České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Platformově nezávislá aplikace určená ke správě zabezpečovacího zařízení

Pavel Bucek

Vedoucí práce: Ing. Pavel Kubalík

Studijní program: Informatika a výpočetní technika

červen 2006

Poděkování

Rád bych poděkoval Pavlovi Kubalíkovi za umožnění realizace námi vymyšleného projektu jako bakalářkou práci. Mé díky patří také kolegům Jiřímu Francovi a Lukášovi Přívozníkovi za vstřícnou spolupráci.

Prohlášení

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

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Mladé Boleslavi dne 21.6. 2006 .....

Abstract

This work is a part of complex concept, which is handling building security and administration. The primal subject of this thesis is design, build and produce server and client application. Server application will transport secure data messages between web application and home security device. Client application will be runnable on various operating systems with support Microsoft Windows and Linux operating system together and will allow to set or modify home security device settings.

Abstrakt

Tato práce je dílčí částí projektu, který zajišťuje zabezpečení a správu objektu pomocí multifunkčního zařízení. Zařízení je možno nastavovat pomocí klientské aplikace nebo webového rozhraní. V následující práci se zabývám implementací dvou částí – serverem pro výměnu dat mezi webovým rozhraním a multifunkčním zařízením a klientskou aplikací.

Cílem serveru je spolehlivý a bezpečný přenos dat mezi zmíněnými součásti. Zároveň bude zaznamenávat události generované zařízením do databáze pro potřeby webového rozhraní a také realizovat naplánované události. Server bude implementován v jazyce C++, protože má velmi dobrou podporu v operačním systému Linux, dá se jím dobře přistupovat k systémovým rozhraním a umožňuje pokročilou práci s procesy a sockety.

Cílem klientské aplikace je umožnit uživateli nastavení chování zařízení, získávání aktuálních hodnot (například z teplotního čidla) a odpovídajícím způsobem je reprezentovat. Aplikace bude vytvořena pomocí knihovny Qt od společnosti Trolltech. Knihovna byla vybrána pro možnost její kompilace pod různými operačními systémy, což zaručuje platformovou nezávislost aplikace.

Obsah

Seznam obrázků

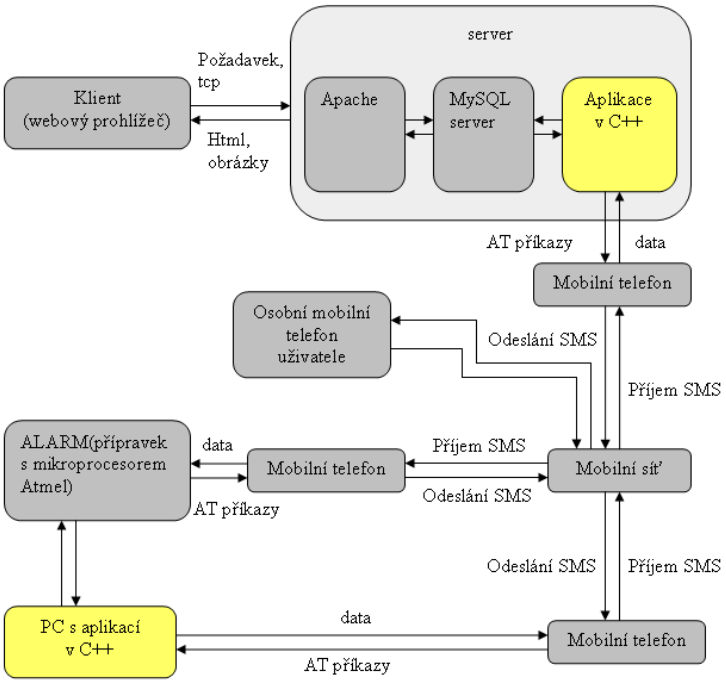
Seznam tabulek

1 Úvod

Má práce se skládá ze dvou částí – server a klientská aplikace. Server bude sloužit jako prostředník mezi web serverem a mobilním telefonem, dále bude umět obsluhovat některé události a zaznamenávat jejich výstupy do databáze. Klientská aplikace bude sloužit k nastavování a získávání dat z přípravku, nejdříve pouze pomocí sériové linky, později i pomocí mobilního telefonu. Bude také možno uložit konfiguraci do souboru a nastavit podle ní další přípravek.

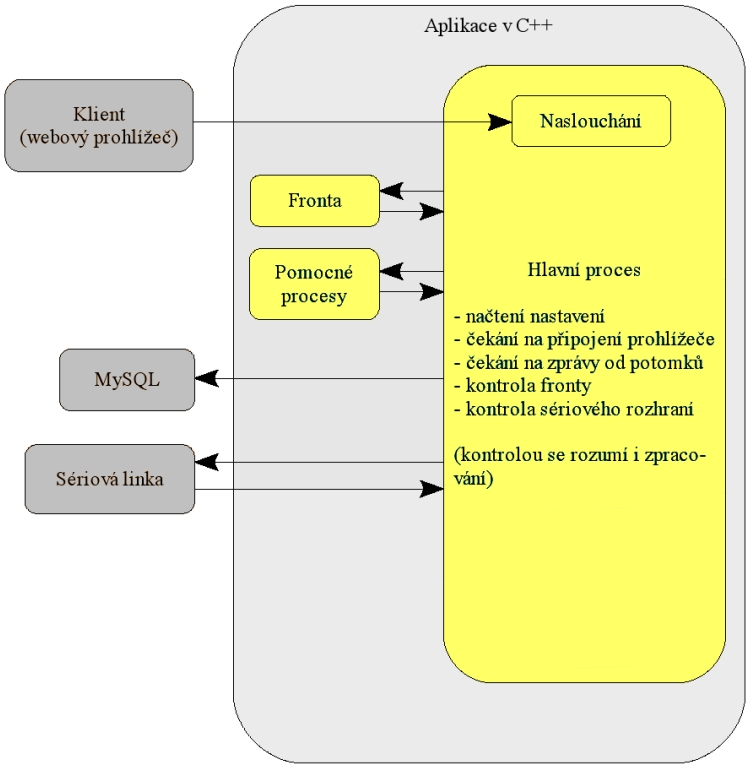
Výše popsané programy jsou součásti projektu **GSM ALARM HOME**, který bude po dokončení zajišťovat komplexní správu zabezpečení objektů. Pomocí webového rozhraní bude možno zobrazit aktuální stav jednotlivých přípravků a měnit jejich nastavení. Přípravek bude moci být spravován i pomocí klientské aplikace (využití například koncovými zákazníky bez přístupu k webovému rozhraní nebo k hromadnému programování přípravků). Přípravek samotný bude schopen jako reakci na vzniklou událost (například překročení prahové hodnoty) zavolat, prozvonit nebo odeslat SMS na dané telefonní číslo. Přehled součástí můžete vidět na obrázku 1.1 (tato práce se zabývá zvýrazněnými částmi).

Po dokončení by měl projekt zajišťovat bezpečnost objektu a zároveň umožňovat uživateli dálkovou správu a diagnózu. Předpokládá se propojení s dalšími navrhovanými přípravky (například autoalarm s možností zjištění polohy).



Obrázek 1.1: Celkový pohled na projekt **GSM ALARM HOME**

Na dalším obrázku si můžete prohlédnout podrobnější schéma serverové aplikace.



Obrázek 1.2: Podrobné schéma serverové části

Popis obrázku 1.2:

- Komunikace se sériovou linkou probíhá pomocí funkcí pro manipulaci se soubory
- Komunikace s MySQL databází probíhá pomocí knihovny MySQL++
- Fronta je mnou implementovaná třída, která bude podrobněji popsána v kapitole Implementace
- Všechna ostatní spojení jsou realizována pomocí socketů
  - `PF_INET` (IPv4 Internet protocols) pro prohlížeč
  - `PF_UNIX` (Local communication) pro komunikaci mezi procesy

1.1 Analýza trhu

Zařízení využívající GSM síť ke komunikaci s uživatelem nebo například pultem centrální ochrany jsou na trhu již celkem rozšířené. V následujících sekcích popíšu základní funkci pár vybraných zařízení a potom je porovnám se specifikacemi našeho projektu.

HouseDog GSM Alarm

První zde představovaný výrobek je do firmy *Protronix, spol. s r.o.*. Svůj výrobek prezentuje jako inteligentní zabezpečovací zařízení s vestavěným GSM modulem, který umí reagovat na porušení bezpečnosti prozvoněním, odesláním SMS zprávy nebo e-mailu.

Prostudováním uživatelského manuálu zjistíte, že zařízení dokáže pracovat ve dvou režimech (plná funkčnost nebo částečná s úsporou spotřeby). V režimu malé spotřeby reaguje pouze na své vstupy, při plné funkčnosti dokáže komunikovat i pomocí SMS příkazů (dokáže nastavit okamžitou hodnotu výstupů podle přijaté SMS). Zařízení také umí komunikovat s pulty centrální ochrany dvou společností.

Společnost Protronix nedodává k přípravku žádný software, dálková správa je realizována pouze pomocí textových zpráv. Nijak nezaznamenává historii provedených příkazů.

Technické parametry

Napájení: 1x230VAC±10% / 50Hz
Příkon: 5VA
Rozsah Pracovních teplot: 0°C až +40°C
Upevnění: montáž na stěnu
Výstupy: 2x přepínací kontakt 230VAC/10A
Vstupy: 2x bezpotecionálový kontakt 12V/10mA
Záložní akumulátor: 12V/1,3A

Cena: 7600 Kč

Informace o tomto výrobku můžete nalézt na adrese:
`http://www.protronix.cz/cs/produkty/housedog/`.

Patriot

Další zařízení distribuuje firma *Eurotel, a. s.*.

GSM modul Eurotel Patriot je schopen přenášet poplachové a doplňkové informace pomocí hlasu.V případě poplachu je schopen zavolat na uživatelem předdefinovaná čísla. Komunikace s Eurotel Patriotem je obousměrná, tzn. je možné i na modul zavolat a pomocí tónové volby zadat předdefinované příkazy. Dokáže komunikovat i pomocí SMS a Bluetooth.

Mezi praktické použití výrobku Eurotel Patriot patří kontrola zabezpečení automobilů, kontrola zabezpečení domů, chat a garáží, zapínání domácích spotřebičů na dálku, nebo odečet spotřeby energií v domácnostech.

K dispozici je i verze s GSM modulem, využitelná zejména pro zabezpečení automobilů. Toto zařízení může fungovat i jako telefon – umožňuje uskutečňovat hovory pomocí připojeného mikrofonu a reproduktoru. Také dokáže vzdáleně ovládat řídící prvky, například vypnout motor.

Obrázek 1.3: Zařízení Patriot od společnosti Eurotel

Společnost Eurotel nezveřejnila technické parametry tohoto výrobku.

Cena: zhruba 10000 Kč

Informace o tomto výrobku můžete nalézt na adrese: `http://www.eurotel.cz`.

GSM-PAGER P16 profi

Tento výrobek distribuuje společnost *ELNIKA plus, spol. s r.o.*

Zabezpečovací zařízení GSM-PAGER P16 profi je určen k zabezpečení, kontrole a ovládání motorových vozidel nebo rekreačních objektů a jejich vybavení. Ve spojení se systémem REX (systém pro ochranu, vyhledávání a logistiku vozidel, věcí a osob) poskytuje nepřetržitou kontrolu vašeho vozidla. Přenos poplachových a dalších provozních stavů je uskutečněn prostřednictví GSM sítě. GSM-PAGER P16 profi je určen ke skryté montáži ve vozidle s napájením z palubní sítě vozidla.

Přehled vlastností

Vstupy

- tři poplachové individuálně uživatelsky nastavitelné vstupy
- dva vstupy pro ovládaní zajištění / odjištění pageru
- jeden analogový vstup 0 až 30V pro dálkové měření analogových veličin (jeden externí a dva interní)
- možnost dálkové změřit napětí napájení pageru
- možnost dálkově změřit napětí baterie mobilu pageru.
- vstup pro připojení externího mikrofonu.
- obousměrná sériová linka RS232

Obrázek 1.4: GSM-PAGER P16 profi

- sběrnice pro rozšiřující moduly (I2C)

Výstupy

- 3 individuálně dálkově ovládané výstupy typu otevřený kolektor
- výstup pro připojení externí sirény s vratnou pojistkou
- zálohovaný výstup pro napájení připojených čidel vybavený vratnou pojistkou
- 1 externí LED pro signalizaci stavů.

Způsob ovládání:

- nastavitelné automatické poplachové volání a přenosu SMS zpráv až na 10 telefonních čísel
- dálkově pomocí krátkých textových zpráv SMS
- dálkově pomocí tónové volby DTMF
- možnost 5 způsobů programování:
  - lokálně pomocí editace položek v telefonním seznamu
  - lokálně pomocí PC přes sériovou linku RS232
  - dálkové pomocí SMS
  - dálkově pomocí DTMF
  - dálkově přes internet (pomocí některé SMS brány)

Tento přípravek také umožňuje dálkové zjištění polohy (pomocí GSM sítě nebo GPS modulu) a může být začleněn do celorepublikového systému pro ochranu a vyhledávání vozidel, věcí a osob (REX).

Technické parametry

Povolený rozsah napájecího napětí: 9 až 18V
Proudový odběr v klidovém stavu: < 0,02A
Maximální proudový odběr: 0,5A
Provozní pásmo: TRI BAND 900/1800/1900MHz
Rozměry: 114x80x33mm
Maximální hmotnost: 170g
Střední doba provozu ze záložní baterie: 1,3Ah 30hodin
Rozsah provozních teplot: -40 až +85°C
Počet logických vstupů: 3 + 2
Počet analogových vstupů: 1 + 2

Cena: okolo 8000 Kč

Informace o tomto výrobku můžete nalézt na adrese: `http://www.elnika.cz`.

Srovnání

Co se týče hardwarové stránky přípravků, nelze přesně určit co je lepší a co ne. Pokud se zaměřím na maximální počet obhospodařovávaných čidel, vyhraje nejspíš zařízení od společnosti Elnika plus s naimplementovanou sběrnicí I2C, pomocí které může efektivně spravovat velké množství senzorů.

Pokud se budu snažit porovnat přípravky s ohledem na možné nastavení a způsoby komunikace, začne náš produkt konkurovat zde uvedeným komerčním produktům. Jedině výrobek distribuovaný společností Eurotel umí navíc spolupracovat přes rozhraní Bluetooth, což zatím žádné z námi vytvořených zařízení nepodporuje. Jinak máme navíc správu pomocí webového rozhraní a aplikaci pro koncového uživatele.

Myslím, že pokud tento projekt dotáhneme do konce, přidáme další možnosti posílání upozornění (komunikace s pultem centrální ochrany) a rozšíříme možnosti přípravků (nebo sloučit dva do jednoho), bude konečný produkt konkurenceschopný, protože se domnívám, že jsou zde uvedené výrobky nadhodnocené a my bychom nejspíš byli schopni poskytovat podobné funkce za nižší cenu.

2 Popis problému

2.1 Server

Server by měl umět následující:

- komunikace s mobilním telefonem
- komunikace s web serverem
- komunikace s přípravkem
    - sériové rozhraní
    - mobilní telefon
- komunikace s databází
- zpracovat více „současně“ příchozích požadavků

2.2 Klientská aplikace

Klientská aplikace by měla splňovat:

- spustitelnost na více platformách (Microsoft Windows a Linux)
- možnost nastavit přípravek podle dat uložených v souboru
    - pomocí sériové linky
    - pomocí mobilního telefonu

3 Analýza

3.1 Server

Server jako program musí být spolehlivý a bezpečný, předpokládá se jeho relativně dlouhá doba běhu. Také by měl rozumně vytěžovat systém na kterém bude spuštěn, čímž je myšleno že nesmí obsahovat aktivní čekání a podobné konstrukce zbytečně pohlcující čas procesoru. Jeho nastavení by mělo být přehledné a snadno modifikovatelné a mělo by být umožněno toto nastavení otestovat – například poslat testovací e–mail nebo zkusit přistoupit do databáze.

Parametry

Parametry nutné pro spuštění serveru budou specifikovány v konfiguračním souboru. Bude tvořen posloupností výrazů (viz tabulka 3.1) a bude zpracován rekurzivním sestupem (viz [3]). Data budou předávána ve struktuře **Settings**.

(1)	*Nastavení*	→	*Výraz* **konec_řádku** *Nastavení*
(2)	*Nastavení*	→	ϵ
(3)	*Výraz*	→	*Atribut* **hodnota**
(4)	*Výraz*	→	*Atribut2 Adresa*
(5)	*Výraz*	→	% komentář
(6)	*Výraz*	→	ϵ
(7)	*Atribut*	→	**PORT**
(8)	*Atribut*	→	**COMPORT**
(9)	*Atribut*	→	**MAXCLIENTS**
(10)	*Atribut*	→	**SMSC**
(11)	*Atribut*	→	**WEBTIMEOUT**
(12)	*Atribut*	→	**SERIALTIMEOUT**
(13)	*Atribut*	→	**XPROCTIMEOUT**
(14)	*Atribut*	→	**SMTP_SERVER**
(15)	*Atribut*	→	**SMTP_SERVER_PORT**
(16)	*Atribut*	→	**SMTP_SENDER**
(17)	*Atribut*	→	**MYSQL_SERVER**
(18)	*Atribut*	→	**MYSQL_SERVER_PORT**
(19)	*Atribut*	→	**MYSQL_USER**
(20)	*Atribut*	→	**MYSQL_PASS**
(21)	*Atribut*	→	**MYSQL_DB**
(22)	*Atribut2*	→	**ALLOWHOST**
(23)	*Adresa*	→	*Číslo.Číslo.Číslo.Číslo*
(24)	*Číslo*	→	**0 – 255**

Tabulka 3.1: Gramatika konfiguračního souboru

Komunikace

Jelikož je server o zprostředkování komunikace, bude třeba navrhnout schémata komunikací – *protokoly* mezi všemi prvky komunikující se serverem, ale i meziprocesovou komunikaci uvnitř serveru.

Procesy

Program se bude skládat z jednoho hlavního procesu (rodiče) a více potomků. Hlavní proces bude řídit vznik potomků a zároveň obsluhovat sériovou linku. Pro tento účel bude vytvořena speciální třída, aby ji bylo možno využít v klientské aplikaci.

Zabezpečení

Server bude muset nějakým způsobem určovat, kdo se k němu bude moci připojit a kdo ne. Tento problém bude řešit parametr **ALLOWHOST**, který reprezentuje IP adresu webserveru, ze kterého se bude možno se serverem komunikovat.

Dále by bylo vhodné přenos zašifrovat. Komunikace by mohla probíhat přes internet a bylo by tak možno převzít řízení přípravku, popřípadě z něj získat informace. Obě dvě věci jsou závažné porušení bezpečnosti, protože by se například dal alarm vypnout před plánovanou krádeží nebo by se dala zjišťovat pozice zákazníka (v případě přípravku pro zabezpečení automobilu). Navíc by se dal server (při znalosti protokolu) zneužít také jako SMS brána.

3.2 Klientská aplikace

Klientská aplikace bude sloužit koncovým uživatelům k nastavování přípravku. Proto by měla mít co nejjednodušší ovládání, ale zároveň by měla umožňovat nastavit všechny funkce přípravku. Také je třeba přesně specifikovat, jak bude možno s přípravkem komunikovat. Zatím se promýšlejí tyto možnosti:

- přímé připojení přes zvláštní sériovou linku (přímý přístup do eeprom)
- přímé přípojení přes sériovou linku (posílání příkazů pomocí protokolu PC–AVR)
- pomocí mobilního telefonu

Také se bude třeba zabývat synchronizací mezi databází a přípravkem. Jedná se o případ, když uživatel, který má přístup k nastavování pomocí webové aplikace, změní nastavení pomocí klientské aplikace například u sebe doma. Vznikne tím nekonzistence uloženého nastavení a stavu přípravku.

4 Návrh řešení

4.1 Server

Server obsahovat základní „nekonečnou“ smyčku. Bohužel je nutné aktivně zjišťovat přítomnost události na sériové lince (příchozí SMS zpráva), takže server bude svým způsobem aktivně čekat, ale perioda této akce bude zvolena tak, aby co nejméně zatěžovala systém a nezabrala příliš mnoho času – zpomaluje reakční dobu serveru.

Navržený průběh programu

1. Načtení nastavení
   - port, maximální počet klientů, hodnoty timeoutů, povolené adresy, ...
2. Čekání na připojení prohlížeče (s timeoutem)
   - Pokud se objeví příchozí konekce, je vytvořen proces, který ji přijme a obsluhuje
   - Komunikace probíhá pomocí socketů (viz [1])
3. Čekání na zprávy od potomků (pokud existují)
   - Případná zpráva je opatřena příznakem procesu a uložena do fronty
4. Kontrola fronty
   - Pokud je ve frontě zpráva, odešle se na sériovou linku
5. Kontrola sériového rozhraní
   - Zkontroluje se obsah SMS zpráv uložených na SIM kartě
   - Pokud je nějaká nalezena, je ihned zpracována a odeslána potomkovi

Činnost obslužného procesu

Potomek vznikne přijetím konekce iniciované web serverem. Jeho práce spočívá v přijmutí zprávy, odeslání ji hlavnímu procesu, přijmutí odpovědi a její odeslání. Je možné, že bude třeba jeho činnost zesložitit o důkladnější model komunikace díky „rychlosti“ přijmutí odpovědi a jejímu zpracování. Tuto věc bude ale možno určit až testováním, protože během psaní serveru není k dispozici žádný funkční model přípravku.

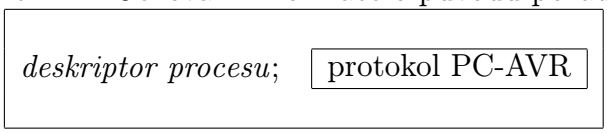
Návrh komunikace

Mezi procesy

Komunikace mezi procesy funguje podobně jako vlastní komunikace mezi prohlížečem a serverem – na bázi socketů. Tuto možnost jsem si vybral kvůli její průhledné implementaci, navíc jsou volání `send()` a `recv()` blokující, tj. není třeba řešit synchronizaci například pomocí semaforů a nenastává aktivní čekání. Také lze u funkce `recv()` jednoduše nastavit *timeout* pomocí funkce `select()`.

Předávat se bude řetězec, obsahující předzpracovaná data přijatá od prohlížeče. Hlavní proces z něj vytvoří strukturu `sms_struct`, kterou posléze uloží do fronty na odeslání. Pro možnost odpovědi musí být ovšem zachována informace z kterého procesu přišla žádost. Dá se to vyřešit zabalením celé informace do dalšího bloku (viz obrázek 4.1). Je to ale jistá komplikace pro návrháře přípravku.

Obrázek 4.1: Uchování informace o původu požadavku



Sériová linka

Výměna informací mezi serverem a mobilním telefonem probíhá výhradně pomocí AT příkazů (viz [2]). Třída `AT_com` implementuje příkazy potřebné pro tuto aplikaci a funkce pro konverzi (PDU[1] ↔ ASCII).

Webová aplikace

Protokol, pomocí kterého se tento dialog odehrává, byl navržen kolegou, který vytváří webovou aplikaci. Uvedu zde základní příkazy, je možné, že se jejich počet časem rozroste o další synchronizační řetězce (díky timeoutům).

příkaz	parametry
`send_sms`	(*tel. číslo*, "*text zprávy*", *příznak čekání odpovědi*)
`ack`	(*počet událostí ve frontě*)
`quit`	
`delivered_sms`	("*text zprávy*")

Tabulka 4.1: Příkazy pro komunikaci s webovou aplikací

[1] Formát používaný v AT příkazech pro manipulaci s textovými zprávami. Viz kapitola Implementace (strana 15)

→	zpráva pro server
←	zpráva pro webovou aplikaci
→	`send_sms`(420123456789, "PC ABC1234 ONOF1 INPU040B", w)
←	`ack`(3)
←	`delivered_sms`("PC OK")
→	`quit`()

Tabulka 4.2: Příklad komunikace serveru s webovou aplikací

4.2 Klientská aplikace

Aplikace má být podle zadání spustitelná v operačním systému Linux i Microsoft Windows, je proto třeba vybrat knihovnu pro grafické uživatelské rozhraní, která existuje v „mutacích“ pro oba tyto operační systémy. Vybral jsem si knihovnu Qt, protože je to jediné řešení tohoto problému, které znám. Navíc jsem se s ní už setkal, takže znám základní principy jejího použití.

Dále bude třeba naimplementovat grafické uživatelské rozhraní a s tím související prostředky (třídy) v knihovně Qt. Nepředpokládá se žádná „speciální“ funkce, takže bych si měl vystačit se standardními knihovnami, reprezentující formuláře a jejich události. S největší pravděpodobností bude vhodné prostředí se záložkami, podobné jako se používá při nastavování vlastností například operačního systému Microsoft Windows (vzhledem k tomu, že se také jedná o nastavení). Také je přehlednější a nevyžaduje takovou plochu, jako kdyby byla všechna nastavení vidět najednou. Takové řešení by navíc ani nebylo možné, protože se předpokládá nastavování velkého počtu reakcí (padesát a více).

5 Realizace

5.1 Server

Programovací jazyk

Už z návrhu je patrno, že bude třeba vybrat jazyk, který umí efektivně tvořit třídy, umožňuje přistupovat k sériové lince a zároveň komunikovat pomocí protokolu TCP s webovou aplikací, MySQL databází a SMTP serverem. Také by bylo ideální, aby program běžel na normálním stroji, tzn. neměl přehnané nároky na hardware. Rozhodoval jsem se mezi Javou a C++ a nakonec jsem si na tento úkol vybral C++, doplněný o knihovnu MySQL++. Javu jsem si nezvolil díky jejím nárokům na hardware a s tím související „nepružností“.

Knihovna MySQL++

MySQL++ je knihovna umožňující komunikovat s MySQL databází v prostředí jazyka C++ a je postavena na principech *Standard Template Library*. Výsledky dotazů vrací ve strukturách podobných STL kontainerům a automaticky převádí datové typy MySQL na jejich protějšky z C++.

Základní použití zprostředkovávají třídy `mysqlpp::Connection`, `mysqlpp::Query` a `mysqlpp::Result`. Způsob použití, veřejné metody a ostatní atributy naleznete v uživatelském manuálu [4].

Implementace

Na začátku jsem se snažil udělat co nejfunkčnější model, který bude pro každou novou příchozí konekci vytvářet obslužný proces (potomka) a čekat na další konekci. To se podařilo celkem bez problémů. Obtížnější bylo navrhnout synchronizaci mezi takto vytvořenými procesy a jejich rodičem. Tento problém jsem vyřešil zvolením způsobu komunikace – sockety. Je to pro mě výhodné v několika směrech. Implementace je snadnější než například sdílená paměť (není třeba řešit synchronizaci přístupu k ní), ale hlavně mohu takto vytvořený socket použít jako parametr funkce `select()`. Tímto si ušetřím dvojí ověřování (například u potomka to je čekání na zprávu od rodiče a zároveň od webového prohlížeče) tím, že je začlením do množiny parametrů jednoho zavolání již zmiňované funkce `select()`.

Další části programu jsem si rozdělil do tříd – aby se s nimi dalo později lépe pracovat, zpřehlednil se kód, popřípadě aby se dali znovu požít.

Rozdělení do tříd:

- AT_com
- Serial
- Seznam < T >

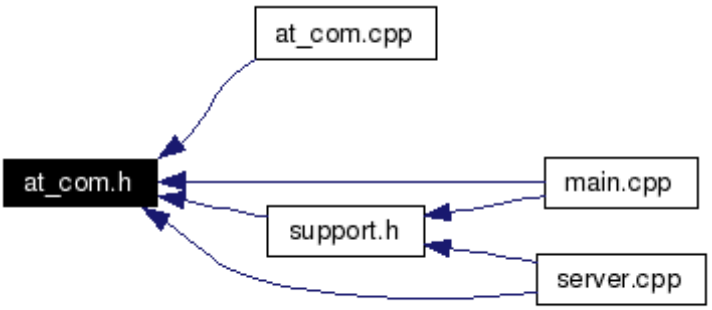
- MsgQueve
- SettingsParser
- WebParser

Také jsem vytvořil pár funkcí, které jsou pro chod server důležité, ale nepřišlo mi vhodné pro ně zakládat třídu (ať pro každou zvlášť nebo jednu pro všechny), tak jsem je umístil do **namespace** `support`. Jsou to funkce obsluhující databázi, SMTP server a také funkce `Tokenize()`, sloužící ke zpracování příchozí zprávy poslané webovou aplikací.

V další části této kapitoly se budu věnovat jednotlivým třídám a jejich hlavním metodám, jejich požitím v programu a implementací.

Třída AT_com

Hlavičkový soubor této třídy se jmenuje **at_com.h** a její metody jsou v souboru **at_com.cpp**.



Obrázek 5.1: Soubory, která přímo nebo nepřímo vkládají **at_com.h**

Hlavní funkce této třídy spočívá ve vytváření AT příkazů, pomocí kterých se potom komunikuje s mobilním telefonem. V jejím hlavičkovém souboru je také definována struktura `sms_struct`, kterou požívá server jako interní reprezentaci SMS zprávy a kterou také můžete najít jako argument některých metod třídy `AT_com` nebo jak návratovou hodnotu.

Veřejné metody

- **AT_com** (**Serial** *_s)

  Konstruktor

  Jediný parametr konstruktoru je ukazatel na třídu `Serial`, kterou pomocí tohoto ukazatele bude moci třída `AT_com` využívat.

- **sms_struct** * **getSMS** (int id)

Metoda sloužící k získání SMS zprávy

Tato metoda umožňuje získat SMS zprávu z mobilního telefonu z dané pozice na sim kartě. Nejdříve sestaví AT příkaz, potom ho pomocí instance třídy `Serial` pošle na sériovou linku mobilního telefonu, přečte odpověď, dekóduje PDU[1] formát a vytvoří instanci struktury `sms_struct`. Ukazatel na ní je návratová hodnota metody.

- void **sendSMS** (**sms_struct** ∗sms)

  Metoda umožnující posílat SMS zprávy

  Tato metoda umí poslat SMS zprávu předanou v ukazateli na strukturu `sms_struct`. Nejdříve převede text do PDU formátu, poté sestaví zbytek AT příkazu a pošle ho na sériovou linku. Bohužel mobilní telefon, na kterém tuto třídu testuji, nepotvrzuje odeslání SMS zprávy, takže metoda nic nevrací a program nedostane informaci o tom, zda byla zpráva odeslána.

- void **deleteSMS** (int id)

  Metoda k mazání SMS zpráv

  Metoda slouží ke smazání přečtené SMS zprávy na určené pozici na sim kartě.

- void **setPDU** ()

  Metoda k nastavení formátu odpovědi od telefonu

  Tato metoda byla přidána do třídy `AT_com` zatím jako poslední, protože se našel telefon, který nepodporuje odesílání SMS zpráv v PDU formátu. Metoda se ho pokusí nastavit a pokud neuspěje, nebudou se další metody „snažit“ dělat cokoli dalšího.

Formát SMS zprávy v AT příkazech a PDU formát

pozice význam

0 – 1 Délka čísla SMS centra – Tento údaj je uveden v počtu oktetů. To znamená, že bude v u nás běžně používaných sítí mít hodnotu 07, protože čísla SMS center mají stejný formát jako běžná čísla (například u společnosti Vodafone je to 420608005681)

2 – 3 Typ SMS centra

4 – 16 SMS centrum – Data jsou uložena po bajtech systémem little-endian. Znamená to, že je třeba je při čtení prohodit.

18 – 19 Délka čísla odesílatele (předpokládáme opět standardní délku, ale v programu je ošetřen i případ, kdy tomu tak není – například reklamní SMS zprávy od operátorů mají často speciální čísla odesílatele)

20 – 23 Typ adresy odesílatelova čísla

[1] Podrobnosti o tomto formátu viz strana 15

24 – 35 Číslo odesílatele (uloženo podobně jako SMS centrum)

36 – 39 Tyto bajty jsou rezervovány pro identifikaci protokolu a datového kódovacího schématu. Nenašel jsem v dokumentaci jejich přesný význam, ani možné hodnoty, tak tyto hodnoty nijak nezpracovávám

40 – 53 Časové údaje – v těchto 14 bajtech je obsažena úplná informace o času přijetí SMS zprávy

54 – 55 Délka textu

56 – konec Samotný text zprávy zakódovaný v PDU formátu

PDU formát

PDU = Protocol Data Unit

Text zprávy je kódován pro ušetření místa a nejspíše i pro zrychlení přenosu pomocí sedmibitových znaků. Telefon ale komunikuje pomocí standardních osmibitových znaků, takže je třeba se s tímto nějak vypořádat. Znaky by se mohli přenášet například s nulovým nejvyšším bitem a bylo by po starostech, ale tvůrci PDU formátu si usmysleli, že využijí přenosovou kapacitu naplno. To udělali tak, že do prvního osmibitového znaku přidali jeden byt z druhého. V dalším znaku jsou už 2 bity následujícího znaku. Tímto způsobem kódování pokračuje, až se dostane zase na „začátek“. Tím je myšleno, že se tato struktura opakuje s periodou 8 znaků.

Algoritmus na převod *ASCII* na PDU naleznete v privátních metodách `pdu_to_ascii()` a `ascii_to_pdu()`.

Další vývoj

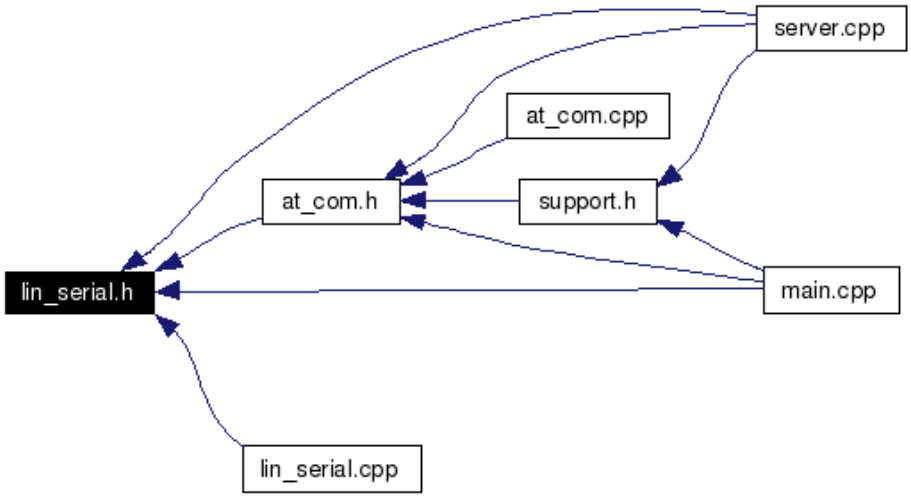
Myslím si, že dále by se měl vývoj této součásti soustředit na zobecnění – podporu všech typů telefonních přístrojů (například ty, které neumí zpracovat PDU formát), efektivnější zpracování přijatého řetězce a možná i změna struktury tak, aby se nemusela konstruktoru předávat instance třídy `Serial`.

Třída Serial

Hlavičkový soubor této třídy se jmenuje **lin_serial.h** a její metody jsou v souboru **lin_serial.cpp**.

Název souboru se liší od jména třídy o prefix – „lin“ zde znamená operační systém, pro který je tato třída napsána. Předpokládám, že v multiplatformní aplikaci bude muset být podobná třída pro operační systém Microsoft Windows.

Třída Serial zprostředkovává komunikaci mezi programem a sériovým rozhraním. Existuje hlavě proto, aby se dala komunikace se sériovou linkou snadno změnit úpravou této třídy.



Obrázek 5.2: Soubory, která přímo nebo nepřímo vkládají **lin_serial.h**

Veřejné metody

- void **Init** (string device)

  Inicializace

  Tato metoda otevře dané zařízení – soubor, jehož jméno je předáno jako parametr. V operačním systému Linux jsou zařízení součástí systému souborů. `COM1`, jak jej známe z operačního systému Windows zde odpovídá souboru `/dev/ttyS0`. Zároveň nastaví přenosovou rychlost a ostatní parametry sériového rozhraní.

- int **Write** (string output)

  Zápis na otevřené zařízení

  Pomocí standardní funkce `write()` zapíše řetězec na sériové rozhraní.

- string **Read** (int length)

  Čtení z otevřeného zařízení

  Pomocí standardní funkce `read()` přečte řetězec ze sériového rozhraní. Pokud na vstupu není připraven žádný znak, vrací metoda řetězec `"NULL"`, jinak vrací načtenou hodnotu.

- void **Close** ()

  Konec práce se zařízením

  Uzavře komunikaci se sérovým portem.

Obrázek 5.3: Soubory, která přímo nebo nepřímo vkládají **seznam.cpp**

Šablona Seznam < T >

Šablona je celá umístěna v souboru **seznam.cpp**.

Tato šablona reprezentuje standardní obecný kontainer. Implementoval jsem ji v rámci jiného předmětu a i v tomto projektu našla uplatnění.

Veřejné metody

- **Seznam** ()

  Konstruktor

- void **begin** ()

  Nastaví vnitřní ukazatel na první prvek

- void **operator++** ()

  Posune vnitřní ukazatel o jeden element

- bool **end** ()

  Metoda zjišťující, zda vnitřní ukazatel dosáhl konce seznamu

- void **add** (T val)

  Přidání prvku do seznamu

- T **get** ()

  Vrací hodnotu z elementu určeného vnitřním ukazatelem

- void **del** (T val)

  Mazání prvku s danou hodnotou

- **~Seznam** ()

  Destruktor

- **Seznam** (const **Seznam**< T > &s)

  Kopírovací konstruktor

- **Seznam**< T > & **operator=** (const **Seznam**< T > &s)

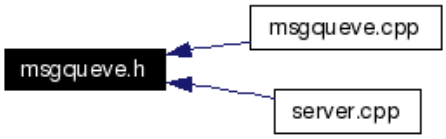
  Operátor =

Další vývoj

Je možné, že tato třída úplně zmizí a nahradí ji nějaký jiný kontainer z knihovny STL. Pokud by zůstala, nejspíše by se hodilo naimplementovat třídu, která bude umět procházet prvky této šablony (externí iterátor), ale pro účely tohoto projektu zatím vyhovuje i bez něj.

Třída MsgQueve

Hlavičkový soubor této třídy se jmenuje **msgqueve.h** a její metody jsou v souboru **msgqueve.cpp**.



Obrázek 5.4: Soubory, která přímo nebo nepřímo vkládají **msgqueve.h**

Třída `MsgQueve` je jedna z hlavních součástí celého serveru. Zjednodušeně se dá říct, že veškerá komunikace prochází právě skrz tuto třídu. Pokud potomek (připomínám, že je tím myšlen podřízený proces) pošle zprávu serveru, ve které je požadavek na zaslání SMS, server pouze zavolá metodu `toSerial()` a tím ji uloží k pozdějšímu zpracování. Další popis této třídy můžete naleznout v kapitole *Návrh řešení*, popřípadě v popisu jednotlivých metod.

K vnitřní reprezentaci zpráv je použita privátní struktura `Elem`, jejíž deklaraci naleznete v souboru **msgqueve.h**. Zprávy jsou uspořádány v jednosměrném spojovém seznamu. Každá zpráva má v sobě uložen identifikátor procesu, kterým byla vytvořena, aby na ni bylo možno zaslat odpověď.

Veřejné metody

- **MsgQueve** ()

  Konstruktor

  Konstruktor vytvoří prázdnou frontu.

- ∼**MsgQueve** ()

  Destruktor

- void **toSerial** (int fd, string phone_num, string text, bool wait)

  Metoda přidávající zprávu určenou pro mobilní telefon

  Tato metoda vytvoří novou instanci struktury `Elem` a nastaví odpovídající hodnoty jejích parametrů (v tomto případě to bude `toSerial` = true). Poté ji zařadí na začátek seznamu. O dalším zpracování ještě významně rozhoduje parametr `wait`, který určuje, zda bude očekávána odpověď.

- void **toWeb** (string phone_num, string text)

  Metoda přidávající zprávu určenou pro webový prohlížeč

  Pokud přijde serveru nějaká SMS zpráva (a *není* to reakce), zavolá se tato metoda. Podle telefonního čísla najde odpovídající instanci struktury `Elem`, změní její příznaky a text. Implementace se poněkud liší od návrhu (strana 11), protože jsem se rozhodl nekomplikovat práci návrhářům přípravků s tím, že budu identifikovat přípravek pouze podle telefonního čísla odesílatele. Mohlo by to způsobit problém, pokud by přípravek během zpracovávání příkazu odeslal nějakou SMS jako reakci na nastavenou podmínku a až potom odpověď na příkaz. Tento případ je nyní ošetřen programově tak, že žádná reakce nepřijde jako zpráva do této třídy.

- void **rmElem** (int fd, bool force=false)

  Metoda pro mazání elementů

  Metoda maže Element podle zadaného identifikátoru potomka. Parametr `force` přibyl kvůli možnosti posílat zprávy bez odpovědi (ušetření nákladů na posílání příkazů). Jde o to, že pokud webová aplikace nevyžaduje odpověď, potomek zanikne hned při poslání zprávy rodičovi. Při této události se také smažou všechny odpovídající položky ve spojovém seznamu, mimo těch, které nečekají na odpověď. S nastaveným parametrem `force` se mažou právě takovéto elementy.

- bool **containsFD** (int fd)

  Metoda pro zjištění přítomnosti elementu s daným deskriptorem

  Metoda projde spojový seznam a vrací informaci o nalezení instance struktury `Elem` s daným identifikátorem procesu.

- bool **containsNum** (string num)

  Metoda pro zjištění přítomnosti elementu s daným telefonním číslem

  Metoda projde spojový seznam a vrací informaci o nalezení instance struktury `Elem` s daným telefonním číslem.

- **sms_struct** $*$ **getSerial** (string smsc)

  Metoda pro získání SMS zprávy ve formátu vhodném pro odeslání

  Pokud existuje element s nastaveným příznakem `toSerial`, tato metoda z ní vytvoří instanci struktury `sms_struct` a vrátí ukazatel na ni. Parametr je zde jen proto, aby se do vracené struktury nemuselo přidávat číslo SMS centra, metoda ho rovnou nastaví.

- string **getWeb** (int fd)

  Metoda pro získání zprávy pro web

  Pokud existuje element s nastaveným příznakem `toWeb` a odpovídajícím deskriptorem procesu, vrátí tato metoda řetězec se zprávou a přenastaví parametry.

- int **getLen** (int fd)

  Metoda vracející počet zpráv čekajících na odeslání přes sériovou linku

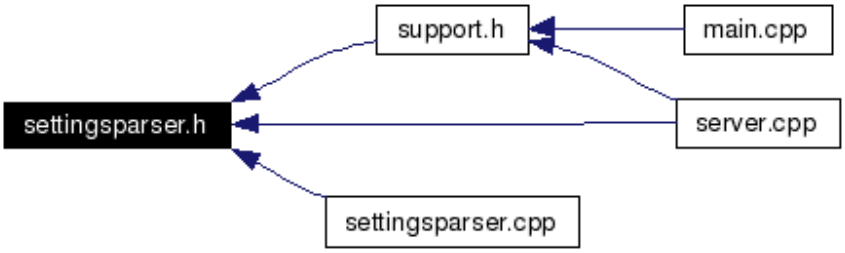
  Pokud fronta přijme zprávu, vrátí webové aplikaci počet zpráv čekající před ní. To právě zjišťuje tato metoda. Většinou je tato hodnota rovna nule, ale při větším zatížení je možné, že nabude na významu.

Další vývoj

Třída by se dala vylepšit efektivnějším a průhlednějším zpracováním „jednosměrných“ zpráv. Také by mohla přibýt podpora pro zprávy určené MySQL databázi, které nebyly v době návrhu uvažovány.

Třída SettingsParser

Hlavičkový soubor této třídy se jmenuje **settingsparser.h** a její metody jsou v souboru **settingsparser.cpp**.



Obrázek 5.5: Soubory, která přímo nebo nepřímo vkládají **settingsparser.h**

Třída slouží k načtení nastavení ze souboru, je-li programu při startu zadán parametr s jeho jménem. Pokud není, třída nastaví atributy na „defaultní“ hodnoty, které ovšem s největší pravděpodobností nebudou vyhovovat. Dalo by se to opravit tak, že se program bez parametru vůbec nespustí, nebo se začne na parametry ptát. Zatím toto vyřešeno není, ale počítá se s řešením v dalším vývoji.

Veřejné metody

- **SettingsParser** ()

  Konstruktor

  Nastavuje počáteční hodnoty atributů.

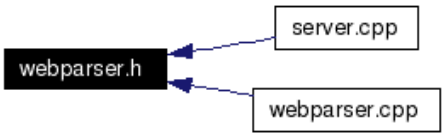
- **Settings SettingsFile** (string filename)

Hlavní metoda třídy

Využití třídy zprostředkovává právě tato metoda. Jako parametr dostane jméno souboru s nastavením a vrací strukturu `Settings`. Třída pomocí svých privátních metod analyzuje vstupní soubor pomocí rekurzivního sestupu podle gramatiky uvedené v kapitole Analýza (strana 8). Podrobnosti ohledně postupu naleznete ve skriptu k předmětu Programovací jazyky a překladače (viz [3]).

Třída WebParser

Hlavičkový soubor této třídy se jmenuje **webparser.h** a její metody jsou v souboru **webparser.cpp**.



Obrázek 5.6: Soubory, která přímo nebo nepřímo vkládají **webparser.h**

Tato třída slouží k překladu příkazů od webové aplikace do formy vhodné ke zpracování serveru. Její výstup není struktura, ale řetězec, ze kterého se struktura snadno vytvoří pomocí funkce `Tokenize`, která je implementovaná v namespace `support`. Výstupní řetězec má parametry oddělené tabulátorem (vzhledem k tomu, že v příkazech od webové aplikace se znak tabulátor nevyskytuje, je jeho použití vhodné). Potomek tento řetězec předá rodiči a ten si ji pomocí již zmíněné funkce převede na strukturu a dále ji zpracuje.

Veřejné metody

- **WebParser** ()

  Konstruktor

  Nastavuje počáteční hodnoty atributů.

- string **Command** (string input)

  Metoda pro překlad přijatého příkazu

  Podobně jako ve třídě `SettingsParser` je tato metoda jediný prostředek k využití celé třídy. Rozdílné jsou parametry a gramatika. Jako vstup dostane řetězec reprezentovaný standardní třídou `string` a vrací přeložený řetězec. Parametr je zpracován stejně jako u předchozí třídy pomocí rekurzivního sestupu.

5.2 Klientská aplikace

Programovací jazyk

Klientská aplikace bude napsána stejně jako serverová část v jazyce C++, aby bylo možno znovu použít některé již napsané části kódu, například komunikace po sériové lince a komunikace s mobilním telefonem. Vzhledem k tomu, že je třeba udělat grafické uživatelské rozhraní a je nutnost zajistit jeho kompatibilitu s různými operačními systémy, bude klientská aplikace využívat knihovnu Qt od společnosti Trolltech. Knihovna je šířena pod licencí GNU GPL, v případě komerčního použití je možnost ji zakoupit a produkt s ní vytvořený prodávat.

Aplikace ale bude muset být podle zadání multiplatformní. Součásti knihovny Qt by měli jít bez problému přeložit na různých operačních systémech, neplatí to ale bohužel se všemi částmi. Komunikaci po sériové lince bude nejspíš třeba napsat nejméně ve dvou provedeních – verze pro Windows a verze pro Linuxové operační systémy.

Knihovna Qt

Knihovna Qt je nadstavba jazyka C++. Není to jen soubor tříd, spíš velmi komplexní prostředí. Dalo by se říci, že je to „jazyk v jazyce“, i když to také není zcela přesné. Zachovává všechny vlastnosti C++ a přidává k nim další konstrukce a rozšiřuje syntax, čímž ztrácí kompatibilitu se standardem a tento kód nelze přímo kompilovat obyčejným překladačem. Zdrojový kód se musí nejdříve zpracovat programem *Qt Meta Object Compiler* (zkráceně „moc“). Poté se již kód přeloží pomocí překladače g++.

Knihovna implementuje spoustu různých tříd, například je v ní napsána „nová“ reprezentace řetězce, třída `QString`. Umožňuje také komunikovat s MySQL databází, má třídu pro parsování XML dokumentů, ale hlavně, umožňuje vytvořit grafické uživatelské rozhraní. Základní prvek pro jeho tvorbu je třída `QWidget`, kterou dědí každá „zobrazitelná“ třída. Obsahuje základní metody pro práci s grafickými objekty, jako například nastavení velikosti a viditelnosti.

Struktura C++ třídy

```
class A {
public:
        A();
        int  value() const { return val; }
        void setValue( int );
private:
        int  val;
};
```

Struktura Qt třídy

```
class A : public QObject {
        Q_OBJECT
public:
        A();
        int  value() const { return val; }
public slots:
        void setValue( int );
signals:
        void valueChanged( int );
private:
        int  val;
};
```

Veškeré třídy obsahující signály či sloty musí mít v deklaraci makro `Q_OBJECT`.

Signály a Sloty

Asi nejpodstatnější modifikace oproti C++ jsou právě signály a sloty.

Signály jsou vysílány objekty při nějaké události, například pří kliknutí na tlačítko. Mohou být vyvolány pouze třídami, které je mají a jsou vykonávány mimo „událostní smyčku“. To znamená, že jsou vykonány okamžitě. Pro praktické použití je třeba signál propojit se slotem, oba musejí mít stejné parametry.

Sloty reagují na vzniklé signály. Jsou to obyčejné C++ funkce, lze je volat i normálním způsobem (není třeba přes signál). Sloty se dělí na privátní (*private*), chráněné (*protected*) a veřejné (*public*), což ovlivňuje možnost jejich volání stejně jako standardní funkce. Signál může být deklarován i jako virtuální.

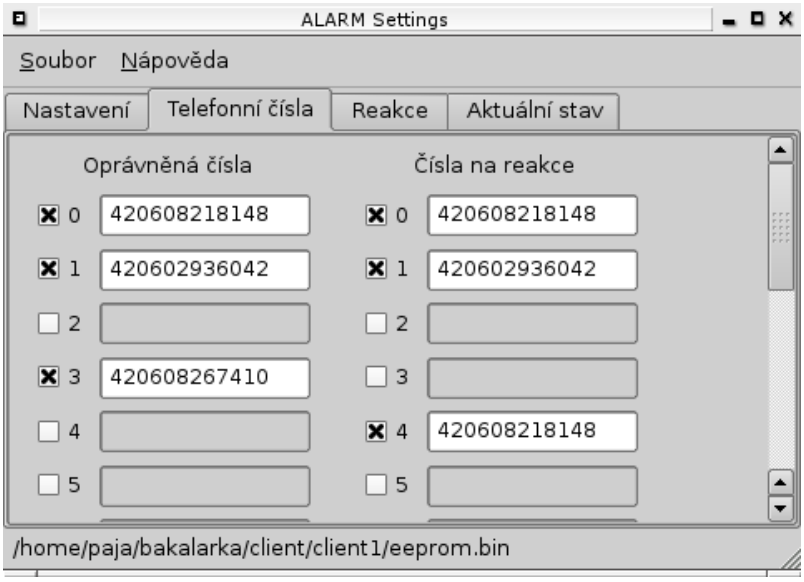
Implementace

Při implementaci jsem se zaměřil hlavně na grafické uživatelské rozhraní, protože jsem s ním (a s knihovnou Qt) neměl zatím moc zkušeností. Výsledkem je funkční a intuitivní prostředek k nastavování přípravků.

Všechny zde uvedené třídy dědí nějakou třídu z knihovny Qt. Tento postup je velmi výhodný a ve spolupráci s touto knihovnou asi nejpoužívanější, protože umožňuje upravit si třídy přesně k potřebám programátora. Jejich úprava většinou spočívá v přidání vhodných slotů a signálů, popřípadě konstruktorů a atributů u složitějších grafických oken (zde například TabWidget).

Rozdělení do tříd:

- EditReaction
- LineEditNum
- LineEditReact



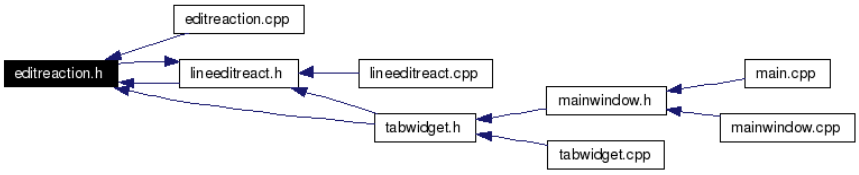
Obrázek 5.7: Nastavování povolených čísel

- MainWindow
- PushButtonReact
- TabWidget

V další části této kapitoly se budu věnovat jednotlivým třídám a jejich hlavním metodám, jejich požitím v programu a implementací.

Třída EditReaction

Hlavičkový soubor této třídy se jmenuje **editreaction.h** a její metody jsou v souboru **editreaction.cpp**.

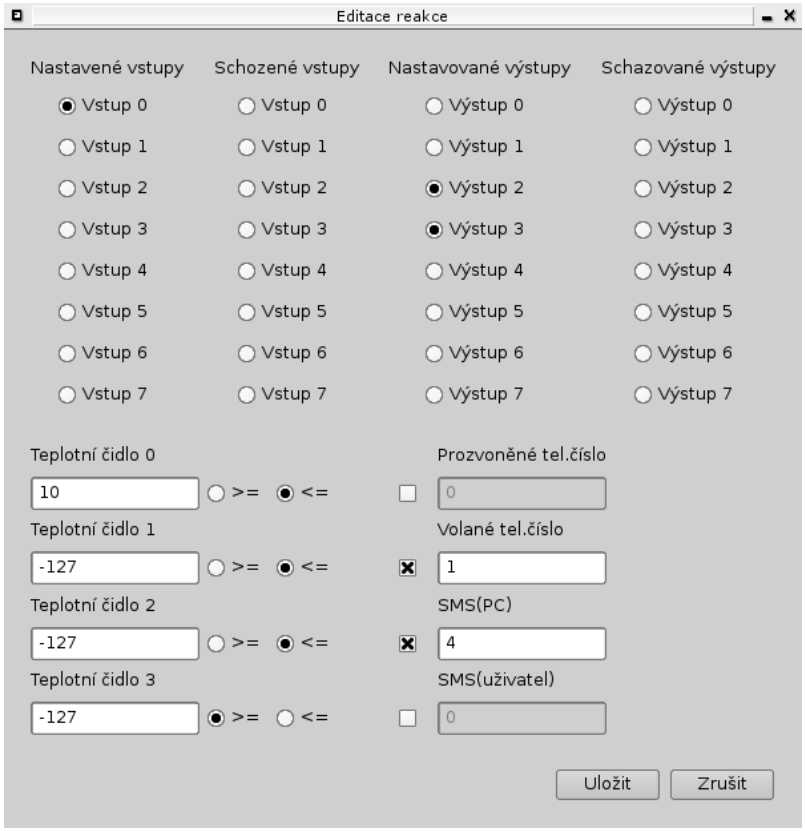


Obrázek 5.8: Soubory, která přímo nebo nepřímo vkládají **editreaction.h**

Třída EditReaction je potomkem třídy `QWidget`. Reprezentuje okno, ve kterém může uživatel vytvářet nebo modifikovat reakce přípravku.

Signály

- void **sendValue** (QString str)



Obrázek 5.9: Okno na editaci reakcí

Signál pro přenos hodnot

Tento signál je vyvolán pokud uživatel klikne na tlačítko Uložit a slouží k předání dat zpět do instance třídy `LineEditReact`.

Veřejné metody

- **EditReaction** (QString &str)

  Konstruktor

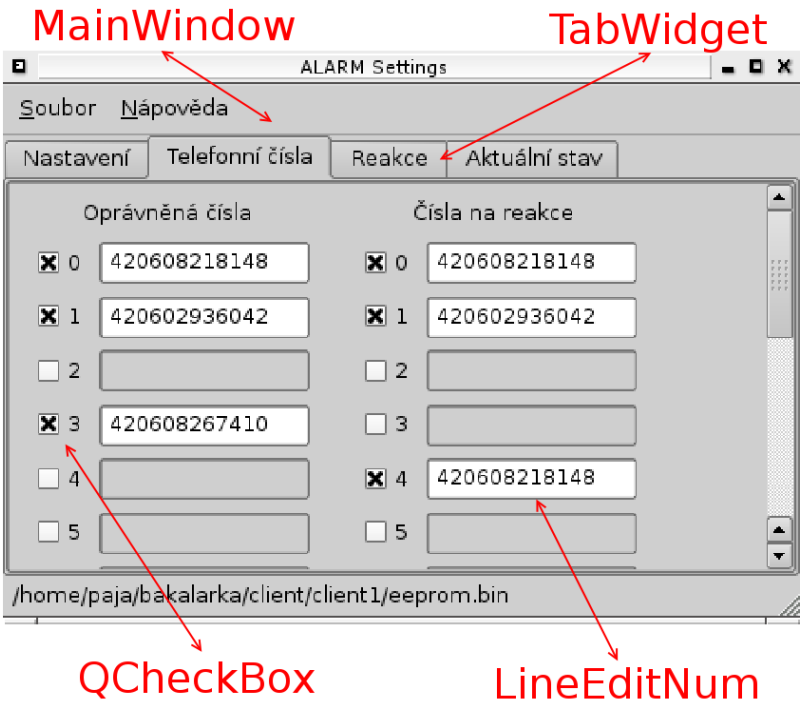
  Nastavuje počáteční hodnoty atributů a vytváří instance prvků a umisťuje je do layoutu, čímž vytvoří vzhled nového okna. Jako parametr slouží hodnota z třídy `LineEditReact`, podle které nastaví stavy vytvořených instancí. Pokud je parametr prázdný řetězec, je vyplněn defaultní hodnotou.

Privátní sloty

- void **OKclicked** (bool state)

  Slot pro tlačítko Uložit

  Po kliknutí na tlačítko Uložit zjistí data ze všech prvků a konvertuje je do reprezentace používané ve třídě `LineEditReact`. Poté vyvolá (emituje) signál sendValue a zavře okno.



Obrázek 5.10: Grafická reprezentace tříd

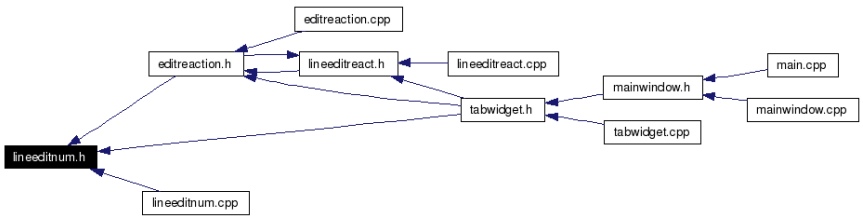
- void **CANCELclicked** (bool state)

  Slot pro tlačítko Zrušit

  Pokud je stisknuto tlačítko Zrušit, zavře se okno bez emitování signálu sendValue (tj. bez uložení změn).

Třída LineEditNum

Hlavičkový soubor této třídy se jmenuje **lineeditnum.h** a její metody jsou v souboru **lineeditnum.cpp**.



Obrázek 5.11: Soubory, která přímo nebo nepřímo vkládají **lineeditnum.h**

Tato třída dědí třídu `QLineEdit`, která reprezentuje vstupní formulář, do kterého lze zadat jednu řádku textu (číslice a znaky). Přesnější určení je možné pomocí definování

masky, popřípadě pomocí jiných mechanismů. Podrobnosti o této třídě naleznete v dokumentaci ke knihovně Qt.
Konkrétně je tato třída použita při zadávání povolených čísel a čísel na reakce.

Veřejné sloty

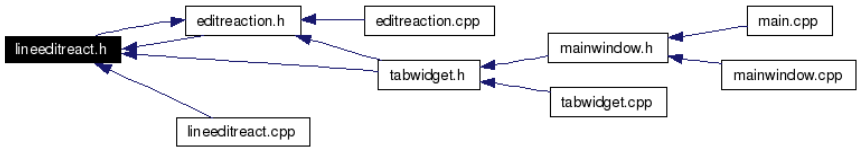
- void **setReadOnlyS** (int state)

  Veřejný slot k nastavení modifikovatelnosti textu

  Jediná modifikace od rodičovské třídy spočívá v přidání slotu, který se spojí se signálem `QCheckBox::stateChanged()`. Tato dvojice objektů potom reaguje na zaškrtnutí instance QCheckBox povolením zápisu textu do instance `LineEditNum`.

Třída LineEditReact

Hlavičkový soubor této třídy se jmenuje **lineeditreact.h** a její metody jsou v souboru **lineeditreact.cpp**.



Obrázek 5.12: Soubory, která přímo nebo nepřímo vkládají **lineeditreact.h**

Podobně jako předchozí třída, i tato dědí `QLineEdit`. Rozšiřuje ji ale o něco více než předchozí. Její obsah reprezentuje nastavení reakce. Vzhledem k tomu, že nemůžu chtít aby uživatel například převáděl hodnoty čidel do hexadecimálního kódu, bylo třeba vymyslet způsob úpravy obsahu této třídy. Nakonec jsem se rozhodl pro otevření dalšího okna (třída `EditReaction`).

Veřejné sloty

- void **setEnabledS** (int state)

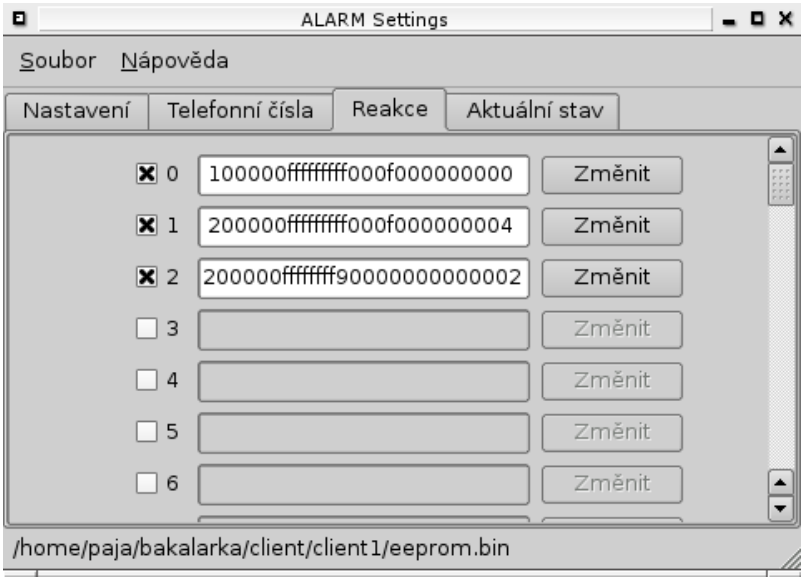
  Slot k nastavení použitelnosti

  Slot slouží k přepínání použitelnosti. V důsledku je to jen kosmetická změna, protože uživatel do tohoto okna stejně zapisovat nemůže. Slouží tedy pouze ke zvýšení přehlednosti.

- void **edit** ()

  Slot k editování obsahu

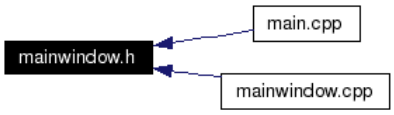
  Pokud chce uživatel editovat obsah, musí kliknout na tlačítko, které vyšle signál právě do tohoto slotu. Pokud toto nastane, vytvoří se instance třídy `EditReact`, předá se jí `LineEditReact::text()` a propojí se signál `EditReaction::sendValue()` se slotem `LineEditReact::setText()`.



Obrázek 5.13: Přehled reakcí

Třída MainWindow

Hlavičkový soubor této třídy se jmenuje **mainwindow.h** a její metody jsou v souboru **mainwindow.cpp**.



Obrázek 5.14: Soubory, která přímo nebo nepřímo vkládají **mainwindow.h**

Tato třída reprezentuje hlavní okno aplikace. Obstarává menu a jeho akce (otevírání a ukládání souboru).

Veřejné metody

- **MainWindow** ()

  Konstruktor

  Vytvoří vzhled vlastního, menu a vytvoří vazby mezi jeho signály. „Tělem“ okna je třída `TabWidget`, která je popsána dále.

Chráněné metody

- void **contextMenuEvent** (QContextMenuEvent ∗event)

  Vytváří kontextové menu

  Reaguje na kliknutí pravým tlačítkem myši. Vytvoří menu, zatím pouze s jednou položkou (ukončení aplikace).

Privátní sloty

- void **open** ()

  Otevření souboru

  Pomocí standardního dialogu otevře soubor s binární reprezentací paměti přípravku. Zkopíruje jeho obsah do paměti (viz privátní proměnné ve zdrojovém kódu). Tato paměť je dále zpracována a jsou podle ni nastaveny ostatní instance.

- void **save** ()

  Uložení souboru

  Pomocí standardního dialogu vybere jméno souboru a posléze do něj uloží obraz eeprom podle nastavených hodnot jednotlivých instancí formuláře.

- void **about** ()

  O aplikaci

  Zobrazí informace o aplikaci.

- void **aboutQt** ()

  O knihovně Qt

  Zobrazí informace o knihovně Qt.

Privátní metody

- void **createActions** ()

  Metoda pro vytvoření akcí

  Vytvoří akce a spojí jejich signály se s privátními sloty (open(), save(), ...).

- void **createMenus** ()

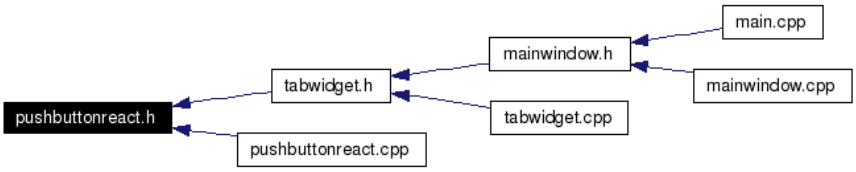
  Metoda pro vytvoření menu

  Pomocí akcí vytvoří strukturu menu. Akce jako kliknutí jsou v okamžiku vytváření už slinkovány (položka menu je vlastně akce vytvořená metodou createActions).

Třída PushButtonReact

Hlavičkový soubor této třídy se jmenuje **pushbuttonreact.h** a její metody jsou v souboru **pushbuttonreact.cpp**.

Třída reprezentuje zaškrtávací tlačítko použité u aktivování a deaktivování reakcí. Je potomkem třídy `QCheckButton`.



Obrázek 5.15: Soubory, která přímo nebo nepřímo vkládají **pushbuttonreact.h**

Veřejné sloty

- void **setEnabledS** (int state)

  Veřejný slot nastavující stav tlačítka

Veřejné metody

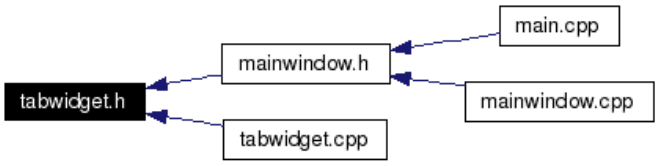
- **PushButtonReact** (const QString &text)

  Konstruktor

  Nastavuje popis tlačítka na hodnotu uvedenou v parametru. Jedná se o takzvaný *inline* konstruktor a jeho jediná činnost je, že předá parametr konstruktoru děděné třídy.

Třída TabWidget

Hlavičkový soubor této třídy se jmenuje **tabwidget.h** a její metody jsou v souboru **tabwidget.cpp**.



Obrázek 5.16: Soubory, která přímo nebo nepřímo vkládají **tabwidget.h**

Tělo třídy `MainWindow`. Reprezentuje prostředí se záložkami, tak jak ho známe třeba z nastavování různých parametrů v operačním systému Microsoft Windows, nebo například z webových prohlížečů vytvořených skupinou *Mozilla Foundation* (`mozilla.org`).

Veřejné metody

- **TabWidget** ()

Konstruktor

Vytvoří všechny instance formulářů a jejich prvků, umístí je do layoutů (použity jsou zde hlavně `QGridLayout` a `QScrollLayout`). Ukazatele na vytvořené instance uchovává v privátních atributech a proto je třeba spousta takzvaných „getrů“ a „setrů“. Jsou to metody, přes které se přistupuje k těmto atributům. Je to klasická konstrukce objektového programování, atributy jsou tak více chráněny (zápis pouze přes „setry“, které mohou kontrolovat správnost nastavovaných dat).

- void **setAllowedNum** (int index, bool on, QString number)

  Nastaví povolené číslo na indexu index *na hodnotu* number *a nastaví platnost podle parametru* on

- void **setReactionNum** (int index, bool on, QString number)

  Nastaví číslo pro reakce na indexu index *na hodnotu* number *a nastaví platnost podle parametru* on

- void **setReaction** (int index, bool on, QString cond)

  Nastaví reakci na indexu index *na hodnotu* cond *a nastaví platnost podle parametru* on

- void **appendConnectText** (QString str)

  Přidá text do informačního okna

- void **checkONOFF** (bool value)

  Nastaví zaškrtávací tlačítko Zapnuto na hodnotu value

- void **checkAck** (bool value)

  Nastaví zaškrtávací tlačítko Posílat potvrzení na hodnotu value

- void **checkVerNum** (bool value)

  Nastaví zaškrtávací tlačítko Verifikovat nastavovatele na hodnotu value

- void **setPassword** (QString str)

  Nastaví heslo na hodnotu str

- void **setPin** (QString **pin**)

  Nastaví pin na hodnotu str

- void **setOutputs** (QString out)

  Nastaví výstupy podle hodnoty hodnotu out

  Parametr je hexadecimální číslo reprezentované řetězcem. Metoda ho převede na binární kód a podle něj nastaví jednotlivé výstupy.

- bool **getONOFF** ()

  Zjistí hodnotu zaškrtávacího tlačítka Zapnuto

- bool **getAck** ()

  Zjistí hodnotu zaškrtávacího tlačítka Posílat potvrzení

- bool **getVerNum** ()

  Zjistí hodnotu zaškrtávacího tlačítka Verifikovat nastavovatele

- unsigned char **getOutputs** ()

  Zjistí hodnotu výstupů

  Výsledek je předán jako neznamínkový znak, jednotlivé bity odpovídají výstupům.

- QString **getPassword** ()

  Zjistí hodnotu hesla

- QString **getPin** ()

  Zjistí hodnotu pinu

- int **getLastReact** ()

  Zjistí poslední nastavenou reakci

- int **getLastReactionNum** ()

  Zjistí poslední nastavené číslo pro reakce

- int **getLastAllowedNum** ()

  Zjistí poslední nastavené povolené číslo

- void **getReact** (int pos, bool &state, QString &react)

  Nastaví parametry state *a* react *na hodnotu reakce z pozice určené parametrem* pos

- void **getAllowedNum** (int pos, bool &state, QString &num)

  Nastaví parametry state *a* num *na hodnotu povoleného čísla pozice určené parametrem* pos

- void **getReactionNum** (int pos, bool &state, QString &num)

  Nastaví parametry state *a* react *na hodnotu povoleného čísla pro reakce z pozice určené parametrem* pos

Poznámka

Možná jste postřehli, že klientská aplikace nesplňuje všechny body ze specifikace. Z časových důvodů se nepodařila naimplementovat komunikace s přípravkem. Bylo by snadné přidat komunikaci s mobilním telefonem a posílat změny přípravku, ale zatím není vyřešen postup zachování integrity nastavení přípravku a údaji v databázi, takže jsem tuto část raději odsunul do dalšího vývoje.

Díky tomu jsem také přesunul uživatelskou příručku pouze na přiložené médium, protože bude nejspíš v době obhajoby této práce aplikace rozsáhlejší.

Další vývoj

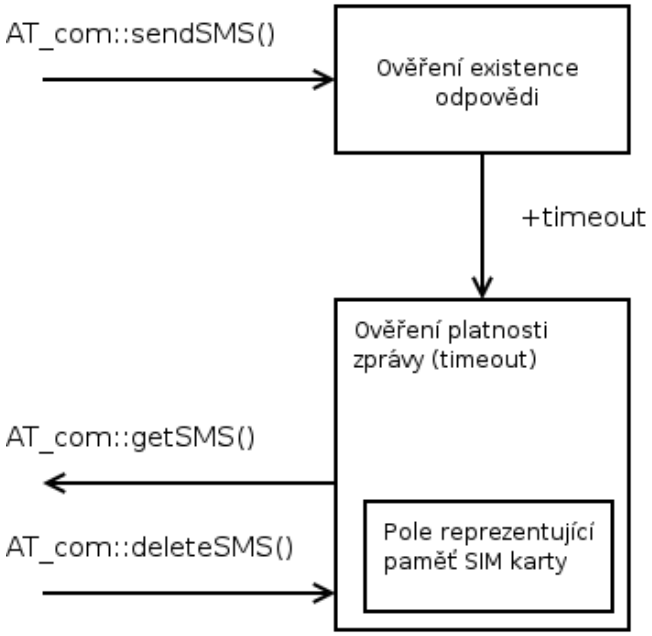
Další vývoj se zaměří na vývoj protokolu pro komunikaci s přípravkem. Jedná se o formát SMS zpráv (bude se jednat o modifikaci protokolu PC–AVR) upravený tak, aby se dalo zjistit, která data je třeba přeposlat do databáze. Další způsob komunikace by měl být přímo s přípravkem. Jedná se o možnost přečíst jeho paměť a uložit ji do souboru a poslat takový soubor zpět do přípravku.

Také by se mělo vylepšit uživatelské rozhraní – v nynější implementaci lze nalézt různé nedostatky, například deaktivované teplotní čidlo je rozlišeno pouze hodnotou (menší než –45°).

6 Testování

Při psaní jednotlivých tříd jsem se je snažil zároveň testovat – ne všude se to ale povedlo v uspokojivé míře. Například komunikace s webovým serverem se testovala relativně špatně, protože tato část existovala v době, kdy ještě nebyl k dispozici žádný, ani částečně funkční přípravek. Vzhledem k tomu, že server pouze zprostředkovává komunikaci, to byl docela problém. Proto jsem se rozhodl napsat simulátor.

6.1 Simulátor přípravku



Obrázek 6.1: Princip funkce simulátoru

Simulátor jsem naimplementoval jako upravenou třídu `AT_com`. Z toho plyne, že by měla mít nejméně všechny metody jako původní a dala se stejně využívat. Pozměnil jsem akorát konstruktor, protože již nebylo třeba komunikovat přes sériové rozhraní, takže nebylo třeba ani předávat ukazatel na instanci třídy `Serial`.

Pro účely testování bylo nejdůležitější poslat jakoukoli zprávu, na kterou se čeká odpověď a také to, aby odpověď přišla s určitým zpožděním. Zpoždění není definováno jako konstanta, jeho velikost se může pohybovat v daném intervalu. K tomu jsem použil dynamicky vytvořené pole zpráv dvakrát větší než velikost simulované sim karty. Není to ideální, ale testování to zcela vyhovovalo. Pokud přišla nějaká ze zpráv, na které byla definována odpověď, uložila se na simulovanou sim kartu zpráva s určitým parametrem – časovou značkou, která vyjadřovala od kdy bude zpráva platná.

Takto naimplementovanou simulací jsme mohli věrohodně emulovat chování přípravku a odladit komunikaci s webovou aplikací.

Pro další testování by se mohla naimplementovat větší inteligence simulátoru – mohl by si pamatovat nastavené hodnoty a nejlépe i generovat odpovídající události. Ale vzhledem k tomu, že v tu dobu byl již částečně funkční přípravek k dispozici, nebylo tak složitého simulátoru třeba a čas jsem věnoval implementaci dalších funkcí.

7 Závěr

První část této práce – serverová aplikace byla navržena a naimplementována přesně v souladu se zadáním. Dokáže přijmout zprávu přes mobilní telefon nebo pomocí protokolu TCP/IP a vhodně na ni zareagovat. Typická reakce je buď poslání SMS zprávy nebo uložení do databáze a zaslání informačního e-mailu.

Druhá část práce je klientská aplikace. Při vývoji aplikace bylo použitím knihovny *Qt* dosaženo toho, že je multiplatformní, dokáže načíst, modifikovat a uložit obsah eeprom paměti přípravku, ovšem pouze ze souboru. K řádnému otestování a odladění komunikace s přípravkem nedošlo, protože jsem v době tvorby aplikace neměl přípravek k dispozici.

Při vytváření této práce jsem se podrobně seznámil s využitím socketů jako komunikačního prostředku a také jsem se naučil efektivně využívat knihovny *MySQL++* a hlavně *Qt* verze 4.0. Projekt nás zaujal natolik, že se mu s kolegy hodláme dále věnovat a vylepšit již hotové komponenty.

8 Literatura

[1] Beej's guide to network programming. `http://beej.us/guide/`.

[2] *SIEMENS mobile – AT Command Set.* pdf document, 2002.

[3] K. Müller. *Programovací jazyky.* ČVUT, Praha, 2002.

[4] Tangentsoft. Dokumentace knihovny mysql++. `http://tangentsoft.net/mysql++/doc/userman/html/`.

[5] Trolltech. Dokumentace knihovny Qt. `http://doc.trolltech.com/4.0/index.html`.

9 Obsah přiloženého CD

- Zdrojové kódy serverové aplikace (+ návod ke kompilaci)
- Zdrojové kódy klientské aplikace (+ návod ke kompilaci)
- Přeložené programy (binární soubory)
- Uživatelská příručka
- Knihovna MySQL++
- Knihovna Qt verze 4.0
- Bakalářská práce ve formátu pdf + LaTeXový zdroj