control System) je jednotný celoevropský vlakový zabezpečovací systém. EVC (European Vital Computer) je centrální počítač vozidlové části systému. RBC (Radio Block Centre) je centrální bezpečnostní jednotka, která prostřednictvím rádia přijímá informace o poloze vlaku a vysílá oprávnění k jízdě do vlaků.

Cílem této práce je vytvořit zprávy pro komunikaci mezi EVC a RBC a implementovat komunikaci s RBC ze strany EVC pro projekt simulátoru ETCS vyvíjeného zčásti na FIT.

Pokyny k vypracování:

1. Analyzujte pravidla komunikace mezi EVC a RBC podle verzí 2.3.0, 3.4.0, 3.6.0 SUBSET-026 System Requirement Specification.
2. Porovnejte pravidla této komunikace mezi těmito 3 verzemi.
3. Na základě analýzy a metodami softwarového inženýrství navrhněte vhodnou reprezentaci zpráv a komunikační modul pro EVC.
4. Implementujte prototyp zpráv a prototyp modulu pro EVC.
5. Podrobte prototypy vhodným testům.
6. Proveďte zhodnocení Vaší práce.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 21. ledna 2022 v Praze.

Bakalářská práce

ETCS - MODUL PRO KOMUNIKACI MEZI EVC A RBC

Daria Roshchupkina

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Matoušek
10. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Ráda bych poděkovala Ing. Janovi Matouškovi za odborné vedení a konzultace po celou dobu mé práce. Dále děkuji Ing. Jiřímu Chludilovi a doc. Ing. Martinovi Lesovi, Ph.D.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

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

V Praze dne 10. května 2022 .....

Abstrakt

Tato práce se zabývá návrhem knihoven proměnných, packetů a zpráv, nutných pro komunikaci mezi EVC a RBC a komunikačního modulu pro komunikaci mezi EVC a RBC na straně EVC, a to pro verze 2.3.0, 3.4.0 a 3.6.0 systémových požadavků kladených na ETCS v rámci SUBSET-026 vypracovaného ERA, pro simulátor ETCS. Tato práce také popisuje implementaci knihoven proměnných pro každou z verzí 2.3.0, 3.4.0 a 3.6.0 a knihoven vybraných packetů pro jednotlivé verze, vzniklou na základě tohoto návrhu. Zároveň popisuje rozdíly a podobnosti proměnných, packetů a zpráv jazyka ETCS, nutných pro komunikaci mezi moduly EVC a RBC v simulátoru ETCS, a komunikace s RBC na straně EVC mezi verzemi 2.3.0, 3.4.0 a 3.6.0.

Klíčová slova vlakový simulátor, ETCS, komunikace mezi EVC a RBC, jazyk ETCS, proměnné jazyka ETCS, zprávy jazyka ETCS, packety jazyka ETCS

Abstract

This thesis is concerned with the design for the ETCS simulator of libraries of variables, packets and messages, which are necessary for communication between EVC and RBC, and with the design for this simulator of the communication module for communication between EVC and RBC on the EVC side, for versions 2.3.0, 3.4.0 and 3.6.0 of the system requirements for ETCS defined by the SUBSET-026 developed by ERA. This work also describes the implementation of the libraries of variables for each of versions 2.3.0, 3.4.0 and 3.6.0 and the implementation of libraries of selected packets for each version, which are based on this design. It also describes the differences and similarities of ETCS language's variables, packets, and messages, which are required for communication between EVC and RBC modules in the ETCS simulator, and communication with RBC from the EVC side between versions 2.3.0, 3.4.0 and 3.6.0 of the ETCS system requirements.

Keywords train simulator, ETCS, communication between EVC and RBC, ETCS language, ETCS language's variables, ETCS language's messages, ETCS language's packets

Seznam zkratek

ATP	Automatic train protection
BTM	Balise Transmission Module
DMI	Driver-machine interface
ERA	European Union Agency for Railways
ERTMS	The European Rail Traffic Management System
ETCS	European Train Control System
EVC	European Vital Computer
GSM-R	Global System for Mobile Communications – Railway
JRU	Juridical Recording Unit
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
ODO	Odometry
RBC	Radio Block Centre
STM	Specific Transmission Module
TIU	Train Interface Unit
TŽK	Tranzitní železniční koridor

Kapitola 1
Úvod

Vlakový simulátor ČVUT s komponentou ETCS (ETCS simulátor) vzniká v rámci společného projektu Fakulty dopravní a Fakulty informatiky ČVUT za účelem realizace prototypu vlakového simulátoru, který by měl umožnit systematicky výcvik strojvůdců v ČR.

V současné době se v celé EU zavádí systém řízení železniční dopravy ERTMS, jehož součástí je ETCS. ERTMS přináší značné výhody. Za prvé, tento systém výrazně zvyšuje bezpečnost železničního systému jak pro cestující, tak pro strojvedoucí. Za druhé, zvyšuje spolehlivost a dochvilnost spojů, tedy ukazatelů, které jsou klíčové pro osobní i nákladní dopravu. Za třetí, je navržen tak, aby byl kompatibilní v celé Evropské unii. Díky tomu zlepšuje přeshraniční spojení mezi jednotlivými státy. Za čtvrté, umožňuje vlakům jezdit vyšší rychlostí, kdy zvyšuje maximální možnou rychlost až na 500 km/h. Navíc tento řídící systém zvyšuje kapacitu tratí. [1]

V ČR se plánuje provoz s výhradním použitím ERTMS na hlavních tratích ve správě Správy železnic, s.o. již od roku 2025. Z tohoto důvodu v současné době musí všichni dopravci vybavit svá vozidla palubní částí ETCS a vyškolit strojvedoucí v obsluze tohoto systému. Realizace odpovídajícího simulátoru je proto vysoce aktuální.

Začala jsem pracovat na vlakovém simulátoru s komponentou ETCS v rámci předmětu BI-SP1 a pokračovala jsem v tom v předmětu BI-SP2. V rámci těchto předmětů jsem se zabývala komunikací mezi moduly EVC a RBC v simulátoru ETCS. Chtěla jsem pokračovat ve výzkumu, vývoji a zlepšování komunikace mezi těmito moduly. Z výše uvedeného důvodu jsem se rozhodla pro volbu tématu ETCS – Modul pro komunikaci mezi EVC a RBC.

Tato bakalářská práce je součástí projektu Vlakový simulátor ČVUT s komponentou ETCS.

Výsledkem této práce je navrhnout proměnné, packety a zprávy nutné pro komunikaci mezi moduly EVC a RBC v simulátoru ETCS a komunikační modul pro komunikaci mezi EVC a RBC na straně modulu EVC pro verze 2.3.0, 3.4.0 a 3.6.0 systémových požadavků kladených na ETCS v rámci SUBSET-026 vypracovaného ERA. Výsledkem této práce je také implementace prototypů proměnných a vybraných packetů. Prototypy jsou implementovány pro všechny výše uvedené verze systémových požadavků.

Teoretická část práce analyzuje rozdíly v komunikace mezi EVC a RBC pro výše uvedené verze systémových požadavků a analyzuje dosavadní implementaci komunikace mezi EVC a RBC v simulátoru ETCS. Tato část také zahrnuje analýzu a tvorbu funkčních a nefunkčních požadavků proměnných, packetů, zpráv a komunikačního modulu. V praktické části je pak návrh založený na požadavcích z teoretické části. Po návrhu následuje implementace, která popisuje fungování prototypů proměnných a packetů. Na závěr se praktická část zabývá testováním těchto prototypů.

Kapitola 2
Cíl práce

Cílem teoretické části práce je analyzovat dosavadní komunikaci mezi moduly EVC a RBC v simulátoru ETCS. Cílem rešeršní části je také určit rozdíly a podobnosti proměnných, packetů a zpráv jazyka ETCS, nutných pro komunikaci mezi moduly EVC a RBC v simulátoru ETCS, a komunikace s RBC na straně EVC mezi verzemi 2.3.0, 3.4.0 a 3.6.0 systémových požadavků kladených na ETCS v rámci SUBSET-026 vypracovaného ERA. Cílem teoretické části je také na základě této analýzy definovat funkční a nefunkční požadavky proměnných, packetů, zpráv nutných pro komunikaci mezi moduly EVC a RBC v simulátoru ETCS a komunikačního modulu pro komunikaci mezi moduly EVC a RBC na straně EVC v simulátoru ETCS.

Cílem praktické části je v souladu s funkčními a nefunkčními požadavky navrhnout proměnné, packety, zprávy a komunikační modul pro komunikaci mezi EVC a RBC na straně modulu EVC pro verze 2.3.0, 3.4.0 a 3.6.0 systémových požadavků kladených na ETCS. Dalším cílem praktické části je na základě návrhu implementovat prototypy proměnných a vybraných packetů a otestovat je.

Kapitola 3
Analýza

Na začátku této kapitoly bude stručně popsán systém ERTMS a jeho součást ETCS. Dále bude popsáno použití ETCS v ČR. Potom bude krátce popsán simulátor ETCS. Dále bude analyzována dosavadní komunikace mezi moduly EVC a RBC v simulátoru ETCS, která je implementována pouze pro verzi 2.3.0 systémových požadavků. Potom budou analyzovány rozdíly v systémových požadavcích na zprávy, packety a proměnné, pomocí kterých komunikují výše uvedené moduly a komunikace s RBC na straně EVC, mezi verzemi 2.3.0, 3.4.0 a 3.6.0. Na konci této kapitoly jsou definovány funkční a nefunkční požadavky kladené na komunikační modul pro komunikaci mezi EVC a RBC na straně EVC a na zprávy, packety a proměnné.

3.1 ERTMS

ERTMS je projekt EU určený ke sjednocení evropských železnic. Cílem ERTMS je nahradit různé národní systémy řízení a zabezpečení vlaků v Evropě. [2] Zavedení ERTMS umožní vytvoření celistvého evropského železničního systému a zvýší konkurenceschopnost evropských železnic. [2] ERTMS se skládá ze dvou součástí – ETCS a GSM-R:

- **ETCS** je systém ATP, který nahradí dosavadní národní systémy ATP. ETCS sleduje pohyb vlaku, zajišťuje jeho bezpečný provoz a současně zobrazuje informace na palubním displeji.
- **GSM-R** je rádiový systém pro poskytování komunikace mezi tratí a vlakem, založený na standardním GSM využívajícím frekvence výhradně pro železniční aplikace s určitými specifickými a pokročilými funkcemi. [2]

ERTMS je založen na specifikacích definovaných ERA. ERA hraje roli autority návrhu systému pro ERTMS. Specifikace mají různé verze a jsou rozděleny na podmnožiny – subsety. Tato závěrečná práce vychází z verzí 2.3.0, 3.4.0 a 3.6.0 SUBSET-026 System Requirements Specification (dále též "subset 026").

3.2 Funkce ETCS

ETCS je jednotné evropské vlakové zabezpečovací zařízení. ETCS umožňuje spolehlivě a kontinuálně kontrolovat jízdu vlaků. Zabezpečovač dohlíží na to, zda se vlak pohybuje v přesně vymezeném úseku trati. ETCS také dohlíží na dodržování nejvyšší dovolené rychlosti v daném úseku a nejvyšší dovolené rychlosti vlaku. Při jejím překročení zabezpečovač zasáhne do řízení vozidla. Před zahájením provozního nebo nouzového brzdění ETCS varuje strojvedoucího, aby měl možnost při řízení traťového vozidla sám snížit rychlost, případně zastavit. [3]

Technické vybavení ETCS se dá rozdělit na dvě podskupiny, a to na traťovou část (systém přímo v kolejišti) a vozidlovou část (jednotka v lokomotivě). [3]

3.2.1 Traťová část

Traťová část je pevně nainstalovanou součástí ETCS. Čím vyšší je aplikační úroveň ETCS tím více klesá počet instalovaných zařízení. Nejdůležitější z nich jsou uvedeny níže. [4]

3.2.1.1 Eurobalíza

Eurobalíza, neboli balíza, je základní prostředek ETCS pro přenos informací do vozidla. Využívá bodového přenosu informací. Při průjezdu vozidla nad balízou vlak získává informace ve formě Eurotelegramu. Eurobalízy se umisťují mezi kolejnice do balízových skupin situovaných u návěstidla. Balízová skupina se skládá z jedné až osmi balíz. Balíza je zobrazena na obrázku 3.1. [4]

Obrázek 3.1 Eurobalíza [16]

3.2.1.2 Euroloop

Euroloop je zařízení pro tratě vybavené aplikační úrovní 1 ETCS. Toto zařízení, podobně jako balíza, přenáší informace z trati do palubní jednotky ETCS. Oproti balíze se liší tím, že pokrývá delší traťový úsek než jeden bod, a tak umožňuje liniový přenos. [15]

3.2.1.3 RBC

RBC (Radiobloková centrála) generuje oprávnění k jízdě na základě informací získaných z traťové části ETCS. Oprávnění je předáno pomocí komunikační sítě GSM-R konkrétnímu vlaku. RBC dokáže přesně identifikovat určitý vlak ve své oblasti. [4]

3.2.2 Vozidlová část

Vozidlová část je technické vybavení ETCS umístěné na vozidle. Všechna vozidla určená pro ETCS musí být vybaveny certifikovanými palubními systémy. Nejpodstatnější z nich jsou uvedeny níže. [4]

3.2.2.1 EVC

EVC (Centrální počítač) je hlavní řídící jednotka ETCS, která bezpečně zpracovává informace ETCS. EVC je propojen s ovládacími prvky vozidla pro regulaci rychlosti, všemi zařízeními strojvedoucího a lokalizačními senzory pro určení polohy vlaku. EVC je znázorněno na obrázku 3.2. [4]

Obrázek 3.2 EVC [4]

3.2.2.2 JRU

JRU (Záznamová jednotka) slouží pro zaznamenávání všech údajů z provozu. [4] JRU musí mít velkou odolnost pro případ mimořádné události. Analogicky lze toto zařízení přirovnat k černé skříňce.

3.2.2.3 DMI

DMI (Displej strojvedoucího) je rozhraní, pomocí kterého strojvedoucí ovládá vozidlovou část a kde se mu zobrazují informace od EVC. [5] DMI je zobrazeno na obrázku 3.3.

Obrázek 3.3 DMI [17]

3.2.2.4 BTM

BTM (Přenosový modul balízy) slouží k příjmu Eurotelegramů z balíz a jejich přenosu do EVC. [5]

3.2.2.5 TIU

TIU (Rozhraní k vlaku) je rozhraní, pomocí kterého EVC získává informace o stavu některých elementů na vozidle a opačným směrem zasílá příkazy a informace vlaku. [5]

3.2.2.6 ODO

ODO (Odometrie) měří rychlost vlaku, ujetou vzdálenost a zasílá tyto informace EVC.

3.3 Aplikační úrovně ETCS

ETCS je tvořen jednotlivými výše uvedenými zařízeními, které svými kombinacemi a zapojením do zabezpečovače umožňují dosáhnout různých úrovní funkce tohoto systému. ETCS má 6 aplikačních úrovní. Aplikační úroveň se značí velkým písmenem L.

Zabezpečovač umožňuje bezproblémovou migraci z jedné aplikační úrovně na jinou. [7]

Jednotlivé aplikační úrovně jsou uvedeny níže.

3.3.1 ETCS L0

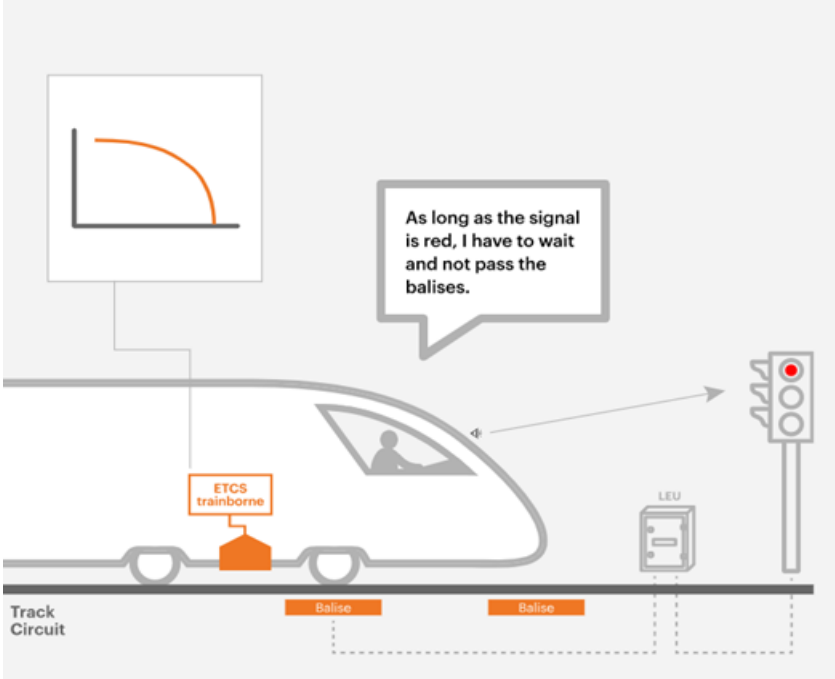
Aplikační úroveň 0 se používá pro chod vlaků vybavených ETCS na tratích, kde infrastruktura ETCS není přítomna, nebo není v provozu. Palubní jednotka ETCS nabízí pouze kontrolu maximální rychlosti. Eurobalízy se používají pouze k příkazu přechodu na jinou úroveň. [6]

3.3.2 ETCS LSTM

Aplikační úroveň STM se používá k umožnění provozu vozidla vybaveného ETCS na infrastruktuře, kde je nainstalován národní systém ATP. Používá se k tomu STM modul, který funguje jako rozhraní mezi palubní částí ETCS a národním ATP systémem. Balízy slouží jenom k příkazu změny úrovně. [6]

3.3.3 ETCS L1

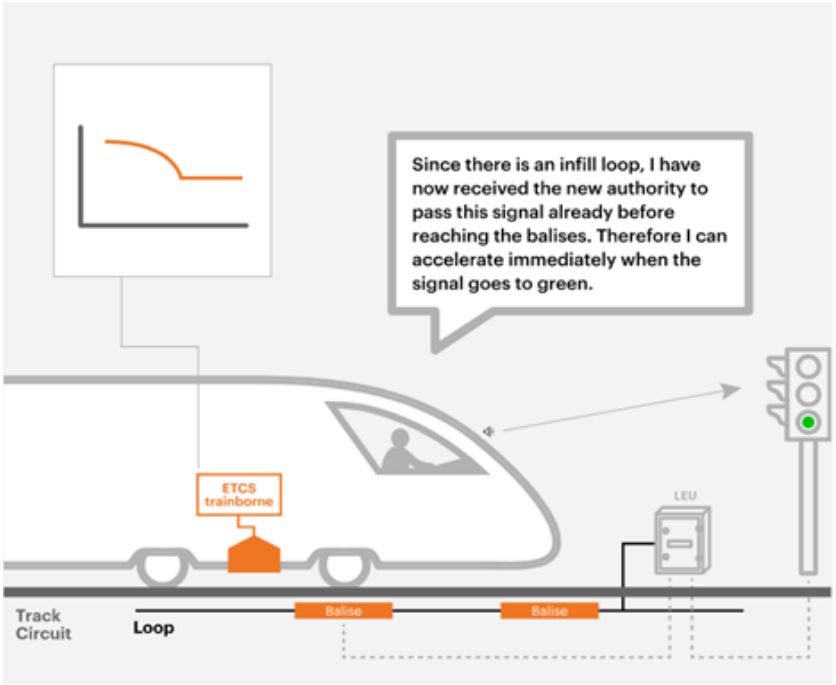
Aplikační úroveň 1 je navržena jako doplněk pro běžné tratě vybavené návěstidly a vlakovými detektory. Eurobalízy jsou instalovány na trati vedle návěstidel a slouží pro přenos informací do řídícího centra a vlaku. Informace obdržené z balíz jsou používány k výpočtu maximální rychlosti vlaku palubním zařízením ETCS, které pomáhá určit, kdy a kde vlak zastavit. Pro nepřetržitý přenos dat na delší vzdálenost lze navíc k Eurobalízám implementovat Euroloop. Řešení s výhradním použitím balíz je znázorněno na obrázku 3.4. Verze s Euroloop a balízami je zobrazena na obrázku 3.5. [7]



Obrázek 3.4 ETCS L1 bez Euroloop [8]

3.3.4 ETCS L2

V porovnaní s úrovní 1 aplikační úroveň 2 nepotřebuje návěstidla. Oprávnění k pohybu palubní jednotka dostává přímo z RBC pomocí GSM-R. Nepřetržitá komunikace mezi RBC a palubním



Obrázek 3.5 ETCS L1 s Euroloop [8]

zařízením umožňuje vlaku dosáhnout optimální rychlosti při zachování bezpečné brzdné dráhy. V úrovni 2 se Eurobalízy používají pouze ke sdělování pevných zpráv. Pevná zpráva je zpráva, která se nemění a zůstává ve formě, v jaké byla nakonfigurována projektanty trati. Taková zpráva slouží primárně k identifikaci polohy vlaku. Aplikační úroveň 2 je znázorněna na obrázku 3.6. [7]

3.3.5 ETCS L3

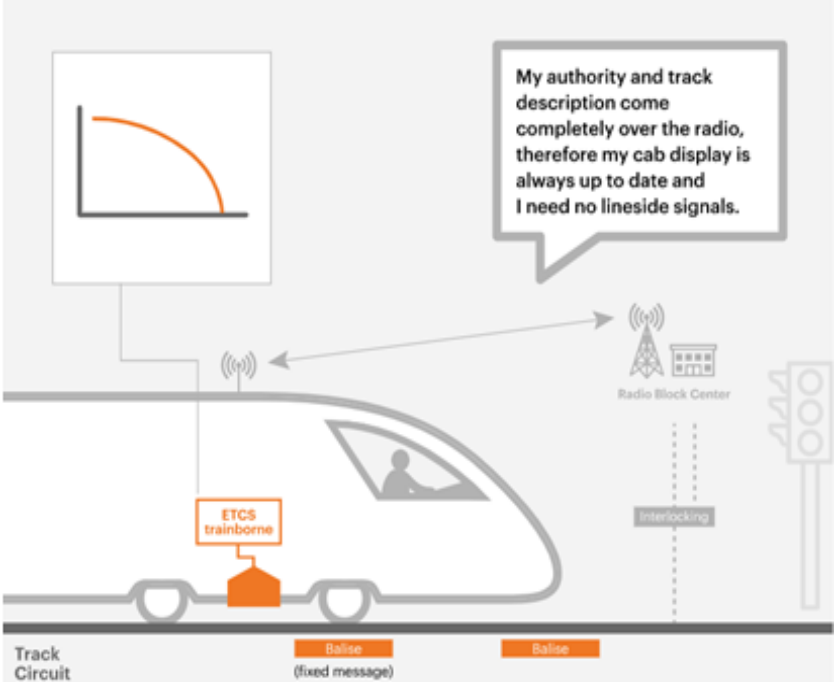
Aplikační úroveň 3 je založena na technologii pohyblivých bloků. V porovnání s úrovněmi 1 a 2 v úrovni 3 ve vlaku je nainstalováno speciální vybavení, pomocí kterého vlak dohlíží na svou celistvost a vysílá informaci o ní do RBC. Vozidlo tak nepřetržitě sleduje svou vlastní polohu. Úroveň 3 je zobrazena na obrázku 3.7. [7]

3.3.6 ETCS LC

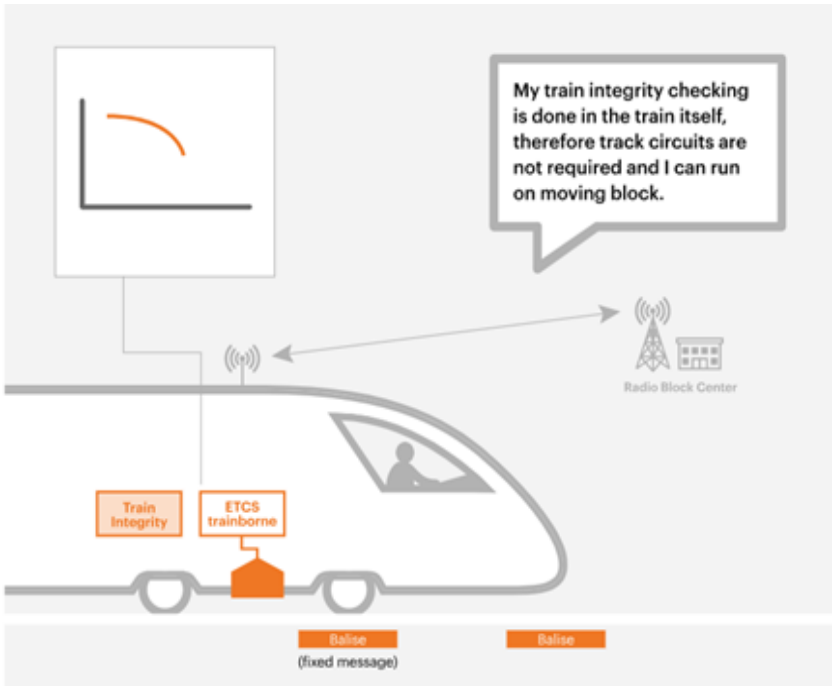
Aplikační úroveň LC (Low Cost) je levnější variantou ETCS určenou pro vedlejší tratě. Úroveň pracuje stejně jako L3, jen počet balíz je minimalizován. [4]

3.4 ETCS v ČR

Počátek aktivit, směřujících k instalaci systému ETCS do podmínek ČR, započal v roce 2001, přičemž bylo rozhodnuto, že další rozvoj rádiových systémů a vlakového zabezpečovacího zařízení bude veden zásadně cestou ERTMS. V roce 2005 byla zahájena realizace pilotního projektu ETCS L2 ve verzi 2.3.0 na úseku Poříčany – Kolín (mimo). Projekt byl uveden do testovacího provozu v roce 2011. Úspěšné zkušenosti s implementací ETCS L2 v rámci pilotního projektu se staly základem pro zadání dalších staveb se systémem ETCS. [9]



Obrázek 3.6 ETCS L2 [8]



Obrázek 3.7 ETCS L3 [8]

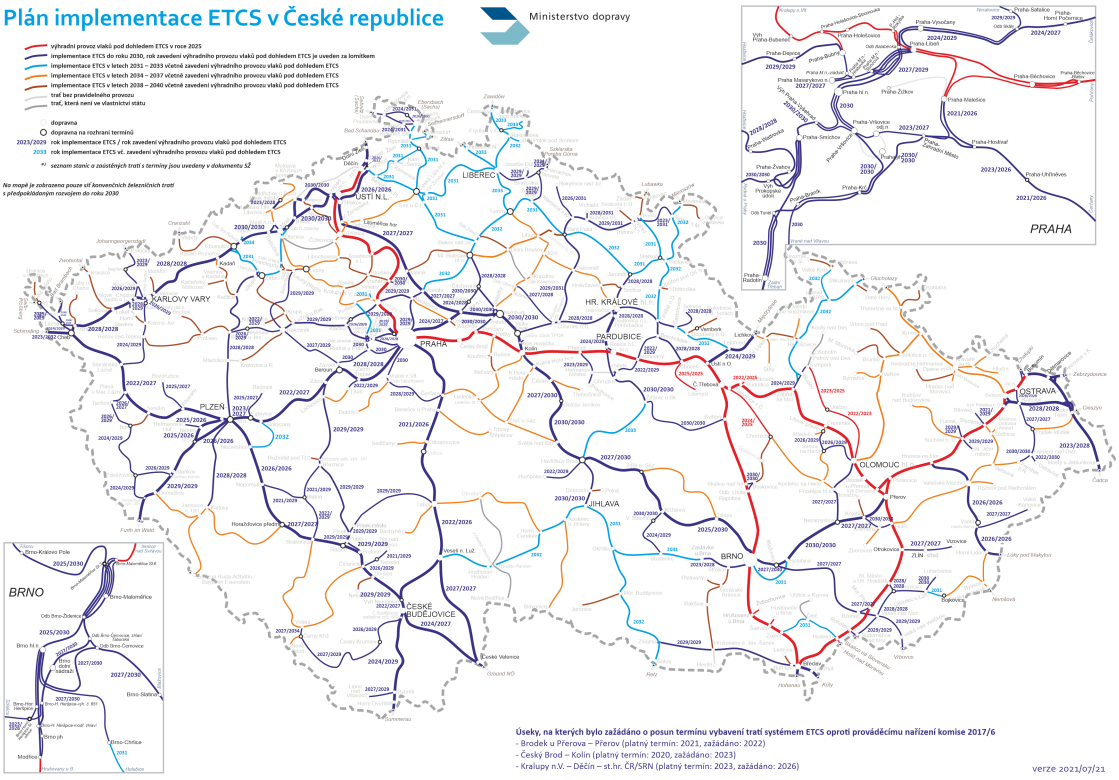
V roce 2017 byl vládou ČR schválen Národní implementační plán ERTMS. Cílem plánu je, aby migrační období do zavedení výhradního provozu vozidel s ETCS bylo co nejkratší a výhody systému ERTMS mohly být co nejdříve využívány. Migrační období pro systém ETCS

v daném traťovém úseku je určeno jako časový interval od okamžiku uvedení systému ETCS do provozu, do okamžiku zavedení provozu všech vlaků pod výhradním dohledem ETCS. Od konce migračního období nesmí do daného traťového úseku vjet vlak, který nedisponuje funkční kompatibilní palubní část ETCS. [9]

V souladu s Národní implementační plánem bude GSM-R a ETCS L2 nainstalován na tratích nákladních evropských koridorů, procházejících mimo síť národních TŽK a na modernizovaných tratích s rychlostí vyšší než 100 km/h. Na dalších tratích s traťovou rychlostí do 100 km/h bude implementován méně nákladný systém ETCS L1. Tato implementace přinese zvýšení bezpečnosti železniční dopravy na těchto tratích, neboť většinou nebyly vybaveny ani národním systémem. V rámci výstavby nových tratí budou do provozu uváděny pouze úseky, vybavené jenom systémem ETCS, na kterých bude možný pouze provoz vozidel, vybavených ETCS. [9]

V roce 2021 bylo systémem ETCS pokryto celkem 569 kilometrů železničních tratí, přičemž 560 z nich bylo ve zkušebním provozu, a ve zbylých úsecích probíhala příprava. [3]

V září 2021 vláda ČR schválila Plán moderního zabezpečení české železnice – implementace ETCS do roku 2040, který určuje harmonogram dalšího zavádění ETCS různých aplikačních úrovní na všechny tuzemské tratě. Tento plán je zobrazen na obrázku 3.8. [3]



Obrázek 3.8 Plán implementace ETCS v ČR [3]

ETCS má zvýšit úroveň bezpečnosti a umožnit lepší využití kapacity tuzemské železniční dopravy. ETCS by mělo například zabránit i takovým nehodám jako nehoda u Perninku v roce 2020 a nehoda v Milavčích na Domažlicku v roce 2021. ETCS má zároveň zkrátit jízdní doby vlaků při jízdách do zahraničí z důvodu odpadnuté nutnosti výměny lokomotiv na hranicích. [10] [11]

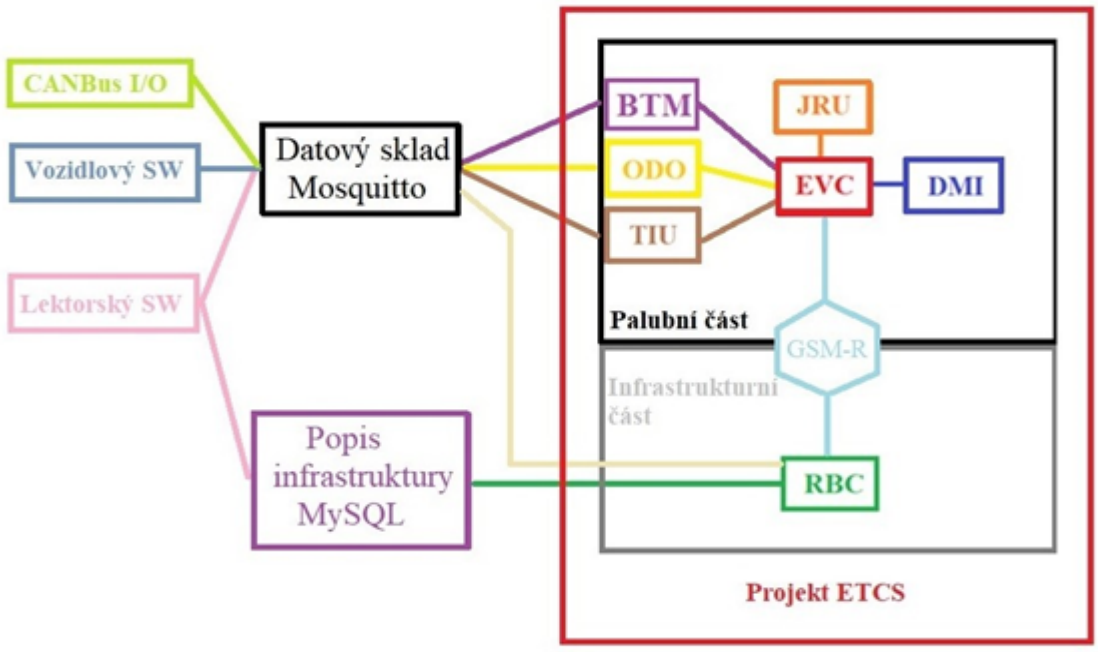
V současné době budování traťové části ETCS zajišťuje Správa železnic. Instalaci mobilní části ETCS si musí zajistit sami dopravci.

3.5 ETCS simulátor

Vlakový simulátor ČVUT s komponentou ETCS je společným projektem Fakulty dopravní a Fakulty informatiky ČVUT. Cílem projektu je realizace prototypu vlakového simulátoru, na kterém by se mohli cvičit strojvedoucí v ČR.

Návrh simulátoru ETCS vychází ze specifikací na ETCS definovaných ERA. Stěžejní specifikace pro návrh simulátoru ETCS jsou SUBSET-026, ERA_ERTMS_015560 a SUBSET-027. Chování simulátoru musí co nejvíc odpovídat projevům skutečného zabezpečovacího zařízení. [12]

Aplikace ETCS je realizována jednotlivými softwarovými moduly, které simulují funkce jak mobilní, tak traťové části systému ETCS. Hlavní moduly jsou RBC, EVC, JRU, TIU, ODO, BTM, DMI a lektorské SW. Všechny tyto moduly, mimo lektorské SW, simulují chování skutečných odpovídajících komponent ETCS. Lektorské SW realizuje povely pro RBC – nastavuje podmínky stavění vlakových cest a definuje parametry pro oprávnění k jízdě. Základní architektura simulátoru ETCS je znázorněna na obrázku 3.9. [12]



Obrázek 3.9 Architektura aplikace ETCS [12]

V současné době existuje plně funkční a otestované demo simulátoru ETCS pro aplikační úroveň ETCS L2, verzi specifikací 2.3.0. V tomto demu jsou implementovány pouze některé funkce této úrovně. Nyní se simulátor ETCS rozšiřuje o další funkce, kdy některé z nich budou implementovány nejenom pro verzi 2.3.0, ale i pro modernější verze 3.4.0 a 3.6.0.

3.6 Současný stav komunikace mezi EVC a RBC v simulátoru ETCS

Dosavadní komunikace mezi EVC a RBC v aplikaci ETCS probíhá na základě kapitol 3, 7 a 8 subsetu 026 System Requirements Specification verze 2.3.0. V textu této podkapitoly se uvádí zmíněné kapitoly pouze jako kapitola 3, kapitola 7 a kapitola 8.

Podle kapitoly 7 se data mezi palubní částí ETCS a radioblokovou centrálou přenáší pomocí radia. [13] V simulátoru ETCS to tak není. Modul EVC přijímá a odesílá data RBC ve formátu JSON. Ve stejném formátu funguje RBC vůči EVC. Pro přenos informací mezi zmíněné moduly se používá MQTT broker. Přijetí, zpracování a odesílaní dat na obou stranách je napsáno v programovacím jazyce C++.

Kapitola 7 stanoví, že pro přenos informací se používá jazyk ETCS založený na proměnných a packetech popsaných v této kapitole a zprávách vysvětlených v kapitole 8. [13] Podle kapitoly 8 se zprávy skládají z konkrétních proměnných a případně i packetů. [14] Zprávy mají své unikátní číslo (ID) a rozlišují se podle odesilatele. [14] Packety také mají ID a rozlišují se podle odesilatele. [13] Moduly EVC a RBC používají v simulátoru pro přenos dat jazyk ETCS, a proto komunikují pomocí zpráv z kapitoly 8.

Podle kapitoly 8 pořadí dat v přenosu musí odpovídat pořadí datových prvků uvedených ve zprávě. [14] V simulátoru ETCS se z důvodu zjednodušení pořadí dat ve zprávě při přenosu nezachovává.

Demo simulátoru používá pouze omezený počet funkcí skutečného ETCS. Z tohoto důvodu byly implementovány jenom zprávy, které jsou potřebné pro správné fungování těchto funkcí. ID zpráv od RBC, které jsou obsažené v demu je 39, 8, 32, 3. ID zpráv od EVC, které jsou v simulátoru je 155, 156, 154, 157, 132, 136, 150, 159, 129. ID packetů od RBC, které jsou obsažené v demu je 15, 58, 21, 27. ID packetů od EVC, které jsou v simulátoru je 0, 1, 4, 9, 3, 11.

Vysvětlení proměnných, packetů a zpráv podle kapitol 7 a 8, jejich současná implementace v aplikaci ETCS, pravidla komunikace mezi EVC a RBC v souladu s kapitolou 3 a dosavadní stav komunikace s RBC na straně EVC v demu simulátoru ETCS jsou popsány níže. Výše zmíněné proměnné, packety, zprávy a komunikace s RBC na straně EVC v aplikaci ETCS byly vytvořeny mnou v rámci předmětů BI-SP1 a BI-SP2.

3.6.1 Proměnné

Kapitola 7 stanoví, že každá proměnná má unikátní název, délku udanou v počtu bitů, popis a rozsahy hodnot, které může nabývat. [13] V demu simulátoru jsou proměnné uvedené jako proměnné mající bezznaménkový celočíselný datový typ o šířce přesně 8, 16, 32 a 64 bitů. Každá proměnná v aplikaci ETCS má datový typ s nejmenším počtem bitů, do kterého se vejde její délka. Například Q_DIR má délku 2 bity. Nejvhodnější typ pro ni je o šířce 8 bitů. Příklad deklarace Q_DIR v aplikaci ETCS je uveden ve výpisu kódu 3.1. Při ukládání hodnoty do proměnné vždy dochází ke kontrole, zda je tato hodnota platná pro tuto proměnnou.

```
/**
 * According to subset 7.
 * Means: Validity direction of transmitted data.
 * Values: from 0 to 3.
 */
std::uint8_t Q_DIR;
```

Výpis kódu 3.1 Deklarace proměnné Q_DIR v simulátoru ETCS

3.6.2 Packety

Packety jsou seskupení více proměnných do jedné jednotky s definovanou vnitřní strukturou. [13] Vnitřní struktura v tomto případě znamená, že proměnné mají v packetech přesně definované pořadí. Struktura packetu odesílaného RBC EVC se skládá z proměnných NID_PACKET (ID packetu), Q_DIR (směr platnosti přenášených dat), L_PACKET (délka packetu v bitech), volitelně z Q_SCALE (měřítko vzdálenosti, které se používá pro všechny informace o vzdálenosti v packetu) a z definované množiny proměnných. [13] Struktura packetu zasílaného RBC EVC je uvedena v tabulce 3.1. Struktura packetu posílaného EVC RBC se skládá z NID_PACKET, L_PACKET, volitelně z Q_SCALE a z definované množiny proměnných. [13] Struktura packetu odesílaného EVC RBC je uvedena v tabulce 3.2.

Tabulka 3.1 Struktura packetu odesílaného RBC EVC

NID_PACKET
Q_DIR
L_PACKET
Q_SCALE (volitelně)
Další proměnné (volitelně)

Tabulka 3.2 Struktura packetu zasílaného EVC RBC

NID_PACKET
L_PACKET
Q_SCALE (volitelně)
Další proměnné (volitelně)

V aplikaci ETCS mají packety strukturu v souladu s požadavky kapitoly 7. Pro každý packet v demu byla vytvořena třída reprezentující tento packet. Členské proměnné takové třídy odpovídají proměnným příslušného packetu. Všechny členské proměnné nejsou veřejné, uživatel tak nemá k členským proměnným přístup, a nemůže proto přiřadit do proměnné neplatnou hodnotu.

Některé proměnné v packetech mohou následovat jen v případech, kdy proměnná před nimi (dále též „rozhodovací proměnná“) má stanovenou hodnotu. Například v packetu 15 Level 2/3 Movement Authority zasílaným RBC EVC, pokud se proměnná Q_ENDTIMER rovná 1, následují po ní proměnné T_ENDTIMER a D_ENDTIMERSTARTLOC. Pokud se Q_ENDTIMER nerovná 1, zmíněné proměnné se dále nevyskytují. V simulátoru ETCS jsou takové případy řešeny tak, že se vytvoří abstraktní bázová třída obsahující rozhodovací proměnnou, ze které dědí třída, kde se příslušná proměnná rovná určené hodnotě a jsou uvedeny následné proměnné, a třída, kde se příslušná proměnná nesmí rovnat určené hodnotě a žádné další proměnné nejsou. Ve třídě reprezentující packet je unique_ptr na abstraktní bázovou třídu, který ukazuje na objekt odvozené třídy.

Některé packety obsahují v sobě proměnnou N_ITER, která udává počet iterací proměnné nebo skupiny proměnných, které po ní následují. Mohou existovat dvě úrovně vnoření iterací. Pokud se N_ITER rovná 0, nenásleduje po ní žádná iterovaná proměnná. V aplikaci ETCS ve třídách reprezentujících packety je iterovaná proměnná představena jako vektor datového typu příslušného dané proměnné.

Instance třídy zastupující packet odpovídají packetům s konkrétními hodnotami proměnných. Po svém vytvoření objekt takové třídy už nemá možnost změnit hodnoty jednotlivých členských proměnných. Důvod absence takové možnosti je, že použití packetů v ETCS simulátoru je omezeno na tvorbu zpráv obsahujících packety, které jsou ihned po vytvoření odeslány, a čtení přijatých zpráv obsahujících packety.

Packety mají stejné počáteční proměnné. Proto by všechny třídy reprezentující packety měly několik stejných členských proměnných a stejný způsob ukládání hodnot do těchto proměnných konstruktorem, ověření platnosti hodnot pro tyto proměnné, zápisu zmíněných proměnných do proudu, převodu proměnných do JSON. Také by měly stejné metody typu getter pro návrat hodnot těchto proměnných. Z těchto důvodů byla navržena určitá hierarchie tříd, která eliminuje opakující se části kódu.

Nejvyšší třída v hierarchii tříd reprezentujících packety je abstraktní třída Packet. Třída Packet obsahuje členské proměnné NID_PACKET a L_PACKET označené specifikátorem protected, konstruktor s proměnnými a konstruktor s JSON objektem obsahujícím položky se stejnými názvy, jako u členských proměnných. Tyto konstruktory slouží pro přiřazení hodnot členským proměnným. Třída Packet zahrnuje veřejné metody typu getter pro návrat hodnot členských proměnných, veřejný přetížený operátor << a deklaraci veřejné abstraktní členské funkce sloužící pro převod členských proměnných do JSON. Třída deklaruje veřejnou abstraktní metodu určenou pro zápis packetu do proudu. Tato metoda se používá v těle přetíženého operátoru <<. Třída Packet obsahuje veřejnou abstraktní metodu sloužící pro vytvoření unique_ptr na tuto třídu, který bude ukazovat na hlubokou kopii objektu nějaké podtřídy. Třída má protected metodu sloužící pro převod NID_PACKET a L_PACKET do JSON a veřejnou metodu zapisující zmíněné proměnné do proudu podle jejich pořadí v packetech. Tyto dvě členské funkce se používají v odvozených třídách. Třída Packet obsahuje virtuální destruktor a privátní členskou funkci, která ověřuje platnost hodnot pro členské proměnné. Z této třídy dědí všechny třídy obsahující v sobě proměnné NID_PACKET a L_PACKET. Deklarace třídy Packet je ve výpisu kódu 3.2.

Abstraktní třída PacketQScale dědí z třídy Packet. Tato třída obsahuje protected členskou proměnnou Q_SCALE. Třída PacketQScale má dva konstruktory. Jeden je s proměnnými, druhý je s JSON objektem. Tato třída má veřejný getter pro získání hodnoty Q_SCALE. Třída PacketQScale má metody, které se používají odvozenými třídami. Jedná z těchto metod slouží pro převod NID_PACKET, L_PACKET a Q_SCALE do JSON a ve svém těle volá zděděnou metodu pro převod NID_PACKET a L_PACKET do JSON. Tato metoda je označena specifikátorem protected. Druhá metoda je určená pro zápis do proudu zmíněných proměnných podle jejich pořadí v packetech. Tato členská funkce ve svém těle jako první volá zděděnou metodu pro zápis NID_PACKET a L_PACKET do streamu. Metoda pro zápis do proudu je veřejná. Třída PacketQScale obsahuje privátní metodu pro ověření platnosti hodnoty pro Q_SCALE. Z třídy PacketQScale dědí všechny třídy reprezentující pakety, které obsahují na začátku NID_PACKET, L_PACKET a Q_SCALE. Deklarace třídy PacketQScale je ve výpisu kódu 3.3.

Abstraktní třída PacketQDir dědí z třídy PacketQScale. PacketQDir obsahuje protected členskou proměnnou Q_DIR a veřejný getter pro získání hodnoty Q_DIR. Tato třída má dva konstruktory. Jeden je s proměnnými, druhý je s JSON objektem. Třída PacketQDir má metody, které se používají odvozenými třídami. Jedná z těchto metod slouží pro převod všech členských proměnných této třídy do JSON a ve svém těle volá zděděnou metodu pro převod NID_PACKET, L_PACKET a Q_SCALE do JSON. Tato metoda je protected. Druhá metoda je určená pro zápis do proudu všech členských proměnných podle jejich pořadí v packetech. Tato metoda nepoužívá zděděnou metodu pro zápis NID_PACKET, L_PACKET a Q_SCALE do streamu. Je to z důvodu, že v souladu s kapitolou 7 Q_DIR se ve všech packetech nachází mezi NID_PACKET a L_PACKET a použitím zděděné metody by došlo k zápisu dat v nesprávném pořadí. Tato členská funkce pro zápis do proudu je veřejná. Třída PacketQDir obsahuje privátní metodu pro ověření platnosti hodnoty pro Q_DIR. Z této třídy dědí třídy reprezentující pakety, které obsahují na začátku proměnné NID_PACKET, Q_DIR, L_PACKET a Q_SCALE. Deklarace třídy PacketQDir je ve výpisu kódu 3.4.

Další abstraktní třída, která dědí z třídy PacketQScale, je třída Report. Třída Report je bázovou třídou pro neabstraktní třídy PositionReport a PositionReportBalise reprezentující packety z EVC do RBC s ID 0 a 1. Třída Report zahrnuje společné proměnné pro tyto dva packety. Vnitřní struktura této třídy je podobná strukturám tříd PacketQScale a PacketQDir. Třída obsahuje dva konstruktory, gettery, metodu pro převod členských proměnných do JSON, metodu pro zápis

```
class Packet {
public:
    Packet(std::uint8_t id, std::uint16_t length);
    explicit Packet(const nlohmann::json &j);
    virtual ~Packet();
    /**
     * Creates from attributes a JSON object.
     */
    virtual nlohmann::json ToJson() const = 0;
    virtual std::unique_ptr<Packet> Clone() const = 0;
    /**
     * Prints packet to stream.
     */
    virtual void Print ( std::ostream & os ) const = 0;
    /**
     * Prints common attributes.
     */
    void PrintIt(std::ostream & os) const;
    friend std::ostream & operator << ( std::ostream & os, const Packet & packet);
    std::uint8_t  GetId() const;
    std::uint16_t GetLength() const;
private:
    /**
     * Checks, if set beforehand attributes are valid according to subset 7.
     */
    bool Valid() const;
protected:
    /**
     * Creates from common attributes a JSON object.
     */
    nlohmann::json ToJsonIt() const;
    /**
     * According to subset 7.
     * Means: Packet identifier.
     */
    std::uint8_t NID_PACKET;
    /**
     * According to subset 7.
     * Means: Packet length in bits.
     * Values: from 0 to 8191.
     */
    std::uint16_t L_PACKET;
};
```

Výpis kódu 3.2 Deklarace třídy Packet v simulátoru ETCS

proměnných do streamu podle jejich pořadí v packetech a členskou funkci pro ověření platnosti hodnot pro členské proměnné. Navíc k tomu tato třída má definované kopírující konstruktor a operátor přiřazení, a to z důvodu, že obsahuje mezi členskými proměnnými několik unique_ptr na bázové třídy s rozhodovacími proměnnými.

```
class PacketQScale : public Packet {
public:
    PacketQScale(std::uint8_t id, std::uint16_t length, std::uint8_t scale);
    explicit PacketQScale(const nlohmann::json &j);
    virtual ~PacketQScale();
    /**
     * Prints common attributes.
     */
    void PrintWithQScale(std::ostream & os) const;
    std::uint8_t GetScale() const;
private:
    /**
     * Checks, if set beforehand attributes are valid according to subset 7.
     */
    bool Valid() const;
protected:
    /**
     * Creates from common attributes a JSON object.
     */
    nlohmann::json ToJsonQScale() const;
    /**
     * According to subset 7.
     * Means: Qualifier for the distance scale.
     * Values: from 0 to 3.
     */
    std::uint8_t Q_SCALE;
};
```

Výpis kódu 3.3 Deklarace třídy PacketQScale v simulátoru ETCS

Všechny třídy reprezentující packety nejsou abstraktní a dědí z abstraktních tříd. Obsah těchto tříd má přibližně stejnou podobu jako obsah jejich nadtříd. Tyto třídy mají členské proměnné neveřejné a přepisují všechny virtuální metody deklarované ve třídě Packet. V konstruktorech tříd zastupujících packety se kontroluje, zda součet délek všech proměnných odpovídá hodnotě L_PACKET a ověřuje se, zda se hodnota uložená v NID_PACKET rovná ID reprezentovaného packetu.

Hierarchie tříd reprezentujících packety je uvedena v diagramu tříd na obrázku 3.10. Do tříd v diagramu nejsou zahrnuté konstruktory a metody, a to z důvodu zjednodušení diagramu. Jejich uvedení by s ohledem na jejich počet vedlo ke značné nepřehlednosti s ohledem na velikost takového diagramu.

3.6.3 Zprávy

Podle kapitoly 8 se vnitřní struktura zpráv skládá z přesně stanovených proměnných, za kterými následují packety. [14] Packety pro zprávy se rozdělují na volitelné a nutné. [14] Kapitola 8 přesně definuje pro každou zprávu nutné packety a množinu volitelných packetů. [14] V demu simulátoru ETCS byl počet volitelných packetů omezen. Zpráva z RBC do EVC s ID 3 může v demu mít pouze volitelné packety s ID 21, 27 a 58. Zprávy z EVC do RBC s ID 136 a 157 mohou v demu obsahovat jenom volitelný packet s ID 4. Zpráva z EVC do RBC s ID 159 může v demu mít pouze volitelný packet s ID 3. Zpráva z EVC do RBC s ID 132 může v demu simulátoru obsahovat jen

```
class PacketQDir : public PacketQScale {
public:
    PacketQDir(std::uint8_t id, std::uint8_t dir, std::uint16_t length, std::uint8_t scale);
    explicit PacketQDir(const nlohmann::json &j);
    virtual ~PacketQDir();
    /**
     * Prints common attributes.
     */
    void PrintWithQDir(std::ostream & os) const;
    std::uint8_t GetDir() const;
private:
    /**
     * Checks, if set beforehand attributes are valid according to subset 7.
     */
    bool Valid() const;
protected:
    /**
     * Creates from common attributes a JSON object.
     */
    nlohmann::json ToJsonQDir() const;
    /**
     * According to subset 7.
     * Means: Validity direction of transmitted data.
     * Values: from 0 to 3.
     */
    std::uint8_t Q_DIR;
};
```

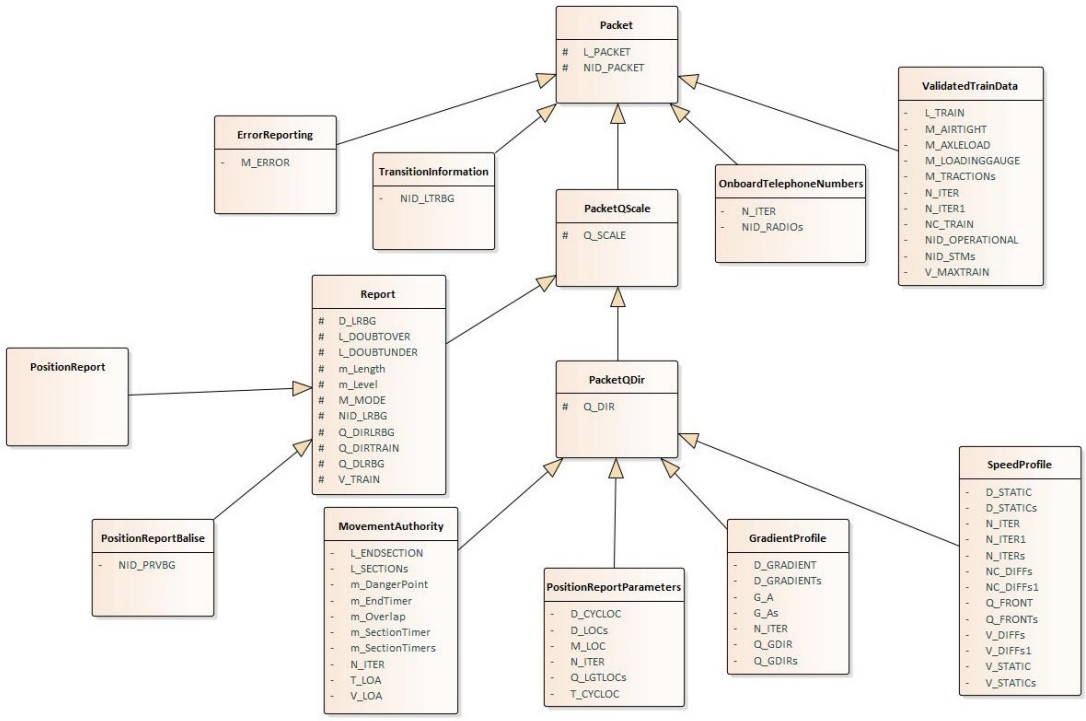
Výpis kódu 3.4 Deklarace třídy PacketQDir v simulátoru ETCS

volitelný packet s ID 9.

V souladu s kapitolou 8 je zakázáno posílat více instancí stejného typu packetů stejným směrem ve stejné zprávě. [14] Několik packetů má z tohoto pravidla výjimku. [14] Tyto packety nejsou součástí dema simulátoru ETCS.

Všechny zprávy mají proměnnou L_MESSAGE, která udává délku zprávy v bytech. [14] Pokud vypočtená délka zprávy v bytech není rovna délce uvedené v L_MESSAGE, bude celá zpráva příjemcem odmítnuta. [14] Podle kapitoly 8 v případě, kdy délka zprávy v bytech ukládaná do L_MESSAGE nevychází celočíselná, přidá se na konec zprávy padding (též výplň). [14] V demu simulátoru se žádný padding na konec zprávy nedodává. Hodnota uložená v L_MESSAGE výplň zahrnuje. Během kontroly se vypočtená délka zprávy v bytech vždy před svým porovnáním s L_MESSAGE zaokrouhluje nahoru.

Podle kapitoly 8 zprávy mají přesně definovanou vnitřní strukturu. Znamená to, že proměnné a packety mají ve zprávách přesně definované pořadí. Podle kapitoly 8 se struktura zprávy odesílané RBC EVC skládá z proměnných NID_MESSAGE (ID zprávy), L_MESSAGE (délka zprávy v bytech zahrnuje v sobě i padding), T_TRAIN (časové razítko), M_ACK (indikátor odeslání potvrzení o přijetí zprávy), NID_LRBG (ID poslední relevantní balízové skupiny), volitelně z dalších proměnných a packetů (nejdříve jsou nutné packety v konkrétním pořadí, poté jsou volitelné packety v jakémkoli pořadí), případně výplně. Struktura zprávy zasílané RBC EVC je uvedena na obrázku 3.11. Struktura zprávy posílané EVC RBC se skládá z NID_MESSAGE, L_MESSAGE, T_TRAIN, NID_ENGINE (ID vlaku), volitelně z dalších proměnných, packetu



Obrázek 3.10 Hierarchie tříd reprezentujících packety v simulátoru ETCS

0 nebo 1 (tento packet mají všechny zprávy z EVC do RBC, s výjimkou zpráv s ID 146, 154, 155, 156 a 159), volitelně také z dalších packetů (nejdřív jsou nutné packety v konkrétním pořadí, poté jsou volitelné packety v jakémkoli pořadí) a výplně. Struktura zprávy odesílané EVC RBC je uvedena na obrázku 3.12. [14]

V aplikaci ETCS mají zprávy strukturu podobnou vnitřní strukturu uvedené v kapitole 8. Liší se pouze tím, že neobsahují na konci výplň.

V demu simulátoru byly vytvořeny třídy reprezentující jednu nebo několik zpráv. Většina zpráv má úplně stejnou strukturu, proto bylo rozhodnuto sjednotit několik zpráv pod jednou třídou. Členské proměnné takových tříd odpovídají proměnným a packetům příslušných zpráv. Všechny členské proměnné jsou neveřejné.

Packety ve zprávách jsou představeny jako std::vector<std::unique_ptr<Packet>>. Na začátku vektoru jsou uvedené všechny nutné packety, které jdou v pořadí odpovídajícímu pořadí nutných packetů ve vnitřní struktuře zprávy z kapitoly 8, za kterými následují volitelné packety v libovolném pořadí.

Instance třídy zastupující zprávu, nebo několik zpráv, odpovídají zprávě s konkrétními hodnotami proměnných a packetů. Po svém vytvoření objekt takové třídy už nemá možnost změnit hodnoty jednotlivých členských proměnných. Důvod absence takové možnosti je, že použití zpráv v ETCS simulátoru je omezeno jejich tvorbou a následujícím odesláním a jejich přijetím a čtením.

Zprávy mají stejné počáteční proměnné. Proto by se většina operací prováděných s těmito proměnnými opakovala v každé třídě zastupující zprávu nebo několik zpráv. Z tohoto důvodu byla navržena určitá hierarchie tříd, která eliminuje opakující se části kódu.

Nejvyšší třída v hierarchii tříd reprezentujících zprávy je abstraktní třída Message. Třída Message obsahuje členské proměnné NID_MESSAGE, L_MESSAGE a T_TRAIN označené specifikátorem protected, konstruktor s proměnnými a konstruktor s JSON objektem obsahujícím položky se stejnými názvy, jako u členských proměnných. Třída má veřejné metody typu getter

Field No.	VARIABLE	Remarks
1	NID_MESSAGE	Message Identification Number
2	L_MESSAGE	Message length including everything (fields 1 to padding).
3	T_TRAIN	Time Stamp from RBC (see chapter 3 – Data Consistency).
4	M_ACK	Indicates whether the message must be acknowledged (or not) by the on-board equipment (message n° 146).
5	NID_LRBG	Identification Number of LRBG.
...	variables as required by NID_MESSAGE	If needed for this message. Used when sending variables which are not included in a packet.
...	packets as required by NID_MESSAGE	If needed for this message.
	Optional packets	Refer to section 8.4.4.4 of this document.
	Padding	If required.

Obrázek 3.11 Struktura zprávy zasílané RBC EVC [14]

Field No.	VARIABLE	Remarks
1	NID_MESSAGE	Message Identification Number
2	L_MESSAGE	Message length including everything (field 1 to padding).
3	T_TRAIN	Time Stamp from Train (see chapter 3 – Data Consistency).
4	NID_ENGINE	Identity of the train.
5	variables as required by NID_MESSAGE	If needed for this message. Used when sending variables which are not included in a packet.
6	Packet 0 or 1	Train-to-track packet type 0 – Position report, or packet type 1 - Position report based on two balise groups. Not included in messages 146, 154, 155, 156 and 159.
7	Other Packets as required by NID_MESSAGE	(only for message 129)
8	Optional packets	
	Padding	If required.

Obrázek 3.12 Struktura zprávy odesílané EVC RBC [14]

pro návrat hodnot členských proměnných, veřejný přetížený operátor << a deklaraci veřejné abstraktní členské funkce sloužící pro převod členských proměnných do JSON. Třída deklaruje veřejnou abstraktní metodu určenou pro zápis zprávy do proudu. Tato metoda se používá v těle

přetíženého operátoru <<. Třída Message má protected metodu sloužící pro převod členských proměnných do JSON a veřejnou metodu zapisující členské proměnné do proudu podle jejich pořadí ve zprávách. Tyto dvě členské funkce se používají v odvozených třídách. Třída Message obsahuje virtuální destruktor a privátní členskou funkci, která ověřuje platnost hodnot pro členské proměnné. Z této třídy dědí všechny třídy obsahující proměnné NID_MESSAGE, L_MESSAGE a T_TRAIN. Deklarace třídy Message je uvedena ve výpisu kódu 3.5.

Abstraktní třída MessageRBC dědí z třídy Message. Tato třída odpovídá svou vnitřní strukturou základní struktuře zpráv z RBC do EVC. MessageRBC obsahuje protected členské proměnné M_ACK a NID_LRBG a veřejné metody typu getter pro získání hodnot těchto členských proměnných. Tato třída má dva konstruktory. Jeden je s proměnnými, druhý je s JSON objektem. Třída MessageRBC má metody, které se používají odvozenými třídami. Jedná z těchto metod slouží pro převod všech členských proměnných této třídy do JSON a ve svém těle volá zděděnou metodu pro převod NID_MESSAGE, L_MESSAGE a T_TRAIN do JSON. Tato metoda je protected. Druhá metoda je určená pro zápis do proudu všech členských proměnných podle jejich pořadí ve zprávách. Tato členská funkce volá jako první ve svém těle zděděnou metodu pro zápis NID_MESSAGE, L_MESSAGE a T_TRAIN do proudu. Tato metoda je veřejná. Třída MessageRBC obsahuje privátní metodu pro ověření platnosti hodnot pro M_ACK a NID_LRBG. Z této třídy dědí všechny třídy reprezentující zprávy od RBC do EVC. Deklarace třídy MessageRBC je uvedena ve výpisu kódu 3.6.

Další abstraktní třída, která dědí z třídy Message, je třída MessageEngineEVC. Tato třída odpovídá svou vnitřní strukturou základní struktuře zpráv z EVC do RBC. Obsah třídy MessageEngineEVC je velice podobný obsahu třídy MessageRBC. MessageEngineEVC obsahuje protected členskou proměnnou NID_ENGINE a podobné metody a konstruktory jako třída MessageRBC, pouze místo operací s M_ACK a NID_LRBG operuje s proměnnou NID_ENGINE. Z této třídy dědí všechny třídy reprezentující zprávy od EVC do RBC. Deklarace třídy MessageEngineEVC je uvedena ve výpisu kódu 3.7.

Všechny třídy reprezentující zprávy nejsou abstraktní a dědí z abstraktních tříd. Obsah těchto tříd má přibližně stejnou podobu jako obsah jejich nadtříd. Tyto třídy mají členské proměnné neveřejné a přepisují všechny virtuální metody deklarované ve třídě Message. V konstruktorech tříd zastupujících zprávy se kontroluje, zda součet délek všech proměnných, a v případě tříd reprezentujících zprávy s packety i packetů, uvedený v bytech a zaokrouhlený nahoru, odpovídá hodnotě L_MESSAGE a ověřuje se, zda hodnota NID_MESSAGE je mezi ID reprezentovaných zpráv. Ve třídách reprezentujících zprávy obsahující packety se ihned po ukládání informací do std::vector<std::unique_ptr<Packet>> ověřuje, zda vektor obsahuje na začátku všechny nutné packety pro tuto zprávu, a to v pořadí podle kapitoly 8, zda zbývající packety patří mezi volitelné packety pro tuto zprávu a zda se neopakují.

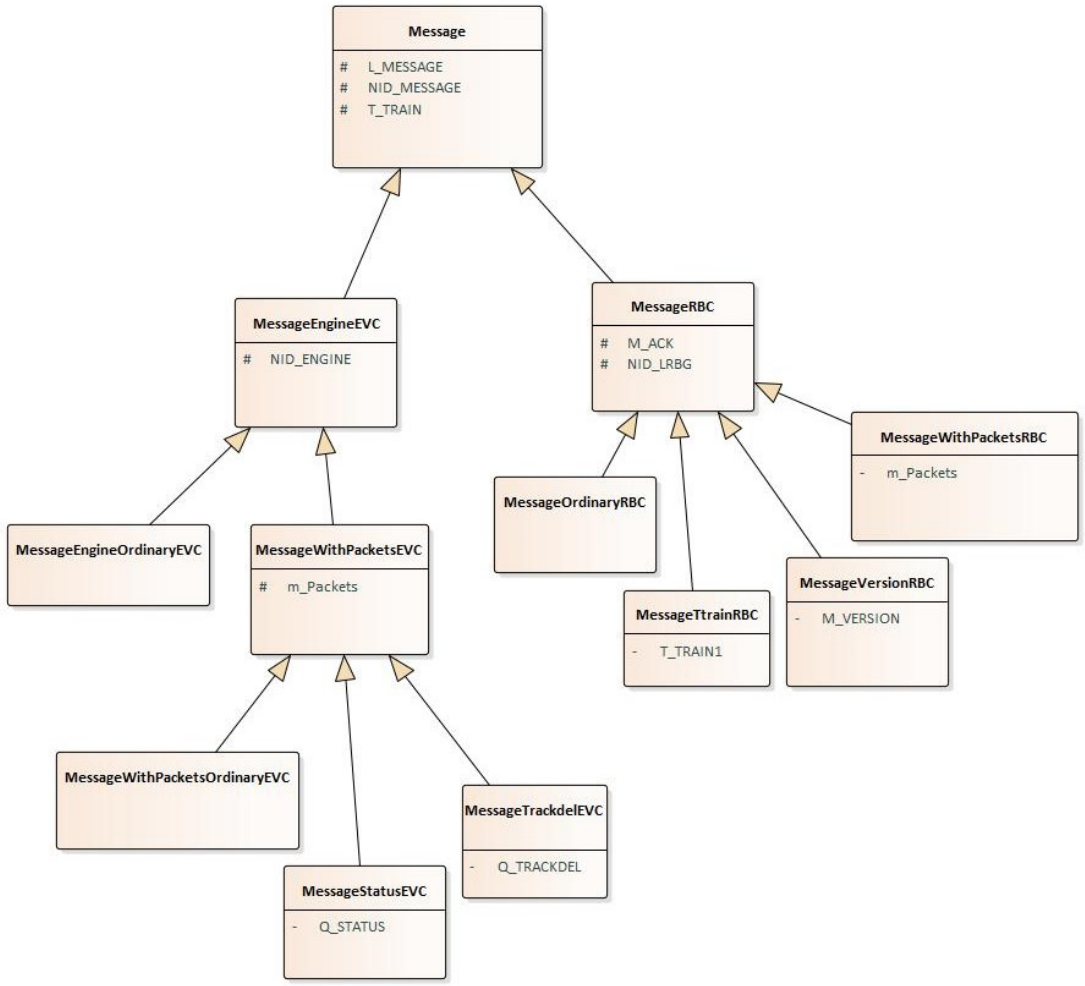
Hierarchie tříd reprezentujících zprávy je uvedena v diagramu tříd na obrázku 3.13. Do tříd v diagramu nejsou zahrnuté konstruktory a metody, a to z důvodu zjednodušení diagramu. Jejich uvedení by s ohledem na jejich počet vedlo ke značné nepřehlednosti s ohledem na velikost takového diagramu.

3.6.4 Výjimky

Pokud je ukládaná hodnota do členské proměnné neplatná, nebo vypočtené délky zpráv a packetů neodpovídají hodnotám L_MESSAGE a L_PACKET, nebo NID_MESSAGE a NID_PACKET neodpovídají ID reprezentované zprávy nebo packetu, vyhodí se vlastní výjimka BadInputException.

3.6.5 Komunikace mezi EVC a RBC na straně EVC

Podle kapitoly 3 může být komunikační relace zahájena jak vlakem, tak i RBC, a to pouze při přesně definovaných podmínkách. [18] V aplikaci ETCS se toto pravidlo dodržuje, avšak je



Obrázek 3.13 Hierarchie tříd reprezentujících zprávy v simulátoru ETCS

omezeno tím, že komunikační relaci může zahajovat jenom EVC, a to pouze během procedury Start of Mission.

Zahájení komunikační relace mezi EVC a RBC v aplikaci ETCS probíhá podle kapitoly 3. Nejdřív EVC odešle RBC zprávu s ID 155. RBC odpoví zprávou 32, která v sobě zahrnuje proměnnou M_VERSION (verze jazyka ETCS). EVC po přijetí zprávy 32 porovná hodnotu této M_VERSION s hodnotou M_VERSION kterou má v sobě uloženou. Pokud tyto dvě proměnné mají různé hodnoty, komunikační relace nemůže být navázána a EVC odešle RBC zprávu 154, svědčící o nekompatibilitě verzí jazyka ETCS, a zprávu 156, oznamující ukončení komunikační relace. RBC odešle jako potvrzení o přijetí zprávy 156 zprávu 39. Jakmile modul EVC dostane toto potvrzení, ukončí svůj běh. Pokud přijatá M_VERSION a M_VERSION uložená v EVC mají stejnou hodnotu, komunikační relace je navázána pro EVC a EVC odešle RBC zprávu 159. Po obdržení této zprávy RBC je komunikační relace navázána i pro RBC.

Když v simulátoru ETCS vlak dojede na konec trati a zastaví se, EVC pošle RBC zprávu 156, dostane zprávu 39 od RBC a ukončí svůj běh.

Přijetí a odesílání zpráv RBC se koná na straně EVC ve třídě Client, která představuje MQTT klienta. V této třídě jsou uloženy všechny potřebné informace pro vytvoření zpráv a packetů.

Ve třídě Client je veřejná metoda pro přijetí zprávy v JSON. Tato metoda zjistí ID zpraco-

vávané zprávy a použije ho v příkazu switch. V těle tohoto příkazu se podle ID zprávy provede konkrétní blok operací. Na začátku každého z těchto bloku se vytvoří pomocí konstruktoru s JSON objektem instance třídy reprezentující zprávu s tímto ID. Potom v každém bloku proběhne kontrola času odeslání zprávy. Zkontroluje se, zda hodnota T_TRAIN obsažená ve zpracovávané zprávě je větší než T_TRAIN ve zprávě naposledy přijaté od RBC a zároveň je větší než T_TRAIN ve zprávě naposledy odeslané EVC RBC. Pokud to tak není, zpracování této zprávy od RBC se ukončí, žádné z jejích proměnných a případně packetů nebudou uložené do vnitřních struktur modulu EVC a tato zpráva se nebude považovat za přijatou. Tato kontrola proměnné T_TRAIN probíhající ve třídě Client je zjednodušením kontroly T_TRAIN popsané v kapitole 3. Po kontrole hodnoty T_TRAIN v každém bloku následuje kontrola NID_LRBG uložené ve zpracovávané zprávě. Pokud hodnota zmíněné NID_LRBG není mezi hodnotami proměnných NID_LRBG již odeslaných EVC RBC, zpráva se nebude dál zpracovávat, žádné z jejích proměnných a případně packetů nebudou uložené v EVC a zmíněná zpráva nebude označená za přijatou. Tato kontrola NID_LRBG je zjednodušenou kontrolou NID_LRBG popsanou v kapitole 3. Po kontrole NID_LRBG zpracovaní zprávy pokračuje a provádí se příslušné operace v závislosti na ID zprávy, případně se na konci bloku odesílá zpráva RBC. Odesílaní zpráv RBC může probíhat i v jiných metodách třídy Client.

Když nastanou v modulu EVC přesně definované podmínky pro odeslání konkrétní zprávy, vytvoří se pomocí konstruktoru s proměnnými objekt třídy zastupující tuto zprávu. Nad tímto objektem se zavolá metoda, která vytvoří ze všech jeho členských proměnných JSON objekt. Tento JSON objekt se potom odešle RBC.

3.7 Rozdíly a podobnosti proměnných, packetů, zpráv a komunikace mezi RBC a EVC na straně EVC mezi verzemi 2.3.0, 3.4.0 a 3.6.0 subsetu 026 System Requirements Specification

V ETCS není povoleno používat různé verze subsetů v různých částech systému, všude musí být obsažena stejná verze.

Ve všech třech verzích 2.3.0, 3.4.0 a 3.6.0 subsetu 026 System Requirements Specification probíhá komunikace mezi EVC a RBC pomocí zpráv uvedených v kapitole 8. Popis zpráv, packetů a proměnných v kapitolách 7 a 8 v různých verzích příslušného subsetu je téměř stejný. Liší se ID zpráv a ID volitelných a nutných packetů pro zprávy obsahující packety. Některé zprávy a packety v novější verzi subsetu 026 mají, na rozdíl od starší verze, některé proměnné navíc, a naopak některé oproti starší verzi postrádají. Proměnné, které byly ve starší verzi subsetu 026, mohou v novější verzi tohoto subsetu mít odlišnou délku v bitech a jiné rozsahy platných hodnot. Postup přijetí EVC zpráv od RBC je téměř stejný ve všech třech verzích.

Podrobnější vysvětlení rozdílů a podobností zpráv, packetů a proměnných potřebných pro komunikaci mezi EVC a RBC v simulátoru ETCS, mezi verzemi 2.3.0, 3.4.0, 3.6.0 subsetu 026, jsou popsány níže. Zde jsou také popsány rozdíly a podobnosti v komunikaci RBC a EVC na straně EVC mezi těmito verzemi příslušného subsetu.

3.7.1 Rozdíly a podobnosti proměnných mezi verzemi 2.3.0, 3.4.0 a 3.6.0 subsetu 026

Verze 2.3.0, 3.4.0 a 3.6.0 kapitoly 7 subsetu 026 mají některé proměnné stejné a některé odlišné. Stejná proměnná v různých verzích může mít odlišné rozsahy platných hodnot a různou délku v bitech. Rozdíly a podobnosti proměnných potřebných pro komunikaci mezi EVC a RBC v

simulátoru ETCS mezi verzemi 2.3.0, 3.4.0 a 3.6.0 kapitoly 7 subsetu 026 jsou podrobně popsané v příloze „Proměnné“.

3.7.2 Rozdíly a podobnosti packetů mezi verzemi 2.3.0, 3.4.0 a 3.6.0 subsetu 026

Verze 2.3.0, 3.4.0 a 3.6.0 kapitoly 8 subsetu 026 zahrnují stejná a odlišná ID packetů, které mohou být obsaženy ve zprávách. V komunikaci mezi EVC a RBC v simulátoru ETCS se budou používat všechny packety zmíněné v kapitole 8, vyjma packetů s ID 44 (a to jak packetu s ID 44 z EVC do RBC, tak i packetu s ID 44 z RBC do EVC) a packetu z RBC do EVC s ID 180. Tyto packety, po dohodě s panem doc. Ing. Martinem Lesem, Ph.D., který je iniciátorem a řešitelem za ČVUT Fakultu Dopravní projektu Vlakový simulátor s komponentou ETCS, nebudou v komunikaci mezi EVC a RBC v simulátoru ETCS použity. Porovnání ID všech packetů potřebných pro komunikaci mezi EVC a RBC v simulátoru ETCS mezi verzemi 2.3.0, 3.4.0 a 3.6.0 kapitoly 8 subsetu 026 je uvedeno v tabulce 3.3.

Tabulka 3.3 Porovnání ID packetů potřebných pro komunikaci mezi EVC a RBC v simulátoru ETCS, mezi verzemi 2.3.0, 3.4.0 a 3.6.0 kapitoly 8 subsetu 026 System Requirements Specification

Stejná ID packetů pro verze 2.3.0, 3.4.0 a 3.6.0	Z RBC do EVC: 3, 5, 39, 51, 41, 42, 45, 57, 58, 65, 66, 68, 70, 71, 72, 76, 79, 131, 138, 139, 140, 21, 27, 49, 80, 63, 15 Z EVC do RBC: 0, 1, 11, 4, 9
Stejná ID packetů pro verze 3.4.0 a 3.6.0 vyjma ID současně obsažených ve verzi 2.3.0	Z RBC do EVC: 88, 40, 52, 64, 69 Z EVC do RBC: 5
Stejná ID packetů pro verze 2.3.0 a 3.4.0 vyjma ID současně obsažených ve verzi 3.6.0	Z EVC do RBC: 3
ID packetů pro verzi 3.6.0 vyjma ID současně obsažených ve verzích 2.3.0 a 3.4.0	Z EVC do RBC: 2

Packet se stejným ID může mít v různých verzích kapitoly 7 subsetu 026 odlišnou vnitřní strukturu a odlišný název.

Rozdíly a podobnosti packetů potřebných pro komunikaci mezi EVC a RBC v simulátoru ETCS mezi verzemi 2.3.0, 3.4.0 a 3.6.0 kapitoly 7 subsetu 026 jsou podrobně popsány v příloze „Packety“, ve které jsou uvedeny podobnosti a odlišnosti ve vnitřní struktuře a názvu každého packetu mezi zmíněnými verzemi.

3.7.3 Rozdíly a podobnosti zpráv mezi verzemi 2.3.0, 3.4.0 a 3.6.0 subsetu 026

Verze 2.3.0, 3.4.0 a 3.6.0 kapitoly 8 subsetu 026 obsahují stejná a odlišná ID zpráv potřebných pro komunikaci mezi EVC a RBC. V komunikaci mezi EVC a RBC v simulátoru ETCS se budou používat všechny zprávy uvedené v kapitole 8, vyjma zprávy z EVC do RBC s ID 153 a zprávy z RBC do EVC s ID 37. Tyto zprávy, po dohodě s panem doc. Ing. Martinem Lesem, Ph.D., nebudou v komunikaci mezi EVC a RBC v simulátoru ETCS použity. Porovnání ID všech zpráv potřebných pro komunikaci mezi EVC a RBC v simulátoru ETCS mezi verzemi 2.3.0, 3.4.0 a 3.6.0 kapitoly 8 subsetu 026 je uvedeno v tabulce 3.4.

Tabulka 3.4 Porovnání ID zpráv potřebných pro komunikaci mezi EVC a RBC v simulátoru ETCS mezi verzemi 2.3.0, 3.4.0 a 3.6.0 kapitoly 8 subsetu 026 System Requirements Specification

Stejná ID zpráv pro verze 2.3.0, 3.4.0 a 3.6.0	Z EVC do RBC: 129, 130, 132, 136, 137, 138, 146, 147, 149, 150, 154, 155, 156, 157, 159 Z RBC do EVC: 2, 3, 6, 8, 9, 15, 16, 18, 24, 27, 28, 32, 33, 34, 39, 40, 41, 43, 45
Stejná ID zpráv pro verze 3.4.0 a 3.6.0 vyjma ID současně obsažených ve verzi 2.3.0	Z EVC do RBC: 158
Stejná ID zpráv pro verze 2.3.0 a 3.4.0 vyjma ID současně obsažených ve verzi 3.6.0	Z RBC do EVC: 38

Zpráva se stejným ID může mít v různých verzích kapitoly 8 subsetu 026 odlišnou vnitřní strukturu a odlišný název.

Rozdíly a podobnosti zpráv potřebných pro komunikaci mezi EVC a RBC v simulátoru ETCS mezi verzemi 2.3.0, 3.4.0 a 3.6.0 kapitoly 8 subsetu 026 jsou podrobně popsány v příloze „Zprávy“, ve které jsou uvedeny podobnosti a odlišnosti ve vnitřní struktuře a názvu každé zprávy mezi zmíněnými verzemi.

3.7.4 Rozdíly a podobnosti komunikace mezi EVC a RBC na straně EVC mezi verzemi 2.3.0, 3.4.0 a 3.6.0 subsetu 026

Ve verzích 2.3.0 a 3.4.0 subsetu 026 může být iniciátorem komunikace mezi EVC a RBC jak EVC, tak RBC. Ve verzi 3.6.0 tohoto subsetu může být iniciátorem komunikace pouze EVC. Ve všech těchto verzích je iniciátorem ukončení komunikace EVC.

Ve verzích 2.3.0, 3.4.0 a 3.6.0 je komunikace mezi EVC a RBC navázána a ukončena stejným způsobem.

Ve verzích 2.3.0, 3.4.0 a 3.6.0 je definováno, že zpráva je přijata EVC pouze, když projde přes všechna ověření popsaná níže. Tato ověření se neliší mezi jednotlivými verzemi. První kontrolou je kontrola časového razítka a ID poslední relevantní balízové skupiny v přijímané zprávě. Tato kontrola je prováděna v souladu s kapitolou 3 subsetu 026 a pouze v některých módech (mód je režim palubní části ETCS, od kterého se odvíjí druh aktivity ETCS na vozidle). Konkrétní módy se liší mezi verzemi 2.3.0, 3.4.0 a 3.6.0. Pokud zpráva prošla přes toto ověření, následuje kontrola, zda zpráva může být přijata ve stávající aplikační úrovni. Pokud zpráva prošla i přes toto ověření, následuje kontrola, zda přijímaná zpráva může být přijata v módu, ve kterém je v okamžiku přijetí palubní část ETCS.

3.8 Požadavky proměnných, packetů, zpráv pro komunikaci mezi EVC a RBC a komunikačního modulu pro komunikaci mezi EVC a RBC na straně EVC

Zásadním požadavkem je, na základě verzí 2.3.0, 3.4.0 a 3.6.0 subsetu 026 System Requirements Specification, vytvořit pro každou z těchto verzí simulátoru ETCS prototypy odpovídající verze proměnných uvedených v příloze „Proměnné“, packetů uvedených v příloze „Packety“ a zpráv

uvedených v příloze „Zprávy“ a prototyp příslušné verze komunikačního modulu pro komunikaci mezi EVC a RBC na straně EVC. Tento komunikační modul by odesílal a přijímal zprávy od RBC prostřednictvím MQTT brokera.

Níže jsou uvedeny požadavky vzniklé po analýze současného řešení komunikace mezi EVC a RBC v demu simulátoru ETCS a po analýze požadavků kladených na proměnné, packety, zprávy a komunikaci s RBC ze strany EVC uvedených v kapitolách 7, 8, 3 a 4 subsetu 026. Tyto požadavky se vztahují k libovolné z verzí 2.3.0, 3.4.0, 3.6.0 ETCS simulátoru.

3.8.1 Funkční požadavky

Funkční požadavky jsou nároky, které se vztahují k funkcionalitě cílové aplikace. Následující sekce popisuje funkční požadavky kladené na proměnné uvedené v příloze „Proměnné“, packety uvedené v příloze „Packety“, zprávy udané v příloze „Zprávy“ a na komunikační modul pro komunikaci mezi EVC a RBC na straně EVC.

3.8.1.1 Funkční požadavky proměnných

- **F1 – Vytvoření knihoven pro verze proměnných 2.3.0, 3.4.0, 3.6.0**

  Všechny proměnné z přílohy „Proměnné“ se budou v simulátoru ETCS používat pro komunikaci mezi EVC a RBC jak na straně EVC, tak i na straně RBC. Proměnných každé verze z verzí 2.3.0, 3.4.0, 3.6.0 je velký počet. Z těchto důvodů proměnné každé verze budou sjednoceny v knihovně odpovídající této verzi.

- **F2 – Vytvoření proměnné**

  Každá proměnná jazyka ETCS, mající konkrétní hodnotu, bude implementována jako instance třídy zastupující tuto proměnnou. Toto rozhodnutí bylo oproti zjednodušené implementaci v demu simulátoru ETCS zvoleno ze třech důvodů. Zaprvé, v kódu dema simulátoru ETCS není vždy jasné, že se jedná o proměnnou jazyka ETCS. Je to proto, že každá proměnná jazyka ETCS je v demu uvedena jako proměnná mající bezznaménkový celočíselný typ a není tak na první pohled jasné, že se jedná o proměnnou jazyka ETCS. Za druhé, při použití v kódu proměnné jazyka ETCS není v demu vždy zřejmě, že tato proměnná v sobě obsahuje platnou hodnotu. Za třetí, pro velký počet proměnných jazyka ETCS nejsou v demu napsány funkce pro ověření platnosti hodnoty ukládané do proměnné, proto se v aplikaci ETCS opakují kusy kódu, ve kterých se ověřuje hodnota ukládaná do proměnné.

  Instanci třídy reprezentující proměnnou jazyka ETCS bude možné vytvořit buď pomocí konstruktoru s proměnnou, která bude mít defaultní hodnotu nastavenou na nulu, nebo konstruktoru s JSON objektem. Vytvoření instance takové třídy pomocí konstruktoru s JSON objektem se bude používat při přijetí moduly EVC a RBC zpráv v JSON, zahrnujících příslušnou proměnnou jazyka ETCS.

- **F3 – Získání názvu proměnné**

  Každá třída reprezentující proměnnou jazyka ETCS bude obsahovat neveřejnou členskou proměnnou, ve které bude uložen název příslušné proměnné jazyka ETCS podle kapitoly 7 subsetu 026 System Requirements Specification. Tento název bude možné získat bez možnosti ho změnit v příslušné členské proměnné.

- **F4 – Získání hodnoty proměnné**

  Každá třída zastupující proměnnou jazyka ETCS bude zahrnovat neveřejnou členskou proměnnou mající celočíselný datový typ, ve které bude uložena hodnota příslušné proměnné jazyka ETCS. Tuto hodnotu bude možné získat bez možnosti ji změnit v příslušné členské proměnné.

- **F5 – Nastavení hodnoty proměnné**

  Po vytvoření instance třídy reprezentující proměnnou jazyka ETCS bude možné změnit hodnotu v členské proměnné obsahující hodnotu příslušné proměnné jazyka ETCS.

- **F6 – Výpis proměnné na konzoli**

  Instanci třídy reprezentující proměnnou jazyka ETCS bude možné vypsat na konzoli. Na konzoli budou vypsány název a hodnota příslušné proměnné.

- **F7 – Převod proměnné do JSON**

  EVC do RBC nebo z RBC do EVC zpráv v JSON obsahujících příslušnou proměnnou jazyka ETCS.

- **F8 – Ověření hodnoty proměnné**

  Třída zastupující proměnnou jazyka ETCS bude mít metodu pro ověření platnosti hodnoty uložené v proměnné obsahující hodnotu příslušné proměnné jazyka ETCS. Tato metoda se bude volat v konstruktorech třídy ihned po uložení hodnoty do příslušné členské proměnné a při dalších změnách hodnoty této členské proměnné.

3.8.1.2 Funkční požadavky packetů

- **F1 – Vytvoření knihoven pro verze packetů 2.3.0, 3.4.0, 3.6.0**

  Všechny packety z přílohy „Packety“ se budou v simulátoru ETCS používat pro komunikaci mezi EVC a RBC jak na straně EVC, tak i na straně RBC. Packetů každé verze z verzí 2.3.0, 3.4.0, 3.6.0 je velké množství. Proto budou packety každé verze sjednoceny v knihovně odpovídající této verzi.

- **F2 – Vytvoření packetu**

  Každý packet mající konkrétní hodnoty bude implementován jako instance třídy zastupující tento packet.

  Instanci třídy reprezentující packet bude možné vytvořit buď pomocí konstruktoru s proměnnými, nebo konstruktoru s JSON objektem. Vytvoření instance takové třídy pomocí konstruktoru s JSON objektem se bude používat při přijetí moduly EVC a RBC zpráv v JSON, zahrnujících příslušný packet.

- **F3 – Získání hodnot packetu**

  Každá třída zastupující packet bude obsahovat neveřejné členské proměnné, ve kterých budou uloženy hodnoty proměnných jazyka ETCS, které jsou obsaženy v tomto packetu. Tyto hodnoty bude možné získat bez možnosti je změnit v příslušných členských proměnných.

- **F4 – Výpis packetu na konzoli**

  Instanci třídy reprezentující packet bude možné vypsat na konzoli. Na konzoli budou vypsány název příslušného packetu a proměnné v něm obsažené.

- **F5 – Převod packetu do JSON**

  Třída zastupující packet bude mít metodu pro převod členských proměnných do JSON. Tato funkcionalita bude použita při odesílání moduly EVC a RBC zpráv v JSON, obsahujících příslušný packet.

- **F6 – Ověření délky a ID packetu**

  Třída zastupující packet bude mít metodu pro ověření délky packetu a jeho ID. Metoda bude kontrolovat, zda je součet délek všech proměnných roven hodnotě proměnné reprezentující délku packetu v bitech a zda hodnota proměnné reprezentující ID packetu se rovná ID zastoupeného packetu. Tato metoda se bude volat v konstruktorech třídy ihned po uložení hodnot do členských proměnných.

3.8.1.3 Funkční požadavky zpráv

- **F1 – Vytvoření knihoven pro verze zpráv 2.3.0, 3.4.0, 3.6.0**

  Všechny zprávy z přílohy „Zprávy“ se budou používat v simulátoru ETCS pro komunikaci mezi EVC a RBC, jak na straně EVC, tak i na straně RBC. Zpráv každé z verzí 2.3.0, 3.4.0, 3.6.0 je velký počet. Z těchto důvodů budou zprávy každé verze sjednoceny v knihovně odpovídající této verze.

- **F2 – Vytvoření zprávy**

  Každá zpráva mající konkrétní hodnoty bude implementována jako instance třídy zastupující tuto zprávu.

  Instanci třídy reprezentující zprávu bude možné vytvořit buď pomocí konstruktoru s proměnnými nebo konstruktoru s JSON objektem. Vytvoření instance takové třídy pomocí konstruktoru s JSON objektem se bude používat při přijetí moduly EVC a RBC zpráv v JSON.

- **F3 – Získání hodnot zprávy**

  Každá třída zastupující zprávu bude obsahovat neveřejné členské proměnné, ve kterých budou uloženy hodnoty proměnných jazyka ETCS a případně packetů, které jsou obsaženy v této zprávě. Tyto hodnoty bude možné získat bez možnosti je změnit v příslušných členských proměnných.

- **F4 – Výpis zprávy na konzoli**

  Instanci třídy reprezentující zprávu bude možné vypsat na konzoli. Na konzoli bude vypsán název příslušné zprávy, proměnné a případně packety v ní obsažené.

- **F5 – Převod zprávy do JSON**

  Třída zastupující zprávu bude mít metodu pro převod členských proměnných do JSON. Tato funkcionalita bude použita při odesílání moduly EVC a RBC zpráv v JSON.

- **F6 – Ověření délky a ID zprávy**

  Třída zastupující zprávu bude mít metodu pro ověření délky a ID zprávy. Metoda bude kontrolovat, zda je součet délek všech proměnných a packetů uvedený v bytech a zaokrouhlený nahoru roven hodnotě proměnné reprezentující délku zprávy v bytech a zda hodnota proměnné reprezentující ID zprávy se rovná ID zastoupené zprávy. Tato metoda se bude volat v konstruktorech třídy ihned po uložení hodnot do členských proměnných.

3.8.1.4 Funkční požadavky komunikačního modulu

- **F1 – Komunikace s modulem RBC prostřednictvím MQTT brokera**

  Komunikační modul bude komunikovat s modulem RBC prostřednictvím MQTT brokera.

- **F2 – Přijetí zpráv v JSON**

  Komunikační modul bude přijímat zprávy z RBC do EVC uvedené v příloze „Zprávy“ v JSON.

- **F3 – Odeslání zpráv v JSON**

  Komunikační modul bude odesílat zprávy z EVC do RBC uvedené v příloze „Zprávy“ v JSON.

- **F4 – Ověření časového razítka v přijímané zprávě**

  Komunikační modul bude ověřovat časové razítko v přijímané zprávě od RBC podle požadavků kapitoly 3 subsetu 026 System Requirements Specification.

- **F5 – Ověření ID poslední relevantní balízové skupiny v přijímané zprávě**

  Komunikační modul bude ověřovat ID poslední relevantní balízové skupiny ve zprávě přijímané od RBC podle požadavků kapitoly 3 subsetu 026 System Requirements Specification.

- **F6 – Ověření, zda přijímaná zpráva může být přijatá ve stávajících módu a aplikační úrovni**

  Komunikační modul bude ověřovat, zda zpráva přijímaná od RBC může být přijatá v módu a aplikační úrovni, ve kterých je v okamžiku přijetí palubní část ETCS. Toto ověření se bude provádět na základě požadavků kapitoly 4 subsetu 026 System Requirements Specification.

- **F7 – Ověření, zda zpráva může být přijata na základě dříve přijatých zpráv**

  Některé zprávy od RBC mohou být přijaty pouze po obdržení předcházejících zpráv od RBC s určitými ID. Některé zprávy od RBC naopak nemohou být přijaty po obdržení předcházejících zpráv od RBC s určitými ID. Komunikační modul bude ověřovat, zda přijímaná zpráva od RBC může být přijatá na základě dříve přijatých zpráv od RBC. Tato kontrola se bude provádět na základě požadavků kapitoly 3 subsetu 026 System Requirements Specification.

3.8.2 Nefunkční požadavky

V této sekci jsou probrány nefunkční požadavky. Nefunkční požadavky se na rozdíl od funkčních zaměřují na nároky cílové aplikace na software, a to například z hlediska spolehlivosti, výkonu a rozšiřitelnosti.

- **N1 – Použití programovacího jazyka C++**

  Výše zmíněné zprávy, packety, proměnné a komunikační modul pro komunikaci mezi EVC a RBC na straně EVC budou implementovány ve všech verzích (2.3.0, 3.4.0 a 3.6.0) v jazyce C++. Je to ze dvou důvodů. Zaprvé, moduly EVC a RBC v aplikaci ETCS jsou napsány v C++. Za druhé, tento jazyk byl zvolen kvůli vlastním zkušenostem s programováním v něm.

- **N2 – Použití MQTT brokera**

  Komunikace mezi moduly v simulátoru ETCS je realizována pomocí MQTT brokera. Po dohodě s vedoucím bakalářské práce bylo rozhodnuto, že se to nebude měnit.

- **N3 – Použití JSON**

  K použití formátu JSON bylo přistoupeno proto, že tento formát je jednoduchý jak pro zápis, tak na čtení a zároveň je snadné pochopit data v něm uložená. S použitím tohoto formátu mám taktéž osobní zkušenosti.

- **N4 – Kompatibilita s různými verzemi ETCS simulátoru**

  Pro každou z verzí (2.3.0, 3.4.0 a 3.6.0) ETCS simulátoru budou implementovány knihovny zpráv, packetů, proměnných a komunikační modul příslušné verze.

- **N5 – Rozšiřitelnost**

  Proměnné, packety, zprávy a komunikační modul budou implementovány ve všech verzích (2.3.0, 3.4.0 a 3.6.0) způsobem, který v budoucnu umožní je jednoduše rozšířit o další funkcionality a umožní snadno přidávat další proměnné, packety a zprávy do aplikace ETCS.

- **N6 – Komunikační modul nebude ukládat obsah přijatých zpráv do vnitřních struktur modulu EVC**

  Komunikační modul nebude ukládat obsah přijatých zpráv do vnitřních struktur modulu EVC, a to z důvodu, že by tento požadavek byl nad rámec bakalářské práce. Komunikační modul po přijetí zprávy zavolá určitou funkci pro uložení obsahu zprávy do vnitřních struktur modulu EVC.

- **N7 – Zprávy nebudou obsahovat padding**

  Podle kapitoly 8 v případě, kdy délka zprávy v bytech ukládaná do proměnné zprávy reprezentující délku zprávy v bytech nevychází celočíselná, přidá se na konec zprávy padding. [14] Z důvodu zjednodušení nebudou zprávy v simulátoru ETCS obsahovat padding. Hodnota uložená v proměnné zprávy reprezentující délku zprávy v bytech bude padding zahrnovat. Před kontrolou rovnosti vypočtené skutečné délky zprávy v bytech a hodnoty uložené v proměnné zprávy reprezentující délku zprávy v bytech se vypočtená délka vždy zaokrouhlí nahoru.

- **N8 – Pořadí dat v přenosu nebude odpovídat pořadí datových prvků uvedených ve zprávě**

  Podle kapitoly 8 pořadí dat v přenosu musí odpovídat pořadí datových prvků uvedených ve zprávě. [14] V simulátoru ETCS se nebude z důvodu zjednodušení pořadí dat ve zprávě při přenosu zachovávat.

```
class Message {
public:
    Message (std::uint8_t id, std::uint16_t length, std::uint32_t train);
    explicit Message(const nlohmann::json &j);
    virtual ~Message();
    /**
     * Creates from attributes a JSON object.
     */
    virtual nlohmann::json ToJson() const = 0;
    /**
     * Prints message to stream.
     */
    virtual void Print ( std::ostream & os ) const = 0;
    /**
     * Prints common attributes.
     */
    void PrintIt ( std::ostream & os ) const;
    friend std::ostream & operator << ( std::ostream & os, const Message & message);
    std::uint8_t  GetId() const;
    std::uint16_t GetLength() const;
    std::uint32_t GetTTrain() const;
private:
    /**
     * Checks, if set beforehand attributes are valid according to subset 7.
     */
    bool Valid() const;
protected:
    /**
     * Creates from common attributes a JSON object.
     */
    nlohmann::json ToJsonIt() const;
    /**
     * According to subset 7.
     * Means: Message identifier.
     */
    std::uint8_t NID_MESSAGE;
    /**
     * According to subset 7.
     * Means: Message length in bytes.
     * Values: from 0 to 1023.
     */
    std::uint16_t L_MESSAGE;
    /**
     * According to subset 7.
     * Means: Trainborne clock.
     */
    std::uint32_t T_TRAIN;
};
```

Výpis kódu 3.5 Deklarace třídy Message v simulátoru ETCS

```
class MessageRBC : public Message {
public:
    MessageRBC(std::uint8_t id, std::uint16_t length, std::uint32_t train,
               std::uint8_t mack, std::uint32_t lrbg);
    explicit MessageRBC(const nlohmann::json &j);
    virtual ~MessageRBC();
    /**
     * Prints common attributes.
     */
    void PrintRBC ( std::ostream & os ) const;
    std::uint8_t GetMAck() const;
    std::uint32_t GetNIDLrbg() const;
private:
    /**
     * Checks, if set beforehand attributes are valid according to subset 7.
     */
    bool Valid() const;
protected:
    /**
     * Creates from common attributes a JSON object.
     */
    nlohmann::json ToJsonRBC() const;
    /**
     * According to subset 7.
     * Means: Qualifier for acknowledgement request.
     * Values: from 0 to 1.
     */
    std::uint8_t M_ACK;
    /**
     * According to subset 7.
     * Means: Identity of  last relevant balise group.
     * Values: from 0 to 16777215.
     */
    std::uint32_t NID_LRBG;
};
```

Výpis kódu 3.6 Deklarace třídy MessageRBC v simulátoru ETCS

```
class MessageEngineEVC : public Message {
public:
    MessageEngineEVC(std::uint8_t id, std::uint16_t length, std::uint32_t train,
                     std::uint32_t engine);
    explicit MessageEngineEVC(const nlohmann::json &j);
    virtual ~MessageEngineEVC();
    /**
     * Prints common attributes.
     */
    void PrintEVC ( std::ostream & os ) const;
    std::uint32_t GetNIDEngine() const;
private:
    /**
     * Checks, if set beforehand attributes are valid according to subset 7.
     */
    bool Valid() const;
protected:
    /**
     * Creates from common attributes a JSON object.
     */
    nlohmann::json ToJsonEVC() const;
    /**
     * According to subset 7.
     * Means: Onboard ETCS identity.
     * Values: from 0 to 16777215.
     */
    std::uint32_t NID_ENGINE;
};
```

Výpis kódu 3.7 Deklarace třídy MessageEngineEVC v simulátoru ETCS

Kapitola 4
Návrh

Tato kapitola popisuje návrh knihoven proměnných, packetů a zpráv. Popisuje také návrh komunikačního modulu pro komunikaci mezi EVC a RBC na straně EVC.

V této kapitole jsou verze 2.3.0, 3.4.0, 3.6.0 subsetu 026 uváděny zkráceně pouze jako verze 2.3.0, verze 3.4.0, verze 3.6.0.

Všechny navržené knihovny jsou dynamické. Vytvoření a použití dynamických knihoven oproti statickým knihovnám bylo zvoleno ze dvou důvodů. Zaprvé, pokud je dynamická knihovna zaktualizována, není potřeba znovu kompilovat moduly aplikace ETCS používající tuto knihovnu. Za druhé, v případech, kdy několik modulů aplikace ETCS používá stejnou knihovnu, použití dynamické knihovny ušetří místo v paměti.

4.1 Návrh proměnných

Každá proměnná jazyka ETCS je reprezentována třídou zastupující tuto proměnnou.

4.1.1 Třída Variable

Všechny třídy reprezentující proměnné jazyka ETCS dědí z generické třídy Variable. Třída Variable je znázorněna na obrázku 4.1.

Třída Variable má parametry T a P. Parametr T je datovým typem používaným pro ukládaní hodnot proměnné. U proměnných majících bezznaménkový datový typ, parametr P udává délku proměnné v bitech. U proměnných majících znaménkový datový typ, parametr P udává délku proměnné v bitech pokrácenou o 1.

Třída Variable má atributy označující název a hodnotu proměnné.

Tato třída má 2 konstruktory. První konstruktor je potřebný pro získaní obsahu proměnné z JSON objektu a druhý je nutný k vytvoření proměnné s hodnotou datového typu T. První konstruktor má mezi svými proměnnými nejenom JSON objekt, ale také i dvě proměnné typu std::string. Jedna z těchto proměnných uvádí pořadí. V JSON objektu může být několik proměnných jazyka ETCS se stejným názvem, proto je potřeba přesně vědět pořadí proměnné, jejíž obsah bude získán. Druhá proměnná uvádí název proměnné.

Třída Variable má dvě virtuální metody pro nastavení hodnoty proměnné po jejím vytvoření. Podobně jako u konstruktorů, jedná z těchto metod přijímá ke zpracovaní JSON objekt, druhá metoda přijímá parametr datového typu T.

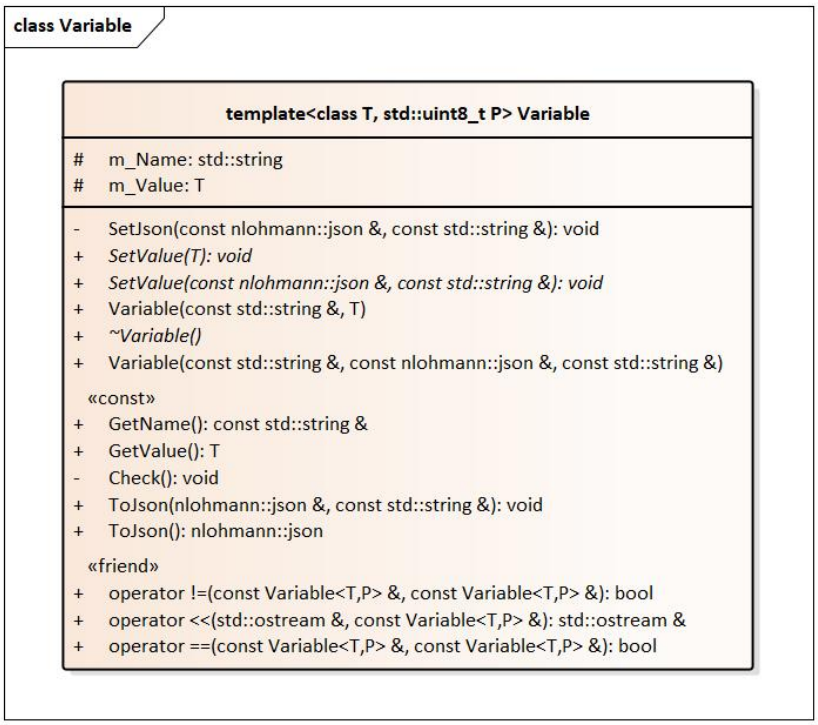
Třída Variable má privátní metody SetJson a Check. Metoda SetJson se volá v konstruktoru s JSON objektem a v metodě s JSON objektem. Metoda Check kontroluje, zda hodnota ukládaná do proměnné je v mezích stanovených parametrem P.

Třída Variable má metody pro získaní hodnot svých členských proměnných. Pomocí těchto metod nelze měnit obsah členských proměnných.

Třída Variable má přetížený operátor << pro výpis proměnné do streamu.

Třída Variable má dvě metody pro převod proměnné do formátu JSON. Jedná z těchto metod vrací JSON objekt a hodí se při vkládaní proměnné v JSON do pole v JSON objektu. Druhá metoda se hodí pro ukládaní proměnné přímo do JSON objektu. Při volaní této metody je potřeba uvést pořadí proměnné v JSON objektu.

Třída Variable má přetížené operátory == a !=, které se hodí při porovnání dvou proměnných.



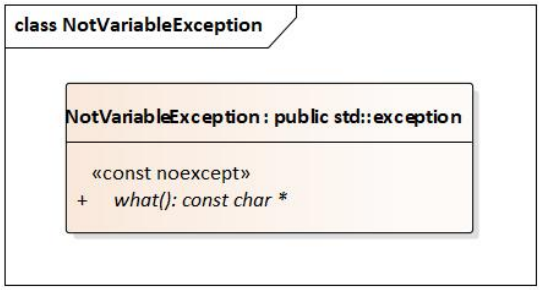
Obrázek 4.1 Třída Variable

4.1.2 Výjimka NotVariableException

Pokud při ukládaní hodnoty do proměnné dojde k chybě, vyhodí se vlastní výjimka NotVariableException. Třída NotVariableException je znázorněna na obrázku 4.2. Tato třída dědí z std::exception a přepisuje metodu what.

4.1.3 Rozdělení proměnných na knihovny

Následující rozdělení proměnných na knihovny se řídí rozdělením proměnných popsaným v příloze Proměnné. Podle popsaného rozdělení se třídy reprezentující proměnné jazyka ETCS, potřebné pro komunikaci mezi EVC a RBC, rozdělí na šest knihoven: knihovnu proměnných společných pro verze 2.3.0, 3.4.0 a 3.6.0, knihovnu proměnných společných pro verze 2.3.0 a 3.4.0 vyjma verze 3.6.0, knihovnu proměnných společných pro verze 3.4.0, 3.6.0 vyjma verze 2.3.0, knihovnu proměnných verze 2.3.0, knihovnu proměnných verze 3.4.0 a knihovnu proměnných verze 3.6.0. Poslední 3 knihovny používají výše uvedené knihovny společných proměnných.



Obrázek 4.2 Třída NotVariableException

Třídy Variable a NotVariableException jsou v knihovně proměnných společných pro verze 2.3.0, 3.4.0, 3.6.0.

4.1.4 Třídy proměnných

Každá třída reprezentující proměnnou jazyka ETCS má dva konstruktory. Jeden z nich je s JSON objektem. Druhý konstruktor je s parametrem obsahujícím defaultní nulovou hodnotu.

Některým třídám nestačí kontrola prováděná v metodě Check v nadtřídě. Kontrola hodnoty ukládané do proměnných reprezentovaných těmito třídami je složitější než kontrola prováděná u většiny proměnných jazyka ETCS. Proto třídy takových proměnných obsahují navíc ke dvěma konstruktorům ještě privátní metodu Check. Tato metoda provádí dodatečnou kontrolu. Tyto třídy také přepisují metody získaní hodnoty proměnné z JSON objektu z třídy Variable.

4.2 Návrh packetů

Každý packet jazyka ETCS je reprezentován třídou zastupující tento packet.

Třídy packetů používají knihovny proměnných. Členské proměnné takových tříd jsou objekty tříd proměnných jazyka ETCS. Některé třídy mají mezi členskými proměnnými std::vector, obsahující objekty třídy proměnné jazyka ETCS, objekty třídy copy_ptr a std::vector, obsahující objekty této třídy.

Všechny členské proměnné tříd packetů nejsou konstantní. Vozidlová část ETCS může ukládat packety, proto by se operátor přiřazení mohl v budoucnu hodit.

4.2.1 Třída copy_ptr

V kapitole 3.6.2 je zmíněno, že v demu simulátoru ETCS v případech s rozhodovací proměnnou se používá std::unique_ptr. Použití std::unique_ptr zaručuje, že třída reprezentující nějaký packet je jediným vlastníkem obsahu uloženého v std::unique_ptr. Tato záruka je důležitá, neboť při změně tohoto obsahu by se mohla změnit skutečná délka packetu a hodnota uložená v proměnné L_PACKET by přestala být platnou.

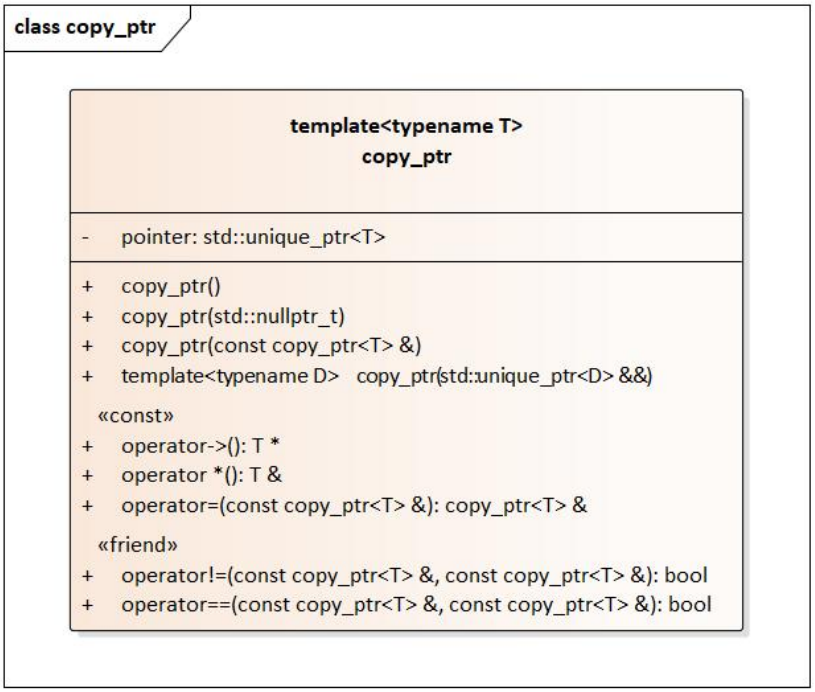
Použití std::unique_ptr přímo ve třídě packetu by donutilo k definovaní kopírujícího konstruktoru a operátoru přiřazení v této třídě. Proto bylo navrhnuto řešení umožňující používat defaultní kopírující konstruktor a operátor přiřazení.

Byla navrhnuta generická třída copy_ptr s parametrem T. Parametr T musí mít v sobě funkci Clone, kterou bude používat copy_ptr. Generická třída copy_ptr „obaluje“ std::unique_ptr. Třída copy_ptr je znázorněna na obrázku 4.3.

Třída copy_ptr má několik konstruktorů, které se používají ve třídách packetů, kopírující konstruktor a operátor přiřazení. Třída copy_ptr má přetížené operátory != a ==. Tyto operá-

tory se používají při kontrole ve třídách packetů, zda se copy_ptr rovná nullptr (použití nullptr by mělo být v případech s rozhodovací proměnou zakázáno). Třída copy_ptr má také přetížené operátory * a ->. Tyto operátory se používají pro získaní obsahu std::unique_ptr.

Ve všech třídách packetů copy_ptr ukazuje na konstantní objekt, a to z důvodu, že obsah objektu nesmí být měněn.



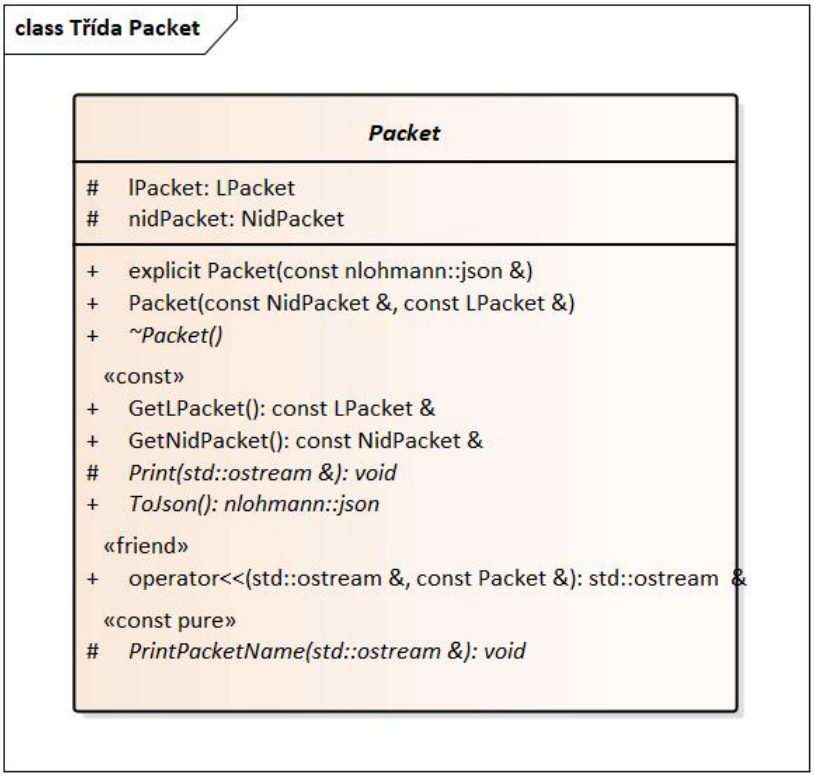
Obrázek 4.3 Třída copy_ptr

4.2.2 Hierarchie tříd packetů

V kapitole 3.6.2 se popisuje hierarchie tříd packetů v demu simulátoru ETCS a důvody vytvoření této hierarchie. V návrhu packetů pro různé verze byla hierarchie tříd pozměněna, ale důvody jejího použití zůstávají stejné. Tedy princip hierarchie tříd je založen na společném obsahu packetů a obsahu metod těchto tříd. Každá třída dědí z třídy mající členské proměnné a metody, které jsou společné pro několika tříd packetů.

Nejvyšší třída v hierarchii tříd je abstraktní třída Packet. Třída Packet je uvedena v diagramu 4.4. Třída Packet má dvě členské proměnné – objekt třídy NidPacket a objekt třídy LPacket. Tyto dvě třídy jsou v knihovně proměnných společných pro verze 2.3.0, 3.4.0 a 3.6.0. Třída Packet má konstruktor s JSON objektem a konstruktor s proměnnými. Třída Packet má přetížený operátor << pro výpis packetu do streamu. Tato třída má také virtuální funkci pro výpis do streamu packetu podle struktury definované v kapitole 7 subsetu 026. Tato funkce se používá v přetíženém operátoru <<. Třída má virtuální metodu pro převod packetu do formátu JSON. Třída Packet má metody pro získaní hodnot svých členských proměnných. Pomocí těchto metod nelze měnit obsah členských proměnných. Třída Packet deklaruje abstraktní metodu pro výpis názvu packetu do streamu.

Třídy, které přepisují metody své nadtřídy, v těle těchto metod na začátku volají tyto metody nadtřídy.



Obrázek 4.4 Třída Packet

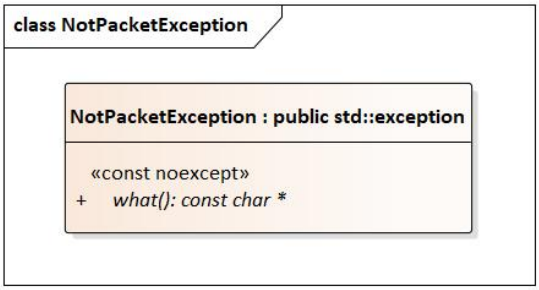
Pouze třídy reprezentující konkrétní packety nejsou abstraktní. Všechny ostatní třídy v hierarchii tříd packetů jsou abstraktními. Je to ze dvou důvodů. Zaprvé, v aplikaci ETCS, po zjištění ID packetu, se mohou volat gettery, které obsahuje jenom třída packetu s tímto konkrétním ID. Pokud by tyto třídy nebyly abstraktními, mohlo by dojit k zpracovaní objektu takové třídy a volaní getteru nad ním, což by vedlo k chybě. Za druhé, vytvoření těchto tříd jako abstraktních taktéž eliminuje možnost, že se mezi EVC a RBC pošle zpráva s neplatným obsahem.

Některé třídy v hierarchii tříd packetů mají privátní metodu Check. Metoda Check se volá v konstruktorech těchto tříd. Metoda Check kontroluje, zda ID uložené do objektu třídy NidPacket je shodné s ID odpovídajícího packetu. Dále kontroluje, zda se skutečná délka packetu rovná délce uložené v objektu třídy LPacket. U některých packetů tato funkce také kontroluje, zda se copy_ptr na abstraktní bázovou třídu obsahující rozhodovací proměnnou nerovná nullptr a zda se počet prvků v std::vector rovná hodnotě uložené v objektu třídy NIter, která reprezentuje proměnnou N_ITER.

Některé třídy packetů mají metodu Clone. Metoda Clone je ve třídách packetů, na které mají copy_ptr třídy zpráv. Ostatní třídy v hierarchii tříd packetů nepotřebují tuto metodu.

4.2.3 Výjimka NotPacketException

Pokud při ukládaní hodnot do packetu dojde k chybě, vyhodí se vlastní výjimka NotPacketException. Třída NotPacketException je uvedena v diagramu 4.5. Tato třída dědí z std::exception a přepisuje metodu what.



Obrázek 4.5 Třída NotPacketException

4.2.4 Rozdělení tříd packetů na knihovny

Rozdělení tříd packetů na knihovny se řídí rozdělením packetů popsaným v příloze Packety. Rozdělení se provádí na základě společných a odlišných částí packetu pro různé verze. Podle tohoto principu se všechny třídy z hierarchie tříd rozdělí na šest knihoven: knihovnu packetů společných pro verze 2.3.0, 3.4.0 a 3.6.0, knihovnu packetů společných pro verze 2.3.0 a 3.4.0 vyjma verze 3.6.0, knihovnu packetů společných pro verze 3.4.0, 3.6.0 vyjma verze 2.3.0, knihovnu packetů verze 2.3.0, knihovnu packetů verze 3.4.0 a knihovnu packetů verze 3.6.0. Poslední 3 knihovny používají výše uvedené knihovny společných packetů.

Hierarchie tříd packetů verze 2.3.0 je uvedena v elektronické příloze, ve složce diagrams, v diagramech „Hierarchie tříd packetů z EVC do RBC verze 2.3.0“ a „Hierarchie tříd packetů z RBC do EVC verze 2.3.0“. Zjednodušené znázornění hierarchie (ve třídách packetů jsou uvedeny pouze členské proměnné) je uvedeno v diagramech D.1 a D.2.

Hierarchie tříd packetů verze 3.4.0 je uvedena v elektronické příloze, ve složce diagrams, v diagramech „Hierarchie tříd packetů z EVC do RBC verze 3.4.0“ a „Hierarchie tříd packetů z RBC do EVC verze 3.4.0“. Zjednodušené znázornění hierarchie (ve třídách packetů jsou uvedeny pouze členské proměnné) je uvedeno v diagramech D.3 a D.4.

Hierarchie tříd packetů verze 3.6.0 je uvedena v elektronické příloze, ve složce diagrams, v diagramech „Hierarchie tříd packetů z EVC do RBC verze 3.6.0“ a „Hierarchie tříd packetů z RBC do EVC verze 3.6.0“. Zjednodušené znázornění hierarchie (ve třídách packetů jsou uvedeny pouze členské proměnné) je uvedeno v diagramech D.5 a D.6.

Všechny diagramy hierarchie tříd packetů ukazují, které třídy packetů jsou, ve které knihovně obsaženy. Z diagramu D.1 lze vidět, že knihovna packetů společných pro verze 2.3.0, 3.4.0 vyjma verze 3.6.0 má TrainPacket3 (třídu packetu z EVC do RBC s ID 3) a používá knihovnu packetů společných pro verze 2.3.0, 3.4.0, 3.6.0 k dědění TrainPacket3 z třídy Packet. Není-li v diagramu třída packetu znázorněna v žádné uvedené knihovně znamená to, že se tato třída vyskytuje pouze v knihovně packetů konkrétní verze. Tedy, v diagramu Y třídy packetů TrainPacket1, TrainPacket0, TrainPacket0or1 a TrainPacket11 jsou v knihovně packetů verze 2.3.0. Knihovny packetů konkrétní verze používají pro přístup k packetům společným pro několika verzí knihovny packetů společných pro několika verzí.

Třídy Packet a NotPacketException jsou obsaženy v knihovně packetů společných pro verze 2.3.0, 3.4.0 a 3.6.0.

4.2.5 Hierarchie tříd s rozhodovacími proměnnými

Hierarchie tříd s rozhodovacími proměnnými je stejná pro všechny třídy obsahující tyto proměnné. Bázová třída má rozhodovací proměnnou (objekt třídy reprezentující příslušnou rozhodovací proměnnou), dva konstruktory (jeden s JSON objektem, druhý s proměnnými). Bázová třída má getter pro získání hodnoty rozhodovací proměnné, pomocí kterého nelze měnit obsah této

proměnné. Bázová třída má dále přetížený operátor <<, virtuální metodu Print, která se volá v těle operátoru <<, a virtuální metodu ToJson. Bázová třída deklaruje metodu Clone. Metoda Clone je potřebná z důvodu, že třídy packetů obsahují copy_ptr na bázovou třídu s rozhodovací proměnnou. Bázová třída má virtuální metodu CountLength. Tato metoda vrací sumu délek všech proměnných uložených ve třídě. Metoda CountLength se volá při počítaní skutečné délky packetu v metodě Check ve třídách packetů. Z bázové třídy dědí dvě třídy. První třída zastupuje případ, kdy rozhodovací proměnná má určené hodnoty označující, že po ní následují další proměnné. Druhá třída představuje případ, kdy rozhodovací proměnná nemá tyto určené hodnoty a žádné další proměnné po ní nenásledují. Obě tyto třídy mají dva konstruktory (jeden s JSON objektem, druhý s proměnnými) a privátní metodu Check potřebnou pro ověření, zda rozhodovací proměnná má správné hodnoty podle popisu těchto tříd. Obě tyto třídy implementují metodu Clone. První třída má v členských proměnných proměnné, které následují po rozhodovací proměnné. Tato třída přepisuje metody ToJson, Print a CountLength. První třída má také gettery pro vracení hodnot proměnných, které následují po rozhodovací proměnné. Pomocí těchto getterů nelze měnit obsah těchto proměnných.

Příklad hierarchie tříd s rozhodovací proměnnou je uveden v diagramu 4.6. V tomto diagramu je uvedena bázová třída QNewcountryIf mající objekt třídy reprezentující rozhodovací proměnnou Q_NEWCOUNTRY. Z QNewcountryIf dědí třídy QNewcountry0 a QNewcountry1. Třída QNewcountry0 reprezentuje případ, kdy se proměnná Q_NEWCOUNTRY rovná 0, a proto po ní žádné další proměnné nenásledují. Třída QNewcountry1 zastupuje opačný případ – proměnná Q_NEWCOUNTRY se rovná 1, a proto po ní následují další proměnné.

Všechny bázové třídy s rozhodovací proměnnou by měly být abstraktní. Je to z důvodu, aby copy_ptr na bázovou třídu neukazoval na objekt bázové třídu.

Ve všech diagramech, kde je uvedena hierarchie packetů, je ve třídách packetů copy_ptr vždy na nějakou bázovou třídu s rozhodovací proměnnou. Bázová třída s rozhodovací proměnnou a obě její podtřídy by měly být obsaženy ve stejné knihovně, ve které je obsažena třída packetu mající copy_ptr na tuto bázovou třídu.

4.3 Návrh zpráv

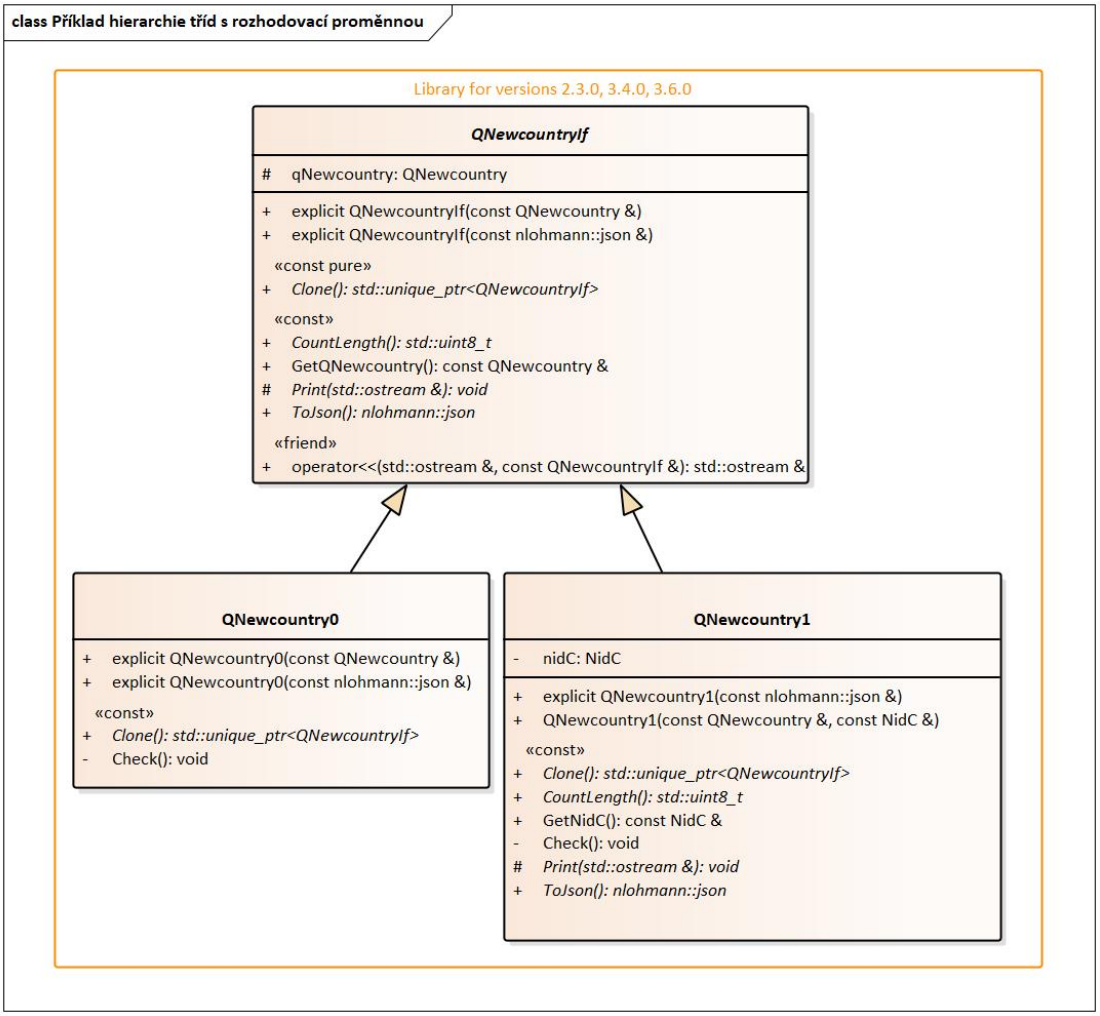
Každá zpráva jazyka ETCS je reprezentována třídou zastupující tuto zprávu.

Pro reprezentaci proměnných a packetů třídy zpráv používají knihovny proměnných a knihovny packetů. Třídy zpráv z RBC do EVC používají pro reprezentaci volitelných packetů objekty podtříd třídy CommonOptionalPackets a std::vector packetů se stejným NID_PACKET, ale s odlišnou Q_DIR. Pro zastupování nutných packetů třídy zpráv z RBC do EVC používají std::vector packetů se stejným NID_PACKET a odlišnou Q_DIR. Pro reprezentaci volitelného packetu používají třídy zpráv z EVC do RBC copy_ptr. Třídy zpráv z EVC do RBC, které musejí mít buď packet s ID 1, nebo packet s ID 0, používají copy_ptr na nadtřídu třídy packetu s ID 0 a třídy packetu s ID 1. Copy_ptr ve všech třídách ukazuje na konstantní objekt, a to z důvodu, že obsah objektu nesmí být změněn. Důvodem použití pro reprezentaci volitelných packetů třídami zpráv z RBC do EVC std::vector a třídami zpráv z EVC do RBC copy_ptr je, že podle bodu 8.4.1.4 kapitoly 8 subsetu 026 mohou mít několik packetů se stejným ID a s odlišným směrem (hodnotou proměnné Q_DIR) pouze zprávy z RBC do EVC.

4.3.1 Hierarchie tříd zpráv

Byla vytvořena určitá hierarchie tříd zpráv. Důvody vytvoření této hierarchie jsou stejné jako důvody vytvoření hierarchie tříd packetů.

Nejvyšší třída v hierarchii tříd je abstraktní třída Message. Třída Message je uvedena v diagramu 4.7. Třída Message má tři členské proměnné – objekt třídy NidMessage, objekt třídy LMessage a objekt třídy TTrain. Tyto tři třídy jsou v knihovně proměnných společných pro



Obrázek 4.6 Příklad hierarchie tříd s rozhodovací proměnnou

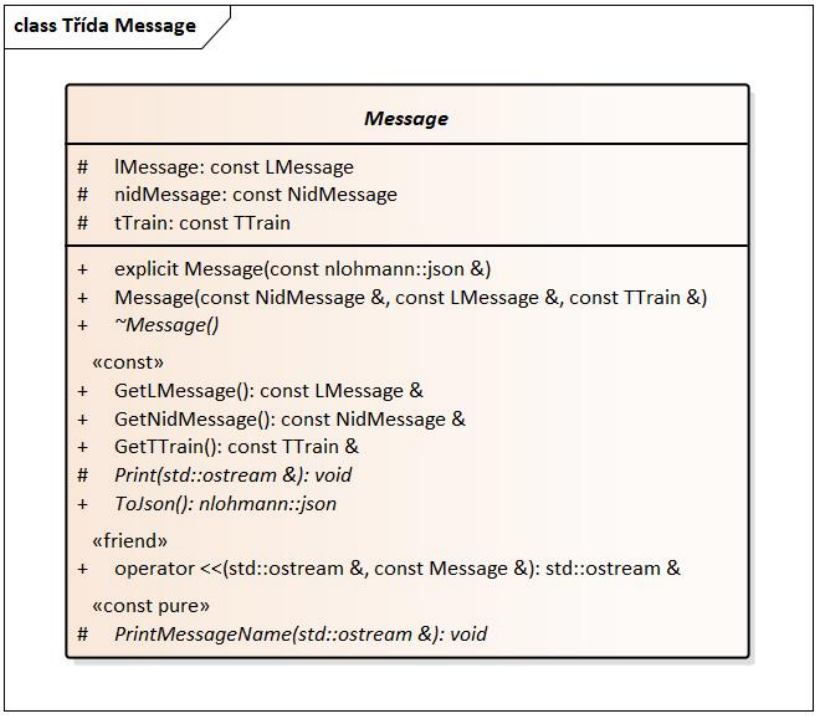
verze 2.3.0, 3.4.0 a 3.6.0. Třída Message má konstruktor s JSON objektem a konstruktor s proměnnými. Třída Message má přetížený operátor << pro výpis zprávy do streamu. Tato třída má také virtuální funkci pro výpis do streamu zprávy podle struktury definované v kapitole 8 subsetu 026. Tato funkce se používá v přetíženém operátoru <<. Třída má virtuální metodu pro převod zprávy do formátu JSON. Třída Message má metody pro získaní hodnot svých členských proměnných. Pomocí těchto metod nelze měnit obsah členských proměnných. Třída Message deklaruje abstraktní metodu pro výpis názvu zprávy do streamu.

Z třídy Message dědí třídy MessageRBC a MessageEVC. Třída MessageRBC odpovídá základní struktuře zprávy z RBC do EVC. Třída MessageEVC odpovídá základní struktuře zprávy z EVC do RBC. Z třídy MessageRBC dědi všechny třídy zpráv z RBC do EVC. Z třídy MessageEVC dědí všechny třídy zpráv z EVC do RBC.

Třídy, které přepisují metody své nadtřídy, v těle těchto metod na začátku volají tyto metody nadtřídy.

Pouze třídy reprezentující konkrétní zprávy nejsou abstraktní. Všechny ostatní třídy v hierarchii tříd zpráv jsou abstraktními. Důvody toho jsou stejné jako u tříd packetů.

Některé třídy v hierarchii tříd mají privátní metodu Check. Metoda Check se volá v konstruktorech těchto tříd. Metoda Check kontroluje, zda ID uložené do objektu třídy NidMessage

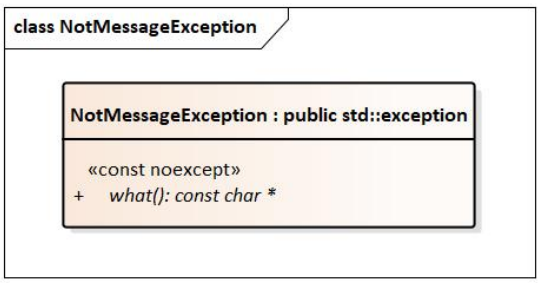


Obrázek 4.7 Třída Message

je shodné s ID odpovídající zprávy. Metoda Check také kontroluje, zda se skutečná délka zprávy rovná délce uložené v objektu třídy LMessage. Ve třídách zpráv z EVC do RBC majících buď packet s ID 0, nebo packet s ID 1, tato funkce také kontroluje, zda se copy_ptr nerovná nullptr. Ve třídách zpráv z RBC do EVC obsahujících vektory packetů se stejným ID tato funkce také kontroluje, zda se ve vektorech packetů se stejným ID nejsou packety se stejnou hodnotou proměnné Q_DIR.

4.3.2 Výjimka NotMessageException

Pokud při ukládaní hodnot do zprávy dojde k chybě, vyhodí se vlastní výjimka NotMessageException. Třída NotMessageException je uvedena v diagramu 4.8. Tato třída dědí z std::exception a přepisuje metodu what.



Obrázek 4.8 Třída NotMessageException

4.3.3 Rozdělení tříd zpráv na knihovny

Rozdělení tříd zpráv na knihovny se řídí rozdělením zpráv popsaným v příloze Zprávy. Rozdělení se provádí na základě společných a odlišných částí zprávy pro různé verze. Podle tohoto principu se všechny třídy z hierarchie tříd rozdělí na šest knihoven: knihovnu zpráv společných pro verze 2.3.0, 3.4.0 a 3.6.0, knihovnu zpráv společných pro verze 2.3.0 a 3.4.0 vyjma verze 3.6.0, knihovnu zpráv společných pro verze 3.4.0, 3.6.0 vyjma verze 2.3.0, knihovnu zpráv verze 2.3.0, knihovnu zpráv verze 3.4.0 a knihovnu zpráv verze 3.6.0. Poslední 3 knihovny používají výše uvedené knihovny společných zpráv.

Hierarchie tříd zpráv verze 2.3.0 je uvedena v elektronické příloze, ve složce diagrams, v diagramu „Hierarchie tříd zpráv verze 2.3.0“. Zjednodušené znázornění hierarchie (ve třídách zpráv jsou uvedeny pouze členské proměnné) je uvedeno v diagramu E.1.

Hierarchie tříd zpráv verze 3.4.0 je uvedena v elektronické příloze, ve složce diagrams, v diagramu „Hierarchie tříd zpráv verze 3.4.0“. Zjednodušené znázornění hierarchie (ve třídách zpráv jsou uvedeny pouze členské proměnné) je uvedeno v diagramu E.2.

Hierarchie tříd zpráv verze 3.6.0 je uvedena v elektronické příloze, ve složce diagrams, v diagramu „Hierarchie tříd zpráv verze 3.6.0“. Zjednodušené znázornění hierarchie (ve třídách zpráv jsou uvedeny pouze členské proměnné) je uvedeno v diagramu E.3.

Všechny diagramy hierarchie zpráv ukazují, které třídy zpráv jsou, ve které knihovně obsaženy. Například, z diagramu E.1 lze vidět, že knihovna zpráv společných pro verze 2.3.0, 3.4.0 vyjma verze 3.6.0 má v sobě třídu Message159 (třídu zprávy s ID 159) a používá knihovnu zpráv společných pro verze 2.3.0, 3.4.0, 3.6.0 pro dědění Message159 z třídy MessageEVC. Není-li v diagramu třída zprávy znázorněna v žádné uvedené knihovně znamená to, že se tato třída vyskytuje pouze v knihovně zpráv konkrétní verze. Příslušná verze je uvedena v názvu diagramu. Knihovny zpráv konkrétní verze používají pro přístup ke zprávám společným pro několika verzí knihovny zpráv společných pro několika verzí.

Třídy Message, MessageRBC, MessageEVC a NotMessageException jsou obsaženy v knihovně zpráv společných pro verze 2.3.0, 3.4.0 a 3.6.0.

4.3.4 Třída CommonOptionalPackets a třídy z ní dědící

V knihovně zpráv společných pro verze 2.3.0, 3.4.0 a 3.6.0 byla navrhnuta třída CommonOptionalPackets, která zahrnuje pro tyto verze stejné obecné volitelné packety pro zprávy z RBC do EVC. Třída CommonOptionalPackets má vektory packetů se stejným ID. Třída CommonOptionalPackets má tři konstruktory – jeden je s JSON objektem, druhý je s vektory packetů a třetí je defaultním konstruktorem. Defaultní konstruktor se hodí v případech, kdy zpráva z RBC do EVC, která může mít obecné volitelné packety, žádné nemá. Třída CommonOptionalPackets má operátor <<, virtuální metodu Print, virtuální metodu ToJson a gettery pro získaní hodnot členských proměnných. Pomocí těchto getterů nelze měnit obsah členských proměnných. Třída CommonOptionalPackets má také privátní metodu Check, která se volá v konstruktorech a kontroluje, zda vektory packetů se stejným ID, mimo vektor packetů s ID 66, obsahují packety s odlišnou Q_DIR. V případě vektoru packetů s ID 66 kontroluje, zda packety se stejným Q_DIR obsahují odlišnou hodnotu proměnné NID_TSR.

V knihovně zpráv verze 2.3.0 z třídy CommonOptionalPackets dědí třída CommonOptionalPackets230, která obsahuje vektory dalších obecných volitelných packetů se stejným ID. Struktura těchto packetů, mimo packet s ID 65, je odlišná mezi verzí 2.3.0 a verzemi 3.4.0 a 3.6.0. Struktura volitelného packetu s ID 65 je ve všech verzích stejná, ale jeho kontrola se liší mezi různými knihovnami zpráv. Třída CommonOptionalPackets230 obsahuje privátní metodu Check, která se volá v konstruktorech. Tato metoda kontroluje, zda vektory packetů se stejným ID, mimo vektor packetů s ID 65, obsahují packety s odlišnou Q_DIR. V případě vektoru packetů s ID 65 metoda Check kontroluje, zda v packetech se stejnou Q_DIR je odlišná hodnota proměnné NID_TSR. Z třídy CommonOptionalPackets230 dědí třídy obsahující vektory neobecných volitelných packetů

se stejným ID. Všechny tyto třídy mají privátní metodu Check, která se volá v konstruktorech. Metoda Check kontroluje, zda vektory packetů se stejným ID mají packety s odlišnou Q_DIR.

Stejným způsobem jako třída CommonOptionalPackets230 v knihovně zpráv verze 2.3.0 je uvedena v knihovně zpráv společných pro verze 3.4.0 a 3.6.0 třída CommonOptionalPackets340And360. V metodě Check třídy CommonOptionalPackets340And360 při kontrole vektoru packetů s ID 65 se ověřuje, zda v packetech se stejnou Q_DIR je odlišná hodnota proměnné NID_TSR pouze v případech, kdy NID_TSR nemá hodnotu 255 označující neodvolatelné omezení rychlosti.

4.4 Návrh komunikačního modulu

Pro zpracovaní zpráv komunikační modul používá knihovny zpráv stejné verze, jako je verze modulu EVC. Například v modulu EVC verze 2.3.0 se používá knihovna zpráv verze 2.3.0.

4.4.1 Funkce přijetí a odeslání zprávy

Hlavními funkcemi komunikačního modulu pro komunikaci mezi EVC a RBC na straně EVC jsou přijetí zprávy z RBC do EVC a odeslání zprávy z EVC do RBC.

Ve všech třech verzích (2.3.0, 3.4.0, 3.6.0) modulu EVC, před přijetím, zpráva z RBC projde přes různé kontroly. Po úspěšném průchodu přes kontroly se obsah zprávy uloží a případně se odešlou zprávy z EVC do RBC. Rozdíl mezi verzemi je pouze v části kontroly zprávy před jejím přijetí. Na začátku přijetí zprávy se vždy kontroluje, zda EVC je v jednom z uvedených módů. Mezi verzí 2.3.0 a verzemi 3.4.0 a 3.6.0 se tyto módy liší. Další rozdíl je v tom, že ve verzích 2.3.0 a 3.4.0 existuje zpráva z RBC s ID 38, ve verzi 3.6.0 pak nikoliv, a proto nejsou ani kontroly s ní spojené. Jednotlivé kroky funkce přijetí zprávy jsou uvedeny v diagramech aktivit. Funkce přijetí zprávy pro verzi 2.3.0 je uvedena v diagramu aktivit 4.9. Funkce přijetí zprávy pro verzi 3.4.0 je uvedena v diagramu aktivit 4.10. Funkce přijetí zprávy pro verzi 3.6.0 je uvedena v diagramu aktivit 4.11.

Ve všech třech verzích modulu EVC je funkce odeslání zprávy z EVC a RBC stejná. Jednotlivé kroky funkce odeslání zprávy z EVC do RBC jsou uvedeny v diagramu aktivit 4.12.

4.4.2 Třída CommunicationModuleRBC

Ve všech třech verzích modulu EVC je komunikační modul pro komunikaci mezi EVC a RBC na straně EVC reprezentován privátní třídou CommunicationModuleRBC vnořenou do třídy Client.

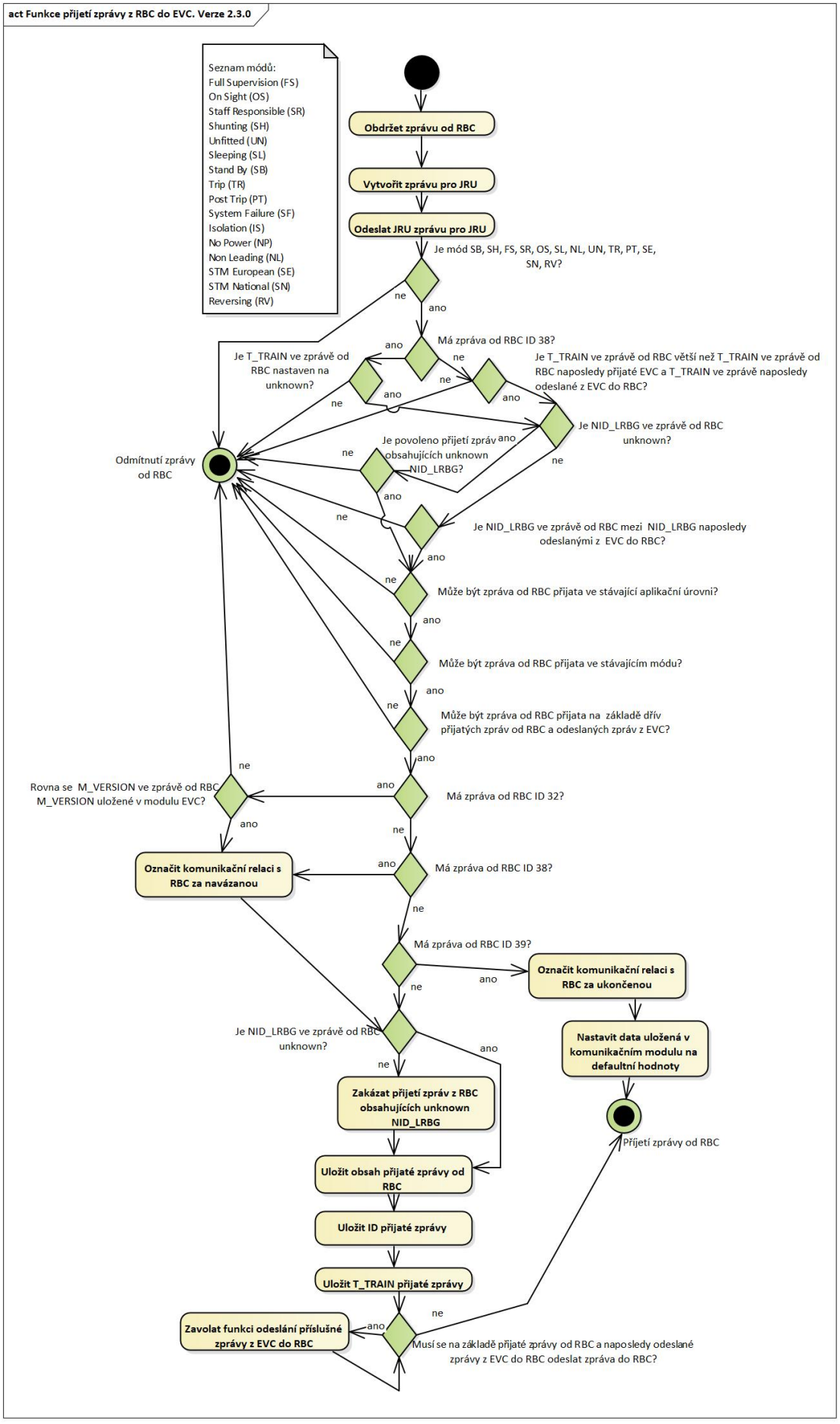
Členskými proměnnými třídy Client jsou objekt třídy CommunicationModuleRBC a objekt třídy DataEVC, která má všechna potřebná data pro modul EVC.

Třída CommunicationModuleRBC a třída Client v modulu EVC verze 2.3.0 jsou uvedeny v diagramu 4.13. Třída CommunicationModuleRBC a třída Client v modulu EVC verze 3.4.0 jsou uvedeny v diagramu 4.14. Třída CommunicationModuleRBC a třída Client v modulu EVC verze 3.6.0 jsou uvedeny v diagramu 4.15. Jediný rozdíl v těchto diagramech je v metodách odeslání zpráv.

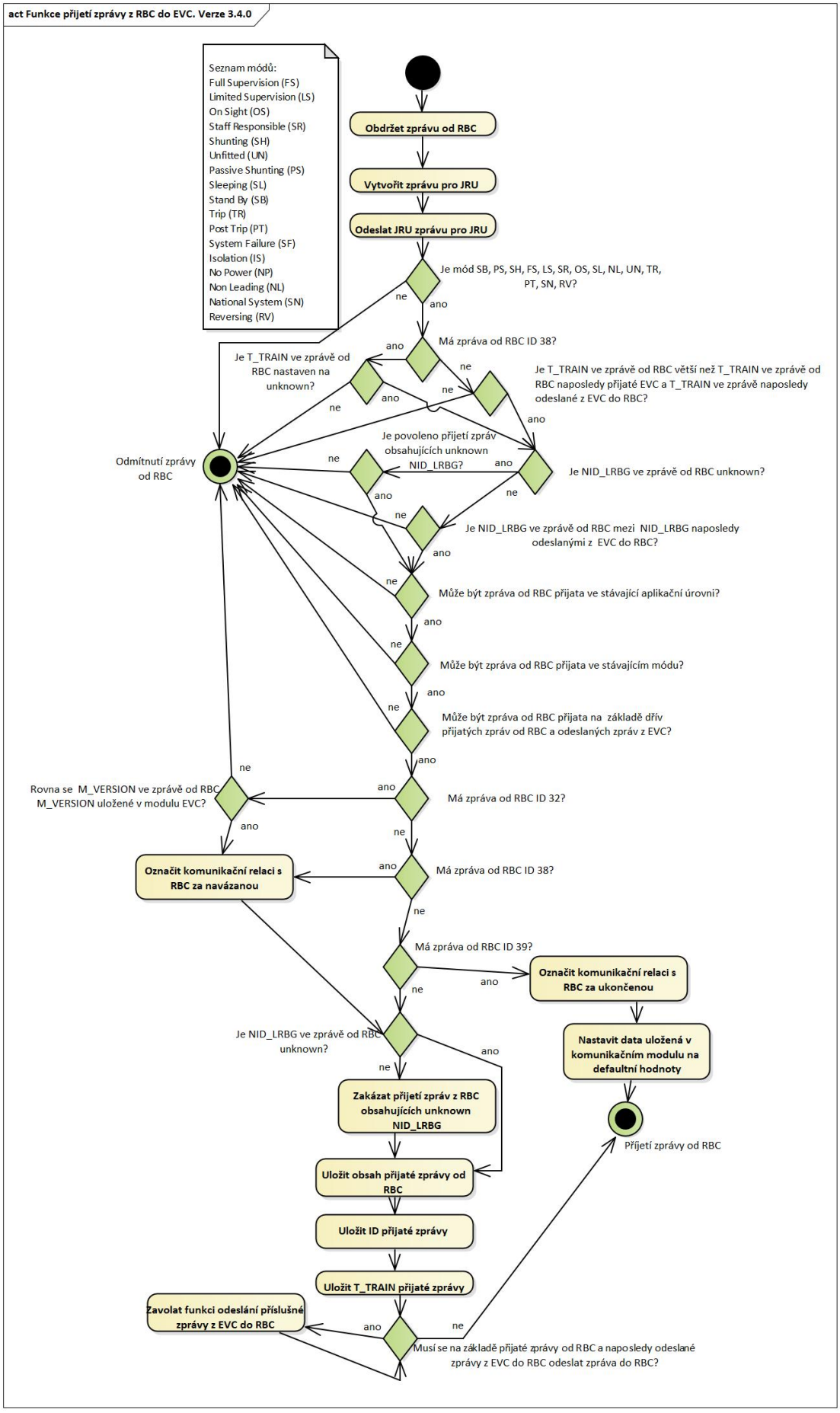
Z důvodu, že komunikační modul musí znát ID naposledy přijaté zprávy a ID naposledy odeslané zprávy, třída CommunicationModuleRBC má mezi členskými proměnnými dva objekty třídy NidMessage, které reprezentují ID naposledy odeslané zprávy a ID naposledy přijaté zprávy. Z důvodu, že komunikační modul musí znát několik NID_LRBG naposledy odeslaných ve zprávě z EVC do RBC, třída CommunicationModuleRBC má mezi členskými proměnnými std::deque objektů třídy NidLrbg reprezentující proměnnou NID_LRBG a konstantní proměnnou mající bezznaménkový datový typ a uvádějící maximální počet NID_LRBG, které mohou být obsaženy v std::deque. Tento počet NID_LRBG musí být podle kapitoly 3 subsetu 026 vždy vetší nebo roven osmi. Kvůli tomu, že komunikační modul musí znát hodnotu proměnné T_TRAIN naposledy

přijaté zprávy, a hodnotu T_TRAIN naposledy odeslané zprávy, třída CommunicationModuleRBC má dva objekty třídy TTrain reprezentující proměnnou T_TRAIN, které reprezentují T_TRAIN naposledy odeslané zprávy a T_TRAIN naposledy přijaté zprávy. Třída CommunicationModuleRBC má také dvě členské proměnné mající datový typ bool. První proměnná určuje, zda je komunikační relace navázána mezi moduly RBC a EVC. Druhá proměnná udává, zda může být přijata zpráva obsahující NID_LRBG nastavenou na hodnotu „unknown“. Třída CommunicationModuleRBC má také referenci na třídu Client. Tato reference je potřeba, neboť v metodách komunikačního modulu je nutný přístup k objektu třídy DataEVC a k metodě SendJRU, které jsou ve třídě Client.

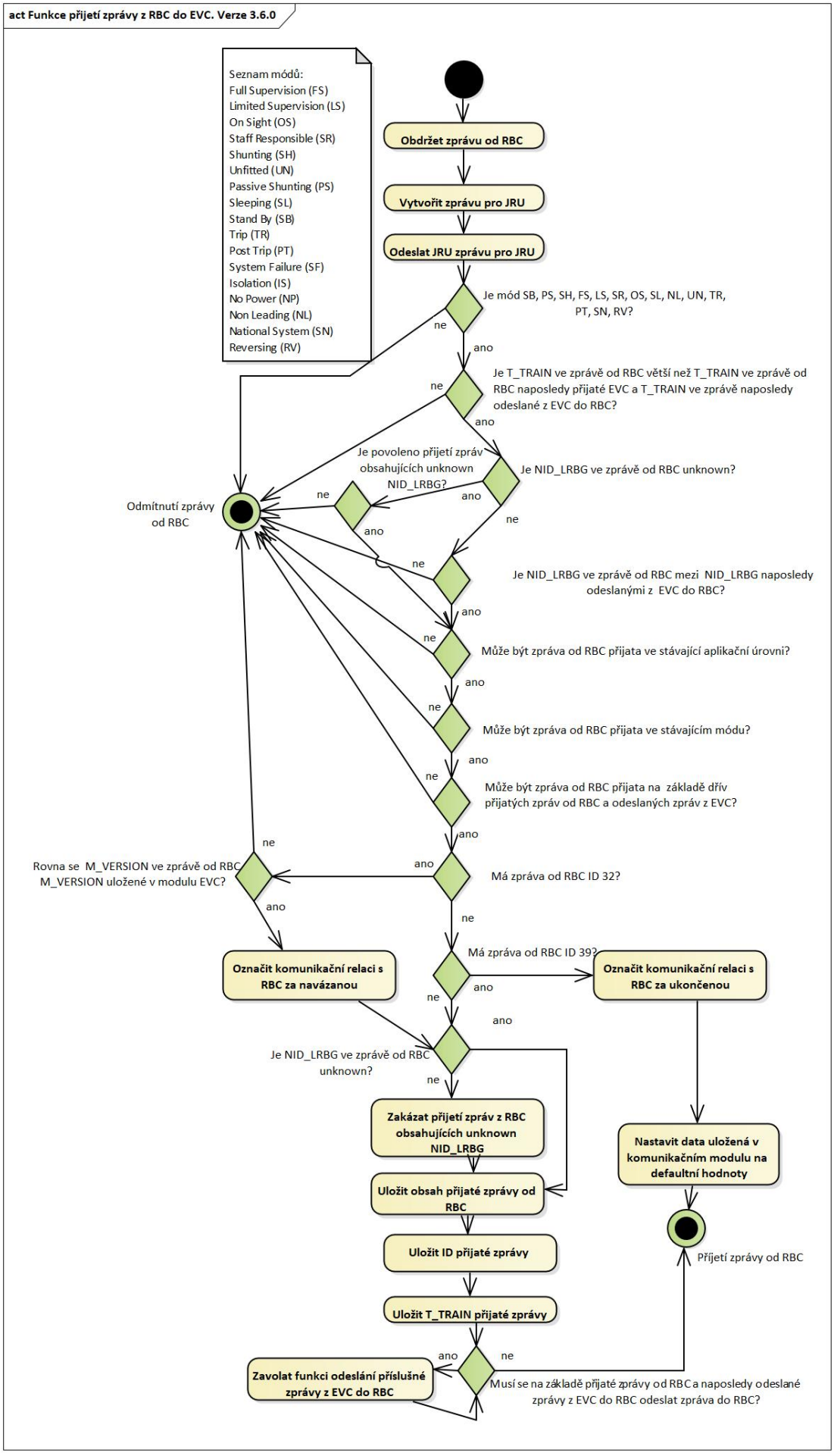
Třída CommunicationModuleRBC má konstruktor se dvěma parametry. První z nich uvádí maximální počet NID_LRBG naposledy odeslaných ve zprávách z EVC do RBC, který musí znát komunikační modul na straně EVC. Druhý parametr je referencí na třídu Client. Komunikační modul má privátní metodu Reset, která nastaví všechny členské proměnné této třídy, mimo proměnnou uváděcí maximální počet pamatovaných NID_LRBG a referenci na třídu Client, na defaultní hodnoty. Třída CommunicationModuleRBC má privátní metodu FindNidLrbg mající na vstupu NID_LRBG. Pokud tato NID_LRBG je „unknown“ a stav přijetí „unknown“ NID_LRBG je nastaven na true, metoda FindNidLrbg vrátí true. Pokud NID_LRBG na vstupu není „unknown“ a je obsažena v std::deque NID_LRBG, metoda FindNidLrbg vrátí true. CommunicationModuleRBC má privátní metodu AddNidLrbg, která má na vstupu NID_LRBG. Pokud zavolaná metoda FindNidLrbg s touto NID_LRBG vrátí false a tato NID_LRBG je „unknown“, stav přijetí „unknown“ NID_LRBG se nastaví na true a metoda AddNidLrbg se ukončí. Pokud zavolaná metoda FindNidLrbg s touto NID_LRBG vrátí false a tato NID_LRBG není „unknown“ a std::deque NID_LRBG má v sobě méně prvků než je maximální počet, tato NID_LRBG se přidá na konec std::deque. Pokud zavolaná metoda FindNidLrbg s touto NID_LRBG vrátí false a tato NID_LRBG není „unknown“ a std::deque NID_LRBG má počet prvků rovný maximálnímu počtu prvků, odebere se první prvek std::deque pomocí pop_front a tato NID_LRBG se přidá na konec std::deque. Třída CommunicationModuleRBC má privátní metodu ValidTTrain, která má na vstupu proměnnou T_TRAIN. Tato metoda vrátí true, pokud tato T_TRAIN je vetší než T_TRAIN naposledy přijatá od RBC a T_TRAIN naposledy odeslaná z EVC. Třída CommunicationModuleRBC má také metodu ValidMVersion, která má na vstupu proměnnou M_VERSION a která vrátí true, pokud se tato M_VERSION rovná M_VERSION obsažené v objektu třídy DataEVC uloženým ve třídě Client. Třída CommunicationModuleRBC přijímá zprávu ve formátu JSON pomocí metody ReceiveMessage. Tato metoda se volá v třídě Client v metodě ReceiveRBC, která se volá ve funkci přijetí dat z různých modulů. Metoda ReceiveRBC ve svém těle na začátku zavolá metodu ReceiveMessage a pokud tato metoda vrátí true, v závislosti na ID a obsahu zprávy, se mohou provést příslušné výpočty. Metoda ReceiveMessage funguje na základě diagramu aktivit pro přijetí zprávy. V metodě ReceiveMessage se používají metody ValidTTrain, FindNidLrbg, ValidMVersion a Reset. Metoda ReceiveMessage sama neukládá obsah zprávy do dat modulu EVC, pro toto uložení se v této metodě volají nad objektem třídy DataEVC příslušné metody. Ve třídě CommunicationModuleRBC se zprávy odesílají pomocí všech metod začínajících na „SendMessage“. Některé tyto metody mají parametry datového typu bool. Tyto parametry uvádějí, zda bude mít zpráva z EVC do RBC volitelné packety nebo ne. Pokud zpráva může mít pouze jeden volitelný packet, metoda pro odeslání zprávy z EVC do RBC má pouze jeden parametr. Pokud zprava může mít dva volitelné packety, má metoda pro odeslání zprávy dva parametry. Tyto metody se volají v metodách třídy Client a v metodě ReceiveMessage, kdy musí dojít k odeslání určité zprávy. Tyto metody fungují na základě diagramu aktivit pro odeslání zprávy. V těle každé z těchto metod se nejdřív vytvoří zpráva a po tom se zavolá privátní metoda SendRBC, která odešle tuto zprávu modulu RBC a poté odešle informace o odeslané zprávě modulu JRU.



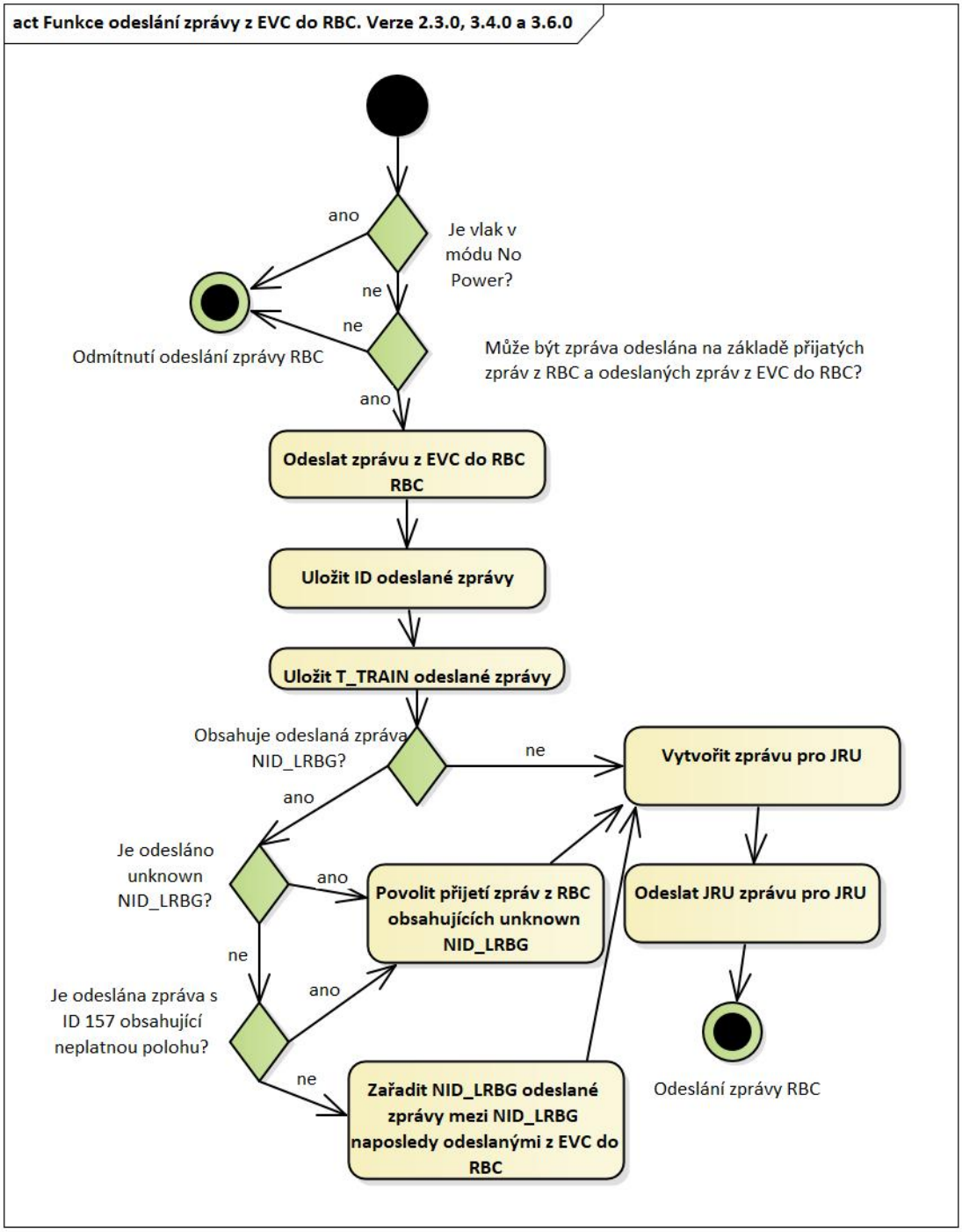
Obrázek 4.9 Funkce přijetí zprávy z RBC do EVC. Verze 2.3.0



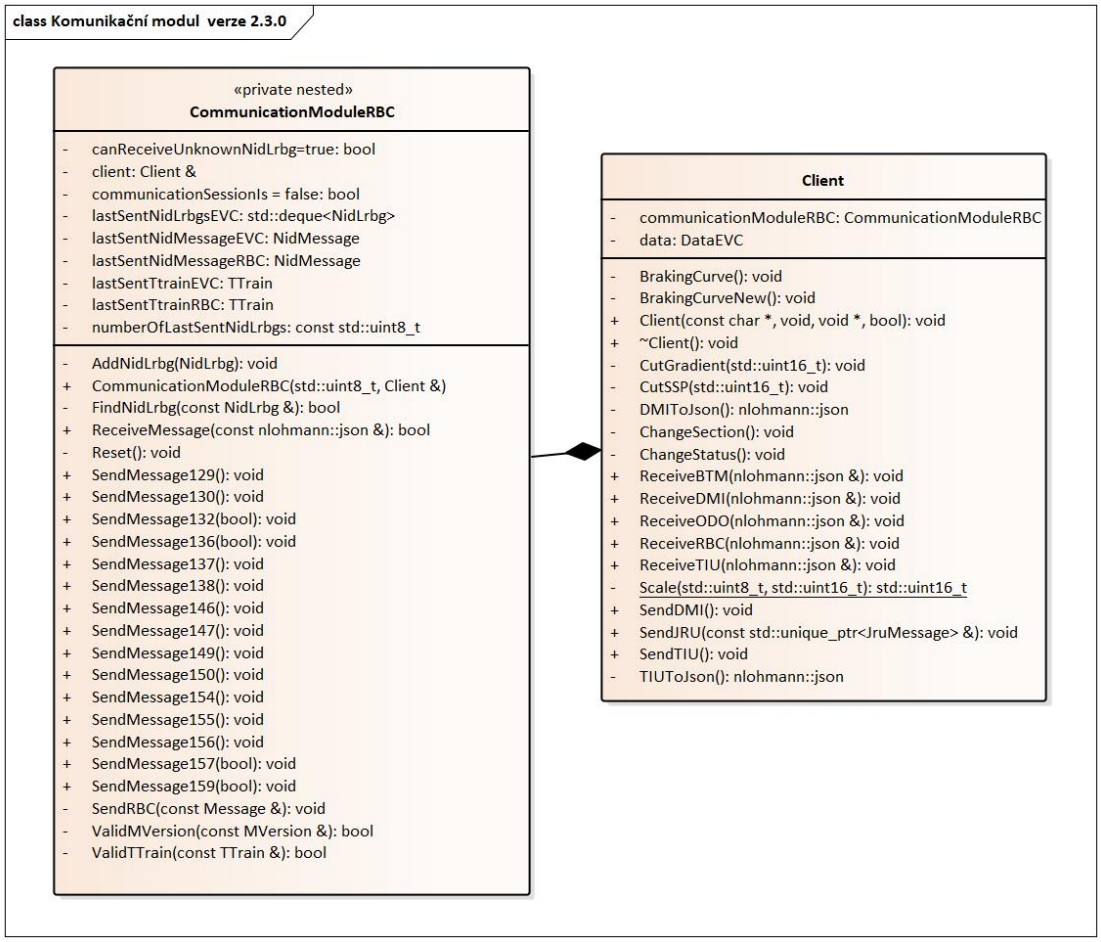
Obrázek 4.10 Funkce přijetí zprávy z RBC do EVC. Verze 3.4.0



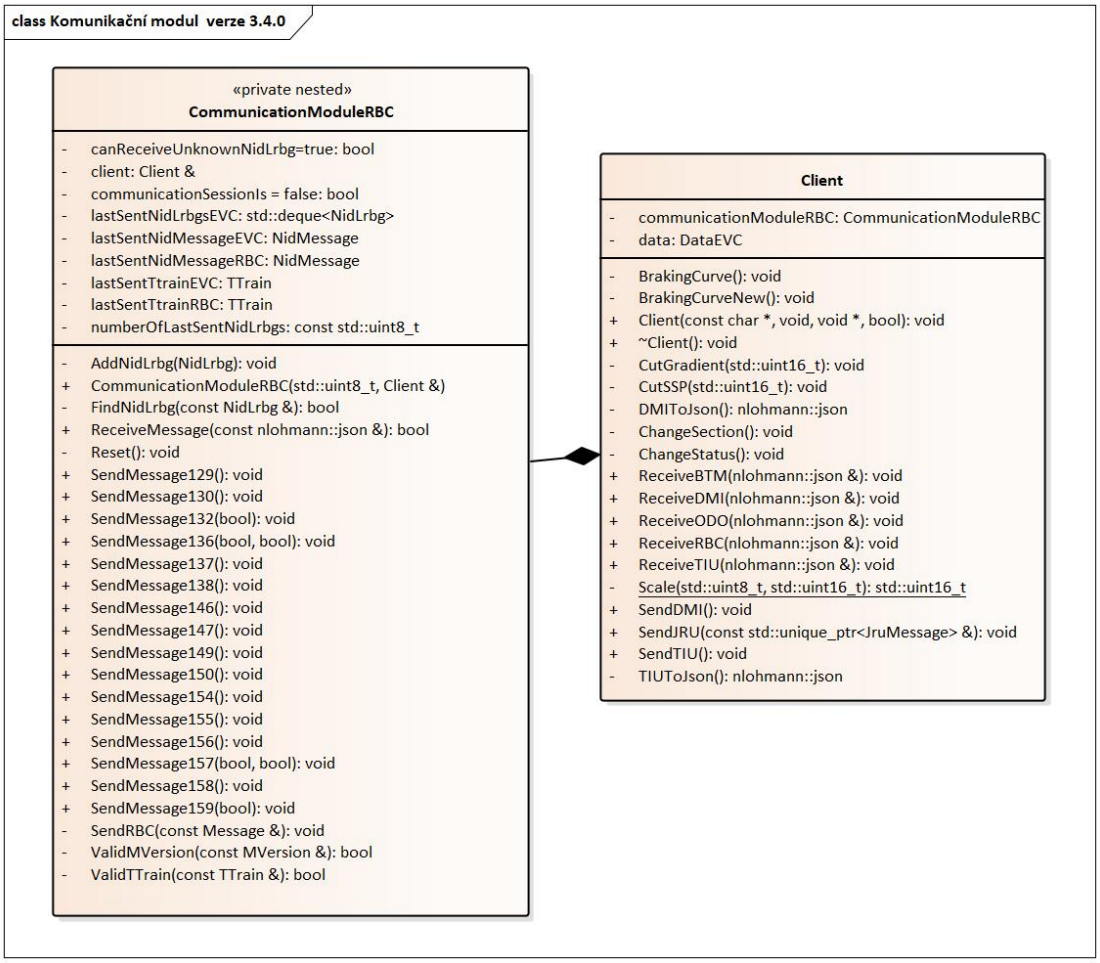
Obrázek 4.11 Funkce přijetí zprávy z RBC do EVC. Verze 3.6.0



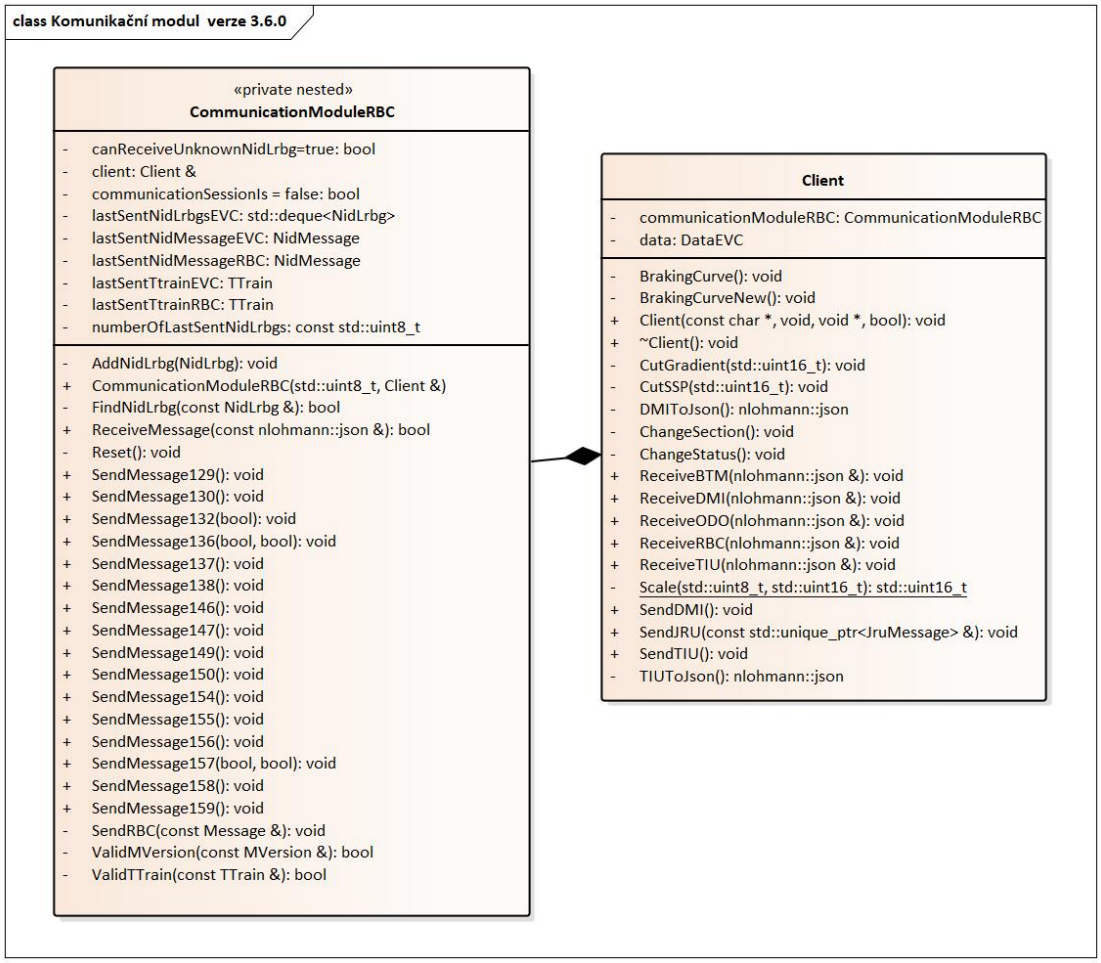
Obrázek 4.12 Funkce odeslání zprávy z EVC do RBC



Obrázek 4.13 Třída reprezentující komunikační modul pro komunikaci mezi EVC a RBC na straně EVC. Verze 2.3.0



Obrázek 4.14 Třída reprezentující komunikační modul pro komunikaci mezi EVC a RBC na straně EVC. Verze 3.4.0



Obrázek 4.15 Třída reprezentující komunikační modul pro komunikaci mezi EVC a RBC na straně EVC. Verze 3.6.0

Kapitola 5
Implementace

V této kapitole je popsána implementace tříd proměnných, tříd packetů a implementace knihoven proměnných a knihoven packetů obsahujících tyto třídy.

5.1 Implementace tříd proměnných

Na základě návrhu uvedeného v předchozí kapitole byly implementovány všechny třídy proměnných, potřebných pro komunikaci mezi moduly EVC a RBC v simulátoru ETCS, a to pro všechny tři verze.

Deklarace třídy Variable je uvedena ve výpisu kódu 5.1.

Implementace metody Check třídy Variable je uvedena ve výpisu kódu 5.2. V této metodě se pro kontrolu platnosti hodnoty používá bitový posun doprava.

Deklarace třídy proměnné, která nepotřebuje jiné kontroly platnosti hodnoty, než které jsou ve třídě Variable, je uvedena ve výpisu kódu 5.3. Deklarace třídy proměnné, která potřebuje dodatečné kontroly, je uvedena ve výpisu kódu 5.4. Z deklarace této třídy lze vidět, že tato třída potřebuje navíc ke dvěma konstruktorům metodu Check pro další ověření ukládané hodnoty a zároveň potřebuje přepsat metody SetValue.

5.2 Implementace tříd packetů

Na základě návrhu byly implementovány třídy packetů z RBC do EVC s ID 15, 131, 138, 139, 39, 42, 45, 57, 58, 65, 66, 71, 40, 64, a to pro všechny tři verze.

Deklarace třídy Packet, nejvyšší třídy v hierarchii tříd packetů, je uvedena ve výpisu kódu 5.5. Deklarace třídy PacketQDir a deklarace třídy PacketQDirQScale jsou uvedeny ve výpisech kódu 5.6 a 5.7. Ve všech třídách, mimo třídu PacketQDir, v přepisovaných metodách Print a ToJson se na začátku vždy volají metody Print a ToJson nadtřídy. Ve třídě PacketQDir v přepisované metodě Print se nevolá metoda Print nadtřídy, a to z důvodu, že proměnná Q_DIR, podle struktury definované v kapitole 7 subsetu 026, je obsažena mezi proměnnými NID_PACKET a L_PACKET. Metoda Print třídy PacketQDir je uvedena ve výpisu kódu 5.8.

Ve třídě reprezentující konkrétní packet v těle konstruktoru s JSON objektem se na začátku kontroluje, zda JSON objekt obsahuje stejný počet datových položek, jako je členských proměnných příslušné třídy. Příklad takové kontroly je ve výpisu kódu 5.9.

Příklad kontroly rovnosti hodnoty uložené do členské proměnné lPacket a skutečné délky packetu je uveden ve výpisu kódu 5.10. Tato kontrola probíhá v privátní metodě Check ve třídě TrackPacket15. V této metodě se nekontroluje ID uložené v nidPacket, rovnost hodnoty nIter a skutečné velikosti jednotlivých vektorů a to, zda se copy_ptr na třídy s rozhodovací proměnnými

```
template <class T, std::uint8_t P>
class Variable {
public:
    Variable(const std::string & name, T value);
    virtual ~Variable() = default;
    Variable(const std::string & name, const nlohmann::json & j,
             const std::string & order = "");
    void ToJson(nlohmann::json & j, const std::string & order = "") const;
    [[nodiscard]] nlohmann::json ToJson() const;
    friend std::ostream & operator << ( std::ostream & os, const Variable<T,P> & variable) {
        os << variable.m_Name << ": " << std::to_string(variable.m_Value) << "\n";
        return os;
    }
    friend bool operator==(const Variable<T,P> & lhs, const Variable<T, P> & rhs) {
        if (lhs.m_Name == rhs.m_Name && lhs.m_Value == rhs.m_Value)
            return true;
        return false;
    }
    friend bool operator!=(const Variable<T,P> & lhs, const Variable<T, P> & rhs) {
        return !(lhs==rhs);
    }
    virtual void SetValue(T value);
    virtual void SetValue(const nlohmann::json & j, const std::string & order);
    T GetValue() const;
    [[nodiscard]] const std::string & GetName() const;
protected:
    std::string m_Name;
    T m_Value;
private:
    void SetJson(const nlohmann::json & j, const std::string & order);
    void Check() const;
};
```

Výpis kódu 5.1 Deklarace třídy Variable

```
template <class T, std::uint8_t P>
void Variable<T, P>::Check() const {
    if (std::numeric_limits<T>::digits <= P)
        return;
    if (P == 0 || !((m_Value >= 0 && m_Value >> P == 0) ||
       (m_Value < 0  && m_Value >> P == -1)))
        throw NotVariableException();
}
```

Výpis kódu 5.2 Implementace metody Check třídy Variable

nerovnají nullptr, neboť tyto kontroly již proběhly předtím v nadtřídě TrackPacket15Base v rámci privátní metody Check, která byla zavolána z konstruktoru nadtřídy. Metoda Check třídy TrackPacket15Base je uvedena ve výpisu kódu 5.11.

```
class DLink : public Variable<std::uint16_t, 15> {
public:
    explicit DLink(std::uint16_t value = 0);
    explicit DLink(const nlohmann::json & j, const std::string & order = "");
};
```

Výpis kódu 5.3 Deklarace třídy proměnné neobsahující dodatečné kontroly

```
class NidRadio : public Variable<std::uint64_t, 64> {
public:
    explicit NidRadio(std::uint64_t value = 0);
    explicit NidRadio(const nlohmann::json & j, const std::string & order = "");
    void SetValue(std::uint64_t value) override;
    void SetValue(const nlohmann::json & j, const std::string & order) override;
private:
    /**
     * Check, if set beforehand m_Value is valid according to
     * its definition in subset 7 version 2.3.0 or 3.4.0 or 3.6.0 (definition is same there).
     * @throws NotVariableException
     */
    void Check() const;
};
```

Výpis kódu 5.4 Deklarace třídy proměnné obsahující dodatečné kontroly

```
class Packet {
public:
    Packet(const NidPacket & nidPacket, const LPacket & lPacket);
    explicit Packet(const nlohmann::json & j);
    virtual ~Packet() = default;
    [[nodiscard]] virtual nlohmann::json ToJson() const;
    friend std::ostream & operator << ( std::ostream & os, const Packet & packet);
    [[nodiscard]] const NidPacket &  GetNidPacket() const;
    [[nodiscard]] const LPacket & GetLPacket() const;
protected:
    virtual void Print (std::ostream & os) const;
    virtual void PrintPacketName(std::ostream & os) const = 0;
    NidPacket nidPacket;
    LPacket lPacket;
};
```

Výpis kódu 5.5 Deklarace třídy Packet

5.3 Implementace knihoven proměnných a packetů

Na základě návrhu v předchozí kapitole bylo vytvořeno 6 knihoven proměnných a 6 knihoven packetů. Tyto knihovny obsahují výše uvedené třídy. Knihovny proměnných konkrétní verze mají mimo implementovaných tříd proměnných také hlavičkové soubory, mající názvy tříd proměn-

```
class PacketQDir : public Packet {
public:
    PacketQDir(const NidPacket & nidPacket, const QDir & qDir, const LPacket & lPacket);
    explicit PacketQDir(const nlohmann::json & j);
    [[nodiscard]] nlohmann::json ToJson() const override;
    [[nodiscard]] const QDir & GetQDir() const;
protected:
    void Print (std::ostream & os) const override;
    QDir qDir;
};
```

Výpis kódu 5.6 Deklarace třídy PacketQDir

```
class PacketQDirQScale : public PacketQDir {
public:
    PacketQDirQScale(const NidPacket & nidPacket, const QDir & qDir,
                     const LPacket & lPacket, const QScale & qScale);
    explicit PacketQDirQScale(const nlohmann::json & j);
    [[nodiscard]] nlohmann::json ToJson() const override;
    [[nodiscard]] const QScale & GetQScale() const;
protected:
    void Print (std::ostream & os) const override;
    QScale qScale;
};
```

Výpis kódu 5.7 Deklarace třídy PacketQDirQScale

```
void PacketQDir::Print (std::ostream & os) const {
    PrintPacketName(os);
    os << nidPacket;
    os << qDir;
    os << lPacket;
}
```

Výpis kódu 5.8 Implementace metody Print třídy PacketQDir

```
TrackPacket45::TrackPacket45(const nlohmann::json & j) : PacketQDir(j), nidMn(j) {
    if (j.size() != 4)
        throw NotPacketException();
    Check();
}
```

Výpis kódu 5.9 Implementace konstruktoru s JSON objektem třídy TracKPacket45

ných obsažených v knihovnách proměnných společných pro několik verzí. Do těchto hlavičkových souborů jsou vloženy hlavičkové soubory obsahující deklarace příslušné třídy z těchto knihoven. V knihovně proměnných konkrétní verze je také hlavičkový soubor AllVariables.hpp, do kte-

```
void TrackPacket15::Check() const {
    std::uint16_t length = 62 + nIter.GetValue() * 15 + qSectiontimerIf -> CountLength() +
                           qEndtimerIf -> CountLength() + qDangerpointIf -> CountLength() +
                           qOverlapIf -> CountLength();
    for (const auto & timer : qSectiontimerIfs)
        length += timer -> CountLength();
    if (lPacket.GetValue() != length)
        throw NotPacketException();
}
```

Výpis kódu 5.10 Implementace metody Check třídy TrackPacket15

```
void TrackPacket15Base::Check() const {
    if (!(nidPacket.GetValue() == 15 && lSections.size() == nIter.GetValue() &&
          qSectiontimerIfs.size() == nIter.GetValue() && qSectiontimerIf != nullptr &&
          qEndtimerIf != nullptr && qDangerpointIf != nullptr && qOverlapIf != nullptr))
        throw NotPacketException();
    for (const auto & timer : qSectiontimerIfs) {
        if (timer == nullptr)
            throw NotPacketException();
    }
}
```

Výpis kódu 5.11 Implementace metody Check třídy TrackPacket15Base

rého jsou vložené zbývající hlavičkové soubory této knihovny. Stejná struktura je i v knihovnách packetů konkrétní verze.

Každá knihovna má adresář include a adresář src. V adresáři include jsou veřejné hlavičkové soubory a v adresáři src jsou interní zdroje knihovny. V každé knihovně jsou CMakeLists.txt a Config.cmake.in. Oba tyto soubory byly vytvořeny na základě CMakeLists.txt a Config.cmake.in uvedených v článku Creating a C++ library with CMake [19]. V každé knihovně v CMakeLists.txt je definováno, že se vytvoří dynamická knihovna. Tato definice je ve výpisu kódu 5.12. Pro sestavení knihovny byla použita sekvence příkazů uvedená ve výpisu kódu 5.13. Během sestavení knihovny se vytvoří adresář install, ve kterém je nainstalovaná knihovna. Pro použití knihovny v jakémkoli projektu stačí mít u sebe tento adresář a uvést v CMakeLists.txt projektu cestu k tomuto adresáři. CMakeLists.txt projektu používajícího knihovnu je uveden ve výpisu kódu 5.14.

```
add_library(${PROJECT_NAME} SHARED)
```

Výpis kódu 5.12 Výňatek z CMakeLists.txt knihovny

```
mkdir build && cd $_
cmake -DCMAKE_BUILD_TYPE=Debug ..
cmake --build . --target install
rm -r ./* && cmake -DCMAKE_BUILD_TYPE=Release ..
cmake --build . --target install
```

Výpis kódu 5.13 Sekvence příkazů pro sestavení knihovny

```
cmake_minimum_required(VERSION 3.12)
project(example)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED True)
set(CMAKE_PREFIX_PATH "../packets_230_340_360/install")
find_package(packets_230_340_360 CONFIG REQUIRED)
add_executable(${PROJECT_NAME} example.cpp)
target_link_libraries(${PROJECT_NAME} PRIVATE packets_230_340_360::packets_230_340_360)
```

Výpis kódu 5.14 CMakeLists.txt projektu používajícího knihovnu packetů packets_230_340_360

Kapitola 6
Testování

V této kapitole je popsáno testování tříd proměnných a tříd packetů, které byly uvedeny v předchozí kapitole. Následně je popsán obsah adresáře tests v adresáři impl v elektronické příloze.

Testování tříd proměnných a tříd packetů je určeno ke zjištění, zda naimplementované třídy fungují tak, jak mají. Testování probíhá pomocí assertů s funkcemi testujícími tyto třídy a vracejícími hodnotu datového typu bool.

6.1 Testování tříd proměnných

Bylo otestováno pouze několik tříd proměnných. Jelikož všechny třídy, mimo tříd majících dodatečné kontroly do nich ukládaných hodnot, mají stejnou strukturu (tj. dva konstruktory s prázdnými těly), byla otestována jenom jedna taková třída. Třídy obsahující dodatečné kontroly byly naopak otestovány všechny.

Bylo otestováno vytvoření proměnné pomocí konstruktoru s číselným parametrem a vytvoření proměnné pomocí konstruktoru s JSON objektem. Příklady takových testů jsou ve výpisech kódů 6.1 a 6.2. Byly otestovány metody SetValue s číselným parametrem a SetValue s JSON objektem. Příklady těchto testů jsou ve výpisech kódů 6.3 a 6.4. Byla také otestována metoda ToJson. Příklad testu je uveden ve výpisu kódu 6.5. Bylo otestováno, že se do proměnné neuloží hodnota vyšší než maximální hodnota, kterou tato proměnná může nabývat. Příklad testu je uveden ve výpisu kódu 6.6. Příklad testu, že se do proměnné neuloží menší než minimální hodnota je pak ve výpisu kódu 6.7.

```
bool createVariable() {
    try {
        LPacket lPacket(42);
        if (lPacket.GetName() == "L_PACKET" && lPacket.GetValue() == 42)
            return true;
    }
    catch(const NotVariableException & ex) {
        return false;
    }
    return false;
}
```

Výpis kódu 6.1 Testování vytvoření proměnné s číselnou hodnotou

```
bool createVariableJSON() {
    nlohmann::json j = {
            {"L_PACKET", 10}
    };
    try {
        LPacket lPacket(j);
        if (lPacket.ToJson() == j)
            return true;
    }
    catch(const NotVariableException & ex) {
        return false;
    }
    return false;
}
```

Výpis kódu 6.2 Testování vytvoření proměnné s JSON objektem

```
bool setVariable() {
    try {
        LPacket lPacket(11);
        lPacket.SetValue(12);
        if (lPacket.GetValue() == 12)
            return true;
    }
    catch(const NotVariableException & ex) {
        return false;
    }
    return false;
}
```

Výpis kódu 6.3 Testování metody SetValue s číselnou hodnotou

U tříd majících dodatečné kontroly ukládané hodnoty bylo otestováno, že se neuloží hodnota mimo validní rozpětí. Příklad testu je ve výpisu kódu 6.8.

6.2 Testování tříd packetů

Byly otestovány všechny implementované třídy packetů.

Bylo otestováno vytvoření packetu pomocí konstruktoru s proměnnými. Příklad tohoto testu je uveden ve výpisu kódu F.1, kde se testuje vytvoření packetu s ID 58. Bylo také otestováno vytvoření packetu pomocí konstruktoru s JSON objektem. Příklad takového testu je ve výpisu kódu F.2. Bylo otestováno, že se packet nevytvoří, pokud se hodnota proměnné L_PACKET nerovná skutečné délce packetu, a vyhodí se výjimka. Příklad tohoto testu je uveden ve výpisu kódu F.3. Bylo také otestováno, že JSON objekt předaný do konstruktoru třídy packetu nesmí mít víc datových položek, než je členských proměnných příslušné třídy. Příklad tohoto testu je ve výpisu kódu F.4.

Pro třídy packetů obsahující vektory proměnných, nebo vektory copy_ptr, bylo otestováno, že se packet nevytvoří, pokud velikost alespoň jednoho vektoru není rovná hodnotě N_ITER, po které tento vektor následuje. Příklad odpovídajícího testu je ve výpisu kódu F.5.

```
bool setVariableJSON() {
    nlohmann::json j;
    j["L_PACKET"] = 100;
    try {
        LPacket lPacket(10);
        lPacket.SetValue(j, "");
        if (lPacket.ToJson() == j)
            return true;
    }
    catch(const NotVariableException & ex) {
        return false;
    }
    return false;
}
```

Výpis kódu 6.4 Testování metody SetValue s JSON objektem

```
bool toJsonVariable() {
    nlohmann::json j = {
            {"L_PACKET3", 4}
    };
    nlohmann::json result;
    try {
        LPacket lPacket(4);
        lPacket.ToJson(result, "3");
        if (result == j)
            return true;
    }
    catch(const NotVariableException & ex) {
        return false;
    }
    return false;
}
```

Výpis kódu 6.5 Testování metody ToJson

Pro třídy packetů obsahující copy_ptr na bázové třídy s rozhodovacími proměnnými bylo otestováno, že se packet nevytvoří, pokud se alespoň jeden z těchto copy_ptr rovná nullptr. Příklad příslušného testu je uveden ve výpisu kódu F.6.

6.3 Výsledky testování

Všechny testy dopadly úspěšně. Všechny funkce, které testují třídy proměnných a třídy packetů a jsou zavolány v assertech, měly návratovou hodnotu true.

```
bool createVariableValueBiggerThanMax() {
    nlohmann::json j = {
            {"L_PACKET", 8192}
    };
    try {
        LPacket lPacket(j);
    }
    catch(const NotVariableException & ex) {
        return true;
    }
    return false;
}
```

Výpis kódu 6.6 Testování, zda se proměnná může vytvořit s hodnotou vyšší než maximální

```
bool createVariableValueLessThanMin() {
    nlohmann::json j = {
            {"VARIABLE", -16800}
    };
    try {
        Variable<std::int16_t, 14> var("VARIABLE", j);
    }
    catch(const NotVariableException & ex) {
        return true;
    }
    return false;
}
```

Výpis kódu 6.7 Testování, zda se proměnná může vytvořit s hodnotou menší než minimální

```
bool createNidMnOnlyF() {
    try {
        NidMn nidMn(0xffffff);
    }
    catch(const NotVariableException & ex) {
        return true;
    }
    return false;
}
```

Výpis kódu 6.8 Testování jedné z dodatečných kontrol při vytvoření proměnné NID_MN

6.4 Obsah adresáře tests

Testování výše uvedených tříd je v elektronické příloze v adresáři tests v adresáři impl. V tomto adresáři je adresář test_230, který obsahuje testy třídy TrackPacket15 reprezentující packet s ID 15 společný pro verze 2.3.0 a 3.4.0 subsetu 026, testy třídy TrackPacket39 repzrezentující packet s ID 39 verze 2.3.0 tohoto subsetu a třídy NidOperational reprezentující proměnnou

NID_OPERATIONAL verze 2.3.0. V adresáři tests je také adresář tests_360, ve kterém jsou testy tříd packetů společných pro verze 3.4.0 a 3.6.0 subsetu 026, testy třídy NidOperational reprezentující proměnnou NID_OPERATIONAL společnou pro verze 3.6.0 a 3.4.0 tohoto subsetu a test třídy TrackPacket15 reprezentující packet s ID 15 verze 3.6.0 příslušného subsetu. Adresář tests také obsahuje adresář tests_230_340_360, kde jsou uvedeny testy několika tříd proměnných a všech tříd packetů společných pro verze 2.3.0, 3.4.0 a 3.6.0 subsetu 026. Adresáři tests_230, tests_360 a tests_230_340_360 mají soubor tests.cpp, ve kterém jsou funkce testující třídy proměnných a packetů a funkce main obsahující asserty s těmito funkcemi. Každý adresář má také CMakeLists.txt a spustitelný soubor.

Kapitola 7
Závěr

Všechny cíle, stanovené na začátku práce, byly splněny.

V teoretické části byla provedena analýza dosavadní komunikace mezi moduly EVC a RBC v simulátoru ETCS. Byly analyzovány a popsány rozdíly a podobnosti proměnných, packetů a zpráv jazyka ETCS, nutných pro komunikaci mezi moduly EVC a RBC v simulátoru ETCS a komunikace s RBC na straně EVC mezi verzemi 2.3.0, 3.4.0 a 3.6.0 systémových požadavků kladených na ETCS. Na základě analýzy dosavadní komunikace mezi moduly EVC a RBC v simulátoru ETCS a analýzy rozdílů a podobností byly definovány funkční a nefunkční požadavky proměnných, packetů, zpráv nutných pro komunikaci mezi moduly EVC a RBC v simulátoru ETCS a komunikačního modulu pro komunikaci mezi moduly EVC a RBC na straně EVC v simulátoru ETCS.

V praktické části byly na základě funkčních a nefunkčních požadavků navrhnuty proměnné, packety, zprávy a komunikační modul pro komunikaci mezi EVC a RBC na straně modulu EVC pro verze 2.3.0, 3.4.0 a 3.6.0 systémových požadavků kladených na ETCS. Na základě tohoto návrhu byly naimplementovány všechny proměnné a vybrané packety, a to pro všechny výše uvedené verze. Pro každou z těchto verzí byla vytvořena knihovna proměnných a knihovna packetů. Byly naimplementovány unit testy pro proměnné a packety. Všechny testy proběhly úspěšně.

Kapitola 8
Literatura

[1] Factsheet # 28 ERTMS ADVANTAGES. In: *ERTMS* [online]. [cit. 2022-03-20]. Dostupné z: https://www.ertms.net/wp-content/uploads/2021/06/28.-ERTMS-Advantages-factsheet_final.pdf

[2] ERTMS in brief. *ERTMS* [online]. [cit. 2022-03-22]. Dostupné z: https://www.ertms.net/about-ertms/ertms-in-brief/

[3] Bezpečnostní komise MD: Další zabezpečování tratí v ČR běží, vláda bude schvalovat plán zabezpečení. *Ministerstvo dopravy ČR* [online]. 2021 [cit. 2022-03-22]. Dostupné z: https://www.mdcr.cz/Media/Media-a-tiskove-zpravy/Bezpecnostni-komise-MD-Dalsi-zabezpecovani-trati

[4] CHVOJKA, Vojtěch. *ETCS: European Train Control System* [online]. Praha, 2020 [cit. 2022-03-23]. Dostupné z: https://cena-dekana.fd.cvut.cz/prezentace/12_rocnik/prace/chvojka.pdf

[5] MACHÁČEK, Miloslav. Instalace ETCS v českých podmínkách. In: BENEŠ, Petr a kol. *Vědeckotechnický sborník ČD č. 33/2012* [online]. Praha: Generální ředitelství Českých drah, 2012, s. 4 [cit. 2022-03-23]. ISSN 1214-9047. Dostupné z: https://vts.cd.cz/documents/168518/195465/3306.pdf/0a40f14a-5c01-4323-82ec-a7478df6d505

[6] PETŘÍK, Lukáš. *Functional specification for a Driver's Cab Simulator with ETCS* [online]. Praha, 2016 [cit. 2022-03-23]. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/68265/F6-DP-2016-Petrik-Lukas-Diplomka.pdf?sequence=1&isAllowed=y. Magisterská práce. České vysoké učení technické, Fakulta dopravní, Ústav dopravní telematiky. Vedoucí práce Martin Leso.

[7] European Rail Traffic Management System (ERTMS). *Railway Technologies* [online]. 2021 [cit. 2022-03-23]. Dostupné z: https://www.railway-technology.com/projects/european-rail-traffic-management-system-ertms/

[8] ERTMS Signaling levels. *ERTMS* [online]. [cit. 2022-03-24]. Dostupné z: https://www.ertms.net/about-ertms/ertms-signaling-levels/

[9] *Národní implementační plán ERTMS* [online]. Praha: Ministerstvo dopravy ČR, 2017 [cit. 2022-03-24]. Dostupné z: https://www.mdcr.cz/getattachment/Dokumenty/Drazni-doprava/Evropska-unie-na-zeleznici/Evropska-unie-na-zeleznici/NIP-ERTMS-2017.pdf.aspx?lang=cs-CZ

[10] Vláda schválila plán zavádění moderního evropského zabezpečení železničních tratí. *Ministerstvo dopravy ČR* [online]. 2021 [cit. 2022-03-24]. Dostupné z: https://mdcr.cz/Media/Media-a-tiskove-zpravy/Vlada-schvalila-plan-zavadeni-moderniho-evropskeho

[11] Časté dotazy. *Správa železnic* [online]. [cit. 2022-03-24]. Dostupné z: https://www.spravazeleznic.cz/digitalizace/etcs/caste-dotazy

[12] LESO, Martin. *Koncepce vlakového simulátoru CVUT: komponenta ETCS*. 2021.

[13] SUBSET-026, *System Requirements Specification Chapter 7 ERTMS/ETCS language*. Version 2.3.0. European Union Agency For Railways, 2006.

[14] SUBSET-026, *System Requirements Specification Chapter 8 Messages*. Version 2.3.0. European Union Agency For Railways, 2006.

[15] ČERVENKA, Jan. *Implementation of the DMI Display for the ETCS On-Board Sub-System*. Praha, 2017 [cit. 2022-03-24]. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/69651/F6-DP-2017-Cervenka-Jan-final.pdf?sequence=1&isAllowed=y. Magisterská práce. České vysoké učení technické, Fakulta dopravní, Ústav dopravní telematiky.

[16] Eurobalise on the track. In: *ResearchGate* [online]. [cit. 2022-03-25]. Dostupné z: https://www.researchgate.net/figure/Eurobalise-on-the-track_fig2_319054004

[17] DMI. In: *Railway Gazette* [online]. [cit. 2022-03-25]. Dostupné z: https://www.railwaygazette.com/europe/revised-ertms-deployment-plan-adopted/43768.article

[18] SUBSET-026, *System Requirements Specification Chapter 3 Principles*. Version 2.3.0. European Union Agency For Railways, 2006.

[19] Creating a C++ library with CMake. *Declaration of VAR* [online]. [cit. 2022-05-08]. Dostupné z: https://decovar.dev/blog/2021/03/08/cmake-cpp-library/

Příloha A

Proměnné

V této příloze jsou uvedeny všechny proměnné z verzí 2.3.0, 3.4.0, 3.6.0 subsetu 026 System Requirements Specification, které jsou potřebné pro komunikaci mezi EVC a RBC v simulátoru ETCS.

Proměnné, které jsou současně obsaženy ve verzích 2.3.0, 3.4.0, 3.6.0 a které mají v těchto verzích stejnou délku v bitech a rozsahy platných hodnot, jsou uvedeny v tabulce 1 společně s jejich délkou v bitech.

Proměnné, které jsou současně obsaženy ve verzích 3.4.0 a 3.6.0 a které mají v těchto verzích stejnou délku v bitech a rozsahy platných hodnot, ale nejsou obsaženy ve verzi 2.3.0 nebo jsou v ní obsaženy, ale mají odlišnou délku v bitech nebo rozsah platných hodnot, jsou uvedeny v tabulce 2 společně s jejich délkou v bitech.

Proměnné, které jsou současně obsaženy ve verzích 2.3.0 a 3.4.0, ale nejsou obsaženy ve verzi 3.6.0 a které mají v těchto verzích stejnou délku v bitech a rozsahy platných hodnot, jsou uvedeny v tabulce 3 společně s jejich délkou v bitech.

Proměnné, které jsou obsaženy pouze ve verzi 2.3.0 nebo mají v jiných verzích odlišnou délku v bitech nebo rozsahy platných hodnot, jsou uvedeny v tabulce 4 společně s jejich délkou v bitech.

Proměnné, které jsou obsaženy pouze ve verzi 3.6.0, jsou uvedeny v tabulce 5 společně s jejich délkou v bitech.

Tabulka 1

Proměnná	Délka v bitech
D_ADHESION	15
D_AXLELOAD	15
D_CYCLOC	15
D_DP	15
D_EMERGENCYSTOP	15
D_ENDTIMERSTARTLOC	15
D_GRADIENT	15
D_LEVELTR	15
D_LINK	15
D_LOC	15
D_LRBG	15
D_MAMODE	15
D_NVOVTRP	15
D_NVPOTRP	15
D_NVROLL	15
D_NVSTFF	15
D_OL	15
D_POSOFF	15
D_RBCTR	15
D_REF	16
D_REVERSE	15
D_SECTIONTIMERSTOPLOC	15
D_SR	15
D_STARTOL	15
D_STARTREVERSE	15
D_STATIC	15
D_SUITABILITY	15
D_TAFDISPLAY	15
D_TEXTDISPLAY	15
D_TRACKCOND	15
D_TRACKINIT	15
D_TRACTION	15
D_TSR	15
D_VALIDNV	15
G_A	8
L_ACKLEVELTR	15
L_ACKMAMODE	15

L_ADHESION	15
L_AXLELOAD	15
L_DOUBTOVER	15
L_DOUBTUNDER	15
L_ENDSECTION	15
L_MAMODE	15
L_MESSAGE	10
L_PACKET	13
L_REVERSEAREA	15
L_SECTION	15
L_TAFDISPLAY	15
L_TEXT	8
L_TEXTDISPLAY	15
L_TRACKCOND	15
L_TRAIN	12
L_TRAININT	15
L_TSR	15
M_ACK	1
M_ADHESION	1
M_AIRTIGHT	2
M_ERROR	8
M_LEVEL	3
M_LEVELTEXTDISPLAY	3
M_LEVELTR	3
M_LOADINGGAUGE	8
M_LOC	3
M_MAMODE	2
M_MODE	4
M_MODETEXTDISPLAY	4
M_NVCONTACT	2
M_NVDERUN	1
M_TRACKCOND	4
N_ITER	5
NC_DIFF	4
NC_TRAIN	15
NID_BG	14
NID_C	10
NID_EM	4
NID_ENGINE	24

NID_LRBG	24
NID_LTRBG	24
NID_MESSAGE	8
NID_MN	24
NID_PACKET	8
NID_RADIO	64
NID_RBC	14
NID_TSR	8
Q_DANGERPOINT	1
Q_DIR	2
Q_DIRLRBG	2
Q_DIRTRAIN	2
Q_DLRBG	2
Q_EMERGENCYSTOP	2
Q_ENDTIMER	1
Q_FRONT	1
Q_GDIR	1
Q_LENGTH	2
Q_LGTLOC	1
Q_LINKORIENTATION	1
Q_LINKREACTION	2
Q_LOCACC	6
Q_MPOSITION	1
Q_NEWCOUNTRY	1
Q_NVDRIVER_ADHES	1
Q_NVEMRRLS	1
Q_ORIENTATION	1
Q_OVERLAP	1
Q_RBC	1
Q_SCALE	2
Q_SECTIONTIMER	1
Q_SLEEPSESSION	1
Q_STATUS	2
Q_SUITABILITY	2
Q_TEXT	8
Q_TEXTCLASS	2
Q_TEXTCONFIRM	2
Q_TEXTDISPLAY	1
Q_TRACKINIT	1

T_CYCLOC	8
T_CYCRQST	8
T_ENDTIMER	10
T_MAR	8
T_NVCONTACT	8
T_NVOVTRP	8
T_OL	10
T_SECTIONTIMER	10
T_TEXTDISPLAY	10
T_TIMEOUTRQST	10
T_TRAIN	32
V_AXLELOAD	7
V_DIFF	7
V_MAMODE	7
V_MAXTRAIN	7
V_NVALLOWOVTRP	7
V_NVONSIGHT	7
V_NVREL	7
V_NVSHUNT	7
V_NVSTFF	7
V_NVSUPOVTRP	7
V_NVUNFIT	7
V_RELEASEDP	7
V_RELEASEOL	7
V_REVERSE	7
V_STATIC	7
V_TRAIN	7
V_TSR	7
X_TEXT	8

Tabulka 2

Proměnná	Délka v bitech
A_NVMAXREDADH1	6
A_NVMAXREDADH2	6
A_NVMAXREDADH3	6
A_NVP12	6
A_NVP23	6
D_CURRENT	15

D_LX	15
D_PBD	15
D_PBDSR	15
G_PBDSR	8
L_LX	15
L_NVKRINT	5
L_PBDSR	15
L_STOPLX	15
M_AXLELOADCAT	7
M_CURRENT	10
M_LINEGAUGE	8
M_NVAVADH	5
M_NVEBCL	4
M_NVKRINT	5
M_NVKTINT	5
M_NVKVINT	7
M_PLATFORM	4
M_POSITION	24
M_VERSION	7
M_VOLTAGE	4
N_AXLE	10
NC_CDDIFF	4
NC_CDTRAIN	4
NID_CTRACTION	10
NID_LX	8
NID_NTC	8
NID_OPERATIONAL	32
NID_PRVLRBG	24
NID_TEXTMESSAGE	8
Q_CONFTEXTDISPLAY	1
Q_DIFF	2
Q_LXSTATUS	1
Q_MAMODE	1
Q_MARQSTREASON	5
Q_NVGUIPERM	1
Q_NVINHSMICPERM	1
Q_NVKINT	1
Q_NVKVINTSET	2
Q_NVLOCACC	6

Q_NVSBFBPERM	1
Q_NVSBTSMPERM	1
Q_PBDSR	1
Q_PLATFORM	2
Q_STOPLX	1
Q_TEXTREPORT	1
V_LX	7
V_NVKVINT	7
V_NVLIMSUPERV	7

Tabulka 3

Proměnná	Délka v bitech
T_LOA	10
V_LOA	7

Tabulka 4

Proměnná	Délka v bitech
M_AXLELOAD	7
M_POSITION	20
M_TRACTION	8
M_VERSION	7
NID_OPERATIONAL	32
NID_PRVBG	24
NID_STM	8
Q_NVSRBKTRG	1
Q_TRACKDEL	1

Tabulka 5

Proměnná	Délka v bitech
T_EMA	10
V_EMA	7

Příloha B

Packety

V této příloze jsou uvedeny všechny packety z verzí 2.3.0, 3.4.0, 3.6.0 subsetu 026 System Requirements Specification, které jsou potřebné pro komunikaci mezi EVC a RBC v simulátoru ETCS.

- Modrou barvou jsou označeny části packetu, které jsou pouze ve verzi 2.3.0.

- Zelenou barvou jsou označeny části packetu, které jsou pouze ve verzích 3.4.0 a 3.6.0.

- Růžovou barvou jsou označeny části packetu, které jsou pouze ve verzích 2.3.0 a 3.4.0.

- Šedou barvou jsou označeny části packetu, které jsou pouze ve verzi 3.6.0.

Vedle ID packetu je napsán název tohoto packetu. Vnitřní struktura packetu je převzata z kapitoly 7 subsetu 026. Vedle každé proměnné je napsána její délka v bitech a případně podmínka, která musí být splněna, aby tato proměnná mohla být obsažena v packetu. Proměnné, které nejsou označeny žádnou barvou, mají stejnou délku v bitech a rozsahy platných hodnot ve všech odpovídajících verzích kapitoly 7 subsetu 026.

Stejná ID packetů pro verze 2.3.0, 3.4.0 a 3.6.0:
Z RBC do EVC:
Packet 5: Linking
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_LINK 15
Q_NEWCOUNTRY 1
NID_C 10 if Q_NEWCOUNTRY = 1
NID_BG 14
Q_LINKORIENTATION 1
Q_LINKREACTION 2
Q_LOCACC 6
N_ITER 5
D_LINK (k) 15
Q_NEWCOUNTRY(k) 1
NID_C (k) 10 if Q_NEWCOUNTRY(k) = 1
NID_BG (k) 14
Q_LINKORIENTATION (k) 1
Q_LINKREACTION (k) 2
Q_LOCACC (k) 6

Packet 39: Track Condition Change of traction power
Track Condition Change of traction system
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_TRACTION 15
M_TRACTION 8
M_VOLTAGE 4
NID_CTRACTION 10 NID_CTRACTION given only if M_VOLTAGE ≠ 0

Packet 51: Axle Load Speed Profile
NID_PACKET 8
Q_DIR 2

L_PACKET 13
Q_SCALE 2
Q_TRACKINIT 1

- D_TRACKINIT 15 Only if Q_TRACKINIT = 1
- D_AXLELOAD 15 Only if Q_TRACKINIT = 0, D_AXLELOAD and the following variables follow
- L_AXLELOAD 15
- Q_FRONT 1
- N_ITER 5
- M_AXLELOAD(n) 7
- M_AXLELOADCAT(n) 7
- V_AXLELOAD(n) 7
- N_ITER 5
- D_AXLELOAD(k) 15
- L_AXLELOAD(k) 15
- Q_FRONT(k) 1
- N_ITER(k) 5
- M_AXLELOAD(k,m) 7
- M_AXLELOADCAT(k,m) 7
- V_AXLELOAD(k,m) 7

Packet 41: Level Transition Order
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_LEVELTR 15
M_LEVELTR 3

- NID_STM 8 If M_LEVELTR = 1 (STM)
- NID_NTC 8 If M_LEVELTR = 1 (NTC)

L_ACKLEVELTR 15
N_ITER 5
M_LEVELTR(k) 3

- NID_STM(k) 8 If M_LEVELTR = 1 (STM)
- NID_NTC(k) 8 If M_LEVELTR(k) = 1 (NTC)

L_ACKLEVELTR(k) 15

Packet 42: Session Management
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_RBC 1
NID_C 10
NID_RBC 14
NID_RADIO 64
Q_SLEEPSESSION 1

Packet 45: Radio Network registration
NID_PACKET 8
Q_DIR 2
L_PACKET 13
NID_MN 24

Packet 57: Movement Authority Request Parameters
NID_PACKET 8
Q_DIR 2
L_PACKET 13
T_MAR 8
T_TIMEOUTRQST 10
T_CYCRQST 8

Packet 58: Position Report Parameters
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
T_CYCLOC 8
D_CYCLOC 15
M_LOC 3
N_ITER 5
D_LOC(k) 15
Q_LGTLOC(k) 1

Packet 65: Temporary Speed Restriction
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
NID_TSR 8
D_TSR 15
L_TSR 15
Q_FRONT 1
V_TSR 7

Packet 66: Temporary Speed Restriction Revocation
NID_PACKET 8
Q_DIR 2
L_PACKET 13
NID_TSR 8

Packet 68: Track Condition
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
Q_TRACKINIT 1

D_TRACKINIT 15 Only if Q_TRACKINIT = 1
D_TRACKCOND 15 Only if Q_TRACKINIT = 0, D_TRACKCOND and the following variables follow
L_TRACKCOND 15
M_TRACKCOND 4
N_ITER 5
D_TRACKCOND(k) 15
L_TRACKCOND(k) 15
M_TRACKCOND(k) 4

Packet 70: Route Suitability Data
NID_PACKET 8
Q_DIR 2
L_PACKET 13

Q_SCALE 2
Q_TRACKINIT 1
D_TRACKINIT 15 Only if Q_TRACKINIT = 1
D_SUITABILITY 15 Only If Q_TRACKINIT = 0, D_SUITABILITY and the following variables follows
Q_SUITABILITY 2
M_LINEGAUGE 8 If Q_SUITABILITY= loading gauge
M_LOADINGGAUGE 8 If Q_SUITABILITY= loading gauge
M_AXLELOAD 7 If Q_SUITABILITY= Max axle load.
M_AXLELOADCAT 7 If Q_SUITABILITY= Max axle load.
M_TRACTION 8 If Q_SUITABILITY = traction power
M_VOLTAGE 4 If Q_SUITABILITY = traction system
NID_CTRACTION 10 If Q_SUITABILITY = traction system and M_VOLTAGE ≠0
N_ITER 5
D_SUITABILITY(k) 15
Q_SUITABILITY(k) 2
M_LOADINGGAUGE(k) 8 If Q_SUITABILITY= loading gauge
M_AXLELOAD(k) 7 If Q_SUITABILITY= Max axle load.
M_TRACTION(k) 8 If Q_SUITABILITY = traction power
M_LINEGAUGE(k) 8 If Q_SUITABILITY(k) = loading gauge
M_AXLELOADCAT(k) 7 If Q_SUITABILITY(k) = Max axle load.
M_VOLTAGE(k) 4 If Q_SUITABILITY(k) = traction system
NID_CTRACTION(k) 10 If Q_SUITABILITY(k) = traction system and M_VOLTAGE(k) ≠0

Packet 71: Adhesion factor
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_ADHESION 15
L_ADHESION 15
M_ADHESION 1

Packet 72: Packet for sending plain text messages
NID_PACKET 8

Q_DIR 2
L_PACKET 13
Q_SCALE 2
Q_TEXTCLASS 2
Q_TEXTDISPLAY 1
D_TEXTDISPLAY 15
M_MODETEXTDISPLAY 4
M_LEVELTEXTDISPLAY 3
    NID_STM 8 If M_LEVELTEXTDISPLAY = 1 (STM)
    NID_NTC 8 If M_LEVELTEXTDISPLAY = 1 (NTC)
L_TEXTDISPLAY 15
T_TEXTDISPLAY 10
M_MODETEXTDISPLAY 4
M_LEVELTEXTDISPLAY 3
    NID_STM 8 If M_LEVELTEXTDISPLAY = 1 (STM)
    NID_NTC 8 If M_LEVELTEXTDISPLAY = 1 (NTC)
Q_TEXTCONFIRM 2
    Q_CONFTEXTDISPLAY 1 If Q_TEXTCONFIRM ≠ 0
    Q_TEXTREPORT 1 If Q_TEXTCONFIRM ≠ 0
        NID_TEXTMESSAGE 8 Only If Q_TEXTREPORT = 1
        NID_C 10 Only If Q_TEXTREPORT = 1
        NID_RBC 14 Only If Q_TEXTREPORT = 1
L_TEXT 8
X_TEXT 8

Packet 76: Packet for sending fixed text messages
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
Q_TEXTCLASS 2
Q_TEXTDISPLAY 1
D_TEXTDISPLAY 15
M_MODETEXTDISPLAY 4
M_LEVELTEXTDISPLAY 3
    NID_STM 8 If M_LEVELTEXTDISPLAY = 1 (STM)
    NID_NTC 8 If M_LEVELTEXTDISPLAY = 1 (NTC)
L_TEXTDISPLAY 15

T_TEXTDISPLAY 10
M_MODETEXTDISPLAY 4
M_LEVELTEXTDISPLAY 3
NID_STM 8 If M_LEVELTEXTDISPLAY = 1 (STM)
NID_NTC 8 If M_LEVELTEXTDISPLAY = 1 (NTC)
Q_TEXTCONFIRM 2
Q_CONFTEXTDISPLAY 1 If Q_TEXTCONFIRM ≠ 0
Q_TEXTREPORT 1 If Q_TEXTCONFIRM ≠ 0
NID_TEXTMESSAGE 8 Only If Q_TEXTREPORT = 1
NID_C 10 Only If Q_TEXTREPORT = 1
NID_RBC 14 Only If Q_TEXTREPORT = 1
Q_TEXT 8

Packet 79: Geographical Position Information
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
Q_NEWCOUNTRY 1
NID_C 10 if Q_NEWCOUNTRY = 1
NID_BG 14
D_POSOFF 15
Q_MPOSITION 1
M_POSITION 20
M_POSITION 24
N_ITER 5
Q_NEWCOUNTRY(k) 1
NID_C(k) 10 if Q_NEWCOUNTRY(k) = 1
NID_BG(k) 14
D_POSOFF(k) 15
Q_MPOSITION(k) 1
M_POSITION(k) 20
M_POSITION(k) 24

Packet 131: RBC transition order
NID_PACKET 8
Q_DIR 2
L_PACKET 13

Q_SCALE 2
D_RBCTR 15
NID_C 10
NID_RBC 14
NID_RADIO 64
Q_SLEEPSESSION 1

Packet 138: Reversing area information
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_STARTREVERSE 15
L_REVERSEAREA 15

Packet 139: Reversing supervision information
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_REVERSE 15
V_REVERSE 7

Packet 140: Train running number from RBC
NID_PACKET 8
Q_DIR 2
L_PACKET 13
NID_OPERATIONAL 32
NID_OPERATIONAL 32
Poznámka: Ve verzích 2.3.0 a 3.4.0, 3.6.0 proměnná NID_OPERATIONAL má odlišný rozsah platných hodnot.

Packet 21: Gradient Profile
NID_PACKET 8
Q_DIR 2
L_PACKET 13

Q_SCALE 2
D_GRADIENT 15
Q_GDIR 1
G_A 8
N_ITER 5
D_GRADIENT(k) 15
Q_GDIR(k) 1
G_A(k) 8

Packet 27: International Static Speed Profile
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_STATIC 15
V_STATIC 7
Q_FRONT 1
N_ITER 5
NC_DIFF(n) 4
Q_DIFF(n) 2
    NC_CDDIFF(n) 4 If Q_DIFF(n) = 0
    NC_DIFF(n) 4 If Q_DIFF(n) = 1 or 2
V_DIFF(n) 7
N_ITER 5
D_STATIC(k) 15
V_STATIC(k) 7
Q_FRONT(k) 1
N_ITER(k) 5
NC_DIFF(k,m) 4
Q_DIFF(k,m) 2
    NC_CDDIFF(k,m) 4 If Q_DIFF(k,m) = 0
    NC_DIFF(k,m) 4 If Q_DIFF(k,m) = 1 or 2
V_DIFF(k,m) 7

Packet 49: List of balises for SH Area
NID_PACKET 8
Q_DIR 2

L_PACKET 13
N_ITER 5
Q_NEWCOUNTRY(k) 1
NID_C(k) 10 if Q_NEWCOUNTRY(k) = 1
NID_BG(k) 14

Packet 80: Mode profile
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_MAMODE 15
M_MAMODE 2
V_MAMODE 7
L_MAMODE 15
L_ACKMAMODE 15
Q_MAMODE 1
N_ITER 5
D_MAMODE(k) 15
M_MAMODE(k) 2
V_MAMODE(k) 7
L_MAMODE(k) 15
L_ACKMAMODE(k) 15
Q_MAMODE(k) 1

Packet 63: List of Balises in SR Authority
NID_PACKET 8
Q_DIR 2
L_PACKET 13
N_ITER 5
Q_NEWCOUNTRY(k) 1
NID_C(k) 10 if Q_NEWCOUNTRY(k) = 1
NID_BG(k) 14

Packet 15: Level 2/3 Movement Authority
NID_PACKET 8
Q_DIR 2

L_PACKET 13
Q_SCALE 2
V_LOA 7
T_LOA 10
V_EMA 7
T_EMA 10
N_ITER 5
L_SECTION(k) 15
Q_SECTIONTIMER(k) 1
    T_SECTIONTIMER(k) 10
    D_SECTIONTIMERSTOPLOC(k) 15
L_ENDSECTION 15
Q_SECTIONTIMER 1
    T_SECTIONTIMER 10
    D_SECTIONTIMERSTOPLOC 15
Q_ENDTIMER 1
    T_ENDTIMER 10
    D_ENDTIMERSTARTLOC 15
Q_DANGERPOINT 1
    D_DP 15
    V_RELEASEDP 7
Q_OVERLAP 1
    D_STARTOL 15
    T_OL 10
    D_OL 15
    V_RELEASEOL 7

Packet 3: National Values
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_VALIDNV 15
NID_C 10
N_ITER 5
NID_C(k) 10
V_NVSHUNT 7
V_NVSTFF 7

V_NVONSIGHT 7
V_NVLIMSUPERV 7
V_NVUNFIT 7
V_NVREL 7
D_NVROLL 15
Q_NVSBTSMPERM 1
Q_NVSRBKTRG 1
Q_NVEMRRLS 1
Q_NVGUIPERM 1
Q_NVSBFBPERM 1
Q_NVINHSMICPERM 1
V_NVALLOWOVTRP 7
V_NVSUPOVTRP 7
D_NVOVTRP 15
T_NVOVTRP 8
D_NVPOTRP 15
M_NVCONTACT 2
T_NVCONTACT 8
M_NVDERUN 1
D_NVSTFF 15
Q_NVDRIVER_ADHES 1
A_NVMAXREDADH1 6
A_NVMAXREDADH2 6
A_NVMAXREDADH3 6
Q_NVLOCACC 6
M_NVAVADH 5
M_NVEBCL 4
Q_NVKINT 1
    Q_NVKVINTSET 2 Only if Q_NVKINT = 1, Q_NVKVINTSET and the following variables follow
        A_NVP12 6 Only if Q_NVKVINTSET = 1
        A_NVP23 6 Only if Q_NVKVINTSET = 1
    V_NVKVINT 7
    M_NVKVINT 7
        M_NVKVINT 7 Only if Q_NVKVINTSET = 1
    N_ITER 5
    V_NVKVINT(n) 7
    M_NVKVINT(n) 7

M_NVKVINT(n) 7 Only if Q_NVKVINTSET = 1
N_ITER 5
Q_NVKVINTSET(k) 2
A_NVP12(k) 6 Only if Q_NVKVINTSET(k) = 1
A_NVP23(k) 6 Only if Q_NVKVINTSET(k) = 1
V_NVKVINT(k) 7
M_NVKVINT(k) 7
M_NVKVINT(k) 7 Only if Q_NVKVINTSET(k) = 1
N_ITER(k) 5
V_NVKVINT(k,m) 7
M_NVKVINT(k,m) 7
M_NVKVINT(k,m) 7 Only if Q_NVKVINTSET(k) = 1
L_NVKRINT 5
M_NVKRINT 5
N_ITER 5
L_NVKRINT(l) 5
M_NVKRINT(l) 5
M_NVKTINT 5

Z EVC do RBC:

Packet 0: Position Report
NID_PACKET 8
L_PACKET 13
Q_SCALE 2
NID_LRBG 24
D_LRBG 15
Q_DIRLRBG 2
Q_DLRBG 2
L_DOUBTOVER 15
L_DOUBTUNDER 15
Q_LENGTH 2
L_TRAININT 15 If Q_LENGTH = “Train integrity confirmed by integrity monitoring device” or “Train integrity confirmed by driver”
V_TRAIN 7
Q_DIRTRAIN 2
M_MODE 4
M_LEVEL 3
NID_STM 8 If M_LEVEL = STM

NID_NTC 8 If M_LEVEL = NTC

Packet 1: Position Report based on two balise groups
NID_PACKET 8
L_PACKET 13
Q_SCALE 2
NID_LRBG 24
NID_PRVBG 24
NID_PRVLRBG 24
D_LRBG 15
Q_DIRLRBG 2
Q_DLRBG 2
L_DOUBTOVER 15
L_DOUBTUNDER 15
Q_LENGTH 2
L_TRAININT 15 If Q_LENGTH = “Train integrity confirmed by integrity monitoring device” or “Train integrity confirmed by driver”
V_TRAIN 7
Q_DIRTRAIN 2
M_MODE 4
M_LEVEL 3
NID_STM 8 If M_LEVEL = STM
NID_NTC 8 If M_LEVEL = NTC

Packet 11: Validated train data
NID_PACKET 8
L_PACKET 13
NC_CDTRAIN 4
NID_OPERATIONAL 32
NC_TRAIN 15
L_TRAIN 12
V_MAXTRAIN 7
M_LOADINGGAUGE 8
M_AXLELOADCAT 7
M_AXLELOAD 7
M_AIRTIGHT 2
N_AXLE 10

N_ITER 5
M_TRACTION (k) 8
M_VOLTAGE(k) 4
NID_CTRACTION(k) 10 given only if M_VOLTAGE(k) ≠ 0
N_ITER 5
NID_STM (k) 8
NID_NTC(k) 8

Packet 4: Error reporting
NID_PACKET 8
L_PACKET 13
M_ERROR 8

Packet 9: Level 2/3 transition information
NID_PACKET 8
L_PACKET 13
NID_LTRBG 24

Stejná ID packetů pro verze 3.4.0 a 3.6.0 vyjma ID současně obsažených ve verzi 2.3.0:
Z RBC do EVC:
Packet 88: Level Crossing information
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
NID_LX 8
D_LX 15
L_LX 15
Q_LXSTATUS 1
V_LX 7 Only if Q_LXSTATUS = 1
Q_STOPLX 1 Only if Q_LXSTATUS = 1
L_STOPLX 15 Only if Q_STOPLX = 1

Packet 40: Track Condition Change of allowed current consumption
NID_PACKET 8

Q_DIR 2
L_PACKET 13
Q_SCALE 2
D_CURRENT 15
M_CURRENT 10

Packet 52: Permitted Braking Distance Information
NID_PACKET 8
Q_DIR 2
L_PACKET 13
Q_SCALE 2
Q_TRACKINIT 1

D_TRACKINIT 15 Only if Q_TRACKINIT = 1
D_PBD 15 Only if Q_TRACKINIT = 0, D_PBD and the following variables follow
Q_GDIR 1
G_PBDSR 8
Q_PBDSR 1
D_PBDSR 15
L_PBDSR 15
N_ITER 5
D_PBD(k) 15
Q_GDIR(k) 1
G_PBDSR(k) 8
Q_PBDSR(k) 1
D_PBDSR(k) 15
L_PBDSR(k) 15

Packet 64: Inhibition of revocable TSRs from balises in L2/3
NID_PACKET 8
Q_DIR 2
L_PACKET 13

Packet 69: Track Condition Station Platforms
NID_PACKET 8
Q_DIR 2
L_PACKET 13

Q_SCALE 2
Q_TRACKINIT 1

D_TRACKINIT 15 Only if Q_TRACKINIT = 1
D_TRACKCOND 15 Only if Q_TRACKINIT = 0, D_TRACKCOND and the following variables follow
L_TRACKCOND 15
M_PLATFORM 4
Q_PLATFORM 2
N_ITER 5
D_TRACKCOND(k) 15
L_TRACKCOND(k) 15
M_PLATFORM(k) 4
Q_PLATFORM(k) 2

Z EVC do RBC:
Packet 5: Train running number
NID_PACKET 8
L_PACKET 13
NID_OPERATIONAL 32

Stejná ID packetů pro verze 2.3.0 a 3.4.0 vyjma ID současně obsažených ve verzi 3.6.0:
Z RBC do EVC:
Žádná ID

Z EVC do RBC:
Packet 3: Onboard telephone numbers
NID_PACKET 8
L_PACKET 13
N_ITER 5
NID_RADIO (k) 64

ID packetů pro verzi 3.6.0 vyjma ID packetů současně obsažených ve verzích 2.3.0 a 3.4.0:
Z RBC do EVC:
Žádná ID

Z EVC do RBC:
Packet 2: Onboard supported system versions
NID_PACKET 8
L_PACKET 13
M_VERSION 7
N_ITER 5
M_VERSION (k) 7

Příloha C

Zprávy

V této příloze jsou uvedeny všechny zprávy z verzí 2.3.0, 3.4.0, 3.6.0 kapitoly 8 subsetu 026 System Requirements Specification, které jsou potřebné pro komunikaci mezi EVC a RBC v simulátoru ETCS.

- Modrou barvou jsou označeny části zprávy, které jsou pouze ve verzi 2.3.0. Pokud není uvedeno jinak, předpokládá se, že packety a proměnné označené touto barvou jsou pouze ve verzi 2.3.0 kapitoly 7 subsetu 026 nebo mají odlišnou vnitřní strukturu nebo v případě packetů odlišný název v jiných verzích (3.4.0 a 3.6.0) zmíněné kapitoly.

- Zelenou barvou jsou označeny části zprávy, které jsou pouze ve verzích 3.4.0 a 3.6.0. Pokud není uvedeno jinak, předpokládá se, že packety a proměnné označené touto barvou jsou současně obsaženy ve verzích 3.4.0 a 3.6.0 kapitoly 7 subsetu 026, ale nejsou obsaženy ve verzi 2.3.0 tohoto subsetu nebo jsou v ní obsaženy, ale mají v ní odlišnou vnitřní strukturu nebo v případě packetů odlišný název.

- Růžovou barvou jsou označeny části zprávy, které jsou pouze ve verzích 2.3.0 a 3.4.0. Pokud není uvedeno jinak, předpokládá se, že packety a proměnné označené touto barvou jsou současně obsaženy ve verzích 2.3.0 a 3.4.0 kapitoly 7 subsetu 026, ale nejsou obsaženy ve verzi 2.3.0 tohoto subsetu nebo jsou v ní obsaženy, ale mají v ní odlišnou vnitřní strukturu nebo v případě packetů odlišný název.

- Šedou barvou jsou označeny části zprávy, které jsou pouze ve verzi 3.6.0. Pokud není uvedeno jinak, předpokládá se, že packety a proměnné označené touto barvou jsou pouze ve verzi 3.6.0 kapitoly 7 subsetu 026 nebo mají odlišnou vnitřní strukturu nebo v případě packetů odlišný název v jiných verzích (3.4.0 a 3.6.0) zmíněné kapitoly.

Vedle ID zprávy je napsán název této zprávy. Vnitřní struktura zprávy je převzata z kapitoly 8 subsetu 026. Proměnné, které nejsou označeny žádnou barvou, mají stejnou délku v bitech a rozsahy platných hodnot ve verzích 2.3.0, 3.4.0 a 3.6.0 kapitoly 7 subsetu 026. Packety, které nejsou označeny žádnou barvou, mají stejnou vnitřní strukturu a název ve verzích 2.3.0, 3.4.0 a 3.6.0 kapitoly 7 subsetu 026.

Stejná ID zpráv pro verze 2.3.0, 3.4.0 a 3.6.0:
Z RBC do EVC:
Message 2: SR Authorisation
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
Q_SCALE
D_SR
Optional Packet 63

Message 3: Movement Authority
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
Packet 15
Packet 15
Optional packets: 21, 49, 5, 42, 45, 57, 58, 65, 66, 68, 71, 131, 138, 139, 27, 80, 3, 39, 51, 41, 70, 72, 76, 79, 140, 27, 80, 3, 39, 51, 41, 70, 72, 76, 79, 140, 40, 52, 64, 69, 88

Message 6: Recognition of exit from TRIP mode
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG

Message 8: Acknowledgement of Train Data
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG

T_TRAIN

Message 9: Request to Shorten MA
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
Packet 15
Packet 15
Optional Packet 80, Packet 80, Packet 49
Poznámka: Packet 49 má stejný název a vnitřní strukturu pro verze 2.3.0, 3.4.0 a 3.6.0. V této zprávě je tento packet obsažen pouze ve verzích 3.4.0 a 3.6.0.

Message 15: Conditional Emergency Stop
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
NID_EM
Q_SCALE
D_REF
Q_DIR
D_EMERGENCYSTOP

Message 16: Unconditional Emergency Stop
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
NID_EM

Message 18: Revocation of Emergency Stop
NID_MESSAGE

L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
NID_EM

Message 24: General message
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
Optional packets: 21, 5, 42, 45, 57, 58, 65, 66, 68, 71, 131, 138, 139, 27, 3, 39, 51, 41, 70, 72, 76, 79, 140, 27, 3, 39, 51, 41, 70, 72, 76, 79, 140, 40, 52, 64, 69, 88

Message 27: SH Refused
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
T_TRAIN

Message 28: SH Authorised
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
T_TRAIN
Optional packets: 49, 5, 42, 45, 57, 58, 65, 66, 68, 71, 131, 138, 139, 3, 39, 51, 41, 70, 72, 76, 79, 140, 3
Poznámka: Packety s ID 5, 42, 45, 57, 58, 65, 66, 68, 71, 131, 138, 139 mají stejný název a stejnou vnitřní strukturu pro verze 2.3.0, 3.4.0 a 3.6.0. V této zprávě jsou tyto packety obsaženy pouze ve verzi 2.3.0.

Message 32: Configuration Determination
RBC/RIU System Version
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
M_VERSION
M_VERSION

Message 33: MA with Shifted Location Reference
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
Q_SCALE
D_REF
Packet 15
Packet 15
Optional packets: 21, 49, 5, 42, 45, 57, 58, 65, 66, 68, 71, 131, 138, 139, 27, 80, 3, 39, 51, 41, 70, 72, 76, 79, 140, 27, 80, 3, 39, 51, 41, 70, 72, 76, 79, 140, 40, 52, 64, 69, 88

Message 34: Track Ahead Free Request
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
Q_SCALE
D_REF
Q_DIR
D_TAFDISPLAY
L_TAFDISPLAY

Message 39: Acknowledgement of termination of a communication session
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK (always set to 0)
NID_LRBG

Message 40: Train Rejected
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG

Message 41: Train Accepted
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG

Message 43: SoM position report confirmed by RBC
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG

Message 45: Assignment of coordinate system
NID_MESSAGE
L_MESSAGE
T_TRAIN
M_ACK
NID_LRBG
Q_ORIENTATION

Z EVC do RBC:

Message 129: Validated Train Data
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
Packet 0 or 1
Packet 0 or 1
Packet 11
Packet 11

Message 130: Request for Shunting
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
Packet 0 or 1
Packet 0 or 1

Message 132: MA Request
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
Q_TRACKDEL
Q_MARQSTREASON
Packet 0 or 1
Packet 0 or 1
Optional Packet 9

Message 136: Train Position Report
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
Packet 0 or 1
Packet 0 or 1

Optional Packet 4, Packet 5

Message 137: Request to Shorten MA is granted
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
T_TRAIN
Packet 0 or 1
Packet 0 or 1

Message 138: Request to Shorten MA is rejected
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
T_TRAIN
Packet 0 or 1
Packet 0 or 1

Message 146: Acknowledgement
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
T_TRAIN

Message 147: Acknowledgement of Emergency Stop
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
NID_EM
Q_EMERGENCYSTOP
Packet 0 or 1
Packet 0 or 1

Message 149: Track Ahead Free Granted
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
Packet 0 or 1
Packet 0 or 1

Message 150: End of Mission
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
Packet 0 or 1
Packet 0 or 1

Message 154: No compatible version
No compatible version supported
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE

Message 155: Initiation of a communication session
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE

Message 156: Termination of a communication session
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE

Message 157: SoM Position Report
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
Q_STATUS
Packet 0 or 1
Packet 0 or 1
Optional Packet 4, Packet 5

Message 159: Session established
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
Packet 2
Optional Packet 3

Stejná ID zpráv pro verze 3.4.0 a 3.6.0 vyjma ID současně obsažených ve verzi 2.3.0:
Z RBC do EVC:
Žádná ID

Z EVC do RBC:
Message 158: Text Message Acknowledged by Driver
NID_MESSAGE
L_MESSAGE
T_TRAIN
NID_ENGINE
NID_TEXTMESSAGE
Packet 0 or 1

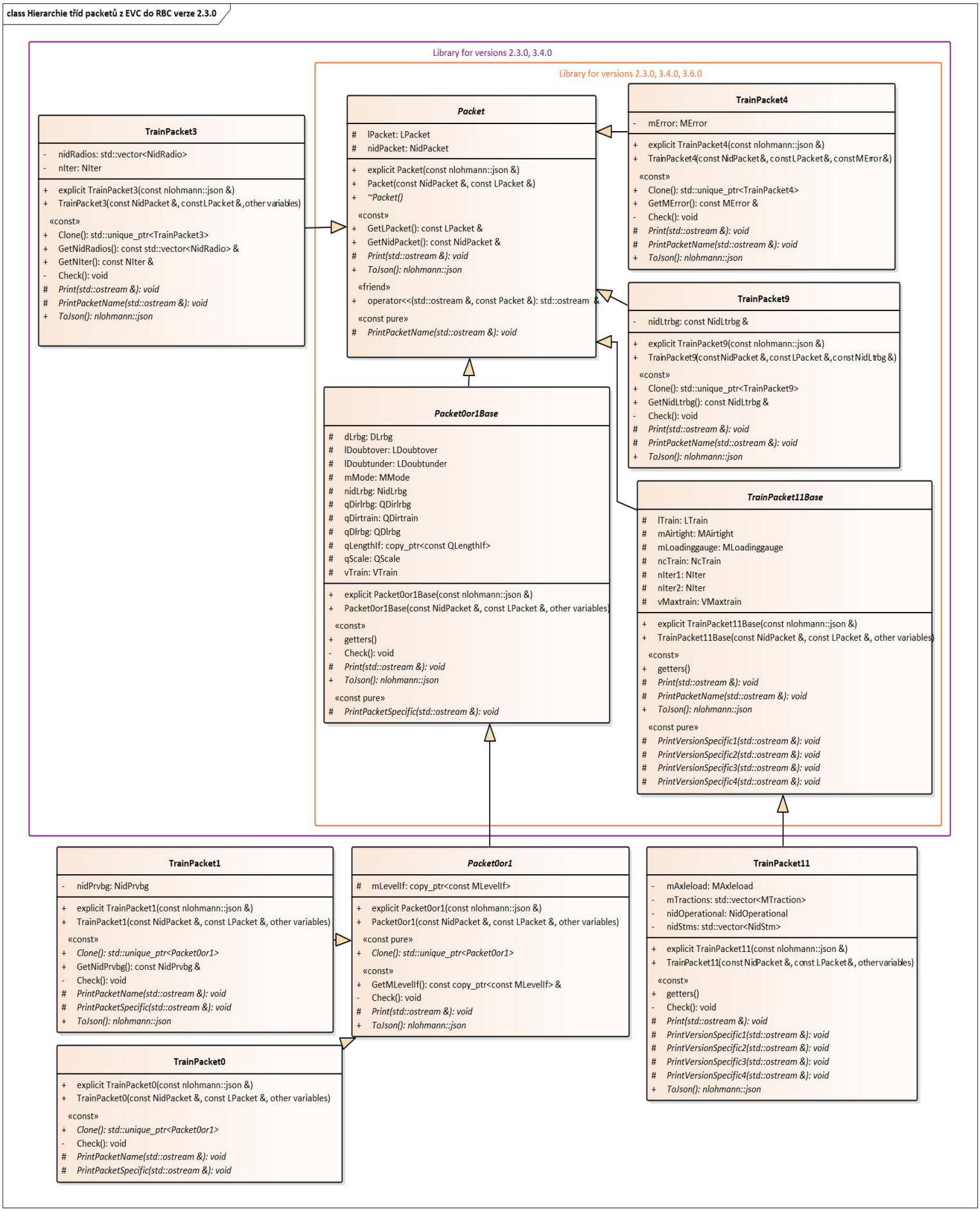
Stejná ID zpráv pro verze 2.3.0 a 3.4.0 vyjma ID současně obsažených ve verzi 3.6.0:
Z RBC do EVC:
Message 38: Initiation of a communication session

NID_MESSAGE
L_MESSAGE
T_TRAIN (always set to unknown)
M_ACK (always set to 1)
NID_LRBG (always set to unknown)

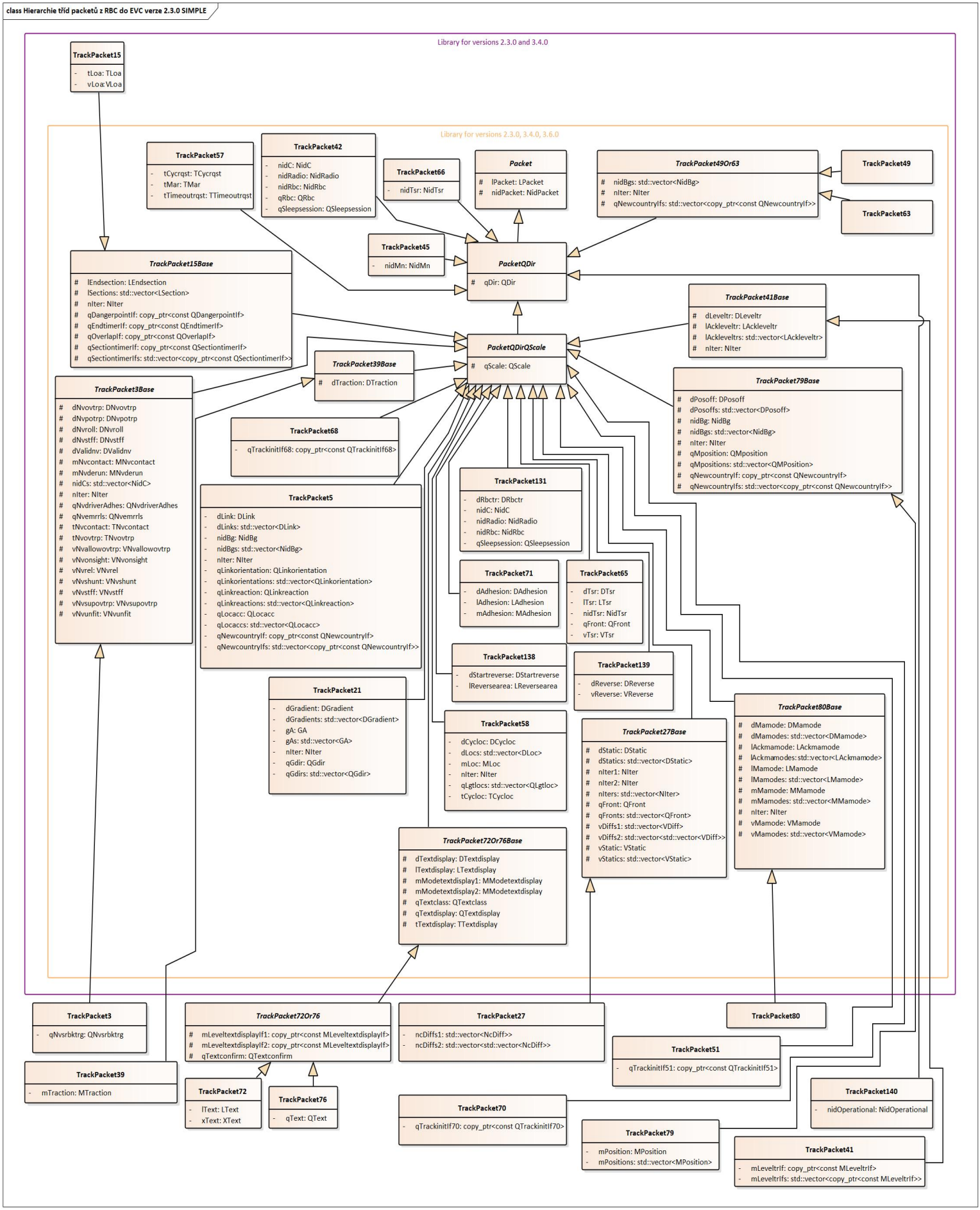
Z EVC do RBC:
Žádná ID

Příloha D

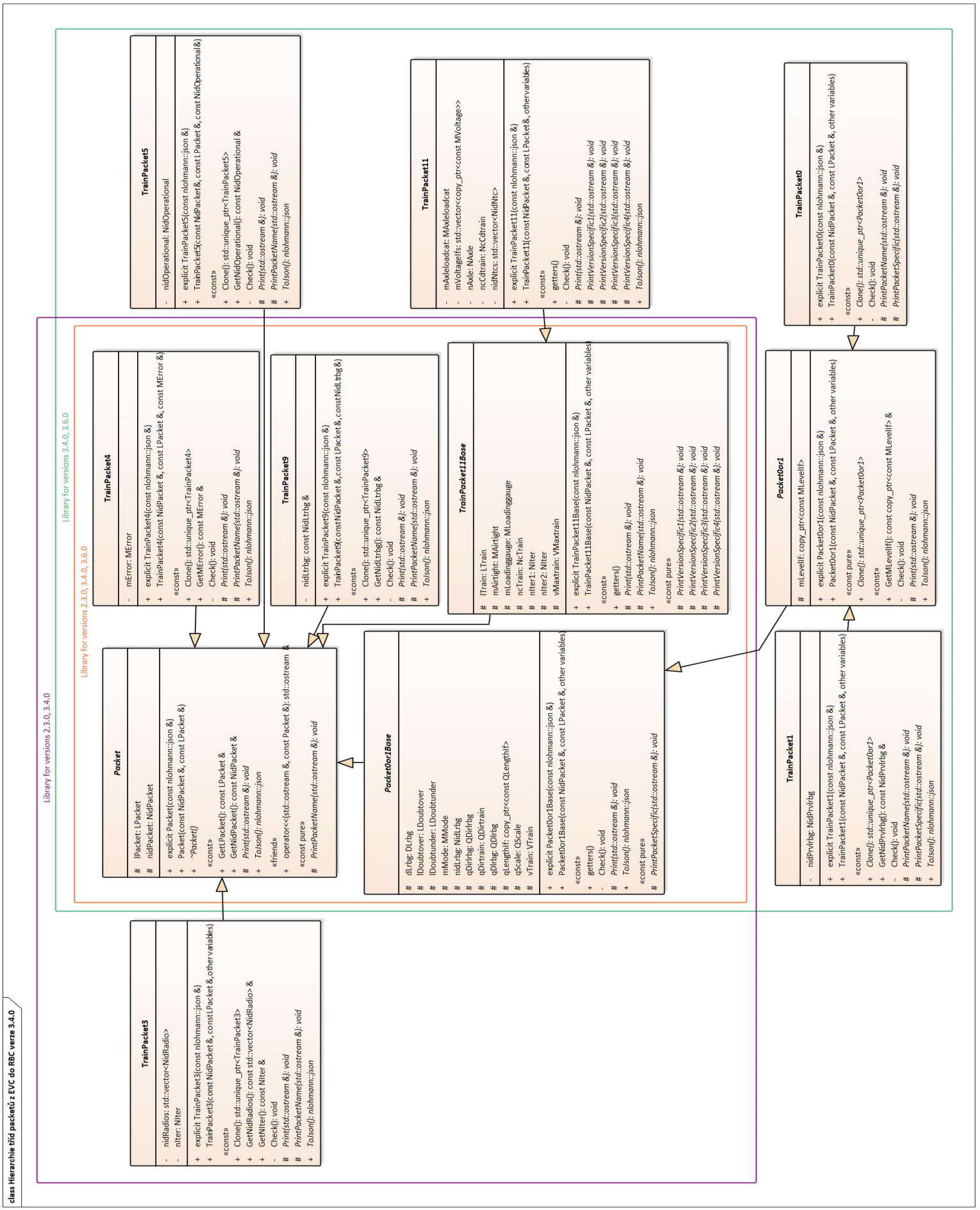
Hierarchie tříd packetů



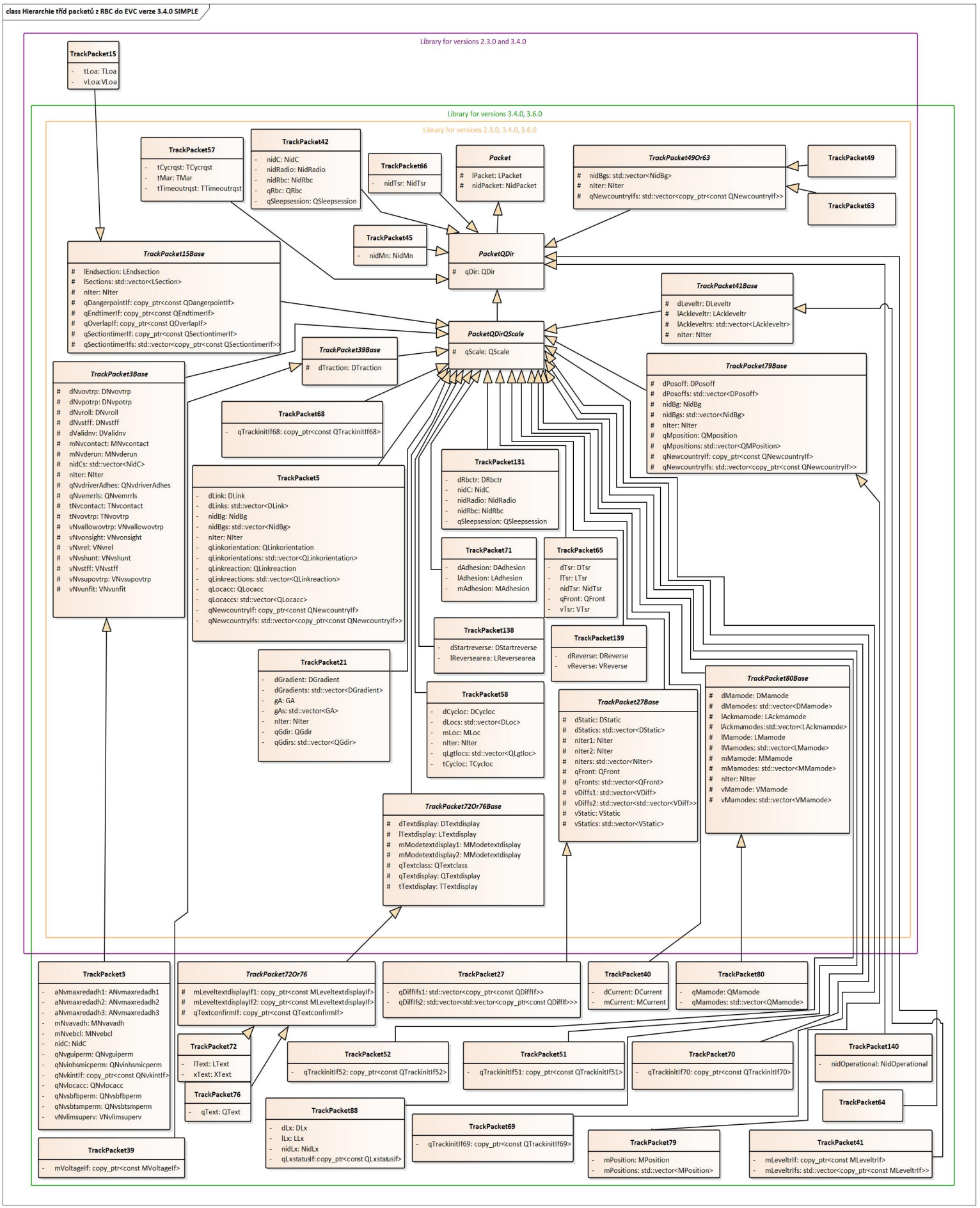
Obrázek D.1 Hierarchie tříd packetů z EVC do RBC verze 2.3.0



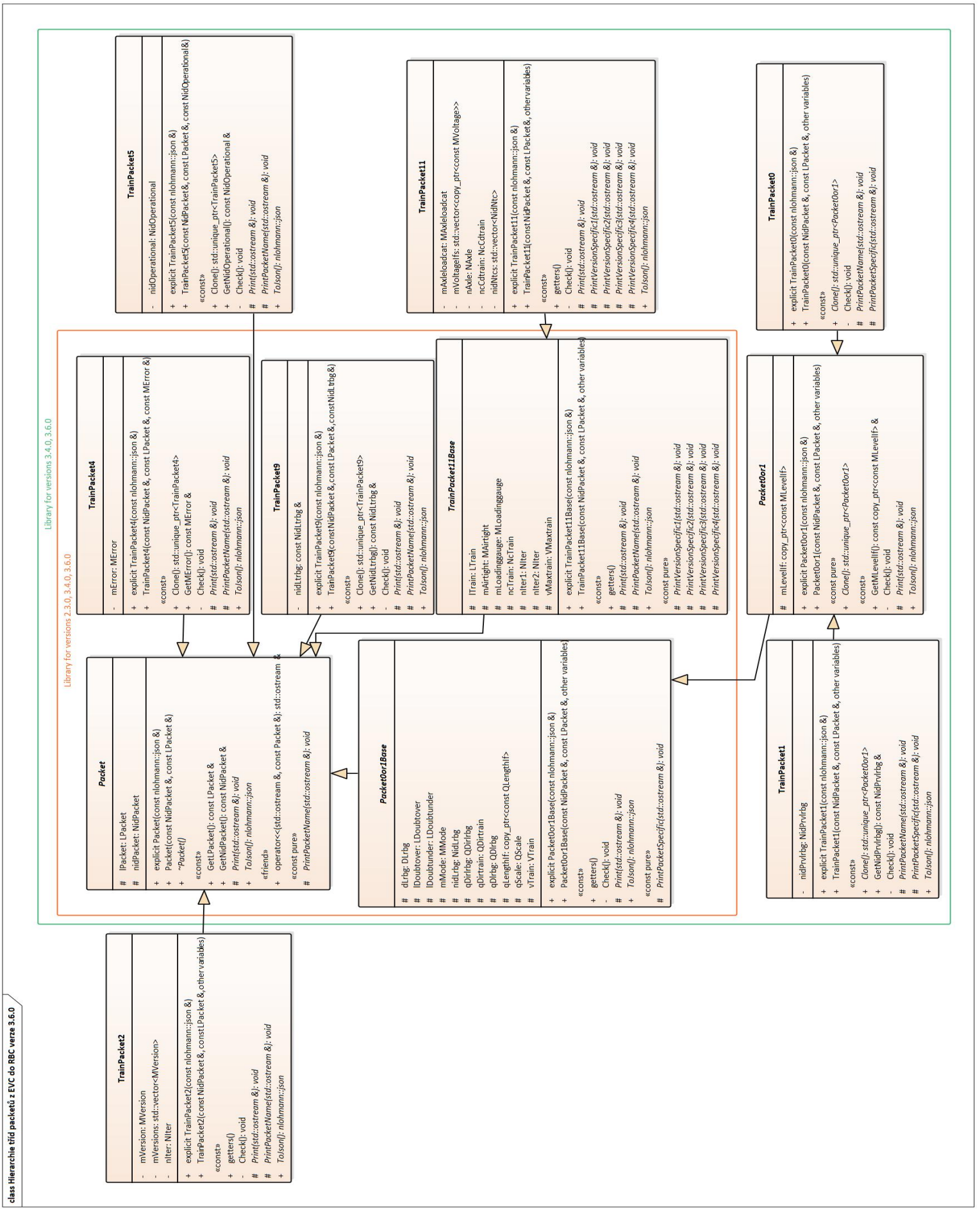
Obrázek D.2 Hierarchie tříd packetů z RBC do EVC verze 2.3.0, zjednodušený diagram



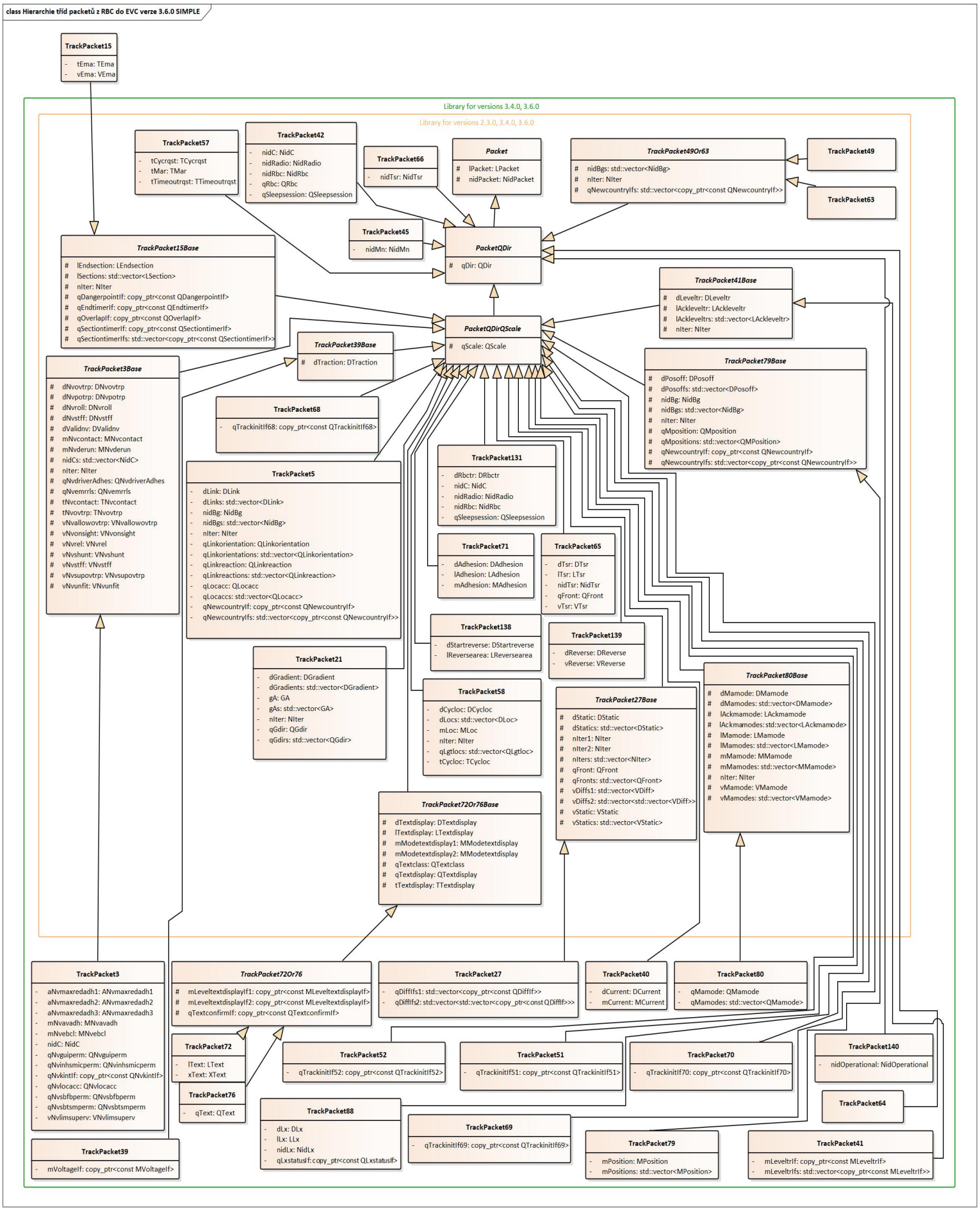
Obrázek D.3 Hierarchie tříd packetů z EVC do RBC verze 3.4.0



Obrázek D.4 Hierarchie tříd packetů z RBC do EVC verze 3.4.0, zjednodušený diagram



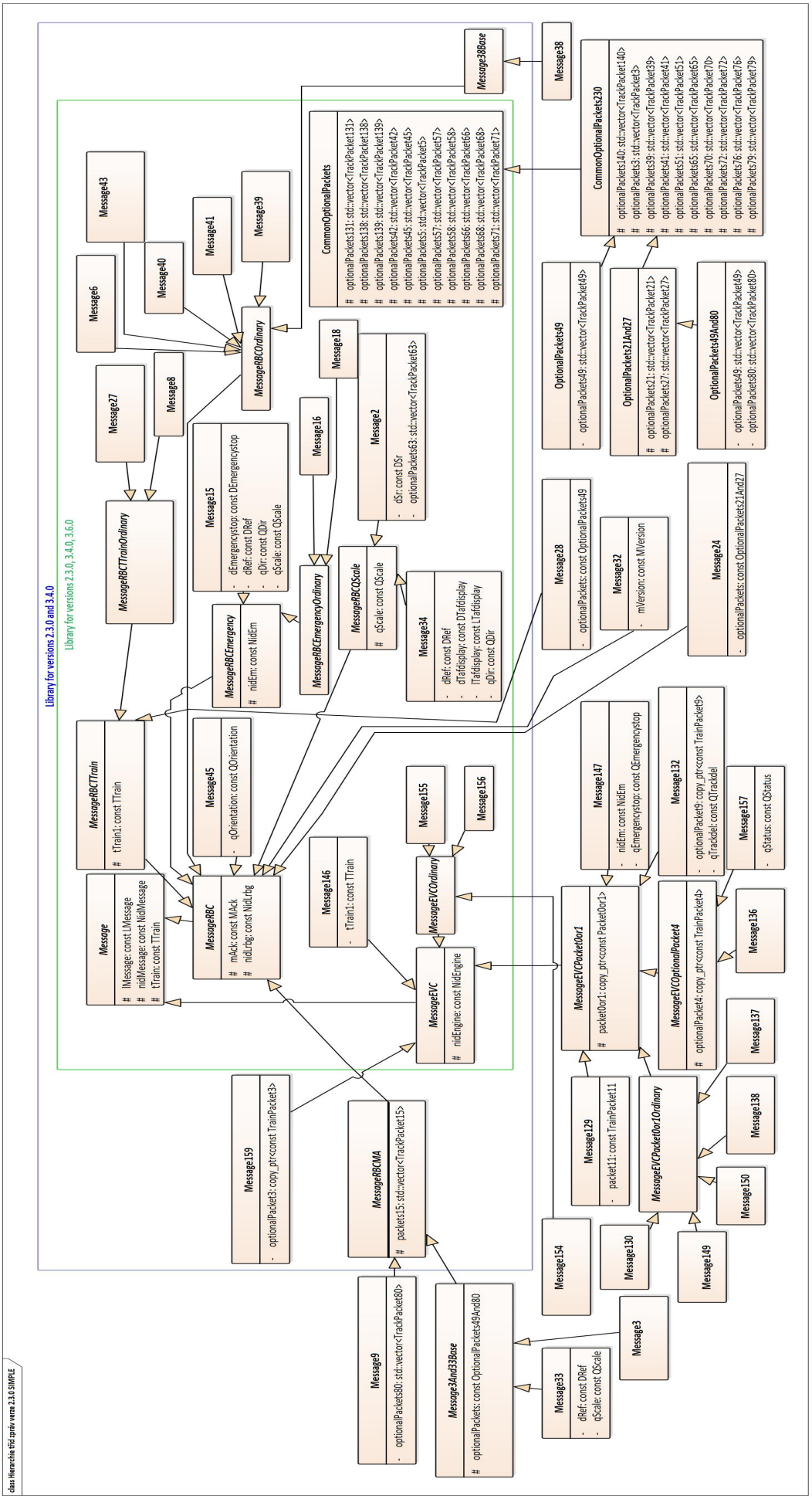
Obrázek D.5 Hierarchie tříd packetů z EVC do RBC verze 3.6.0



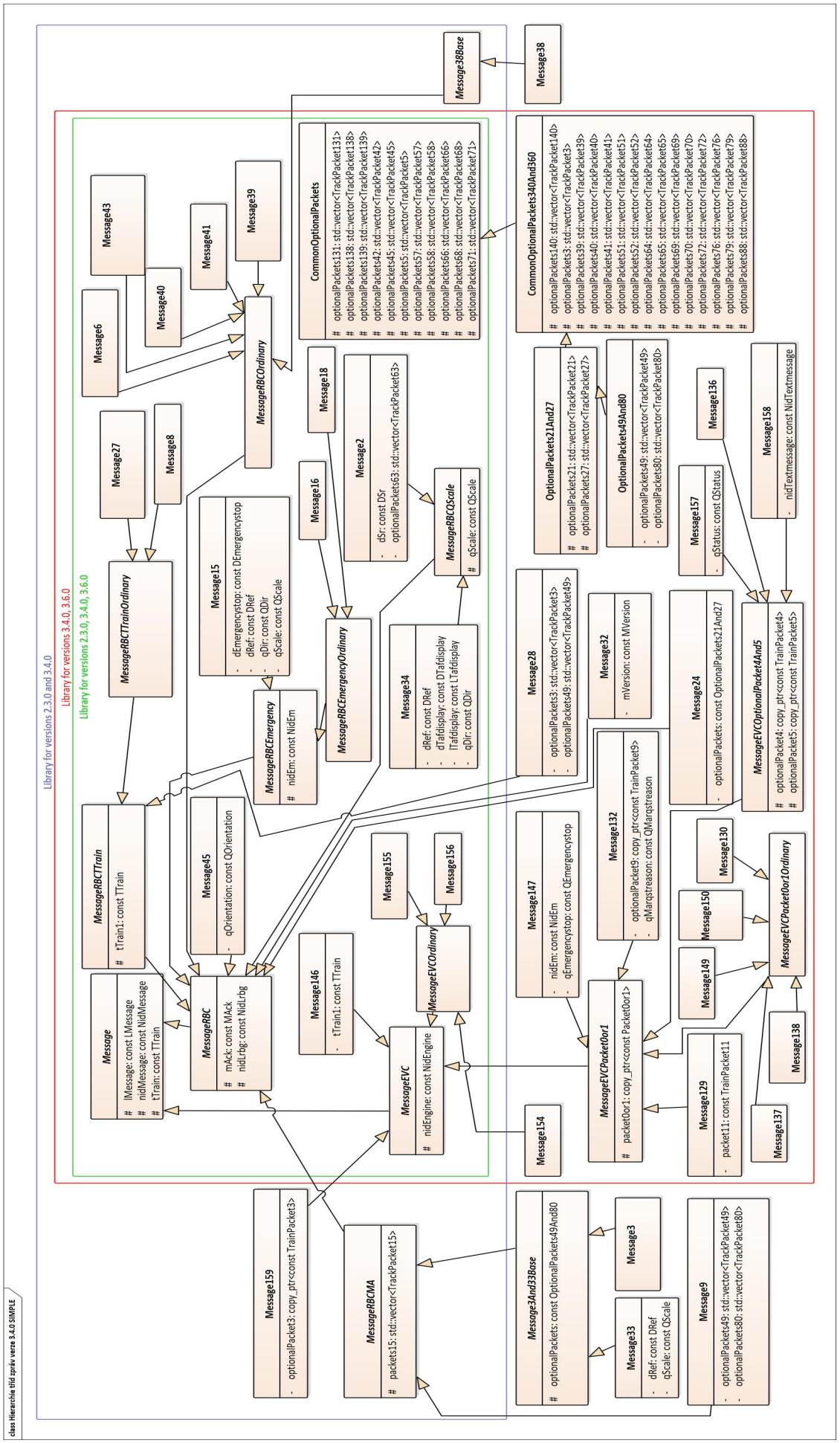
Obrázek D.6 Hierarchie tříd packetů z RBC do EVC verze 3.6.0, zjednodušený diagram

Příloha E

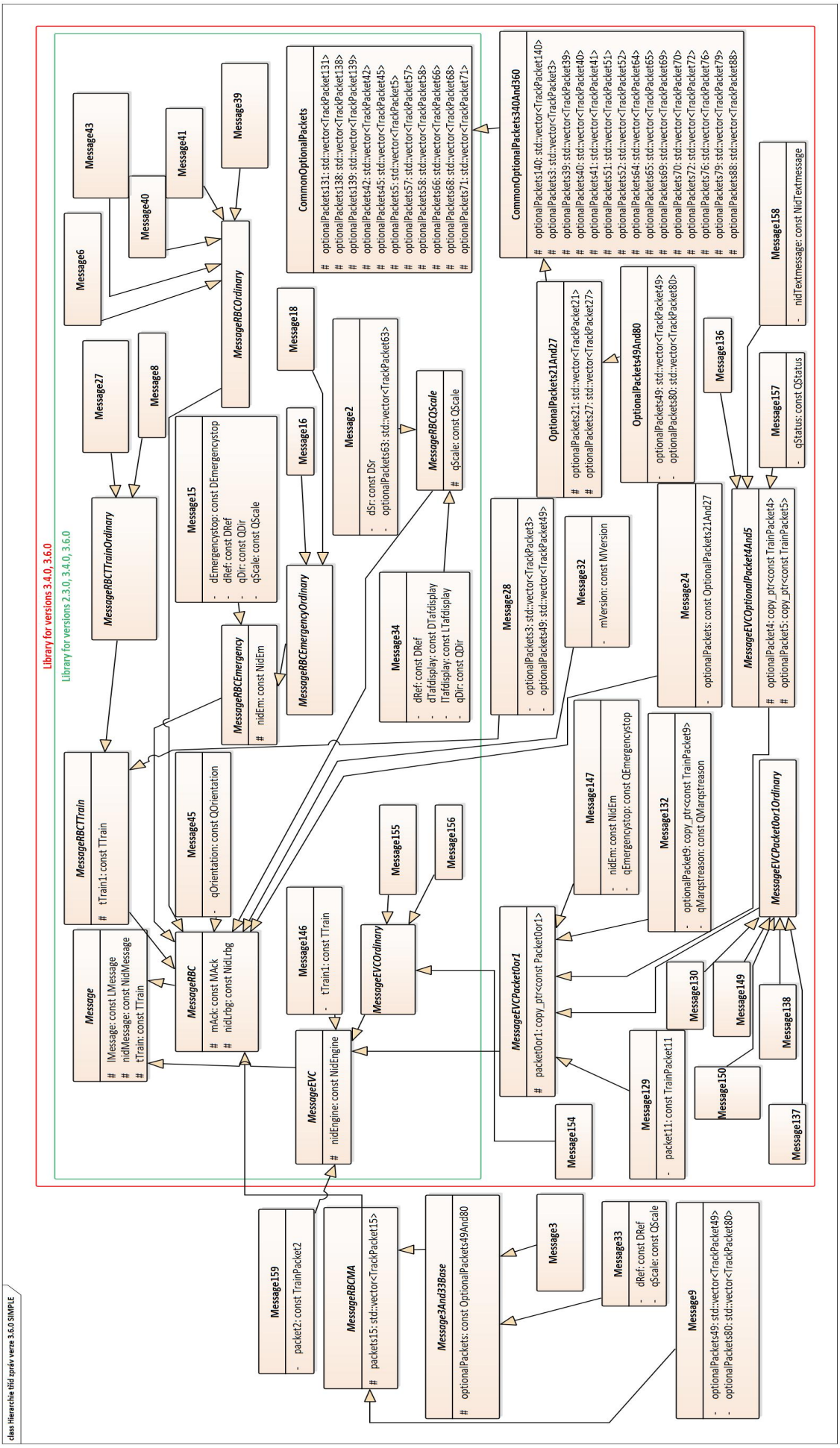
Hierarchie tříd zpráv



Obrázek E.1 Hierarchie tříd zpráv verze 2.3.0, zjednodušený diagram



Obrázek E.2 Hierarchie tříd zpráv verze 3.4.0, zjednodušený diagram



Obrázek E.3 Hierarchie tříd zpráv verze 3.6.0, zjednodušený diagram

Příloha F

Testování tříd packetů

```
bool createPacket58() {
    try {
        std::vector<DLoc> dLocs;
        dLocs.emplace_back(DLoc(101));
        dLocs.emplace_back(DLoc(200));
        std::vector<QLgtloc> qLgtlocs;
        qLgtlocs.emplace_back(QLgtloc(1));
        qLgtlocs.emplace_back(QLgtloc(0));
        TrackPacket58 trackPacket58(NidPacket(58), QDir(1), LPacket(88),
                                    QScale(1), TCycloc(100), DCycloc(1000),
                                    MLoc(5), NIter(2), dLocs, qLgtlocs);
        if (trackPacket58.GetNidPacket() == NidPacket(58) &&
            trackPacket58.GetQDir() == QDir(1) &&
            trackPacket58.GetLPacket() == LPacket(88) &&
            trackPacket58.GetQScale() == QScale(1) &&
            trackPacket58.GetTCycloc() == TCycloc(100) &&
            trackPacket58.GetDCycloc() == DCycloc(1000) &&
            trackPacket58.GetMLoc() == MLoc(5) &&
            trackPacket58.GetNIter() == NIter(2) &&
            trackPacket58.GetDLocs() == dLocs &&
            trackPacket58.GetQLgtlocs() == qLgtlocs)
            return true;
    }
    catch(const NotPacketException & ex){
        return false;
    }
    return false;
}
```

Výpis kódu F.1 Testování vytvoření packetu pomocí konstruktoru s proměnnými

```
bool createPacket58JSON() {
    nlohmann::json j = {
            {"NID_PACKET", 58},
            {"Q_DIR", 1},
            {"L_PACKET", 88},
            {"Q_SCALE", 1},
            {"N_ITER", 2},
            {"T_CYCLOC", 1},
            {"D_CYCLOC", 10},
            {"M_LOC", 4}
    };
    nlohmann::json dLoc1, dLoc2, qLgtloc1, qLgtloc2;
    dLoc1["D_LOC"] = 1;
    dLoc2["D_LOC"] = 2;
    qLgtloc1["Q_LGTLOC"] = 0;
    qLgtloc2["Q_LGTLOC"] = 1;
    j["dLocs"].emplace_back(dLoc1);
    j["dLocs"].emplace_back(dLoc2);
    j["qLgtlocs"].emplace_back(qLgtloc1);
    j["qLgtlocs"].emplace_back(qLgtloc2);
    try {
        TrackPacket58 trackPacket58(j);
        if (trackPacket58.ToJson() == j)
            return true;
    }
    catch (const NotPacketException & ex) {
        return false;
    }
    return false;
}
```

Výpis kódu F.2 Testování vytvoření packetu pomocí konstruktoru s JSON objektem

```
bool createPacket58BadLength() {
    nlohmann::json j = {
            {"NID_PACKET", 58},
            {"Q_DIR", 1},
            {"L_PACKET", 98},
            {"Q_SCALE", 1},
            {"N_ITER", 2},
            {"T_CYCLOC", 1},
            {"D_CYCLOC", 10},
            {"M_LOC", 4}
    };
    nlohmann::json dLoc1, dLoc2, qLgtloc1, qLgtloc2;
    dLoc1["D_LOC"] = 1;
    dLoc2["D_LOC"] = 2;
    qLgtloc1["Q_LGTLOC"] = 0;
    qLgtloc2["Q_LGTLOC"] = 1;
    j["dLocs"].emplace_back(dLoc1);
    j["dLocs"].emplace_back(dLoc2);
    j["qLgtlocs"].emplace_back(qLgtloc1);
    j["qLgtlocs"].emplace_back(qLgtloc2);
    try {
        TrackPacket58 trackPacket58(j);
    }
    catch (const NotPacketException & ex) {
        return true;
    }
    return false;
}
```

Výpis kódu F.3 Testování, že se nevytvoří packet s hodnotou L_PACKET neodpovídající skutečné délce packetu

```
bool createPacket58MoreVar() {
    nlohmann::json j = {
            {"NID_PACKET", 58},
            {"Q_DIR", 1},
            {"L_PACKET", 88},
            {"Q_SCALE", 1},
            {"N_ITER", 2},
            {"T_CYCLOC", 1},
            {"D_CYCLOC", 10},
            {"M_LOC", 4},
            {"N_ITER2", 2}
    };
    nlohmann::json dLoc1, dLoc2, qLgtloc1, qLgtloc2;
    dLoc1["D_LOC"] = 1;
    dLoc2["D_LOC"] = 2;
    qLgtloc1["Q_LGTLOC"] = 0;
    qLgtloc2["Q_LGTLOC"] = 1;
    j["dLocs"].emplace_back(dLoc1);
    j["dLocs"].emplace_back(dLoc2);
    j["qLgtlocs"].emplace_back(qLgtloc1);
    j["qLgtlocs"].emplace_back(qLgtloc2);
    try {
        TrackPacket58 trackPacket58(j);
    }
    catch (const NotPacketException & ex) {
        return true;
    }
    return false;
}
```

Výpis kódu F.4 Testování, že se nevytvoří packet z JSON objektu obsahujícího více datových položek než je členských proměnných třídy příslušného packetu

```
bool createPacket58BadVectorSize() {
    nlohmann::json j = {
            {"NID_PACKET", 58},
            {"Q_DIR", 1},
            {"L_PACKET", 88},
            {"Q_SCALE", 1},
            {"N_ITER", 2},
            {"T_CYCLOC", 1},
            {"D_CYCLOC", 10},
            {"M_LOC", 4},
            {"N_ITER2", 2}
    };
    nlohmann::json dLoc1, qLgtloc1, qLgtloc2;
    dLoc1["D_LOC"] = 1;
    qLgtloc1["Q_LGTLOC"] = 0;
    qLgtloc2["Q_LGTLOC"] = 1;
    j["dLocs"].emplace_back(dLoc1);
    j["qLgtlocs"].emplace_back(qLgtloc1);
    j["qLgtlocs"].emplace_back(qLgtloc2);
    try {
        TrackPacket58 trackPacket58(j);
    }
    catch (const NotPacketException & ex) {
        return true;
    }
    return false;
}
```

Výpis kódu F.5 Testování, že se nevytvoří packet s vektory, jejichž velikost neodpovídá hodnotě N_ITER

```
bool createTrackPacket15Nullptr() {
    try {
        std::vector<LSection> lSections;
        lSections.emplace_back(LSection(12));
        lSections.emplace_back(LSection(13));
        std::vector<copy_ptr<const QSectiontimerIf>> qSectiontimerIfs;
        qSectiontimerIfs.emplace_back(std::make_unique<QSectiontimer0>(QSectiontimer(0)));
        qSectiontimerIfs.emplace_back(nullptr);
        copy_ptr<const QSectiontimerIf> qSectiontimerIf(std::make_unique<QSectiontimer0>
                                                        (QSectiontimer(0)));
        copy_ptr<const QEndtimerIf> qEndtimerIf(std::make_unique<QEndtimer0>(QEndtimer(0)));
        copy_ptr<const QDangerpointIf> qDangerpointIf(std::make_unique<QDangerpoint0>
                                                      (QDangerpoint(0)));
        copy_ptr<const QOverlapIf> qOverlapIf(std::make_unique<QOverlap0>(QOverlap(0)));
        TrackPacket15 trackPacket15(NidPacket(15), QDir(1), LPacket(97), QScale(0),
                                    VLoa(10),TLoa(14), NIter(2), lSections,
                                    qSectiontimerIfs, LEndsection(9), qSectiontimerIf,
                                    qEndtimerIf, qDangerpointIf, qOverlapIf);
    }
    catch(const NotPacketException & ex){
        return true;
    }
    return false;
}
```

Výpis kódu F.6 Testování, že nedojde k vytvoření packetu s copy_ptr na bázovou třídu s rozhodovací proměnnou, který je rovný nullptr

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `impl`
  - `src` ..... zdrojové kódy implementace
  - `tests` ..... testy
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF