České vysoké učení technické v Praze
Fakulta elektrotechnická

# DIPLOMOVÁ PRÁCE

## Nástroje pro práci s CANopen slovníky a jejich integrace do vestavěných zařízení

**Praha, 2012** **Autor: Bc. Ivan Fridrich**

## Prohlášení

Prohlašuji, že jsem svou diplomovou ( bakalářskou) práci vypracoval samostatně a použil jsem pouze podklady ( literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne _____ _____

podpis

## Poděkování

Velmi rád bych poděkoval a vyslovil uznání všem, kteří mi pomáhali při vzniku této práce. Děkuji především vedoucímu práce Ing. Pavlu Píšovi, Ph.D.. Dále bych chtěl také poděkovat Ing. Františku Vackovi. V neposlední řadě také své rodině a přátelům.

# Abstrakt

Hlavním cílem práce je tvorba nástroje pro testování, konfiguraci a podporu vývoje zařízení, která jsou propojená sběrnicí CAN a na vyšší vrstvě využívají protokol CANopen. Po úvodu do problematiky sběrnice CAN a protokolu CANopen následuje rozbor využitelnosti již existujících open-source řešení. Za základ byl zvolený projekt QCANanalyzer. Grafické rozhraní programu je založeno na knihovně Qt, se kterou je čtenář v krátkosti seznámen. Dále následuje rozbor stávající implementace programu QCAnanalyzer a popis provedených úprav a implementace subsystému pro pohodlný přístup ke slovníkům CANopen zařízení s využitím popisu v souborech EDS. Dále jsou demonstrované možnosti návrhu CANopen zařízení s využitím knihovny CANfestival. Popsané jsou i další programy a nástroje, které byly při vývoji a testování použité.

# Abstract

Implementation of a tools for testing, configuration and development of devices that are connected to the CAN bus and use CANopen higher layer protocol is a main goal of the presented work. The first chapter focuses to an introduction to the CAN bus communication and CANopen protocol. It is followed by an analysis of existing open-source solutions providing CAN/CANopen support which could be integrated and used to achieve thesis goals. More detailed description of QCANanalyzer program follows because that project has served as a base for work described in next chapters. The program is based on Qt GUI libraries. A basic concepts used in Qt libraries and Qt based application is described later. This is followed by an analysis of functionality already provided by QCAnanalyzer. Then the program changes and implementation of the subsystem for a comfortable access to dictionaries, using the CANopen device described in the EDS file, is documented. Functionality of the implemented CANopen dictionary tool are demonstrated on a preparation of CANopen device with help of CANfestival library. Described are even some other programs and tools used for developing and testing.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Ivan Fridrich**

Studijní program: Otevřená informatika (magisterský)
Obor: Počítačové inženýrství

Název tématu: **Nástroje pro práci s CANopen slovníky a jejich integrace do vestavných zařízení**

Pokyny pro vypracování:

1. Implementujte nástroj pro práci se slovníkem CANopen v grafickém prostředí Qt4
2. Integrujte jej do programu pro monitoring a analýzu provozu na sběrnici CAN
3. Vytvořte testovací CANopen zařízení s využitím knihoven CANfestival a již existujícího hardware s mikrokontrolérem s procesorovým jádrem Cortex-M
4. Navrhněte postup pro příprav a testování vlastních slovníků na použitém hardware

Seznam odborné literatury:


[1] CiA Standard 301 (DS301), CANopen application layer and communication profile.
[2] Webové stránky projektu OrtCAN http://ortcan.sourceforge.net/
[3] Webové stránky projektu CANfestival http://www.canfestival.org/
[4] Webové stránky obdobného projektu pro Windows http://rs.canlab.cz/
[5] LPC17xx User manual, Rev. 2, NXP Semiconductors, 19 August 2010


Vedoucí: Ing. Pavel Píša, Ph.D.

Platnost zadání: do konce letního semestru 2012/2013

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 8. 12. 2011

# Obsah

# Seznam obrázků

# Seznam tabulek

# Kapitola 1

# Úvod

Sběrnice CAN byla navržena pro použití v automobilech, kde přispívá k zjednodušení propojení a především přidává konfigurovatelnost a diagnostiku řídících jednotek, senzorů a akčních členů. Pro otevřenost a kvalitu svého návrhu a dostupnost na mnoha řídicích mikrokontrolérech se také velmi rozšířila v oblasti automatizace. Slouží zde k propojení řídících jednotek, senzorů a akčních členů. Tato sběrnice definuje podle ISO/OSI modelu fyzickou a linkovou vrstvu. Na této sběrnici je založen vyšší protokol CANopen, který definuje síťovou vrstvu a vyšší vrstvy.

Standard CANopen specifikuje přístupy k servisním parametrům zařízení uspořádaných do objektového slovníku (dále OD - object dictionary). Specifikuje pravidla pro konfiguraci přenosů procesních dat a standardní profily chování a OD slovníků pro předdefinované kategorie zařízení.

Každé zařízení má definovaný objektový slovník v souboru s popisem rozhranní zařízení specifikovaném normou (soubor EDS - electronic datsheet). CANopen definuje i další pomocné protokoly.

Při vývoji je vhodné mít nástroj na monitorování a odesílání zpráv na sběrnici. Z toho důvodu byl vytvořen projekt QCANanalyzer. Jedná se o grafický program, který v původní verzi implementoval pouze záznam provozu na sběrnici a posílání základních zpráv na sběrnici CAN. Mým cílem bylo doplnit tento projekt o podporu komunikace CANopen. To zahrnuje nástroje pro čtení a úpravu EDS souborů a napojení na již implementované drivery pro přístup k sběrnici CAN. Během rozvoje projektu bylo také opraveno množství chyb a dokončena funkce pro grafické zobrazení průběhu hodnot procesních proměnných přenášených CAN zprávami.

QCANalyser je napsaný v jazyce C/C++ je založen na knihovně Qt a na knihovně

grafických komponent pro technické aplikace Qwt.

Pojednání o grafických technologiích je součástí této práce. Software bylo potřeba řádně otestovat. K tomuto účelu posloužili pomocné programy projektu Ortcan do nehož je QCANalyser začleněn. K testování bylo též použito softwarové CANopen zařízení vytvořené s využitím CANopen frameworku CanFestival. Cílem práce bylo vytvořit softwarové nástroje pro komplexní práci s CANopen, zahrnující práci se slovníky v EDS souborech a komunikaci po sběrnici CANopen. V práci se dozvíte podrobně o technologiích CAN a CANopen, o grafických softwarových technologiích, bude zde rozebrána funkčnost QCANalyseru a moje práce na něm, dozvíte se o projektech Ortcan, CanFestival a o pomocných nástrojích, které byly využity k testování QCANalyseru.

# Kapitola 2

# Sběrnice CAN

## 2.1 Úvod

V této sekci bude rozebrána sběrnice CAN.
CAN (Controller Area Network) je sériová sběrnice využívaná nejčastěji pro realizaci komunikačních senzorových sítí například v automobilech nebo v průmyslu. Je vhodná pro *real-time* systémy. Vyvinutá byla firmou Robert Bosh GmBH na konci osmdesátých let. Sběrnice CAN ve verzi 2.0 byla uvedena na trh v roce 1991. Postupem doby se modifikovala do dvou vzájemně kompatibilních systému CAN 2.0A a CAN 2.0B. V roce 1993 byla sběrnice standardizována mezinárodním standardem ISO 11898, v České republice pak v normě CSN EN 50325. CAN sběrnice je specifická v tom, že jednotlivá zařízení na sběrnici nemají žádné vlastní adresy, nýbrž jednotlivé zprávy mají svoje *identifikátory*, a tak je možné velmi jednoduše přidávat další zařízení bez nutnosti zásahu do software již připojených zařízení.

## 2.2 Fyzická vrstva

Je to třívodičová sběrnice (CANH, CANL a GND) někdy doplněná ještě o stínění a vodič s napájecím napětím +5 V. Vysoká odolnost proti souhlasnému rušení je zajištěna diferenciálním přenosem po dvou vodičích (CANH a CANL). Sběrnice rozlišuje dvě napěťové úrovně tzv. *recesivní* a *dominantní* úroveň. Dominantní úroveň je když na vodiči CANH je napětí mezi 3,5 V a 5 V a na vodiči CANL menší než 1,5 V. V recesivním stavu není rozdíl napětí vodičů větší než 0,5 V. Pokud alespoň jedno zařízení na sběrnici nastaví výstup

do dominantní úrovně, pak je tato úroveň přenesena na všechna zařízení na sběrnici – sběrnice realizuje operaci *logický součin*. Kvůli zajištění proti vlnovým odrazům, musí být na obou koncích vedení rezistor o hodnotě cca 120 Ω. Protože každý kabel má svoji indukčnost a kapacitu, což způsobuje zpoždění signálu, maximální přenosové rychlosti CAN sběrnice jsou závislé na délce kabelu, viz. tabulka.

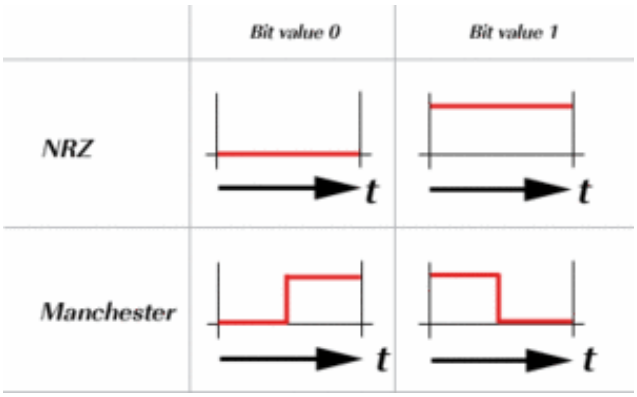
| Bit rate | Bus lenght | Nominal bit-time |
|---|---|---|
| 1 Mbit/s | 30 m | 1 us |
| 800 kbit/s | 50 m | 1.25 us |
| 500 kbit/s | 100 m | 2 us |
| 250 kbit/s | 250 m | 4 us |
| 125 kbit/s | 500 m | 8 us |
| 62,5 kbit/s | 1000 m | 20 us |
| 20 kbit/s | 2500 m | 50 us |
| 10 kbit/s | 5000 m | 100 us |

Tabulka 2.1: Tabulka přenosových rychlostí CAN

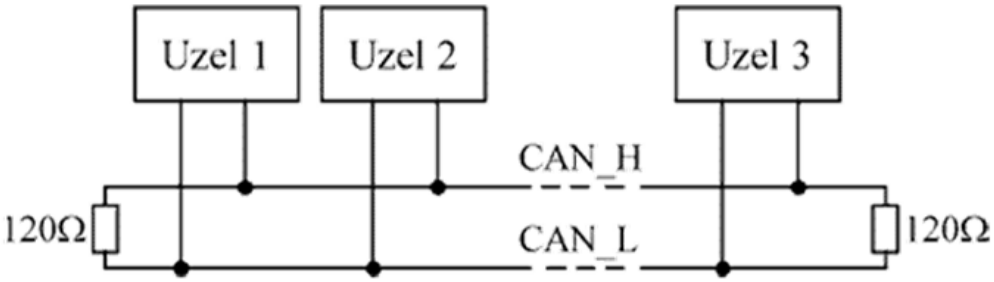
## Kódování bitů

Bity jsou kódovány metodou *NRZ (No Return to Zero)* což znamená, že vysílaná recesivní nebo dominantní úroveň je neměnná po celou dobu vysílání jednoho bitu (na rozdíl např. od kódování Manchaster).

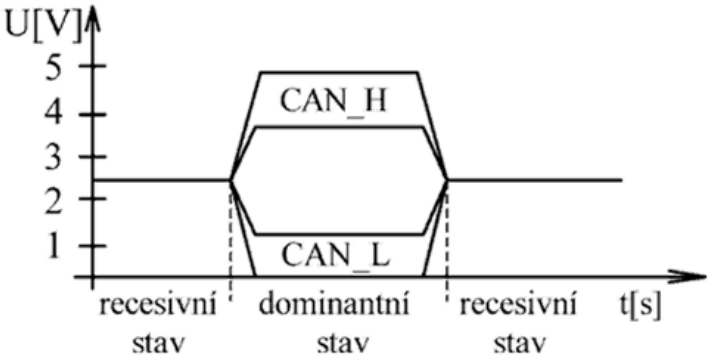
Pokud by na sběrnici bylo vysíláno mnoho logických nul nebo jedniček za sebou, mohlo by dojít k ztrátě synchronizace (chybí hrany). Taková situace je ošetřena technikou zvanou *bitstuffing*. Pokud je vyslaných pět hodnot stejné logické úrovně (pět jedniček nebo nul), tak je přidaný jeden bit opačné hodnoty navíc. Tím se ve vysílání objeví hrana, podle které se mohou zařízení na sběrnici synchronizovat.




Obrázek 2.1: Kódování NRZ a Manchester




Obrázek 2.2: Struktura sítě CAN




Obrázek 2.3: Napěťové úrovně CAN

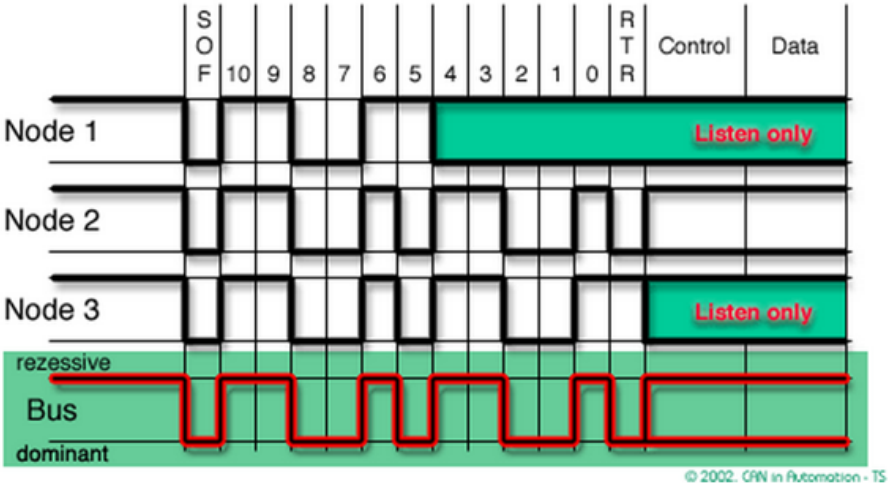
## 2.3 Linková vrstva

Všechny propojené uzly mohou soutěžit o řízení sběrnice a poté vyslat zprávu - sběrnice je typu multimaster. Linková vrstva se dělí na dvě části a to *MAC (Medium Access Control)*

a *LLC (Logical Link Control)*.

### 2.3.1 Vrstva MAC

Tato vrstva má za úkol kódování dat do paketu, řízení priority vstupu na sběrnici pomocí identifikátoru, detekci hlášení chyb a potvrzování správně přijatých dat.




Obrázek 2.4: Příklad arbitrace na sběrnici CAN

### 2.3.2 Vrstva LLC

Tato vrstva má za úkol filtraci zpráv podle identifikátoru (*Acceptance Filtering*), provádění hlášení o přetížení (*Overload Notification*).

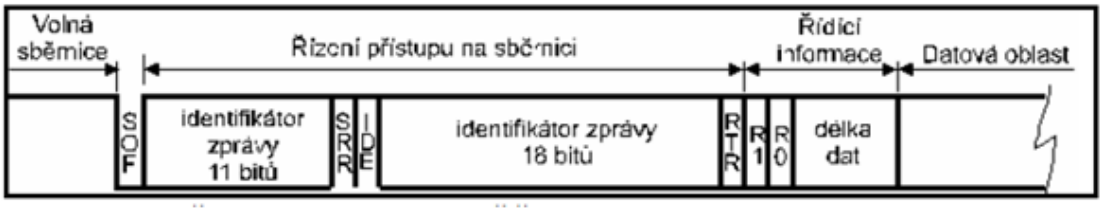
## 2.4 Rámce CAN

## 2.5 Data frame

Tento rámec se používá pro přenos dat mezi jednotlivými uzly. Jednotlivé rámce jsou odděleny sekvencí bitů *Interframe Space*, která má délku minimálně tři bity. Interframe space je okamžik kdy je sběrnice volná a tedy každý uzel může zahájit vysílání odesláním dominantního bitu *Start of Frame*. Maximální počet datových bitů je osm a minimální

nula. Přenos zprávy s nulovým počtem datových bytů je rychlý, protože zpráva má malou délku. Taková zpráva se používá pro rychlé přenosy, kde identifikátor zprávy je zároveň i nositelem informace. Používají se dva typy datových rámců CAN2.0A a CAN2.0B.


Obrázek 2.5: Datový rámec CAN2.0A




Obrázek 2.6: Datový rámec CAN2.0B

## Arbitration field

**Identifikátor zprávy** – 11 bitů, identifikuje význam zprávy, hodnota též využita pro arbitráž.

**RTR (remote request)** - 1 bit, identifikuje, zda se jedná o data nebo datovou žádost. V datové zprávě je tento bit dominantní, v žádosti o data je recesivní.

## Control field

**R0 a R1** – rezervované bity, R0 říká jestli se jedná o *Standart Frame* nebo *Extended Frame*.

**Délka dat** – 4 bity, počet přenášených bajtů, povolené hodnoty jsou 0 – 8.

**Datová oblast** – data, maximálně 8 bajtů, minimálně 0 bajtů, je vysíláno od MSB.

## CRC field

**CRC kód** – kontrolní součet,15 bitů.

**ERC** – oddělovač CRC, 1 bit recesive.

## ACK field

**ACK** – bit potvrzení, jeden bit.
**ACD** – oddělovač potvrzeni, jeden bit.

**Konec zprávy (End Of Frame)** – 7 bit recessive.
**Mezera mezi zprávami (interframe space)** – 3 recessive bity.

## 2.6 Remote request frame

Tento rámec se používá při komunikaci jako žádost o data. Například pokud se jedná o centralizovaný systém, může zařízení, které zpracovává údaje od měřících jednotek s CAN rozhraním, použít tento rámec, aby vyzval ostatní zařízení k přenosu dat. Některé CAN řadiče podporují automatické odeslání dat při přijetí příslušného Remote request frame již na hardwarové úrovni. Tento rámec je stejný jako Data Frame až na hodnotu bitu RTR (Remote Transfer Request), který má u toho rámce recesivní úroveň. Tento rámec neobsahuje část Data Field, avšak hodnota v DLC (Data Lenght Code) může být nenulová, zpravidla v ní je kolik očekáváme datových bytů v odpovědi.

## 2.7 Error frame

Je odeslán uzlem, který detekoval chybu. Rámec se skládá ze dvou částí: Error Flag a Error Delimiter. Error flag je několik po sobě následujících bitů recesivní úrovně ( Error Pasive Flag), které vysílá zařízení v Error Passive módu, nebo dominantní úrovně (Error Active Flag), pokud zařízení, které chybu detekovalo je v Error Active módu. Při vysílání Error Flag není dodržen Bit Stuffing, což detekují ostatní zařízení na sběrnici jako chybu. Po odvysílání prvních šesti bitů odešle na sběrnici jeden bit recesivní úrovně a čeká, až se na sběrnici objeví recesivní úroveň. Poté dokončí přenos Error Frame odesláním sedmi bitů recesivní úrovně.

## 2.8 Overload frame

Tento rámec je odesílán posluchačem, který žádá o prodlevu mezi datovými rámci. Skládá se ze dvou částí – Overload Flag a Overload Delimiter. Overload flag je šest po sobě jdoucích bitů dominantní úrovně. Po odeslání prvních šesti bitů následuje odeslání recesivního bitu a čeká se na hranu z recesivní do dominantní úrovně. Po detekování této hrany odešlou všechna zařízení na sběrnici sedm recesivních bitů. Mezi Data Frame a Remote Frame mohou být maximálně dva Overload Frame.

# Kapitola 3

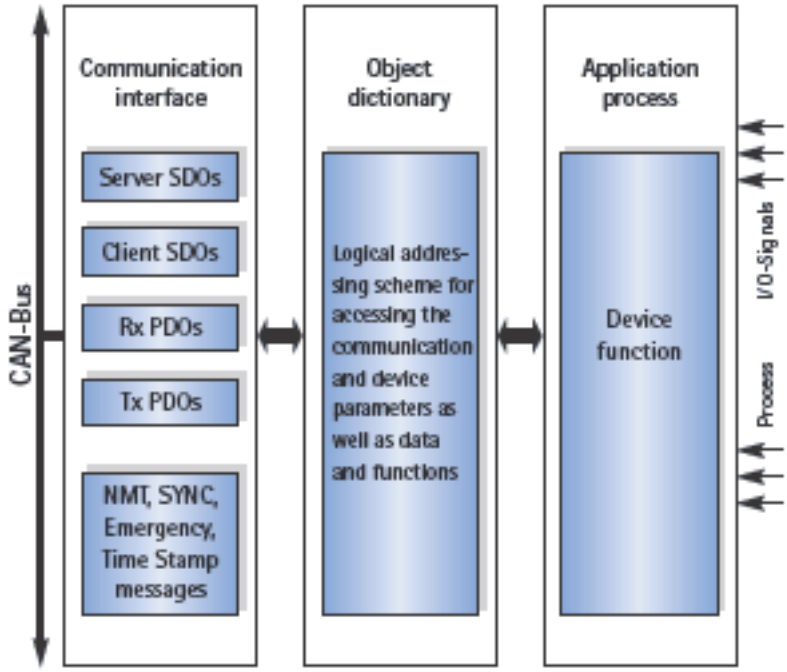
# CANopen

## 3.1 Úvod

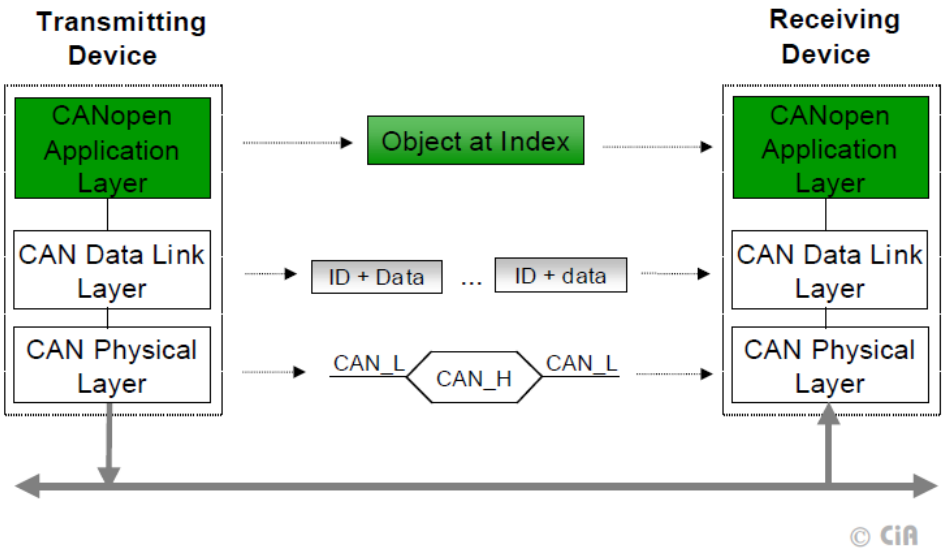
CAnopen je jednou z implementací vyšších komunikačních vrstev nad fyzickou sběrnicí CAN, která definuje pouze fyzickou a linkovou část ISO/OSI modelu.

CANopen definuje vše od síťové vrstvy výše. Na sběrnici CAN jsou však postaveny i jiné protokoly, jako například DeviceNet. CANopen byl nadefinovaný organizací CiA (CAN in Automation). První verze protokolu byla zveřejněna v roce 1994, poslední verze 4.2 byla zveřejněna v roce 2007. CANopen může být postaven i na jiných technologiích, jako je Ethernet Powerlink nebo například EtherCAT.

Aplikační vrstva CANopen a komunikační profil dle CiA DS 301 podporuje přímý přístup k parametrům zařízení a přenos jeho časově kritických procesních dat. Síťový management CANopen výrazně zjednodušuje návrh řídících systému. Poskytuje síťové služby *NMT*, časovou synchronizaci procesů *SYNC* a *TIME STAMP*, zabezpečení provozu *NODE GUARDING*, chybová hlášení *EMERGENCY*, přenos řídících dat *PDO (Proces Data Objects)* a především poskytuje možnost přenosu rozsáhlých servisních nastavení prostřednictvím služby *SDO (Service Data Objects)*. Tyto pojmy budou vysvětleny následně.




Obrázek 3.1: Model CANopen zařízení.




Obrázek 3.2: ISO/OSI model CANopen.

Protokol CANopen definuje všechny komunikačními objekty spolu s nezbytnými informacemi o vlastnostech a funkčních schopnostech jednotlivých zařízení. Komunikační objekty jsou zařazeny v tzv. slovníku objektů *(OD - Object Dictionary)* uloženém v zařízení, které je součástí sítě, a sloužícím jako rozhraní mezi samotným zařízením a

aplikačním programem. Každý komunikační objekt je dostupný prostřednictvím SDO (Service Data Objects) pomocí šestnáctibitového indexu, v případě objektů typu polí a záznamů (objektů složených s několika dalších objektů) doplněného osmibitovým subindexem. Každému komunikačnímu objektu je přiřazen jeden nebo více identifikátorů, které implicitně definují jeho prioritu na sběrnici. Přiřazení identifikátorů jednotlivým komunikačním objektům je jednou ze zásadních otázek při návrhu systému. K usnadnění návrhu jednoduchých sítí definuje protokol CANopen výchozí hodnoty identifikátorů pro všechny povinné objekty. Tyto hodnoty se inicializují v předprovozním stavu sítě, a je-li to nutné, lze je dále dynamicky modifikovat.

| Object | Function code (ID-bits 10-7) | COB-ID | Communication paramteres at OD index |
|---|---|---|---|
| NMT Module Control | 0000 | 000h | - |
| SYNC | 0001 | 080h | 1005h, 1006h, 1007h |
| TIME STAMP | 0010 | 100h | 1012h, 1013h |

Tabulka 3.1: Předdefinované objekty broadcast master-slave komunikace CANopen.

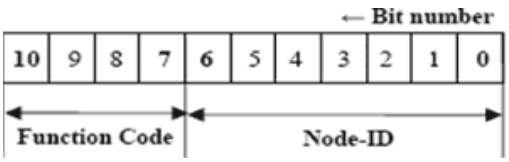
| Object | Function code (ID-bits 10-7) | COB-ID | Communication parameters at OD index |
|---|---|---|---|
| EMERGENCY | 0001 | 081h - 0FFh | 1024h, 1015h |
| PDO 1 (transmit) | 0011 | 181h - 1FFh | 1800h |
| PDO 1 (receive) | 0100 | 201h - 27Fh | 1400h |
| PDO 2 (transmit) | 0101 | 281h - 2FFh | 1801h |
| PDO 2 (receive) | 0110 | 301h - 37Fh | 1401h |
| PDO 3 (transmit) | 0111 | 381h - 3FFh | 1802h |
| PDO 3 (receive) | 1000 | 401h - 47Fh | 1402h |
| PDO 4 (transmit) | 1001 | 481h - 4FFh | 1803h |
| PDO 4 (receive) | 1010 | 501h - 57Fh | 1403h |
| SDO (transmit/server) | 1011 | 581h - 5FFh | 1200h |
| SDO (receive/client) | 1100 | 601h - 67Fh | 1200h |
| NMT Error Control | 1110 | 701h - 77Fh | 1016h, 1017h |

Tabulka 3.2: Předdefinované peer-to-peer objekty master-slave komunikace CANopen.

| Index | Object |
|---|---|
| 0000 | not used |
| 0001 - 001F | Static Data Types (standart data types, e.g Boolean, Integer16) |
| 0020 - 003F | Complex Data Types (preddefined structures composed of standard data types e.g. PDOCommPar, SDOParameter) |
| 0040 - 005F | Manufacturer Specific Complex Data Types |
| 0060 - 007F | Device Profile Specific Static Data Types |
| 0080 - 009F | Device Profile Specific Complex Data Types |
| 00A0 - 0FFF | reserved |
| 1000 - 1FFF | Communication Profile Area (e.g. DeviceType, Error Register, Number of PDOs supported) |
| 2000 - 5FFF | Manufacturer Specific Profile Area |
| 6000 - 9FFF | Standardised Device Profile Area |
| A000 - FFFF | reserved |

Tabulka 3.3: Slovník objektů CANopen (Object dictionary).

Zařízení pracující podle protokolu CANopen smějí používat jen identifikátory odpovídající komunikačním objektům podporovaným protokolem. Implicitní schéma přiřazení identifikátorů má funkční část, určující zásadně prioritu objektu, a část označenou jako Node-ID, která umožňuje rozlišovat mezi dvěma zařízeními plnícími stejnou funkci. Nastavení Node-ID je obvykle provedeno hardwarovým přepínačem. Rozsah Node-ID je 1 až 127. V případě nulové hodnoty Node-ID je zpráva adresována všem uzlům (All-Node ID). Identifikátor jednotlivých komunikačních objektů je označen zkratkou COB-ID (Communication Object ID).




Obrázek 3.3: COB-ID pro 11 bitový CAN.

| Value | Type |
|---|---|
| 0001 | Boolean |
| 0002 | Integer8 |
| 0003 | Integer16 |
| 0004 | Integer32 |
| 0005 | Unsigned8 |
| 0006 | Unsigned16 |
| 0007 | Unsigned32 |
| 0008 | Real32 |
| 0009 | Visible_String |
| 000A | Octet_String |
| 000B | Unicode_String |
| 000C | Time_Of_Day |
| 000D | Time_Difference |
| 000E | Bit_String |
| 000F | Domain |
| 0010 | Ineger24 |
| 0011 | Real64 |
| 0012 | Integer40 |
| 0013 | Integer48 |
| 0014 | Integer56 |
| 0015 | Integer64 |
| 0016 | Unsigned24 |
| 0017 | reserved |
| 0018 | Unsigned40 |
| 0019 | Unsigned48 |
| 001A | Unsigned56 |
| 001B | Unsigned64 |
| 001C - 001F | reserved |

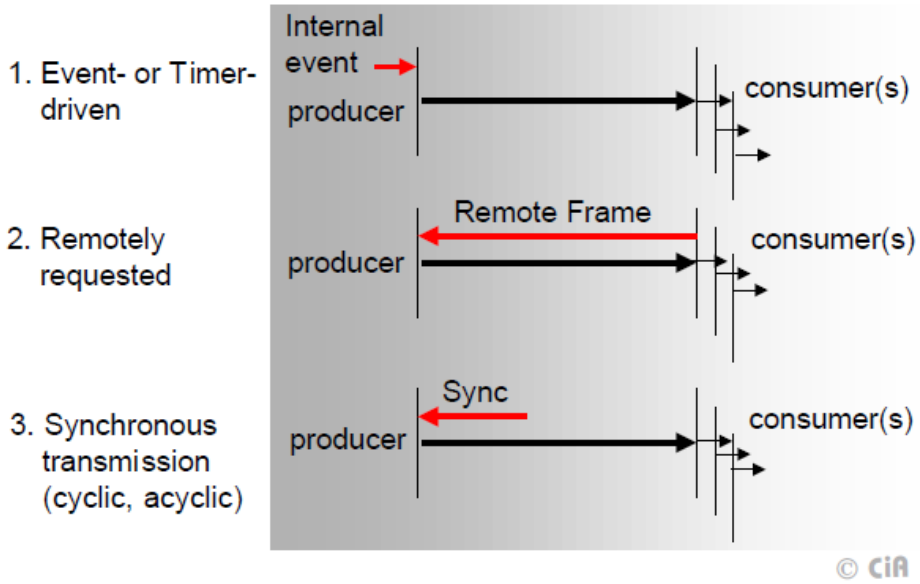
Tabulka 3.4: Tabulka datových typů.

## 3.2 Proces Data Objects - PDO

Zprávy PDO přenášejí technologická data. Každý typ PDO zprávy musí mít unikátní identifikátor CAN a může být vysílán pouze jediným uzlem sítě, přičemž přijat může být libovolným počtem zařízení. Vyslání zprávy s PDO může být inicializováno vnitřní událostí, vnitřním časovačem, požadavky vznesenými jinými zařízeními v síti nebo přijetím synchronizační zprávy. PDO zprávy jsou především používány pro přenos dat v reálném čase a typicky mají vyšší prioritu pro přístup na sběrnici CAN než objekty SDO. PDO zpráva je maximalně 8 bytů dlouhá.

PDO zpráva není po přijetí zařízením potvrzované zprávou s odpovědí. Naopak vlastní frame je potvrzovaný na úrovni linkové vrstvy.

Počet a délka PDO zpráv je definovaná v profilu zařízení.




Obrázek 3.4: PDO komunikace.

## 3.3 Service Data Object - SDO

SDO protokol slouží k zápisu a čtení hodnot z objektového slovníku vzdáleného zařízení. Zařízení, ke kterému je přistupováno, ze označuje SDO server, zařízení, které přistupuje k SDO serveru, se označuje SDO klient. Komunikace je vždy zahájena SDO klientem. Pokud je zpráva delší než 8 bajtů, SDO protokol umožňuje segmentaci zapisovaných dat

a skládání zpráv tvořících odpověď. Pokud je dat více než 4 bajty, musí být použita segmentace. Máme dva typy SDO komunikace: SDO přenos dat a SDO blokový přenos dat, který je vhodný pro přenos větších objemů dat. SDO zprávy jsou využívány zejména pro dlouhé přenosy dat s nízkou prioritou přístupu na sběrnici a jsou vysílány asynchronně. Jejich normální využití je pro konfiguraci uzlů. CANopen specifikuje, že každý uzel na síti musí mít nejméně jeden komunikačn objekt.

| Byte 0 | Byte 1-2 | Byte 3 | Byte 3-7 |
|---|---|---|---|
| SDO command specifier | Object Index | Object Subindex | ** |

Tabulka 3.5: Formát SDO zprávy.

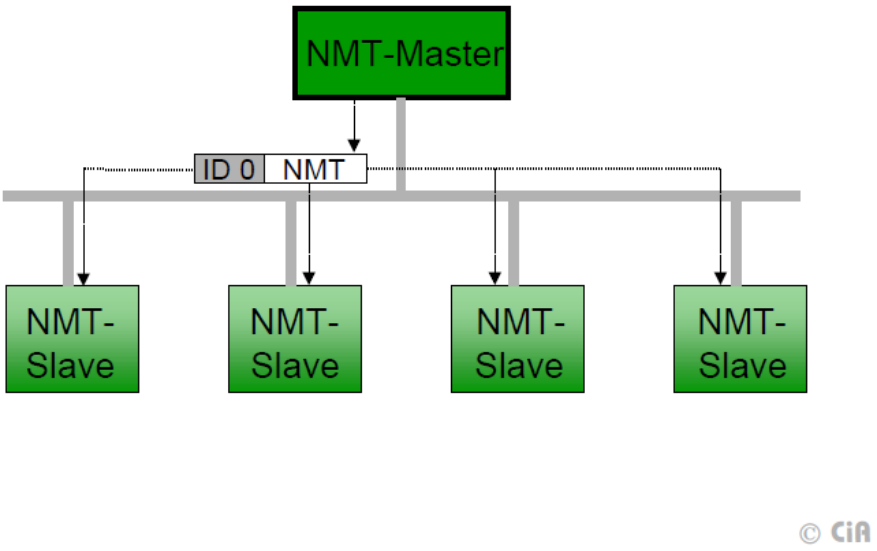
| Byte 0 | Byte 1-7 |
|---|---|
| SDO command specifier (segmented transfer) | up to 7 bytes of data |

Tabulka 3.6: Formát SDO segmentované zprávy.

SDO command specifier obsahuje bit definující zda se jedná o upload či download, dále bit, který definuje jestli je to žádost či odpověď. Dále je tu bit říkající, zda se jedna o nesegmentovaný či segmentování přenos. V bajtu je obsažen počet bytů zprávy. Nakonec je tam alternující bit pro každý segment zprávy.

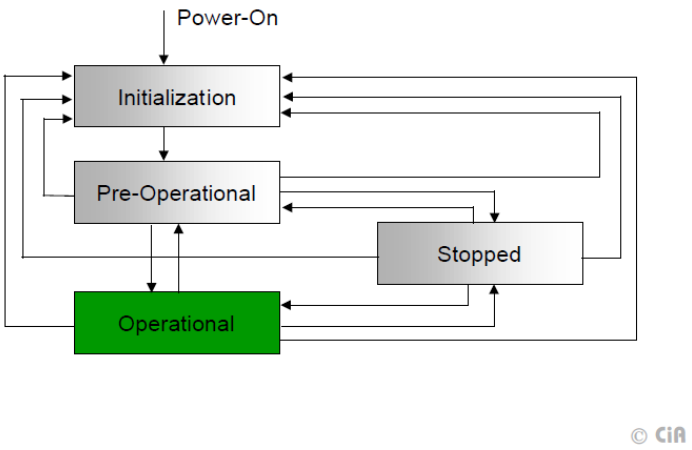
## 3.4 NMT - Network Management

CANopen network management funguje na principu master-slave komunikace. V síti musí být vždy jeden NMT master, ostaní uzly jsou NMT slaves. NMT slave je identifikován ID svého uzlu, které má v síti unikátní.




Obrázek 3.5: Schéma funkce NMT.

CANopen slave v sobě implentuje stavový automat, který hned po přípojení napájení přechází do stavu *Initialization* a následně do stavu *Pre-Operational* po odeslání *Boot-up* objektu. V tomto stavu může zařízení komunikovat pomocí SDO objektů, PDO komunikace je nepovolená. NMT master může přepnout jednotlivá slave zařízení nebo všechna najednou do stavu *Operational*. Ve stavu *Operational* je PDO komunikace povolená. Ve stavu *Stopped* nesmí zařízení komunikovat ani PDO ani SDO objekty.




Obrázek 3.6: Stavový model CANopen zařízení.

NMT Control Object poskytuje prostředky pro řízení stavu podřízených zařízení v CAN síti. Zpráva má díky nulovému identifikátoru (COB-ID) nejvyšší prioritu. Zpráva

se skládá ze dvou byte z nichž první Byte CS (Command Specifier) obsahuje příkaz a druhý Byte jednoznačnou identifikaci uzlu dle Node-ID.

| COB-ID | Byte 0 | Byte 1 |
|---|---|---|
| 0x00 | CS | node-id |

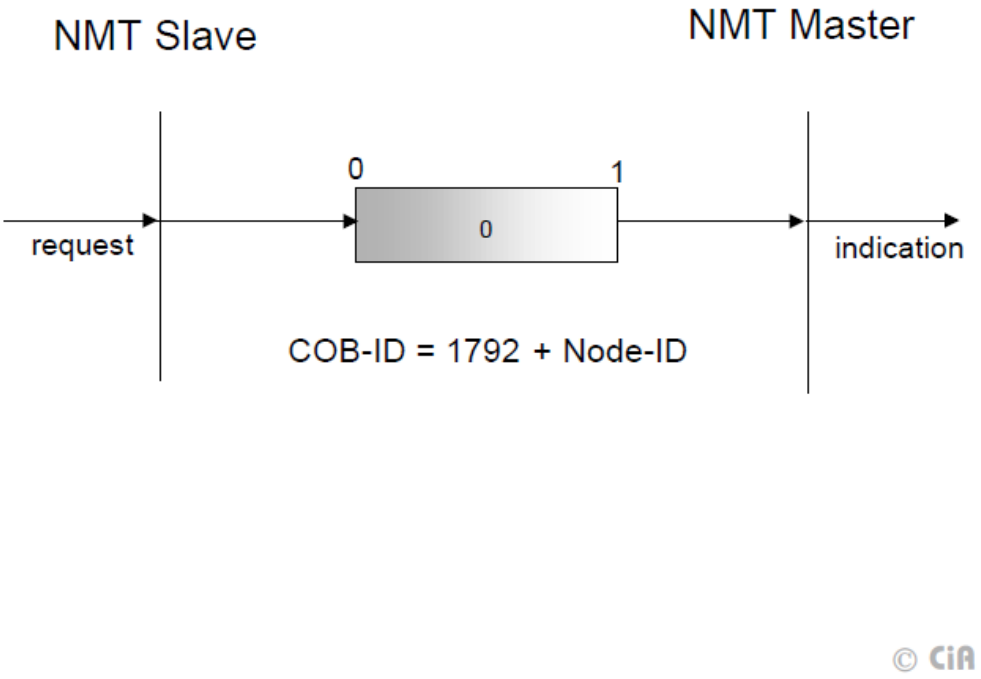
Tabulka 3.7: Formát NMT paketu.

| Command Specifier | NMT Service |
|---|---|
| 1 | Start Remote node |
| 2 | Stop Remote Node |
| 128 | Enter Pre-operational State |
| 129 | Reset Node |
| 130 | Reset Communication |

Tabulka 3.8: Command specifikátory.

NMT poskytuje pět příkazů, které jsou dány bajtem *CS (Command Specifier)*. Node-ID definuje zařízení, pro které je zpráva určena. Pokud Node-ID je nula, je zpráva určena pro všechna zařízení na sběrnici.

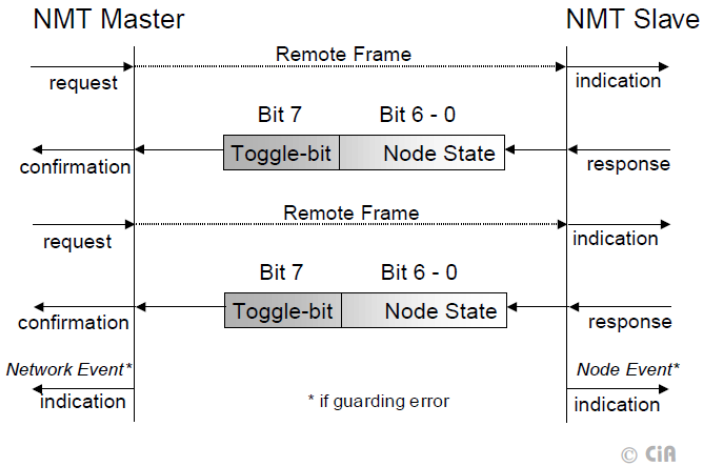
Každý NMT slave před přechodem do stavu Pre-Opeerational vysílá zprávu Boot-up. Tato zprává má jeden datový bajt, který je nulový. Tato zpráva je odeslána i po resetu komunikace.




Obrázek 3.7: Boot-up objekt.

## Node Guarding protkol

Pro detekci výpadku zařízení, které neposílájí pravidelně PDO zprávy slouží mechanismus jménem *Node Guarding* protkol. NMT master si udržuje databázi aktivních slave zařízení a pravidleně jim zasílá Node Guarding paket, na který zařízení odpovídají. Zařízení si zároveň hlídají pravidelnost příchozích Node Gurading paketů a v případě výpadku detekují ztrátu spojení. Node Guarding protokol je započat NMT masterem v Pre-Operational stavu, ve stavu Stopped je taktéž aktivní.




Obrázek 3.8: Node Guard protokol.

| COB-ID |
|---|
| 0x700+Node_ID |

Tabulka 3.9: Node Guartding request paket.

| COB-ID | Byte 0 |
|---|---|
| 0x700+Nod_ID | bit 7: toggle, bit 6-0 device state |

Tabulka 3.10: Node Guartding response paket.

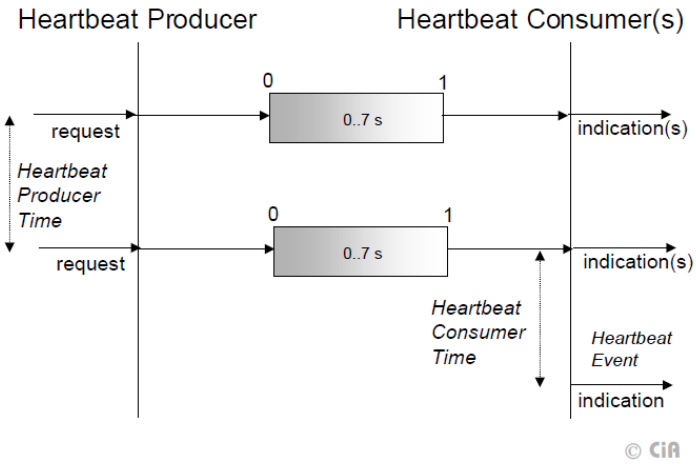
| State | Meaning |
|---|---|
| 0 | Boot-up |
| 3 or 127 | Pre-operational |
| 4 | Stopped |
| 5 | Operational |

Tabulka 3.11: Stavy uzlů v CANopen síti.

## Heartbeat protokol

Heartbeat producent pravidelně vysíla heartbeat paket. Konzument kontroluje pravidelné přijetí tohoto paketu. Pokud paket nepříjde v daném časovém okně, vygeneruje chybovou

událost. Tento protokol je volitelný. Uzel nemůže používat najednou Heartbeat protokol a Node guarding protokol.




Obrázek 3.9: Heartbeat protokol.

| COB-ID | Byte 0 |
|---|---|
| 0x700+NODE-ID | state |

Tabulka 3.12: Formát heartbeat zprávy.

## Emergency

Emergency zpráva se vysílá pokud dojde uvnitř zařízení k nějaké fatalní chybě. Tato zpráva má vysokou prioritu.

| COB-ID | Byte 0-1 | Byte 2 | Byte 3-7 |
|---|---|---|---|
| 0x080+NODE-ID | Emergency error code | Error register (Object 0x1001) | Manufacturer sepcific error field |

Tabulka 3.13: Formát emergency zprávy.

| Emergency error code | Meaning |
|---|---|
| 00xx | Error Reset or No Error |
| 10xx | Generic Error |
| 20xx | Current |
| 21xx | current, device input side |
| 22xx | current, inside the device |
| 23xx | current, device output side |
| 30xx | Voltage |
| 31xx | mains voltage |
| 32xx | voltage inside the device |
| 33xx | output voltage |
| 40xx | Temperature |
| 41xx | ambient temperature |
| 42xx | device temperature |
| 50xx | Device hardware |
| 60xx | Device software |
| 61xx | internal software |
| 62xx | user software |
| 63xx | data set |
| 70xx | Additional modules |
| 80xx | Monitoring |
| 81xx | Communication |
| 8110 | CAN overrun |
| 8120 | Error Passive |
| 8130 | Life Guard Error or Heartbeat Error |
| 8140 | Recovered from Bus-Off |
| 82xx | Protocol Error |
| 8210 | PDO not processed due to length error |
| 8220 | Length exceeded |
| 90xx | External error |
| F0xx | Additional functions |
| FFxx | Device specific |

Tabulka 3.14: Emergency error codes.

| Bit | |
|---|---|
| 0 | generic |
| 1 | current |
| 2 | voltage |
| 3 | temperature |
| 4 | communication |
| 5 | device profile specicific |
| 6 | reserved (=0) |
| 7 | manufacturer specific |

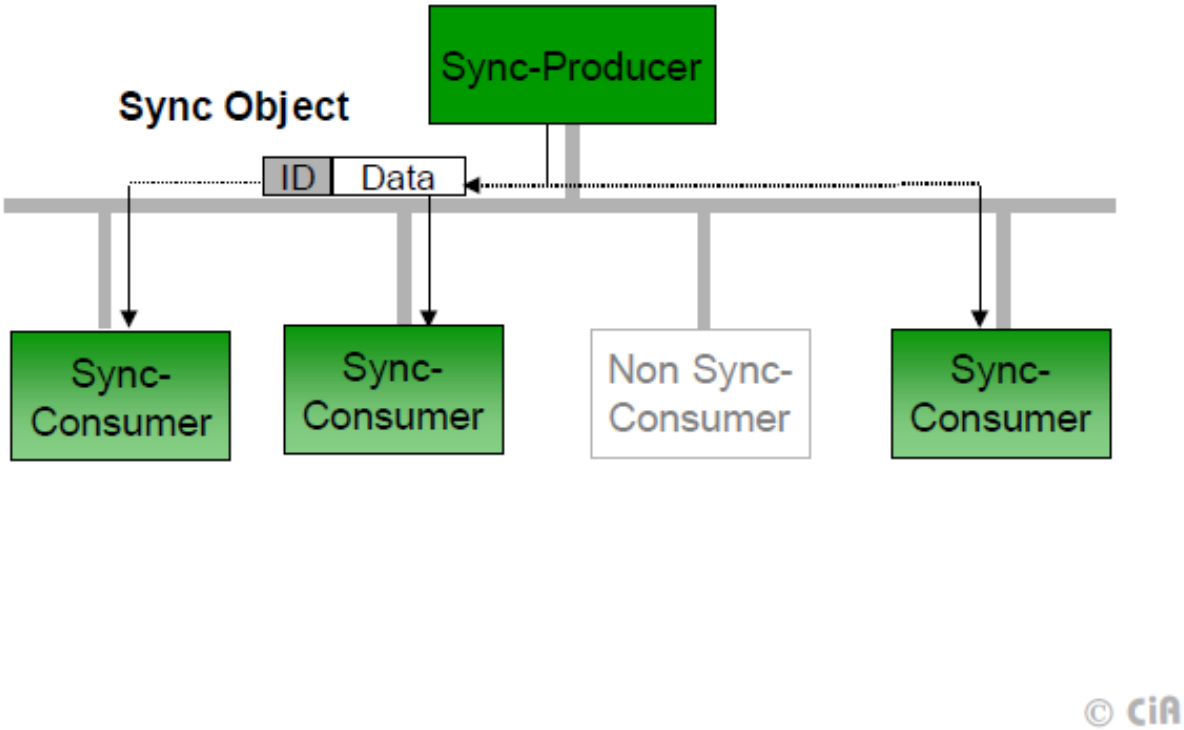
Tabulka 3.15: Popis bitů error registeru - objekt 0x1001.

## SYNC - synchronization object

Používá se k synchronizace celé CANopen sítě. V síti je vždy jeden producent, který v pravidlených intervalech vysílá SYNC zprávu.

SYNC objekt je v Object Dictionary je konfigurovaný položkou na indexu 0x1006.

SYNC zpráva má vysokou prioritu a žádný datový obsah, zkládá se z jedné CAN zprávy, tímto je dosahováno vysoké přesnosti synchronizace. Perioda SYNC zpráv je ovlivněna jitterem z důsledku odesílání předchozích zpráv nebo odesílání zpráv s vyšší prioritou.




Obrázek 3.10: SYNC protokol.

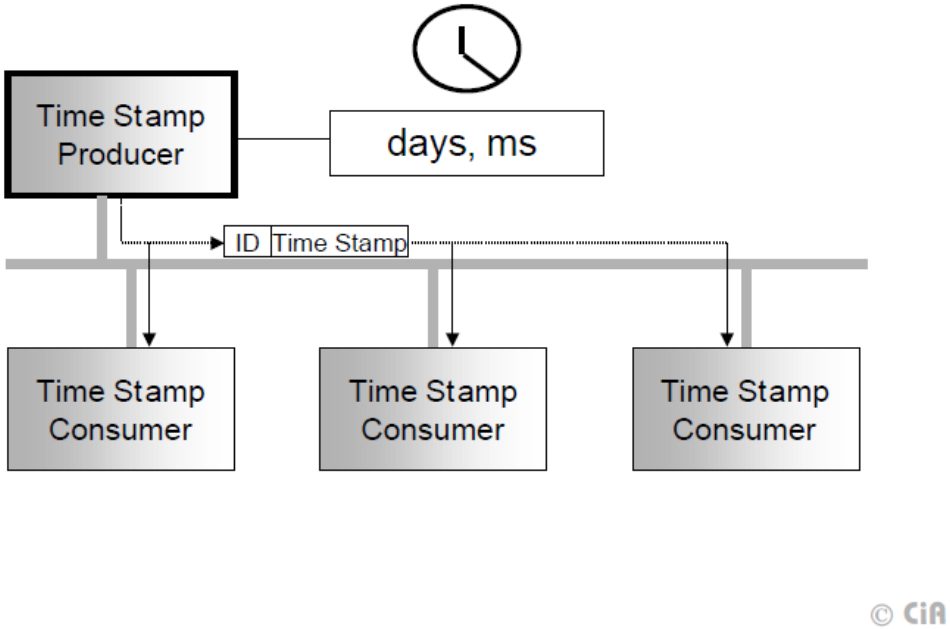
| COB-ID |
|---|
| 0x080 |

Tabulka 3.16: Formát SYNC zprávy.

## Time Stamp

| COB-ID | Byte 0-5 |
|---|---|
| 0x100 | unsigned28(ms from midnight)+unsigned16(day from 1.1.1984) |

Tabulka 3.17: Formát Time Stamp zprávy.




Obrázek 3.11: Time Stamp protokol.

## EDS soubor

Objektový slovník zařízení je configurovaný pomocí souboru EDS (Electronic Data Sheet). Podle CiA306 má tento soubor formát INI souboru. Výrobce je povinen tento soubor přiložit k výrobku pro CANopen síť. Soubor smí používat pouze ASCII znaky, řádek se ukončuje znaky LF nebo CR LF, řádek nesmí mít více jak 255 znaků. Obsahuje informace týkající se souboru, dále informace o zařízení a jeho objektový slovník. Soubor musí dodržovat přísnou syntaxi. Mohou se vyskytovat prázdné řádky.

Prvky EDS souboru:

a) Vlastnost

`name=value`

Poznámka: Levá strana je povinná. Pravá povinná není. Po obou strnách rovnítka nesmí být mezera. Name není case sensitive. Identifikátor Name je vždy brán jako řetězec ne jako číslice. Value může být řetězec, nebo integer v dekadickém, hexadxecimálním nebo oktalovém formátu.

b) Sekce

`[nazev_sekce]`

Poznámka: Mimo závorky se nesmí nacházet žádný znak. Název sekce není case sensitive.

c) Komentář

`; comment text`

Poznámka: Komentář musí být na samostatném řádku.

EDS soubor standartně obsahuje sekci `[FileInfo]`. Zahrnuje tyto vlastnosti.

`FileName` - file name (according to OS restrictions).
`FileVersion` - actual file version (Unsigned8).
`FileRevision` - actual file revision (Unsigned8).
`EDSVersion` - Version of Specification (3 characters) in the Format x.y. If the entry is missing, this is equal to 3.0.
`Descriptio` - file description (max 243 characters).
`CreationTime` - file creation time (characters in format hh:mm(AM,PM)).
`CreationDate` - date of file creation (characters in format mm-dd-yyyy).
`CreatedBy name` - or description of file creator (max 245 characters.)
`ModificationTime` - time of last modification (characters in format hh:mm(AM,PM)).
`ModificationDate` - date of last file modification (characters in format mm-dd-yyyy).
`ModifiedBy` - name or description of the modification (max 244 characters).

Příklad:

```
[FileInfo]
FileName=vendor1.eds
FileVersion=1
FileRevision=2
EDSVersion=4.0
Description=EDS for simple I/O-device
```

```
CreationTime=09:45AM
CreationDate=05-15-1995
CreatedBy=Zaphod Beeblebrox
ModificationTime=11:30PM
ModificationDate=08-21-1995
ModifiedBy=Zaphod Beeblebrox
```

Další sekce se nazývá `[DeviceInfo]` a dává nám základní informace o zařízení, která EDS popisuje. obsahuje tyto vlstnosti:

`VendorName` - vendor name (max 244 characters).
`VendorNumber` - unique vendor ID according to identity object sub 1 (Unsigned32).
`ProductName` - product name (max 243 characters).
`ProductNumber` - product code according to identity object sub 2 (Unsigned32).
`RevisionNumber` - product revision number according to identity object sub 3 (Unsigned32).
`OrderCode` - order code for this product (max 245 characters).
`BaudRate_10` - supported baud rates (Boolean, 0 = not supported, 1 = supported).
`BaudRate_20`
`BaudRate_50`
`BaudRate_125`
`BaudRate_250`
`BaudRate_500`
`BaudRate_800`
`BaudRate_1000`
`SimpleBootUpMaster` - simple boot-up master functionality (Boolean, 0 = not supported, 1 = supported).
`SimpleBootUpSlave` - simple boot-up slave functionality (Boolean, 0 = not supported, 1 = supported).
`Granularity` this value gives you the granularity allowed for the mapping on this device - most of the existing devices support a granularity of 8 (Unsigned8; 0 - mapping not modifiable, 1-64 granularity).
`DynamicChannelsSupported` - according to DS-302 this entry describes the facility of dynamic variable generation. If value is not 0, the additional section DynamicChannels exists. Details are given in CiA DS-302 and CiA DS-405.

`GroupMessaging` - according to DS-301 Annex A this entry describes the facility of multiplexed PDOs. (Boolean, 0 = not supported, 1 = supported) Details are given in DS-301.
`NrOfRXPDO` - number of Receive PDOs supported. (Unsigned8)
`NrOfTXPDO` - number of Transmit PDOs supported. (Unsigned8)
`LSS_Supported` - support of LSS functionality (Boolean, 0 = not supported, 1 = supported).

Příklad DeviceInfo:

```
[DeviceInfo]
VendorName=Nepp Ltd.
VendorNumber=156678
ProductName=E/A 64
ProductNumber=45570
RevisionNumber=1
OrderCode=BUY ME - 177/65/0815
LSS_Supported=0
BaudRate_50=1
BaudRate_250=1
BaudRate_500=1
BaudRate_1000=1
SimpleBootUpSlave=1
SimpleBootUpMaster=0
NrOfRxPdo=1
NrOfTxPdo=2
```

V následující sekci EDS souboru je popsaný Objektový slovník přístupný v zařízení. Obsahuje seznam všech objektů, které zařízení podporuje, jejich limitní paramtery, defaultní hodnoty, datové typy a další informace o nich. objekty se dělí do tří skupin - Mandatory objects, Optional objects a Manufacturer specific objects.

Mandatory objects obsahuji minimálně 100h a 1001h. Pro CANopen verzi 4.0 je tu ještě objekt 1018h. Optional objects obsahují objekty s indexy 1000h až 1FFFh a 6000h až FFFFh. Manufacturer objects jsou objekty od 2000h do 5FFFh. Každá sekce obsahuje záznam `SupportedObjects`, což je číselný údaj o počtu objektů v sekci. Následně jsou všechny vypsány. Příklad:

```
[OptionalObjects]
SupportedObjects=10
1=0x1003
2=0x1004
3=0x1005
4=0x1008
5=0x1009
6=0x100A
7=0x100C
8=0x100D
9=0x1010
10=0x1011
```

Všechny objekty jsou popisovány podle jednoho schématu.

`SubNumber` - počet subindexů k tomuto indexu, pokud subobjekty nejsou tento parametr není uveden nebo není vyplněn.
`ParameterName` - jméno parametru až 241 znaků.
`DataType` - index datového typu v objektovém slovníku.
`LowLimit` - spodní limit hodnoty pokud se dá aplikovat.
`HighLimit` - horní limit hodnoty pokud se dá aplikovat.
`AccessType` - ro - readonly, wo - write only, rw - read/write, const - konstanta.
`DefaultValue` - výchozí hodnota.
`PDOMapping` - logická hodnota říkající jestli se dá objekt mapovat do zprávy PDO (0 - nemapovatelné, 1 - mapovatelné).
`ObjFlags` - vlitelné pole pro speciální účely.

Příklad:

```
[1000]
ParameterName=Device Type
ObjectType=0x7
DataType=0x0007
AccessType=ro
```

```
DefaultValue=
PDOMapping=0

[1003]
SubNumber=2
ParameterName=Pre-defined Error Field
ObjectType=8

[1003sub0]
ParameterName=Number of Errors
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=0x1
PDOMapping=0

[1003sub1]
ParameterName=Standard Error Field
ObjectType=0x7
DataType=0x0007
AccessType=ro
DefaultValue=0x0
PDOMapping=0
```

Sekce komenářů může být přítomna v EDS souboru, má tento formát:

```
[Comments]
Lines=3
Line1=|-------------|
Line2=| Don't panic |
Line3=|-------------|
```

DCF soubor neboli Device Configuration File obsahuje informace o konkrétním jednom zařízení. Obsahuje navíc ještě informace o přenosové rychlosti a NODE-ID. Je taktéž definovaný CANopen.

# Kapitola 4

# Technologie pro tvorbu GUI

Jádrem této práce bylo vytvořit grafické uživatelské rozhranní pro práci s EDS soubory a komunikaci po CANopen, a proto bylo nutné vybrat vhodný grafický framework. Uvažoval jsem o těchto možnostech:

- .NET Framework
- GTK+
- Java Swing
- Qt

V následujících sekcích si jednotlivé technologie rozebereme.

## 4.1 .NET Framework

Tento framework byl vytvořen firmou Microsoft v roce 2002. Dnes je již používaná čtvrtá verze tohoto produktu. Zároveň s .NET Frameworkem bylo vydáno i integrované vývojové prostředí s názvem Visual Studio. Tento framework slouží výhradně k chodu pod operačním systémem Windows. To je velká nevýhoda toho frameworku pro tuto práci. Jelikož tato práce je koncipována jako OpenSource, je tento framework zcela nepoužitelný pro naši aplikaci. .NET je však velmi výkonný framework nejen pro tvorbu grafických aplikací. Framework poskytuje moduly pro operace se soubory, kreslení grafiky, interakci

s databázemi, manipulaci s XML dokumenty, síťové služby a mnoho dalších služeb. Podporuje mnoho programovacích jazyků jako C#, VB.NET, F#, Delphi, J#,IronPython atd.

## 4.2 GTK+

GTK+ je multiplatformní nástroj na vytváření grafických aplikací. Nabízí velké množství komponent k použití. Je to volný software šířený pod LGPL licencí a je součástí GNU projektu. GTK podporuje nejenom C a C++ jazyky ale i Perl, Python a mnoho dalších. V dnešní době se GTK používá v mnoha aplikacích, jako na je například desktopové prostředí GNOME. GTK je založeno na jazyce C stejně jako drivery použité v naší aplikaci, takže by je šlo velmi jednoduše napojit na grafické komponenty. Takový software by pracoval velmi rychle a pravděpodobně by nebyly ani problémy s portabilitou. Nevýhodou je ale neobjektový přístup, díky čemuž je GTK velmi složité na použití. Další nevýhodou GTK je velmi slabá dokumentace a ne moc velká komunita kolem něj. Vývoj s GTK by byl velmi pomalý a neefektivní, a proto jsem se rozhodl GTK+ nepoužít.

## 4.3 Java Swing

Swing je knihovna uživatelských prvků na platformě Java pro ovládání počítače pomocí grafického rozhraní. Je to součást JFC (Java Foundation Classes) od firmy SUN. Tento toolkit je stejně jako jazyk Java multiplatformní.

Další informace byste našli na stránkách projektu Swing. Tento toolkit má velmi kvalitní dokumentaci a je okolo něj široká komunita. Dokonce projekt Ortcan má napsanou v Jave velmi jednoduchou aplikaci na monitorování CAN sběrnice a odesílání zpráv.

## 4.4 Qt

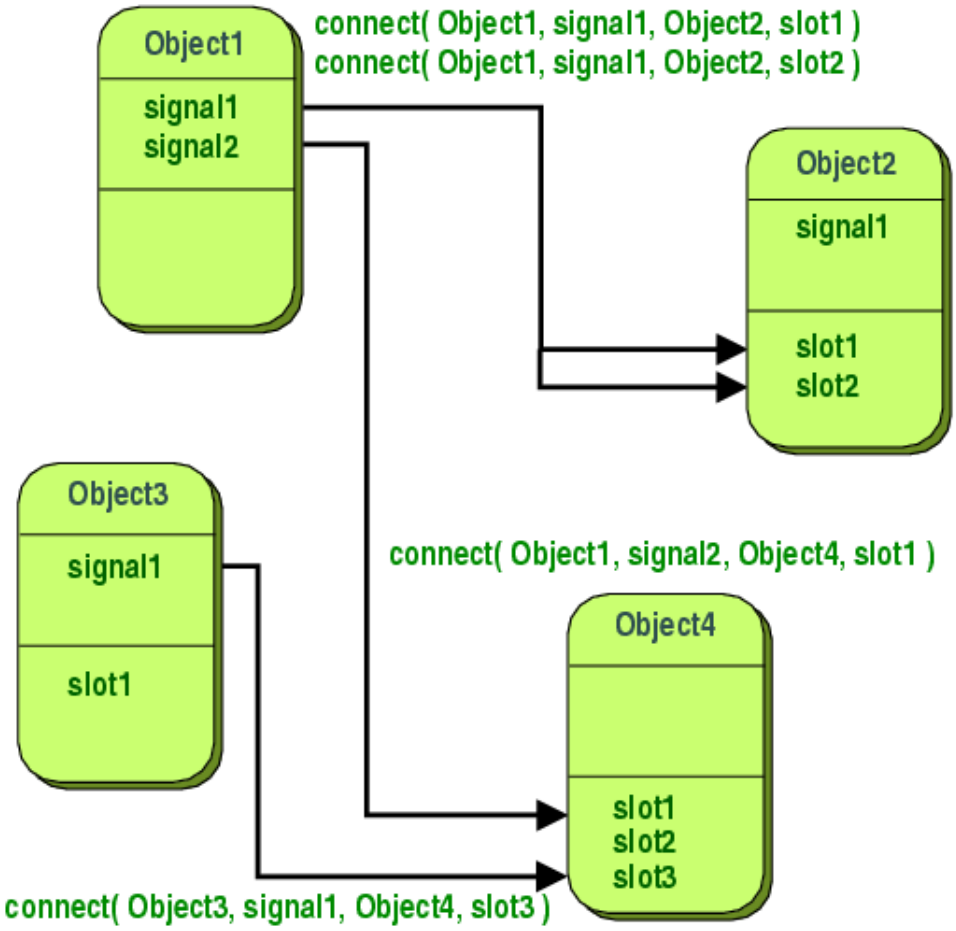

Obrázek 4.1: Logo Qt.

Knihovna Qt je kompletní framework nabízející nejenom grafické služby. Byl vydán firmou TrollTech. Je to multiplatformní vysoce výkoný framework. Nabízí nástroje qmake, Qt Linguist, Qt Designer a Qt Assistant. Využívá několik nestandartních C++ rozšíření. Existují k němu vazby pro jazyky Ada, C++, C#, D, Haskell, Harbour, Java, Lisp, Lua, Pascal, Perl, Python, PHP, QML, R, Ruby, Scheme, TCL atd.. Zahrnuje tyto moduly QtCore, QtGui, QtMultimedia, QtNetwork, QtOpenGL, QtOpenVG, QtScript, QtScriptTools, QtSql, QtSvg, QtWebKit, QtXml, QtXmlPatterns, Qt3Support. Qt je použitelné na těchto operačních systémech Embedded Linux, Mac OS X, Microsoft Windows, X11, Wayland, Windows CE, Symbian, MeeGo, Haiku a Amiga OS. Trolltech nabízí GPL, LGPL i komerční licence. Qt se stále rozvíjí a komunita užívající Qt se stále rozšiřuje. Další informace mohou být nalezeny na stránkách Qt. Obecně Qt je o něco pomalejší než GTK kvůli overheadu daným objektovým přístupem, ale zase je rychlejší než Java nebo .NET Framework. Qt má velmi povedenou a obsáhlou dokumentaci s monoho praktickými ukázkami použití jednotlivých tříd. Některé Qt widgety podporují Model-View architekturu.

## 4.5 Signal slot koncept

Signál slot slouží ke komunikaci mezi objekty. Je to koncept je jazyková konstrukce v Qt, která slouží k velmi jednoduché implementaci observer patternu. Tímto se Qt framework

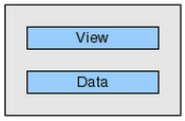
velmi odlišuje od ostatních frameworků. Spočívá v tom, že komponenty (widgety) mohou posílat signály obsahující další informace (parametry), které mohou být přijaté dalšími komponentami na vyžádání pomocí takzvaných slotů. Tento systém se velmi hodí pro grafická rozhranní. Signály a sloty mohou být použité i pro asynchronní operace (sokety, roury, sériová zařízení). Pokud chce třída využívat signal slot mechanismus musí dědit vlastnosti os třídy QObject. Jeden slot může příjmat i více signálů. V jiných frameworcích se používají callbacky. Callback je pointer na funkci. Callbacky mají jeden problém - nejsou typově bezpečné.




Obrázek 4.2: Signal slot.

## 4.6 Model View architektura

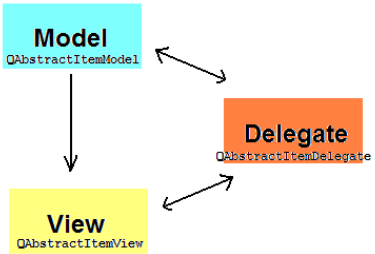
V této práci bylo využito model-view architektury v EDS editoru, a tak zde je krátké pojednání o ní. Tabulka, seznam a strom jsou grafické komponenty často používané v grafických aplikacích. Existují dva druhy přístupu k nim. Tradiční přístup je mít datové prvky uložené přímo v komponentě. Tento přístup je velmi intuitivní, ale v netriviálních aplikacích může způsobit problémy se synchronizací dat. Druhý způsob je využití model-view architektury. Komponenty přistupují k datům pomocí standartizovaného rozhranní. Tento způsob se může zdát složitější, ale ve skutečnosti je mnohem čistší, a přináší mnoho výhod. Separování dat od zobrazení brání datové duplikaci. Jeden model se dá použít pro více zobrazení a vice versa. QT framework proto nabízí dva druhy komponent - pro standartní přístup a pro model-view.




Obrázek 4.3: Použití standartní komponenty.


Obrázek 4.4: Použití model view komponenty.

Základem model-view architektury je oddělit data (model) od zobrazení (view).




Obrázek 4.5: Schéma model-view architektury.

Model - poskytuje data, buď uložená přímo v modelu, nebo warpuje skrze interface z jiné třídy
Delegate - pomáhá view vykreslovat data z modelu a zařizuje editaci modelu.
View - zobrazuje data na obrazovce a žádá delegate na vykreslení nebo editaci dat v modelu

Qt framework poskytuje tyto view: QListView, QTableView a QTreeView a jejich standartní verze QListWidget, QTableWidget a QTreeWidget.

## 4.7 Závěr

Rozhodl jsem se pro technologii Qt a pokračovat tak v práci na již vyvýjeném QCANalyseru. QCANalyser má již vytvořené napojení na CAN drivery, takže implementace CANopen bude o to jednodušší.

# Kapitola 5

# Přidružené projekty

## 5.1 Ortcan

Ortcan je soubor softwarových nástrojů týkajících se CANu vyvinutý na Českém Vysokém Učení Technickém v Praze. Vývoj těchto komponent začal v roce 2001 v rámci evropského projektu OCERA. Komponenty již byly využity po celém světě na různých akademických a produkčních projektech. Komponenty jsou udržovány především zaměstnanci Real-Time skupiny na oddělení DCE na fakultě Elektrotechnické na ČVUT v Praze. Na jejich stránkách je mnoho souvisejících projektů, benchmarků a diplomových práci týkajících se Ortcanu.

K přístupu na CAN sběrnici je potřeba hardware doplněný o driver. Projekt Ortcan poskytujej driver pro Linux a ostatní vestavěné systémy k získání přístupu na sběrnici. Driver se jmenuje LinCAN.

Společné rozhranní k přístupu k různým CAN rozhranním je implementováno v Virtual/Versatile CAN API knihovně neboli - libVCA.

Ortcan dále nabízí svě grafická rozhranní a monitoring a interakci s CAN sběrnicí. první je Javovský projekt jménem CANmonitor, který umí nejen CAN, ale i CANopen. Druhý se jmenuje QCANalyser a je postavený na Qt. Ten umožňuje pouze monitoring a interakci s CAN sběrnicí. Mojí prací bylo ho doplniy ještě o CANopen.

Ortcan obsahuje jeho mnoho dalších utilit, ale o těch bude napsáno v sekci Testování.

## 5.2 CanFestival

CanFestival je CANopen framework licencovaný licencí GPLv2 a LGPLv2. Poskytuje nezávislou ANSI-C CANopen platformu, která umožňuje implementovat CANopen slave nebo master zařízení do PC, Real-Time řídicích systémů a mikrokontrolérů. S CanFestivalem můžete:

- jakékoliv PC nebo mikrokontrolér změnit na CANopen zařízení
- druha itemize editovat objektový slovník a EDS soubory

CanFestival podporuje CANopen protokoly: PDO, SDO, NMT, SYNC, Node Guarding, Live Guarding, Heartbeat a Bootup. CanFestival poskytuje grafické uživatelské rozhranní (CanFestivalGUI) a rozhranní na příkazové řádce (CanFestival), které pomáhají vytvářet a editovat Objektové slovníky CANopen zařízení.

# Kapitola 6

# QCANalyser

Zde bude krátce představen projekt QCANalyser, který jsem dále rozvíjel.

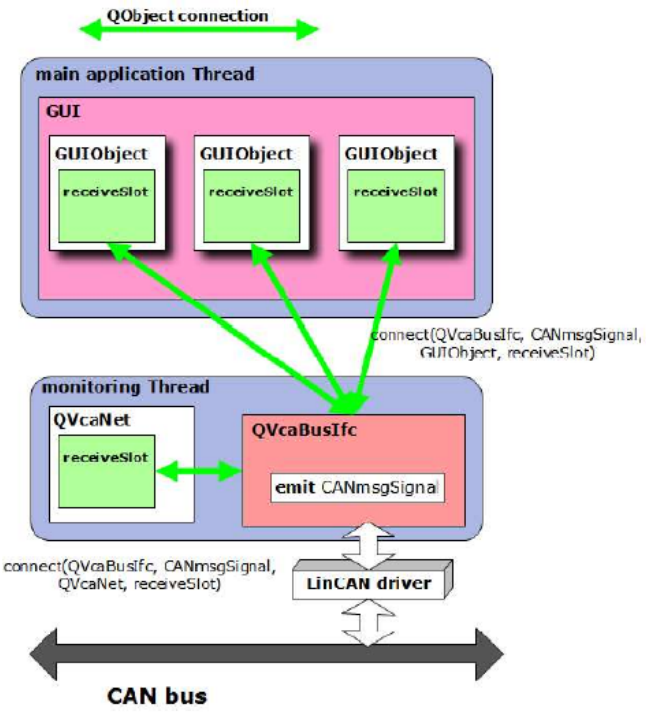
QCANalyzer je grafická aplikace založená na Qt, která umožňuje uživateli interagovat s CAN sběrnicí. Umožňuje velmi jednoduše sledovat traffic na sběrnici a to jak v reálném čase tak offline ze souborů. QCANalyser byl vytvořen Ing. Milošem Gajdošem ve spolupráci s Ing. Pavlem Píšou, Phd. a Ing. Františkem Vackem na Katedře Řízení na pražské ČVUT FEL.

Aplikace umožňuje:

- zobrazení CAN trafficu
- odesílání CAN zpráv
- filtrování přijatých zpráv
- zobrázení dat v grafu
- ukládání dat a nastavení do souboru

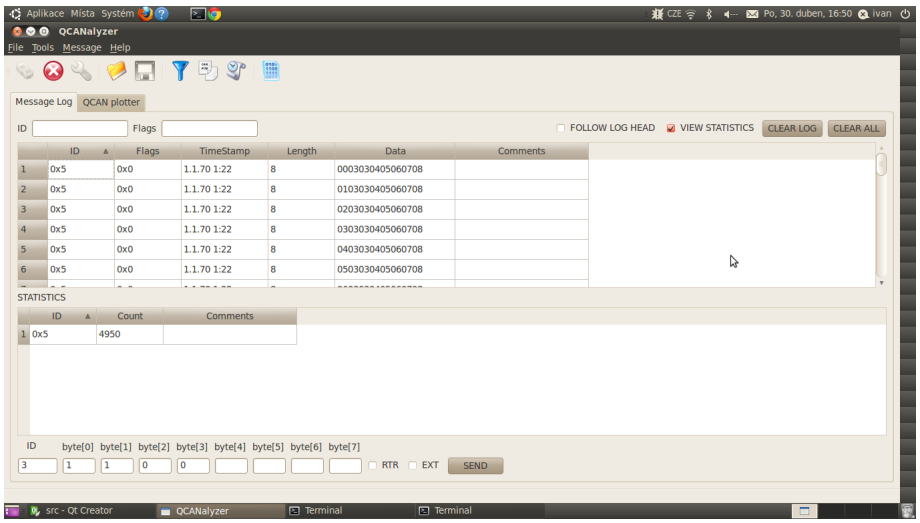
Grafická část je oddělená od QVCA knihovny kterou aplikace využíva k monitorování. Obě části běží v samostatných vláknech. Vlákna umožňují běžet dvoum segmentům kódu najednou nezávisle na sobě. Tento mechanismus zabraňuje pozastavení některé činnosti v době kdy některá funkce trvá déle. Zvyšuje to tak celkový výkon aplikace. Komunikace mezi vlákne probíhá díky thread save mechanismu signal-slot, který Qt framework poskytuje. Architektura QCANalyseru je naznačena na obrázku. Obrázek byl převzat z diplomové práce Miloše Gajdoše. podrobnější informace by jste našli v Diplomové práci Miloše Gajdoše s názvem CAN bus communication protocol support and monitoring z

roku 2008.




Obrázek 6.1: Architektura QCANalyseru.

K této aplikaci je připojen můj modul pro CANopen. Modul beží prakticky nezávisle na monitorování CAN sběrnice, což výrazně ulehčilo vývoj.




Obrázek 6.2: QCANanalyser - hlavní obrzovka.

# Kapitola 7

# Implementace CANopen do QCANalyseru

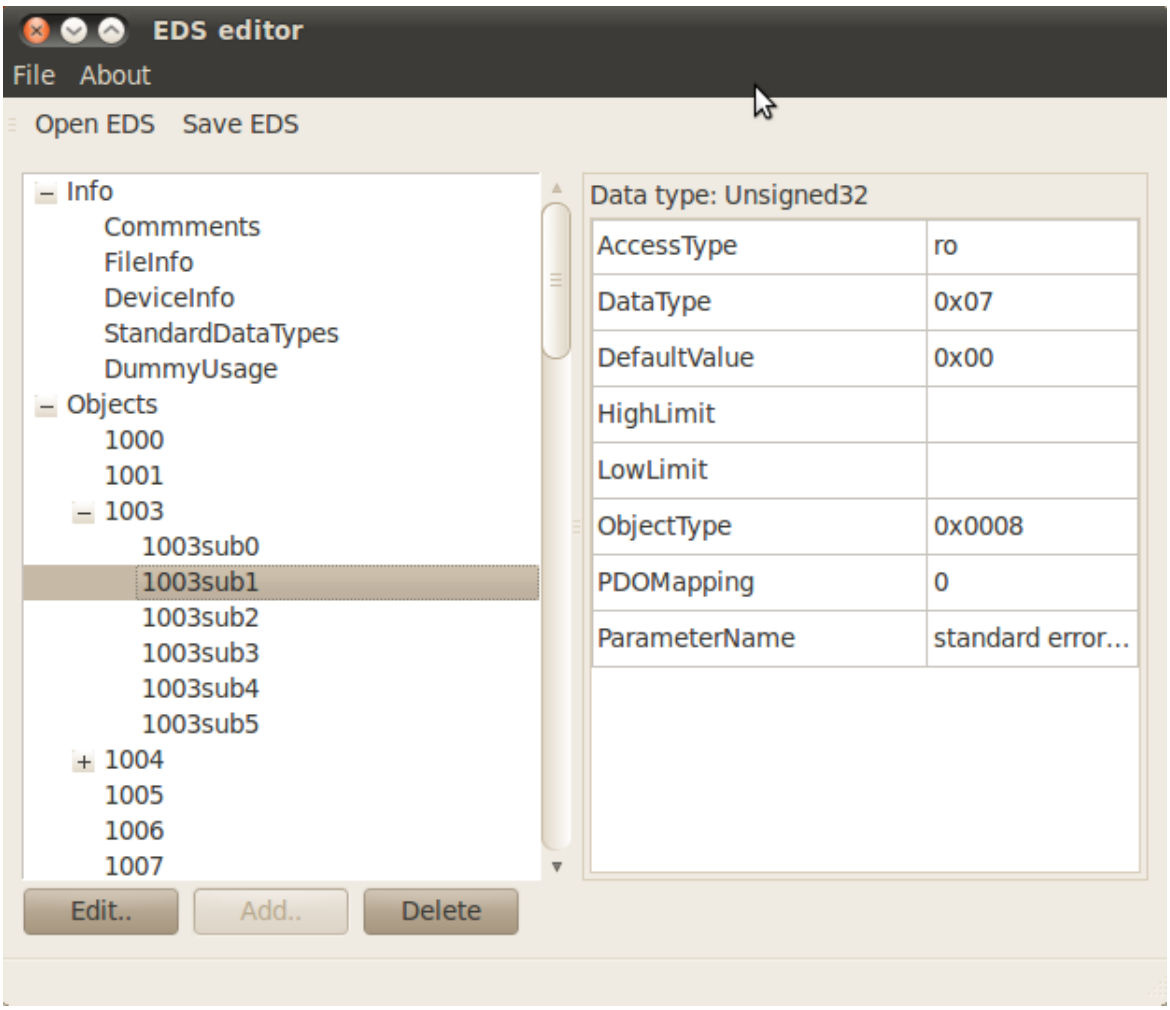
V této kapitole bude popsán vývoj CANopen rozhranní do QCANanalyseru.

## 7.1 EDS editor

Prvním úkolem bylo vytvořit grafické rozhranní pro vytváření a editaci EDS souborů. Rozhodl jsem se, že tento program vytvořím odděleně od QCANanalyseru pro jednodušší testování, a také proto, že takový program by mohl být užitečný i jako samotný nástroj.

EDS editor umožňuje:

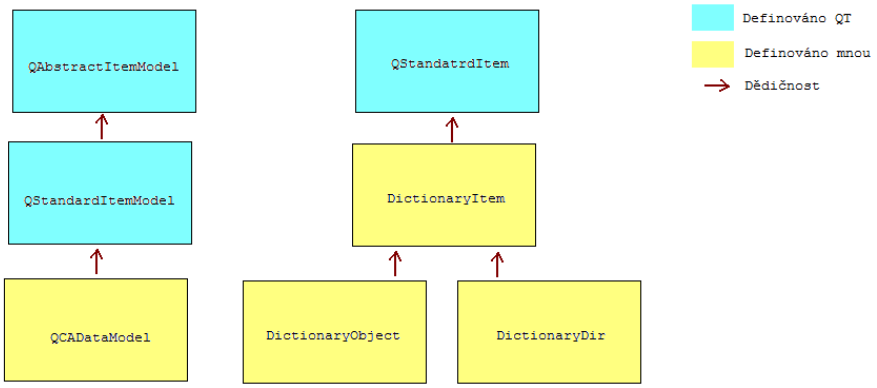
- vytváření nových EDS souborů podle šablony
- editování EDS souborů
- načítání EDS souborů s kontrolou syntaxe
- ukládání EDS souborů




Obrázek 7.1: EDS editor.

## Popis tříd EDS editoru

EDS editor využívá Model-View architekturu. Datový model EDS souboru je třída QCADataModel.Tato třída dědí vlastnosti od třídy QStandardItemModel z Qt knihovny. Použití tohoto mechanismu umožňuje zobrazení celého modelu v komponentě QTreeView. Prvky modelu tvoří adresáře DictionaryDir a listy DictionaryObject. Tyto třídy jsou potomky třídy DictionaryItem která je potomkem třídy QStandardItem z knihovny Qt. Adresář DictionaryDir reprezentuje objekty, které mají své podobjekty.




Obrázek 7.2: Struktura tříd datového modelu.

### Použití EDS editoru

Po spuštění EDS editoru se objeví prázdná šablona EDS souboru. Tuto šablonu si můžeme vyplnit vlastními objekty a informacemi a uložit pomocí tlačítka Save EDS jako vlastní EDS soubor. Pomocí tlačítka Open EDS otevřeme již hotový EDS soubor například od výrobce nějakého zařízení. Během načítání dojde ke kontrole syntaxe, a pokud je v souboru nějaká syntaktická chyba, program nás o ní upozorní v dialogovém okně. Pokud klikneme vlevo ve stromové struktuře na nějaký objekt, můžeme editovat jeho parametry pomocí tlačítka Edit. Můžeme ho smazat pomocí tlačítka Delete. Pokud chceme přidat nový objekt nebo subobjekt, klikneme na tlačítko Add. V pravé části obrazovky se nám zobrazují informace o daném objektu. Tato tabulka je needitovatelná. Nad tabulkou je uvedeno o jaký datový typ se jedná na základě parametru DataType daného objektu.

## 7.2 QCANalyser

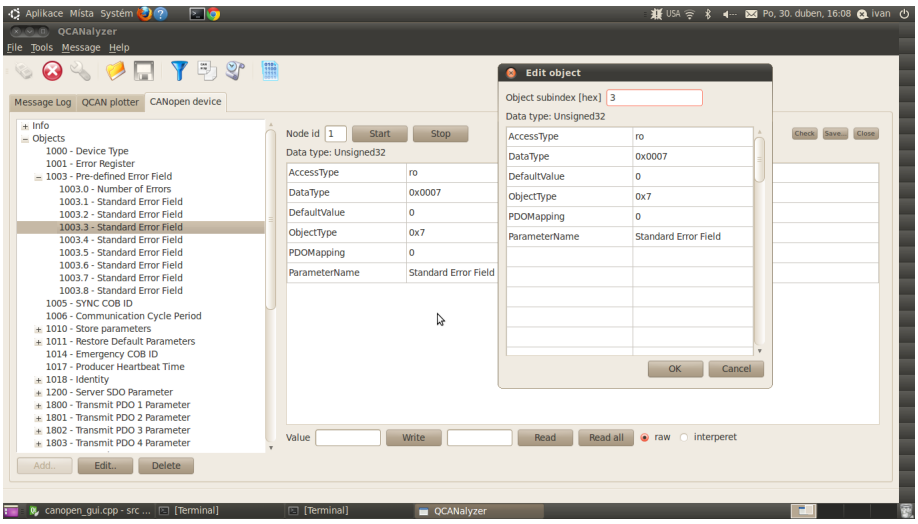
V této sekci je napsáno o implementaci CANopen do QCANalyseru a je zde i uveden návod k použití. Bude zde pojednáno i o opravě vykreslování dat CAN zpráv do grafu.

## Implementace CANopen

Editor a prohlížeč CANopen EDS souborů je implementovaný v jedné třídě s názvem canopen_gui. Objekty této třídy jsou napojené na hlavní okno přes signál, které informuje tento objekt o připojení-odpojení rozhranní typu QVcaBusIfcAbstract. Toto rozhraní je abstraktní - nezáleží na tom jakým způsobem je rozhranní realizováno. Každé okno pro CANopen v sobě obsahuje jednu instanci třídy QCamodel, která obsahuje naparsované objekty EDS souboru. Tento model je uložen a zobrazen v TreeViewWidgetu.

## Návod na použití CANopen

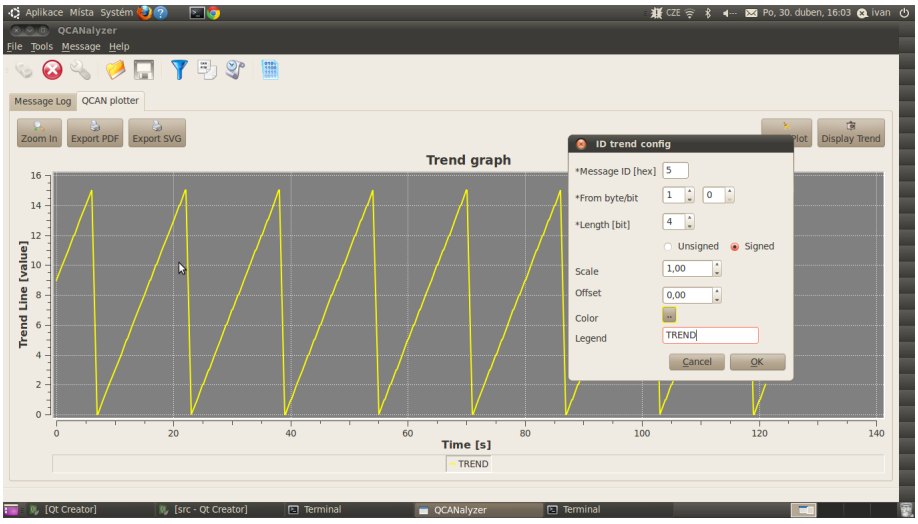
Vlevo ve stromové struktuře jsou komunikační objekty, podobjekty a informace o CANopen zařízení. V pravé části je kolonka Node id. Do této kolonky se zadává, jaké je číslo nodu daného zařízení - číslo musí mít hodnoty od 1 do 127. Pokud zadáme 0, bude to platit pro všechna zařízení na sběrnici. Tlačítko start vyšle paket, který přepne zařízení do stavu operational. Tlačítko Stop přepne zařízení do stavu Stopped. Tlačítko Check zkontroluje syntaxi EDS souboru a upozorní nás chybějící údaje. Tlačítko Save uloží dané EDS do souboru. Tlačítko Close zavře celé okno pro CANopen. Těchto oken si samozřejmě můžeme otevřít více najednou. Pomocí tlačítka write zapisujeme do zařízení hodnotu vybraného objektu. Pomocí tlačítka Read čteme hodnotu daného objektu. Tlačítko Read all přečte všechny hodnoty objektového slovníku. Můžeme pomocí radio button zvolit zda chceme hodnoty zapisovat a číst v čistém binárním formátu (Little Endian), nebo zda chceme pracovat s interpretovanými hodnotami v dekadickém formátu. Talčítka Add, Edit a Delete slouží přidávání, modifikaci a odebírání s objektů z objektového slovníku. Při čtení nebo zápisu dat se nám objeví CAN zprávy v hlavním okně QCANanalyseru, takže je můžeme snadno analyzovat.




Obrázek 7.3: CANopen v QCANalyseru.

# Úprava vykreslování dat CAN zpráv do grafu

Součástí práce též bylo opravit vykreslování do grafu hodnot od CAN zpráv. 5ešení je vidět na obrázku. Uživatel si zvolí od kterého bajtu a bitu se budou data brát, a jak budou dlouhá. Taktéž si zvolí, jestli to bude znaménkové číslo nebo bez znaménka.




Obrázek 7.4: Opravené vykreslování dat do grafu.

# Kapitola 8

# Testování

V této sekci bude krátké pojednání o tom, jak byla testována implementace CANopen do QCANalyseru.

Po implementaci bylo nutné vše otestovat. Nejprve se testovala kvalita parseru pro načítání a ukládání EDS souborů. Test probíhal tak, že se vzal korektní EDS soubor jednoho výrobku, načetl se do EDS editoru a následně se dal uložit. Výsledné dva soubory se potom porovaly pomocí nástroje diff. Tento nástroj zobrazuje odlišnosti dvou textových souborů. Vysledek dopadl absolutně dobře.

K testování CANopen komunikace bylo využito několik nástrojů.

- Readburst
- Canslave
- CanFestival slave
- Sendburst

Tento program je součástí projektu Ortcan.

Readburst je pomocný program pro zachycování dění na driverů CAN. Umožnil mi sledovat přenos SDO a PDO parametrů a vyhodnotit tak jejich korektnost. Tento program je součástí projektu Ortcan.

Canslave je program pro vytvoření softwarového CANopen zařízení. Tento program je součástí projektu Ortcan.

CanFestival slave je program, který jsem vytvořil z příkladu uvedeného CanFestivalem. I pomocí něho se dala ověřit správná funkčnost QCANalyseru.

Sendburst je program, která na sběrnici v pravidelných intervalech zprávy. To bylo využito pro testování vykreslování dat do grafu.

# Kapitola 9

# Správa zdrojových kódů a jejich úprav

Při programování rozsáhlejších projektů jako běžně Open Source projekty, je vhodné použít pro správu díla nějaký vhodný verzovací systém. Verzovací systémy evidují změny provedené v jednotlivých verzích během stádia vývoje softwarového projektu. Eviduje se kdo, kdy a jak změnil řádky zdrojového kódu programu. To slouží nejenom k úplnému přehledu všech změn, ale také možnost vidět přesný stav sledovaných dat v kdykoliv v minulosti a také vrátit se k předchozí verzi daného programu v případě, že během dalšího vývoje dojde k chybám. Každé změně provedené ve zdrojových kódech je přidělováno unikátní číslo, označované většinou jako číslo revize. Velmi významný je verzovací systém pokud se na něm podílí více programátorů současně. Systém pak slouží k detekci a odstraňování kolizí. V Open Source se tohoto velmi často využívá, neboť na projektu mohou pracovat i stovky programátorů z celého světa, aniž by museli být v kontaktu. Jakýkoliv větší projekt si dnes nelze bez verzování představit. Nejznámějšími představiteli verzovacích systémů jsou Git, CVS a Subversion, všechny voně dostupné. CVS a Subversion jsou centralizované verzovací systémy, Git je distribuovaný systém. Centralizované systémy vyžadují komunikaci se serverem, na kterém je projekt uložen. Na projektu s distribuovaným verzováním se dá pracovat lokálně, bez nutnosti přístupu k serveru. Verzovací systémy většinou neuchovávají úplný stav každé revize, ale pouze rozdíly mezi jednotlivými revizemi (pomocí nástrojů typu diff). Informační hodnota je stejná a data jsou velmi malá.

Pro můj projekt byl vybrán verzovací systém Git. Git je distribuovaný verzovací systém založený Linusem Torvaldsem. Původně sloužil pro vývoj jádra Linuxu. Git je

nízko úrovňový nástroj vybaven velkým množstvím příkazů, a tak se s ním dá dělat mnoho rozličných operací možná za cenu větší složitosti. Git je velmi efektivní pro práci s velkými projekty, oproti CSV a SVN je asi desetkrát rychlejší.

Repozitář git projektu QCANalyser se nachází na webové stránce:
http://ortcan.git.sourceforge.net/git/gitweb-index.cgi

# Kapitola 10

# Závěr

V rámci práce byl úspěšně navržen nástroj pro práci a přístup ke CANopen slovníkům a byl plně integrován do programu QCANanalyzer. Načítání a práce se slovníky byla ověřena s mnoha EDS soubory od různých výrobců CANopen zařízení.

Volba vyvíjet aplikaci s využitím Qt frameworku se ukázala jako velmi vhodná a mechanismus signal-slot ušetřil mnoho práce s vývojem.

Obtížný úkol představovalo studium již existujících zdrojových kódů, které se staly základem, na kterém je moje implementace postavená. Přesto, že tato fáze zabrala více času než jsem předpokládal, tak se jí podařilo úspěšně zvládnout a text práce přináší rozříření dokumentace některých částí již existujících.

Ze začátku byl problém se zorientovat ve zdrojových kódech, ale s pomocí vedoucího se to zvládlo nakonec velmi rychle. Nástroj QCANalyser se dá považovat za plnohodnotný nástroj pro práci s CAN a CANopen.

CanFestival se ukázal jako velmi zajímavý framework pro práci s CANopen. Implementování je velmi jednoduché. Pouze dokumentace k projektu je není úplně dostatečná.

Původně bylo v plánu implementovat CanFestival do hardwarového zařízení. Lze však říci, že převod do hardware je záležitost přímočará a zaručuje shodné chování jako na PC, a proto bylo od toho záměru upuštěno.

# Literatura

1 CAN international users and manufacturers group web site, http://www.can-cia.org/
2 The Peak company web site , http://www.peak-system.com/.
3 VSCP project web site, http://www.vscp.org/.
4 OCERA project web pages, http://www.ocera.org/.
5 OCERA project development web pages, https://sourceforge.net/projects/ocera/.
6 Gtk+ project documentation , http://gtk.org/.
7 Qt framework web pages, http://trolltech.com/products/qt.
8 Java-SWING tutorial web pages, http://java.sun.com/docs/books/tutorial/uiswing/.
9 Qt framework online documentation, http://doc.trolltech.com/4.3/.
10 Qwt Project Online Documentation, http://qwt.sourceforge.net/.

# Příloha A

# Obsah přiloženého CD

- Diplomová práce ve formátu pdf
- EDS editor
- QCANanlyser - upravený
- Projekt CanFestival se slave zařízením