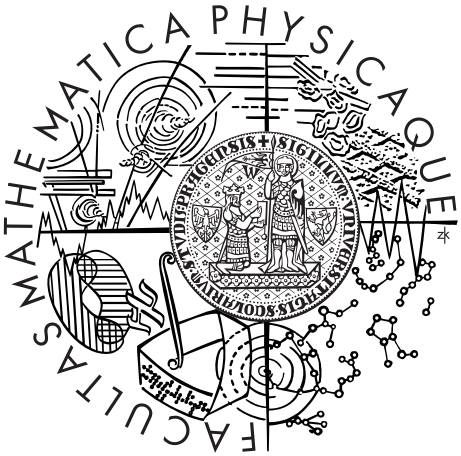
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Pavel Taufer

Domácí automatizace

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Mareš, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2015

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

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

V ..... dne ..... Podpis autora

Název práce: Domácí automatizace

Autor: Pavel Taufer

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Mareš, Ph.D., Katedra aplikované matematiky

Abstrakt: Navrhujeme jednotný systém pro správu a monitorování domácnosti – například ovládání světel, termostatu, žaluzií a podobně. Zkoumáme různé způsoby komunikace se zařízeními používanými v automatizaci domácnosti a navrhujeme jednotné rozhraní pro komunikaci s těmito zařízeními nezávisle na konkrétním hardware. Součástí práce je také ukázková implementace této abstrakce, sada konkrétních automatizačních programů a grafické uživatelské rozhraní.

Klíčová slova: domácí automatizace, monitorování, dálkové řízení

Title: Home automation

Author: Pavel Taufer

Department: Department of Applied Mathematics

Supervisor: Mgr. Martin Mareš, Ph.D., Department of Applied Mathematics

Abstract: We design an unified system for home management and monitoring – for example controlling lights, thermostat, roller blinds, etc. We research various methods of communication with devices used in home automation. We design an unified interface for communication with said devices independent on specific hardware. Then we implement the designed abstraction, a set of specific automation software and a graphical user interface.

Keywords: home automation, monitoring, remote control

Rád bych zde poděkoval panu Mgr. Martinu Marešovi, Ph.D. za jeho rady a čas, který mi věnoval při řešení dané problematiky.

Obsah

Úvod

V této práci se hodláme zabývat automatizací přístrojů v domácnosti. V dnešní době je to poměrně široká oblast. Lidé už zkoušeli automatizovat prakticky cokoliv, co lze připojit k elektřině. Existuje více směrů, kterými se automatizace ubírá: jedná se primárně o aspekt praktičnosti (garáž se automaticky zavře po zaparkování auta, robotický vysavač vysaje psí chlupy), dále aspekt zabezpečení (kamery, alarmy, detektory kouře) a v neposlední řadě aspekt zábavy a pohodlí (multimediální systémy, možnost si vzdáleně nahrát film v televizi). V dnešní době je obzvláště populární si například dálkově zapnout saunu či vyhřívání vířivky, to je však již „luxusní“ nadstavba nad obvyklými použitími.

Pojďme se na několik aplikací automatizace podívat konkrétněji. Vhodným příkladem na úvod je ovládání osvětlení. Samotných světel je dnes mnoho druhů, ať už co se týče barev, či schopnosti plynulého stmívání. Spínače mohou být rozmístěny různě po místnostech. Je zde mnoho možností, jak propojit světlo se spínači. Klasické zapojení obsahuje jeden spínač a jedno či více světel, přičemž světla svítí, právě když je spínač sepnutý. Dále je možné použít křížové spínače, které umožní ovládat konkrétní světlo z více míst. Není ale potřeba zůstávat pouze u spínačů, je možné používat i třeba tlačítka. Zde je již potřeba o něco složitější elektronika, kde stisk tlačítka mění stav světla a jeho podržením lze upravovat intenzitu světla. Světlo může samo zhasnout po určité době od stisku (schodišťový automat). Pro předchozí využití se také občas používají místo tlačítek detektory pohybu. Hezkým příkladem na závěr je spínání světel na základě intenzity venkovního osvětlení.

Další oblastí pro automatizaci je termoregulace. Po domě mohou být rozmístěna teplotní čidla a měřit teploty v jednotlivých místnostech. Na základě jednotlivých teplot pak může být ovládán kotel či jednotlivá topení. Také bývá možné si nastavit různé teploty podle denní doby nebo dne v týdnu. Rovněž je možno přidat čidla do oken, která budou kontrolovat, jestli je okno otevřené či zavřené. To lze použít pro prevenci případného teplotního úniku v chladných měsících.

Obecně řízení vícero zařízení může být složité a vzájemně propojené, například při nastavení domácnosti na noční režim či dovolenou. Je tedy na místě použít na ovládání jednotlivých zařízení počítač s patřičnými programy, který bude mimo jiné dobře konfigurovatelný.

Zkusme o těchto aplikacích nyní uvažovat abstraktněji. Základní nástroje používané k domácí automatizaci se dají zařadit do následujících kategorií:

Čidlo je vstupem řídicího systému. Ať už se jedná o teplotní čidlo či spínač, je to zařízení snímající stav reálného světa a tento stav v digitální podobě předává dál.

Aktor je zařízení, které fyzicky vykoná příkaz přijatý od systému (ovladač žaluzií, vnitřní spínač v lampě).

Ovladač se stará o ovládání aktorů na základě informací od čidel. Může se jednat jak o samostatné specializované zařízení, tak například o počítačový program.

Čidla a aktory budu ve zbytku práce označovat souhrnně jako *prvky*. Systém domácí automatizace se většinou skládá z prvků rozmístěných v domácnosti a programů, které je ovládají. Ovládání samotné může být ruční, kdy jsou aktory přímo ovládány uživatelem, nebo automatické, kdy program na základě informací od čidel aktory ovládá či komunikuje s jinými programy.

Do kategorie ručního ovládání spadá například možnost přímo rozsvěcet světla či otevřít garážová vrata. K tomu může být například použito grafické rozhraní schopné zobrazit aktuální stav a ovlivnit ho.

Mezi kandidáty na automatické ovládání spadá například termostat, který bude spouštět ventilátory v jednotlivých topeních podle teplotních čidel rozmístěných po domě. Dalšími příklady jsou ovládání světel, které by podporovalo ovládání z více míst či automatické zhasnutí, hlídač pohybu, který pošle SMS v případě, dojde k detekci pohybu v zamčeném domě, či třeba kontrola zavření oken, která upozorní posledního odcházejícího člověka, že se v domě nachází otevřené okno.

Existuje mnoho druhů prvků, které se liší nejen funkcí, ale také tím, jak se s nimi komunikuje. To komplikuje situaci například ve chvíli, kdy máme teplotní čidla od různých výrobců a chceme, aby je uměl sledovat jeden program vykonávající funkci termostatu.

Nacházíme se v situaci, kdy máme mnoho prvků, které chceme propojit s různými programy. Zde přichází na řadu tato práce, jejímž cílem je navrhnout a implementovat systém, který by dokázal poskytnout abstraktní pohled na jednotlivé prvky (a fyzická zařízení na různých sběrnicích). Systém by dále měl nabízet možnost připojit další programy, například pro vizualizaci stavu zařízení či řízení zařízení. Měl by být ovladatelný z libovolného množství míst, a to ručně, nebo automaticky. Stavíme zde vlastně analogii operačního systému, který má ovladače na různé druhy hardware (prvků, zařízení, sběrnic) a poskytuje aplikacím jejich abstrakci.

1. Přehled protokolů

Téma domácí automatizace není v dnešní době ničím novým. Existuje mnoho systémů, protokolů a přístupů k domácí automatizaci. V této kapitole podáme přehled těch nejběžnějších z nich. Budeme postupovat od druhů bitového přenosu, přes způsoby správy sběrnice, základní komunikační rozhraní pro spojení dvou a více zařízení, až po protokoly zajišťující i například spolehlivost přenosu či směrování.

1.1 Propojení čidel a aktorů

V úvodu jsme ukázali, že v domácnosti je mnoho příležitostí k umístění různých čidel/aktorů. Nastává tak otázka, jak jednotlivá čidla ovládat. Základní možnost by byla vést ke každému čidlu separátní pár vodičů z centrálního místa. Tento přístup je ale nevhodný z mnoha důvodů, ať již kvůli množství vodičů nebo potížím s napájením kvůli odporu způsobeným délkou vodičů.

Pokud je možné zasahovat do struktury zdí, je možné předem rozvést po celé domácnosti sadu vodičů (např. ethernetové kabely), které následně mohou být využity jako fyzická vrstva pro některý z protokolů pro komunikaci více zařízení. Jednotlivá čidla se následně mohou připojit buď přímo k vodičům sběrnice, dokáže-li čidlo komunikovat protokolem dané sběrnice, nebo je možné použít další zařízení, které umožňuje spravovat jedno či více čidel a zároveň dokáže komunikovat po sběrnici.

Jestliže není možno zasahovat do zdí, lze použít bezdrátové technologie. I zde platí, že můžeme využít hardware, který zprostředkuje jeden bezdrátový modul více čidlům.

1.2 Druhy bitového přenosu

Pro libovolný datový přenos je důležité, aby odesílatel a příjemce postupovali stejně rychle už na nejnižší úrovni. Obecně se tento problém nazývá synchronizace. V případě bitových datových přenosů se jedná o synchronizaci na úrovni jednotlivých bitů [7].

Můžeme předpokládat, že jak odesílatel, tak příjemce mají k dispozici stejně jdoucí hodiny. Pomocí těchto hodin by oba byli schopni při přenosu určit, kolikátý bit je zrovna přenášen. Problém je, že při vyšších rychlostech se mohou začít projevovat drobné odchylky hodin příjemce a odesílatele, např. přečtením jednoho bitu dvakrát či jeho přeskočením.

1.2.1 Synchronní přenos s hodinami

Nejjednodušším řešením je posílat paralelně s daty „hodinový“ signál odesílatele. Příjemce následně využívá tento signál pro čtení dat na ostatních vodičích. Nevýhodou je nutnost použití dalšího vodiče. Takovému způsobu se říká synchronní přenos s hodinami.

1.2.2 Asynchronní přenos

Dalším způsobem přenosu bitu je asynchronní přenos. Odesílatel přímo sděluje příjemci informaci o začátku vysílání bitu. Díky této informaci může být doba vysílání jednoho bitu různě dlouhá, takže se může měnit i přenosová rychlost a odesílatel ani příjemce nemusí mít přesně stejně jdoucí hodiny. Tento typ přenosu vyžaduje alespoň tři různé stavy přenášeného signálu.

1.2.3 Arytmický přenos

Možným kompromisním řešením je očekávat od hodin odesílatele i příjemce dostatečně přesný chod hodin po dobu přenosu několika jednotek až desítek bitů. Potom je možné přenášet data po skupinách bitů s danou velikostí. Na začátek každé skupiny odesílatel umístí značku (tzv. start bit). Ta je určena pro příjemce k „seřízení“ jeho hodin a následně odešle celou skupinu. Je na příjemci, aby za pomoci svých hodin rozpoznal příjem jednotlivých bitů. Odesilatel může pro kontrolu na závěr přenosu uvést tzv. paritní bit pro kontrolu správnosti přenesených bitů a celý přenos ukončit další značkou (tzv. stop bit).

V praxi se tato metoda používá obzvláště v případech, kdy přenášená data mají podobu znaků. Arytmický přenos umožňuje odesílat data s libovolnými mezerami mezi znaky, postrádá tedy pevně daný rytmus a proto je a-rytmický. Termín „arytmický“ se v běžné odborné praxi nepoužívá. Místo něj se pro tento popsaný způsob používá také označení „asynchronní“.

1.2.4 Samo-synchronní přenos

Další možností je synchronizace příjemce ze samotných dat. V případě, že jsou bity reprezentovány změnou signálu (hranou), dá se tato hrana využít i pro synchronizaci hodin příjemce. Je však potřeba zajistit dostatečně častý výskyt hran. Pokud existují sekvence znaků, kde by hrozila de-synchronizace, je možné do ní vložit navíc takový bit, který zajistí synchronizaci. Příjemce pak tento bit bude při čtení obsahu zprávy ignorovat.

1.3 Komunikace více zařízení

V případě, že chceme komunikovat pouze mezi dvojicí zařízení, stačí je propojit datovými vodiči a následně využít některou metodu bitového přenosu. Pokud však chceme, aby mohlo komunikovat více zařízení mezi sebou, je situace složitější. Principiálně nejjednodušším řešením je propojit přímými spoji všechny dvojice zařízení, která spolu mají komunikovat. V tomto řešení je nevýhodou například až kvadratický počet spojů při narůstajícím počtu zařízení, popřípadě přehlcení některých zařízení.

V dalších příkladech budeme předpokládat, že všechna zařízení jsou připojena k jednomu komunikačnímu kanálu. Tomuto kanálu budeme říkat sběrnice. V případě, že se na sběrnici odesílá data více zařízení naráz, dochází ke kolizi a přenášená data mohou být poškozena. Kolizím se buďto můžeme snažit zabránit nebo naopak navrhnout sběrnici tak, aby se z nich uměla zotavit.

Jedním z preventivních řešení je zajistit, aby v jednu chvíli mohlo na sběrnici vysílat pouze jedno zařízení. Toto zařízení se také označuje jako *master* a ostatní zařízení na sběrnici jsou označována jako *slave*. Komunikaci může iniciovat právě master, ostatní zařízení pouze odpovídají na dotazy. Jedna možnost je tzv. polling, kdy se předem určený master opakovaně ptá jednotlivých zařízení na jejich stav.

Masterem přitom nemusí být stále totéž zařízení. Můžeme měnit, které zařízení je master, za použití tzv. *tokenu*. Ten, kdo drží token, je v danou chvíli master a může komunikovat s ostatními. Ve chvíli, kdy nemá potřebu komunikovat, předává token dalšímu zařízení, čímž na toto další zařízení převede status mastera a původní master se stane slavem. Výhodou komunikace master/slave je například jednodušší protokol a nižší nároky na jednotlivá zařízení. Nevýhodou je například to, že token se může ztratit důsledkem šumu či odpojení zařízení. Může také docházet ke zpožďování, například když většina zařízení nemá co říci a pouze předává token dále.

V případě, že umožníme všem zařízením zahájit vysílání na základě vlastního uvážení (tzv. multi-master), může dojít ke kolizím. Většinou zařízení čeká na klidový stav na sběrnici, než začne vysílat. Může ale dojít k tomu, že dvě či více zařízení zahájí vysílání ve stejný okamžik. Další postup závisí na schopnosti čtení sběrnice zařízením ve chvíli kdy samo vysílá.

Jeden z postupů pro řešení kolizí je tzv. *bitová arbitrace*. Aby jí bylo možné použít, musí sběrnice splňovat následující vlastnosti.

- Všechna zařízení dokáží přečíst hodnotu sběrnice dříve, než dojde k vyslání dalšího bitu.

- Sběrnice musí mít 2 stavy: *dominantní* (např. „0“) a *recesivní* (např. „1“). Pokud alespoň jedno zařízení vysílá dominantní hodnotu, je na sběrnici dominantní stav, jinak je na sběrnici stav recesivní.

Pro odeslání zprávy čekají všechna zařízení na klidový stav na sběrnici. Poté začnou odesílat bitovou reprezentaci svého identifikátoru od nejvýznamnějšího bitu. Jednotlivá zařízení kontrolují stav na sběrnici při odesílání. Pokud se liší od odesílaného bitu, přeruší odesílání současné zprávy a opakují ho při dalším klidovém stavu na sběrnici. Tímto způsobem bude odeslána vždy právě jedna zpráva po klidovém stavu. V našem případě by tedy vždy „vyhrálo“ zařízení s nejnižším identifikátorem.

Výhodami jsou například absence centrálního zařízení (arbitra) a jednoduchost přidávání zařízení. Nevýhodami jsou například zpoždění odeslání některých zpráv, pokud je na sběrnici příliš velký provoz, nebo omezení délky sběrnice a komunikační rychlosti.

Jestliže nemůžeme využít metodu bitové arbitrace, tak je v případě kolize nutné znovu odeslat všechny zprávy, které byly vyslány přes sebe. Délka prodlevy mezi zprávami záleží na protokolu. Často se používá náhodný čas z postupně se zvětšujícího intervalu, pokud se kolize řetězí.

1.4 Sériová komunikační rozhraní

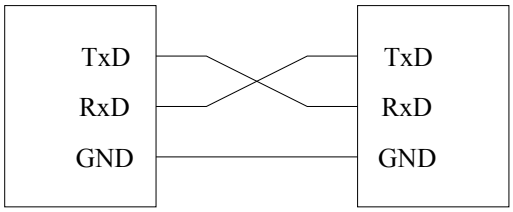
V této podkapitole si představíme několik konkrétních sběrnic používaných v domácí automatizaci.

1.4.1 RS 232

RS 232 je sériové komunikační rozhraní, které umožňuje vytvořit spojení a přenos dat mezi dvojicí lokálních zařízení. Umožňuje synchronní i arytmický přenos dat.

Signály jsou reprezentovány různými napěťovými úrovněmi vzhledem k společnému zemnícímu vodiči. Jednotlivé stavy se pohybují v napětích $-15\,\mathrm{V}$ až $-5\,\mathrm{V}$ pro záporný stav a $5\,\mathrm{V}$ až $15\,\mathrm{V}$ pro kladný stav oproti zemnímu vodiči. Délka vedení je omezená na 30 až 60 m kvůli odporu vedení a nedostatečné ochraně před vnějším rušením. Pro kontrolu správnosti přenosu dat je možné použít paritní bit. Standard popisuje konektor s 25 vodiči, ale existuje více způsobů propojení.

Základní verze používá 3 vodiče: jeden pro společné uzemněni a dva pro přenos dat v jednotlivých směrech. Datové signály se v rámci jednoho zařízení označují „TxD“ pro odchozí data (transmit) a „RxD“ pro příchozí data (recieve). Následné propojení dvou zařízení je vidět na obrázku 1.1. Dále se vyskytují varianty, kde se používá 9 či 10 signálů, které umožňují předávat informaci, je-li například zařízení připraveno ke čtení nebo má data k odeslání. Ve více-signálových variantách je také často odděleno ochranné a signálové uzemnění na samostatné vodiče.

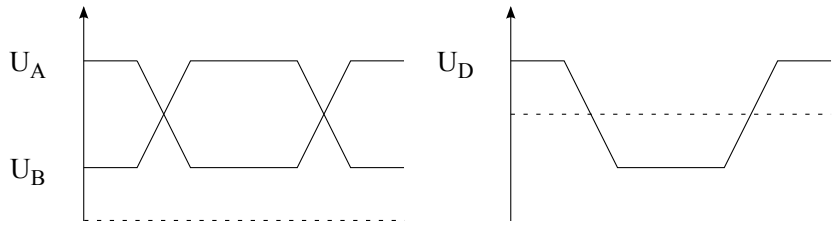


Obrázek 1.1: RS 232 asynchronní přenos přes 3 vodiče

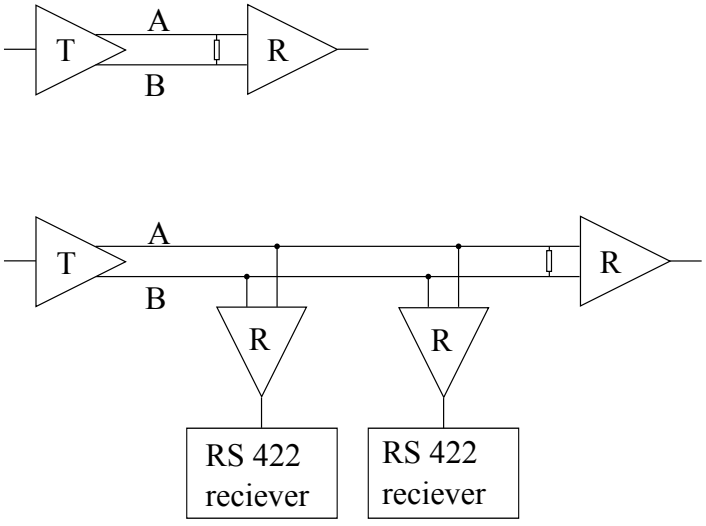
1.4.2 RS 422

Komunikační rozhraní RS 422 vychází z rozhraní RS 232. Cílem bylo vytvořit rozhraní odolnější proti rušení. Díky používání stejného komunikačního protokolu lze používat RS 422 jako prodloužení linky RS 232. Rozdílem oproti RS 232 je, že pro přenos každého signálu je namísto jednoho vodiče použit jeden kroucený pár. Daný signál „D“ je následně přenášený po krouceném páru jako rozdíl potenciálů dané dvojice vodičů „A“ a „B“ (tzv. diferenciální vodiče). Signál D je rozložen mezi dva vodiče A a B tak, aby platilo: $U_D = U_A - U_B$, viz obrázek 1.2. Pro redukci odrazů a šumu je použit terminační odpor spojující vodiče A a B.

Právě díky tomu, že jsou vodiče krouceného páru blízko u sebe, je rušivý vliv okolí na ně téměř stejný, tudíž hodnota rozdílu není rušením ovlivněna. RS 422 navíc také podporuje tzv. multidrop zapojení s až 10 připojenými zařízeními. V tomto zapojení mohou připojená zařízená poslouchat stav na sběrnici, ale nemohou na ni vysílat, viz obrázek 1.3 [5].



Obrázek 1.2: Diferenciální vodiče



Obrázek 1.3: RS 422 multidrop

1.4.3 SPI

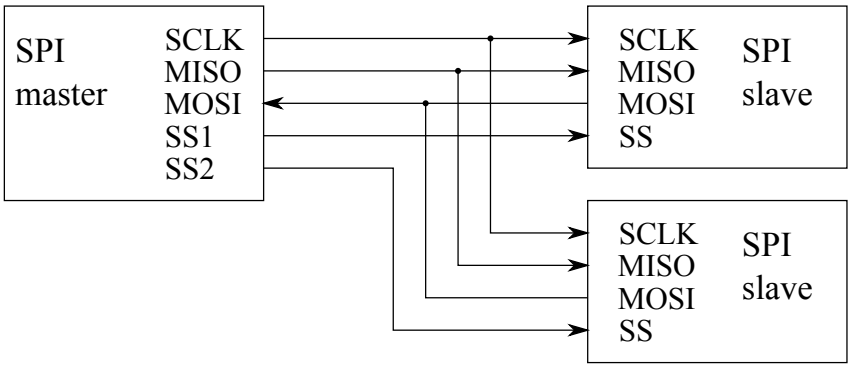
Sběrnice SPI je sériové komunikační rozhraní umožňující obousměrnou komunikaci dvou a více zařízení v master/slave režimu. Master je pevně určen a může komunikovat pouze s jedním slavem najednou.

K přenosu dat jsou užity tři signály: jeden přenáší hodinové pulzy ve směru master → slave (SCLK – serial clock) a zbylé dva přenáší data ve směru master → slave (MOSI – master output, slave input) a master ← slave (MISO – master input, slave output). Tyto tři signály jsou připojeny ke všem zařízením. Aby slave poznal, že data jsou pro něj, vede ke každému slave zařízení od mastera separátní signál (CS či SS – chip select či slave select).

Počet zařízení je tím pádem omezen počtem možných signálů od mastera. Rychlost přenosu udává master pomocí signálu SCLK. Tu je například možné měnit v průběhu přenosu. Slave také nemusí mít tak přesný vlastní hodinový signál.

Výhodou je například rychlejší přenos, jelikož není potřeba data opatřovat hlavičkou o tom, komu jsou určena. Další výhodou je jednoduchost potřebného hardware v porovnání s dalšími master/slave sběrnicemi.

Nevýhodami jsou velký počet signálů u mastera a nemožnost přidání zařízení bez úpravy hardware mastera (přidání dalšího SS signálu). SPI také nezajišťuje kontrolu proti chybám při přenosu a je možné ho používat jen na krátké vzdálenosti.



Obrázek 1.4: Ukázka zapojení sběrnice SPI

1.4.4 RS 485

RS 485 je sériové komunikační rozhraní typu multi-master, umožňující propojit až 32 zařízení. Pro ochranu proti rušení využívá podobně jako RS 422 diferenciální vedení signálu přes kroucený pár. Existují dvě provedení — dvouvodičová a čtyřvodičová verze. Čtyřvodičová verze umožňuje plně duplexní komunikaci, dvouvodičová pouze poloduplexní. Většinou je jedna jednotka učena jako master a řídí ostatní jednotky.

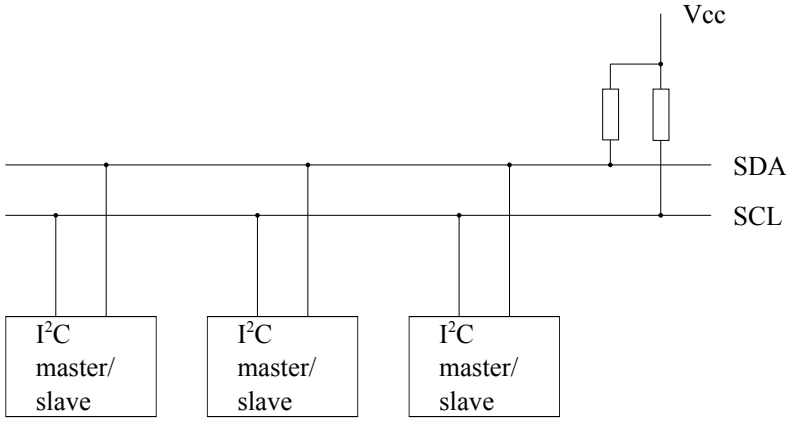
Délka vedení může být až 1200 m a komunikační rychlosti mohou dosahovat až 10 Mb/s. Vedení se nesmí větvit. Jednotlivá zařízení se připojují buďto přímo na samotnou linku, nebo lze použít krátké připojovací kabely. Je možno používat

zařízení na opakování signálu. Dalšího zlepšení kvality signálu lze dosáhnout terminačními rezistory na koncích linky. U delších vedení je maximální přenosová rychlost nižší [5].

1.4.5 I²C

Sběrnice I²C byla vyvinuta počátkem 80. let firmou Philips, původně za účelem snadnějšího propojení procesoru s ostatními obvody v televizoru. Jedná se o synchronní komunikační rozhraní typu multi-master, kdy je možno, aby na jedné sběrnici bylo více zařízení typu master a používá se metoda bitové arbitrace. Krom samotného přenosu dat se stará také o adresaci.

Jednotlivá zařízení jsou připojena na dvojici vodičů SDA (serial data) a SCL (serial clock) a na společné uzemnění, viz obrázek 1.5. Jedná se tedy o synchronní přenos s hodinami. Je potřeba detekovat kolize a následně je řešit. Maximální délka sběrnice se pohybuje v řádu desítek centimetrů [4]. V základu umožňuje adresovat až 112 zařízení pomocí 7-bitového adresování (16 adres je vyhrazeno pro speciální použití), existuje rozšíření přidávající dalších 1024 adres. Výhodou je možnost měnit rychlost přenosu (udávaná pomocí vodiče SCL) a nízký počet vodičů. Původní komunikační rychlost byla omezena na 100 kb/s, posléze vznikla rozšíření umožňující rychlost až 3.4 Mb/s [10].



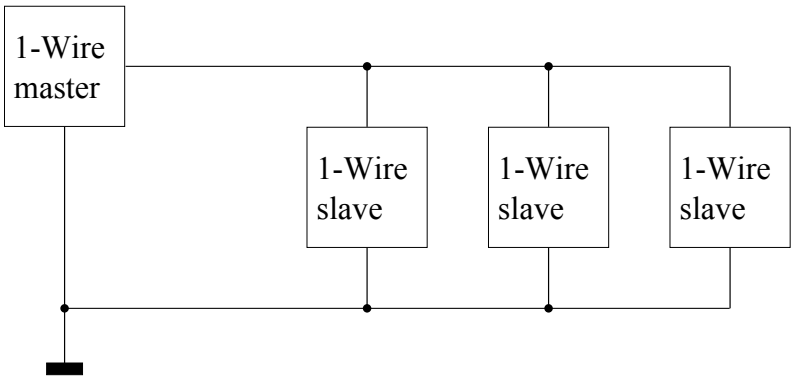
Obrázek 1.5: Ukázka zapojení sběrnice I²C

1.4.6 1-Wire

Sběrnice 1-Wire [3] je komunikační systém založený na master/slave komunikaci za použití jednoho vodiče přenášejícího data i napájení a společného uzemnění. Je vyvíjená firmou Dallas Semiconductor. Síť 1-Wire zařízení je označována MicroLan. Maximální komunikační rychlost je 16.3 kb/s. Je možné komunikovat po vodičích dosahujících vzdáleností i 100 m.

Pro identifikování zařízení se používá 64-bitové sériové číslo určené při výrobě. K zjištění identifikátorů zařízení připojených ke sběrnici se využívá stromové prohledávání adresního prostoru. Master postupně odesílá prefixy adres a všechna slave zařízení s daným prefixem vždy odpoví vysláním logické 0. Ve chvíli, kdy

se žádné zařízení neozve, master ví, že ke sběrnici není připojeno žádné zařízení s tímto prefixem.



Obrázek 1.6: Příklad zapojení 1-Wire

1.4.7 ZigBee

Komunikační protokol ZigBee [6] je standard pro dvousměrnou bezdrátovou komunikaci mezi zařízeními s nízkou spotřebou. Používá se nejen jako jeden ze standardů v domácí automatizaci, ale i v mnoha dalších případech. Fyzická vrstva odpovídá standardu IEEE 802.15.4 . ZigBee dále specifikuje vrstvy od linkové, přes síťovou, až po aplikační. Umožňuje zapojení pomocí hvězdicové, stromové nebo smíšené topologie. Podporuje šifrování za pomocí až 128bitových klíčů. Umožňuje komunikovat rychlostmi až 250 kb/s v pásmu 2.4 GHz, 40 kb/s v pásmu 915 Mhz a 20 kb/s v pásmu 868 Mhz. Používá 64-bitový adresní prostor.

1.4.8 Modbus

Modbus [1] je protokol aplikační vrstvy (7. vrstva OSI modelu), jehož úkolem je zprostředkovávat komunikaci master/slave mezi zařízeními spojenými na různých typech sběrnic a sítí. Dále bude v této práci využíván, proto je popsán podrobněji. Vytvořen byl firmou MODICON v roce 1979 a v praxi je stále používaný. Jde o protokol založený na požadavcích a odpovědích.

Datový model zařízení v protokolu Modbus je složen ze čtyř tabulek podle druhu obsahu: diskrétní vstupy (1 bit, pro čtení), diskrétní výstupy (1 bit, pro čtení i zápis), vstupní registry (16 bitů, pro čtení) a uchovávací registry (16 bitů, pro čtení i zápis).

Adresa	Kód funkce	Data	CRC nebo LRC

Obrázek 1.7: Struktura požadavku Modbus

Na obrázku 1.7 je znázorněn požadavek protokolu Modbus. Kód funkce popisuje akci, jakou má slave určený adresou vykonat. Kódy funkcí jsou z rozsahu 0 až 127, protokol definuje většinu z nich. Nejvyšší bit bajtu pro kód funkce je vyhrazen pro informaci, zda došlo k chybě při zpracovávání požadavku. Pokud zpracování funkce proběhlo bez problémů, odpovídá slave odpovědí skládající se z jeho adresy, kódu vykonané funkce, dat od funkce a kontrolního součtu. V případě chyby při zpracování požadavku slave odesílá adresu, kód funkce zvětšený o 128, kód chyby a kontrolní součet. Je také možné poslat zprávu všem slaveům, na níž není posílána odpověď. Příkladem funkcí je čtení, či zápis jednoho či více diskrétních výstupů či uchovávacích registrů (diskrétní vstupy a vstupní registry mají pouze varianty pro čtení).

Dále umožňuje komunikovat po různých fyzických sběrnicích. Nejrozšířenější je přenos dat za použití sériové linky, ale existují rozšíření umožňujicí komunikaci pomocí TCP/IP nebo bezdrátově.

Variantu využívající sériovou linku je možno provozovat například na sériových rozhraních RS 232, RS 485 a podobných. Umožňuje adresovat až 247 zařízení. Základní přenosové rychlosti jsou 9 600 b/s a 19 200 b/s. Jsou definovány dva režimy komunikace v závislosti na formátu dat posílaných na sběrnici: ASCII a RTU.

- V režimu ASCII je každý bajt požadavku zakódován pomocí dvojice ASCII znaků a pro určení začátku a konce zprávy jsou vymezeny znaky „:“ a dvojce znaků CR, LF. Samotné znaky jsou odeslány jako 7 datových bitů spolu se start bitem, stop bitem a paritním bitem. Zpráva je chráněna kontrolním součtem.

- V režimu RTU je každý bajt požadavku přímo přenášen na sběrnici doplněný o start bit, stop bit a paritní bit, dohromady se pro tuto skupinu 11 bitů používá označení znak. Začátek a konec zprávy jsou určeny klidovým stavem sběrnice po dobu, po kterou trvá odeslat 3.5 znaku. Mezi dvěma znaky nesmí být prodleva větší než doba na odeslání 1.5 znaku. Jako kontrolní funkce se používá CRC.

ASCII režim umožňuje větší prodlevy, ale je také pomalejší, takže se používá spíše RTU.

1.4.9 KNX

KNX je otevřený standard určený pro domácí automatizaci založený na OSI modelu. Vychází z předchozích standardů EHS, BatiBUS a EIB. Neexistuje zde centrální jednotka s programem na spravování sběrnice, ale „program“ je zde rozdělen mezi jednotlivé „inteligentní“ koncové uzly. Jejich interakce poté určuje „program“ dané sběrnice. Existují KNX standardy pro fyzický přenos dat použitím krouceného páru, Ethernetu, infračerveného záření, radiových vln nebo přes elektrické vedení [8].

KNX používá 16-ti bitový adresní prostor. Sběrnice je dělena až do 15 *oblastí*. Každá *oblast* se skládá z *hlavní linky* a může obsahovat až 15 dalších *linek*. Oblasti jsou spojeny *páteřní linkou*. Na každé *lince* může být až 255 zařízení a 1 směrovač. Celkem tedy umožňuje KNX zapojit až 61 455 zařízení. Jednotlivá zařízení mohou

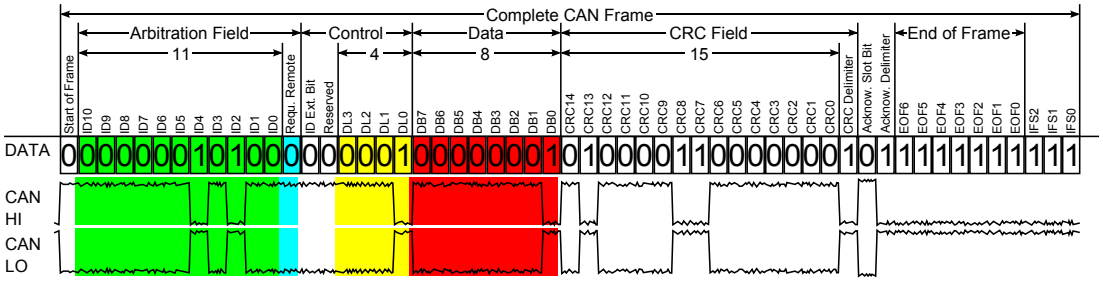
komunikovat přímo mezi sebou na základě adresy zařízení nebo adresy skupin zařízení. Síť je zapojena do stromové topologie, smyčky jsou explicitně zakázány. Umožňuje používat opakovače, díky nimž je možno komunikovat až na vzdálenost 4 km. Při odesílání zpráv se používá metoda bitové arbitrace. Řeší se tím také priorita zpráv.

1.4.10 CAN

Sběrnice CAN (controller area network) je sběrnice určená pro komunikaci mikroprocesorů a jiných zařízení v rámci automobilu bez použití centrální jednotky-/počítače. Je zaměřená hlavně na přenos krátkých zpráv po krátkých vzdálenostech.

Umožňuje dosahovat přenosových rychlostí až 1 Mb/s. O vývoj se stará společnost Bosch [9]. Při odesílání zpráv se používá metoda bitové arbitrace. Protokol sběrnice CAN nepředepisuje konkrétní typ fyzického spojení. Lze ho provozovat na libovolné sběrnici, která má alespoň 2 stavy, dominantní a recesivní. Toto je požadavkem pro navození komunikace pomocí bitové arbitrace.

Protokol CAN specifikuje základní (11 bitů) a rozšířenou (29 bitů) adresaci zpráv. Zařízení nemá pevně danou vlastní adresu. Protokol CAN místo toho navrhuje systém masek a filtrů, díky kterému si může zařízení nastavit, jaká podmnožina adres zpráv ho zajímá. Maska určuje, které bity jsou důležité a filtr říká, jakou mají mít hodnotu. Pokud adresa zprávy po aplikování masky odpovídá filtru, je dále zpracovávána.



Obrázek 1.8: CAN struktura požadavku (Převzato z Wikipedie [13])

Zpráva (viz obrázek 1.8) se skládá z následujících části:

Start of Frame – úvodní dominantní bit značící začátek zprávy. Slouží k synchronizaci zařízení po klidu na sběrnici.

Identifikátor zprávy – díky vlastnostem bitové arbitrace lze jednoduše upřednostňovat důležitější zprávy tím, že jim dáme nižší hodnotu identifikátoru. V obrázku je značen zeleně.

RTR bit – určuje jestli se jedná o vzdálenou nebo datovou zprávu. Datová obsahuje data podle adresy. Vzdálená značí požadavek, aby příslušné zařízení odeslalo datovou zprávu stejné adresy. V obrázku je značen modře.

Řídící bity – informace o typu identifikátoru a délce přenášených dat.

Data – samotná přenášená data, v délce 0–8 bajtů.

CRC – kontrolní součet předchozích bitů.

ACK bit – při odesílání zprávy je nastaven ACK bit na recesivní hodnotu. Poté co jiné (libovolné, všechna) zařízení tuto zprávu přečte, odešle potvrzovací zprávu s nastaveným ACK bitem na dominantní hodnotu. Díky tomu pozná odesílatel zprávy, že došlo k jejímu úspěšnému přenesení, a nemusí ji odesílat znovu. Pokud ale některé zařízení narazilo na chybu při čtení, odešle sérii dominantních bitů a odesílatel musí odeslání opakovat.

EOF – na závěr se se nachází 7 bitů značících konec zprávy.

IFS – interframe space, 7 bitů, které dávají jednotlivým uzlům čas pro předání aktuálně zpracovávané zprávy k dalšímu zpracování. V obrázku jsou zobrazeny pouze první 3.

Existují speciální čipy (např. MCP2551, MCP2561) umožňující odesílat a přijímat CAN zprávy pro další zpracování. Filtrování (je-li zapnuto) je poté provedeno již přímo na úrovni těchto čipů a zařízení se dozví pouze o zprávách, o něž má zájem. Některé čipy poskytují více masek a filtrů pro jemnější filtrování.

1.4.11 X10

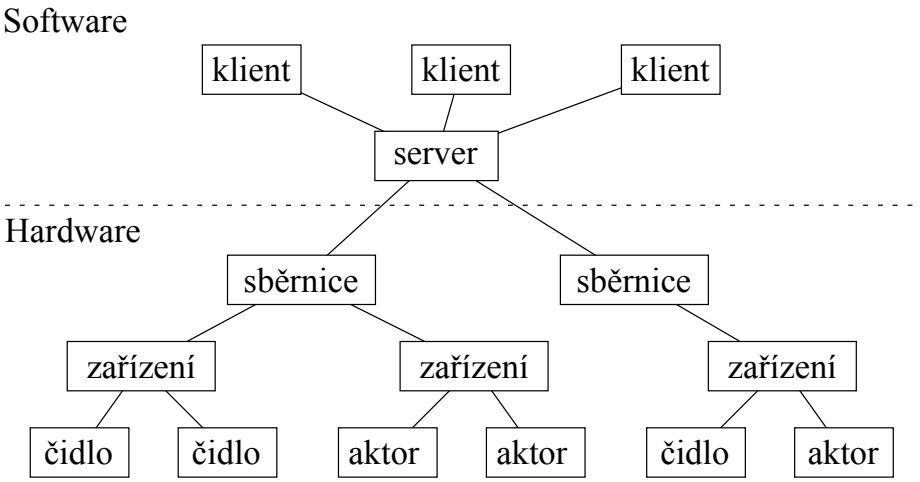
Komunikační protokol X10 je synchronní sériový protokol, který byl navržen firmou Pico Electronics přímo pro domácí automatizaci. K přenosu digitálních informací využívá hlavně elektrické vedení, ale existuje také varianta pro bezdrátovou komunikaci. Funguje pomocí zakódování 1 bitu informace při každé změně znaménka napětí střídavého proudu v elektrickém vedení. Dosahuje tedy rychlosti 100 b/s.

Existuje jak varianta, kdy zařízení pouze poslouchají příkazy, tak varianta, kdy mohou zařízení odpovídat. Primárně je určený k ovládání světel. Umožňuje adresovat až 256 zařízení. Logická 1 je reprezentovaná jako 120 kHz pulz po dobu 1 ms od nulového stavu ve vedení, následovaný 1 ms klidu. Logická 0 je reprezentovaná nejprve 1 ms klidu a 1 ms 120 kHz pulzu. Pro redundanci se každý bit vysílá dvakrát. Délka přenosu je závislá na konkrétní elektroinstalaci [11].

2. Návrh systému

Předchozí kapitola ukázala, že svět domácí automatizace je heterogenní. V této kapitole máme za cíl vytvořit abstraktní model, který nám usnadní další práci. Chceme vyrobit program, nadále označovaný jako „server“, který bude umět na jedné straně komunikovat s různými druhy hardware a na straně druhé komunikovat s různými programy.

Podívejme se nyní na schématické zobrazení situace zobrazené na obrázku 2.1.



Obrázek 2.1: Schéma situace

Na straně hardware se nejblíže serveru nachází *sběrnice*, se kterými by měl server umět komunikovat pomocí jejich protokolů. Na sběrnicích jsou připojena *zařízení*, která umí komunikovat protokolem příslušné sběrnice. Na jednotlivých zařízeních jsou umístěny *prvky*, ať již čidla nebo aktory. Zařízení má na starost umožnit pomocí protokolu sběrnice ovládat a zjišťovat stav svých prvků.

Na straně software se nachází jednotliví *klienti*. Jedná se o externí programy starající se například o automatizaci či uživatelské prostředí. S hardware komunikují vždy prostřednictvím serveru.

2.1 Návrh serveru

Hlavním cílem této práce je vhodně navrhnout server a implementovat jej. Dalším cílem je navrhnout grafické uživatelské rozhraní, které bude umožňovat ovládání jednotlivých součástí systému.

Pro server vytyčíme následující vlastnosti:

- Server spravuje komunikaci se zařízeními na různých sběrnicích za použití jednotného protokolu.
- Server je modulární, například co se týče přidávaní nových zařízení nebo sběrnic do systému.
- Dovoluje klientům zjišťovat a ovlivňovat stav prvků spravovaných serverem a objednat si upozornění při změně stavu prvků. Klienti také mají možnost komunikovat mezi sebou skrze server.

- Umožňuje koexistenci manuálního a automatického řízení. Např. (dočasně) převzít manuálně kontrolu nad automatickými akcemi.

Aby byl server modulární a dalo se o něm dobře přemýšlet, je rozdělen do hierarchických vrstev, podobně jako např. rodina protokolů TCP/IP. Každá vrstva má specifickou funkci, ukrývá před vyššími vrstvami informace nedůležité pro další zpracování a využívá služeb nižších vrstev. Každá vrstva má definované rozhraní pro komunikaci se sousedními vrstvami.

Vrstev zvolíme 5 a to následující:

Fyzická vrstva obsahuje veškerý hardware a software nacházející se mimo server směrem k fyzickým zařízením.

Vrstva sběrnic obsahuje ovladače pro různé typy fyzických sběrnic. Tyto ovladače následně vytváří pro konkrétní fyzické sběrnice jejich logické reprezentace, dále značené jako logické sběrnice. Ty poté zprostředkovávají komunikaci po fyzických sběrnicích.

Vrstva zařízení obsahuje ovladače pro různé typy fyzických zařízení [obdobně jako u sběrnic zavedeme i logická zařízení]. Tyto ovladače následně vytváří pro konkrétní fyzická zařízení jejich logické reprezentace, dále značené jako logická zařízení. Logická zařízení se starají se o práci s daným zařízením a jeho prvky.

Vrstva zpráv zajišťuje komunikaci mezi serverem a klienty pomocí zpráv.

Aplikační vrstva obsahuje klienty využívající služeb serveru.

2.1.1 Fyzická vrstva

První vrstvou je fyzická vrstva. Obsahuje veškerý hardware a software nacházející se mimo server směrem k fyzickým zařízením. Tato vrstva je nejméně specifická, jelikož má umožnit spravovat prakticky libovolnou periferii. Lze ji rozdělit do tří částí: hardware (jednotlivé sběrnice, zařízení a prvky), rozhraní mezi sběrnicí a počítačem, na němž bude spuštěn server, a ovladače operačního systému pro jednotlivá rozhraní.

Popis hardwaru sběrnic byl již naznačen v předchozí kapitole a nepotřebujeme se jím více zabývat .

Rozhraní mezi počítačem a sběrnicí je již o něco konkrétnější. Většina dnešních počítačů obsahuje USB porty, přes které je možno připojit redukce na další typy sběrnic (např. RS 485). Dále se stále vyskytuje sériový port (RS 232). Také se rozšiřují jednodeskové počítače umožňující komunikaci přes různé sběrnice přímo (např. Banana Pi obsahuje přímo možnost zapojení sběrnice SPI, CAN či I^2C [12]).

Sběrnice samotná ale nemusí být připojena přímo fyzicky k počítači, na němž je spuštěn server. Například může být připojena na jiném počítači, který je spojen s serverovým počítačem pomocí síťového spojení. Může také být umístěna na zařízení připojeném k serveru pomocí jiné sběrnice (např. server – sběrnice CAN – zařízení – sběrnice OneWire).

Operační systémy poskytují různá rozhraní pro přístup ke sběrnicím, např. unixové OS poskytují přístup k zařízením jako k souborům ve složce `/dev`. Zpravidla systémová rozhraní umožňují čtení a zápis dat a detekci připojení sběrnice.

Konkrétní použité příklady jsou uvedeny v kapitole 5 o testovacím prostředí.

2.1.2 Vrstva sběrnic

Druhou vrstvou je vrstva sběrnic. Jedná se o první abstraktní vrstvu serveru. Stará se o komunikaci na sběrnici (s ostatními zařízeními) pomocí rozhraní fyzické vrstvy. Přebírá požadavky k odeslání od vrstvy zařízení a za použití protokolu dané sběrnice je odesílá. Hlásí také vyšší vrstvě příchozí pakety a změny stavu sběrnice (rozbití, spravení).

Požadavek pro sběrnici označuje souhrnně paket pro sběrnici (sestavený dle jejího protokolu) plus pomocná data pro vykonávání aktivit ve vrstvě sběrnic. Např. které logické zařízení požadavek iniciovalo, proběhla-li komunikace bez problémů nebo jestli má být požadavek periodicky opakován.

Životní cyklus požadavku může probíhat několika způsoby. Vždy se v něm ale vyskytují tito aktéři:

Logické zařízení – součást vyšší vrstvy, může iniciovat požadavky a přijímat odpovědi na ně. Dále označováno jen jako zařízení.

Logická sběrnice – součást této vrstvy. Dále označováno jen jako sběrnice.

Paket – konkrétní data požadavku pro odeslání na sběrnici.

Fyzická sběrnice – fyzická sběrnice připojená k serveru pomocí rozhraní fyzické vrstvy, umožňuje předávat pakety pro daný typ sběrnice.

Fyzické zařízení – fyzické zařízení, s nímž je vedena komunikace týkající se požadavku.

Nejjednodušší případ je komunikace master-slave, kdy server v úloze mastera iniciuje komunikaci dotazem a čeká na odpověď:

Zde nejprve sběrnice obdrží od vyšší vrstvy požadavek k odeslání a zařadí ho do svojí fronty požadavků. Ve chvíli, kdy na požadavek dojde řada, odešle sběrnice paket pomocí rozhraní fyzické vrstvy. Mimo server nyní fyzické zařízení zpracuje přijatá data od fyzické sběrnice a vyšle na ni paket reprezentující odpověď. Tento paket dorazí přes fyzické rozhraní ke sběrnici a ta následně předá zařízení požadavek doplněný o data z odpovědi. Má-li být požadavek opakován, je znovu přidán sběrnicí do její fronty požadavků na příslušné místo. Logická sběrnice pokračuje ve vykonávání požadavků z fronty.

Složitější variantou je multimaster komunikace, kdy mezi konkrétním dotazem a odpovědí na něj může proběhnout další komunikace. Také mohou přicházet pakety bez předchozího dotazu. Zde je potřeba rozdělit předchozí proces na dvě části: přijímání paketů a odesílání paketů serverem.

Pro odeslání většinou multimaster fyzické rozhraní umožňuje uložit několik paketů k odeslání, které budou odeslány, jakmile to bude možné. Naopak při přijímání mohou odpovědi přicházet v libovolném pořadí, je proto potřeba využít identifikátory konkrétního typu sběrnice pro určení toho, kterým zařízením předat požadavek předat.

Nyní víme, jak vypadá životní cyklus požadavku. U většiny sběrnicových protokolů je možné vyřizovat nejvýše jeden požadavek zároveň. Požadavky od zařízení jsou tedy vyřizovány asynchronně a je potřeba je uskladňovat, než přijde řada na jejich odeslání. Existují různé možnosti v jakém pořadí zpracovávat jednotlivé požadavky, například obyčejná nebo prioritní fronta (halda). Požadavky mohou být periodické nebo jednorázové. Sběrnice ale mají omezenou přenosovou kapacitu a je nutné myslet na to, že může dojít k zahlcení přenosového pásma.

Jedním přístupem, jak tento problém řešit, je plánovat si, kdy nejdříve má být požadavek vykonán. Poté co je možné na sběrnici odeslat požadavek, zjistí ovladač sběrnice, existuje-li požadavek, který již může být odeslán. Z takovýchto požadavků vybere ten, který měl být odeslán nejdříve. Pokud není požadavek pro odeslání, vyčká po dobu do nejbližšího požadavku. Výhodou je jednoduchost a rovnoměrné zpomalení při zahlcení (s možností upřednostňování kritických požadavků).

Další možný přístup je postup použitý v bitové arbitraci. Všechny požadavky mají danou prioritu a jsou vyřizovány v pořadí podle ní. Zde je výhodou, že je zjevné, které požadavky budou upřednostněny, ale je potřeba být velmi pečlivý při návrhu. Hrozí například, že by pak některé požadavky nebyly vyřízeny nikdy.

Podobně jako je k fyzické sběrnici většinou připojeno více zařízení, musí reprezentace sběrnice umožnit připojení více logických zařízení podle vlastností příslušné sběrnice.

Rozhraní vrstvy sběrnic se skládá ze tří složek. Sběrnice umožňuje přijmout požadavek k odeslání od vyšší vrstvy. Při přijetí dat ze sběrnice předává požadavek příslušným zařízením nacházejícím se vyšší vrstvě. Pokud je tento požadavek odpověďí na předchozí požadavek od zařízení, je odpověď předána tomuto zařízení. Jinak je zařízením je požadavek předán všem zařízením, která si zaregistrovala tento typ požadavků (CAN), či všem, co jsou připojena ke sběrnici. Sběrnice také vysílá zprávu v případě problémů a zotavení z nich, viz dále.

Příkladem může být ovladač sběrnice Modbus, který komunikuje s fyzickou sběrnicí RS485 pomocí sériového portu `/dev/USBtty0`. Ovladač přijímá požadavky a jeden po druhém je vyřizuje. Vyřízení požadavku probíhá odesláním Modbusových paketu a vyčkáním na příslušnou odpověď. Pokud nedorazí odpověď včas nebo dojde k jiné chybě, je tato chyba propagována výše. Zde je nahlášeno zařízení, které vyslalo původní požadavek. Jinak je zařízení předána přijatá odpověď. Dále pokud dojde k chybě s rozhraním samotným, hlásí sběrnice všem svým zařízením, že je nefunknční. Podobně, když dojde k nápravě.

2.1.3 Vrstva zařízení

Na vrstvě zařízení sídlí logická reprezentace fyzických zařízení. Každé zařízení je připojeno k jedné sběrnici. Zařízení se stará o to, aby mělo aktuální reprezentaci stavu příslušného fyzického zařízení a jeho prvků. Jsou zde konstruovány požadavky pro konkrétní typ sběrnice, na které je zařízení připojeno. Také zde dochází ke zpracování odpovědí či příchozích zpráv od sběrnice. Jedním ze smyslů této vrstvy je ušetřit komunikaci na sběrnici. Například pokud je na zařízení více prvků a jde o master/slave správu sběrnice. Obecně je u většiny protokolů možnost zjišťovat stav více čidel na jednom zařízení najednou. Dále je zde možné odhalit nekomunikující zařízení pro všechny jeho prvky.

Dalším problémem je přístup k jednotlivým prvkům zařízení, ať již čtení hod-

noty čidla, nastavení aktoru nebo vykonání některé další činnosti zařízením či jeho prvkem. Vzhledem k různorodé charakteristice jednotlivých prvků by bylo nevhodné použít striktní systém pro jejich ovládání.

Uvažme například následující prvky: osvětlení, spínač, světelnou závoru a elektromagnet na otevření dveří.

- U osvětlení nás jednak zajímá, jestli světlo svítí, popřípadě jakou intenzitou či barvou. Dále je vhodné mít možnost nastavit jeho intenzitu či barvu. Také by mohlo být šikovné mít možnost pouze dát příkaz „změň stav“, který zhasne či rozsvítí osvětlení, podle toho, v jakém stavu bylo. Dalším příkladem je ovládání kotle či radiátorů.

- Spínač je jednodušší, jelikož zde není co nastavovat. Pouze je možné zjistit, v jakém stavu se spínač nachází, zdali je sepnutý či rozepnutý. Podobným příkladem je například teplotní čidlo.

- Světelná závora generuje událost, např. když člověk projde turnikety metra. Podstatným zde není stav závory, ale jeho změna.

- U elektromagnetu také neexistuje „stav“, v němž by se nacházel, ale chceme mít možnost mu vyslat příkaz k otevření dveří.

Zde jsme došli k systému podobnému objektově orientovanému programování. Každé zařízení poskytuje několik rozhraní, dále označovaných jako *interface*. Interface může příslušet jak jednomu prvku (např spínač), tak více prvkům spolu souvisejícím (jednotlivé RGB kanály u barevné LED). Tento interface se skládá ze sady atributů (proměnných) a funkcí (metod).

Atributy zde obsahují hodnotu či stav v němž se může prvek zařízení nacházet. Výše jsme viděli, že prvky, které mají stav, je možné je zařadit do dvou typů:

Read-write prvek má stav a zařízení ho přímo ovlivňuje – například osvětlení.

Read-only prvek má stav ale zařízení ho nemůže přímo ovlivnit - například spínač.

Zařízení si pamatuje stav svých prvků a jejich atributů. Když dojde ke změně hodnoty atributu, ovlivní pomocí požadavku na sběrnici skutečný stav prvku. Dále si zařízení aktualizuje informaci o skutečném stavu svých prvků na základě příchozích požadavků od sběrnice a hlásí, když dojde ke změnám.

Pokud prvek nemá stav, neodpovídá mu přímo žádný atribut. Čidla rovnou generují zprávy a aktory je možné pouze ovládat funkcemi. Může ale například existovat atribut udávající počet aktivací čidla nebo aktoru.

Pro ovládání jednotlivých fyzických zařízení a prvků je základem to, aby plnily funkci, kterou od nich uživatel očekává, tedy aby spínače světel ovládaly příslušné světelné okruhy, termostat ovládal ohřev vody a podobně. Jsou ale situace, kdy by bylo vhodnější převzít řízení nad těmito „přirozenými“ akcemi. Například pokud chceme vypnout světlo pro bezpečnější výměnu žárovky. Proto zavedeme možnost mít pro každý atribut několik hodnot s různou *prioritou*.

Základní dvě úrovně priorit jsou nastíněné výše, ale budeme uvažovat o obecném počtu priorit P. Jako priority budeme uvažovat celá čísla z rozsahu 0 až $P - 1$. Atribut tedy uchovává až P různých hodnot s různými prioritami. Při

nastavování hodnoty atributu je potřeba uvést i její prioritu. Skutečný stav prvku poté ovlivňuje hodnota s nejvyšší prioritou. Je také zapotřebí mít možnost vydat příkaz, že hodnota atributu s danou prioritou má být zrušena. Hodnota s prioritou 0 nemůže být zrušena.

Funkce operují nad atributy a vstupem od klientů. Mohou měnit a ovlivňovat atributy a tím i stav fyzického zařízení. Základní funkce jsou *set* (nastavení hodnoty atributu s prioritou), *get* (zjištění hodnoty atributu) a *unset-priority* (zrušení hodnoty s danou prioritou). Dále je možné pro jednotlivé interface tvořit specializované funkce, např. změna stavu světla.

Nyní víme, jak funguje komunikace od interface až po prvky. Od vyšší vrstvy dostávají jednotlivé interface zprávy. Zpráva se v tomto případě skládá ze jména funkce k zavolání a parametrů pro danou funkci. Příslušná funkce je následně zavolána s danými parametry a může například odeslat skrze zařízení požadavky pro vrstvu sběrnic. Dále zařízení vysílá zprávu v případě problémů, ať již sběrnice či zařízení samotného, a zotavení se z nich, o čemž bude pojednáno dále.

Jako příklad můžeme vzít zařízení fungující jako termostat připojené na sběrnici CAN obsahující připojené teplotní čidlo a relé. Poskytuje dva interface, teplotní čidlo a relé. Teplotní čidlo má atribut současnou teplotu a funkci na zjištění teploty. Relé má atribut na stav sepnutí a funkce na zjištění, nastavení a změnu stavu.

2.1.4 Vrstva zpráv

Pro zprostředkování komunikace mezi jednotlivými složkami serveru a klienty, případně klienty samotnými, zavedeme vrstvu zpráv.

Zprávou budeme označovat strukturovanou informaci předávanou buď od serveru klientům, či od klientů serveru. Zprávou je také informace předávaná mezi klienty prostředníctvím serveru, jelikož prochází serverem a je v každý okamžik buď jednoho nebo druhého druhu. Zprávy od klientů serveru budeme označovat jako *příkazy*.

Zprávy od serveru se rozlišují do dvou druhů: všeobecné informace o stavu serveru a jeho částí a zprávy pro konkrétního klienta.

Všeobecné zprávy od serveru se řadí do následujících typů: informace o událostech (např. na prvcích), chybové zprávy, změny stavu částí serveru, informace o nových klientech a debugovací informace. Každá zpráva obsahuje identifikátor části serveru, která ji vyslala.

Některé klienty zajímají pouze některá konkrétní rozhraní, například termostat nemusí znát stav světel. Vrstva zpráv proto umožňuje klientům nastavit si *preference*, jaké typy zpráv chtějí od kterých částí serveru dostávat. Výjimkou je informace o nových klientech, která je zasílána všem klientům.

Příkazy se rozlišují do dvou druhů: příkazy pro nastavení serveru (jméno klienta, jeho preference o tom, jaké informační zprávy chce dostávat) a příkazy pro části serveru (hlavně interface, ale i pro zařízení, sběrnice a jiné klienty). Příkazy pro ostatní klienty jsou také filtrovány preferencemi.

Pro spravování jednotlivých zařízení a sběrnic je potřeba jednotný systém označování jednotlivých prvků na zařízeních. Většina protokolů sběrnic používá číselná označení, ale vzájemně jsou nekompatibilní. Pro přehlednost použijeme textová *jména* jako identifikátory jednotlivých složek systému. Každá část serveru

(sběrnice, zařízení či klient) má své jméno, které je unikátní v rámci serveru. Není kladeno žádné explicitní omezení na jeho formát.

Zpráva se skládá z posloupnosti tokenů, formát je inspirovaný shellovými příkazy. Formát bude rozebrán podrobněji v následující kapitole.

Jednotlivé zprávy jsou vyřizovány asynchronně. Klienti si při zprávě pro server volí vlastní identifikátor zprávy, podle kterého mohou spojit odpověď od serveru s jejich původním dotazem.

Aby mohli klienti vzájemně komunikovat, dáme jim podobné prostředky jako prvkům. Tedy klienti mohou vytvářet atributy na straně serveru, propagovat zprávy ostatním klientům, kteří o ně mají zájem, a je na nich možné volat funkce. Díky tomu může například klient uživatelského rozhraní zobrazit informace i o jiných připojených klientech. Jeden klient může mít také na starost pouze monitorování ostatních klientů.

Rozhraní poskytované nižším vrstvám serveru se skládá z možností odeslat zprávu všem klientům či konkrétnímu klientovi. Dále vrstva zpráv na základě příkazů klienta volá funkce serveru nebo jeho částí (interface, zařízení či sběrnice).

Klienti se mohou připojit k serveru pomocí lokáních soketů a odesílat zprávy pro server či mezi sebou.

2.1.5 Aplikační vrstva

Do aplikační vrstvy spadají všichni *klienti*, které se připojují k serveru a komunikují s ním za pomoci zpráv. Každý klient se identifikuje jménem a klienti mohou mezi sebou komunikovat prostřednictvím vrstvy zpráv. Patří sem dva základní druhy programů – automatické a uživatelem ovládané.

Do kategorie automatických klientů spadají všichni klienti zajišťující automatiku některých prvků na základě jiných prvků. Může se jednat o správu světel na základě spínačů v místnosti, schodišťový automat, termoregulaci, zaznamenávání počasí a mnohé další.

Mezi uživatelem ovládané klienty spadá například uživatelské rozhraní, které může vizualizovat aktuální stav všech zařízení v systému. Také je možné například mít klienta umožňujícího zasílat SMS zprávy a informovat tak majitele o otevřeném okně po odchodu z domu.

Konkrétní použité příklady jsou uvedeny v kapitole 4 o ukázkových klientech.

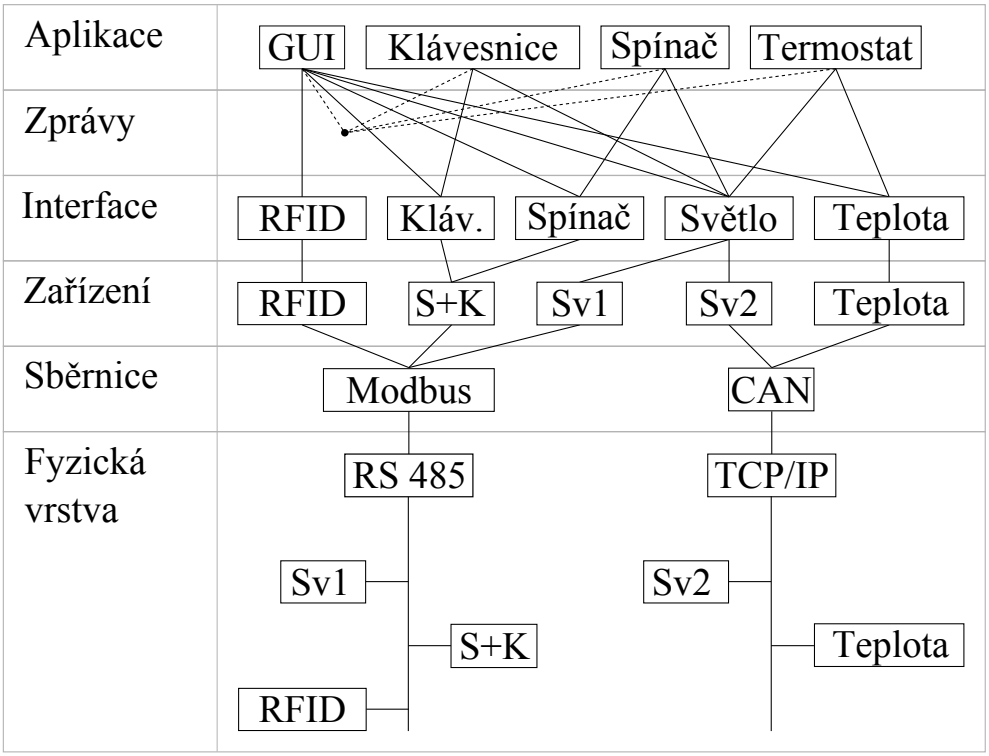
2.1.6 Ukázka systému

Na obrázku 2.2 je vidět graficky znázorněný konkrétní příklad zapojení celého systému.

Jednotliví klienti budou popsáni podrobněji v kapitole 5 o ukázkových klientech.

Vrstva zpráv je zde znázorněna bez konkrétních složek, jelikož se stará pouze o předávání zpráv. Čárkovaně je vyznačena komunikace mezi klienty (zde mezi GUI a ostatními klienty), plné čáry reprezentují komunikaci klient – server.

Vrstva zařízení je zde reprezentovaná dvěma částmi, zařízeními a interface. Následují tabulky popisující jednotlivá zařízení a interface.



Obrázek 2.2: Ukázka zapojení daných příkladů

interface	*skutečný prvek*	*co dělá*
RFID	RFID čtečka	hlásí detekované RFID tagy
Klávesnice	16-tlačítková klávesnice	hlásí zmáčknuté klávesy
Spínač	dvoustavový spínač	hlásí změnu stavu
Světlo	žárovka či LED	umožňuje nastavit intenzitu osvětlení
Teplota	teplotní čidlo	hlásí teplotu

zařízení	*z čeho se skládá*
RFID	Arduino Nano s RFID čtečkou s RS485 modulem
S+K	Arduino Nano se spínačem a klávesnicí s RS485 modulem
Sv1	Arduino Nano se sadou 8 LED s RS485 modulem
Sv2	Arduino Mini se sadou 6 LED s CAN modulem MCP2515
Teplota	Arduino Mini se teplotním čidlem s CAN modulem MCP2515

Vrstva sběrnic obsahuje ovladače na sběrnice Modbus a CAN.

Fyzická vrstva obsahuje konkrétní zařízení, viz kapitola o testovacím prostředí.

2.2 Zotavování se z chyb

Běžně dochází k nehodám a ani naše situace není výjimkou. Jako u všech technických zařízení i zde se může stát, že libovolná fyzická část systému přestane reagovat či bude dávat chybná data. Příčiny mohou být různé, ať již výpadek proudu, fyzické poškození hardware, softwarová chyba nebo třeba odpojení napájecích či datových vodičů.

Server by měl být schopen detekovat problémové situace, nahlásit je dál a také poznat, že došlo k jejich napravení. Jednotlivé vrstvy se starají o sebe a o to, co je pod nimi, a hlásí problémy výš.

Konkrétně mohou nastat tyto problémy:

Fyzické zařízení neodpovídá – jedno konkrétní fyzické zařízení přestalo odpovídat či odpovídá špatně, ale sběrnice (a další zařízení na ní) stále fungují. Mohlo dojít k poškození zařízení, přerušení spojení či jiné nepředvídané události. Je potřeba prověřit samotné zařízení.

Problémy sběrnice – celá jedna sběrnice vykazuje problém, například odpojení rozhraní fyzické vrstvy či je na sběrnici neplatný stav. Je potřeba prověřit fyzickou sběrnici.

Klient nečte co mu server posílá – klientovi jsou zasílány zprávy, ale klient je neodebírá, tedy jsou hromaděny v bufferu. Server v tuto chvíli zruší preference daného klienta a dá mu o tom zprávu.

Klient nereaguje – klient v poslední minutě nezaslal žádnou zprávu, nejspíše došlo k chybě v programu. Zde dochází k odpojení klienta.

Chyby sběrnic a zařízení se server snaží opravit pokusem o opětovné připojení jednou za daný čas. V případě úspěchu je informace o opětovném připojení šířena pomocí zpráv.

Problémy mohou nastat ale i v komunikaci mezi klienty. Uvažme například situaci, kdy první klient počítá předpověď počasí a druhý klient ovládá tepelné čerpadlo. Druhý klient se zeptá prvního na předpověď. První klient počítá, ale zhavaruje. Druhý klient zde dostane pouze informaci, že se první klient odpojil, a na základě této informace pozná, že již předpověď, na kterou čeká, nedostane.

3. Implementace serveru

V předchozí kapitoly jsme položili, jak by měl server fungovat. V této kapitole se nachází popis konkrétní implementace, jejího vnitřního fungovaní a komunikačního protokolu mezi serverem a klienty.

3.1 Obecné informace o serveru

Server funguje pod systémem UNIX. Je implementován v programovacím jazyce C s použitím knihovny LibUCW.

Celý server je řízený událostmi, jako hlavní událostní smyčka se používá modul Mainloop z LibUCW [17]. Mainloop umožňuje jednoduše spravovat více aktivit, které se mají odehrát v reakci na události. Server využívá dvou služeb Mainloopu: timer a Record I/O.

Timer umožňuje vyvolat událost v daném čase. Čas události může být absolutní či relativní a může být měněn i po nastavení timeru. Timer obsahuje uživatelem nastavený ukazatel na funkci (callback), která se má zavolat na ošetření události. Používá se například pro detekci neaktivního spojení či vynucení odpovědi v daném čase.

Record I/O hlídá aktivitu na souborových deskriptorech. Umožňuje nastavit funkce k zavolání, když je deskriptor připraven ke čtení či k zápisu, a když dojde k problému s deskriptorem (např. přerušení spojení či chyba při čtení nebo zápisu). Funkce zavolaná při datech ke čtení dostane informaci o tom, kolik a jaká data má aktuálně k dispozici v bufferu, a vrací, kolik dat ze začátku bufferu zpracovala. Může také vrátit nulu, pokud chce doplnit více dat.

Record I/O také umožňuje kontrolovat délku bufferu a ohlásit událost, pokud délka nezpracovaných dat převýšila uživatelem definovanou mez. Record I/O dále obsahuje timer, například pro detekci toho, že se dlouho nic na spojení nestalo. Informace o vypršení timeru je interpretována jako druh chyby a předána obslužné funkci.

Přiklad užití Record I/O je již zmíněná komunikace přes TCP/IP, ale je možné použít souborové deskriptory i při komunikaci mezi procesy, ať již přímo, či přes lokální sokety (unix domain socket).

V kódu programu jsou proměnné a jednotlivá rozhraní pojmenovány anglicky. Zde budou označovány česky, vždy s uvedením jejich anglické verze.

V předchozí kapitole jsme nastínili, jaké funkce mají plnit jednotlivé vrstvy serveru. Zde si je rozebereme po jednotlivých vrstvách, jak se mají konkrétně chovat.

3.2 Společná data

Program je psaný v jazyce C, který ale nemá objekty. Je možné je ale napodobit pomocí vnořených struktur a callbacků. Jako společného předka zvolíme strukturu „entita“, která bude obsahovat informace společné pro všechny části serveru.

Server si uchovává seznam všech entit a zajišťuje unikátnost jejich jmen. Poskytuje rozhraní pro nalezení entit podle jména či podle typu. Také poskytuje rozhraní pro zjištění stavu entity. Dále má server strukturu pro uchování konfigurace pro celý server. Skládá se z těchto částí:

- jméno logovacího souboru,
- informace co vše se má logovat,
- jméno konfiguračního souboru – dodáno jako parametr při spuštění,
- interval pro obnovu spojení se zařízeními a sběrnicemi,
- nastavení lokálního soketu pro připojení klientů.

Entita (`entity`) obsahuje následující:

- struktury pro načtení konfigurace pro zařízení a sběrnice,
- informace o typu entity – může být sběrnice, zařízení, interface nebo klient,
- jméno,
- aktuální stav – záleží na typu entity, viz dále,
- seznam funkcí,
- seznam atributů,
- data na kontrolu závislostí, viz níže.

Entity reprezentující sběrnice a zařízení včetně interface jsou popsané v konfiguračním souboru serveru načítaném při jeho startu. Obecný formát konfigurace je popsán v dokumentaci LibUCW [15]. Na parsovaní konfigurace se používá parser z knihovny LibUCW [16] a každý ovladač poskytuje deklaraci pro tento parser o tom, jak vypadá konfigurace pro tento ovladač.

Jelikož konfigurace není stromová, ale plochá, je nutné vytvářet logické spojení mezi zařízeními a sběrnicemi až po načtení konfigurace. Jelikož ale může být sběrnice připojena prostřednictvím zařízení na jiné sběrnici, mohou být vytvořeny konfigurace, kde by došlo k cyklické závislosti. Proto po úvodním načtení konfigurace dochází ke kontrole splnění závislostí jednotlivých částí serveru a jejich necykličnosti pomocí prohledávání. Každá konfigurovaná entita musí stanovit, na kterých entitách je závislá (orientovaný graf).

3.3 Vrstva sběrnic

Nejprve se zaměříme na vrstvu sběrnic, obsahující ovladače pro jednotlivé sběrnice. Sběrnice je rozšířením entity, tedy umí vše, co ona.

Každá sběrnice (`bus`) obsahuje:

- vše, co entita,
- informaci o typu sběrnice – v tuto chvíli jsou implementovány ovladače na fyzické sběrnice Modbus a CAN přes TCP/IP,
- seznam zařízení k ní připojených,
- informace o požadavcích (fronta, timer, počet),

- callback *fun_connect* – je zavolán, když server zkouší obnovit spojení obnovit spojení s fyzickou sběrnicí.

Dále si sběrnice pamatuje data potřebná pro připojení a chod konkrétního typu fyzické sběrnice. Může se jednat například o cestu k fyzickému rozhraní, rychlost komunikace, prostor pro ukládání přijetých zpráv a jiné.

Sběrnice umožňuje přijímat a vyřizovat požadavky od zařízení. Každý požadavek si pamatuje:

- jestli a kdy má být opakován,
- jakému zařízení a sběrnici přísluší.

Dále ho mohou konkrétní sběrnice rozšiřovat o další data potřebná ke zpracování požadavku. Server také poskytuje možnost využít sadu funkcí na vyřizování požadavků pomocí prioritní fronty. Sběrnici poté stačí pouze říci, kdy nejdříve má být požadavek vykonán, a o ostatní se již postará server. Sběrnice mohou také požadavky odebírat z fronty. Ovladač sběrnice si může implementovat i vlastní způsob pro určení pořadí vyřizování požadavků. Rozhraní vrstvy sběrnic se má podle předchozí kapitoly skládat ze tří částí.

Sběrnice musí poskytovat funkci pro přijímání požadavků od zařízení. Požadavky musí obsahovat strukturu požadavku a dále mohou mít data specifická pro sběrnici. Funkce samotná nemá přesně strukturované jméno a parametry, jelikož příslušná zařízení ví, k jakému typu sběrnice se připojují.

Každé zařízení implementuje funkci *fun_reply*, kterou sběrnice zavolá, když má požadavek od fyzického rozhraní k předání danému zařízení.

Sběrnice musí implementovat funkci *fun_connect*, pomocí které se ji server může pokusit uvést do funkčního stavu. Tato funkce je jednak volána při spuštění serveru a dále při pokusu o zotavení se po rozbití. Sběrnice se může po dobu běhu serveru nacházet v pěti různých stavech: config, connecting, ready, sending a broken. Přechody mezi stavy jsou znázorněny v obrázku 3.1.

Popis jednotlivých stavů:

Config – načtení konfigurace.

Connecting – logická sběrnice se snaží navázat spojení s fyzickou sběrnicí pomocí rozhraní fyzické vrstvy.

Ready – spojení bylo úspěšně navázáno, mohou být vykonávány požadavky od logických zařízení.

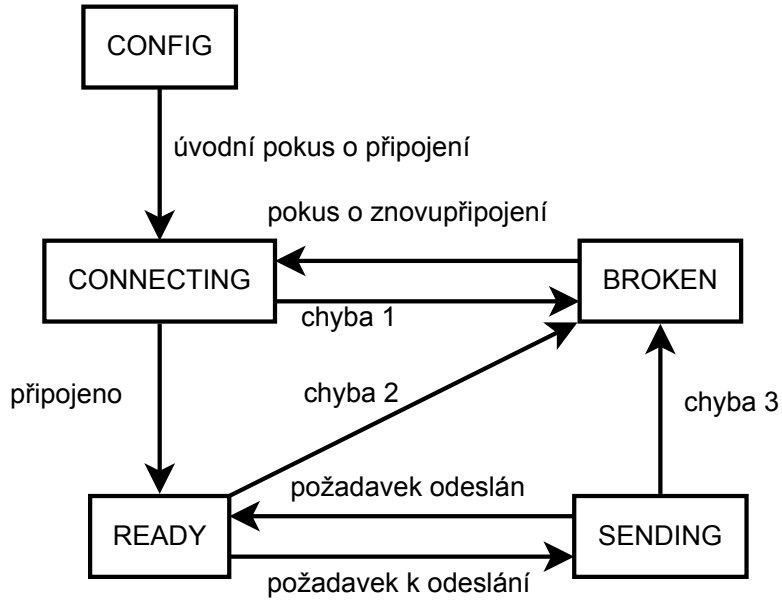
Sending – dochází k odesílání požadavku, případně aktivnímu čekání na odpověď.

Broken – nastal problém na úrovni fyzické vrstvy, sběrnici není možno využít k odeslání požadavků – všechna zařízení připojená k dané sběrnici jsou informována o změně jejího stavu a jsou také nastaveny jako broken.

Popis přechodů mezi stavy:

Úvodní pokus o připojení – po načtení konfigurace se volá *fun_connect* dané sběrnice.

Chyba 1 – chyba při připojování – možné příčiny jsou nefunkční/nepřipojené fyzické rozhraní, špatná konfigurace.



Obrázek 3.1: Stavy sběrnice a přechody mezi nimi

Chyba 2 – chyba při čekání – přerušení spojení pomocí fyzického rozhraní.

Chyba 3 – chyba při odesílání – zpracování konkrétního požadavku neproběhlo korektně, na vině může být vadné zařízení i sběrnice samotná.

3.4 Vrstva zařízení

Dále se budeme zabývat vrstvou zařízení obsahující ovladače pro jednotlivá zařízení. Zařízení je rozšířením entity, tedy umí vše, co ona.

Každé zařízení (`device`) obsahuje:

- vše, co entita,
- informaci o typu zařízení – v tuto chvíli je implementováno několik ovladačů pro zařízení připojená ke sběrnicím Modbus a CAN,
- data pro spojení se sběrnicí – její jméno, ukazatel na ní a uzel do jejího seznamu zařízení,
- callback *fun_reply* – je zavolán při příchozím požadavku pro toto zařízení od sběrnice,
- callback *fun_broken* – je zavolán, když se rozbije sběrnice či konkrétní zařízení,
- callback *fun_connect* – je zavolán, když server zkouší obnovit spojení s fyzickým zařízením prostřednictvím sběrnice.

Ovladač musí implementovat všechny tři zmíněné funkce. Funkce *fun_connect* se také stará o nastavení fyzického a logického zařízení do výchozího stavu (ať již zhasnutí světel či zjištění aktuální teploty).

Hlavní součástí zařízení jsou interface, které poskytuje. Jednotlivá zařízení mohou obsahovat více interface, ať již stejných či různých. Každý interface obsahuje:

- vše, co entita,
- typ interface,
- informace, aby zařízení poznalo, o který interface se jedná,
- callback *fun_update* – je zavolán, když dojde ke změně nastavení hodnot atributů, má za úkol zaktualizovat stav prvků.

Interface stejného typu musí poskytovat stejné atributy, funkce a akce, ale jejich implementace je již ponechána na konkrétním ovladači zařízení. Není tedy například obecný ovladač pro interface světlo, ale jednotlivé ovladače zařízení ho implementují samy.

V předchozí kapitole jsme nastínili, jak mají fungovat atributy pro P priorit. Atribut se skládá z dvou částí: obecných informací o atributu a konkrétních dat atributu podle jeho typu. Obecné informace jsou:

- jméno atributu,
- textový popis a informace o jednotce atributu,
- je-li atribut read-write či read-only,
- současná priorita,
- pole o délce P indikující, je-li daná priorita nastavena,
- je-li zrovna prováděna změna hodnoty a hodnota nastavovaná na zařízení,
- typ atributu určující, jakého datového typu je hodnota atributu a jak má server nakládat při práci s hodnotou,
- ukazatel na datovou složku atributu.

Datová složka atributu zpravidla obsahuje následující:

- pole na hodnoty o délce P,
- aktuální hodnota na zařízení,
- je-li zrovna prováděna změna hodnoty a hodnota nastavovaná na zařízení,
- validační data, například pro omezení rozsahu hodnot na úrovni atributu.

Server má implementovanou datovou složku typu int a její podtypy podle rozsahu (int, uint16 a byte). Do budoucna je možné naimplementovat mimo jíné datovou složku string, například pro displeje.

Atributy jsou zakládány po načtení konfigurace a jsou reinicializovány na začátku každého pokusu o připojení interface, tak aby byly v konsistentním stavu. Server funguje dominantně, tedy se snaží u všech atributů s možností zápisu nastavit hodnotu, která má aktuálně nejvyšší prioritu.

Funkce interface jsou reprezentovány tabulkou pro každou entitu obsahující trojice jméno funkce, popis funkce a ukazatel na funkci v C. Funkce interface jsou volány prostřednictvím mechanismu zpráv. Dostávají jako parametr entitu, na kterou má být funkce zavolána, a zbytek zprávy. Mohou ovlivňovat stav interface

a zařízení. K dispozici je několik základních funkcí interface pro ovlivňování a zjišťování hodnot atributů.

Funkce zpracovávající atributy mohou zpracovávat více atributů najednou:

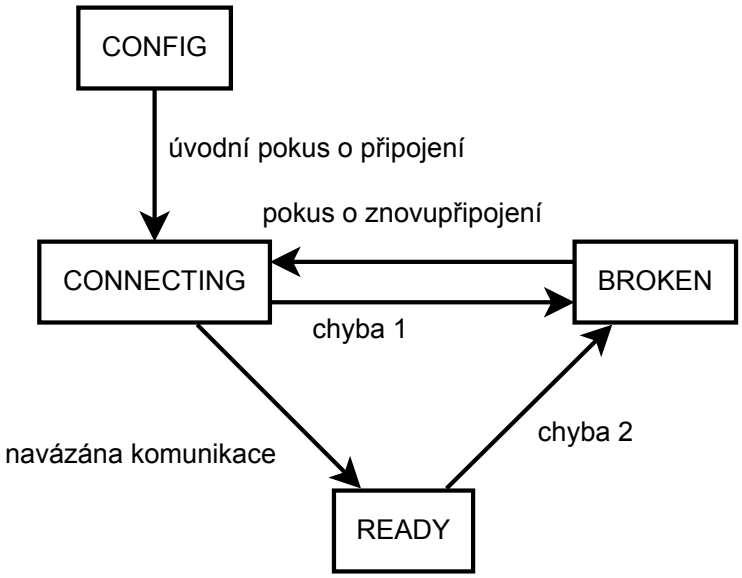
- `get` $name_1$ $name_2$ ... – zjistí hodnotu daných atributů se jmény *name*,
- `set` $name_1$ $priority_1$ $value_1$ $name_2$ $priority_2$ $value_2$ ... – nastaví hodnoty a priority daných atributů,
- `unset-priority` $name_1$ $priority_1$ $name_2$ $priority_2$ ... – zruší prioritu daných atributu,
- `list` – vypíše seznam atributů a funkcí dané entity,
- `current` – vypíše současný stav atributů.

Při zpracovávání požadavků pro zařízení je potřeba zjistit, v jakém vztahu je současná a předchozí známá hodnota na jednotlivých prvcích. Liší-li se, je vygenerovaná zpráva prostřednictvím vrstvy zpráv o události na zařízení. Zprávy takto generované jsou dvou typů, v závislosti na tom, jestli jde o první hodnotu po připojení zařízení, tedy žádnou předchozí hodnotu nemá server k dispozici, či nikoli.

Akce nejsou popsány přímo ve strukturách serveru, ale mají odpovídat popisu rozhraní.

Popis naimplementovaných interface se nachází níže v sekci 3.7.

Již máme tedy popsané, jak jsou naimplementované interface zařízení. Zbývá ještě popsat, v jakých stavech se může zařízení po dobu běhu serveru nacházet. Jsou celkem čtyři: config, connecting, ready a broken. Přechody mezi stavy jsou znázorněny v obrázku 3.2.



Obrázek 3.2: Stavy zařízení a přechody mezi nimi

Popis jednotlivých stavů:

Config – načtení konfigurace.

Connecting – logické zařízení se snaží navázat spojení s fyzickým zařízením pomocí rozhraní vrstvy sběrnic.

Ready – spojení bylo úspěšně navázáno, může probíhat komunikace se zařízením.

Broken – nastal problém na úrovni sběrnice, se zařízením není možno komunikovat.

Popis přechodů mezi stavy:

Úvodní pokus o připojení – po načtení konfigurace se volá *fun_connect* daného zařízení.

Chyba 1 – chyba při připojování – možné příčiny jsou nefunkční/nepřipojené zařízení, špatná konfigurace.

Chyba 2 – chyba po připojení – zařízení přestalo odpovídat či došlo k rozbití celé sběrnice.

Interface mají stejný stav jako jejich zařízení.

3.5 Vrstva zpráv

Závěrečnou částí implementace serveru je vrstva zpráv. Obsahuje systém na zprostředkování komunikace mezi serverem a klienty pomocí zpráv. Také spravuje připojené klienty, umožňuje jim vytvářet atributy a komunikovat mezi sebou skrze server.

Pro předávání zpráv je použit textový protokol. Vstup od klientů je čten z příchozího spojení, připojeného k lokálnímu soketu (jméno je zvoleno v konfiguraci serveru). Vstup je rozdělen na zprávy a ty jsou dále rozděleny na *tokeny*. Tokenizovaná zpráva je poté zpracována.

Systém rozdělování vstupu na zprávy a tokeny je inspirován shellem. Vstup je zpracováván po zprávách, ukončených znakem nového řádku. Řádek je rozdělen na tokeny. Tokeny jsou odděleny nenulovým množstvím mezer či tabulátorů. Pro zadání některého ze speciálních znaků je možné obalit znaky do uvozovek. Mezi dvojicí uvozovek ztrácí mezery, tabulátory a nové řádky svůj speciální význam a stávají se součástí tokenu. Také lze použít zpětné lomítko pro odebrání speciálního významu následujícího znaku (včetně uvozovek). Několik ukázek tokenizování je k dispozici v následující tabulce. Všechny příklady jsou bez koncového znaku nového řádku.

vstup	token	token
`ukázkový␣text`	`ukázkový`	`text`
`"token␣s␣mezerami"`	`token␣s␣mezerami`	
`toto"␣je␣dlouhý␣"token`	`toto␣je␣dlouhý␣token`	
`"následuje␣uvozovka"␣\"`	`následuje␣uvozovka`	`"`

3.5.1 Příkazy

Příkazy (zprávy od klientů) se sestávají z nejméně dvou tokenů. První token označuje identifikátor příkazu, který bude také sloužit jako identifikátor všech odpovědí na tento příkaz. Druhý token je povinný, jelikož obsahově prázdný příkaz nemá smysl.

Seznam podporovaných příkazů (bez úvodního identifikátoru zprávy):

- `alive` – informace, že klient je stále připojen a funkční. Je potřeba ho periodicky posílat, aby server věděl že je klient v pořádku.
- `server` – sada příkazů komunikující se serverem jako celkem:
  - `server quit` – vypne server.
  - `server list` – vypíše všechny entity připojené k serveru a základní informace o nich.
  - `server log` – nastaví úroveň logování stavu serveru.
- `entity` *name function ...* – příkaz, který zavolá funkci *function* na entitě se jménem *name*, pokud entita existuje a má danou funkci. Zbývající tokeny jsou k dispozici funkci k dalšímu zpracování.
- `message` – sada příkazů na nastavení preferencí klienta:
  - `message set` *name type* – nastaví preference u entity *name* na typ *type*.
  - `message subscribe` *name* – nastaví preference na všechny typy zpráv s výjimkou debugovacích a zašle klientovi informaci o aktuálním stavu entity a jejích atributů.
  - `message subscribe-all` – jako subscribe na všechny entity v serveru.
  - `message propagate` – *type message* – server odešle od klienta zprávu *message* pro ostatní klienty, kteří mají zájem o zprávy typu *type* od tohoto klienta. *Type* je jeden z typů distribuovaných zpráv.
  - `message one` – *name type message* – server odešle od klienta zprávu *message* typu *type* pro klienta s jménem *name*. *Type* je jeden z typů zpráv pro konkrétního klienta.
- `client` – sada příkazů na uložení informací o klientovi na straně serveru:
  - `client name` *name* – nastavení jména klienta na *name*.
  - `client attr-add` *name type ...* – přidá klientovi atribut *name* typu *type* specifikovaný dalšími tokeny.
  - `client attr-set` *name priority value* – nastaví hodnotu atributu klienta, jako u interface.
  - `client attr-unset-priority` *name priority* – odnastaví hodnotu atributu klienta, jako u interface.

3.5.2 Zprávy od serveru konkrétnímu klientovi

Z předchozí kapitoly víme, že server posílá klientům dva druhy zpráv: pro konkrétního klienta a všeobecné informace o serveru. Každé zpráva od serveru začíná tokenem obsahující typ. Zde se zaměříme na zprávy určené pro konkrétního klienta. Jsou to jednak zprávy od serveru v reakci na příkaz od klienta a za druhé meziklientové zprávy. Typy zpráv jsou implementovány pomocí bitového maskování. Konkrétní typy zpráv konkrétnímu klienovi jsou:

0x040 – Zpracování příkazu proběhlo v pořádku. Může obsahovat další informace, například když klient žádal o zjištění hodnoty atributu.

0x080 – Zpracování příkazu skončilo s chybou. Obsahuje popis chyby.

0x100 – Jiný klient posílá klientovi zprávu skrze server. Obsahuje zprávu pro klienta.

Zprávy od serveru jsou v následujícím formátu:
type message_id name ...
Zde *type* je typ zprávy, *message_id* je identifikátor zprávy a *name* je jméno entity, od níž zpráva pochází.

3.5.3 Všeobecné zprávy od serveru

Všeobecné zprávy jsou filtrovány preferencemi podle entity a typu zprávy. Používá se zde stejná bitová maska jako pro zprávy pro konkrétní klienty. Konkrétní typy distribuovaných zpráv jsou:

0x001 – informace pro ladění,

0x002 – informace o změně hodnoty atributu při běhu serveru,

0x004 – podobně jako předchozí, ale nastává při připojování zařízení,

0x008 – informace o změně stavu celé entity (rozbití, spravení),

0x010 – konkrétní informace o chybách,

0x020 – informace o novém klientovi.

Všeobecná zpráva se od zpráv konkrétnímu klientovi liší pouze vynecháním identifikátoru zprávy.

3.6 Komunikační protokol klientů

Klienti jsou sice obecné programy, ale je potřeba stanovit konkrétní komunikační protokol. Jelikož má jít hlavně o programy běžící na stejném systému, na kterém běží server, budeme pro spojení používat lokální soket. Klienti mají několik povinností, které musí dodržovat při komunikaci se serverem.

Na začátku po připojení se musí představit unikátním jménem, které si sami volí. Toto jméno je následně použito při další komunikaci, obzvláště mezi klienty. Bez zadaného jména klient nemůže používat ostatní příkazy.

V průběhu připojení musí klienti pravidelně komunikovat, aby server měl přehled o stavu svých klientů. V konfiguraci serveru je uvedená doba, po které je nekomunikující klient odpojen. K dispozici je příkaz `alive`, pokud by klient neměl serveru co říci. Základní timeout je 60 s.

Délka příkazu nesmí přesáhnout délku definovanou v konfiguraci. Základní délka je 4096 znaků, což je výrazně více, než mají současné zprávy.

Následuje příklad komunikace mezi serverem a klientem. Jednotlivé zprávy jsou na řádcích, první znak označuje směr komunikace (`>` pro server, `<` od serveru):

```
> X entity bedroom_light set intensity 1 0
< X 32 bedroom_light set ok
< _ 2 bedroom_light intensity 1 0
```

Nejprve klient iniciuje na interface se jménem *bedroom_light* zhasnutí světla změnou jeho intenzity s vyšší prioritou. Druhý řádek obsahuje potvrzení přijetí zprávy od klienta. Poté, když dojde ke změně hodnoty atributu, je rozeslána zpráva na třetím řádku o aktuálním stavu atributu všem klientům, kteří o ni mají zájem podle svých preferencí.

3.7 Příklady interface

V serveru jsou použita následující rozhraní. Sloupec typ určuje, jestli se jedná o akci, atribut nebo funkci. Funkce zpracovávající atributy zde nejsou uváděny.

Rozhraní odpovídající světlu:

typ	*jméno*	*popis*
atribut	`intensity`	úroveň osvětlení, 0–255
funkce	`on`	nastaví intensity na 255 s prioritou 0
funkce	`off`	nastaví `intensity` na 0 s prioritou 0
funkce	`change`	přepíná `intensity` mezi 0 a 255 s prioritou 0

Rozhraní odpovídající spínači:

typ	*jméno*	*popis*
atribut	`state`	stav spínače, 0/1

Rozhraní odpovídající RFID čtečce:

typ	*jméno*	*popis*
akce	`newid` *id*	identifikátor *id* detekovaného RFID tagu

Rozhraní odpovídající teplotnímu čidlu:

typ	*jméno*	*popis*
atribut	`temperature`	poslední naměřená teplota, v desetinách °C
funkce	`measure`	provede změření teploty

Rozhraní odpovídající klávesnici:

typ	*jméno*	*popis*
akce	`pressed` *K*	zmáčknutá klávesa *K*

3.8 Bezpečnost

Jedním ze smyslů serveru je ovládání stav prvků rozmístěných po domácnosti. To může vést k usnadnění a zpříjemnění života obyvatelů domácnosti, ale i mnoha problémům v rukou potenciálního útočníka. Proto je důležité dbát na zabezpečení celého systému. Implementace serveru a klientů předpokládá, že všechny programy jsou spuštěny na jednom počítači, komunikují skrze lokální soket a přístup k těmto soketům lze omezit běžnými prostředky filesystémových práv.

Pokud by bylo žádoucí implementovat i vzdálený přístup k serveru, je potřeba vyřešit autentifikaci a šifrování. K tomu se používají prostředky zabezpečení na transportní vrstvě, například TLS [18]. V takovém případě by ale bylo i potřeba přidat systém přístupových práv určující, který klient smí provádět jaké příkazy.

4. Ukázkoví klienti

Zde rozebereme několik klientů používaných se serverem. Jedná se o sadu jednoduchých automatik a grafické uživatelské rozhraní k ovládání celého serveru, dále označované jako GUI.

4.1 GUI

Nejprve se budeme zabývat GUI. Je implementováno v jazyce C++ za použití knihovny Qt. Kromě základního zobrazení grafických prvků využívá také jazyk QML pro popis uživatelského rozhraní pro jednotlivé části serveru.

QML je zde použito jako relativně jednoduše upravitelný způsob popisu uživatelského rozhraní. To se hodí, pokud chceme dát uživateli možnost sestavit si uživatelské rozhraní pro ovládání vlastní domácnosti z již připravených prvků.

4.1.1 QML

QML (Qt Meta Language, nebo taky Qt Modelling Language) je deklarativní skriptovací jazyk vyvinutý společností Nokia určený k vývoji uživatelského rozhraní na dotykových platformách. Zakládá se na JavaScriptu a je schopný poskytnout frameworky jak pro jazyk C++, tak pro Android a jiná mobilní zařízení. Je také k dispozici vývojové prostředí Qt Creator, které umožňuje mimo jiné i drag and drop tvorbu QML souborů.

QML kód se sestává ze stromové struktury objektů. S jazykem QML je k dispozici knihovna základních QML objektů Qt Quick, obsahující mnoho základních objektů. Je možné tvořit vlastní QML objekty. Jméno nového objektu je určeno jménem souboru. Následuje kód příkladu „Hello, World!“ a jeho zobrazení, viz obrázek 4.1.

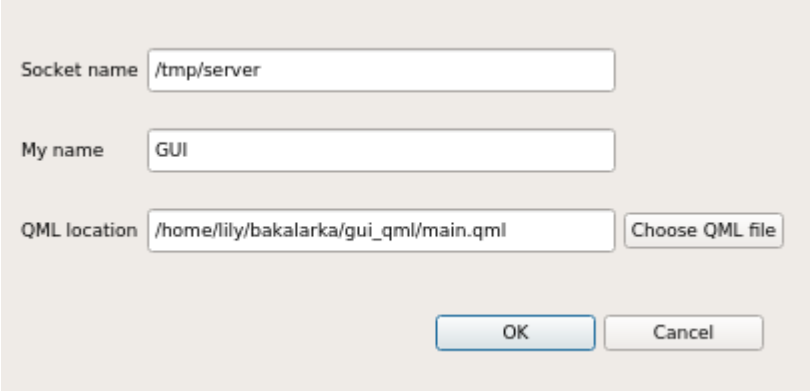
```
import QtQuick 2.2

Rectangle {
  width: 80
  height: 30
  border.width: 3
  Text {
    anchors.centerIn: parent
    text: "Hello, World!"
  }
}
```

Obrázek 4.1: QML Hello World

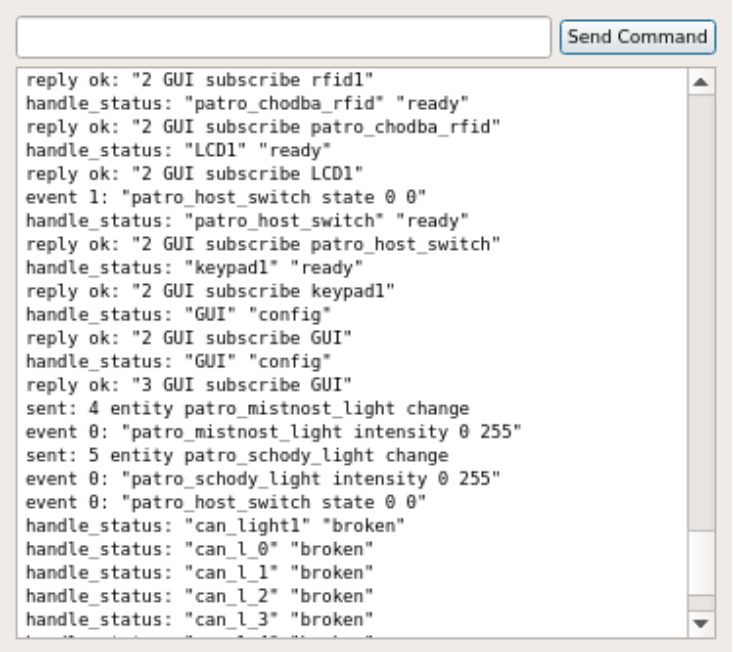
Nyní se zaměřme na to, jak GUI QML používá. GUI obsahuje možnost dodat do něj vlastní QML kód ke zobrazení. Při zachování pojmenovávacích konvencí je GUI následně schopno volat funkce v uživatelově QML struktuře při příchozích zprávách od serveru.

Když přijde zpráva od serveru popisující změnu stavu či událost některé entity, pokusí se GUI najít v dodaném QML struktuře objekt se stejným jménem. Pokud je nalezen, je mu předán zbytek zprávy ke zpracování. V QML může mít více objektů jedno jméno. QML prostředí má také k dispozici rozhraní GUI pro odesílání příkazů serveru.



Obrázek 4.2: GUI nastavení

GUI umožňuje nastavit jaký QML soubor má být zobrazen, k jakému lokálnímu soketu se má GUI připojit a jaké má mít jméno v rámci serveru, viz obrázek 4.2. Po spuštění se GUI pokusí připojit k serveru podle předchozí konfigurace. Pro pohodlí používání podporuje GUI více konfigurací (konfigurační soubor je možné zadat jako první parametr při spuštění). Dále je k dispozici log událostí s možností odesílat příkazy serveru ručně, viz obrázek 4.3.



Obrázek 4.3: GUI log a manuální odesílání příkazů

Součástí této práce je implementace několika QML objektů, které odpovídají naimplementovaným interface. Následuje příklad QML objektu reprezentující světlo (soubor `Light.qml`). Kliknutím na něj dojde ke změně stavu světla.

```
import QtQuick 2.2

Rectangle {
  width: 29
  height: 29
  color: "#00000000";
  function handleMsg(tokens) {
    if(tokens[2] > 0) {
      light_circle.color = "yellow"
    }else {
      light_circle.color = "white"
    }
    if(tokens[1] >0) {
      color = "green";
    }else {
      color = "#00000000";
    }
  }
  function status(s) {
    if(s === "broken") {
      color = "red";
      light_circle.color = "red";
    }else if(s === "ready") {
      color = "#00000000";
      light_circle.color = "white";
    }
  }
  Rectangle {
    x: 1;
    y: 1;
    width: 27;
    height: 27;
    id: light_circle
    color: "white"
    border.color: "black"
    border.width: 1
    radius: width*0.5
    MouseArea {
      anchors.fill: parent
      onClicked: msg.sendMessage("entity " +
          parent.parent.objectName + " change");
    }
  }
}
```

Jeden z dostupných QML objektů jsou záložky (TabView). Je poté možné si například rozložit ovládací prvky domácnosti do několika záložek, například

Obrázek 4.4: Světlo: zapnuto, vypnuto, rozbito

po patrech či po místnostech. Následuje ukázka o dvou záložkách reprezentující patra domu.

```
import QtQuick 2.1
import QtQuick.Controls 1.2
import QtQuick.Window 2.0
import "."

TabView {
  id: tabV
  objectName: "top"
  width: 800
  height: 600
  Tab {
    active: true
    objectName: "tab_patro"
    title: "Patro"
    Patro {

    }
  }
  Tab {
    active: true
    objectName: "tab_prizemi"
    title: "Prizemi"
    Prizemi {

    }
  }
}
```

Dále následuje ukázka objektu Patro, využívající mimo jiné dříve ukázaný objekt Light.

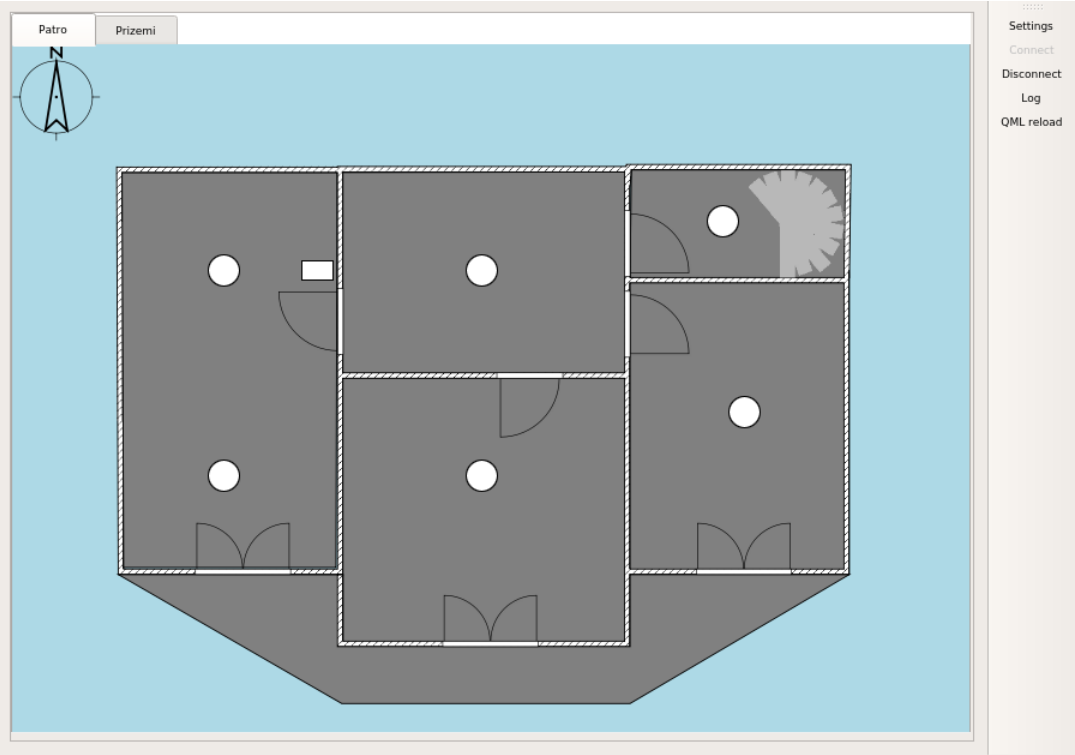
```
import QtQuick 2.0
import QtQuick.Controls 1.2
import "qml_objects"

Rectangle {
  objectName: "patro"
  id: patro
  width: 800
  height: 550
  color: "#add9e6"
  Image {
   x: 0
```

```
    y: 0
     sourceSize.width: 800
     sourceSize.height: 550
     source: "svg/stavba_patro.svg"
     Light {
       id: patro_host_light
       objectName: "patro_host_light"
       x: 162
       y: 174
     }
     Light {
       id: patro_mistnost_light
       objectName: "patro_mistnost_light"
       x: 377
       y: 345
     }
     Light {
       id: patro_schody_light
       objectName: "patro_schody_light"
       x: 578
       y: 133
     }
     Light {
       id: patro_chodba_light
       objectName: "patro_chodba_light"
       x: 377
       y: 174
     }
     Light {
       id: patro_mistnost2_light
       objectName: "patro_mistnost2_light"
       x: 596
       y: 292
     }
     Light {
       id: patro_mistnost3_light
       objectName: "patro_mistnost3_light"
       x: 162
       y: 345
     }
     Switch {
       id: patro_host_switch
       objectName: "patro_host_switch"
       x: 240
       y: 174
     }
   }
}
```

Jak může GUI vypadat, vidíme na následujících screenshotech aplikace. Na obrázku 4.5 vidíme stav GUI po spuštění. Zobrazuje se první záložka popisující patro

domácnosti. Vidíme na něm sadu světel a jeden spínač, reprezentovaný obdélníkem. Vpravo nahoře je zobrazeno schodiště se světlem, které se objevuje také v záložce přízemí.



Obrázek 4.5: GUI patro po spuštění

Na obrázku 4.6 je vidět stav GUI po připojení, rozsvícení dvou světel a sepnutí spínače.

Na posledním obrázku 4.7 je vidět přízemí domu, kde jsou některá světla rozbita, protože došlo k přerušení napájení zařízení Sv2, které se o ně stará. Také je zde vidět reprezentace klávesnice, RFID a teplotního čidla. Klávesnice je reprezentována vlevo nahoře, zobrazuje poslední hodnotu a při nové události zabliká modře. Podobně je udělaná i RFID reprezentace, která se nachází pod klávesnicí.

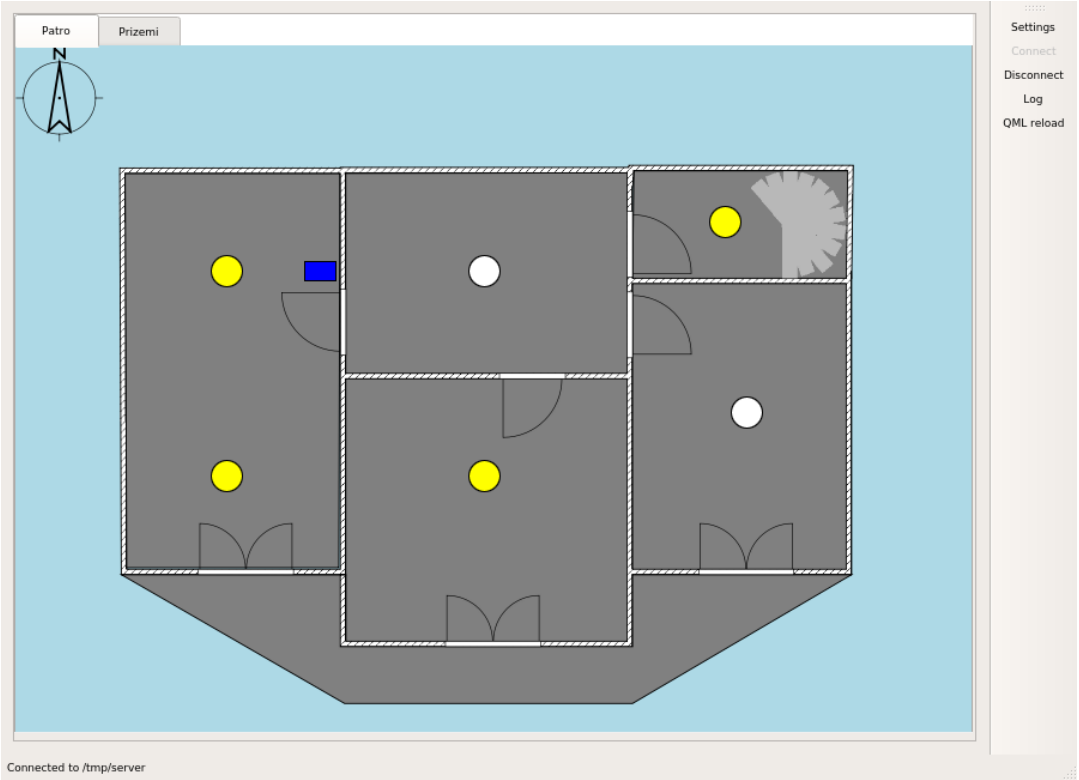
Nákresy domu byly vytvořeny pomocí programu Sweet Home 3D [26] a neodpovídají skutečnému domu.

4.2 Jednoduchá automatika

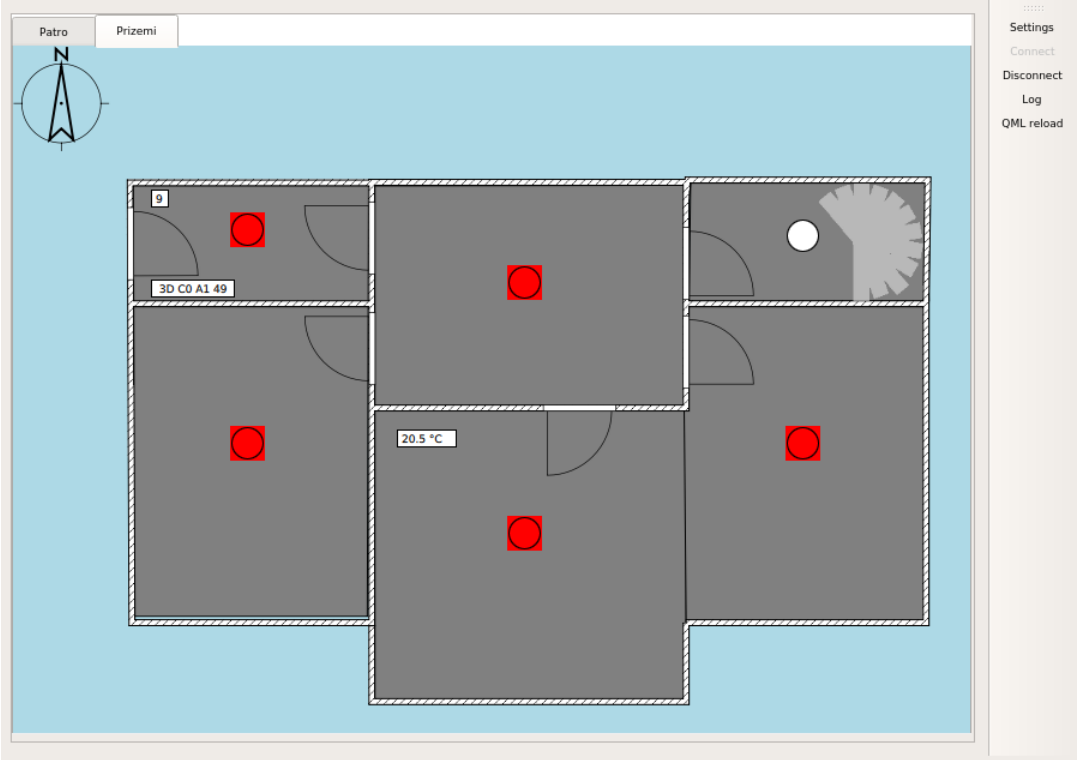
Všechny jednoduché automatiky jsou psané v jazyce C a využívají služeb knihovny LibUCW [14]. Využívají moduly Mainloop a Conf. Zpravidla se jedná o přímé či zpožděné reakce na události.

4.2.1 Spínač

Spínač je klient zajištující ovládání světla na základě sady spínačů. Obsahuje dva režimy, jednoduchý a časovaný.



Obrázek 4.6: GUI patro po připojení



Obrázek 4.7: GUI přízemí rozbité

V jednoduchém režimu program hlídá sadu daných interface spínačů a při změně sepnutí některého z nich změní stav vypnuto/zapnuto přiřazeného světla.

V režimu s časovaným zhasnutím je při zhasnutém světle při stisknutí libovolného ze spínačů světlo rozsvíceno na daný časový interval. Při rozsvíceném světle je prodloužena doba svícení o daný časový interval. Tato varianta poskytuje klientský atribut obsahující timestamp, kdy světlo zhasne.

Ukázka konfigurace jednoduchého režimu:

```
Switch {
  name spinac1
  socket /tmp/server
  mode simple
  lightName controlled1
  Interface {
    name patro_host_switch1
  }
}
```

Ukázka konfigurace časovaného režimu:

```
Switch {
  name spinac_casovany1
  socket /tmp/server
  mode timed
  lightTime 3000 # in milliseconds
  lightName controlled1
  Interface {
    name patro_host_switch1
  }
}
```

4.2.2 Termostat

Termostat je klient, který na základě změny teploty umožňuje ovládat relé (v příkladu jde světlo).

Má nastavitelnou podporu hystereze, například na zkompenzování zpožděné zpětné vazby prostředí a šumu čidel. Zpožděná zpětná vazba je jev, kdy je odpověď prostředí na regulační zásah zpožděna. V případě termostatu chvíli trvá, než je přeneseno teplo z kotle do radiátorů a z nich do vzduchu v okolí termostatu samotného.

Hystereze označuje například situaci, kdy se systém může nacházet ve dvou stavech, mezi kterými máme definovány různé prahové hodnoty pro oba přechody mezi stavy. Obecně může být i více než dva prahy při větším počtu stavů.

Proto termostat umožňuje nastavit velikost hystereze. Například dejme tomu, že je hystereze nastavena na 1 °C, teplota k udržování je nastavena na 21 °C a současná teplota je 19 °C. Termostat zapne zdroj tepla a vypne ho ve chvíli, kdy naměří teplotu 21 °C. K dalšímu zapnutí zdroje tepla dojde, až když teplota klesne na 20 °C.

Ukázka konfigurace:

```
Thermostat {
  name termostat
  socket /tmp/server
  hysteresis 10 # tenths of degree C
  splitTemperature 210 # tenths of degree C
  temperatureName controlled2
  temperatureName can_temp1
}
```

4.2.3 Klávesnice

Klávesnice je klient, který převádí zmáčknutou klávesu na klávesnici na přepínání příslušného světla. Jedno světlo může být ovládáno i více tlačítky a jedno tlačítko může ovládat více světel.

Ukázka konfigurace:

```
Keypad {
  name klavesnice
  socket /tmp/server
  keypad_name keypad1
  Interface {
    light_name can_l_0
    char 0
  }
  Interface {
    light_name can_l_1
    char 1
  }
  Interface {
    light_name can_l_2
    char 2
  }
  Interface {
    light_name can_l_3
    char 3
  }
  Interface {
    light_name can_l_4
    char 9
  }
  Interface {
    light_name can_l_5
    char B
  }
  Interface {
    light_name patro_mistnost_light
    char C
  }
  Interface {
```

```
    light_name patro_mistnost3_light
    char D
  }
}
```

5. Testovací prostředí

V této kapitole se nachází popis hardware sběrnic a zařízení použitých pro testování serveru. Konkrétně se jedná o zapojení dvou sběrnic, Modbus přes RS485 a CAN sběrnice přes vzdálený počítač (BananaPi). Jednotlivá zařízení jsou postavena na platformě Arduino. Přehled toho, jaká fyzická zařízení a sběrnice zde potkáme, byl již ukázán dříve v obrázku 2.2.

5.1 Arduino

Arduino je open-source vývojový kit založený na mikrokontroleru ATMega. Je možné zakoupit již zkompletované zařízení, či si jej složit z jednotlivých součástek za použití dostupných schémat a návrhů plošných spojů. Součástí vývojového kitu je IDE zajišťující překlad programů v jazyce Arduino Programmable Language a jejich nahrání na čip připojeného Arduina. Jazyk Arduino Programmable Language vychází z jazyků C a C++.

Jednotlivá zařízení nabízí možnost připojovat další prvky a komunikovat s nimi prostřednictvím analogových či digitálních pinů.

5.2 Sběrnice a zařízení

Sestavení samotných zařízení a sběrnic je pro testovací účely provedeno na nepájivých deskách.

5.2.1 Sběrnice CAN

Fyzická sběrnice CAN samotná se skládá z dvou vodičů, CAN H a CAN L. Je připojena k jednodeskovému počítači BananaPi, který rovnou obsahuje CAN transceiver. Software spuštěný na BananaPi pro komunikaci s CAN budeme označovat jako „vzdálená CAN“. Komunikace mezi vzdálenou CAN a fyzickou sběrnici je provedena pomocí ovladače Can4linux [19].

Komunikaci mezi serverem a vzdálenou CAN je provedena pomocí TCP/IP spojení a jednoduchého textového protokolu, který posílá hexadecimální reprezentaci jednotlivých bajtů CAN zpráv mezi serverem a software spolu s identifikátorem zprávy a řídícími informacemi.

Jednotlivá zařízení jsou připojena k fyzické sběrnici pomocí čipu MCP2515 a komunikují s čipem skrze rozhraní SPI. Čip podporuje filtrování pomocí masek a filtrů.

Komunikace po sběrnici samotné probíhá rychlostí 500 kb/s.

5.2.2 Sběrnice Modbus

Fyzická sběrnice Modbus je tvořena čtyřmi vodiči, A, B, 5V a zem. Sběrnice je na obou koncích zakončena rezistory o hodnotě 120 Ω. Pro prevenci rušení jsou vodiče A, resp. B staženy k zemi, resp. 5V. Fyzická sběrnice je připojena k počítači se serverem pomocí převodníku z USB na RS485. Komunikace poté probíhá

skrze příslušný soubor v `/dev/` . V serverovém ovladači je přímo implementovaná komunikace pomocí protokolu Modbus. Vychází z kódu knihovny libmodbus [21], ale kód je upraven pro spolupráci s Record I/O.

Jednotlivá zařízení jsou připojena ke sběrnici pomocí vlastního obvodu, který obsahuje volitelné galvanické oddělení sběrnice od zařízení a čip 75176A [23] pro komunikaci pomocí RS485. Tento čip zprostředkovává komunikaci mezi zařízením a sběrnicí pomocí vodičů RST, Tx a Rx.

Pro komunikaci protokolem Modbus je použita vlastní knihovna vycházející z knihovny simple-modbus [22] pro Arduino. Implementuje příkazy Modbusu pro čtení a zápis do uchovávacích registrů.

Komunikace po sběrnici samotné probíhá rychlostí 9.6 kbps v režimu RTU. Modbus funguje v master/slave režimu, je tedy nutné se pravidelně ptát zařízení na jejich aktuální stav.

5.2.3 Zařízení Sv1

Fyzické zařízení Sv1 se skládá z Arduino Nano připojeného ke sběrnici Modbus. Na jeho digitální piny je připojeno 8 LED. Zařízení má adresu 200 v rámci sběrnice. Uchovávací registry 0–7 odpovídají jednotlivým LED, možné hodnoty jasu jsou 0–255.

5.2.4 Zařízení S+K

Fyzické zařízení S+K se skládá z Arduino Nano připojeného ke sběrnici Modbus. Na jeho digitální piny je připojena 16tlačítková klávesnice a spínač. Zařízení má adresu 211 v rámci sběrnice. Uchovávací registry zařízení jsou popsány v následující tabulce.

registr	obsah
0	aktuální stav spínače
1	počet zmáčknutí kláves
2	indikátor, je-li aktuálně zmáčknuta klávesa
3–7	posledních 5 zmáčknutých kláves

5.2.5 Zařízení RFID

Fyzické zařízení RFID se skládá z Arduino Nano připojeného ke sběrnici Modbus. K němu je přes sběrnici SPI připojena RFID čtečka MFRC522 [24]. Při detekování RFID tagu v blízkosti čtečky je uloženo identifikátor daného tagu. Zařízení má adresu 100 v rámci sběrnice. Krom samotného posledního detekovaného tagu dává zařízení i k dispozici počet detekovaných tagů. Porovnáním aktuálního a předchozího počtu ovladač zařízení určí, má-li vyvolat událost nového RFID tagu. Uchovávací registry zařízení jsou popsány v následující tabulce.

registr	obsah
0	počet detekovaných RFID
1	délka posledního id tagu v bajtech
2–6	poslední detekovaný id tag

5.2.6 Zařízení Sv2

Fyzické zařízení Sv2 se skládá z Arduino Mini připojeného ke sběrnici CAN. Na jeho digitální piny je připojeno 6 LED. Zařízení si filtruje od sběrnice zprávy s identifikátorem 210 a vysílá zprávy s identifikátorem 211. Jednotlivé led jsou nastavovány podle hodnot na datových pozicích 0–5 ve zprávě. Po nastavení je odeslána potvrzující zpráva s aktuálními hodnotami.

5.2.7 Zařízení Teplota

Fyzické zařízení Teplota se skládá z Arduino Mini připojeného ke sběrnici CAN. Je k němu připojeno teplotní čidlo DS18S20 [25] pomocí sběrnice OneWire. Zařízení si filtruje od sběrnice zprávy s identifikátorem 42 a vysílá zprávy s identifikátorem 43. Při příchozí zprávě provede měření a odešle zprávu s aktuálně naměřenou teplotou. Jednotka je 1/16 °C. Server poté tuto teplotu převede na desetiny stupně Celsia.

6. Uživatelská dokumentace

V příloze této práce naleznete zdrojové kódy všech programů a doplňkové soubory. Následuje konkrétní popis obsahu přílohy:

arduino_modbus/ Arduino zdrojový kód pro Modbus komunikaci.

auto/ zdrojové kódy jednoduchých automatik.

- **keypad/** automatika Klávesnice.
- **switch/** automatika Spínač.
- **thermostat/** automatika Termostat.

can_remote/ zdrojový kód pro vzdálenou část CAN sběrnice.

config/ konfigurační soubory pro jednoduché automatiky a server.

gui/ zdrojový kód GUI.

gui_qml/ ukázkové QML soubory a obrázky pro GUI.

- **qml_objects/** připravené QML objekty reprezentující rozhraní.
- **svg/** podkladové obrázky.
- **main.qml** hlavní QML soubor.

libucw/ knihovna LibUCW.

Makefile pro zkompilování zdrojových kódů.

server/ zdrojový kód serveru.

- **doc/** dokumentace popisující soubory.

sketchbook/ zdrojové kódy pro jednotlivé Arduino zařízení a potřebné knihovny.

- **libraries/** použité knihovny.
- **can_light/** Arduino kód pro zařízení Sv2.
- **can_temp/** Arduino kód pro zařízení Teplota.
- **modbus_sk/** Arduino kód pro zařízení S+K.
- **modbus_light/** Arduino kód pro zařízení Sv1.
- **modbus_rfid/** Arduino kód pro zařízení RFID.

6.1 Instalace

Pro zkompilování programů s výjimkou vzdálené CAN stačí spustit příkaz `make` v hlavním adresáři. Pro zkompilování vzdálené CAN je potřeba spustit příkaz `make can_remote`.

Server, vzdálená CAN a jednoduché automatiky vyžadují pro nainstalování a spuštění knihovnu LibUCW [14]. Vzdálená CAN vyžaduje také knihovnu can4linux. Pro zkompilování a spuštění GUI je potřeba Qt verze minimálně 5.3.

Zkompilované soubory jsou následně k dispozici ve složce `bin/`. Server je potřeba spouštět z hlavního adresáře příkazem `bin/server config/server.cf` z důvodu relativních cest při načítání konfigurace. Případně je možné upravit cesty v konfiguračním souboru na absolutní.

Vzdálená CAN je spuštěna příkazem `bin/can_remote` *`dev port`*. Dev značí cestu k CAN zařízení (např. `/dev/can0`), port značí na kterém portu má program poslouchat na příchozí TCP/IP spojení.

Pro nahrávání programů do jednotlivých zařízení je potřeba Arduino verze minimálně 1.0.7.

Pro zkompilování zařízení připojených ke sběrnici Modbus je potřeba přidat do prostředí Arduino novou konfiguraci, nacházející se ve složce `arduino_modbus`. Tuto složku je potřeba přidat do složky *`ARDUINO`*`/hardware/arduino/cores` a následně v *`ARDUINO`*`/hardware/arduion/boards.txt` vytvořit záznam o této konfiguraci. ARDUINO značí adresář obsahující instalaci Arduino software

V této práci jsou použity následující knihovny pro Arduino:

Arduino-Temperature-Control-Library – knihovna pro komunikaci s teplotními čidly pomocí sběrnice OneWire.

CAN_BUS_Shield_master – knihovna starající se o komunikaci s čipem MCP2515 pro připojení zařízení ke sběrnici CAN pomocí sběrnice SPI.

MFRC522 – knihovna pro komunikaci čipem MRFC522 pro čtení RFID tagů pomocí sběrnice SPI.

OneWire – knihovna pro komunikaci pomocí OneWire protokolu.

6.2 Formát konfigurace serveru

Server je konfigurovatelným prostřednictvím konfiguračního souboru ve formátu podle LibUCW [16]. Cesta ke konfiguračnímu souboru je serveru předána jako první parametr při spuštění. Následuje ukázka konkrétní konfigurace serveru.

První částí konfigurace je soubor `config/server.cf`, který obsahuje obecná nastavení serveru a informace pro nastavení pro komunikaci s klienty. Na konci jsou vloženy soubory obsahující konfiguraci jednotlivých sběrnic.

```
Server {
  LogName /tmp/server.log
  LogLevel 31 #message_type

  #clients
  SocketName /tmp/server
```

```
  SocketReadBufSize         4096   # input buffer size
  SocketReadRecordMax       1024   # max command lenght
  SocketWriteBufSize        4096   # output buffer size
  SocketWriteThrottleRead   3072   # threshold
  ReconnectTime             60000  # milliseconds

  Include config/server_modbus.cf  # Modbus configuration
  Include config/server_can.cf     # CAN configuration
}
```

Dále následuje ukázka konfigurace sběrnice Modbus nacházející se v souboru `config/server_modbus.cf`. Nejprve je popsána sběrnice a poté následují jednotlivá zařízení.

```
BusModbus {
    # path to the Modbus device
  Ttypath /dev/serial/by-id/
      usb-FTDI_FT232R_USB_UART_A602UEVS-if00-port0
  TimeoutMillis 1000   # timeout for devices to respond
  Name  modbus1        # server name of this bus
  Baud  9600           # communication speed
  Parity E             # parity used
  Byte  8              # bytes used
  Stopbit 1            # stopbit used
}

DeviceModbusRFID {
  Bus modbus1          # server name of the bus
  Name  rfid1          # server name of this device
  Slaveid 100          # Modbus id
    # how often will the device be polled
  PollTime 1000
    # server name of the rfid interface
  RfidName vchod_rfid
}

DeviceModbusSK {
  Bus modbus1          # server name of the bus
  Name  s+k1           # server name of the device
  Slaveid 211          # Modbus id
    # how often will the device be polled
  PollTime 450
    # server name of the switch interface
  SwitchName patro_host_switch
    # server name of the keypad interface
  KeypadName keypad1
}

DeviceModbusLight {
  Bus modbus1          # server name of the bus
  Name  light          # server name of the device
  SlaveId 200          # Modbus id
```

```
    # server names of the light interface
  LightName0 patro_mistnost_light
  LightName1 patro_host_light
  LightName2 patro_chodba_light
  LightName3 patro_schody_light
  LightName4 patro_mistnost2_light
  LightName5 patro_mistnost3_light
  LightName6 controlled1
  LightName7 controlled2
}
```

Závěr

V této práci jsme navrhli systém poskytující dalším programům abstraktní pohled na prvky domácí automatizace. Navržený systém je konstruován modulárně, aby bylo relativně snadné ho rozšiřovat o ovládání dalších prvků. Předvedli jsme jeho možnou implementaci a vyzkoušeli ji na skutečném testovacím hardware.

Samotné řízení jednotlivých prvků je poté přenecháno na starost externím programům, které se k systému připojují. Ukázali jsme několik programů starajících se o chod domácnosti pomocí prvků v ní skrze tento systém. Také jsme předvedli grafické uživatelské rozhraní s možností použití uživatelem vytvořené vizuální reprezentace domácnosti.

Ukázková implementace potvrdila, že návrh je funkční a prakticky použitelný. V budoucnu je možné ho dále rozšiřovat o další funkce.

– ASK jak lépe uvést následující?

Bude potřeba doplnit podporu pro další druhy fyzických zařízení a sběrnic. Například LCD displeje na zdech pro zobrazování stavu domu a přímé ovládání dotykem či tlačítky bez potřeby dalšího zařízení.

Dále by se v některých situacích mohlo hodit ovládat skupiny zařízení najednou, například světla rozmístěná po větším pokoji. Zde by mohlo být použité například virtuální rozhraní, které by předávalo jednotlivé příkazy všem interfacům.

Domácnost nemusí být nutně statické prostředí a občas by mohlo být pro uživatele příjemné mít možnost změnit konfiguraci serveru za jeho běhu (například přidávat zařízení). Jeden ze způsobů jak to umožnit, je provést znovunačtení konfigurace a detekování změn oproti předchozímu stavu.

Dalším možným rozšířením je vytvořit distribuovanou verzi serveru. Server by byl spuštěn ve více instancích na různých počítačích, ale navenek by fungoval jako jeden celek. Servery by mohly být spojeny například pomocí TCP/IP. Toto by nejspíše vyžadovalo zavedení systému uživatelů a jejich práv, což je další funkce, o kterou by systém bylo možné v budoucnu rozšířit.

Příjemný by mohl být rovněž klient fungující podobně jako GUI, ale fungující skrze webový prohlížeč.

Seznam použité literatury

[1] Modbus. *MODBUS Application Protocol Specification V1.1b3* [online]. [cit. 2014-05-01] Dostupné z: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[2] Modbus-IDA.ORG. *MODBUS over serial line specification and implementation guide* [online]. [cit. 2014-05-01] Dostupné z: http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

[3] AWTREY, Dan. *Design guide – introduction to 1-wire network issues* [online]. [cit. 2014-05-01] Dostupné z: http://www.1wire.org/Files/Articles/1-Wire-Design%20Guide%20v1.0.pdf

[4] DUDÁČEK, K. *Sériová rozhraní SPI, Microwire, I2C a CAN* [online]. [cit. 2014-05-01] Dostupné z: http://home.zcu.cz/~dudacek/NMS/Seriova_rozhrani.pdf

[5] SOLTERO Manny, ZHANG Jing, and COCKRIL Chris. *RS-422 and RS-485 Standards Overview and System Configurations* [online]. [cit. 2014-05-02] Dostupné z: http://www.ti.com/lit/an/slla070d/slla070d.pdf

[6] ZigBee Alliance. *ZigBee Specification Document 053474r17* [online]. [cit. 2014-05-02] Dostupné z: http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/kjb79_ajm232/pmeter/ZigBee%20Specification.pdf

[7] PETERKA Jan. *Synchronní, asynchronní a arytmický přenos* [online]. [cit. 2014-05-02] Dostupné z: http://www.earchiv.cz/a96/a650k150.php3

[8] KNX. *KNX System Specifications* [cit. 2014-05-07] Dostupné z: http://www.knx.org/media/docs/downloads/KNX-Standard/KNX%20Standard%20Public%20Documents/03_01_01%20Architecture%20v3.0.zip

[9] BOSCH. *CAN Specification Version 2.0* [online]. [cit. 2014-05-07] Dostupné z: http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf

[10] NXP Semiconductors. *THE I²C-BUS SPECIFICATION VERSION 2.1* [online]. [cit. 2014-05-09] Dostupné z: http://www.nxp.com/documents/user_manual/UM10204.pdf

[11] *Standard and Extended X10 Code Protocol* [online]. [cit. 2014-05-21] Dostupné z:

http://developer.telldus.com/raw-attachment/ticket/124/X10%20xtdcode.pdf

[12] *BananaPi* [online] [cit. 2014-10-05] Dostupné z: http://www.bananapi.com/index.php/component/content/article?layout=edit&id=24

[13] *CAN-Bus-frame in base format without stuffbits* [online] [cit. 2014-12-05] Dostupné z Wikipedie – otevřené encyklopedie: http://commons.wikimedia.org/wiki/File:CAN-Bus-frame_in_base_format_without_stuffbits.svg

[14] Martin Mareš, Robert Špalek et al. *LibUCW* [online] [cit. 2015-04-29] Dostupné z: http://www.ucw.cz/libucw/

[15] Martin Mareš, Robert Špalek et al. *LibUCW Configuration files documentation* [online] [cit. 2015-04-29] Dostupné z: http://www.ucw.cz/libucw/doc/ucw/config.html

[16] Martin Mareš, Robert Špalek et al. *LibUCW Configuration parser documentation* [online] [cit. 2015-04-29] Dostupné z: http://www.ucw.cz/libucw/doc/ucw/conf.html

[17] Martin Mareš, Robert Špalek et al. *LibUCW Mainloop documentation* [online] [cit. 2015-04-29] Dostupné z: http://www.ucw.cz/libucw/doc/ucw/mainloop.html

[18] Dierks, T. and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246* [online], August 2008 [cit. 2015-04-30]. Dostupné z: http://www.rfc-editor.org/info/rfc5246

[19] *Can4linux projekt* [online] [cit. 2015-04-30]. Dostupné z: http://sourceforge.net/projects/can4linux/

[20] Microchip Technology Inc. *MCP2515 Stand-Alone CAN Controller With SPI™ Interface* [online] [cit. 2015-04-30]. Dostupné z: http://ww1.microchip.com/downloads/en/DeviceDoc/21801d.pdf

[21] *libmodbus* [online] [cit. 2015-04-30]. Dostupné z: http://libmodbus.org/

[22] *simple-modbus* [online] [cit. 2015-04-30]. Dostupné z: https://code.google.com/p/simple-modbus/

[23] Texas Instruments. *SN75176A Differential Bus Transceiver* [online] [cit. 2015-04-30]. Dostupné z: http://www.ti.com/lit/ds/symlink/sn75176a.pdf

[24] NXP Semiconductors N.V. *MFRC522 Standard 3V MIFARE reader solution* [online] [cit. 2015-04-30]. Dostupné z: http://www.nxp.com/documents/data_sheet/MFRC522.pdf

[25] NXP Semiconductors N.V. *DS18S20 High-Precision 1-Wire Digital Thermometer* [online] [cit. 2015-04-30]. Dostupné z: `http://datasheets.maximintegrated.com/en/ds/DS18S20.pdf`

[26] *Sweet Home 3D* [online] [cit. 2015-04-30]. Dostupné z: `http://www.sweethome3d.com/`