VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

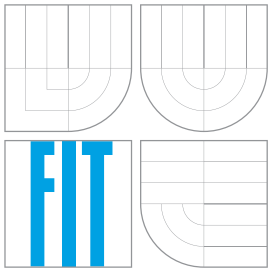
NÁZEV PRÁCE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JMÉNO PŘÍJMENÍ
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁZEV PRÁCE

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JMÉNO PŘÍJMENÍ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JMÉNO PŘÍJMENÍ, Ph.D.

BRNO 2008

Abstrakt

Výtah (abstrakt) práce v českém jazyce.

Abstract

Výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Klíčová slova v českém jazyce.

Keywords

Klíčová slova v anglickém jazyce.

Citace

Jméno Příjmení: Název práce, bakalářská práce, Brno, FIT VUT v Brně, 2008

Název práce

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana ...

.....
Jméno Příjmení
27. února 2012

Poděkování

Zde je možné uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc.

© Jméno Příjmení, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **2**

2 Radioamatéři **3**

3 Rozbor zadání **4**

4 Návrh aplikace **5**
4.1 Návrh komunikačního protokolu 5
4.1.1 Schéma packetu 5
4.2 Návrh serveru 6
4.2.1 Moduly 6
4.3 Klientská knihovna 6
4.4 Klient 6

5 Implementace **8**
5.1 Implementace serverové aplikace 8
5.1.1 Implementace databázového rozhraní 9
5.1.2 Implementace modulů 9
5.1.3 Seznam implementovaných modulů 10
5.1.4 Implementace logování 12
5.2 Implementace klientské knihovny 12
5.2.1 Abstraktní datové typy 12
5.2.2 Komunikace s klientskou aplikací 14
5.2.3 Komunikace se serverem 15
5.3 Implementace klientské aplikace 15

6 Závěr **16**

Kapitola 1

Úvod

Kapitola 2

Radioamatéři

Kapitola 3

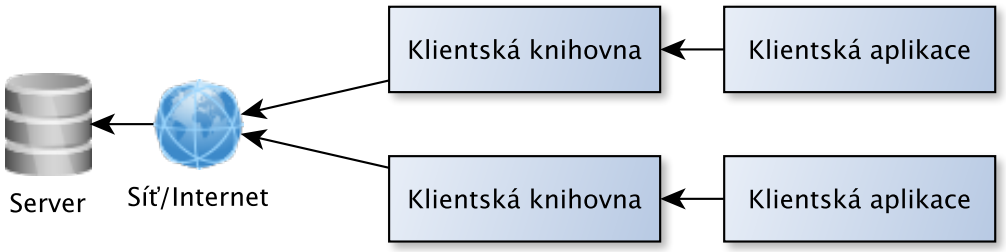
Rozbor zadání

Kapitola 4

Návrh aplikace

Cílem této kapitoly popsání návrhu aplikace a komunikačního protokolu.

Aplikace se skládá z modulárního serveru, klientské knihovny a grafického uživatelského rozhraní. V následujících podkapitolách jsou jednotlivé části stručně popsány.



Obrázek 4.1: Základní činnost programu.

4.1 Návrh komunikačního protokolu

Byl zvolen protokol inspirovaný protokolem HTTP [**?**], převážně kvůli jeho jednoduchosti a účelnosti z hlediska modularity. Jednotlivé moduly serveru mají své vlastní URI a klientské požadavky jsou pak směrovány podle URI na konkrétní modul, který vygeneruje odpověď poslanou klientovi.

Protokol je zpětně kompatibilní s protokolem HTTP, ale jsou použity jen některé jeho části.

Data přenášená protokolem HTTP jsou ve formátu CSV [**?**], kde první řádek reprezentuje hlavičku dat.

4.1.1 Schéma packetu

Dotaz na modul poskytující URI „/logbook“:

```
GET /logbook HTTP/1.1
```

Odpověď serveru:

```
HTTP/1.1 200 OK
Content-Type: text/hamlog
Content-Length: 74

id;user_id;callsign;date;qth;loc
1;1;TEST;2011;qt;
2;1;LKS;2011;;location
```

4.2 Návrh serveru

Server je konzolová aplikace zpracovávající klientské požadavky. Server je schopen obsluhovat více uživatelů současně. Uživatelé se k serveru přihlašují pomocí jména a hesla. Přihlášení je prováděno metodou Digest Access Authentication definovanou v RFC 2617. Noví uživatelé se musí nejprve registrovat.

Návrh serveru počítá s použitím jakékoliv databáze pro uchování perzistentních dat. V rámci této bakalářské práce jsem se rozhodl použít databázi SQLite3.

Server je modulární a veškeré služby, které uživateli poskytuje, jsou součástí modulů.

4.2.1 Moduly

Moduly umožňují rozšiřovat dynamicky funkčnost serveru. Každý nový požadavek, které server od klienta přijme je předán příslušnému modulu na základě URI. Modul jej zpracuje a odešle klientovi zpět odpověď. Klient si může od serveru vyžádat seznam všech modulů pomocí požadavku na speciální URI „/modules“.

Jednotlivé moduly serveru jsou realizovány jako dynamické knihovny. Při spuštění serveru jsou nahrány všechny moduly z adresáře nastavitelného pomocí konfiguračního souboru.

Každý modul obsahuje následující informace:

- URI
- Typ
- Popis

4.3 Klientská knihovna

Cílem klientské knihovny je poskytnout grafickým rozhraním jednotné API pro přístup k serveru a tím pádem zamezit duplikování kódu mezi případnými grafickými rozhraními.

Klientská knihovna má minimální závislosti a je multiplatformní.

4.4 Klient

Klient slouží koncovému uživateli k připojení k serveru, prezentaci aktuálních dat a jejich změně. Pro komunikace se serverem klient využívá klientskou knihovnu. Pro komunikaci s uživatelem pak klient využívá grafického rozhraní. Po startu klienta je uživatel vyzván k přihlášení se k serveru. Uživateli je rovněž nabídnuta možnost registrace nového účtu.

Po přihlášení zobrazí klientská aplikace veškeré záznamy v deníku a umožní jejich editaci. Klientská aplikace také zobrazuje aktuální vysílání získané ze služby DXCluster.

Kapitola 5

Implementace

V této kapitole je popsána implementace serverové aplikace, klientské aplikace a klientské knihovny.

5.1 Implementace serverové aplikace

Server byl implementován v jazyce C++ umožňujícím lepší dekompozici aplikace a použití objektového paradigmatu. Byla rovněž použita knihovna Boost, která poskytuje základní metody pro asynchronní síťovou komunikaci a nabízí programátorské prostředky nad rámec standardní STL knihovny.

Pro implementaci modulu QRZ bylo potřeba použít knihovnu pro zpracování XML. K tomuto účelu jsem použil knihovnu TinyXML.

Třída Server

Třída Server je základní třídou serveru. Vytváří soket, na kterém server přijímá připojení z klientské knihovny. Jakmile je akceptováno nové připojení, je vytvořena instance třídy Session, která dále zpracovává všechny požadavky klienta.

Třída Session

Tato třída reprezentuje sezení jednoho uživatele. Při obdržení nových dat od klienta jsou tato předána instanci třídy RequestParser, která slouží k jejich rozparsování. Pokud byla obdržena kompletní zpráva, je předána instanci třídy ModuleManager metodou handleRequest, která pak řídí její další zpracování. Výsledná odpověď je pak v třídou Session poslána zpět klientovi.

Třída RequestParser

Třída RequestParser reprezentuje konečný automat pro parsování zpráv podle specifikace komunikačního protokolu. Metoda parse zpracovává přijatá data, parsuje je, a výsledek uchovává v instanci třídy Request. Pokud dojde během parsování k chybě, vrací funkce parse hodnotu false.

5.1.1 Implementace databázového rozhraní

Návrh a implementace serveru umožňuje použití libovolného databázového rozhraní. V rámci bakalářské práce je však podpována pouze databáze SQLite3. Třída implementující konkrétní databázové rozhraní musí dědit třídu StorageBackend a implementovat její čistě virtuální metody.

Třída StorageBackend

Třída StorageBackend je základní abstraktní třídou pro implementaci jakéhokoliv databázového rozhraní. Je typu singleton a jejím smyslem je poskytnout rozhraní pro získávání dat z databázového rozhraní bez znalosti o jakou databázi se jedná. Názvy metod a podtříd jsou pojmenovány terminologií známou z SQL databází, ale prakticky lze pomocí třídy StorageBackend implementovat rozhraní pro přístup k jakémukoliv typu databáze.

Třída StorageBackend Obsahuje základní podtřídy pro definici dotazů typu SELECT, INSERT, UPDATE a CREATE známých z jazyka SQL:

- StorageBackend::Column - Obsahuje veškeré informace o sloupci tabulky (jméno, typ, velikost, příznaky pro NOT NULL, UNIQUE a PRIMARY KEY). Slouží pro definici sloupce při vytváření nové tabulky metodou StorageBackend::createTable().
- StorageBackend::Select - Zapouzdřuje data potřebná pro provedení výběru dat z databáze. Obsahuje jméno tabulky, ze které se výběr provádí, a ukazatel na dvourozměrně pole, do kterého se uloží výsledky. Dále umožňuje třída StorageBackend::Select definovat omezení výběru (v SQL jazyce klíčové slovo WHERE) a umožňuje výběr konkrétních sloupců, které vrátí ve výsledku. Instance této třídy je předána metodě StorageBackend::select() nebo StorageBackend::remove().
- StorageBackend::Insert - Obsahuje data pro vložení (v SQL jazyce INSERT) nebo aktualizaci (v SQL jazyce UPDATE) záznamu v databázi. Obsahuje název tabulky, ve které se budou data měnit, a samotná data ve formě název sloupce - hodnota. Umožňuje definovat omezení (v SQL jazyce WHERE) aplikovaná při aktualizaci dat. Instance této třídy je předána metodě StorageBackend::insert() nebo StorageBackend::update().

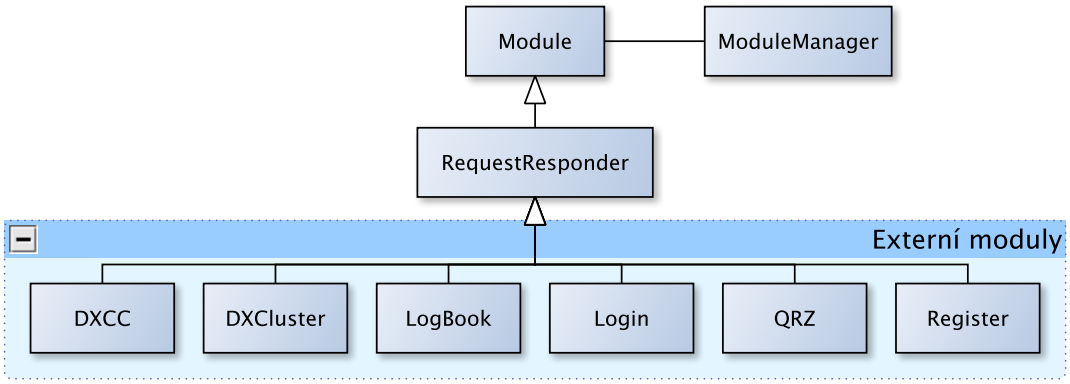
V závislosti na implementaci metod třídy StorageBackend je pak vyvolána konkrétní změna v databázi. Třída StorageBackend dále umožňuje získání identifikačního čísla naposledy vloženého záznamu metodou lastInsertedID().

Třída SQLite3Backend

Tato třída dědí třídu StorageBackend a implementuje její metody pro použití databázového systému SQLite3. V metodách update(), insert(), select(), createTable() a remove() se na základě předaných dat vygeneruje dotaz v SQL jazyce, spustí se a je vrácen výsledek.

5.1.2 Implementace modulů

Moduly jsou implementovány jako dynamické knihovny. Každý modul musí dědit třídu Modul a implementovat její čistě virtuální (pure virtual) metody. Veškeré klientské požadavky jsou pak směrovány na konkrétní modul podle URI instancí třídy ModuleManager.



Obrázek 5.1: Diagram tříd modulů.

Třída Module

Třída Module poskytuje základní třídu, kterou musí implementovat každý externí modul. Obsahuje základní informace o modulu (jeho jméno, typ a popis).

Třída RequestResponder

Tato třída dědí třídu Module a rozšiřuje ji o data a metody specifické pro modul odpovídající na klientské požadavky. Přiřazuje modulu jeho URI a informaci o tom, jestli musí být uživatel pro jeho použití přihlášen. Obsahuje také deklaraci metody handleRequest(), která je volána instancí třídy ModuleManager pro každý příchozí požadavek smerující na modul.

Třída ModuleManager

Třída ModuleManager je typu singleton (jedináček) a zabezpečuje veškerou práci serveru s externími moduly. Pomocí metody loadModules lze načíst všechny moduly z adresáře zvoleného v konfiguračním souboru. Veškeré požadavky od klientů jsou předány instanci této třídy metodou handleRequest, která je pak dále směruje podle URI na konkrétní modul. Třída také umožňuje poslat seznam všech modulů klientovi.

5.1.3 Seznam implementovaných modulů

V této podkapitole jsou popsány jednotlivé implementované moduly.

Modul Register

Modul Register (běžící na URI „/register“) slouží k registraci nových uživatelů. Při svém spuštění vytvoří pomocí instance třídy StorageBackend tabulku „users“. V metodě handleRequest pak přijímá případné požadavky na registraci uživatele a přidá nového uživatele do databáze. Pokud je již uživatel zaregistrován, vrací klientovi chybový kód.

Modul Login

Modul Login běží na URI „/login“. Jeho cílem je umožnit uživatelům přihlášení k systému. V metodě handleRequest je implementován princip přihlášení WWW-Authenticate.

Modul LogBook

Tento modul je základem celého projektu, protože umožňuje uživateli ukládat nové logy na server. Při svém načtení vytvoří tabulku „logbook“. V metodě handleRequest na základě URI provádí následující akce:

- URI „/logbook“ - Jako odpověd na dotaz pošle celý logbook v CSV formátu.
- URI „/logbook/add“ - Přidá do tabulky „logbook“ nový záznam podle CSV dat přijatých v dotazu.
- URI „/logbook/remove“ - Odstraní z tabulky „logbook“ záznam definovaný pomocí ID přijatého v dotazu.
- URI „/logbook/call“ - Jako odpověď na dotaz pošle pouze záznamy s CALL definovanou v CSV datech v dotazu.

Modul DXCC

Modul DXCC (běží na URI „/dxcc“) umožňuje získat z volací značky bližší informace o její lokaci. Data o jednotlivých prefixech jsou po startu modulu načtena z souboru „cty.csv“ v CSV formátu. V metodě handleRequest modul získá z požadavku prefix volací značky, vyhledá jej v datech načtených při startu a jako odpověď odešle informace o lokaci značky. Pokud prefix není nalezen, vrací se chybový kód.

Modul DXCluster

Modul DXCluster (běžící na URI „/dxcluster“) slouží k připojení k DXClusteru dxspots.com. Při prvním požadavku od klienta dojde k připojení na DXCluster. Veškerá data přijatá z DXClusteru jsou rozparsována a uložena v CSV formátu. Na každý další klientský požadavek odpoví modul daty získanými z DXClusteru. Jde tedy o jistou formu pollingu, kdy si klient opakovaně žádá o nová data.

Modul QRZ

Modul QRZ (běžící na URI „/qrz“) umožňuje získávat uživateli další informace o ostatních uživatelích na základě jejich volací značky. K tomuto využívá službu qrz.com. V metodě handleRequest se na základě URI provádí následující akce:

- URI „/qrz“ - Pošle QRZ serveru požadavek pro získání informací o uživateli na základě jeho volací značky.
- URI „/qrz/register“ - Umožňuje uživateli zvolení nebo změnu hesla použitého pro přihlášení k QRZ serveru.

5.1.4 Implementace logování

Logování je implementováno s použitím knihovny Log4cxx vyvíjené Apache Software Foundation a licencované pod licencí Apache License. Pokud však není při kompilaci knihovna Log4cxx nalezena, je pro logování použit standardní výstup. Výhodou Použití Log4cxx je možnost široké konfigurace logování pomoci konfiguračního souboru, možnost přesměrovat logování do souboru a tento pak automaticky rotovat na základě jeho velikosti nebo času.

Každá třída serveru má vlastní statickou instanci třídy log4cxx::LoggerPtr, kterou využívá k logování. Standardní výstup logování vypadá následovně:

5.2 Implementace klientské knihovny

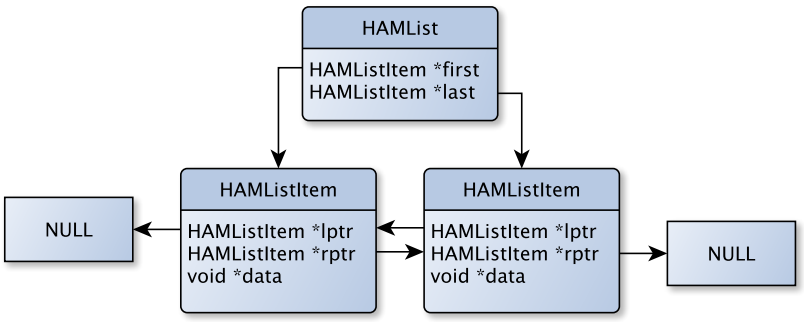
Klientská knihovna spojuje serverovou aplikaci se samotným klientským rozhraním. Klientská knihovna je navržena a implementována tak, aby ji bylo možno použít s jakýmkoliv grafickým (případně i konzolovým) rozhraním. Kvůli přenositelnosti a širší využitelnosti je napsána v jazyce C s důrazem na co nejméně závislostí na jiných knihovnách.

Klientská knihovna je rozdělena do menších bloků, které budou v této kapitole postupně popsány.

5.2.1 Abstraktní datové typy

V této podkapitole je popsána implementace abstraktních datových typů požitých v klientské knihovně.

HAMList - Seznam



Obrázek 5.2: Diagram dvousměrného seznamu.

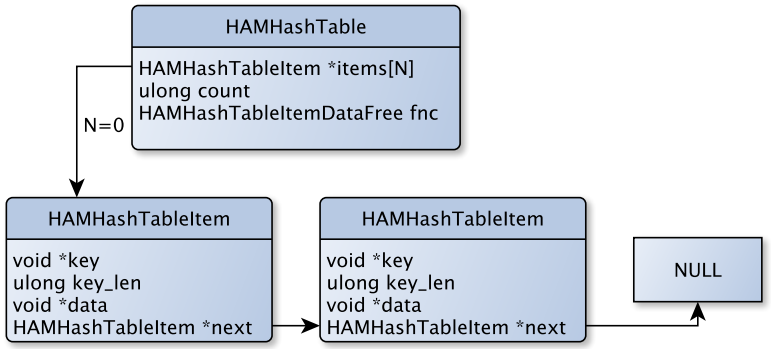
HAMList je implementací dvousměrného seznamu. Základní datové struktury použité pro definici seznamu jsou HAMList a HAMListItem:

```
typedef struct _HAMListItem {
void *data;
struct _HAMListItem *lptr;
struct _HAMListItem *rptr;
} HAMListItem;

typedef struct _HAMList {
HAMListItem *first;
HAMListItem *last;
HAMListItemDataFree free_func;
} HAMList;
```

Každá položka HAMListItem obsahuje odkaz na svého předchůdce (lptr) a následníka (rptr) a samotná data spjatá s položkou (data). Struktura HAMList obsahuje odkaz na první a poslední položku a ukazatel na funkci free_func, která je použita pro uvolnění uživatelských dat z paměti. Pokud není tato funkce definována, nejsou uživatelská data při uvolňování seznamu z paměti uvolněna.

HAMHashTable - Hashovací tabulka



Obrázek 5.3: Diagram hash tabulky seznamu.

HAMHashTable je implementací hash tabulky. Základní datové struktury použité při implementaci hash tabulky jsou HAMHashTableItem a HAMHashTable:

```
typedef struct _HAMHashTableItem {
const void *key;
void *data;
unsigned long key_len;
struct _HAMHashTableItem *next;
```

```
} HAMHashTableItem;

typedef struct _HAMHashTable {
HAMHashTableItem *items[HAM_HASH_LEN];
unsigned long count;
HAMHashTableItemDataFree free_func;
} HAMHashTable;
```

Každá položka uložená v hash tabulce obsahuje svůj klíč (key), jeho délku (key_len), data svázaná s položkou a ukazatel na další položku. Při vložení nové položky do tabulky je vypočten hash jejího kliče pomocí SDBM hashovacího algoritmu. Na základě hodnoty hashe je ukazatel na položku uložen do pole položek items. P

5.2.2 Komunikace s klientskou aplikací

Pro komunikaci s klientskou aplikací je podstatné napojení na její smyčku událostí a možnost předávat asynchronně výsledky požadavků odeslaných serveru. V této podkapitole jsou popsány řešení obou těchto problémů

EventLoop

Eventloop (neboli smyčka událostí) sdružuje metody sloužící k napojení na hlavní smyčku klientské aplikace. Pro správnou funkci klientské knihovny musí klientská aplikace implementovat všechny funkce definované ve struktuře HAMEventLoopUICallbacks a předat je klientské knihovně prostřednictvím metody ham_eventloop_set_ui_callbacks().

Funkce definované ve struktuře HAMEventLoopUICallbacks jsou pak používány dalšími částmi klientské knihovny na následující činnosti:

- timeout_add - Přidá do hlavní smyčky programu běžící v klientské aplikaci nový časocač. Klientská aplikace musí vrátit ukazatel na strukturu jednoznačně identifikující časovač a volat opakovaně funkci předánou jakou ukazatel ve zvoleném intervalu.
- timeout_remove - Odebere z hlavní smyčky časovač na základě jeho ukazatele na strukturu, která jej identifikuje.
- input_add - Přidá do hlavní smyčky klientské aplikace ukazatel na funkci, která je volána když jsou k dispozici nová data na definovaném soketu. Klientská aplikace musí vrátit ukazatel na strukturu jednoznačně identifikující tuto událost.
- input_remove - Odebere z hlavní smyčky ukazatel na funkci vstupu na základě ukazatele na strukturu, která jej identifikuje.

Díky této abstrakci je tak možno napojit klientskou knihovnu na jakoukoliv smyčku událostí.

Signály

Jednotlivé části klientské knihovny umožňují definovat signály, na které se pak může klientská aplikace napojit. Seznam signálů je uložen v hash tabulce, kde klíčem je název signálu a daty seznam funkcí, které jsou zavolány pokud je signál emitován. K registraci nových signálů slouží funkce ham_signals_register_signal().

Klientské aplikaci je umožněno funkcí ham_signals_register_handler() zaregistrovat funkci, která je zavolána při emitování signálu. Funkce musí být ve formátu HAMFetchHandler. Při registraci signálu lze rovněž definovat ukazatel na data, která jsou při emitování signálu zpracovávající funkci předána. Toho lze využít pro udržování kontextu při zpracovávání signálu.

5.2.3 Komunikace se serverem

Tato podkapitola popisuje implementaci komunikace se serverem v klientské knihovně. Je zde popsáno rozhraní pro připojení k serveru, parser komunikačního protokolu a pomocné struktury Reply a Request pro reprezetanci odchozích a příchozích paketů.

Připojení k serveru

Pro připojení k serveru je nutné vytvořit novou instanci struktury Connection funkcí ham_connection_new(). Touto funkcí se definuje adresa a port serveru, uživatelské jméno a heslo. Samotné připojení proběhne až po zavolání funkce ham_connection_connect(). Tato funkce vytvoří nový soket pro připojení k serveru a pomocí funkce ham_eventloop_input_add() přidá do hlavní smyčky programu ukazatel na funkci pro parsování přijatých dat.

Parsování dat

K parsování dat v klientské knihovně slouží HAMParser. TODO

5.3 Implementace klientské aplikace

Referenční klientská aplikace byla naprogramována v jazyce C++ s využitím grafického frameworku Qt. V této kapitole jsou stručně popsány jednotlivé třídy klientské aplikace a jejich napojení na klientskou knihovnu.

Kapitola 6

Závěr

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků se zvlášť vyznačeným vlastním přínosem studenta. Povinně se zde objeví i zhodnocení z pohledu dalšího vývoje projektu, student uvede náměty vycházející ze zkušeností s řešeným projektem a uvede rovněž návaznosti na právě dokončené projekty.

Literatura