České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

DIPLOMOVÁ PRÁCE

Open source komponenty pro sběrnici μLan

Vedoucí práce: Ing. František Vacek

Praha, 2011

Autor: Bc. Jan Štefan

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

V Praze dne _____ _____

podpis

Poděkování

Děkuji především vedoucímu diplomové práce Ing. Františku Vackovi a garantovi diplomové práce Ing. Pavlu Píšovi Ph.D. za pomoc při realizaci projektu.

Abstrakt

Cílem tohoto dokumentu je popsat průběh vývoje softwaru pro připojování virtuálních zařízení k průmyslové sběrnici μLan. Práce obsahuje rozbor komunikace po sběrnici μLan a její fyzické vrstvy RS-485. Dále seznamuje s problematikou funkcionálně deklarativního jazyka QML. V práci je uveden rozbor implementace aplikace a jejích jednotlivých vrstev. Součástí práce je návod na zprovoznění sběrnice μLan pro operační systém Linux.

klíčová slova: μLan, QML, Qt Framework, průmyslová sběrnice

Abstract

The goal of this thesis is to describe program developing process for connecting virtual devices to industrial bus μLan. An analysis of comunication μLan bus and its physics layer RS-485 is made within this project. Furthermore this paper deals with the issue of the functional declarative language QML. The thesis presents an analysis of the implementation of application and its individual layers. The part of thesis is install guide on commisioning μLan bus for the Linux operating system.

keywords: μLan, QML, Qt Framework, industrial bus

vložit originální zadáníííííííííííííííííí !!!!!

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

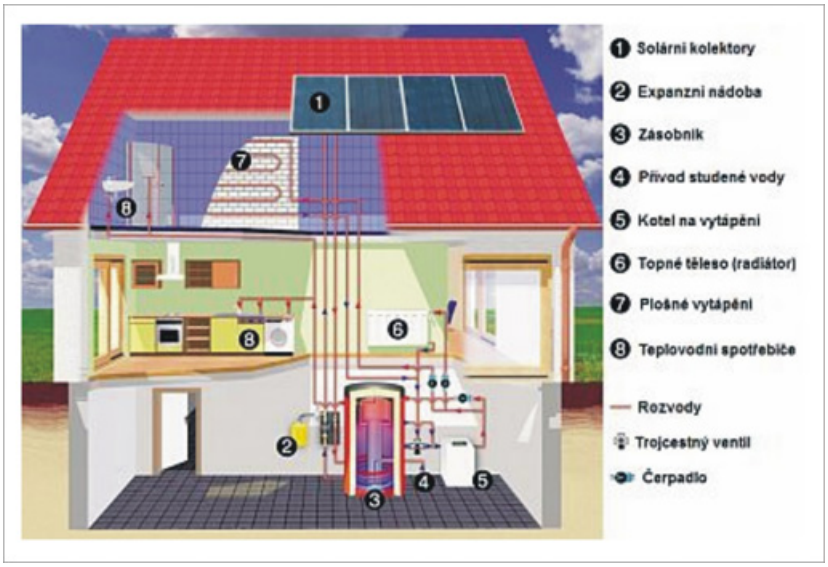
1.1 Motivace: Nástroj pro simulování řídicích systémů

Motivací této práce je vytvořit aplikaci, která slouží jako simulátor návrhu řídicího systému. V aplikaci se otevřou virtuální zařízení, která se připojí na průmyslovou sběrnici. Každému zařízení se nadefinuje chování v rámci sítě a prostřednictvím aplikace *μLAN-admin* napsané kolegou Bc. Martinem Boháčkem se nakonfiguruje, která zařízení spolu mají komunikovat a jak. Virtuální a reálná zařízení mají společnou komunikační vrstvu napsanou v jazyce C, takže nastavení je mezi nimi přenositelné. Uživatel si tak bude moci nejprve vše vyzkoušet na virtuálních zařízeních a nastavení použít i pro reálná zařízení. Výhoda, která z této vlastnosti plyne, je, že uživatel může ovládat reálná zařízení virtuálními a naopak.

1.2 Řídicí systémy inteligentních domů

Navržená sada nástrojů se může použít například pro namodelování řídicího systému pro inteligentní dům. V dnešní době se stává moderním instalovat do našich domů kompletní řídicí systémy. Při stavbě nebo rekonstrukci našich domovů bychom měli myslet na to, že je 21. století a tomu by měl odpovídat i výběr elektroinstalace. Použití moderních technologií v oblasti inteligentních budov nám vše zjednoduší a dopřeje nám větší komfort než klasická elektroinstalace. V této oblasti již působí velké množství firem, které nám nabízejí různá technologická řešení, takže záleží jen na nás, kterému dáme přednost.

Inteligentní dům (viz. obrázek 1.1[1]) je ve většině případů řízen centrálním systémem, ke kterému jsou připojeny jednotlivé vstupní senzory a výstupní prvky. Chování systému je dané jeho naprogramováním, takže pokud ho chceme změnit nemusíme vyměňovat celou elektroinstalaci, ale pouze přeprogramujeme řídicí systém. Inteligentní dům za nás může vykonávat mnoho činností, o které se následně nemusíme starat. Například systém může zapínat topné jednotky hodinu před tím, než se vrátíme z práce. Nebo může vypnout topení nacházející se pod oknem, potřebujeme-li otevřít okno. Řídicí jednotka může v ranních hodinách automaticky vytahovat žaluzie. Pokud bychom v našem domě měli nainstalovaný složitější vytápěcí systém, který by se skládal z topné jednotky na tuhá paliva (například krbová vložka), solárních panelů (k ohřevu teplé vody či vyhřívání bazénu) a plynového kotle lze řídit distribuci tepla prostřednictvím centrálního tepelného výměníku. V letním období za příznivých slunečních podmínek není potřeba vytápět dům. Přebytečná energie se může využít k ohřevu vody v bazénu. Nebo pokud zatopíme v krbu, vypne se plynový kotel. Máme-li plně nahřátý centrální výměník, můžeme opět ohřívat bazén. Zkrátka možností využití je nepřeberné množství a záleží na tom, jak složitý systém si chceme sestavit.



Obrázek 1.1: Inteligentní dům

Tyto systémy samozřejmě umožnují dálkovou správu, ať už přes vizualizaci na touchpanelech, prostřednictvím SMS nebo přes internet. Takže pokud se vracíme z práce domů dříve než obvykle, můžeme si v domě ručně zapnout vytápění přes chytrý telefon.

[1] www.hqline.com

1.3 Stanovení cílů

V diplomové práci jsem si vytyčil následující cíle:

- vytvořit aplikaci, která umožnuje připojit virtuální zařízení transparentní pro μLAN síť
- postavit virtuální zařízení na stejném *firmwaru* jako reálná zařízení
- vytvořit sadu konfiguračních souborů, ve kterých lze měnit vlastnosti virtuálních zařízení bez znalosti jazyka C
- pro tvorbu aplikace využít Qt Framework
- vytvořit několik příkladů virtuálních zařízení v jazyce *QML*
- vytvořit dokumentaci, podle které bude každý uživatel schopen si μLAN zprovoznit sám na svém počítači

Kapitola 2

Protokol μLAN

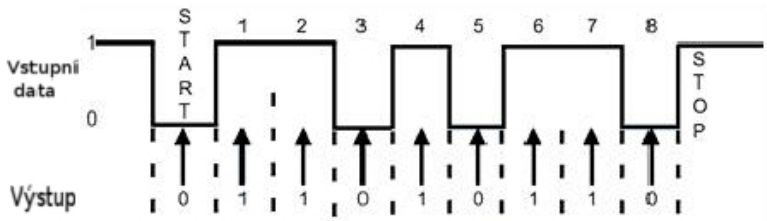
Diplomová práce využívá a dále rozvíjí open source projekt *μLAN protocol for RS-485 9-bit network*, jehož hlavním tvůrcem je doktor Pavel Píša. Protokol μLAN kombinuje dobré vlastnosti ostatních průmyslových sběrnic. Od ostatních řešení se tento projekt liší například tím, že řídicí systém není centralizovaný. Každé zařízení má svou vlastní inteligenci. Vzhledem k tomu, že systém využívá ke komunikaci fyzickou vrstvu *RS-485*, potřebujeme pro přenos dat pouze dva vodiče. Pro rozvod μLAN sítě v domě můžeme například využít již zabudovanou telefonní linku. Další výhodou tohoto řešení je, že μLAN síť nemá pevně určeného *mastera*, který by sběrnici arbitroval, takže chod celé šítě není závislý na chodu jednotlivých zařízení. Síť funguje na poměrně velké vzdálenosti, což by mělo stačit i na zautomatizování rozlehlého sídla. Následující popis protokolu μLAN byl inspirován dokumentem „ul_drv - uLan RS-485 Communication Driver“ vytvořeným doktorem Pavlem Píšou.

2.1 Fyzická vrstva RS-485

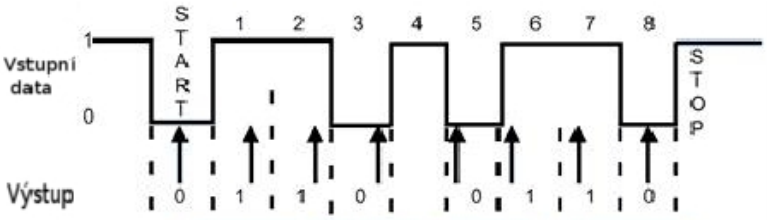
2.1.1 Obecný popis

Protokol RS-485 [4] se v průmyslu používá především pro přenos dat na velkou vzdálenost. Data se po něm přenášejí sériovým způsobem bez nutnosti použití modulace. Sběrnice némá vyvedený samostatný hodinový signál, který by sloužil pro synchronizaci vysílaných a přijímaných dat. Proto se veškeré zprávy na sběrnici posílají asynchronně. Pro jednosměrný přenos dat se používají pouze dva signálové vodiče. Lze ještě přidat třetí vodič, který určuje signálovou nulu. Veškeré řízení přenosu dat je prováděno progra-

mově, většinou na základě harwarového handshackingu. Přijímač a vysílač musí pracovat na stejné předem určené frekvenci, aby nedošlo k chybě přenosu rámce. Chyba vzniká, když se vzorkuje na přechodu mezi dvěma bity. Názorně to můžeme vidět na obrázku 2.1 a 2.2[1].



Obrázek 2.1: Formát dat RS485 bez chyby



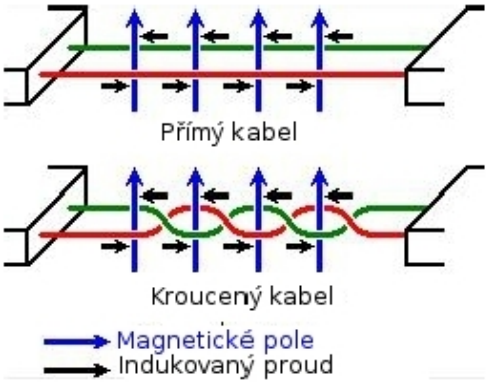
Obrázek 2.2: Formát dat RS485 s chybou

2.1.2 Charakteristika sběrnice

Přenos dat po dvoudrátovém vedení zajišťuje stav bitu podle rozdílu napěťového potenciálu mezi vodiči (diferenciální přenos). Výhoda přenosu po dvou vodičích spočívá v možnosti použití kroucené dvojlinky, která umožnuje velkou přenosovou rychlost bez většího vyzařování signálu do okolí nebo naopak (menší zatížení dat šumem). Každý delší vodič totiž vysílá a přijímá elektromagnetické vlnění. U kroucené dvojlinky jsou oba nosiče informace vlněním ovlivněny stejně, takže přijímač může být velmi citlivý. Na rozeznání jiné logické úrovně stačí napětí 200mV.

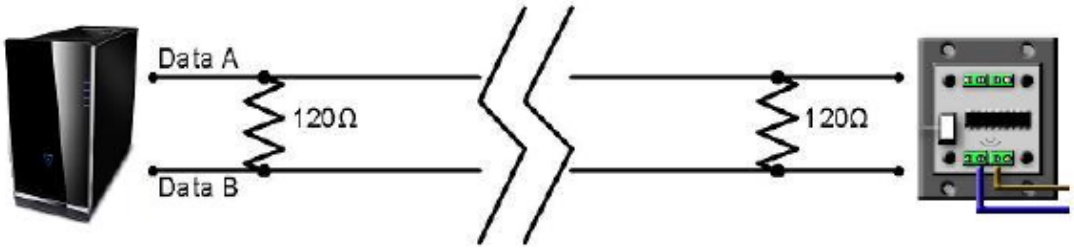
[1] http://www.root.cz/clanky/sbernice-rs-422-rs-423-a-rs-485/

Na obrázku 2.3[2] je ukázán vliv magnetického pole na rušení.



Obrázek 2.3: Vliv elektromagnetického pole na kabel

Přenos může být uskutečněn až na vzdálenost 1200 m s přenosovou rychlostí 100 kbps. Na vzdálenost do 15 metrů můžeme docílit rychlosti až 10 Mbps. Při realizaci RS-485 musíme na oba konce sběrnice připojit rezistory o velikosti přibližně 120Ω, což by mělo odpovídat impedanci kroucené dvojlinky. Typické zapojení vidíme na obrázku 2.4[3].



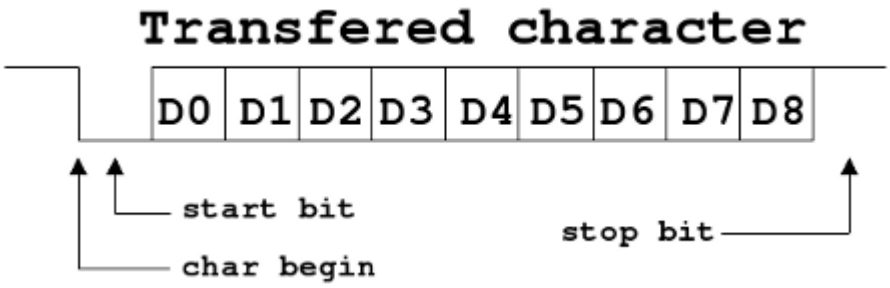
Obrázek 2.4: Zapojení kroucené dvojlinky

[2]http://www.root.cz/clanky/sbernice-rs-422-rs-423-a-rs-485/
[3]Zdroj: vlastní zpracování

2.2 Popis sběrnice μLAN

2.2.1 Formát dat

μLAN [2] je devíti bitový asynchronní komunikační protokol, který je postaven na fyzické vrstvě RS-485. Použití devíti bitových znaků zjednodušuje přenos binárních dat a může snížit zatížení procesoru, který se nemusí starat o přeposílání znaků dalším zařízením. Znak začíná start bitem, následuje sekvence devíti bitů a končí stop bitem (viz. obr 2.5[4]). Znaky, které mají na tučně zvýrazněném bitu D8 hodnotu 1, jsou takzvané speciální znaky. V datovém rámci se mohou tyto znaky vyskytovat pouze na přesně definovaných pozicích. Zároveň slouží k označení přenášených dat a rozhodují o stavu obsazení sběrnice.



Obrázek 2.5: Formát znaku u μLANu

2.2.2 Datový rámec

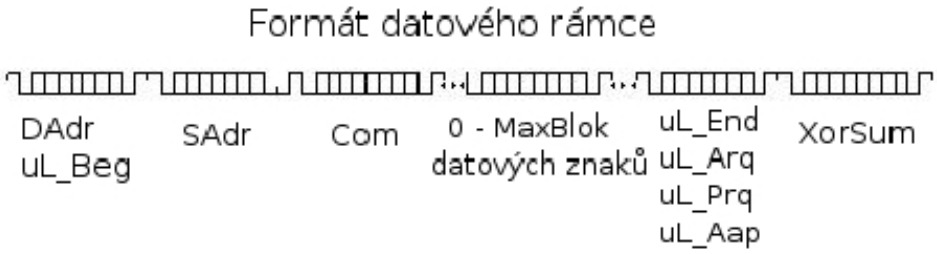
Datový rámec se skládá ze sekvence znaků. První znak určuje adresu příjemce *DAdr* nebo neadresný znak *uL_Beg*, který má bit D8 nastaven v logické 1. Zpráva je vyslána na sběrnici, každé zařízení ji obdrží a podle prvního znaku se rozhodne, zda-li zpráva byla určena pro něj. Následují znaky s adresou odesílatele *SAdr* a kódem služby *Com* (command). Poté jsou odeslána data, která mají nastaven bit D8 v logické 0. Není s nimi odesílána délka dat. Ta je určena maximální povolenou délkou. Maximální délka dat je stanovena podle konkrétní aplikace. Záleží na počtu periferií v síti, na nejdelší přípustné odezvě mezi nimi a na rychlosti komunikace. Poslední znak dat, který má nastaven bit D8, rozhodne o následujícím průběhu komunikace.

[4] http://ulan.sourceforge.net/

Jsou zde čtyři možnosti:

- datový rámec se okamžitě ukončí - *uL_End*
- zašle se kontrola o doručení datového rámce - *uL_Arq*
- datový rámec je okamžitě zpracován příjemcem - *uL_Prq*
- příjemce potvrdí přijetí datového rámce a následně ho zpracuje - *uL_Aap*

Jako poslední znak je odeslán kontrolní součet *XorSum*. Příklad datového rámce je uveden na obrázku 2.6[5].



Obrázek 2.6: Datový rámec zprávy

2.2.3 Přenos dat

Zařízení odesílající zprávu musí čekat na uvolnění sběrnice. Jakmile je sběrnice volná, připojí se na ni, odešle datový rámec a případně odešle potvrzení o přijetí. Následně je sběrnice uvolněna. Pokud není detekována chyba, zpráva se označí jako doručená. Aby zařízení mohlo zprávu přijmout, musí mít nastavenou adresu a musí umět rozeznat znaky, které mají nastavený bit D8.

2.2.4 Arbitrace sběrnice

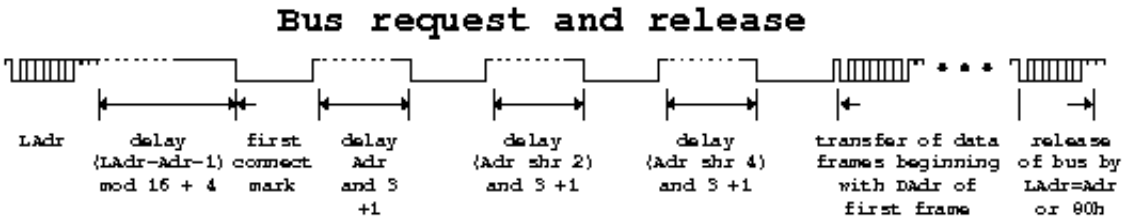
Všechna zařízení v μLAN síti jsou si rovna. Než dostane zařízení právo vysílat, proběhne přípravná fáze (tzv. arbitrace). Při arbitraci se zaručí, že může vysílat pouze jedno zařízení. To vše je doplněno o rotování priority mezi přístroji. Je určen minimální čas,

[5] http://ulan.sourceforge.net/

během kterého se nesmí přístroj připojit. V závislosti na adrese předcházejícího přístroje s oprávněním vysílat. Zařízení jsou schopna přijímat znaky, které mají nastaven bit D8. To vše znamená, že každý zná adresu zařízení *LAdr*, které opustilo sběrnici. Každé zařízení si spočítá dobu, po kterou bude testovat klid na sběrnici, podle vzorce (2.1) na základě své adresy *Adr*, adresy posledního zařízení *LAdr* a doby přenosu jednoho znaku T_{chr}.

$$T_{arbW} = ((LAdr - Adr - 1) \mod 16 + 4)T_{chr} \tag{2.1}$$

Všechna zařízení žádající o přístup na sběrnici začnou vysílat kombinaci prázdných intervalů a *000h* znaků s periodou T_{arbW}. Vyslání prázdného znaku znamená odpojení výstupního budiče a po tuto dobu je sběrnice v logické 1. Pokud zařízení vyšle prázdný znak a sběrnice je v logické 0, znamená to, že se na sběrnici pokouší připojit jiné zařízení. Zařízení, které vysílalo prázdný znak, považuje sběrnici za obsazenou. Podle vypočteného času z výše uvedeného vztahu je kombinace vysílaných signálů nastavena tak, že pouze jedno zařízení najde při poslání prázdného znaku sběrnici v logické 1. To se opakuje ještě dvakrát, než přístroj získá sběrnici. Tento způsob arbitrace zajišťuje synchronizaci maximálně mezi 64 přístroji. Aktuální stav sběrnice se sleduje na bitu D7. Zde se definuje příznak obsazenosti *uLF_NB*. Po příjmu kteréhokoli znaku se příznak vynuluje. K testu aktivity sběrnice slouží přístrojům funkce *uL_STROKE*. Ta sleduje, jestli mezi prvním a druhým voláním funkce byl přijat alespoň jeden znak. Maximální prodleva během komunikace nesmí přektočit dobu, za kterou se přenesou tři znaky. Graficky je průběh arbitrace znázorněn na obrázku 2.7[6].



Obrázek 2.7: Schéma arbitrace sběrnice μLAN

[6]http://ulan.sourceforge.net/

2.3 Vyšší vrstvy protokolu μLAN

Níže popsané vrstvy komunikují prostřednictvím servisních zpráv (SDO – Service Data Object). SDO zprávy jsou odpovědi zaslané na základě požadavku z μLAN sítě. Dotazovatel musí mít dostatek informací na dekódování zpráv.

2.3.1 Network Control Messages (uLNCS)

uLNCS je vrstva, která svými příkazy umožnuje správu μLAN sítě. Modul po zapnutí napájení a připojení do μLAN sítě může zažádat o přidělení adresy, která není v databázi serveru dynamických adres. Žádosti jsou na síť posílány formou *broadcastu*. Nová adresa může být modulu zaslána buď na základě požadavku od uživatele, nebo od serveru. Každá zpráva tohoto typu má příznak *UL_CMD_NCS* (network control service). Server, který zprávu obdrží na základě tohoto příkazu, ví, že v prvním bajtu zprávy je uložen upřesňující požadavek (subcommand). Seznam těchto subcommandů je uveden v tabulce 2.1[7]. Na následujících čtyřech bajtech zprávy je uloženo sériové číslo (s/n). Při nastavování adresy je nutné, aby si modul zkontroloval sériové číslo obsažené ve zprávě, protože to je jediná možnost, jak modul pozná, že mu zpráva o nastavení adresy byla určena. Počáteční adresy se doporučují nastavit v rozmezí 0 až 99. Hlavní výhoda tohoto způsobu přidělování adres je, že není potřeba skenovat celou síť. *Broadcastem* se vyšle příkaz a přijmou se odpovědi od zařízení.

Tabulka 2.1: Subcommandy vrstvy *uLNCS*

Subcommand	**Formát dat**	**Vysvětlivka**
ULNCS_RQ_ADDR	SN0 SN1 SN2 SN3	Požadavek o novou adresu
ULNCS_SET_ADDR	SN0 .. SN3 NEW_ADDR	Nastavení nové adresy modulu
ULNCS_SID_RQ		Požadavek o identifikaci
ULNCS_SID_RPLY	SN0 .. SN3 SID_string	Odpověď na identifikaci
ULNCS_RQ_ADDR	SN0 SN1 SN2 SN3	Pernamentní změna adresy

[7] http://ulan.sourceforge.net/

2.3.2 Dynamic address assignment (uLDY)

Tato vrstva se nejčastěji používá v prostředí, kde se často mění připojené moduly mezi μLAN sítěmi. Vrstva používá porovnávání sériových čísel modulů, implementace síťových služeb (UL_CMD_NCS) a zprávy s okamžitou odpovědí (UL_CMD_SNST). Ke každé síti musí být připojen jeden server dynamických adres, který do sítě publikuje svojí adresu formou *broadcastu*.

2.3.3 μLAN Obejct Interface Layer (uLOI)

Objektový slovník (OI) je vrstva, ve které může mít každé zařízení definované datové proměnné viditelné v rámci μLAN sítě. Vrstva komunikuje prostřednictvím asynchronních SDO zpráv. Formát zpráv je navržen tak, aby byl co nejkratší a nejobecnější.

Pro komunikaci s objektovým slovníkem musí mít zpráva příznak *UL_CMD_OISV*. Každá zpráva se skládá z tříbajtové hlavičky a serializovaných dat (viz tabulka 2.2[8]). Z těchto dat dva bajty určují identifikační číslo (OID) pro přístup k objektovým datům nebo službám. Ve zbytku zprávy jsou samotná data nebo metadata.

Tabulka 2.2: Formát zprávy pro OI

název	**délka**	**význam**
frame cmd	1 bajt	příznak zprávy
bcmd	1 bajt	příznak pro odpověď
sn	1 bajt	sériové číslo
bsn	1 bajt	s/n pro odpověď
OID number	2 bajty	objektový identifikátor
OI_data	1 .. n bajtů	data nebo metadata

uLOI vrstva umožnuje uchovávat popis objektů přímo v modulech. Každý slovník musí mít nadefinovanou minimální sadu objektů (viz. tabulka 2.3[9]) a dále může mít libovolný počet dalších nově nadefinovaných objektů.

[8] http://ulan.sourceforge.net/
[9] http://ulan.sourceforge.net/

Tabulka 2.3: Tabulka minimální sady objektů

Objekt	**Funkce**
ULOI_DOII	obsahuje vlastnosti objektů určených pro zápis
ULOI_DOIO	obsahuje vlastnosti objektů určených pro čtení
ULOI_QOII	obsahuje identifikační čísla objektů pro zápis
ULOI_QOIO	obsahuje identifikační čísla objektů pro čtení
ULOI_RDRQ	zjišťuje hodnoty objektů
ULOI_SNCHK	kontrola sériového čísla modulu
I_STATUS	objekt pro čtení stavu modulu
I_ERRCLR	pouze pro zápis, nastavuje status na 0

Definice každé proměnné musí nezbytně obsahovat několik parametrů. Název, identifikační číslo, typ proměnné a informaci o tom, jestli je proměnná pro zápis, pro čtení nebo pro zápis i pro čtení. Pro představu uvádím příklad definice jedné proměnné:

```
ULOI_GENOBJDES(VARNAME, I_VARNAME, "type/additional info", uloi_rdfnc,
&oi_varname, uloi_wrfnc, &oi_varname)
```

Název je označení, kterým se proměnná prezentuje v rámci μLAN sítě. Identifikační číslo musí být unikátní v rámci jednoho slovníku. Protokol si rezervuje rozsah OID čísel v rozsahu od 0 do 127. Z toho důvodu u nově definovaných proměnných musí být OID z rozsahu od 128 do 32767. Při deklaraci je třeba, aby po sobě jdoucí objekty měly rostoucí identifikátory. Posloupnost může obsahovat mezery. Typ proměnné je řetězec, podle kterého se určuje skutečný datový typ. Seznam datových typů je uveden v tabulce 2.4. Do řetězce je možné vložit další pomocné informace. Tyto informace se vkládají za lomítko. Objekt je určen pro čtení, pokud má nadefinovaný ukazatel na data a funkci, která tato data čte. Knihovna μLAN obsahuje funkce pro čtení základních datových typů. Například celá čísla se znaménkem čte funkce *uloi_int_rdfnc* a hodnoty bez znaménka čte funkce *uloi_uint_rdfnc*. Těmto funkcím se předává parametr ukazatel na data, která se mají přečíst. Na stejném principu fungují objekty určené pro zápis. Odpovídající funkce se nazývají *uloi_int_wrfnc* a *uloi_uint_wrfnc*. Pokud chceme provést některé operace, jež se mají vykonat během zápisu nebo čtení, je možné objektu předat jméno vlastní funkce.

2.4 Datové typy protokolu μLAN

Všechna data jsou ve formátu little-endian (LE). Datový typ a délka musí být známa u jednotlivých objektů na straně klienta předem.

Seznam základních datových typů protokolu μLAN je uveden v tabulce 2.4[10]. První znak kódu proměnné určuje, o který typ proměnné se jedná, a druhý znak nám říká, na kolika bajtech je datový typ uložen.

Tabulka 2.4: Základní datové typy protokolu μLAN

kód	**popis**	**kódování a délka**
s1, s2, s4	signed integer	LE, délka 1,2 a 4 bajty
u1, u2, u4	unsigned integer	LE, délka 1,2 a 4 bajty
f2, f4, f8	floating point number	LE, délka 2,4 a 8 bajtů
vs, vsXX	visible string in UTF-8	první bajt udává délku řetězce

Do kódu datového typu se může přidat doplňující informace, které jsou od kódu typu odděleny lomítkem. Například za znakem tečka (".") může být uvedeno, na kolik desetinných míst chceme zaokrouhlit číslo s plovoucí řádovou čárkou. Kód datového typu pak vypadá následovně "f4/.2".

U definice polí píšeme před datový typ v hranatých závorkách číslo určující délku pole. Například "[5]u4"definuje pole o velikosti pěti čtyř bajtových *integerů* bez znaménka.

Datový typ *Visible string* definuje svoji maximální délku číslem, které následuje za znaky "vs". Například definice proměnné "vs20" říká, že se jedná o datový typ *Visible string* o délce dvaceti znaků.

Protokol μLAN obsahuje kromě obecných datových typů ještě speciální datové struktury. Struktury slouží pro ukládání informací o namapování μLAN sítě a konfiguraci zařízení. Jedná se o *PICO*, *POCO* a *PEV2C* struktury, které jsou detailněji popsány v následující kapitole.

[10] http://ulan.sourceforge.net/

2.5 Propojení μLAN sítě přes kanály

Pro výměnu dat mezi jednotlivými moduly slouží vrstva podporující posílání procesních zpráv (PDO - Process Data Objects). Veškerá data jsou v PDO zprávách označena identifikačním číslem kanálu (CID - Channel ID). Data objektových slovníků lze mezi sebou mapovat v relaci n:n. Jedna zpráva může obsahovat více dat s více CIDy. Moduly se při přijetí PDO zprávy rozhodují pouze na základě identifikačního čísla. Modul přečte všechny CIDy ze zprávy a porovná je s čísly, která má uložená v mapovacích tabulkách. Pokud jsou data určena pro daný modul, data zpracuje, jinak je ignoruje. Jediný požadavek na namapování dvou proměnných je, aby se shodoval jejich datový typ.

2.5.1 PDO zprávy

PDO [2] jsou nepotvrzované asynchronní zprávy. Moduly vysílají tyto zprávy nezávisle formou *broadcastu* a jsou označeny příznakem *UL_CMD_PDO*, který určuje asynchronnost datového rámce. První dva bajty zprávy jsou rezervované pro budoucí statické rozšíření a následující bajt může být v budoucnu použit jako hlavička PDO zpráv. V současné verzi protokolu se první tři bajty neoužívají a jsou vyplněny nulami. Každý datový blok PDO zprávy se dále skládá z dvoubajtového identifikačního čísla kanálu, jednobajtové informace o délce dat (do budoucna je zde prostor pro rozšíření až na dvoubajtovou informaci o délce) a dat samotných. Struktura zprávy je vidět níže v tabulce 2.5[11]. Maximální délka PDO zprávy je 127 bajtů.

Tabulka 2.5: Formát PDO zprávy

Res Lo	Res Hi	Ext len(el)	Ext	CID	data len(dl)	data	CID...
1 bajt	1 bajt	1 bajt	0..el bajtů	2 bajty	1(2) bajty	dl bajtů	

2.5.2 Mapování sítě

Vzhledem k tomu, že se zprávy posílají formou *broadcastu*, můžou je číst všechna zařízení připojená na sběrnici. Pokud chceme aby se modul účastnil datové výměny, musí mít vyplněné tabulky, které říkají, na kterém komunikačním kanálu má modul zprávy přijímat.

[11] http://ulan.sourceforge.net/

Všechny tyto informace jsou v zařízení uloženy prostřednictvím objektového slovníku u každého modulu zvlášť pomocí speciálních struktur. Seznam jednotlivých struktur je uveden v tabulce 2.6[12].

Tabulka 2.6: Objekty definující mapování procesních dat

název	**parametry**	**popis**
ULOI_PICO	pole "[ ]u2,u2,u2,u2"	mapování vstupních PDO zpráv
ULOI_POCO	pole "[ ]u2,u2,u2,u2"	mapování výstupních PDO zpráv
ULOI_PIOM	bajtové pole "[ ]u1"	mapování metadat
ULOI_PEV2C	pole "[ ]u2,u2,u2,u2"	mapování PDO událostí na CIDy
ULOI_PDC2EV	pole nespecifikováno	mapování změn dat
ULOI_PMAP_CLEAR	příkaz "e"	vymazání mapování PDO zpráv
ULOI_PMAP_SAVE	příkaz "e"	uložení mapování PDO zpráv

Základním stavebním kamenem jsou takzvané PICO (PDO input CID/OID) a POCO (PDO output CID/OID) tabulky. Obě dvě tabulky mají stejný formát. Každý mapovací uzel je definovaný čtveřicí dvoubajtových integerů bez znamének a propojuje číslo kanálu (CID) s jednotlivými prvky z objektového slovníku podle OIDu. První dva bajty udávají číslo komunikačního kanálu, ze kterého má modul přijímat zprávy. Druhé dva bajty upřesňují jak se bude zacházet s následujícími čtyřmi bajty. V dalších dvou bajtech je uloženo číslo OIDu nebo číslo udávající bajtový offset ukazující na pole z tabulky ULOI_PIOM (PDO input/output mapping metadata). Tento způsob mapování umožňuje serializaci více dat určených jedním CIDem. V posledních dvou bajtech je informace o délce předchozích dat nebo metadat. Tyto jsou informace shrnuty v tabulce 2.7[13].

[12] http://ulan.sourceforge.net/
[13] http://ulan.sourceforge.net/

Tabulka 2.7: Mapovací uzel PICO/POCO tabulek

název	**typ**	**popis**
CID	u2	identifikační číslo kanálu
flg	u2	upřesňující flag
meta/OID	u2	OID nebo bajtový offset v ULOI_PIOM
meta len	u2	délka metadat nebo délka OIDu

2.5.3 Vyvolávání událostí

V tabulce ULOI_PEV2C (PDO event to CID mapping) je uvedeno, která čísla událostí (EvNum) odpovídají kterým číslům kanálů (CID). Jedno číslo události může být namapováno na více komunikačních kanálech a naopak. Každý uzel je opět definovaný čtveřicí dvoubajtových integerů (viz. tabulka 2.8[14]). První *integer* určuje číslo události, se kterým se má zpráva aktivovat. To zajišťuje volání funkce *uloi_evarr_set_ev*. Druhý *integer* udává, do kterého kanálu se zpráva má emitovat. Třetí číslo určuje adresu zařízení, kterému se má zpráva poslat. Pokud adresu nastavíme nulovou, bude se zpráva vysílat formou *broadcastu*. Poslední dva bajty udávají upřesňující příznak.

Tabulka 2.8: Mapovacího uzlel PEV2C tabulky

název	**typ**	**popis**
EvNum	u2	číslo vyvolané události
CID	u2	číslo kanálu
DAdr	u2	cílová adresa
Flg	u2	upřesňující příznak

[14] http://ulan.sourceforge.net/

Kapitola 3

QML

3.1 Obecné informace o QML

QML [3] je zkratkou vycházející z celého názvu *Qt Meta Language* nebo *Qt Modeling Language*. *QML* je součástí *Qt Quick*, který patří do vývojářského balíčku Qt Framework od firmy Nokia. Jedná se o deklarativní jazyk popisující vzhled a chování uživatelských aplikací. Hlavní použití *QML* je v oblasti tvorby aplikací pro mobilní telefony.

Uživatelské rozhraní *QML* je definováno jako strom objektů. Každému elementu je možné nadefinovat celou řadu vlastností. Funkční chování *QML* zajišťuje JavaScript. K jednotlivým elementům můžeme přistupovat prostřednictvím C++ komponent naimplementovaných v Qt Frameworku nebo měnit jejich vlastnosti. Modul poskytující funkce pro práci s QML se jmenuje *Qt Declarative*.

3.2 Základní synaxe jazyku QML

```
import Qt 4.7
Rectangle {
    width: 200
    height: 200
    Image {
      source: "pics/logo.png"
      anchors.centerIn: parent
} }
```

Prvním příkazem `import Qt 4.7` vkládáme modul *Qt Quick*, který obsahuje standartní QML elementy. Bez tohoto importu by nefungovaly následující příkazy. V kódu se vytváří dva objekty (typu obdélník a obrázek). Definice jednotlivých objektů jsou uzavřeny do složených závorek. Mezi ně se definují vlastnosti (*property*) objektů pomocí syntaxe `property: hodnota`. Každá *property* musí být umístěna na novém řádku nebo musí být oddělena od ostatních středníkem. Objekt `Rectangle{}` má definovanou velikost, barvu a potomka typu obrázek. Objekt `Image{}` má v parametru `source` uvedenou relativní cestu k obrázku vůči umístění QML souboru. Příkaz `anchors.centerIn` zarovnává umístění obrázku do středu svého rodiče (obdélníku). Tímto způsobem můžeme vytvářet libovolně složité grafické objekty.

Komentáře se tvoří stejně jako například v C++ nebo v JavaScriptu. Inline komentář se dělá příkazem `\\` a komentáře obsahující více řádků `\* komentář *\`.

3.3 Identifikace QML objektů

Každému QML objektu můžeme nadefinovat *property* `id: název`. Její hodnota musí být v rámci jednoho QML projektu unikátní. Přes tento název se můžeme na objekt odkazovat a případně měnit jednotlivé *property* objektu. ID je objektu přiděleno při jeho vytvoření a nelze ho změnit za chodu aplikace. Identifikátor musí začínat malým písmenem nebo podtržítkem a nesmí obsahovat jiné znaky než písmena, číslice a podtržítka.

```
import Qt 4.7
Rectangle {
  id: myRectangle
}
```

3.4 Používání výrazů v QML

QML využívá JavaScript. Jakýkoli platný JavaScriptovský výraz může být přidělen libovolné *property*. Výrazy mohou obsahovat odkazy na jiné vlastnosti. V příkladu níže uvedeném jsou do sebe vnořené dva obdélníky. Velikost potomka je závislá na velikosti rodiče, takže při změně velikosti rodiče se automaticky změní i velikost potomka. Na rodiče se

můžeme odkazovat buď pomocí jeho ID, nebo přes klíčové slovo *parent*.

```
import Qt 4.7
Rectangle {
  id: myParentRectangle
  Rectangle{
    width: myParentRectangle.width - 100
    \\ width: parent.width - 100
    height: 50 * 10 - 200
  }
}
```

QML podporuje mnoho různých datových typů, včetně těch základních jako jsou *string*, *real*, *int* a *bool*. QML *property* jsou takzvaně "type-safe", což znamená, že do *property* typu integer můžeme přiřadit pouze data stejného typu. V opačném případě dostaneme chybu.

3.5 Ostatní vlastnosti

QML obsahuje velké množství dalších vlastností, o kterých se v této práci nebudu dále rozepisovat. Zmíním pouze ty, jež byly použity v rámci diplomové práce.

3.5.1 Stavy

Každý objekt použitý v QML může mít nadefinován libovolný počet různých stavů. Uživatelské aplikace mohou být navrženy tak, že stavy jedněch objektů ovlivňují stavy jiných objektů. Vezměme například dopravní semafor, který má tři světla. Každá barva má dva stavy, rozsvíceno a zhasnuto. Semaforu můžeme nastavit takové chování, že pokud se zhasne žluté a zelené signalizační světlo, rozsvítí se červené. Zároveň můžeme říct, že žluté světlo se může rozsvítit, pouze pokud jsou ostatní světla zhasnutá.

Konfigurace stavů jsou definovány v rámci objektu `states:[ ]`. Jednotlivé stavy pak udáváme pomocí klíčového slova `State{}`.

Je možné nastavit následující chování:

- zobrazit nebo schovat různé objekty
- spustit, pozastavit nebo zastavit animaci
- zavolat skript při změně stavu
- změnit hodnotu nějaké *property*
- zobrazit jiný obrázek

Každý stav musí mít unikátní jméno datového typu string. Pokud stav chceme změnit, do property `state` přiřadíme požadovaný identifikátor. Pro konkretizaci představy uvedu příklad definice obdélníku se stavy "GREEN"a "RED", které mění jeho barvu.

```
Rectangle {
 id: myRect
 state: "GREEN"
 states: [
     State {
         name: "GREEN"
         PropertyChanges { target: myRect; color: "green"}
     },
     State {
         name: "RED"
         PropertyChanges { target: myRect; color: "red"}
     }
  ]
}
```

3.5.2 Animace

Animaci objektů vytvoříme přiřazením animujícího elementu do jeho *property* v závislosti na typu chování, které je zapotřebí. Mezi tyto elementy patří například průhlednost, hladký přechod, plynulá rotace a další.

3.5.3 Signály a sloty

QML zdědil systém signálů a slotů, který je vytvořen pomocí Qt Frameworku v C++. Signály umožnují informovat ostatní objekty o tom, že se stala nějaká událost. Například objekt *MouseArea* vyšle ostatním objektům signál *clicked*, který informuje o tom, že na danou oblast bylo kliknuto. Nový signál se deklaruje následující syntaxí.

```
signal <jméno>[([<datový typ> <jméno parametru>[, ...]])]
```

Prvním znakem názvu signálu musí být malé písmeno. Není možné deklarovat dva signály se stejným jménem v rámci jednoho objektu. QML umožnuje přetížit již existující signály a nastavit jim novou funkčnost. Vytvořením signálu je také automaticky vytvořen slot, který se jmenuje `on<JménoSignálu>`. Každá *property* má dvojici signál `<JménoSignálu> Changed` a slot `on<JménoSignálu>Changed`. Tento signál je vyslán vždy, když se změní hodnota *property*.

Vyslání signálu je možné voláním metody objektu. Propojení signálu a slotu se řídí podle stejných pravidel jako předchozí případy. Při zavolání metody s parametry je emitován signál, který parametry předá obslužnému slotu. Výše popsané chování je patrné z následujícího příkladu. Signál `sendMessage` je emitován po úspěšné inicializaci objektu. Signál obslouží slot `onSendMessage` a vypíše do konzole přijatou zprávu.

```
Rectangle {
  id: myRect
  signal sendMessage(string message)

  onSendMessage: {
    console.log(message)
  }
  Component.onCompleted: myRect.sendMessage("Ahoj světe!!!")
}
```

QML dále umožňuje propojování signálů metodou `connect()` s JavaScriptovskými metodami nebo jinými signály. Každá JavaSkriptovská metoda deklarovaná v QML se chová jako Qt slot. Propojení signál-signál je možné libovolně řetězit.

Vzhledem k tomu, že QML používá Qt Framework, je možné propojit signály deklarované v QML kódu se signály a sloty deklarovanými v C++ kódu a naopak. To byl také hlavní důvod k tomu, proč byl pro vývoj jednotlivých zařízení vybrán Qt Framework.

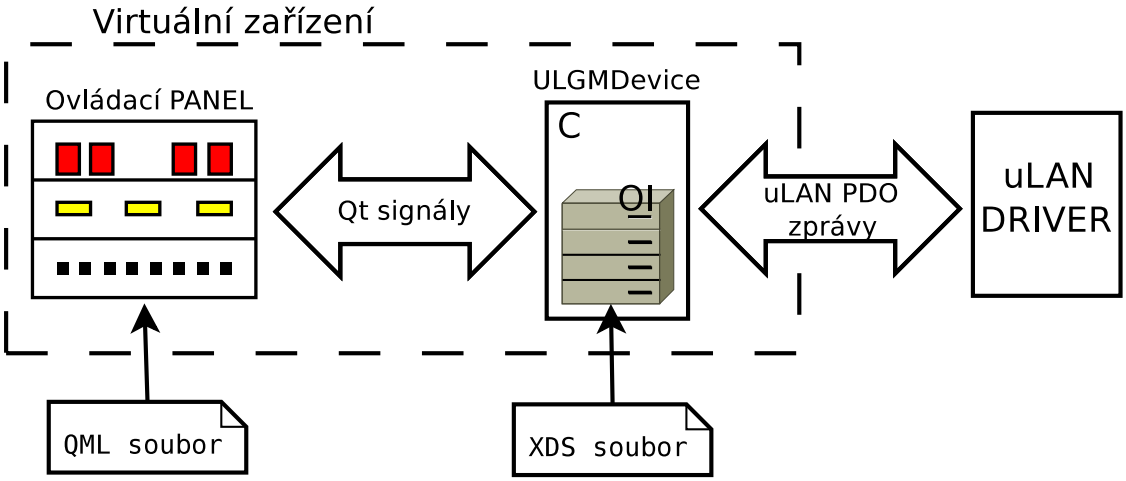
Kapitola 4

Realizace projektu μLAN-GenMod

μLAN-GenMod je aplikace, která umožnuje připojit virtuální moduly na sběrnici μLAN. Každý modul je definován dvojicí souborů. Grafická reprezentace virtuálního zařízení je popsána v QML souboru. V XDS souboru je uloženo nastavení potřebné pro připojení zařízení do sítě a vytvoření objektového slovníku.

4.1 Architektura virtuálního zařízení

Aplikace se skládá z několika částí. Jejich struktura je názorně ukázána na obrázku 4.1[1].



Obrázek 4.1: Architektura virtuálního zařízení

[1]Zdroj: vlastní zpracování

Jednu část tvoří vrstva napsaná v jazyce C, která umožnuje dynamické vytváření objektů v objektovém slovníku. Třída implementující tuto vrstvu se jmenuje *uloi_dyndev_base*. *Uloi_dyndev* vrstva realizuje veškerou nízkoúrovňovou μLAN komunikaci a je totožná ve virtuálním i ve fyzickém zařízení. To zaručuje stoprocentní přenositelnost konfigurace virtuální sítě do sítě reálné.

Další vrstvu tvoří *Qt wrapper*. *Qt wrapper* je naimplementovaný v jazyce *C++* ve třídě *ULGMDevice*. Vrstva zajištuje Qt objektům možnost přístupu k μLAN funkcím vrstvy *uloi_dyndev_base* a poskytuje jí objektový slovník, ke kterému vrstva *uloi_dyndev_base* přistupuje pomocí *getter* a *setter* funkcí.

Nejvyšší vrstvu tvoří model fyzického rozhraní virtuálního zařízení realizované pomocí *QML DeclarativeView*. Tato vrstva skrze mechanismus *signal-slot* komunikuje s vrstvou ULGMDevice a zprostředkovává jí vnější technologický proces. Jedná se například o změnu teploty, zásah uživatele (zapnutí vypínače), ovládání motorů nebo ovládání displeje zařízení.

O propojení jádra zařízení a grafické reprezentace zařízení se stará třída *ULGMDeviceWidget*, která sdružuje výše uvedené vrstvy.

Jednotlivé vrstvy jsou podrobněji popsány v následujících podkapitolách.

4.1.1 Vrstva uloi_dyndev_base

Vrstva *uloi_dyndev_base* (μLAN object interface dynamic device) dovoluje jednotlivým zařízením definovat objektový slovník ve chvíli, kdy už je zařízení připojeno v μLAN síti. Tuto možnost původní verze μLANu nepodporovala. Před připojením nového zařízení do sítě bylo nutné objektový slovník předem nadefinovat ve zdrojovém kódu.

Reálné μLAN zařízení postavené na firmwaru *uloi_dyndev_base* se bude chovat stejně v rámci μLANu sítě jako virtuální zařízení, pokud bude mít stejnou konfiguraci. Tím je mezi nimi zajištěna přenositelnost konfigurace.

Třídu *uloi_dyndev_base* naimplementoval Ing. Pavel Píša Ph.D.

4.1.2 ULGMDevice vrstva

ULGMDevice vrstva obsahuje třídu *ULGMCoreDevice*, ve které je implementovaný most mezi *Qt* objekty a *C* funkcemi.

Zařízení musí splňovat následující požadavky:

- musí se umět připojit do sítě
- musí si umět načíst objektový slovník
- musí mít nadefinované obslužné funkce, pro přístup nižší vrstvy k objektovému slovníku

Splnění prvního požadavku zajišťuje funkce `connectToNet`, která se volá s parametry `dev_name`, `module_no`, `subdevice_no` a `dev_label`. První parametr udává, které zařízení se má k síti připojit. Druhý parametr určuje, jakou bude mít virtuální zařízení adresu. Třetí parametr určuje číslo submodulu. μLAN driver, který je zavedený v jádře, může obsluhovat pouze jedno virtuální zařízení s konkrétní adresou. Toto zařízení pak zprostředkovává zprávy pro jednotlivé submoduly. Navenek se submoduly chovají jako samostatná zařízení s adresou udávající právě číslo submodulu. Čtvrtý parametr je název, pod kterým se bude zařízení prezentovat v síti.

Druhý požadavek splňuje funkce `addProperties`. Funkci se předává list objektů typu *ULGMProperty*. Každá *property* nese informace načtené z XDS souboru (viz kapitola 4.1.5). Samotný objektový slovník je tedy uložen v rámci vrstvy *ULGMDevice*. Jak již bylo zmíněno v kapitole 2.3.3, každá *property* v objektovém slovníku musí mít nadefinované obslužné funkce podle toho, jestli je *property* určená pro zápis, nebo pro čtení a nebo pro oboje. Přes obslužné funkce k objektům přistupuje nižší vrstva *uloi_dyndev_base*.

Při dynamickém vytváření objektů mohou mít jednotlivé *property* různé datové typy. μLAN-GenMod aplikace zachází se všemi *property* jako s datovým typem *QVariant*, a z toho důvodu musely být naimplementovány obecné funkce pro zápis a pro čtení. Funkce pro zápis se jmenuje `general_wrfnc` (*setter*) a funkce pro čtení se jmenuje `general_rdfnc` (*getter*). Výše zmíněné funkce zajišťují splnění třetího požadavku.

Funkce se volají, pokud je ze strany μLANu požadavek na zápis nebo na čtení *property* v objektovém slovníku. Funkce `general_wrfnc` emituje signál `setValueUL`, který dá virtuálnímu zařízení informaci o změně hodnoty *property*. V rámci těchto funkcí se zároveň provádí přetypování přenášených dat. Data musí do μLANu vstupovat s konkrétními datovými typy, které byly popsány v kapitole 2.4. Data vstupující do virtuálních zařízení musí být datového typu *QVariant*. Veškeré nutné informace o skutečných datových typech lze získat introspekcí proměnné typu *QVariant*.

Knihovna zajišťující samotné přetypování se jmenuje *libulqconv* a byla naimplementována kolegou Bc. Martinem Boháčkem. Knihovna *libulqconv* definuje třídu *ULQDataTypeFactory*. Třída obsahuje funkce, které převádějí data mezi μLAN formátem a *QVariantem*. Funkce `u2q` slouží pro převod binárních dat μLANu do *QVariantu* a obrázený převod zajišťuje funkce `q2u`. Tabulka 4.1[2] znázorňuje namapování jednotlivých datových typů.

Tabulka 4.1: Tabulka mapování datových typů

datový typ μLAN	**datový typ C++/Qt**
s1 s2 s4	int
u1 u2 u4	unsigned int
f2 f4 f8	double
vs	QString
e	QVariant::Invalid
PICO	ULQ_PICO
POCO	ULQ_POCO
PEV2C	ULQ_PEV2C

[2]zdroj: Bc. Martin Boháček

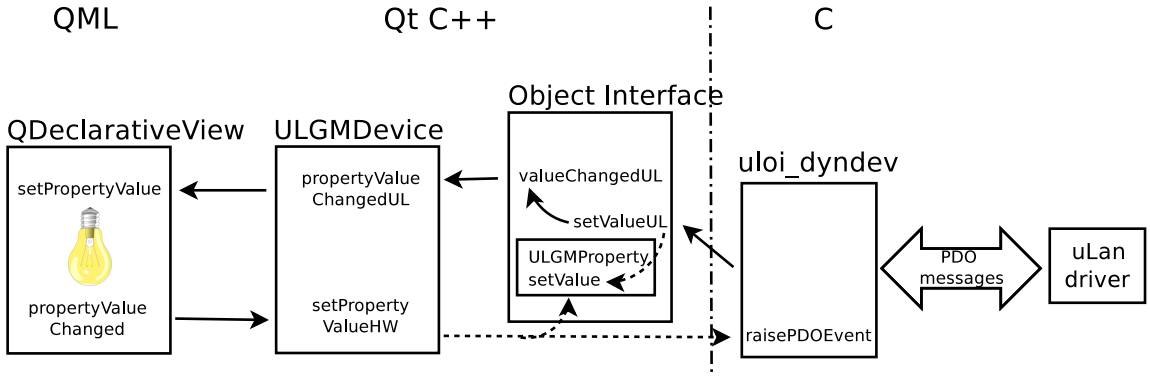
4.1.3 ULGMDeviceWidget vrstva

Vrstva *ULGMDeviceWidget* používá pro zobrazení *QML* instanci třídy *QDeclarativeView*, kterou spojuje s vrstvou *ULGMDevice*. V aplikacích, které používají *QML*, je nutné k načtení a spuštění *QML* dokumentů zavolat *QML runtime*, který je součásti *UI Declarative* balíčku [3]. To lze provést prostřednictvím třídy *QDeclarativeView* (zobrazovací třída) a třídy *QDeclarativeEngine* (provádí potřebné výpočty).

Každé virtuální zařízení má definovanou *inline* funkci `view`, která vrací proměnnou typu *QDeclarativeView*. Do proměnné se příkazem `setSource` načte *QML* kód. *QML* kód je zobrazen v podokně MDI widgetu, který je popsán v kapitole 4.2.2. Funkce `setResizeMode` zajistí, že se velikost *QML* bude přizpůsobovat velikosti podokna. Příklad implementace zobrazující *QML* kód v *MDI* podokně je uveden níže.

```
ULGMDeviceWidget* mywidget = new ULGMDeviceWidget();
QMdiSubWindow *subwin = ui->mdiArea->addSubWindow(mywidget);
mywidget->view()->setResizeMode(QDeclarativeView::SizeRootObjectToView);
mywidget->view()->setSource(QUrl::fromLocalFile(file_name));
subwin->show();
```

Struktura propojení jednotlivých vrstev je podrobně zobrazena na obrázku 4.2[3]. Plné čáry znázorňují spojení typu *signal-slot* a přerušované čáry znázorňují volání funkcí.



Obrázek 4.2: Propojení jednotlivých vrstev

Funkce, která propojí *ULGMDevice* a *QDeclarativeView*, se jmenuje `setGMDevice`. Funkce projde všechny objekty v *QDeclarativeView* a zjistí, jestli zařízení má deklarovány nějaké signály nebo sloty. Pokud je v *QML* implementován signál `propertyValueChanged`,

[3]Zdroj: vlastní zpracování

zavolá se funkce `connect`, která ho propojí se slotem `setPropertyValueHW` třídy *ULGM-Device*. Signál se emituje ve chvíli, kdy uživatel provede v zařízení nějakou změnu (například zapne vypínač). Pokud je v *QML* nadeklarována funkce (slot) `setPropertyValue`, funkce `connect` ho propojí se signálem `propertyValueChangedUL`. Obě dvě propojení se provedou na základě shodného jména *QML* objektu (viz. kapitola 3.3) a názvu proměnné v objektovém slovníku.

Funkce `setPropertyValueHW` vykoná dvě operace. Nejprve se funkcí `setValue` nastaví aktuální hodnota *ULGMProperty* v objektovém slovníku vrstvy *ULGMDevice*. Poté se informace o změně stavu musí zpropagovat do μLANu. Funkce `raisePdoEvent` nastaví *PDO* zprávě číslo události a funkcí `processPdoEvents` se zpráva odešle na μLAN.

Zprávy, které přijdou z μLANu, obsluhují *gettery* a *settery* popsané v kapitole 4.1.2. *Setter* emituje signál `setValueUL`, který prostřednictvím funkce `setValue` nastaví aktuální hodnotu *ULGMProperty*. Každá *ULGMProperty* má definovaný signál `valueChangedUL`, který je při vytváření objektového slovníku funkcí `connect` připojen k jednomu signálu `propertyValueChangedUL` zařízení *ULGMDevice*. Jinými slovy při vytvoření objektového slovníku vznikne spojení signálů všech *ULGMProperty* s jedním signálem zařízení *ULGM-Device*. Tento signál, jak již bylo uvedeno výše, je propojen se slotem `setPropertyValue`.

4.1.4 XDS - XML Device Description

XDS je soubor, ve kterém jsou uloženy informace pro připojení zařízení do sítě a konfigurace objektového slovníku. V každém XDS souboru musí být mezi párovými tagy `<device>` následující informace:

- adresa zařízení
- jméno, pod kterým bude zařízení v síti vystupovat
- jednotlivé *property*

Property se do objektového slovníku vkládají mezi párové tagy `<property>`. Každá *property* musí obsahovat:

- OID
- jméno
- datový typ proměnné
- inicializační hodnotu

Property mohou být dvojího typu (pro zápis a pro čtení). Pokud se jedná o *property*, která bude na síť vysílat informaci o svém stavu, musí mít navíc uvedeno číslo události (event ID, viz. kapitola 2.5.3). V aktuální implementaci může být číslo události maximálně 200. Formát *XDS* souboru je popsán *RELAX NG* schématem (viz. níže).

```
element device {
 element addr { xsd:integer },
 element devname { xsd:string },
 element property {
   element oid { xsd:integer },
   element name { xsd:string },
   element type { xsd:string },
   element value { text },
   element eventid { xsd:integer }?
 }*
}
```

4.2 Implementace virtuálních zařízení

Balíček *UI Declarative* obsahuje nástroj *Qt QML Viewer*, který zobrazí QML kód. Nástroj byl použit pro vývoj a testování *QML* kódu. Uživatel se při implementování nového *QML* kódu vyhne psaní C++ aplikace a načítání *QML runtime*.

Pro účely diplomové práce byla v QML navržena čtyři virtuální zařízení, jejichž popis je uveden v následujících podkapitolách.

4.2.1 Zařízení žárovka

Zařízení žárovka (lamp) je pasivní prvek sítě μLAN. Má pouze dva stavy (svítí, nesvítí) a slot *setPropertyValue*, který mezi těmito stavy přepíná. Když přijde povel k přepnutí žárovky, funkce *setPropertyValue* zjistí aktuální stav zařízení a vymění zdrojový obrázek odpovídající požadovanému stavu (viz. následující zdrojový kód).

```
function setPropertyValue(prop_name, val) {
   if (prop_name == oidName) {
      if(val == 1) lamp.state = "on";
      else lamp.state = "off";
   }
}
```

Vzhled zařízení žárovka je na ukázán na obrázku 4.3[4].

Obrázek 4.3: Zařízení žárovka

4.2.2 Zařízení vypínač

Zařízení vypínač (switch) je aktivní prvek μLAN sítě. Od zařízení žárovka se liší pouze tím, že místo slotu má definovaný signál *propertyValueChanged*. Při kliknutí na aktivní oblast vypínače se stanou tři události. Změní se aktuální stav vypínače, vymění se zdrojový obrázek aktivní části vypínače a emituje se signál.

[4] http://openclipart.org/

Definice signálu a funkce, která ho emituje, vypadají následovně:

```
signal propertyValueChanged(variant name, variant val)
function change_state() {
   if (mySwitch.state == "on") {
      mySwitch.state = "off";
   } else {
      mySwitch.state = "on";
   }
   propertyValueChanged(mySwitch.ul_oidName, mySwitch.state == "on"?1:0);
}
```

Aktivní část obrázku se definuje objektem *MouseArea*, který pak má vlastnost *onClicked* (viz. zdrojový kód níže).

```
Image {
   source: "switch_off.png"
   MouseArea {
      onClicked: mySwitch.change_state()
   }
}
```

Vzhled zařízení vypínač je na ukázán na obrázku 4.4[5].

Obrázek 4.4: Zařízení vypínač

[5]http://openclipart.org/

4.2.3 Zařízení stmívač

Zařízení stmívač (slider) je aktivní prvek, který v signálu *propertyValueChanged* posílá informaci o své aktuální poloze. Rozsah vysílaných hodnot je od 0 do 100. Aktivní část se opět definuje objektem *MouseArea* s následujícími vlastnostmi.

```
MouseArea {
   drag.target: parent; drag.axis: Drag.XAxis
   drag.minimumX: 2; drag.maximumX: container.width - 32
}
```

Pokud myší změníme polohu *slideru*, obslouží se vlastnost `onXChanged`, která podobně jako u zařízení vypínač vyvolá signál funkcí *change_state()*.

Vzhled zařízení stmívač je na ukázán na obrázku 4.5.

Obrázek 4.5: Zařízení stmívač

4.2.4 Zařízení multimetr

Zařízení multimetr je pasivní prvek, který reprezentuje analogový budík. Úhel natočení ručičky zobrazuje aktuální hodnotu. Grafická reprezentace je rozdělena na dvě části. První je statický obrázek se stupnicí čísel a druhý je obrázek jehly, kterou podle aktuální hodnoty rotujeme kolem určeného bodu. Funkčnost zařízení opět zajištuje funkce *setPropertyValue* a vlastnosti obrázku jehly. Vlastnost zajištující rotaci obrázku se definuje příkazem `transform: Rotation {}`. Zde musí být uveden úhel natočení a souřadnice středu otáčení. Pro efektnější vzhled je změně úhlu natočení přiřazena *Spring* animace, která pohybu jehly zaručí překmitnutí s definovaným tlumením.

Kód zajišťující tuto funkčnost je uveden níže.

```
transform: Rotation {
   id: needleRotation
   origin.x: needle.width/2; origin.y: needle.height/10
   angle: meter.value + 52
   Behavior on angle {
      SpringAnimation {
         spring: 1.4
         damping: .15
      }
   }
}
```

Vzhled zařízení multimetr je na ukázán na obrázku 4.6[6].

Obrázek 4.6: Zařízení multimeter

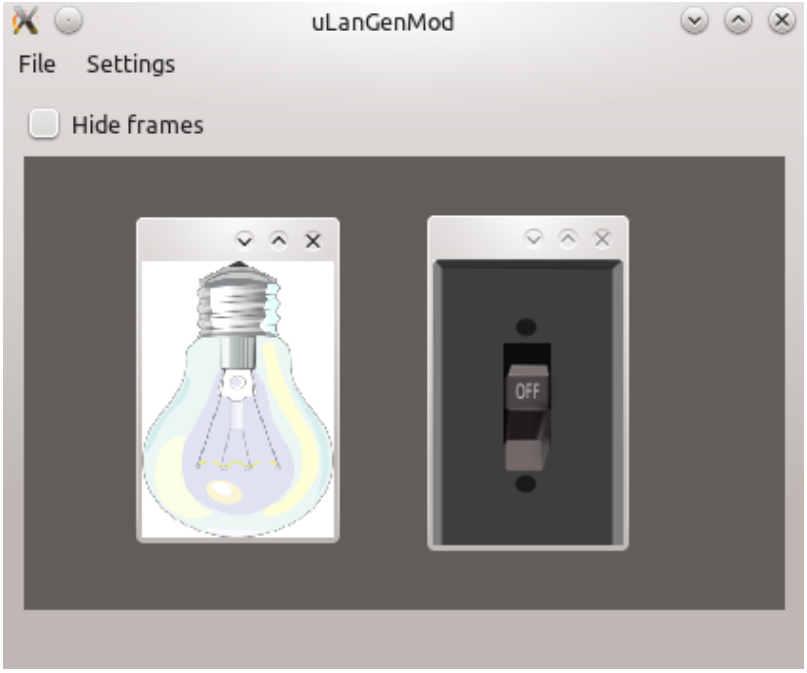
4.3 Popis aplikace μLAN-GenMod

4.3.1 Hlavní okno

Hlavní okno obsahuje menu a MDI oblast. Hlavní menu obsahuje základní funkce pro ovládání programu. V záložce *Open* najdeme funkce pro otevření QML souboru, XDS souboru, XML konfigurace sítě a obrázku, který se zobrazí na pozadí. XDS soubor je možné otevřít, pouze pokud je vybrané některé z aktivních podoken.

[6] http://openclipart.org/

Záložka *Save* obsahuje funkci pro uložení aktuálního stavu sítě virtuálních zařízení. V záložce *Close* je funkce *All Devices*, která zavře všechna otevřená podokna, a funkce *Background*, která odstraní aktuálně zobrazené pozadí. Funkce *Quit* ukončí aplikaci.



Obrázek 4.7: Hlavní okno aplikace μLAN-GenMod

4.3.2 MDI - Multiple Document Interface

Třída *QMdiArea* umožňuje zobrazení a správu více podoken v rámci jedné oblasti. V aplikaci μLAN-GenMod je *MdiArea* umístěna do centrální části hlavního okna. Podokna v *MDI* oblasti lze libovolně pozicovat (například kaskádovitě nebo dlaždicovitě). Podokna jsou datového typu *QMdiSubWindow*. Nová podokna se do oblasti přidávají metodou `addSubWindow`. Každé podokno obsahuje záhlaví, vnitřní ovládací prvky a okenní rám, který lze funkcí *Hide Frames* schovat. V každém podokně je zobrazen objekt třídy *ULGMDeviceWidget*, který reprezentuje μLAN zařízení, jehož vlastnosti jsou specifikovány v *XDS* a *QML* souborech. Konkrétní implementace virtuálních μLAN zařízení jsou popsány v kapitole 4.1.6.

Na obrázku 4.8[7] je ukázka rozmístění zařízení v MDI oblasti.

Obrázek 4.8: MDI oblast s virtuálními zařízeními

4.3.3 Konfigurace sítě

Aplikace μLAN-GenMod ukládá nebo načítá konfiguraci sítě z nebo do *XML* souboru. Zde se ukládají informace o každém otevřeném podokně v MDI oblasti mezi párové tagy `<Subwins>`. Konfigurace jednotlivých podoken je uložena mezi párovými tagy `<Subwin0>`, kde číslo vyskytující se v tagu udává pořadové číslo podokna dané konfigurace.

Každé podokno si dále ukládá následující informace:

- velikost
- pozici
- cestu ke QML souboru
- cestu ke XDS souboru

[7]zdroj: vlastní zpracování

Cesty k oběma souborům se ukládají relativně vůči domovskému adresáři. Při otevírání konfigurace se cesta k souboru složí z relativní cesty a z cesty `"ROOT_PATH"` uložené v *INI* souboru (viz. kapitola 4.2.3). Příklad *XML* konfigurace sítě se dvěmi podokny je uvedena níže.

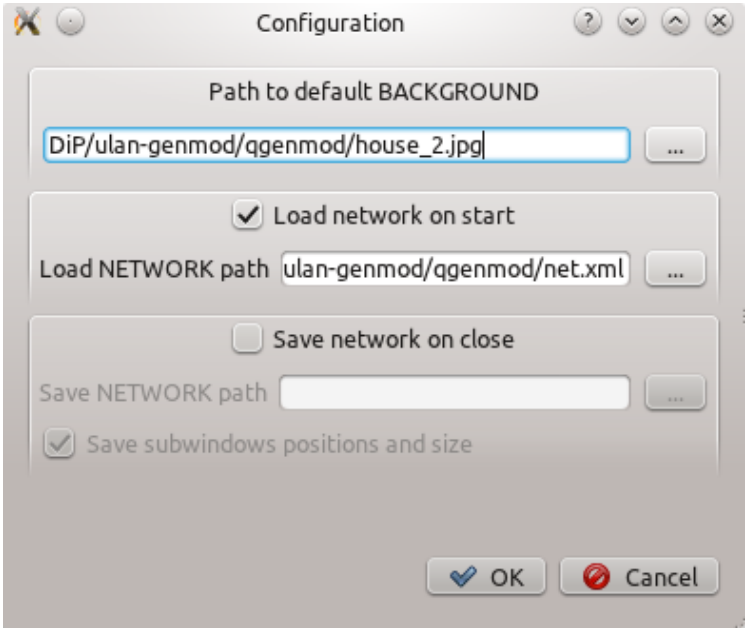
```
<?xml version="1.0" encoding="UTF-8"?>
<Subwins>
  <Subwin0>
    <height>172</height>
      <width>100</width>
      <posx>7</posx>
      <posy>7</posy>
      <qmlpath>ulansrc/ulan-genmod/qgenmod/components/lamp/...
                                        ...Lamp.qml</qmlpath>
      <xdspath>ulansrc/ulan-genmod/qgenmod/components/lamp/...
                                        ...Lamp.xds</xdspath>
  </Subwin0>
  <Subwin1>
      <height>206</height>
      <width>125</width>
      <posx>113</posx>
      <posy>6</posy>
      <qmlpath>ulansrc/ulan-genmod/qgenmod/components/switch/...
                                        ...Switch.qml</qmlpath>
      <xdspath>ulansrc/ulan-genmod/qgenmod/components/switch/...
                                        ...Switch.xds</xdspath>
  </Subwin1>
</Subwins>
```

4.3.4 Konfigurace programu

V menu *Settings* najdeme funkci *Configure*, která vyvolá dialogové okno. V něm můžeme nastavit chování aplikace. Do pole *Path to default BACKGROUND* se nastavuje cesta k obrázku který se po spuštění programu zobrazí na pozadí. Pokud zaškrtneme volbu *Load network on start* a zadáme cestu ke XML konfiguraci sítě, tak se po spuštění aplikace

načte zvolená konfigurace. Volba *Save network on close* po zavření aplikace ukládá stav otevřených podoken do zvoleného souboru. Zaškrtnutím *check boxu* můžeme zvolit, jestli chceme do konfigurace ukládat i pozice jednotlivých podoken.

Qt Framework má pro ukládání nastavení aplikace třídu *QSettings*. Třída *QSettings* implicitně ukládá tyto informace v operačním systému *Windows* do registrů, v operačním systému *Mac OS* do XML souboru předvoleb a v operačním systému *UNIX* se volby ukládají do *INI* souborů. V operačním systému *UNIX* jsou konfigurační soubory obvykle uloženy v domovském adresáři v souboru */.config/uLan/ULGMApplication.ini*. Vzhled konfiguračního dialogového okna je ukázán na obrázku 4.9[8].



Obrázek 4.9: Konfigurační dialogové okno

Deklarace objektu *QSettings* a načtení a uložení jednoho parametru se ve zdrojovém kódu implementuje následovně:

```
QSettings settings(QSettings::IniFormat, QSettings::UserScope,...
                                                ... _PROJECT, _APP);
settings.setValue("ROOT_PATH", "/home/user/");
settings.value("ROOT_PATH").toString();
```

Konstanta `_PROJECT` koresponduje s názvem adresáře v adresáři */.config* a konstanta `_APP` odpovídá názvu *INI* souboru, do kterého se volby ukládají. Vstupní parametr

[8]zdroj: vlastní zpracování

`QSettings::IniFormat` explicitně říká, že se bude konfigurace na všech platformách ukládat stejně.

Do konfiguračního souboru se při prvním spuštění aplikace uloží cesta domovského adresáře. Ostatní cesty se pak ukládají relativně vůči domovskému adresáři. Při zavření aplikace se do *INI* souboru uloží aktuální velikost hlavního okna.

Pro představu uvádím příklad *INI* souboru.

```
[General]
SAVE_NW_ON_CLOSE=N
NW_SAVE_PATH=
LOAD_NW_ON_START=Y
NW_LOAD_PATH=ulansrc/ulan-genmod/qgenmod/net.xml
SAVE_SUBWIN_POS=Y
BG_LOAD_PATH=ulansrc/ulan-genmod/qgenmod/house_2.jpg
MAINWINDOW_HEIGHT=447
MAINWINDOW_WIDTH=932
ROOT_PATH=/home/stefic/
```

Kapitola 5

Instalace

V této kapitole je krok po kroku uvedeno jak nainstalovat μLAN s aplikacemi μLAN-GenMod a μLAN-Admin. Návod byl vytvořen na operačním systému Linux a vůči domovskému adresáři */home*. Verze operačního systému, na němž byl návod testován je *Ubuntu 11.10* a verze jádra *3.0.0.14-generic*, což je 32-bitový operační systém.

- první krok je instalace potřebných balíčků
- druhý krok je stažení projektu μLAN
- třetí krok je příprava prostředí pro kompilaci
- čtvrtý krok je konfigurace μLAN driveru
- pátý krok je načtení μLAN driveru do jádra operačního systému
- šestý krok je stažení a instalace aplikace μLAN-Admin
- sedmý krok je stažení a instalace aplikace μLAN-GenMod

Celá sekvence níže zmíněných příkazů je k dispozici ve skriptu *ulan_build_script*, který je na přiloženém CD v příloze B.

5.1 Potřebné balíčky

V prvé řadě je nutné stáhnout následující instalační balíčky. Celý projekt je verzován v *Gitu* a sestaven *gcc* kompilátorem, který je obsažen v balíčku *build-essetial*.

Pro správné přeložení jsou potřeba následující knihovny:

- linux-headers-generic
- libqt4-dev
- libqjson-dev

Tyto kroky zařídí následující příkaz:

```
sudo aptitude install git build-essential linux-headers-generic ...
                        ... libqt4-dev libqjson-dev libxml++1.0-dev
```

5.2 Stažení projektu μLAN

Další krok je stažení projektu *μLAN protocol for RS-485 9-bit network*, který je uložen na webu *www.sf.net*. Hlavní projekt stáhneme příkazem `git clone`. Protože hlavní projekt v sobě má vnořené další repozitáře, musíme pro jejich stažení jěště aktualizovat celý strom projektu příkazem `git update`. To vše provedeme násedující sérií příkazů:

```
mkdir ulansrc; cd ulansrc/
git clone git://ulan.git.sourceforge.net/gitroot/ulan/ulan
cd ulan/
git submodule update --init
cd ..
```

5.3 Příprava prostředí pro kompilaci

Skript *build-ulan-host*, který se nachází v adresáři `/ulan/scripts/`, připraví prostředí pro přeložení projektu. Projekt se tak nebude kompilovat přímo mezi zdrojové kódy, a tím si zachová přehlednost. Skript vytvoří nový adresář `/ulan-build`, s potřebnými symbolickými linky. Protože celý projekt využívá pokročilý sestavovací systém OMK, je v rámci skriptu vytvořen konfigurační soubor *config.omk* Skript se pouští následujícím příkazem:

```
ulan/scripts/build-ulan-host
```

5.4 Konfigurace μLAN driveru

Pro správnou kompilaci se v konfiguračním souboru *config.omk* musí nastavit následující flagy:

- *CONFIG_OC_UL_DRV_WITH_VIRTUAL* zajistí, že se driver přeloží s podporou virtuálních zařízení
- *CONFIG_OC_ UL_DRV_WITH_MULTI_DEV* zajistí, aby jedno virtuální mohlo k μLAN síti připojit více submodulů
- *CONFIG_OC_UL_DRV_WITH_IAC* povoluje podporu okamžité identifikace (command UL_CMD_SID) a dynamické adresace (command UL_CMD_SNST)
- *CONFIG_ULOI_CON_IO_OPS* povolí volbu způsobu načítání dat a metada (využití v PICO a POCO tabulkách a v PDO zprávách) pro interpretaci sekvencí a pro manipulaci s daty v objektovém slovníku
- *CONFIG_ULAN_DY* zajistí povolení služby dynamické adresace (UL_CMD_SNTS) a Network Control Services (UL_CMD_NCS)
- *CONFIG_ULOI_LT* zajistí povolení kompilace knihoven podpory uLOI
- *-fPIC* je nutné nastavit pro správné přeložení na 64-bitových operačních systémech

Seznam příkazů je uveden níže:

```
cd ulan-build/host/
echo 'CONFIG_OC_UL_DRV_WITH_VIRTUAL=y' > config.omk
echo 'CONFIG_OC_UL_DRV_WITH_MULTI_DEV=y' >> config.omk
echo 'CONFIG_OC_UL_DRV_WITH_IAC=y' >> config.omk
echo 'CONFIG_ULOI_CON_IO_OPS=y' >> config.omk
echo 'CONFIG_ULAN_DY=y' >> config.omk
echo 'CONFIG_ULOI_LT=y' >> config.omk
echo 'CFLAGS += -fPIC' >> config.omk
make
```

5.5 Načtení μLAN driveru do jádra OS

Nejprve je potřeba zkopírovat přeložený driver na místo, kde o něm bude jádro vědět, a zinicializovat moduly příkazem `depmod`. V souboru *10-ulan.rules* jsou pravidla pro zavedení driveru a pro vytvoření symbolického linku na driver. Symbolický link je nutný, aby uživatel mohl do driveru zapisovat a používat ho. Příkaz `modprobe` zavede driver do jádra. Parametr `chip=virtual` určuje, že jsou podporována virtuální zařízení. Pokud bychom zaváděli driver pro komunikaci s embedded moduly, tak bychom parametr `chip=virtual` vynechali.

```
sudo mkdir /lib/modules/`uname -r`/extra/
sudo cp _compiled/modules/ul_drv.ko /lib/modules/`uname -r`/extra/
cd ../..
sudo depmod -a
sudo bash -c 'echo SUBSYSTEM==\"ulan\",GROUP=\"plugdev\",MODE=\"0660\"
                                ... > /etc/udev/rules.d/10-ulan.rules'
sudo bash -c 'echo KERNEL==\"ulan0\",SUBSYSTEM==\"ulan\",SYMLINK+=
                      ... \"ulan\" >> /etc/udev/rules.d/10-ulan.rules'
sudo modprobe ul_drv chip=virtual
```

5.6 μLAN-Admin

Aplikace μLAN-Admin napsaná kolegou Martinem Boháčkem slouží pro monitorování a správu μLAN sítě. Protože se v rámci této aplikace kompiluje knihovna *libulqconv*, kterou využívá aplikace μLAN-GenMod, je nutné μLAN-Admin aplikaci přeložit jako první. Spustitelný binární kód se nachází v adresáři *ulan-admin/_compiled/bin*.

```
git clone git://ulan.git.sourceforge.net/gitroot/ulan/ulan-admin
cd ulan-admin
qmake -r
make
cd ..
```

5.7 μLAN-GenMod

Stažení a zkompilování μLAN-GenMod aplikace provedeme následující sekvencí příkazů. Spustitelný binární kód se nachází v adresáři *ulan-genmod/_compiled/bin*.

```
git clone git://ulan.git.sourceforge.net/gitroot/ulan/ulan-genmod
cd ulan-genmod/qgenmod
make
cd ../..
```

5.8 Wikipedie

V rámci diplomové práce byla k projektu vytvořena wikipedie, kterou je možné nalézt na následující webové adrese:

`http://sourceforge.net/apps/mediawiki/ulan`

Pro tvorbu wikipedie byla použita *open source* aplikace *MediaWiki*, která je napsaná v jazyce PHP. Na této webové stránce je možné si stáhnout instalační skript *ulan_build_script*.

Kapitola 6

Nasazení projektu

6.1 Praktická ukázka použití

Protokol μLAN je použit v následujících aplikacích.

6.1.1 Model automatizovaného domu

Model automatizovaného domu byl vytvořen na Katedře řídicí techniky FEL ČVUT. Jednotlivé komponenty pro model dodala firma *MIDAM*. Model se skládá z několika žárovek, vypínačů, modelu topení a senzorů detekujících stav oken (otevřené/zavřené). Model demonstruje interakce mezi jednotlivými zařízeními. Model je ukázán na obrázku 6.1.

6.1.2 μLAN-Admin

μLAN-Admin je sada knihoven poskytujících přístup k proměnným definovaných v reálných nebo virtuálních μLANních zařízeních. Základ tvoří knihovna *libulproxy*, která umožňuje tunelovat μLAN komunikaci přes TCP protokol, což poskytuje možnost dálkové správy μLAN sítě.

Součástí μLAN-Admin je aplikace *ulanbrowser*, která poskytuje informace o všech zařízeních připojených na stejné síti. Aplikace umožňuje prohlížení a editování proměnných uložených v objektových slovnících jednotlivých zařízení. *Ulanbrowser* obsahuje funkce pro ukládání a načítání aktuální konfigurace μLAN sítě (PICO, POCO a PEV2C tabulek). Vzhled aplikace je ukázán na obrázku A.1 v příloze.

Obrázek 6.1: Model automatizovaného domu

6.2 Real-Time Linux Workshop

V Praze v prostorách Fakulty elektrotechnické ČVUT v Dejvicích se konal 20.-22.10.2011 třináctý ročník *Real-Time Linux Workshopu* (RTLW) (viz. obrázek A.2 v příloze). Konference byla zaměřená především na průmyslové projekty založené na Linuxovém jádře. Sešlo se zde hlavní jádro vývojářů, které realizuje úpravy jaderných subsystémů poskytujících jádru real-time vlastnosti.

S kolegou Martinem Boháčkem jsme se konference účastnili jako spoluautoři článku *Process Data Connection Channels in μLAN Network for Home Automation and Other Distributed Aplications*. Na konferenci jsme dělali doprovodnou předváděcí akci k projektu *μLAN protocol for RS-485 9-bit network*, který prezentoval v rámci *RTLW* doktor Pavel Píša. Spolu s modelem automatizovaného domu jsme předváděli aplikace μLAN-Admin a μLAN-GenMod (viz. obrázek A.3 v příloze). I přes to, že zařízení použitá v modelu automatizovaného domu měla poměrně složité objektové slovníky, podařilo se nám zprovoznit komunikaci mezi zařízeními reálnými a zařízeními virtuálními.

6.3 Dokumentace

Veškeré informace o projektu *µLAN protocol for RS-485 9-bit network* jsou k dispozici na webové stránce `http://ulan.sourceforge.net/`. Celý projekt je open source pod *LGPL* licencí, takže se dá použít i pro komerční využití.

Další užitečné odkazy:

- `http://sourceforge.net/projects/ulan/` - domovská stránka projektu
- `http://ulan.git.sourceforge.net/` - repozitář projektu
- `http://cmp.felk.cvut.cz/~pisa/` - domovská stránka Pavla Píši
- `http://sourceforge.net/apps/mediawiki/ulan` - wikipedie k projektu

Kapitola 7

Závěr

Práce na projektu pro mě byla velkým přínosem. Při realizaci projektu jsem se seznámil s vývojovým prostředím *Qt* a jazykem *QML*. Vývojem softwaru jsem si zopakoval práci s verzovacím systémem *Git* a práci se sázecím systémem LaTeX[6]. Při tvorbě dokumentace jsem se seznámil s tvorbou wikipedie.

Podařilo se mi vytvořit aplikaci, která umožňuje připojit virtuální zařízení na sběrnici μLAN. Pro tvorbu aplikace jsem použil *Qt Framework*, který mi práci na projektu značně usnadnil. Virtuální zařízení jsou postavena na stejném *firmwaru* jako zařízení reálná, takže konfigurace je mezi zařízeními přenositelná. Vytvořil jsem čtyři vzorová virtuální zařízení popsaná ve funkcionálně deklarativním jazyce *QML*. Ke každému virtuálnímu zažízení jsem vytvořil konfigurační soubor XDS (XML Device deScription), díky němuž je teď možné virtuálním zařízením definovat objektové slovníky bez znalosti programovacího jazyka C. V rámci této práce jsem vytvořil podrobný návod, podle kterého si každý uživatel může protokol μLAN pohodlně zprovoznit na svém počítači.

Myslím si, že jsem splnil všechny cíle vytyčené v úvodu této práce a že jsem dodržel formální zadání diplomové práce.

Vzhledem k tomu, že v současné době neexistuje reálné zařízení postavené na firmwaru *uloi_dyndev_base*, nebylo možné přenostitelnost konfigurace reálně vyzkoušet. Proto by dalším logickým krokem měla být výroba zařízení postaveného na tomto *firmwaru*. Přenositelnost konfigurace je předpokládána, protože se původní *firmware* liší od nového pouze vrstvou *uloi_dyndev_base*. Aplikace μLAN-GenMod by se dala rozšířit o možnost generování mapovacího konfiguračního souboru. Tento soubor by musel být kompatibilní s aplikací *ulanbrowser*. Mapování jednotlivých zařízení by se mohlo provádět obdobně jako tomu je například u aplikace *Simulink*. Uživatel by se tím vyhnul vyplňování *PICO* a *POCO* tabulek.

Za odměnu považuji to, že jsme se s kolegou Bc. Martinem Boháčkem, mohli účastnit mezinárodní konference *13th Real Time Linux Workshopu* jako spoluautoři článku *Process Data Connection Channels in μLAN Network for Home Automation and Other Distributed Aplications*. V rámci konference jsme předvedli naše práce a jejich interakci s modelem automatizovaného domu, který nám zapůjčila Katedra řídicí techniky FEL ČVUT.

Mé přání je, aby byl projekt v praxi využitelný a aby si protokol μLAN vybudoval srovnatelnou pozici v průmyslu nebo v elektroinstalacích inteligentních budov jako ostatní úspěšné protokoly.

Literatura

[1]ECKEL, Bruce. (2000). *Myslíme v jazyku C++*. Praha: GRADA Publishing, spol. s.r.o. ISBN 80-247-9009-2

[2]PÍŠA, Pavel. *ul_drv - uLan RS-485 Communication Driver*. [cit. 2011-12-10]. [on-line]. Dostupné z: http://ulan.sourceforge.net/

[3]Kolektivní autorství. *Qt Reference Documentation*. [cit. 2011-12-20]. [on-line]. Dostupné z: http://doc.qt.nokia.com/

[4]TIŠNOVSKÝ, Pavel *Sběrnice RS-422, RS-423 a RS-485*. [cit. 2011-12-18]. [on-line]. Dostupné z: http://www.root.cz/clanky/sbernice-rs-422-rs-423-a-rs-485/

[5]Kolektivní autorství. *Wikipedia*. [on-line]. Dostupné z: http://en.wikipedia.org/

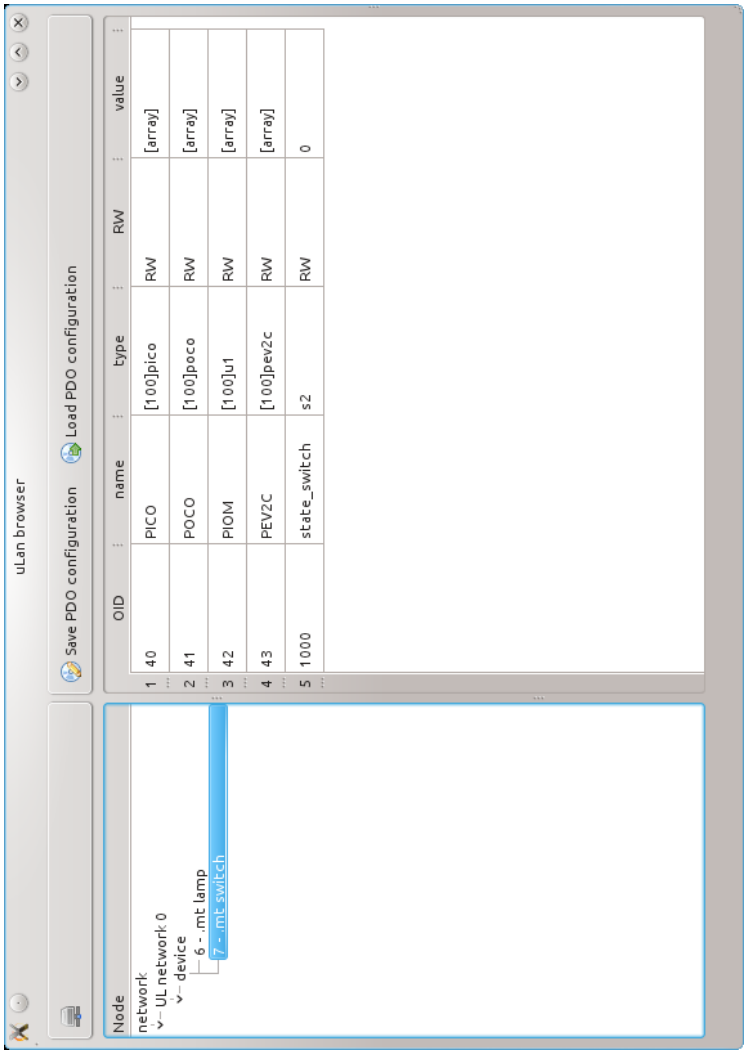
[6]OLŠÁK, Petr. (2000). *Typograficky system TeX*. Praha: KONVOJ. ISBN 80-85615-91-6

[7]Kolektivní autorství (2011). *Proceedings of the 13th Real Time Linux Workshop*. Praha: Open Source Automation Development Lab (OSADL) eG ISBN 978-3-0003-6193-7

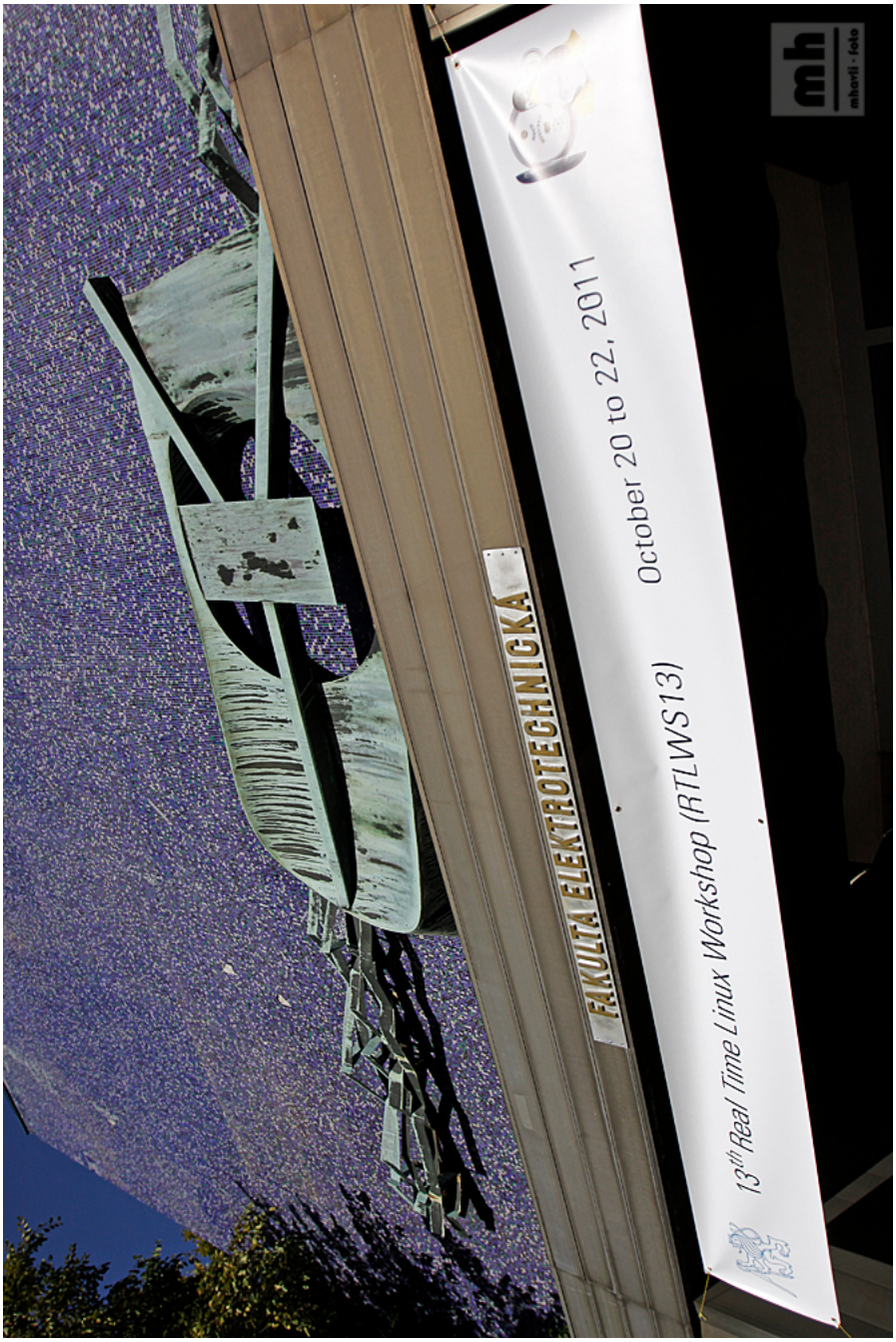
[8]ŠTEFAN, Jan. (2009). *Návrh čidla pro řízení hydroponického systému*. Praha: bakalářská práce

Příloha A

Obrázky



Obrázek A.1: Aplikace μLAN-Browser



Obrázek A.2: 13th Real Time Linux Workshop

Obrázek A.3: Martin Boháček a Jan Štefan na RTLWS

Příloha B

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém je uložen skript *ulan_build_script* a *RELAX NG* schéma v souboru *xds_schema.rng*.