

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

SYSTÉM PRO INTERNETOVOU KOMUNIKACI
SYSTEM FOR INTERNET COMMUNICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ KUBOWSKÝ
AUTHOR

VEDOUCÍ PRÁCE — ING. PETR KRČEK
SUPERVISOR

BRNO 2011

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

(zde bude vložen originál, nebo kopie zadání této práce)

ABSTRAKT

Tato bakalářská práce se zabývá vypracováním návrhu a realizací systému pro internetovou komunikaci. Systém umožňuje komunikaci mezi dvěma uživateli a komunikaci více uživatelů v diskuzi. Systém je založen na síťové architektuře klient/server a je vyvíjen pod operačním systémem Windows7. Pro naprogramování aplikace je použit programovací jazyk C#.

ABSTRACT

This bachelor´s thesis solves development and realization of a system for internet communication. The System must allow communication betwen two clients and communication of more users in discussion. The System is based on client/server network architecture. It was developed under OS Windows 7 and implemented using C# programing language.

KLÍČOVÁ SLOVA

C#, C sharp, .NET, .NET Framework, Internetová komunikace, klient/sever, Windows NT

KEYWORDS

C#, C sharp, .NET, .NET Framework, Internet communication, client/sever, Windows NT

PODĚKOVÁNÍ

Děkuji vedoucímu mé bakalářské práce Ing. Petru Krčkovi, za pomoc poskytnutou při jejím vypracování.

Obsah:

Zadání závěrečné práce....3
Abstrakt....5
Poděkování....7
1 Úvod....11
2 Přehled stávajících programů pro internetovou komunikaci....13
2.1 Google Talk....13
2.2 Skype....13
2.3 Windows Live Messenger (Windows Messenger)....14
2.4 ICQ....14
2.5 Yahoo! Messenger....14
2.6 Trillian....15
2.7 Pidgin....15
3 Vlastní návrh systému....17
3.1 Návrh komunikace....17
3.2 Návrh serverové aplikace....18
3.2.1 Obecný popis....18
3.2.2 Třída spik_data („paměťová oblast“)....19
3.2.3 Třída spik_db....20
3.2.4 Třída spik_xmlfile....21
3.2.5 Třída spik_clientWork....22
3.2.6 Třída spik_srvWork....23
3.2.7 Třída spik_server....23
3.3 Návrh klientské aplikace....24
3.3.1 Obecný popis....24
3.3.2 Třída spik_xmlfile....25
3.3.3 Třída spik_memory („paměťová oblast“)....25
3.3.4 Třída spik_error....26
3.3.5 Třída spik_tcp_connection....26
3.3.6 Třída applicationLogoutClass....27
3.3.7 Třída Main....27
3.3.8 Třída hlavního formuláře....29
3.3.9 Informační formulář....30
3.3.10 Třída formuláře pro komunikaci mezi uživateli....31
3.3.11 Rozšíření třídy vykreslení pro komunikaci mezi uživateli....31
3.3.12 Třída formuláře vyhledávání....32
3.3.13 Třída formuláře komunikace v diskuzi....33
3.3.14 Rozšíření třídy formuláře komunikace v diskuzi....33
4 Ovládání aplikace....35
4.1 Serverová aplikace....35
4.2 Klientská aplikace....36
5 Závěr....41
Seznam použité literatury....43

1 ÚVOD

Samotný internet vznikl za účelem sdílení informací a vývoj komunikačních programů byl jen přirozený krok. Na počátku byly aplikace umožňující jen vyvěšení textu a jeho čtení ostatními uživateli, poté přišla samotná textová komunikace používající vlastní protokoly a pokračovalo se přes hlasovou komunikaci až k dnešním videokonferencím.

V práci nalezneme nejdříve přehled nejvýznamnějších aktuálních programů pro internetovou komunikaci, zde jsou jednotlivě popsány. Poté následuje kapitola o vlastním návrhu systému, ve které nejdříve projdeme návrh komunikační úrovně a návrhy jednotlivých aplikací rozšířené o popis jejich vytvořených tříd. Předposlední kapitola popisuje hotové aplikace a závěrečná kapitola přináší shrnutí práce.

Pro vypracování samotné aplikace byl zvolen programovací jazyk C#. Pro její vývoj bylo použito vývojové prostředí Microsoft Visual C# 2010 Express. Toto vývojové prostředí bylo zvoleno proto, že bylo vyvinuto stejnou společností jako samotný jazyk a jedná se o verzi použitelnou pro studijní účely zdarma. Pro rozšíření znalostí tohoto jazyka na úroveň nutnou k realizaci aplikace bylo čerpáno z online nápovědy vývojového prostředí [1] a ze série článků na téma C# [2], které lze nalézt na serveru www.zive.cz.

Při vytváření návrhu samotného systému bylo třeba rozhodnout o třech základních předpokladech, ze kterých bude návrh vycházet. Prvním předpokladem bylo, zda budeme uvažovat tok dat pouze ve směrech klient-server a server-klient nebo zda budeme uvažovat kromě směrů klient-server a server-klient i směr klient-klient. Druhým předpokladem bylo, zda budeme uvažovat ukládání dat (například: tabulka registrovaných uživatelů) v podobě nějakého souboru nebo zdali budeme tato data ukládat do nějaké databáze. Posledním hlavním předpokladem bylo, jaký použijeme komunikační protokol pro přenos samotných dat.

V případě prvního předpokladu bylo rozhodnuto, že v rámci sítě bude probíhat komunikace pouze ve směrech klient-server a server-klient, tedy že veškerá data budou probíhat přes serverovou aplikaci. Komunikace, která bude z pohledu uživatele vypadat jako klient-klient, bude ve skutečnosti komunikací klient-server-klient. Takto bude pro serverovou aplikaci jednodušší udržet si přehled o probíhajících datech a otevírá se nám zde možnost případného rozšíření aplikace o ukládání dat pro nepřítomné klienty. U druhého předpokladu bylo rozhodnuto použít databázi pro ukládání dat serveru, kterými jsou tabulka uživatelů a tabulka místností. Data jednotlivých aplikací (nastavení serveru, seznam známých uživatelů klienta, …) jsou ukládány do XML souborů. Pro jmenované tabulky byla zvolena databáze z důvodu jejich možné velikosti a podstaty jejich dat. Třetí předpoklad byl vyřešen zvolením protokolu TCP, který zajišťuje spojení se spolehlivým doručením dat. Tedy není potřeba nějak ověřovat zda-li data došla, popřípadě v jakém stavu. Tento protokol byl zvolen s ohledem na účel vyvíjené aplikace.

Aplikace byla odzkoušena v prostředí školní sítě. I když je aplikace v praxi použitelná, původně byla vyvíjena za účelem použití jako učební pomůcky ve cvičeních předmětu Počítačové sítě.

2 PŘEHLED STÁVAJÍCÍCH PROGRAMŮ PRO INTERNETOVOU KOMUNIKACI

Pro samotný přehled stávajících programů pro internetovou komunikaci bylo vybráno 7 nejvýznamnějších aplikací, které jsou nejvíce rozšířeny, jmenovitě Google Talk, Skype, Windows Live Messenger, ICQ, Yahoo! Messenger, Trillian a Pidgin.

2.1 Google Talk



Obr. 1 Logo aplikace Google Talk.

Jde o aplikaci pro internetovou komunikaci od společnosti Google, která je založená na protokolech XMPP (dříve Jabber) a Jingle. Pro samotnou textovou komunikaci využívá tato aplikace právě protokol XMPP, druhý protokol využívá k vyšším službám jako je hlasová komunikace.

Samotná aplikace není nijak zpoplatněná, pouze se vyžaduje od případného klienta mít vytvořený Google účet, se kterým se jde přihlásit i k dalším službám jako například Gmail. Google Talk je funkční pod operačními systémy Windows 2000, XP a novějšími.

Kromě samotného Google Talk klienta se mohou do komunikační sítě aplikace Google Talk také připojit klienti jako například Miranda, Pidgin, Trillian a další.

Samotný Google Talk klient je jednoduchý a nemá omezenou délku zprávy, přičemž samotná komunikace může být šifrována.

Předcházející odstavce čerpají z [3].

2.2 Skype



Obr. 2 Logo aplikace Skype.

Jde o aplikaci pro internetovou komunikaci od společnosti Skype Limited, která je spíše známější jako aplikace pro telefonování přes internet, ale která také umožňuje textovou komunikaci.

Samotná textová komunikace je zdarma, stejně jako některé další služby v rámci sítě Skype klientů. Aplikace také nabízí několik zpoplatněných služeb.

Aplikace jako taková je svým provozem náročnější na systém.

Předcházející odstavce čerpají z [4].

2.3 Windows Live Messenger (Windows Messenger)



Obr. 3 Logo aplikace Windows Messenger.

Windows Live Messenger je aplikace pro internetovou komunikaci od společnosti Microsoft, která navazuje na aplikaci Windows Messenger, a která je známa také pod názvem MSN Messenger. Kromě textové komunikace, pracující v tomto případě s kratšími zprávami, umožňuje tato aplikace stejně jako další zde zmíněné i vyšší funkce jako je videokonference.

Windows Live Messenger bývá standardní součástí novějších operačních systémů Windows, což napomohlo zvýšit počet uživatelů této aplikace. Toto zahrnutí této aplikace ovšem vede i k jistým rizikům co se týče bezpečnosti systému.

Do sítě aplikace Windows Live Messenger se jde přihlásit i skrze webové rozhraní a také prostřednictvím několika dalších aplikací, což je ale v rozporu s podmínkami užití aplikace Windows Live Messenger.

Předcházející odstavce čerpají z [5] a [8].

2.4 ICQ



Obr. 4 Logo aplikace ICQ.

ICQ je aplikace pro internetovou komunikaci od společnosti ICQ LLC, která umožňuje kromě rychlé textové komunikace a vyšších služeb i uchovávání odeslaných zpráv v rámci ICQ sítě pro uživatele, kteří nejsou aktuálně přihlášeni, a jejich doručení při přihlášení příjemce. Samotný přenos dat probíhá v reálném čase.

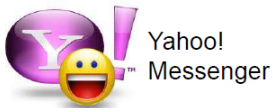
Samotná zkratka ICQ se v angličtině čte stejně jako sousloví „I seek you“, což se dá přeložit jako „hledám tě“.

Tato aplikace je pro užití zdarma, ovšem je vyžadováno použití oficiálního klienta a data zaslaná skrze síť této aplikace se stávají majetkem firmy provozující ICQ.

Jedná se asi o nejznámější aplikaci pro internetovou komunikaci dneška.

Předcházející odstavce čerpají z [6] a [7].

2.5 Yahoo! Messenger



Obr. 5 Logo aplikace Yahoo! Messenger.

Jde o aplikaci pro internetovou komunikaci od firmy Yahoo! Inc, která kromě samotné textové komunikace umožňuje i vyšší funkce jako například videopřenos. Aplikace je vyvíjena pro operační systémy Windows, Linux a MacOS.

Do sítě aplikace Yahoo! Messenger je možné se připojit i pomocí několika neoficiálních aplikací jako například Miranda, nebo Trillian.

Samotné použití této aplikace je podmíněno účtem u emailové služby Yahoo!

Předchozí odstavce čerpají z [8].

2.6 Trillian



Obr. 6 Logo aplikace Trillian.

Jde o aplikaci pro internetovou komunikaci od společnosti Cerulean Studios, která podporuje užití více protokolů, což znamená, že se s ní přihlásíte do sítě aplikací jako například ICQ, MSN Messenger a Yahoo!. Aplikace umožňuje přihlášení na více účtů zároveň, přičemž nezáleží, na kterou z podporovaných sítí se přihlašujete.

Aplikace podporuje i vyšší služby těchto protokolů jako je videokonference a přidává pár vlastních služeb.

Samotná aplikace je pro uživatele v základní verzi zdarma a to bez reklam a nutnosti registrace, přičemž tato základní verze nese označení Basic. Další rozšíření aplikace o podporu služeb, které základní verze nepokrývá, je zpoplatněno.

Předchozí ostavce čerpají z [9] a [10].

2.7 Pidgin



Obr. 7 Logo aplikace pidgin.

Pidgin je aplikace pro internetovou komunikaci vyvíjená pod licencí GNU GPL, která je vyvíjena pro operační systémy Windows, Linux a další UNIX systémy. Aplikace podporuje komunikační protokoly aplikací jako jsou například ICQ, MSN Messenger, Yahoo! a IRC.

Tato aplikace pro internetovou komunikaci standardně nepodporuje audio a video přenosy dat. Ovšem jde funkčně rozšířit formou vyvíjených pluginů. Vývoj samotné aplikace je ve stádiu stálého vývoje.

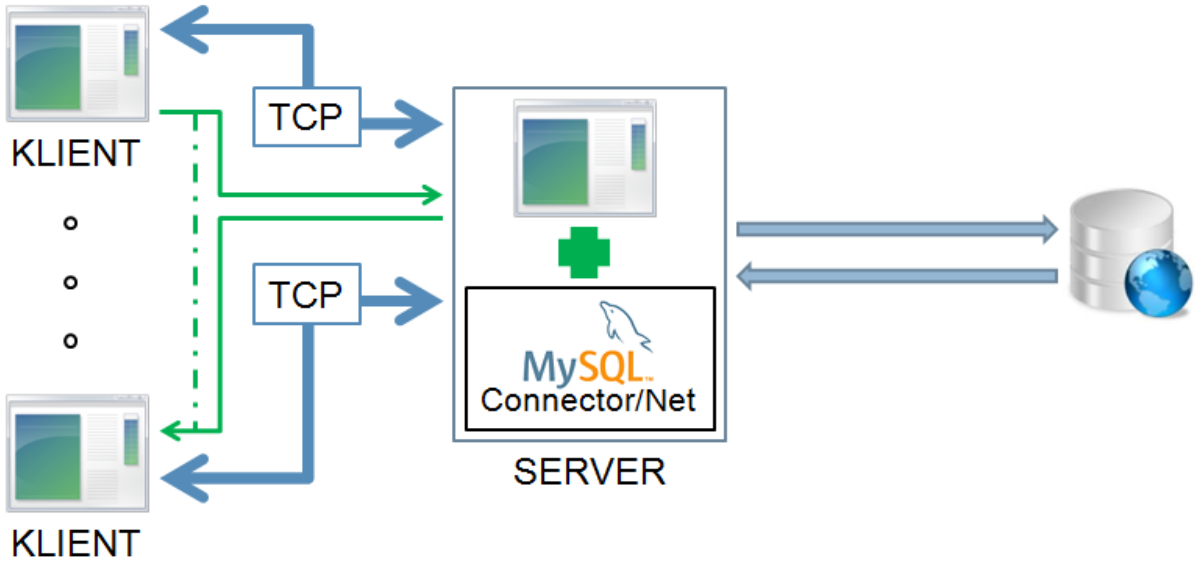
Aplikace je zcela zdarma a bez reklam. Zdrojový kód aplikace je „open source“, což umožňuje vytvářet vlastní modifikace.

Předchozí odstavce čerpají z [8] a [11].

3 VLASTNÍ NÁVRH SYSTÉMU

Klient a server byly vyvíjeny jako samostatné aplikace.

3.1 Návrh komunikace



Obr. 8 Schéma návrhu komunikace.

Jak je vidět na schématu návrhu komunikace (obr.8), tak byl určen předpoklad toku dat jen ve směrech server-klient a klient-server, za použití protokolu TCP. Samotný server navíc využívá ve formě přiložené knihovny MySQL Connector pro .NET, který je vyvinut pod licencí GPL License. Server používá MySQL Connector pro spojení s MySQL databází.

Pokud chce klient zaslat data jinému klientovi, tak zašle tato data s patřičnou hlavičkou serveru, ten si přečte hlavičku a zjistí, že je třeba tato data přeposlat, nahradí informaci o příjemci informací o odesilateli a zašle tato data s upravenou hlavičkou cílovému klientovi. Tento proces je ve schématu návrhu komunikace (obr.8) znázorněn zelenými šipkami, přičemž přerušovaná čára reprezentuje úmyslnou cestu dat od klienta klientovi a plná skutečný tok dat.

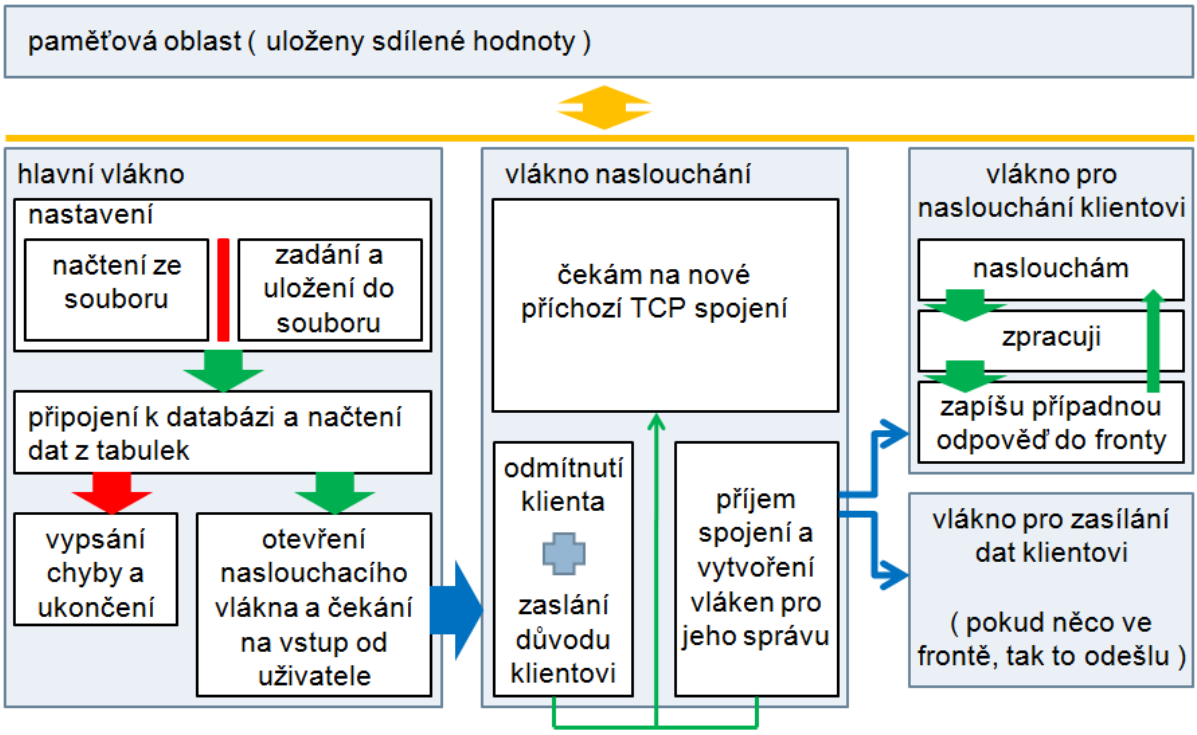
Předpokládá se i možnost fungování několika serverů najednou, přičemž si klient vybere, na který se připojí.

Samotná data jsou přenášena v bajtové posloupnosti, do které jsou převáděna z textové podoby, se kterou se pracuje v rámci jednotlivých aplikací, prostřednictvím kódování UTF8. Přijímací strana si data opět převede do textové podoby.

3.2 Návrh serverové aplikace

Serverová aplikace navrhovaného systému je řešena jako konzolová aplikace. Samotný návrh předpokládá možnost připojení se na SQL server. Na něm je potřeba vytvořit odpovídající tabulky, jejichž konstrukční příkazy jsou uvedeny v přikládaném SQL souboru. Předpokládá se užití 2+(*x**2) vláken, kde *x* je počet připojených klientů.

3.2.1 Obecný popis



Obr. 9 Schéma návrhu serverové aplikace.

Samotná aplikace využívá takzvanou „paměťovou oblast“, která je zde v podobě třídy se statickými hodnotami pro ukládání globálních údajů. Tuto třídu používají všechna aktivní vlákna a z tohoto důvodu je v rámci této třídy řešena možnost případné kolize vláken při čtení a zapisování.

Při spuštění samotné aplikace se začne provádět kód hlavního vlákna, které se při prvním spuštění nebo v případě nenalezení svého konfiguračního xml souboru dotáže na hodnoty, které je třeba nastavit. Jedná se o naslouchací port, přístupové údaje pro databázi a maximální počet přihlášených uživatelů. Tyto informace se v textové podobě uloží do xml souboru, ze kterého se při dalších spuštěních budou načítat. Poté se hlavní vlákno pokusí připojit k databázi a načíst si údaje z tabulky uživatelů a tabulky místností, přičemž pokud se to nepodaří, je vypsána chybová hláška a aplikace ukončena. Pokud se to podaří, je otevřeno nové vlákno pro naslouchání pro příchozí spojení a na konci hlavního vlákna spuštěna smyčka, čekající na vstup od uživatele, který zpracuje jako příkaz. Při zapsání příkazu help se nám zobrazí seznam implementovaných příkazů.

Vlákno pro naslouchání po svém spuštění provádí podmíněnou smyčku, která na svém začátku čeká na další příchozí spojení. Při příjmutí nového spojení provede několik operací. V případě překročení limitu maximálního počtu uživatelů zašle klientovi odpovídající informaci a poté ukončí spojení. Nebo vytvoří soukromé vlákno pro naslouchání danému klientovi,

zapíše odpovídající údaje v „paměťové oblasti“ a předá spojení vytvořenému vláknu.

Vlákno pro naslouchání klientovi na svém začátku spustí nové vlákno pro zasílání dat danému klientovi a poté začne provádět vlastní podmíněnou smyčku. Smyčka na svém začátku čeká na blok příchozích dat. Pokud nastane problém se spojením nebo s přijatými daty, tak jsou tato vlákna a spojení s klientem ukončena. Pokud jsou příchozí data v pořádku, jsou tato data odpovídajícím způsobem zpracována. Pokud potřebujeme v závislosti na příchozích datech zaslat nějaká data klientovi, tak tato data v případě již přihlášeného uživatele zapíšeme do fronty, jenž je přidělena každému přihlášenému uživateli v „paměťové oblasti“, nebo v případě nepřihlášeného uživatele rovnou odešleme zpět. Pokud je potřeba zaslat data jinému přihlášenému uživateli, tak jsou tato data zapsána do jeho přidělené fronty.

Vlákno pro zasílání dat klientovi provádí pouze podmíněnou smyčku, ve které kontroluje přidělenou frontu, jestli v ní jsou nějaké čekající záznamy. Tyto případné záznamy z fronty odebírá a zasílá přidělenému klientovi.

V rámci funkce konzole serverové aplikace, jenž je součástí hlavního vlákna, lze použít příkaz pro zobrazování přichozích a odchozích dat, přičemž jednotliví připojení klienti zde jsou označeni symbolem # a číselným indexem. Směr toku dat (příchozí/odchozí) je znázorněn malou šipkou.

3.2.2 Třída spik_data („paměťová oblast“)

```
public class spik_data
{
    // objekty pro zamikani pro vlakna :
    private Object threadLock = new Object(); // pro promenou s poctem pripojenych
    private Object threadLock2 = new Object(); // pro zapis/cteni/mazani u tabulek
    // hodnoty :
    public static bool echoMode = false;
    public static int i  = 0; // nastaveni indexu - clients
    public static Queue emptyIndex  = new Queue(); // seznam uvolnenych id pro pole clients
    public static int connections = 0;
    public static ArrayList DBdata; // udaje pro pripojeni k DB
    public static ArrayList Rooms = new ArrayList(); // udaje pro mistnosti
    public static ArrayList Users = new ArrayList(); // udaje pro uzivatele
    public static ArrayList Clients = new ArrayList(); // udeje pripojenych klientu
    // metody :
    public ArrayList dataValue(int e)...
    public Object dataValue(int e, int x, int y)...
    public int dataSave(int e,ArrayList d)...
    public bool dataSave(int e, int x, int y, Object value)...
    public bool dataDelete(int e, int index)...
    public int connectionsRead()...
    public void connectionsUpdate(int e)...
    public void connectionsReset()...
    public void Clear()...
    public void innerQueueEnqueue(int e, int x, int y, string data)...
    public string innerQueueDequeue(int e, int x, int y)...
    public void innerQueueClear(int e, int x, int y)...
    public void echoModeWork(bool e)...
    public bool echoModeWork()...
}
```

Obr. 10 Náhled třídy spik_data používané v serverové aplikaci.

Tato třída slouží pro ukládání hodnot, které si aplikace musí pamatovat.

```
Object ObjektProUzamceni = new Object();
lock (ObjektProUzamceni)
{
    // blok příkazů
}
```

Obr. 11 Obecné zapsání kódu využívajícího příkaz „lock“.

Přístup více vláken k těmto hodnotám je zde ošetřen za pomoci dvou objektů a příkazu „lock“, zahrnutého v odpovídajících metodách této třídy. Zajišťuje uzamčení uvedeného objektu pro provádějící vlákno, provedení následujícího bloku příkazů a odemčení uzamčeného objektu pro ostatní vlákna. Pokud je objekt již uzamčen, vlákno čeká na jeho odemčení. Není důležité, která metoda je volána, ale že příkaz „lock“ pracuje se stejným objektem. (obr.11)

Údaje o místnostech, uživatelích a připojených klientech jsou zde uchovávány ve formě „tabulek“, které jsou konstruovány jako dvourozměrné pole objektů. Platí, že každý jednotlivý údaj má přidělen svůj sloupec a každý jednotlivý záznam (uživatel/klient/místnost) má přidělen svůj řádek.

Pro zápis dat do zmíněných „tabulek“ je zápis zprostředkováván pomocí dvou variant metody *dataSave*, které voláme podle toho, zda-li chceme zapsat najednou celý nový řádek nebo jen do daného sloupce existujícího řádku. Stejně tak můžeme také číst z těchto tabulek celý řádek nebo jen daný sloupec vybraného řádku, v závislosti na volané variantě metody *dataValue*. Pouze v případě mazání dat se zbavujeme vždy celých řádků (obr.10).

Třída také zahrnuje tři metody pro manipulaci s uloženou hodnotou aktuálního počtu uživatelů, metodu pro vyčištění od všech uložených dat, tři metody pro manipulaci s jednotlivými frontami, které jsou zahrnuty v „tabulkách“ zmíněných výše, a dvě metody pro nastavování a čtení hodnoty, podle které bude nebo nebude serverová aplikace vypisovat přijatá a odchozí data (obr.10).

3.2.3 Třída spik_db

```
public class spik_db
{
    private MySqlConnection activeConnection;
    private MySqlCommand activeCommand;
    public MySqlDataReader activeReader;
    private int affectedRowsValue;
    public bool makeConnection(string[] access)...
    public bool destroyConnection()...
    public bool doSQLcommand(bool e, string cmd)...
    public int affectedRows()...
}
```

Obr. 12 Náhled třídy spik_db používané v serverové aplikaci.

Tato třída využívá třídy zahrnuté v přiložené knihovně MySQL Connector (mysql.data.dll), jmenovitě *MySqlConnection*, *MySqlComand* a *MysqlDataReader*.

Třída obsahuje metody pro připojení k databázi, odpojení od databáze, zpracování SQL příkazu a vrácení počtu ovlivněných řádků. V případě metody pro vykonání SQL příkazu je součástí vstupních dat i logická hodnota určující zda metoda má nebo nemá očekávat u daného SQL příkaz nějakou návratovou hodnotu od databáze.

3.2.4 Třída spik_xmlfile

```
public class spik_xmlfile
{
    // hodnoty :
    private string file;
    // metody :
    public bool create()...
    public bool delete()...
    public ArrayList readData()...
    public bool writeData(ArrayList save)...
    public spik_xmlfile(string e)...
}
```

Obr. 13 Náhled třídy spik_xmlfile používané v serverové aplikaci.

Tato třída umožňuje aplikaci pracovat s XML souborem o definované struktuře. V aplikaci je používána pro ukládání a načítání konfiguračních údajů serverové aplikace, mezi které patří například přístupové údaje k databázi.

Třída obsahuje metody pro vytvoření souboru, smazání souboru, čtení souboru a zápis do souboru. V případě metody pro čtení je kontrolována hodnota načtených hodnot s hodnotou očekávaných hodnot a pokud tyto hodnoty nesouhlasí, je otevíraný soubor považován za poškozený a v důsledku toho smazán, načež se aplikace na odpovídající údaje dotáže.

3.2.5 Třída spik_clientWork

```
public class spik_clientWork
{
    // predpokladane promene
    string usersTable = "users";
    string roomsTable = "rooms";
    // nacteme objekty serveru
    spik_data client_data = new spik_data();
    spik_db client_mysql = new spik_db();
    // promene :
    private int clientID = -1;
    public bool doWork = true;
    ...
    // metody :
    public spik_clientWork(int index, TcpClient newClient)...
    private byte[] dataBlockWrite(string e, int i)...
    private string dataBlockRead(byte[] e)...
    private void clientCommunicationWrite()...
    public void clientCommunication()...
    public void Close()...
    private bool userLogout()...
    private bool userLogin(string data)...
    ...
    private void clientCommand(string cmd)...
}
```

Obr. 14 Náhled třídy spik_clientWork používané v serverové aplikaci (zkráceno).

Tato třída je využívána vlákny, která se starají o obsluhu jednotlivých klientů, přičemž každý pár vláken určený jednomu uživateli má svou instanci této třídy. Součástí třídy jsou pevně definované hodnoty odpovídající názvům tabulek v databázi, které jsou zde využívány při ukládání nově registrovaného uživatele či nově vytvořené místnosti.

Jsou zde zahrnuty třídy *spik_data* a *spik_db* pro ukládání a načítání potřebných dat a také několik vnitřních proměnných pro uchovávání vnitřních stavů.

Metody *dataBlockWrite* a *dataBlockRead* převádějí daný blok dat, která budeme odesílat nebo jsem právě přijali, z bajtové do textové a z textové do bajtové podoby. K samotnému převodu mezi textovou a bajtovou podobou je použito kódování UTF8. Metoda *Close* slouží pro ukončení komunikace s daným klientem a přidělených vláken. (obr.14)

Metoda *clientCommunicationWrite* je používána vláknem pro zasílání dat danému klientovi a zahrnuje smyčku kontrolující a zpracovávající frontu přidělenou uživateli. Metoda *clientCommunication* je hlavní metodou této třídy a zajišťuje zpracování příchozích dat od klienta, přičemž kontroluje, jestli je spojení stále aktivní. Při samotném zpracování příchozích dat je daný blok dat v textové podobě předáván metodě *clientCommand*, která v případě známé hlavičky provede odpovídající zpracování přijatých dat. Tato operace je většinou reprezentována voláním odpovídající metody této třídy, mezi které patří například metoda *userLogin*.

V „paměťové oblasti“ je v tabulce aktivních připojení veden ukazatel pro každou instanci této třídy a každá instance této třídy si pamatuje hodnotu indexu svého řádku.

3.2.6 Třída spik_srvWork

```
public class spik_srvWork
{
    // nacteme objekty serveru
    spik_data server_data = new spik_data();
    spik_db server_db = new spik_db();
    // promene :
    private TcpListener listener;
    private bool listenEnabled = true;
    ...
    // public promene :
    public string usersTable, roomsTable;
    // metody :
    private byte[] dataBlockWrite(string e, int i)...
    private void server_listen()...
    private bool server_listenThread()...
    private bool server_console(string command)...
    public void DoWork()...
}
```

Obr. 15 Náhled třídy spik_srvWork používané v serverové aplikaci.(zkráceno)

Tato třída a její metody jsou používány hlavním vláknem programu a vláknem pro nasluchání (pro nové příchozí připojení). Třída má definovány dvě veřejné vnitřní hodnoty, do kterých se nastavuje hodnota názvu tabulek. Ty se předpokládají v databázi.

Metody *server_listen* a *server_listenThread* jsou užívány naslouchajícím vláknem a zahrnují kompletní kód prováděného úkonu. Metoda *server_console* zpracovává vstup uživatele po započetí aktivního naslouchání serveru a pokud rozpozná známý příkaz, provede odpovídající akci. (obr.15)

Hlavní metodou této třídy je metoda *DoWork*, která v sobě zahrnuje provedení celé procedury hlavního vlákna. To je uvedeno v schématu návrhu serverové aplikace (obr.9).

3.2.7 Třída spik_server

```
public class spik_server
{
    public static void Main(string[] args)
    {
        spik_srvWork server = new spik_srvWork();
        server.usersTable = "users";
        server.roomsTable = "rooms";
        server.DoWork();
    }
}
```

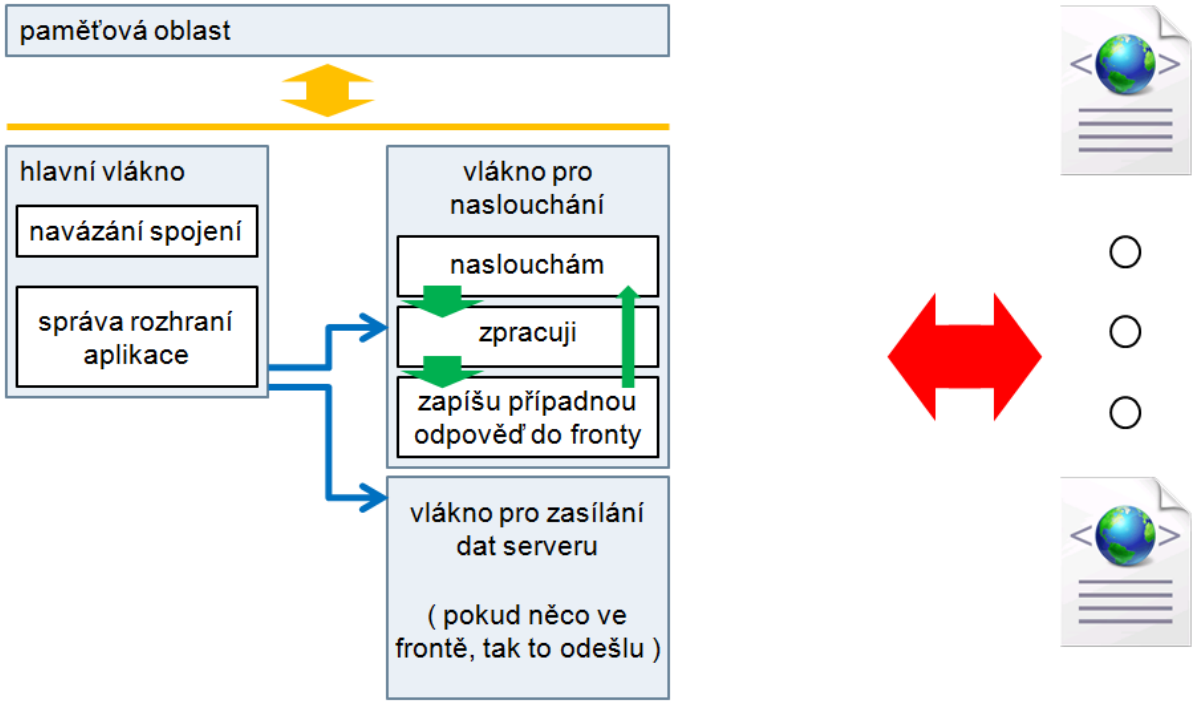
Obr. 16 Náhled třídy spik_server používané v serverové aplikaci.

Tato třída je hlavní třídou samotné serverové aplikace a zahrnuje vytvoření objektu třídy *spik_srvWork*, u které provede nastavení názvů databázových tabulek a zavolá její hlavní metodu *DoWork*.

3.3 Návrh klientské aplikace

Klientská aplikace navrhovaného systému je řešena jako formulářová aplikace. Samotný návrh předpokládá použití jednoho nebo tří vláken, v závislosti na tom, zda-li je aktivní spojení se serverem. Ve výčtu vytvořených formulářů nebudu uvádět třídy zajišťující vykreslení samotné plochy formulářů, které byly vygenerovány vývojovým prostředím. Pouze v případě, že došlo k rozšíření této třídy, bude uvedeno toto rozšíření.

3.3.1 Obecný popis



Obr. 17 Schéma návrhu klientské aplikace.

Samotná aplikace využívá takzvané „paměťové oblasti“. Jedná se o obdobu „paměťové oblasti“ serverové aplikace.

Při spuštění samotné aplikace se začne provádět kód hlavního vlákna, které v případě nenalezení svého konfiguračního xml souboru načte naposledy použité údaje, kterými jsou adresa serveru, port na kterém server naslouchá a uživatelské jméno. V rámci tohoto vlákna je spravováno i překleslování formulářů a zobrazení hlavního formuláře aplikace.

V případě, že se aplikace připojí k serveru, jsou vytvořena dvě vlákna pro poslech příchozích zpráv ze serveru a pro odesílání dat na server. Obdobně jako u serverové aplikace zde vlákno pro odesílání dat pracuje s frontou a vlákno pro příjem dat zpracovává přijatá data a zajišťuje zařazení případné reakce do fronty k odeslání.

Klientská aplikace si ukládá při přihlašování poslední uvedené hodnoty jako je například adresa serveru do XML souboru. Taktéž si aplikace ukládá zvlášť pro každého uživatele XML soubor se seznamem známých místností a uživatelů. Jedinečnost názvu je zajištěna zahrnutím loginu uživatele v názvu ukládaného souboru.

Aplikace zahrnuje kromě vlastního formuláře i formulář pro zasílání a zobrazení zpráv klientům, pro práci s místností, pro hledání uživatelů a místností v systému a pro zobrazení

informací o programu. Jednotlivé formuláře mají více možných vykreslení v závislosti na situaci, ve které jsou použity.

3.3.2 Třída spik_xmlfile

```
public class spik_xmlfile
{
    // hodnoty :
    private string file;
    // metody :
    public bool create()...
    public bool delete()...
    public ArrayList readData()...
    public ArrayList readDataList(string type)...
    public bool writeData(ArrayList save)...
    public bool writeDataList(ArrayList save, ArrayList save2)...
    public spik_xmlfile(string e)...
}
```

Obr. 18 Náhled třídy spik_xmlfile používané v klientské aplikaci.

Tato třída umožňuje aplikaci pracovat s XML souborem o definované struktuře. V aplikaci je používána pro ukládání a načítání konfiguračních údajů klienské aplikace a seznamů známých uživatelů a místností pro jednotlivé uživatele.

Jedná se o obdobu stejnojmenné třídy v serverové aplikaci.

3.3.3 Třída spik_memory („paměťová oblast“)

```
// class for static variables
public class spik_memory
{
    // hodnoty :
    public static int draw_usersX, draw_usersY, draw_roomsX, draw_roomsY;
    public static bool clientReady = false; // is client loged in ?
    ...
    // metody :
    public int MessageFormsNew(ArrayList d)...
    public bool MessageFormsDel(int i)...
    ...
}
```

Obr. 19 Náhled třídy spik_memory používané v klientské aplikaci (zkráceno).

Tato třída slouží pro ukládání hodnot, které si aplikace musí pamatovat.

Kromě dat uchovávaných pro daného uživatele se zde ukládají i data spojená s vykreslováním některých částí aplikace a také ukazatele na objekty, u kterých je potřeba průběžně měnit vykreslení. Ukládají se zde i objekty jednotlivých formulářů, kromě informačního formuláře, vyhledávacího formuláře a formuláře pro vytvoření nové místnosti. Tyto tři zmíněné výjimky jsou vytvořeny jako objekty třídy hlavního formuláře.

Jde o obdobu „paměťové oblasti“ u serverové aplikace, která ale v případě klientské aplikace obsahuje podstatně více metod, v závislosti na vyšší rozmanitosti uchovávaných údajů v případě klientské aplikace.

3.3.4 Třída spik_error

```
public class spik_error
{
    // nacteni zdilenych dat :
    private spik_memory loadMemory = new spik_memory();
    // promene :
    public static int errorType;
    // metody :
    public void errorShow(int e, string text)...
    public void errorContinue()...
}
```

Obr. 20 Náhled třídy spik_error používané v klientské aplikaci.

Tato třída je využívána pro zpracování případných chyb, které jsou v aplikaci odchytávány. V případě, že dojde v kontrolovaném místě k chybě, zavolá se metoda *errorShow*, která zajistí nastavení vnitřní číselné hodnoty odpovídající chybě a překreslení hlavního formuláře do stavu zobrazení chyby. Z tohoto zobrazení se navrací do normálního stavu prostřednictvím tlačítka, které zavolá metodu *errorContinue*. Ta v závislosti na typu chyby překreslí formulář do stavu, který tomu odpovídá (obr.20).

3.3.5 Třída spik_tcp_connection

```
// class for TCP connection
public class spik_tcp_connection
{
    // nacteni dat :
    private spik_memory loadMemory = new spik_memory();
    private spik_xmlfile loadXML;
    private spik_xmlfile loadData = new spik_xmlfile("spik_client.xml");
    // promene :
    private ArrayList resultsForUserFindForm = new ArrayList();
    private bool errorActive = false;
    public bool doWork = true;
    ...
    // metody :
    private byte[] dataBlockWrite(string e, int i)...
    private string dataBlockRead(byte[] e)...
    public spik_tcp_connection(string ip, int port)...
    private void clientCommunicationWrite()...
    public void clientCommunication()...
    public void Close()...
    ...
    private void doCommand(string cmd)...
}
```

Obr. 21 Náhled třídy spik_tcp_connection používané v klientské aplikaci (zkráceno).

Tato třída je používána naslouchajícím vláknem a vláknem pro zasílání dat klientovi, přičemž tato vlákna mezi sebou sdílejí jednu instanci této třídy. Jde o obdobu třídy *spik_clientWork* serverové aplikace.

Třída zahrnuje vytvoření vnitřních objektů zastupujících třídy *spik_memory* a *spik_xmlfile*. (obr.21)

Hlavními metodami této třídy jsou metody *spik_tcp_connection*, která vytváří spojení se serverem, které bude spravováno v rámci této třídy, a třída *clientCommunication*, která zajišťuje práci naslouchacího vlákna a zároveň při svém prvním spuštění volá zmiňované vlákno pro odesílání dat, kterému předává metodu *clientCommunicationWrite*.

Metody *dataBlockWrite* a *dataBlockRead* převádějí daný blok dat, která budeme odesílat nebo jsme právě přijali, z bajtové do textové a z textové do bajtové podoby.

Pokud jsou příchozí data v pořádku, jsou naslouchacím vláknem předána v textové podobě metodě *doCommand*. Ta v případě rozpoznání odpovídající hlavičky provede odpovídající akci, která většinou zahrnuje zavolání odpovídající metody této třídy s parametry dle obdržených dat. Takto volané metody často potřebují v souvislosti s přijatými daty vykreslit odpovídající změnu v grafice aplikace, popřípadě zobrazit či skrýt některý formulář. To je zde řešeno voláním odpovídajících metod třídy hlavního formuláře s úplným přeposláním parametrů, výběru parametrů nebo parametrů určenými místní metodou. U některých hlaviček metody *doCommand* jsou metody třídy hlavního formuláře volány rovnou.

Umístění metod provádějících změnu ve vykreslení jednotlivých formulářů je prováděno z důvodu možné kolize vláken. Kolize je v případě těchto metod ošetřena použitím delegátů v rámci třídy hlavního formuláře.

3.3.6 Třída applicationLogoutClass

```
public class applicationLogoutClass
{
    // funkce pro odhlaseni pri ukonceni aplikace ( ukoncime i dalsi vlakna )
    public void applicationLogout(object sender, EventArgs e)...
}
```

Obr. 22 Náhled třídy applicationLogoutClass v klientské aplikaci.

Tato třída zahrnuje pouze jednu metodu, která provede korektní odhlášení ze serveru a která je přidána handleru. Ten ji spustí v případě ukončování aplikace klienta. Třída je zahrnuta z důvodu možnosti ukončení aplikace jiným nežli korektním způsobem.

3.3.7 Třída Main

```
static void Main()
{
    // pripravne operace
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    // nacteni dat
    spik_memory loadMemory = new spik_memory();
    // inicializace aplikace
    loadMemory.MainFormWork(new mainForm());
    loadMemory.MainFormWork().Width = 219;
    loadMemory.MainFormWork().Height = 449;
    // nastaveni handleru pro odhlaseni pri ukonceni aplikace ( ukoncime i dalsi vlakna )
    applicationLogoutClass alc = new applicationLogoutClass();
    Application.ApplicationExit += new EventHandler(alc.applicationLogout);
    // spusteni hlavniho okna aplikace
    Application.Run(loadMemory.MainFormWork());
}
```

Obr. 23 Náhled třídy Main v klientské aplikaci.

Tato třída je hlavní třídou klientské aplikace a je spouštěna při spuštění aplikace. Po svém spuštění provede nastavení a uložení objektu hlavního formuláře do „paměťové oblasti“. Poté je samotné aplikaci pro událost jejího ukončení přidán handler, který volá metodu pro korektní odhlášení třídy *applicationLogoutClass* (obr.22), a na závěr je vykreslen hlavní formulář.

3.3.8 Třída hlavního formuláře

```
public partial class mainForm : Form
{
    // delegati :
    public delegate void callErrorCallback(int e, string text);
    public delegate void workModeUpdateCallback(string e);
    ...
    public delegate void newRoom_savedCallback(string data);
    // dalsi form objekty :
    aboutForm aboutWin = new aboutForm();
    findUserForm uFindForm = new findUserForm();
    roomForm newRoomForm = new roomForm();
    // nacteni dat :
    spik_memory loadMemory = new spik_memory();
    spik_xmlfile loadCfg = new spik_xmlfile("spik_client.xml");
    spik_error loadError = new spik_error();
    // promene :
    IPAddress temporary;
    int temporary2;
    // hlavni metoda :
    public mainForm()
    {
        InitializeComponent();
        // dokonceni inicializace
        this.backgroundWork();
    }
    // metody s delegaty :
    public void ChangeIconOfUser(PictureBox e, string status)...
    public void changeModule(ArrayList e)...
    ...
    public void newRoom_saved(string data)...
    // metoda pro spravu objektu messageForm
    public ArrayList returnOrCreateLineForMsgFormInMTable(string type, string name)...
    // metoda provadena po inicializaci
    private void backgroundWork()...
    // metoda pro zapocati komunikace se serverem
    private void connectionToServerAndLogin()...
    // metoda pro korektni odhlaseni
    private void clientLogout()...
    // metody pro windowForm eventy :
    private void oProgramuToolStripMenuItem_Click(object sender, EventArgs e)...
    ...
    private void createRoomToolStripMenuItem_Click(object sender, EventArgs e)...
}
```

Obr. 24 Náhled třídy hlavního formuláře klientské aplikace zpracovávající jeho události (zkráceno).

Tato třída kromě metod pro své události zahrnuje i metody volané jinými třídami, které slouží pro překreslování samotných formulářů. Tyto metody, které jsou rozšířeny o delegáty, ošetřují možné kolize vláken, protože objekt hlavního formuláře vytváří, vykresluje a zpracovává hlavní vlákno. Tyto aplikace lze nalézt pod komentářem „metody s delegáty“ (obr.24) a zahrnují kontrolu, je-li vlákno volající metodu majitelem hlavního formuláře (hlavní vlákno). Pokud ne, samotnou metodu neprovede, tuto metodu předá prostřednictvím jejího delegáta vláknu, které vlastní hlavní formulář(hlavní vlákno) s odpovídajícími vstupními parametry (obr.25).

```
// delegát :
public delegate void delegatProMetodu(int e);
// metoda s delegátem :
public void Metoda(int e)
{
    if (this.InvokeRequired) // patřím pod aktuální vlákno ?
    {
        // vytvoření objektu typu [objekt typu delegát]
        delegatProMetodu alt = new delegatProMetodu(Metoda);
        // předáme objekt vláknu vlastnícího třídu spolu s parametry
        this.Invoke(alt, new object[] { e });
    }
    else
    {
       // vlastní kód metody
    }
}
```

Obr. 25 Příklad použití metody s delegátem.

Samotný delegát je v prostření jazyka C# ukazatelem na metodu třídy.

V rámci třídy hlavního formuláře jsou vytvořeny objekty pro informační formulář, vyhledávací formulář, formulář pro novou místnost, třídu zpracovávající chybová hlášení, třídu pro přístup do „paměťové oblasti“ a třídu pro práci s XML soubory. Dále je třída rozšířena o metodu pracující s objekty formulářů probíhající komunikace mezi uživateli a objekty diskuze, jenž jsou ukládány v paměťové oblasti. Samotná inicializace formuláře je rozšířena o metodu, která nám nastaví výchozí nastavení formuláře a v případě nalezení konfiguračního XML souboru provede předvyplnění naposledy použitých údajů, jmenovitě login, adresa serveru, port.

K manipulaci s formuláři probíhající komunikace mezi uživateli nebo v rámci místností slouží kromě jednotlivých metod odpovídajících události i metoda *returnOrCreateLineForMsqFormInMTable*, která vrací ukazatel na řádek dat odpovídajícího formuláře v paměťové tabulce. Pokud je daný formulář aktivní, provede se pouze samotné navrácení ukazatele. Pokud formulář již není aktivní (v paměťové oblasti není uložen jeho objekt, ale pouze data zahrnující proběhlou komunikaci) je vytvořen nový objekt, do kterého jsou předána uložená data proběhlé komunikace. Na místo těchto dat je uložen samotný objekt a je vrácen ukazatel na odpovídající řádek.

Samotné navázání spojení se serverem a pokus o přihlášení či registraci je zde řešen pomocí metody *connectionToServerAndLogin*, která provede spuštění vláken pro správu komunikace. Přidělí jim objekt třídy *spik_tcp_connection* (obr.21) a provede nastavení bool proměnné, na základě které proběhne pokus o přihlášení nebo registraci s automatickým přihlášením.

3.3.9 Informační formulář

Tento formulář slouží pouze k zobrazení několika informací o klientské aplikaci.

3.3.10 Třída formuláře pro komunikaci mezi uživateli

```
public partial class messageForm : Form
{
    private delegate void AddLineOfTextCallback(string data);

    public void AddLineOftext(string data)...

    public void Scroll()...

    public void chageIconOfUser(object sender, EventArgs e)...

    public messageForm()
    {
        InitializeComponent();
        this.Click += new System.EventHandler(this.chageIconOfUser);
    }

    private void messageForm_Load(object sender, EventArgs e)...

    private void msgSendButton_Click(object sender, EventArgs e)...

    private void button1_Click(object sender, EventArgs e)...
}
```

Obr. 26 Náhled třídy formuláře komunikace mezi uživateli zpracovávající jeho události.

V rámci této třídy jsou zahrnuty metody pro externí manipulaci s objektem formuláře a metody pro zpracování jeho událostí.

Metoda *AddLineOfText* zajišťuje rozšíření zobrazovaného textu o svou vstupní hodnotu a automatické oddělení informace odřádkováním. Metoda *Scroll* zajišťuje správné odrolování zobrazovaného textu tak, aby byl vidět poslední příspěvek. Metoda *changeIconOfUser*, která je zde přidělena události kliknutí na formulář, slouží k překreslení ikonky do výchozího stavu. Ikonka mohla indikovat nepřečtenou zprávu.

Metody přidělené k událostem kliknutí na jednotlivá tlačítka formuláře zajišťují odeslání vlastní zprávy druhému uživateli a přidání případně neznámého uživatele do svého seznamu známých uživatelů.

3.3.11 Rozšíření třídy vykreslení pro komunikaci mezi uživateli

```
spik_memory loadMemory = new spik_memory();
string type;
int userID;

public void myType(string e)...
public void myUserId(int e)...
public int myUserId()...
```

Obr. 27 Kód rozšiřující třídu pro vykreslení formuláře pro komunikaci mezi uživateli.

```
// zmena ikonky
this.loadMemory.MessageFormsWork("U", ((ArrayList)this.loadMemory
   ↳.ConnectionUserListValue()[0])
   ↳[this.myUserId()].ToString())[4] = true;
this.loadMemory.MainFormWork().ChangeIconOfUser(this.loadMemory
   ↳.ConnectionUserListPicture(this.userID), ((ArrayList)this.loadMemory
   ↳.ConnectionUserListValue()[1])[this.userID].ToString());


// zapis dat pred jejich ztratou

string[] existingText = this.dataText.Lines;
ArrayList load = loadMemory.MessageFormsWork(type, this.Text);
load[3] = existingText;
load[5] = true;
load[2] = false;
```

Obr. 28 Kód rozšiřující metodu Dispose třídy pro vykreslení formuláře pro komunikaci mezi uživateli.

Třída, která se stará o vykreslení formuláře a jeho prvků, je na svém začátku rozšířena o objekt zprostředkovávající přístup k „paměťové oblasti“ a metody pro nastavení/čtení dvou vnitřních proměnných (obr.27). Dále je rozšířena metoda *Dispose* této třídy, která je volána v případě ukončení formuláře, aby po něm „uklidila“, o kód zajišťující uložení obsažených dat zpráv do „paměťové oblasti“ (odkud budou případně načítány) a korekci zobrazované ikonky v seznamu uživatelů do výchozího stavu (obr.28).

3.3.12 Třída formuláře vyhledávání

```
public partial class findUserForm : Form
{
    public bool livingStatus; // true - user, false - room
    spik_memory loadMemory = new spik_memory();

    public findUserForm()...

    private void findUserGet_Click(object sender, EventArgs e)...

    private void findUserAdd_Click(object sender, EventArgs e)...
}
```

Obr. 29 Náhled třídy formuláře vyhledávání zpracovávající jeho události.

Třída zajišťuje část obsluhy formuláře pro vyhledávání uživatelů/místností v rámci již registrovaných/vytvořených. Bool proměnná *livingStatus* určuje, zda-li se jedná o vyhledávání místností nebo uživatelů a objekt *loadMemory* zde slouží pro zprostředkování manipulace s „paměťovou oblastí“. Metoda *findUserGet_Click* zasílá požadavek na vyhledání serveru, přičemž samotný návrat od serveru je předán hlavnímu formuláři a vláknu, jenž ho vlastní (hlavní). Ten ho předá objektu tohoto formuláře ve formě hodnot pro seznam výsledků. Metoda *findUserAdd_Click* zajišťuje zapsání vybraných výsledků do našeho seznamu známých uživatelů.

3.3.13 Třída formuláře komunikace v diskuzi

```
public partial class roomForm : Form
{
    private delegate void AddLineOfTextCallback(string data);

    public void AddLineOftext(string data)...

    public void Scroll()...

    public void setForm()...
    public void setModule(bool[] e, int[] l)...

    public roomForm()...

    private void sendToRoomButton_Click(object sender, EventArgs e)...

    private void button1_Click(object sender, EventArgs e)...

    private void button2_Click(object sender, EventArgs e)...
}
```

Obr. 30 Náhled třídy formuláře komunikace v diskuzi zpracovávající jeho události.

Tato třída je podobná třídě pro obsluhu událostí u formuláře pro komunikaci mezi klienty (obr.26), takže zahrnuje také svou obdobu metod *AddLineOfText* a *Scroll*.

Objekt tohoto formuláře zahrnuje čtyři možná vykreslení a to pro zobrazení komunikace v rámci místnosti a uživatelů v místnosti, pro vytvoření nové místnosti, pro dotázání se na heslo, kterým může být místnost chráněna, a pro stav zpracovávání nějaké další události (např. ukládání místnosti). O zobrazení korektního vykreslení odpovídajícího situaci se starají metody *setForm* a *setModule*. Dále jsou zde zahrnuty metody pro obsluhu jednotlivých tlačítek těchto vykreslení (odeslání hesla, uložení místnosti, odeslání vlastního příspěvku).

Data zaslaná serverem a určená pro objekt tohoto formuláře jsou předávána v odpovídající formě prostřednictvím metod hlavního formuláře, metody jsou ošetřeny delegátem (s objektem manipuluje vlákno, které ho vlastní). Stejným způsobem jsou vyvolávány změny vykreslení, které nejsou pokryty vlastními metodami.

3.3.14 Rozšíření třídy formuláře komunikace v diskuzi

```
spik_memory loadMemory = new spik_memory();
string type;
int userID;
bool createF = false;

public void createForm(bool e)...
public void myType(string e)...
public void myUserId(int e)...
public int myUserId()...
```

Obr. 31 Kód rozšiřující třídu pro vykreslení formuláře pro komunikaci v diskuzi.

```
if (this.createF == false)
{
    // zapis dat pred jejich ztratou :
    ArrayList load = loadMemory.MessageFormsWork(this.type, this.Text);
    if (load.Count > 6)
    {
        string[] existingText = this.roomText.Lines;
        ArrayList listOfUsers = new ArrayList();
        foreach (object oneUser in this.listBox1.Items)
            listOfUsers.Add(oneUser.ToString());
        load[3] = new ArrayList { existingText, listOfUsers };
        load[2] = false;
        load[5] = true;
    }
}
```

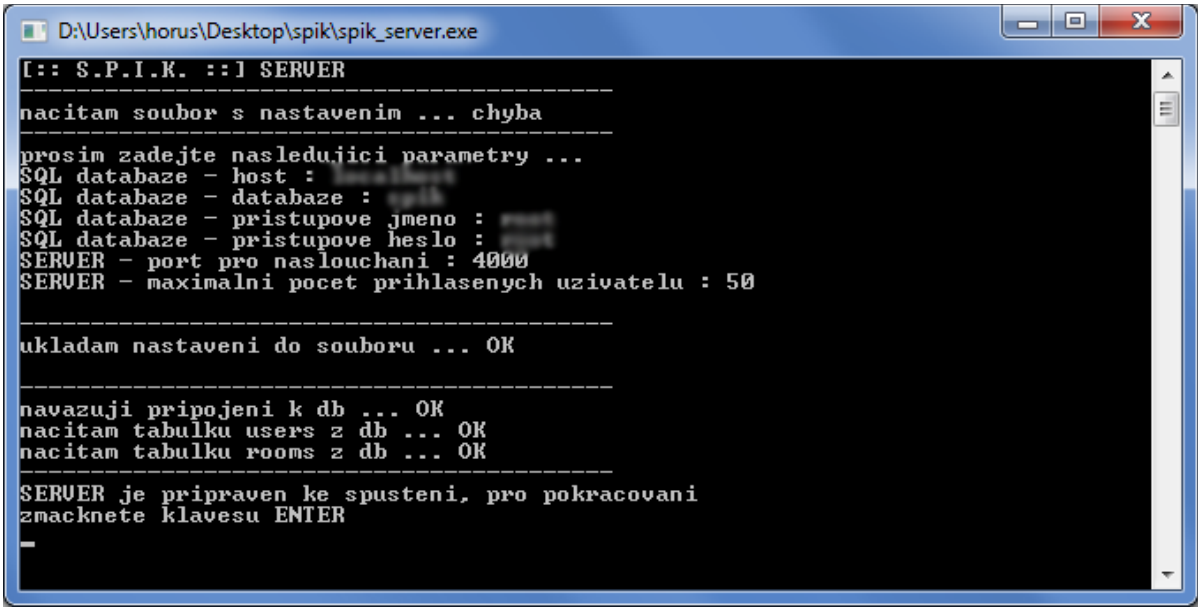
Obr. 32 Kód rozšiřující metodu Dispose třídy pro vykreslení formuláře pro komunikaci v diskuzi.

Třída, která se stará o vykreslení formuláře a jeho prvků, je na svém začátku rozšířena o objekt zprostředkovávající přístup k „paměťové oblasti“ a metody pro nastavení/čtení tří vnitřních proměnných (obr.31). Z těchto proměnných je nejdůležitější proměnná *createF*, podle které se rozhoduje, zda-li jde o objekt komunikace nebo jen objekt spravující vytváření nových místností. Dále je rozšířena metoda *Dispose* této třídy, která je volána v případě ukončení formuláře, aby po něm „uklidila“. Rozšíření zajišťuje, jde-li o objekt komunikace, uložení obsažených dat zpráv a seznamu uživatelů v místnosti do „paměťové oblasti“ (odkud budou případně načítány) a korekci zobrazované ikonky v seznamu uživatelů do výchozího stavu (obr.32).

4 OVLÁDÁNÍ APLIKACE

Zde budou uvedeny náhledy na jednotlivé stavy aplikací a jejich stručný popis. V případě serverové aplikace půjde sice pouze o jeden stav, protože jde o konzolozou aplikaci, ale v případě klientské aplikace jich bude více.

4.1 Serverová aplikace

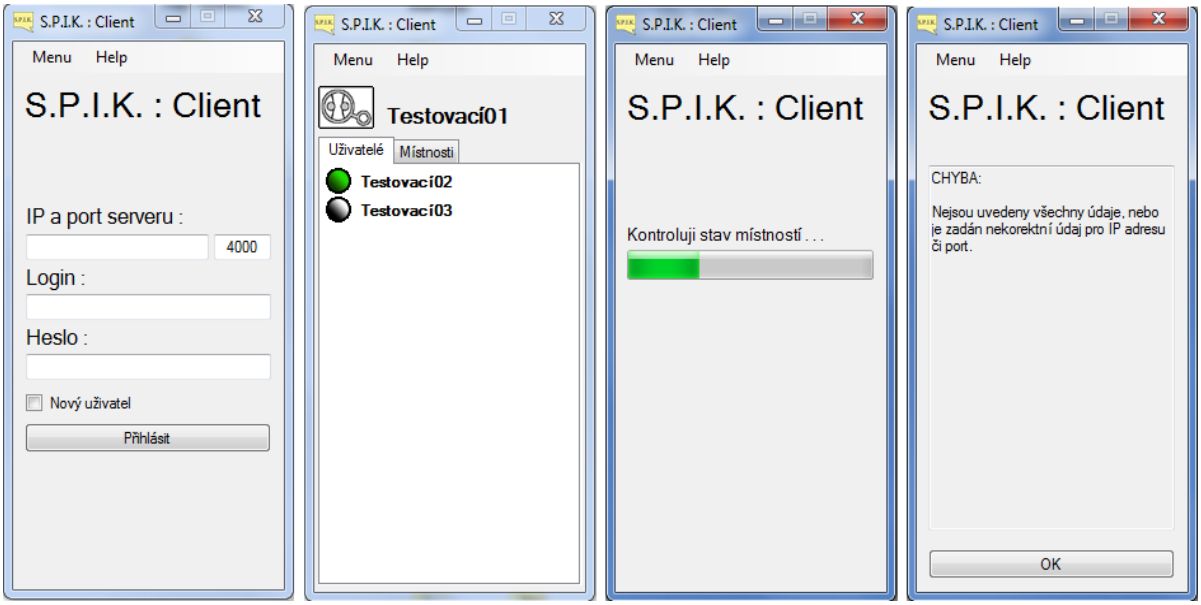


Obr. 33 Náhled na hotovou serverovou aplikaci.

Serverová aplikace je, jak již bylo zmíněno, řešena jako aplikace konzolová. Na náhledu (obr.33) můžeme vidět případ prvního spuštění nebo spuštění v případě odstraněného konfiguračního XML souboru, při kterém jsme dotázáni také na přístupové údaje k databázi. Do této databáze budeme ukládat údaje registrovaných uživatelů a místností. Server předpokládá, že odpovídající tabulky jsou v databázi již vytvořeny. Z tohoto důvodu je k aplikaci přiložen SQL soubor obsahující kód pro vytvoření těchto tabulek.

V aktuální verzi rozeznává aplikace sedm různých příkazů. Prvním je příkaz *help* který zajistí vypsání podporovaných příkazů na výstup konzole. Další jsou příkazy *onum* a *omax* pro zobrazení počtu přihlášených uživatelů a nastaveného maxima uživatelů na výstup konzole. Příkaz *clear* smaže údaje vypsané na výstupu konzole a vypíše hlavičku. Příkaz *echo on* zapne vypisování příchozích a odchozích dat aplikace, přičemž klienti jsou zde označováni unikátní přidělenou číselnou hodnotou začínající od nuly. Pro vypnutí zobrazování příchozích a odchozích dat serveru je příkaz *echo off* . Poslední příkaz *exit* provede korektní ukončení aplikace. Jedná se o zastavení naslouchání, zaslání informace o ukončování všem přihlášeným klientům a jejich následné odpojení, a ukončení samotné aplikace.

4.2 Klientská aplikace



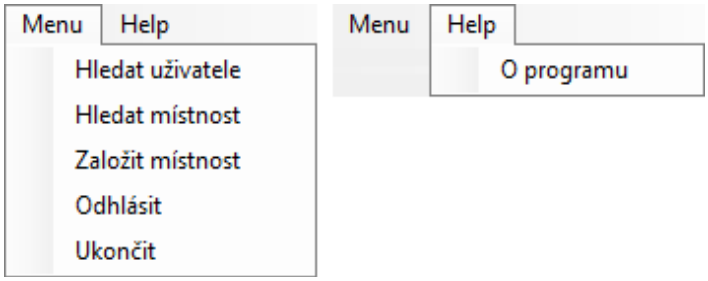
Obr. 34 Náhled na hotovou klientskou aplikaci – hlavní okno (formulář). Z leva : nepřihlášené zobrazení, přihlášené zobrazení, pracující zobrazení, zobrazení chybové hlášky.

Klientská aplikace je řešena jako grafická aplikace. V základním zobrazení se nám nabízí nabídka ve vrchní liště aplikace, z které můžeme spouštět různé činnosti (například odhlášení). Samotné vykreslení okna nastává ve čtyřech variantách pro odpovídající situace (obr.34).

V případě, že dojde k některé ze zachytávaných chyb v rámci klientské aplikace, je zobrazena informace o nastalé chybě. Pomocí přítomného tlačítka se vrátíme na situaci odpovídající vykreslení. Mezi chybové hlášky je zařazena i informace o automatickém odpojení od serveru, odpojení vykonává serverová aplikace při svém ukončování.

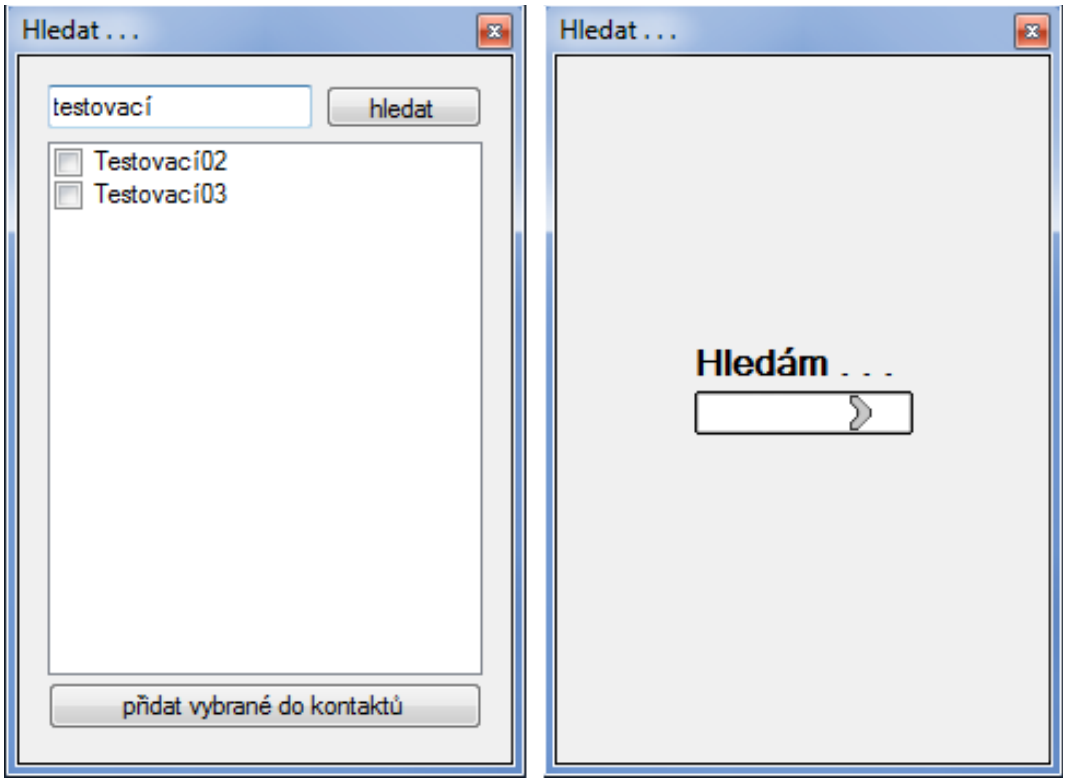
Vykreslení pro znázornění průběhu probíhající složitější operace je zde využíváno pro přihlašování a ukládání nových uživatelů/místností do seznamu.

Vykreslení pro přihlášený stav zahrnuje seznam uložených známých uživatelů a místností ve formě dvou záložek. Uživatelé mohou mít před jménem čtyři různé ikony. Zelenou ikonku pro uživatele přihlášeného v systému, černou pro odhlášeného, ikonku zprávy pro indikaci nepřečtené zprávy od uživatele a černý kroužek pro uživatele, kterého neznáme (již nám během tohoto přihlášení napsal a my jsme zprávu přečetli). Místnosti mohou mít tři ikonky, a to zamčený nebo odemčený zámek, podle toho zda je nebo není místnost chráněna heslem, a ikonku nepřečtené zprávy. Při kliknutí na jméno známého uživatele nebo místnosti se otevře odpovídající okno (obr.39), které bude pro tuto komunikaci přiděleno. Dále zde vidíme v hlavičce vypsáno naše uživatelské jméno.



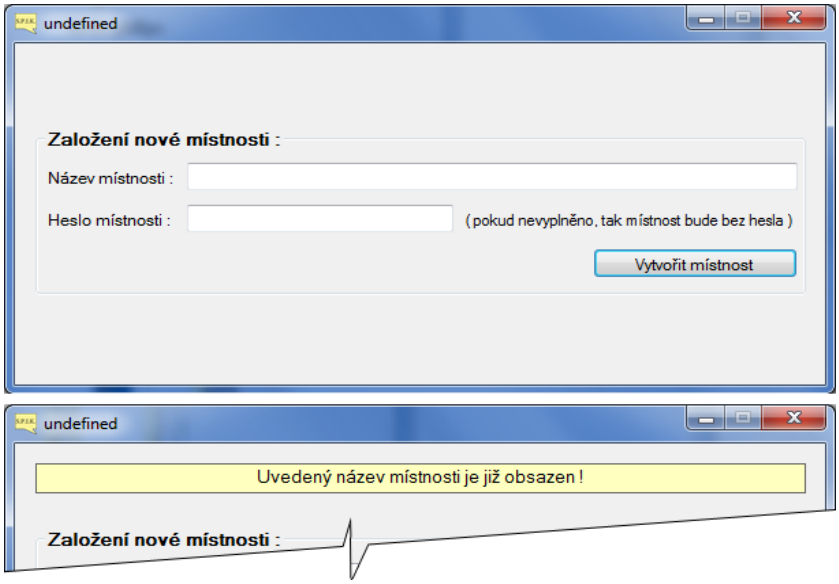
Obr. 35 Náhled na rozbalené záložky nabídky hlavního okna (formuláře)

Samotná nabídka, která je součástí všech vykreslení hlavního okna, je řešena formou dvou záložek. Některé položky nabídky (například: Odhlásit) se podle situace zamykají nebo odemykají. Tato modifikace vychází z předpokladů, že nepřihlášený uživatel se nemůže odhlásit a podobně.



Obr. 36 Náhled na hotovou klientskou aplikaci – vyhledávací okno (formulář).

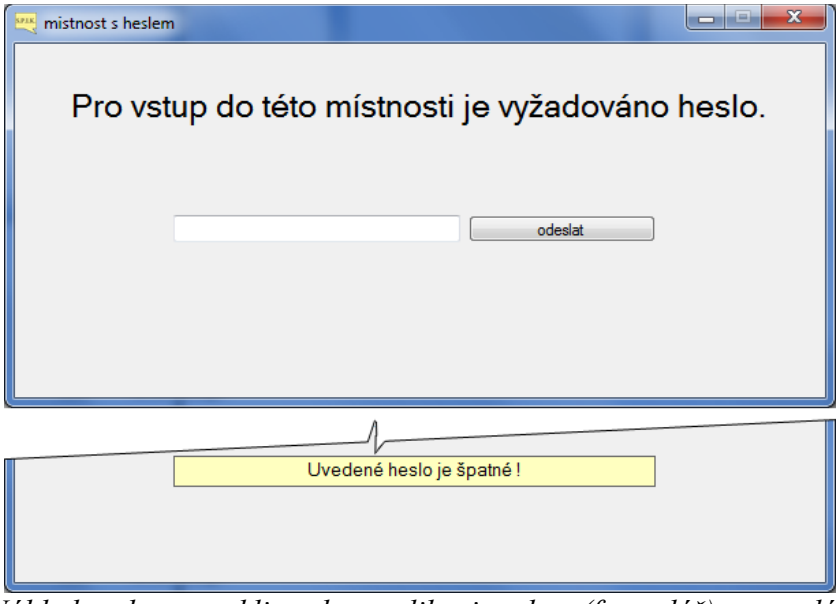
Z nabídky (obr.35) si můžeme po přihlášení otevřít samostatné okno pro vyhledávání (obr.36) uživatelů nebo místností, které chceme přidat do našeho seznamu. V obou případech se jedná o stejné okno, pouze samotná aplikace si vnitřně nastaví, o které vyhledávání se jedná, aby zaslala dotaz serveru s odpovídající hlavičkou. Po odeslání dotazu můžeme z výsledků vybrat ty, o které máme zájem, a přidat je do našeho seznamu. V případě, že vybraný uživatel/místnost se již nachází v našem seznamu, se v jeho případě nic neprovede. V opačném případě by nám vznikaly duplicitní záznamy v našem seznamu.



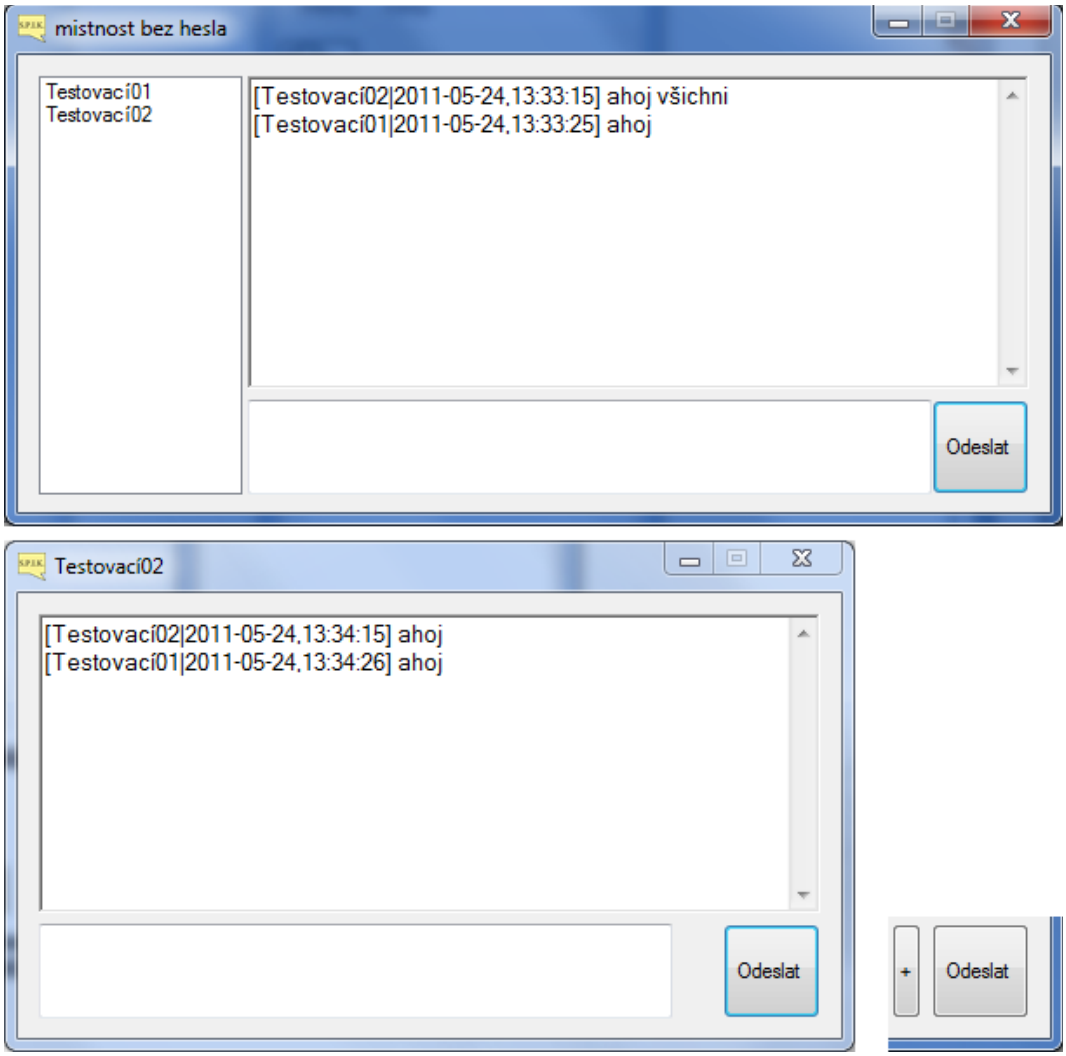
Obr. 37 Náhled na hotovou klientskou aplikaci – okno (formulář) pro založení místnosti.

Při výběru položky „Založit místnost“ z nabídky (obr.35) se nám zobrazí okno pro zadání názvu nové místnosti a nastavení hesla pro její uzamčení. Pokud chceme ponechat místnost odemčenou, tedy bez hesla, tuto hodnotu nevyplňujeme. V případě, že vybereme již obsazené jméno místnosti, jsme na tento fakt upozorněni. Pokud vytvoření místnosti proběhne úspěšně, jsme automaticky přihlášeni do této místnosti.

V případě, že vstupujeme do místnosti, kterou jsme vytvořili, není po nás požadováno heslo. Běžně se do místnosti vstupuje kliknutím na její název v našem seznamu. Uživatel po vstupu do místnosti zůstává v místnosti přítomen až do svého odhlášení a pokud má při svém přihlášení v seznamu nějaké odemčené (bez hesla) místnosti, je do nich automaticky přihlášen. Pokud vstupujeme do místnosti zamčené (s heslem), která není naše, jsme požádáni o zadání vstupního hesla (obr.38). V případě nesprávného hesla jsme upozorněni, že neodpovídá (obr.38).



Obr. 38 Náhled na hotovou klientskou aplikaci – okno (formulář) pro zadání hesla k místnosti.



Obr. 39 Náhled na hotovou klientskou aplikaci – okna (formuláře) pro správu komunikace. Shora : okno pro komunikaci v rámci místnosti, okno pro komunikaci uživatel-uživatel + náhled na „+“ tlačítko.

V případě komunikace v rámci místnosti vidíme v levé části okna zobrazen seznam online uživatelů přihlášených k místnosti. V případě komunikace uživatel-uživatel může být zobrazeno i „+“ tlačítko. Toto tlačítko je zobrazeno pro uživatele, kterého nemáme v seznamu. Toto tlačítko umožňuje přidat tohoto uživatele do našeho seznamu známých uživatelů.

5 ZÁVĚR

Závěrem můžeme shrnout, že klientská a serverová aplikace byly vyvíjeny pod operačním systémem Windows7. Tím se splnila podmínka cílového prostředí, které mělo být z řady Windows NT. Část zadání požadující možnost jak přímé komunikace uživatel-uživatel, tak komunikaci v rámci diskuzí byla rozšířena o možnost diskuze v uzavřeném okruhu uživatelů pomocí možnosti uzamčení místnosti heslem. Aplikace tedy splňuje zadání a navíc ho lehce rozšiřuje.

Protože byla aplikace vyvíjena jako výuková pomůcka pro cvičení z předmětu Počítačové sítě, největší část textu byla věnována popisu důležitých částí zdrojového kódu. Celý zdrojový kód včetně komentářů je přiložen na CD.

Data, která si klientská a serverová aplikace ukládají buď do jednotlivých XML souborů nebo do samotné databáze, nejsou nijak šifrována. Stejný fakt platí i pro data posílaná mezi serverem a klientem, data jsou pouze převáděna prostřednictvím tabulky UTF8 do bajtové podoby a zpět. (Zasílá se bajtová podoba.) Z tohoto vyplývá, že je tuto aplikaci, co se týče budoucího možného užití, možno rozšířit o užití nějakého kódování za účelem zvýšení bezpečnosti samotné aplikace.

Také je možno rozšířit aplikaci o přímou práci s užívanou databází. Serverová aplikace se k databázi připojuje pouze při svém spouštění a v případě registrace nového uživatele či místnosti. Pokud by aplikace pracovala vždy přímo s databází (i v situacích, ve kterých využívá uloženou kopii tabulek), tak by se nám naskytlo několik nových možností pro tuto aplikaci. Například provozování více serverů užívajících jednu databázi, přičemž by se změna projevila u všech serverů a ne jen aktivních až po provedené změně. A také by bylo možno provést registraci i jiným nástrojem, nežli skrze připojeného klienta, bez nutnosti restartovat server. Přímá práce s databází také umožňuje rychlejší přístup k požadovaným údajům. S narůstajícím počtem uložených záznamů (uživatelů a místností) bude tento rozdíl v rychlosti narůstat.

SEZNAM POUŽITÉ LITERATURY

[1] Microsoft : Online nápověda pro Microsoft Visual C# 2010 [online], 2011 [cit. 02.05.2011], <http://msdn.microsoft.com/query/dev10.query?appId=Dev10IDEF1&l=EN-US&k=k%28MSDNSTART%29>

[2] Petr Puš : Poznáváme C# a Microsoft .NET [online], 2004-2006, 20.9.2006 [cit. 27.03.2011], <http://www.zive.cz/Autori/sc-44/default.aspx?author=405>

[3] Wikipedie : Google Talk [online], 2011, 5. 3. 2011 [cit. 27.03.2011], <http://cs.wikipedia.org/wiki/Google_Talk>

[4] Wikipedie : Skype [online], 2011, 13. 5. 2011 [cit. 16.4.2011], <http://cs.wikipedia.org/wiki/Skype>

[5] Wikipedie : Windows Live Messenger [online], 2009, 5. 3. 2011 [cit. 27.3.2011], <http://cs.wikipedia.org/wiki/MSN_Messenger>

[6] Programy pro internetovou komunikaci [online], 2006, 12.8.2006 [cit. 27.03.2011], <http://www.pc-help.cz/viewtopic.php?f=85&t=6891>

[7] MUDr. Zbyněk Mlčoch : Co je to ICQ, historie ICQ, zajímavosti a kuriozity ze světa ICQ [online], [cit. 27.3.2011], <http://www.zbynekmlcoch.cz/info/internet/co_je_to_icq_historie_icq_zajimavosti_a_kuriozity_ze_sveta_icq.html>

[8] Wikipedie : ICQ [online], 2010, 21.4.2011 [cit. 22. 4. 2011], <http://cs.wikipedia.org/wiki/Icq>

[9] Komunikace po internetu (Instant Messaging) [online], 2007, 4.11.2007 [cit. 27.3.2011], <http://www.novymalin.net/clanky/komunikace-po-internetu-instant-messaging.html>

[10] Trillian Instant Messenger [online], 2011 [cit. 27.3.2011], <http://www.trillian.cz/trillian>

[11] About Pidgin [online], [cit. 27.3.2011], <http://www.pidgin.im/about/>