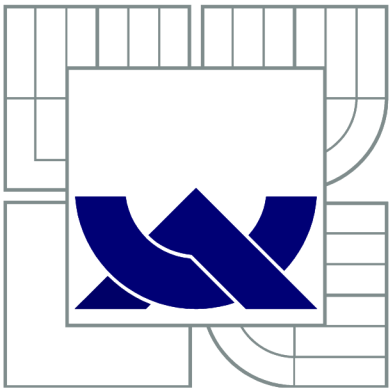
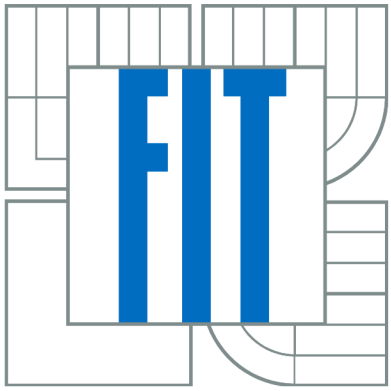


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KLIENT XMPP V NOKIA QT SDK

XMPP CLIENT USING NOKIA QT SDK

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ VAŇÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. DAVID BAŘINA

Brno 2011

Abstrakt

Tato bakalářská práce popisuje tvorbu aplikace ve vývojovém prostředí Nokia Qt SDK. Zdůrazňuje odlišnosti od vývoje aplikace pro stolní počítač a popisuje jednotlivé kroky potřebné k vytvoření funkční mobilní aplikace. V práci jsou dále popsány základy protokolu XMPP. Výsledná aplikace bude jednoduchý XMPP klient.

Abstract

This thesis describes the creation of application in a development environment Qt Nokia SDK. It emphasizes the differences between development of desktop applications and describes the steps necessary to create a functional mobile applications. This thesis also describes the basics of XMPP protocol. The resulting application is simple XMPP client.

Klíčová slova

Qt, Nokia Qt SDK, Symbian, mobilní aplikace, mobilní zařízení, XMPP, Jabber

Keywords

Qt, Nokia Qt SDK , Symbian, mobile application, mobile devices, XMPP, Jabber

Citace

Tomáš Vaňák: Klient XMPP v Nokia Qt SDK, bakalářská práce, Brno, FIT VUT v Brně, 2011

KLIENT XMPP V NOKIA QT SDK

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Vaňák
16. května 2011

Poděkování

Děkuji tímto Ing. Davidu Bařinovi za jeho podporu a odbornou pomoc při tvorbě této práce.

Obsah

1 Úvod

Mobilní telefony se už dávno nepoužívají jenom k telefonování a posílání SMS zpráv. Stále častěji se z nich stávají multifunkční zařízení sloužící k nejrůznějším činnostem. Jedním z posledních „hitů“ ve světě mobilních zařízení se staly operační systémy poskytující možnost tvorby vlastních aplikací, které největším podílem přispívají na možnostech chytrého telefonu. Díky tomu mobilní telefony pomalu přebraly funkce jiných zařízení, jako jsou MP3 přehrávače, fotoaparáty, GPS a mnohé další, a nahradily je.

Operačních systémů pro mobilní telefony je velké množství. Nastává však problém, když chceme vytvořit aplikaci, která bude použitelná pro více telefonů – více operačních systémů. Často je potřeba vytvořit dva různé programy.

Cílem této práce je vytvořit aplikaci pro mobilní telefon pomocí nástroje Nokia Qt SDK a hlavně popsat tento vývoj. Osobně jsem nově vlastníkem mobilního telefonu s operačním systémem Symbian. V této práci se budu snažit především upozornit, na co si musí vývojář dát při tvorbě aplikací pro tento operační systém pozor. Stejně jako mnoho dalších lidí i já jsem propadl kouzlu moderní komunikace – instant messagingu (IM). Mnohokrát jsem přemýšlel nad tím, jak taková komunikace funguje „uvnitř“. Tato práce je dobrou příležitostí pro nahlédnutí IM klientům „pod sukni“. Výsledná aplikace proto bude klient pro komunikaci pomocí XMPP – jednoho z moderních a stále častěji využívaných (nejen) instant messagingových protokolů.

2 Teorie

2.1 Symbian

Symbian je otevřený operační systém a softwarová platforma navržená pro „chytré“ mobilní telefony a udržovaná Nokií. Platforma Symbian je nástupce Symbian OS a Nokia Series 60. Na rozdíl od Symbian OS, který vyžaduje dodatečný systém uživatelského rozhraní, Symbian obsahuje komponenty uživatelského rozhraní založené na S60 5th Edition. Poslední verze – Symbian^3 – byla oficiálně uvolněna ve 4. čtvrtletí roku 2010 a poprvé použita v Nokii N8.
Symbian OS byl vyvinut společností Symbian Ltd. Je to následovník systému EPOC používaného v kapesních počítačích Psion.

2.1.1 Historie

Platforma Symbian vznikla sloučením a integrací softwarových aktiv Nokie, NTT DoCoMo, Sony Ericsson a Symbian Ltd., včetně komponent Symbian OS a jeho jádra, platformy S60 a částí uživatelského rozhraní UIQ a MOAP(S).

Vývoj Symbianu měla mít na starosti nadace Symbian Foundation, která byla poprvé oznámena v červnu roku 2008 a její činnost byla oficiálně zahájena v dubnu roku 2009. Jejím cílem bylo zveřejnit zdrojový kód pro celou platformu Symbian pod licencí EPL[1]. To se stalo 4. února 2010. Nicméně některé důležité složky v rámci Symbian OS byly licencovány třetími stranami, které bránily nadaci zveřejnění pod EPL.
V listopadu 2010 oznámila Symbian foundation, že vzhledem k nedostatku podpory ze strany členů, se přemění pouze na licenční organizaci. Nokia oznámila, že převezme správu nad platformou Symbian. Symbian Foundation zůstane jako držitel obchodní známky a nebude mít žádnou rozhodovací funkci.
Prodej zařízení založených na Symbianu představoval za 3. čtvrtletí roku 2010 34,4 %, což je zhruba 26,5 miliónů kusů[2].

2.1.2 Vývoj aplikací

Od roku 2010 Symbian přešel k využívání C++/Qt jako SDK[3], které může být použito s Qt Creatorem a nebo s Carbidem[4]. Qt podporuje starší Symbian S60 3rd a 5th vydání stejně jako nové platformy Symbianu. Podporuje také Windows, Linux, Mac OS X, vestavěné systémy, Maemo a do budoucna je plánována podpora i pro MeeGo.
Aplikace pro Symbian lze vyvíjet i s pomocí Pythonu, Adobe Flash nebo Java ME.

Symbian OS dříve využíval specifickou verzi C++ pro Symbian spolu s Carbide.c++ IDE jakožto nativní prostředí pro vývoj aplikací.

---

[1] **E**clipse **P**ublic **L**icense je otevřená softwarová licence, kterou používá pro svůj software Eclipse Foundation;
[2] http://mobil.idnes.cz/mob_tech.asp?c=A101025_173734_mob_tech_lhc
[3] Software Development Kit – sada nástrojů pro vývoj softwaru;
[4] Dříve doporučovaný nástroj pro vývoj aplikací pro Symbian;

2.2 Instant messaging

Instant messaging je internetová služba umožňující svým uživatelům (obvykle dvěma, ale některé služby umožňují i více) komunikovat spolu v reálném čase. Jako rozšíření téměř všechny tyto služby umožňují i sledovat, kteří uživatelé jsou právě připojeni, zasílat soubory, případně podporují i jiný druh komunikace. Na rozdíl od e-mailu má instant messaging výhodu v tom, že uživatel vidí, jestli je druhý účastník k dispozici či nikoliv a reakce může být proto okamžitá. Uživatele však nikdo nenutí reagovat na zprávy hned, díky čemuž je tento druh komunikace méně vyrušující než třeba telefonní hovor. IM je ideální pro rychlou výměnu internetových adres, kusů zdrojového kódu a dalších věcí, které se například telefonicky špatně předávají.

3 Nokia Qt SDK

Jedná se o vývojové prostředí zaměřené na tvorbu aplikací pro operační systémy, které do svých mobilních zařízení nasadí Nokia. V současné době jsou podporovány Symbian a Maemo. Do budoucna má být podporováno i MeeGo případně i další operační systémy, které Nokia vyvine. Při této práci jsem se zaměřil na vývoj aplikace pro operační systém Symbian, proto v dalším textu budu mluvit převážně o této platformě.

První vydaná verze vyšla v červnu 2010. Poslední stabilní verzí je Nokia Qt SDK 1.1 (pro Windows, Linux i Mac OS-X) vydaná 4. května 2011. Ta zahrnuje:

- Qt Creator 2.1
- Qt Simulator 1.1
- knihovny pro Qt (verze 4.7.3) a Qt Mobility (verze 1.1.3)
- balíčky pro Symbian
- Remote Compiler, který dovoluje překládat projekty pro všechny podporované platformy
- dokumentaci pro všechny komponenty
- sjednocuje vývojová prostředí pro desktopové a mobilní aplikace v jednom SDK
- první verze, která umí vytvořené aplikace zveřejňovat v Ovi Store

3.1 Qt

Qt je jedna z nejpopulárnějších multiplatformních knihoven pro tvorbu aplikací s grafickým uživatelským rozhraním, ale současně se dá stejně tak použít i pro vývoj konzolových aplikací. Pomocí Qt je možné psát aplikace pro stolní počítače, mobilní a vestavěné operační systémy a to bez nutnosti přepsat zdrojový kód. Je to knihovna programovacího jazyka C++. Zároveň ale existuje i pro jiné jazyky. Jedná se obvykle o navázání[5] (binding) daného jazyka na knihovnu Qt (například PyQt[6], QtAda[7]). Pro jiné jazyky ale nejsou implementovány všechny moduly, kterými Qt disponuje.

Qt toolkit byl vytvořen společností Trolltech v roce 1999. V roce 2008 jej koupila finská Nokia se záměrem mít vývojové prostředí pro Symbian, Maemo, a další platformy, kde člověk napíše jednu Qt aplikaci na mobil, případně její desktopovou verzi a jenom překladem by se daly „spojit“ různé platformy. Je dostupný jako komerční varianta, nebo pod otevřenou licencí LGPL[8]. Obě verze se prakticky neliší. Jde v nich vytvářet naprosto rovnocenné programy, ale v případě licence LGPL nemůže vývojář požadovat za výsledný produkt žádný finanční obnos a nemá k dispozici odbornou technickou podporu.

[5] http://qt.nokia.com/products/programming-language-support/
[6] http://www.riverbankcomputing.co.uk/software/pyqt/intro
[7] http://www.qtada.com/en/index.html
[8] GNU **L**esser **G**eneral **P**ublic **L**icense je licence svobodného softwaru, publikovaná Free Software Foundation;

Qt si může každý stáhnout z internetových stránek Nokie[9]. K dispozici jsou jak potřebné knihovny, tak vývojové prostředí nazvané Qt Creator[10]. Jeho součástí je editor zdrojového kódu, designer pro grafické uživatelské rozhraní, debugger a systém pro správu verzí. Současná verze, která vyšla 4. května 2011[11], je Qt 4.7.3.

3.1.1 Prodej komerčního Qt

11. února 2011 Nokia s Microsoftem oznámili společnou spolupráci. Hlavním důvodem uzavřeného partnerství má být vytvoření protiváhy operačnímu systému Android od Google. Nokia bude mít možnost nasazovat ve svých mobilních telefonech operační systém Windows Phone 7 a získá přístup ke službám Microsoftu (vyhledávač Bing). Microsoft bude moci využívat například bezplatnou navigaci Nokie.

Tím ale vyvstala otázka, co se stane s Qt, jakožto Nokií nejvíce podporovaným toolkitem pro vývoj mobilních aplikací? Nokia prohlásila[12], že Qt nebude portován na Windows Phone 7. Měl by být ale i nadále vyvíjen, protože i do budoucna bude Nokia prodávat nové telefony (nižší kategorie) s operačním systémem Symbian. Navíc by se měl letos objevit i první telefon s operačním systémem MeeGo, který je založený na Qt. Navíc Nokia je jen jednou z mnoha set firem, které Qt využívají. Na Qt běží třeba Skype, Google Earth, vizualizační nástroj VirtualBox nebo linuxové uživatelské prostředí KDE.

7. března 2011 se však Nokia dohodla na prodeji komerční licence Qt a služeb pro podniky finské firmě Digia. Prodej byl dokončen 22. března 2011. Nokia bude i nadále investovat do vývoje Qt jak pro LGPL tak pro komerční licenci. Digia má již 7 let zkušeností s Qt a je rozhodnuta pokračovat v rozšiřování komerční licence stejně jako nekomerční komunity.

3.1.2 QObject

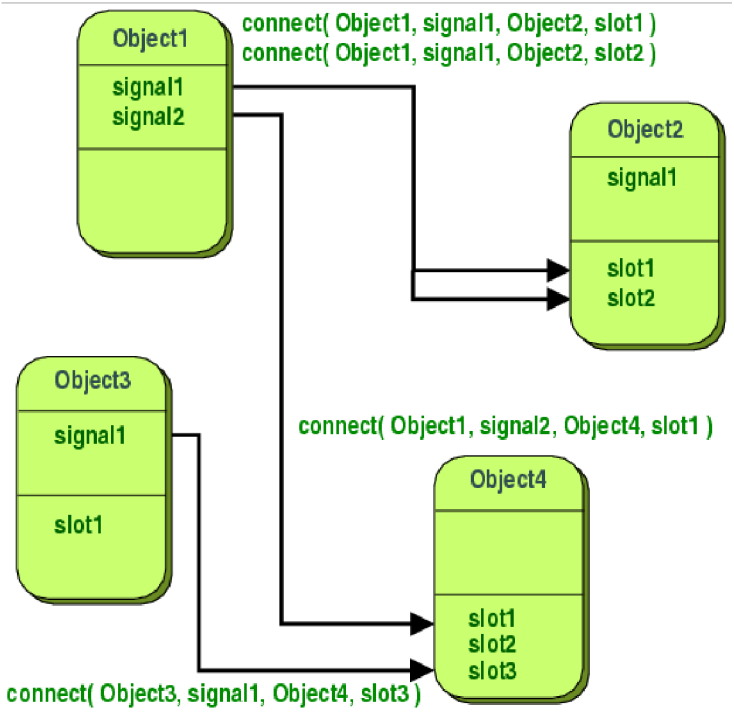
Mezi nejdůležitější třídy toolkitu Qt patří QObject. Je to základní třída pro všechny objekty Qt. Bez něj není možná komunikace mezi objekty pomocí signálů a slotů. Objekty Qt jsou organizované do stromů. Když se vytvoří nový objekt, který dědí QObject, nastaví si rodiče objektu. Zároveň se v rodiči zavolá metoda *insertChild()*, díky čemuž se nový objekt objeví v rodiči v seznamu jeho potomků (zpřístupněný metodou *childrens()*). To má za následek například to, že když se zavolá destruktor rodiče, tak jsou zároveň destruovány i všichni jeho potomci.
Zároveň se u každého objektu Qt můžeme zeptat na jméno – *name()*, na jméno třídy – *className()*, nebo třeba z jaké třídy vychází – *inherits()*.
Všechny objekty, které dědí QObject mohou také přijímat a filtrovat události, které jsou hlavním prostředkem komunikace mezi programem a uživatelem.

Každá třída, která dědí QObject, by měla mít zároveň definované makro Q_OBJECT. Bez něj nebude možné využívat signály a sloty a další prostředky Qt.

[9] http://qt.nokia.com/downloads
[10] http://qt.nokia.com/products/developer-tools/
[11] http://labs.qt.nokia.com/2011/05/04/qt-4-7-3-and-qt-mobility-1-1-3-have-been-released/
[12] http://www.itwriting.com/blog/3872-qt-will-not-be-ported-to-windows-phone-7-says-nokia.html

3.1.3 Signály a sloty

Pomocí signálů a slotů je možné realizovat komunikaci mezi objekty. Je to základní princip většiny toolkitů pro uživatelské rozhraní – někdo vyšle nějaký signál, který si někdo jiný odchytí a zareaguje na něj. Jsou typově bezpečné (na rozdíl například od callbacků v GTK). Signál je takzvaně emitován (klíčové slovo emit, případně makro Q_EMIT). Slot je metoda, která reaguje na vyslaný signál. Signál je potřeba připojit na daný slot. To dělá metoda *QObject::connect(&objekt1, SIGNAL(muj_signal(int)), &objekt2, SLOT(muj_slot(int)));*
Jeden objekt může emitovat/přijímat několik signálů, jeden signál může být připojen na více slotů. Záleží na fantazii a hlavně potřebách programátora.



Obrázek 3.1 Signály a sloty v Qt

3.1.4 MOC

Meta-**O**bject **c**ompiler, nazývaný *„moc“*, je nástroj podobný preprocesoru. Běží nad zdrojovým kódem Qt programu, interpretuje jistá makra z C++ kódu (která jsou speciální pro Qt, například Q_OBJECT) a generuje z nich dodatečný C++ kód s „meta informacemi“ o třídách využívaných v programu, který by se jinak velmi obtížně tvořil. Tyto informace využívá Qt k dosažení funkčnosti, která není k dispozici v klasickém C++ (signály a sloty, asynchronní volání funkcí).

3.1.5 Moduly

Qt toolkit je rozdělen na několik modulů [7]. Každý modul, který chceme použít, musíme uvést v projektovém souboru do proměnné Qt.

```
QT = core gui multimedia network opengl openvg script scripttools sql \
       svg webkit xml xmlpatterns declarative phonon qt3support
```

Mezi základní patří:

QtCore – zahrnuje základní funkcionalitu pro programy bez GUI. Definuje spoustu základních tříd, na kterých staví další třídy. Například:

- QString – práce s textovými řetězci
- QHash – poskytuje asociativní pole, v některých jazycích známé jako „slovník“
- QList – ukládá seznam hodnot a poskytuje k nim rychlý přístup na základě indexu
- QObject (kapitola 3.1.2)

QtGui – poskytuje většinu tříd grafického uživatelského rozhraní. Například:

- QMainWindow – vykresluje hlavní okno aplikace, včetně menu apod.
- QWidget – základní třída pro všechny objekty GUI v Qt
- různá tlačítka, scroll bary, menu,….

QtMultimedia – nízkoúrovňové programování multimediálních aplikací. Poskytuje rozhraní pro přijímání/vysílání audio/video záznamů z různých vstupních a výstupních zařízení

QtNetwork – poskytuje třídy pro programování síťových aplikací. Například:

- QTcpSocket – umožňuje TCP komunikaci
- QSslSocket – rozšiřuje možnosti QTcpSocket o ustanovení zabezpečeného TCP spojení umožňující šifrovaný přenos dat
- QHostAddress – zapouzdřuje práci s IP adresou
- QTcpServer – umožňuje implementaci TCP serveru, který naslouchá na určitém portu
- QHttp – umožňuje přímo stahovat/odesílat data v HTTP formátu

QtOpenGL – umožňuje jednoduché použití OpenGL[13] v Qt aplikacích. Poskytuje OpenGL widget třídu, která může být použita jako jakýkoliv jiný Qt widget, ve kterém ale bude umožněno použít OpenGL API.

QtOpenVG – poskytuje podporu pro vykreslování OpenVG[14]

QtScript – třídy, které umožňují vytvářet základní skriptovatelné Qt aplikace; skript použitý v tomto modulu je do značné míry podobný JavaScriptu, ovšem s několika rozšířeními, jako jsou signály a sloty. QtScript využívá JavaScriptCore z WebKitu.

[13] Průmyslový standard specifikující multiplatformní rozhraní (API) pro tvorbu aplikací počítačové grafiky;
[14] Standardní API pro akcelerovanou 2D vektorovou grafiku;

QtScriptTools – dodatečné komponenty pro skriptovatelné aplikace. Poskytuje jedinou třídu QScriptEngineDebugger. Ta umožňuje do aplikací, které využívají QtScrip, vložit debugger. Díky tomu si uživatel bude moci debuggovat svůj skript.

QtSql – třídy integrující otevřené i uzavřené databáze; zahrnuje upravitelné datové modely pro databázové tabulky, které mohou být použity s GUI třídami

QtSvg – podpora pro zobrazení obsahu SVG[15] souborů. Podobně jako u QtOpenGL je zde k dispozici QSvgWidget, který se dá použít stejně jako jakýkoliv jiný widget. Do něj lze už načíst SVG soubor a pracovat s ním.

QtWebKit – poskytuje základ webového prohlížeče, stejně tak třídy pro práci s webovým obsahem. Je založený na otevřeném WebKitu[16]. Umí vykreslovat SVG, (X)HTML, aplikovat CSS styly a skriptovat v JavaScriptu.

QtXml – poskytuje nástroje pro načítání a zápis XML dokumentů a C++ implementace SAX[17] a DOM[18]. Mezi nejdůležitější třídy patří:

- QDomDocument – třída reprezentující celý XML dokument (kořen stromu dokumentu)
- QDomElement – třída reprezentující jeden element v DOM stromě
- QDomText – třída reprezentující textová data v elementu

QtXmlPatterns – podpora pro XPath[19], XQuery[20], XSLT[21] a validaci XML schémat

QtDeclarative – podpora pro psaní vlastního vysoce dynamického uživatelského rozhraní. Je zde využito deklarativního jazyka QML.

Phonon – multimediální knihovna umožňující aplikacím přehrávat audio a video

Qt3Support – usnadňuje přenos z Qt3 na Qt4

[15] **S**calable **V**ector **G**raphics je značkovací jazyk a formát souboru, který popisuje dvojrozměrnou vektorovou grafiku pomocí XML;
[16] http://www.webkit.org/
[17] Simple API for XML umožňuje sériový přístup k XML. Jde o tzv proudové zpracování, při kterém se dokument rozdělí na jednotlivé jeho části (počáteční/koncová značka, obsahy elementů, komentáře, atd.). Postupně se pak volají jednotlivé události, které ohlašují nalezení konkrétní části;
[18] Objektově orientovaná reprezentace XML nebo HTML dokumentu. DOM je API umožňující přístup či modifikaci obsahu, struktury, nebo stylu dokumentu, či jeho částí;
[19] Počítačový jazyk, pomocí kterého lze adresovat části XML dokumentu;
[20] Dotazovací a funkcionální programovací jazyk, který je navržen pro provádění dotazů nad XML;
[21] Slouží k převodům zdrojových dat ve formátu XML do libovolného jiného požadovaného formátu;

3.1.6 Struktura programu

Popíšeme si, jak může vypadat jednoduchý Qt GUI[22] program [2]:

```
1 #include <QApplication>
2 #include <QLabel>
3 int main(int argc, char *argv[])
4 {
5             QApplication app(argc, argv);
6             QLabel *label = new QLabel("Hello Qt!");
7             label->show();
8             return app.exec();
9 }
```

Na začátku programového kódu (stejně jako v klasickém C++) se nachází definice tříd (řádky 1 a 2), které budeme používat. Pro každou Qt třídu existuje hlavičkový soubor se stejným názvem (včetně velikostí písmen) jako třída, kterou definuje.
Po spuštění programu se zavolá funkce *main()*, které se předají parametry příkazového řádku (řádek 3).
Na začátku každého programu s grafickým uživatelským rozhraním je potřeba vytvořit objekt QApplication (řádek 5), který řídí tok programu s GUI. Konstruktoru QApplication se předávají parametry z příkazového řádku.
Následuje vytvoření (řádek 6) instance třídy hlavního okna aplikace. Ta může dále vytvářet nová okna, zobrazovat je, schovávat, atd. Pro vytvoření okna se v Qt používá třída QWidget a třídy z ní odvozené. Zde je pro jednoduchost vytvořen pouze widget[23] s textem.
Takto vytvořený objekt sice existuje, ale nebyl by na obrazovce vidět, proto ho zviditelníme metodou *show()* (řádek 7).
Na konci programu (řádek 8) se zavolá metoda *exec()*. Ta spustí smyčku událostí. Jedná se o jakýsi druh vyčkávacího režimu, kdy program vyčkává na akce uživatele (obvykle na kliknutí myší nebo stisknutí tlačítka na klávesnici) a to až do doby, dokud není uživatelem ukončen. Každá taková akce vygeneruje událost, na kterou může program reagovat. Tím se GUI programy drasticky liší od tradičních programů, které obvykle zpracují vstup, vygenerují nějaký výstup a skončí bez zásahu uživatele.

3.1.7 Lokalizace

Většina uživatelů je nejraději, když s nimi program komunikuje jejich rodným jazykem. Proto je dobré programy lokalizovat. Každý text, který chceme mít v Qt aplikaci lokalizovaný, musí mít řetězce viditelné pro uživatele „označeny“ pro překlad pomocí metod

*QString QObject::tr(const char *);* nebo *QString QObject::trUtf8(const char *);*

a všechny tyto texty musí být napsány v jednom určitém jazyce. Dále je potřeba přidat do projektového souboru názvy souborů s překladem

```
TRANSLATIONS = cs_CZ.ts de_DE.ts en_GB.ts
```

[22] **G**raphical **U**ser **I**nterface - grafické uživatelské rozhraní;
[23] Widgetem se myslí jakýkoliv objekt GUI;

Kde „cs“ je ISO 639 kód pro češtinu a „CZ“ je ISO 3166 kód pro Českou republiku. Tyto soubory s překladem nemusíme psát celé ručně.
K dispozici máme nástroj *lupdate*, kterému předáme jako parametr projektový soubor. Z něj si zjistí, jaké překlady budeme chtít dělat, projde všechny zdrojové soubory a vytvoří nám (nebo v případě příštího spuštění upraví stávající) pro každý požadovaný překlad soubor *.ts. Což je XML dokument, do kterého doplníme pro každý jazyk příslušný překlad. To můžeme udělat buď ručně, nebo použít nástroj Qt Linguist.
S těmito soubory ale neumí implicitně naše aplikace pracovat. Proto použijeme poslední utilitu – *lrelease*, které předáme opět parametrem název projektového souboru. Ta nám vytvoří pro každý *.ts soubor odpovídající *.qm soubor. To je binární soubor, se kterým už naše aplikace umí pracovat.

V aplikaci překlad aplikujeme tak, že vytvoříme instanci třídy QTranslator, její metodě *load(file.absoluteFilePath())* předáme jako parameter *.qm soubor s požadovaným překladem. Zavoláme metodu *installTranslator()* objektu QApplication/QCoreApplication a předáme jí jako parametr instanci našeho vytvořeného QTranslatoru. To způsobí vygenerování události LanguageChange, kterou bude QApplication propagovat do všech oken, ve kterých je reimplementována metoda *changeEvent(QEvent *e)*. V této metodě zavoláme metody *ui->retranslateUi(this)* a *retranslate()*, čímž je celý překlad hotov. *retranslateUi(this)* zajistí překlad tříd uživatelského rozraní generovaných Qt Designérem a *retranslate()* je metoda, kterou si musíme vytvořit, aby přeložila objekty GUI, které jsme si vytvořili ručně.
Nyní už pokaždé, když budeme chtít změnit jazyk, stačí zavolat metodu *installTranslator()* a předat jí parametrem příslušnou instanci třídy QTranslator.

3.2 Instalace

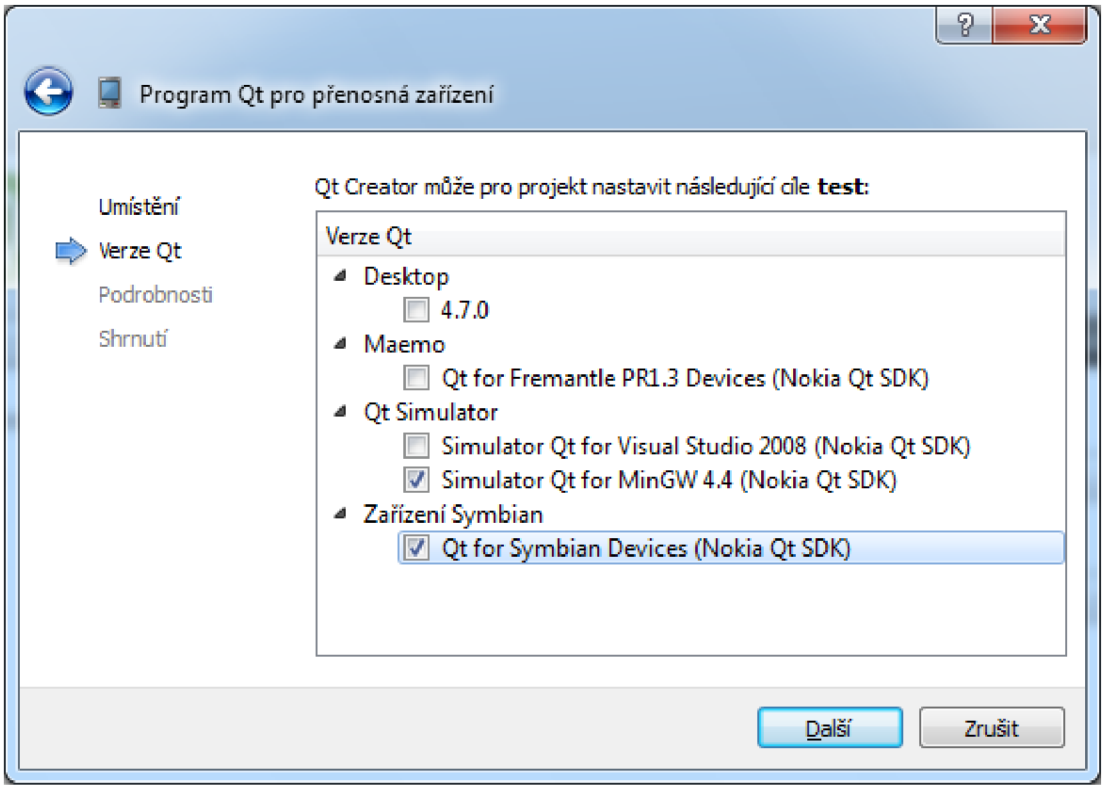
Nokia Qt SDK je volně ke stažení z internetových stránek[24] Nokie.

3.2.1 Instalace softwaru do počítače

Programátoři, kteří již vyvíjejí v Qt aplikace (pro desktop) a mají ho nainstalované v počítači, si musí dát pozor na kompatibilitu verzí. Nokia Qt SDK zahrnuje nový Qt Creator, který není se starou verzí kompatibilní. Po spuštění by se spustil nový Qt Creator, ve kterém by ale nebylo umožněno programovat pro desktop, pouze pro mobilní zařízení. Jestliže však verze souhlasí, je možné po instalaci jednoduše přepínat mezi vývojem pro stolní počítače a pro mobilní zařízení.

Poté, co je software v počítači nainstalovaný, si při založení projektu můžeme vybrat, pro jaké zařízení chceme programovat – pro stolní počítač, Symbian, Maemo a nebo chceme testovat v simulátoru (Obrázek 3.2).

[24] http://www.forum.nokia.com/info/sw.nokia.com/id/e920da1a-5b18-42df-82c3-907413e525fb/Nokia_Qt_SDK.html



Obrázek 3.2 Výběr cílů překladu

3.2.2 Instalace Qt do mobilního telefonu

Abychom mohli zkompilovat aplikaci pro mobilní telefon, musíme nainstalovat Qt do mobilního zařízení. K tomu je potřeba USB kabel pro propojení mobilního telefonu s počítačem a na počítači nainstalované Nokia Ovi Suite nebo Nokia PC Suite.

Připojíme mobilní zařízení pomocí USB kabelu k počítači. Při prvním připojení se nainstalují potřebné ovladače. Do zařízení je následně potřeba nainstalovat Qt. To se dělá pomocí souboru *qt_installer.sis*. Můžeme buď ručně zkopírovat soubor do zařízení a poté nainstalovat (najdeme ho v adresáři <NokiaQtSDK_install_path>\Symbian\sis\qt_installer.sis) nebo použít zástupce v nabídce start StartMenu >Nokia Qt SDK > Symbian > Install Qt to Symbian device a následovat instrukce, které uvidíme na obrazovce.

Obdobným způsobem nainstalujeme Qt Mobility, pokud je chceme využít.

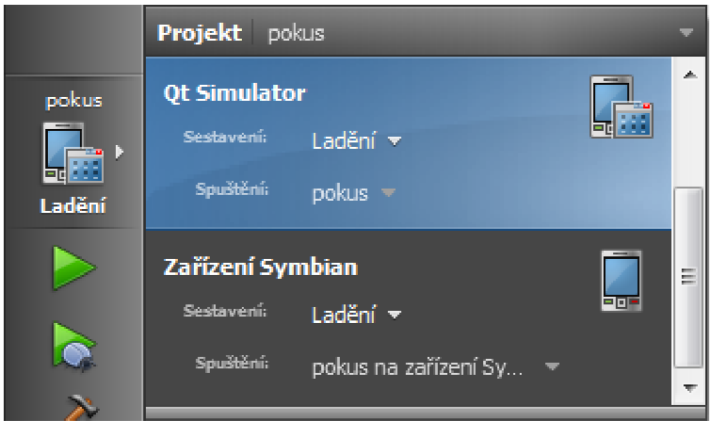
Nakonec je potřeba nainstalovat aplikaci TRK, pomocí které bude Qt Creator komunikovat s naším mobilním zařízením. Postup je stejný jako v předchozím případě, pouze musíme vybrat, jaký typ Symbianu vlastníme.

3.3 Připojení zařízení k počítači

Na začátku, než začneme psát nějakou aplikaci, musíme zajistit, aby Qt Creator komunikoval s mobilním zařízením, na kterém máme nainstalované Qt.

1) Spustíme Qt Creator.
2) Vytvoříme nový projekt nebo otevřeme již existující.
3) Vlevo dole vybereme jako možnost cílového zařízení (Obrázek 3.3) zařízení Symbian.
4) Mobilní zařízení s nainstalovaným Qt připojíme pomocí USB kabelu k počítači – ikona zařízení se změní z „nepřipojeno“ (Obrázek 3.4) na „připojeno“ (Obrázek 3.5).
5) Na zařízení spustíme aplikaci TRK.
6) V TRK stiskneme Volby > Nastavení a u možnosti Připojení zvolíme USB.
7) Vrátíme se do hlavní nabídky a stiskneme Volby > Připojit.

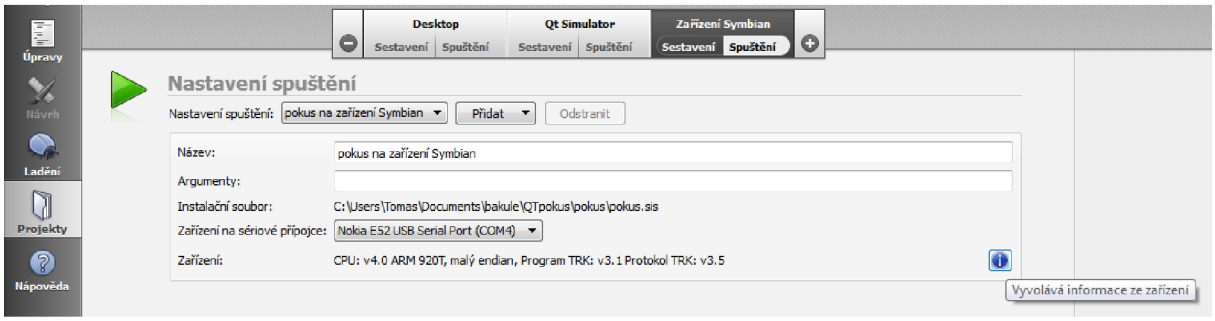
Nyní by měli být Qt Creator a mobilní zařízení správně propojeni. Ověříme si to tak, že klikneme na levém panelu na „Projekty“, nahoře vybereme „Zařízení Symbian“ a záložku „Spuštění“. Po kliknutí na modrou ikonku „Vyvolání informace ze zařízení” se zobrazí informace o připojeném mobilním zařízení (Obrázek 3.6). Jestliže jste provedli něco špatně, zobrazí se okno s informací, že je něco špatně (Obrázek 3.7).



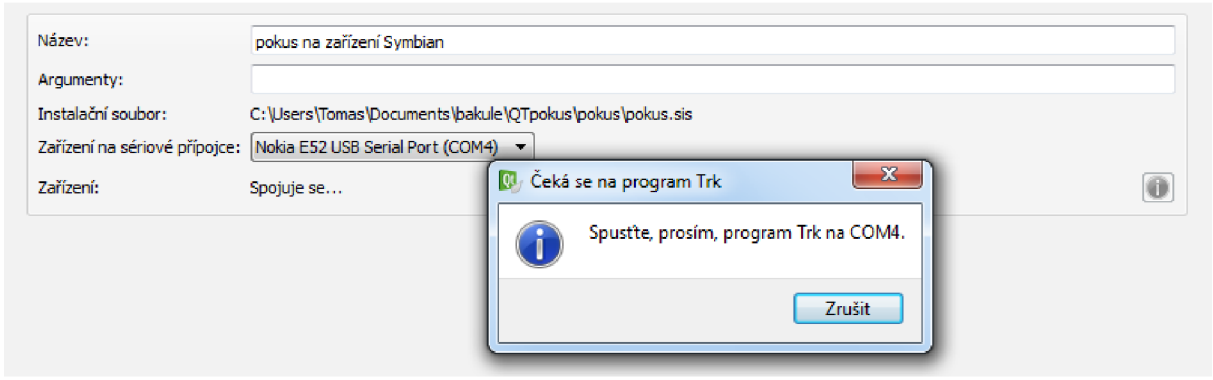
Obrázek 3.3 Výběr cílového zařízení

Obrázek 3.5 Připojené mobilní zařízení

Obrázek 3.4 Nepřipojené mobilní zařízení



Obrázek 3.6 Informace o správně připojeném mobilním zařízení



Obrázek 3.7 Hlášení o špatně připojeném mobilním zařízení

3.4 Tvorba aplikace

Samotné programování aplikací pro mobilní zařízení se příliš neliší od tvorby programů pro stolní počítače. Programátor si musí ale dát pozor na 2 věci:

a) Zobrazování oken
b) Projektový soubor

3.4.1 Zobrazování oken

Při psaní desktopové aplikace se okna obvykle zobrazují pomocí metody *show()*. Takto zobrazená okna by se mohla na mobilním zařízení se Symbianem zobrazit špatně. Symbian nezobrazuje nová okna automaticky maximalizované, proto by se mělo zobrazení provádět metodou *showMaximized()*, nejlépe pak při psaní multiplatformní aplikace použitím podmíněného překladu:

```
#ifdef Q_OS_SYMBIAN
    window->showMaximized();
#else
    window->show();
#endif
```

3.4.2 Projektový soubor

Každý Qt projekt je popsán v projektovém souboru[25] (*.pro). Informace obsažené v těchto souborech využívá qmake k vygenerování Makefile obsahujícího všechny příkazy potřebné k sestavení projektu. Projektový soubor obvykle obsahuje seznam zdrojových a hlavičkových souborů, obecné informace o konfiguraci, podrobnosti specifické pro aplikaci, například seznam externích knihoven nebo cest, které je třeba připojit k projektu.
Projektový soubor může obsahovat i další prvky, jako jsou komentáře, deklarace proměnných nebo vestavěné funkce. Ve většině jednoduchých projektů stačí deklarovat zdrojové a hlavičkové soubory, které mají být použity k sestavení projektu.

Proměnné jsou v projektovém souboru používány k uložení seznamu řetězců. V nejjednodušším projektu tyto proměnné informují qmake o konfiguraci překladu, nebo zahrnují názvy souborů a cesty potřebné k sestavení projektu.
Příklad proměnné se seznamem hlavičkových souborů:

```
HEADERS = mainwindow.h paintwidget.h
```

Pro přidání řetězce do již existujícího seznamu řetězců se používá operátor „+=“

```
CONFIG += qt
```

K přiřazení obsahu jedné proměnné do jiné se používají 2 znaky $$

```
TEMP_SOURCES = $$SOURCES
```

Komentář začíná znakem ‚#’ a pokračuje do konce řádku

Vestavěné funkce mohou usnadňovat práci s proměnnými a provádět s nimi různé operace. Jedna z nejužitečnějších vestavěných funkcí je *include(other.pro)*. Ta na místo, kde je vložena, vloží obsah souboru, který je jí dán parametrem.

Podmíněný překlad je k dispozici za pomoci složených závorek

```
win32
{
     SOURCES += paintwidget_win.cpp
}
unix
{
     SOURCES += paintwidget_unix.cpp
}
mac
{
     SOURCES += paintwidget_mac.cpp
}
```

Symbian a projektový soubor

Mezi specifické rysy této platformy patří zpracování statických dat, velikost zásobníku a haldy, specifické nastavení překladače a unikátní identifikátory pro aplikaci nebo knihovny.

```
symbian
{
     # Specifická nastavení pro Symbian
}
```

[25] http://www.civilnet.cn/book/embedded/GUI/Qt_assistant/qmake-manual.html

Zpracování statických dat
Pokud aplikace využívá jakákoliv statická data, operační systém o tom musí být informován. Je to proto, že Symbian se snaží šetřit paměť, když nejsou statická data používána. Abychom specifikovali, že budeme statická data využívat, musíme přidat do projektového souboru:

```
TARGET.EPOCALLOWDLLDATA = 1
```

Defaultní hodnota je 0

Velikost zásobníku a haldy
Platforma Symbian používá předdefinované velikosti pro zásobník a haldu. Pokud aplikace překročí jejich limit, může spadnout nebo nedokončí nějakou operaci. Pády aplikací, které nemají zdánlivě žádné příčiny, jsou obvykle způsobeny nedostatečným zásobníkem/haldou.
Velikost zásobníku má maximální hodnotu, zatímco velikost haldy má minimální i maximální hodnotu, všechno specifikováno v bytech. Minimální hodnota brání aplikaci ve spuštění, jestliže není k dispozici dostatek paměti. Minimální a maximální hodnoty jsou od sebe odděleny mezerou. Například:

```
TARGET.EPOCHEAPSIZE = 10000 10000000
TARGET.EPOCSTACKSIZE = 0x8000
```

Jedinečné identifikátory
Aplikace Symbianu mohu mít k sobě přiřazeny jedinečné identifikátory. Jsou podporovány 4 typy identifikátorů: `UID2`, `UID3`, `SID` a `VID`. Je možno je specifikovat takto:

```
TARGET.UID2 = 0x00000001
TARGET.UID3 = 0x00000002
TARGET.SID  = 0x00000003
TARGET.VID  = 0x00000004
```

Jestliže UID2 není nastaveno, jeho hodnota bude nastavena na stejnou hodnotu jako je UID3. Jestliže UID3 není nastaveno, qmake pro něj automaticky vygeneruje vhodné UID pro vývoj a ladění. Tato hodnota by měla být manuálně nastavena pro aplikace, které mají být vydány. Za účelem získání originálního UID je vhodné kontaktovat Nokii. Výchozí hodnoty pro SID a VID jsou prázdné hodnoty.

Capabilities[26]
Definují extra oprávnění pro aplikaci, jako například přístup ke všem souborům v souborovém systému:

```
TARGET.CAPABILITY += AllFiles
```

Je také možné specifikovat, jaká oprávnění aplikace mít nemůže. To se zapíše tak, že se nejdříve aplikaci dají všechna oprávnění a pak se některá odeberou, například když chceme, aby aplikace mohla všechno kromě přístupu k síťovým zařízením:

```
TARGET.CAPABILITY = ALL -NetworkServices
```

3.4.3 Certifikáty

Qt využívá informace v projektovém souboru aplikace k vytvoření instalačního balíčku v Symbianovském nativním formátu *.sis*. Výchozí instalační balíček je navržen pro instalaci během vývoje, ale pro finální verzi jej může být potřeba upravit. SIS instalační balíčky musí

[26] http://wiki.forum.nokia.com/index.php/Category:Capabilities

být před instalací digitálně podepsané[27]. V opačném případě je není možné nainstalovat na zařízení se Symbianem. Existuje několik typů podpisů.

Symbian Signed. Aplikace, které jsou takto podepsané, mohou být instalovány na zařízení bez zobrazení varovného hlášení na Symbian zařízeních. Mnoho distribučních kanálů (Handango[28], Nokia Ovi Store[29]) vyžaduje, aby aplikace zahrnovala Symbian Signed. Aplikace mohou být Symbian signed využitím buď *Express Signed* nebo *Certified Signed*.

Express Signed je pro aplikace využívající jen „uživatelská“ a „systémová“ oprávnění. Umožňuje podepsat aplikaci okamžitě a je k tomu potřeba jedno Content ID (placené).

Certified Signed je potřeba, pokud aplikace požaduje zvýšená oprávnění. Zahrnuje kompletní oprávnění pro komerční aplikace. Je ale finančně nákladnější, protože je potřeba nechat aplikaci nezávisle otestovat, jestli není „škodlivá“.

Self-signed je podpis vytvořený vývojářem. Qt Creator IDE poskytuje aplikacím implicitně tento digitální podpis. Je dostatečný, pokud aplikace využívá pouze takzvaná „uživatelská oprávnění“ (user capabilities). Mezi ně patří: `NetworkServices`, `LocalServices`, `Location`, `ReadUserData`, `WriteUserData`, a `UserEnvironment`. Při instalaci je uživatel zařízení upozorněn, že aplikace je „nedůvěryhodná“ a jsou mu vypsána oprávnění, která aplikace vyžaduje. Mnoho distribučních kanálů nepřímá self-signed aplikace (Ovi Store vyžaduje Symbian Signed).

Developer certificates (DevCerts) je vyžadováno pro aplikace, které během vývoje vyžadují vyšší oprávnění než „uživatelská oprávnění“. Aplikace podepsaná DevCerts může být používána pouze na jednom konkrétním zařízení (nebo na několika konkrétních zařízení), pro které byl certifikát vydán (pro určité EMEI[30]).
Pro koncového uživatele může být tento certifikát vhodný pro podepisování nepodepsaných aplikací.

3.5 Qt Mobility

Projekt Qt Mobility přináší řadu nových rozhraní API pro Qt s vlastnostmi, které jsou dobře známé ze světa mobilních zařízení, zejména telefonů. Tyto API umožní vývojářům používat tyto funkce jednoduše z jednoho frameworku a aplikovat je na mobilní telefony, notebooky i stolní počítače. Framework nejen zlepšuje používání těchto technologií, ale hlavně zajišťuje použitelnost ve světě mobilních zařízení. Pomocí něj se dá jednoduše přistupovat například k různým senzorům, GPS zařízení, seznamu kontaktů, práci se zprávami (SMS) apod.
V této práci nejsou Qt Mobility využity, pokud by však bylo potřeba je využít, musí se do projektového souboru přidat:

```
CONFIG += mobility
```

[27] http://doc.qt.nokia.com/qtcreator-snapshot/creator-deployment-symbian.html
[28] http://www.handango.com/homepage/Homepage.jsp?storeId=2218
[29] http://store.ovi.com/
[30] **I**nternational **M**obile **E**quipment **I**dentity – jedinečný patnáctimístný kód, který jednoznačně určuje jakýkoliv mobilní telefon. Zjistit se dá vytočením čísla *#06#;

a poté přidat do proměnné MOBILITY ty moduly, které budeme chtít využít, například

```
MOBILITY += systeminfo
```

3.6 Qt Simulátor

V případě, že jsme jako cíl překladu (Obrázek 3.3) zadali Qt Simulátor, po spuštění se nám zobrazí 2 okna (Obrázek 3.8). Jedno z nich vypadá jako mobilní telefon. Je možné si zvolit různé typy mobilního telefonu – mobilní telefon klasické konstrukce, dotykový, atd. Na displeji tohoto virtuálního telefonu se zobrazí spuštěná aplikace, kterou jsme právě přeložili a je možné s ní pracovat podobně, jako by byla ve skutečném telefonu.
Druhé okno slouží k nastavení parametrů zařízení, jako je například aktuální kapacita baterie, stav senzorů, atd.

Cílem Qt Simulátoru je usnadnit vývojáři mobilních aplikací práci. Díky němu není nutné mít neustále připojený mobilní telefon k počítači a vše testovat na něm. Většina funkčnosti aplikace se dá vyzkoušet v simulátoru a na mobilním zařízení stačí testovat jen občas.

V současné verzi není bohužel implementována podpora menu, které tak nelze v Simulátoru otestovat a je nutné použít reálné zařízení.

Obrázek 3.8 Qt Simulátor

4 XMPP

Extensible **M**essaging and **P**resence **P**rotocol, neboli „rozšiřitelný protokol pro posílání zpráv a zjišťování stavu“ je aplikační protokol pro komunikaci v reálném čase, využívající XML jako základní formát pro výměnu informací. XMPP definuje způsob, jak posílat malé kousky XML z jedné entity do jiné v takřka reálném čase. XMPP je používán řadou lidí, aniž o tom většina z nich ví. Například chatovací služby v Gmailu a Facebooku jsou realizovány pomocí tohoto protokolu. Uživatelé používající Jabber mohou pro aktualizaci Twitter použít i svého oblíbeného XMPP klienta[31].

4.1 Historie

V roce 1998 začal Jeremie Miller pracovat na otevřeném protokolu pro IM komunikaci. 4. ledna 1999 vytvořil server nazývaný *Jabberd*. Zanedlouho vytvořila komunita vývojářů otevřené klienty pro různé platformy. Během roku 1999 a počátkem roku 2000 se pracovalo na vytvoření protokolu, který dnes známe jako XMPP. Vyvrcholilo to v květnu 2000 vydáním *Jabberd 1.0*.
V srpnu 2001 bylo založeno Jabber Software Foundation[32]. Od té doby tato nezisková organizace zdokumentovala a zveřejnila protokoly používané uvnitř vývojářské komunity a definovala velké množství rozšíření k základním protokolům.
Po několika letech od vývoje se členové vývojářské komunity rozhodli formalizovat základní protokoly s IETF, která standardizovala většinu základních technologií používaných na internetu. V roce 2004 vznikla RFC 3920 [5] (obecná specifikace protokolu) a 3921 [6] (samotný instant messaging a zobrazení stavu).

4.2 XML

Extensible **M**arkup **L**anguage (XML), neboli „rozšiřitelný značkovací jazyk“ byl vyvinut a standardizován konsorciem W3C[33]. Je to jednoduchý textově založený formát pro reprezentaci a sdílení strukturovaných informací mezi programy, mezi lidmi nebo mezi programem a člověkem, a to jak lokálně, tak přes síť. Byl odvozen od staršího formátu SGML (ISO 8879), aby byl lépe využitelný pro webové použití.
Syntaxe je velmi podobná značkovacímu jazyku HTML. Nicméně pravidla XML jsou mnohem přísnější. XML nástroje nebudou zpracovávat soubory obsahující chyby (u HTML je to možné a děje se tak běžně). To znamená, že téměř všechny XML dokumenty mohou být spolehlivě zpracovány počítačovým softwarem.

4.2.1 Stručný popis

Základem každého XML dokumentu jsou elementy. Ty se skládají z počáteční značky, obsahu elementu a koncové značky. Značka neboli tag se zapisuje pomocí úhlových závorek < a > a názvu elementu. Koncový tag je stejný jako počáteční tag, ale za první úhlovou

[31] http://www.root.cz/zpravicky/twitter-pres-xmpp-Jabber/
[32] V roce 2007 přejmenována na XMPP Standards Foundation;
[33] http://www.w3.org

závorku se vloží znak lomítko. Příklad elementu: <ovoce> pomeranč </ovoce>. Může existovat i prázdný element, ten se zapisuje <ovoce></ovoce> a nebo zkráceně <ovoce/>.
Elementy se mohou [1] vkládat do sebe:

```
<adresa>
        <mesto> Brno </mesto>
        <ulice> Lidická </ulice>
<adresa/>
```

Každý XML dokument musí mít alespoň jeden element, ten se nazývá kořenový. XML je case sensitive (rozlišuje malá a velká písmena), takže <automobil>, <AUTOMOBIL> a <Automobil> jsou 3 odlišné elementy.

Elementy mohou mít atributy. Zapisují se do počátečních tagů, případně do značky prázdného elementu. Element <zelenina typ="salát"/> má jeden atribut. Jeho název je „*typ*" a hodnota je „*salát*". Hodnota atributu se od názvu atributu odděluje rovnítkem a musí být vždy v uvozovkách nebo apostrofech.

4.3 Principy komunikace

Každá dobrá internetová technologie má svoji architekturu neboli způsob, jakým spolu jednotlivé entity komunikují. Například Word Wide Web je tvořen milióny webových serverů, ke kterým se připojují klienti, nejčastěji internetové prohlížeče.

Přestože XMPP není spjat s žádnou síťovou architekturou, nejčastěji se používá model klient-server. Takže klienti se pomocí TCP spojení připojují na server. Na rozdíl od jiných IM služeb je XMPP decentralizovaná [3]. Existuje velké množství serverů, na které se připojují klienti a vyřizují jejich požadavky, posílají jejich zprávy.

Když odesíláte e-mail, Váš e-mailový klient se připojí k Vašemu „domácímu" e-mailovému serveru. Ten přepošle (v nejjednodušším případě) Vaši zprávu „domácímu" e-mailovému serveru adresáta. Až se osoba, které jste e-mail odeslali, připojí ke svému mail serveru, ten mu předá Váš e-mail.
Takto funguje i XMPP komunikace. Když například uživatel Julie@kapuletova.com bude chtít poslat zprávu uživateli Romeo@montek.com, Juliin IM klient odešle tuto zprávu na server kapuletova.com. Server kapuletova.com naváže spojení se serverem montek.com a odešle mu Juliinu zprávu. Server montek.com vyčká na připojení uživatele Romeo a odešle mu zprávu.

Již jsem zde použil formát uživatelského jména. Z historických důvodů se adresa entity v XMPP nazývá „Jabber Identifier" neboli JID. Jeho syntaxe je:
[název uzlu] @ *[doména]* / *[zdroj]* například tedy uživatel@host.cz/zdroj. Všechna JID jsou založena na předchozí struktuře. Nejčastěji se používá pro identifikaci uživatele IM, serveru, ke kterému se uživatel připojuje a uživatelova zdroje (například specifický klient). Nicméně jsou možné i jiné zdroje než klienti. Třeba u multiuživatelské chatovací místnosti může být adresa ve tvaru „*místnost@služba*", kde „místnost" je název chatovací místnosti, „služba" je hostname multiuživatelské chatovací služby. Účastník takovéto místnosti se poté adresuje jako *místnost@služba/nick*.

Každá část JID může mít maximální délku 1023 bytů. Celková délka JID může mít maximálně 3071 bytů (včetně oddělovačů).
Na rozdíl například od e-mailu má XMPP velikou výhodu v možnosti mít v názvu jakékoliv Unicodové znaky, tedy i znaky národních abeced. Tím pádem může být uživatelské jméno ve tvaru třeba Jiří@Český.cz. Nadšenec pro matematiku si může dát uživatelské jméno ∞@math.org, což je u e-mailových adres nemožné. Stejně jako u e-mailových adres se však ani u JID nerozlišují malá a velká písmena.

Poslední část JID (zdroj) je důležitá pro připojení více klientů. Představme si situaci, kdy bychom měli jen JID ve tvaru uživatel@host.cz. Pomocí nějakého IM klienta bychom se doma připojili k našemu serveru a vesele bychom chatovali. Pak bychom ale museli jít do práce a počítač (včetně IM klienta) bychom nechali zapnutý. V práci bychom ale v předchozím chatování chtěli pokračovat. Spustili bychom tedy IM klienta na našem pracovním počítači. Ten by se přihlásil k našemu serveru. Novým přihlášením obvykle staré spojení zaniká. Tím by se IM klient na našem domácím PC odpojil. Většina IM klientů je ovšem nastavena tak, že při ztrátě spojení se ho pokoušejí znovu navázat. Takže náš domácí klient se pokusí o spojení s naším serverem, podaří se mu to a nám se v práci přeruší spojení. Takto se budou oba klienti neustále přetahovat o spojení.
V XMPP je to řešeno tak, že se při přihlášení uživatel identifikuje jednak svým uživatelským jménem a heslem, ale také zdrojem a prioritou (ta se použije, jen pokud uživatel nezadá zdroj, a zprávy se tak budou odesílat klientovi s nejvyšší prioritou). Jelikož je každý klient nastaven jakožto jiný zdroj, může se uživatel připojit z několika míst současně.

Pro XMPP je definováno i vlastní URI: xmpp:Jabber.org nebo xmpp:user@Jabber.org.

Pro zabezpečení komunikace a ověřování je možné využít TLS[34] a SASL[35].

4.4 XML Stream a Stanza

Pro výměnu malých, strukturovaných užitečných informací mezi entitami jsou v XMPP definovány 2 termíny – XML Stream a XML Stanza [5, 6].

XML Stream je kontejner pro výměnu XML elementů mezi dvěma entitami přes síť. Hlavní myšlenkou je popsat veškerou komunikaci jedním XML dokumentem. Celá tato komunikace se pak nazývá rostoucí XML dokument. Začátek XML streamu je jednoznačně určen otevřením XML tagu <stream> (s příslušnými atributy) a konec ukončující značkou </stream>. Mezi nimi je veškerá komunikace.

XML Stanza je jedna zpráva zaslaná přes XML stream. Jedna stanza[36] odpovídá jednomu XML elementu. Máme 2 základní jmenné prostory „Jabber:client“ a „Jabber:server“ a pro každý z nich jsou definovány 3 elementy: <message/>, <presence/>, <iq/>. Pro všechny z nich je definováno 5 atributů:

[34] **T**ransport **L**ayer **S**ecurity je protokol poskytující zabezpečenou komunikaci. RFC 2595;
[35] **S**imple **A**uthentication and **S**ecurity **L**ayer je metoda pro přidávání nebo zlepšování ověřování v protokolech klient/server;
[36] Z důvodů neexistujícího adekvátního českého překladu budu používat pojem stanza;

to

- určuje příjemce zprávy
- zpráva, která má být doručena určitému příjemci, musí mít tento atribut
- zpráva, která je odeslána k přímému zpracování serverem (například změna stavu), nesmí mít tento atribut

from

- určuje odesílatele zprávy
- jestliže server příjme od klienta zprávu, musí k ní přidat (nebo přepsat existující) atribut *from* obsahující úplné JID (včetně zdroje)
- server nesmí odeslat klientovi stanzu bez atributu *from*

id

- používá se k číslování jednotlivých zpráv v delší komunikaci
- pro zprávy typu <message/> a <presence/> je doporučeno používat tento atribut
- pro zprávy typu <iq/> je používání tohoto atributu vyžadováno
- pokud generovaná stanza obsahuje atribut *id*, pak odpověď na ni, nebo error stanzu, musí obsahovat *id* atribut se stejnou hodnotou, jaká byla v generované zprávě

type

- blíže specifikuje, čeho se daná stanza týká

xml:lang

- pokud je zpráva určena pro uživatele, pak se v tomto atributu nachází jazyk zprávy

příklad:

```
<message from='julie@kapulet.com/Z_balkónu' to='romeo@montague.net'
xml:lang='cs'>
        <body>Zdaliž Ty nejsi Romeo Montek?</body>
</message>
```

Element <message> slouží k výměně zpráv mezi uživateli. Jestliže zpráva je odeslána mimo kontext nějakého existujícího chatu, nebo přijaté zprávy, měl by atribut *to* obsahovat raději JID bez zdrojové části (nick@doména).
Jestliže je zpráva odeslána jako odpověď na nějakou zprávu, kdy adresa odesílatele byla ve tvaru (nick@doména/zdroj), měl by atribut *to* (v odpovědi) obsahovat JID se zdrojovou částí, protože odesílatel ví, komu přesně zpráva patří.
Atribut *type* může nabývat 5 hodnot:

- chat – označuje zprávu, která byla odeslána v kontextu nějaké delší konverzace
- error – je generována, když nastane chyba
- groupchat – je odeslána v multiuživatelském chatu
- headline – je to zpráva informativního charakteru (sportovní informace, počasí,…) na kterou se neočekává odpověď
- normal – samostatná zpráva, odeslaná bez návaznosti na jakoukoliv předchozí komunikaci. Zpráva typu „normal“ je typicky zpráva začínající novou konverzaci.

Element <message> má 3 subelementy:

Element <*body*> je obvykle zahrnutý do elementu <message>, ale je volitelný. Obsahuje textovou zprávu, kterou chceme, aby si příjemce přečetl. Má jediný atribut – „xml:lang“. Je možné, aby těchto elementů bylo obsaženo v elementu <message> více, například pro multijazyčnou zprávu.

Element <*subjekt*> obsahuje jakýsi titulek zprávy. Opět má jediný volitelný atribut – „xml:lang“.

Element <*thread*> se používá k jednoznačné identifikaci konverzace. V běžné konverzaci se to dá nahradit použitím typu zprávy „normal“, „headline“, atd. U zpráv, jejichž příjemcem má být nějaký stroj, je lepší použít tento element (například ve hrách).
Jelikož je to jednoznačný identifikátor vlákna konverzace, nesmí být použit v jednom elementu <message> víc než jeden element <thread>. Může obsahovat atribut *parent*, identifikující jiné vlákno, jehož je následovníkem.

```
<message from='juliet@kapuletova.com/Z_balkónu' id='z94nb37h'
         to='romeo@montek.net' type='chat' xml:lang='en'>
        <subject>
                I implore you!
        </subject>
        <subject xml:lang='cs'>
                &#x00DA;p&#x011B;nliv&#x011B; pros&#x00ED;m!
        </subject>
        <body>
                Wherefore art thou, Romeo?
        </body>
        <body xml:lang='cs'>
                Pro&#x010D;e&#x017D; jsi ty, Romeo?
        </body>
        <thread parent='e0ffe42b28561960c6b12b944a092794b9683a38'>
                0e3141cd80894871a68e6fe6b1ec56fa
        </thread>
</message>
```

Element <presence> je speciální mechanismus, díky němuž několik entit dostane informaci o stavu entity, kterou popisuje. Po odeslání inicializačního stavu přítomnosti může klient kdykoliv aktualizovat svůj stav odesláním *presence* stanzas.
Atribut *type* může nabývat hodnot:

- error – došlo k chybě při zpracování předchozí zaslané stanzy <presence>. Musí být zahrnut element <error>
- probe – dotaz na stav entity. Měl by být generován pouze serverem jménem uživatele
- subscribe – žádost o povolení přijímat stavy určité entity
- subscribed – dovolení jiné entitě přijímat moje stavy
- unavailable – uživatel již nebude k dispozici pro komunikaci
- unsubscribe – nebude se nadále přijímat stav určité entity
- unsubscribed – žádost o přijímání stavů byla zamítnuta, nebo bylo zrušeno předchozí udělené oprávnění

Jestliže hodnota atributu *type* není ani jedna z výše uvedených, příjemce by měl vrátit stanza error <bad-request/>. Pro tento atribut není žádná hodnota „available“. Chybějící atribut *type* signalizuje, že entita je dostupná pro komunikaci.
Element <presence> má 3 subelementy:

Element <*show*> specifikuje dodatečné informace o dostupnosti entity. Presence stanza nesmí obsahovat víc než jeden element <show>. Není pro něj definován žádný atribut. Obsah elementu musí být jedno z:

- away – entita/zdroj je dočasně pryč
- chat – entita/zdroj právě aktivně chatuje
- dnd – entita/zdroj je zaneprázdněn
- xa – entita/zdroj je pryč na delší dobu

Jestliže element <show> chybí, předpokládá se, že entita je on-line a dostupná.

Element <*status*> obsahuje přirozenou řečí napsaný popis dostupnosti entity. Používá se spolu s elementem <show> k podrobnějšímu popisu, když presence stanza nemá atribut *type*. Může mít jediný atribut – *xml:lang*

```
<presence xml:lang='en'>
     <show>dnd</show>
     <status>Wooing Juliet</status>
     <status xml:lang='cs'>Dvo&#x0159;&#x00ED;m se Julii</status>
</presence>
```

Element <*priority*> specifikuje úroveň priority zdroje. Hodnota musí být celé číslo mezi -128 a +127. Presence stanza nesmí obsahovat víc než jeden element <priority>.

```
<presence type='unavailable' xml:lang='en'>
        <status>Busy IRL</status>
        <priority>1</priority>
</presence>
```

Element <iq> je mechanismus typu „dotaz – odpověď“. Slouží pro výměnu dat mezi uživateli. Na každý dotaz **musí** přijít odpověď (podobně jako u HTML). Kvůli identifikaci dvojice dotaz-odpověď, **musí** mít každá z těchto zpráv stejný identifikátor.

Klient : <iq type=“get“ id=“1“>
Server : <iq type=“result“ id=“1“>
Klient : <iq type=“set“ id=“2“>
Server : <iq type=“error“ id=“2“>

Pro IQ stanzas (**I**nfo/**Q**uery) je nezbytná přítomnost atributu *type*. Jeho možné hodnoty jsou:

- get – žádost o data
- set – žádost o nastavení dat
- result – odpověď na korektní get nebo set
- error – došlo k chybě při zpracování dodaných dat nebo při zpracování předchozího set nebo get

Na přijatou stanza typu result/error nesmí příjemce odpovědět odesláním stanza typu result/error. Příjemce ale může poslat další dotaz. IQ stanza typu „get“ nebo „set“ musí obsahovat právě jeden synovský element specifikující sémantiku dotazu. IQ stanza typu „result“ musí obsahovat jeden nebo žádný synovský element. IQ stanza typu „error“ by měl zahrnovat synovský element obsažený v příslušném set/get a musí zahrnovat element <child/>.

Příklad komunikace:

1) Klient otevře TCP spojení s XMPP serverem, který naslouchá na portu 5222[37].
2) Klient otevře stream k serveru:

   Klient serveru:

```
<stream:stream
        xmlns='Jabber:client'
        xmlns:stream='http://etherx.Jabber.org/streams'
        to='example.com'
        version='1.0'>
```

3) Server odpoví otevřením streamu ke klientovi

   Server klientovi:

```
<stream:stream
        xmlns='Jabber:client'
        xmlns:stream='http://etherx.Jabber.org/streams'
        id='c2s_123'
        from='example.com'
        version='1.0'>
```

4) Obě strany si vymění informace ohledně (ne)použití šifrování.
5) Jestli je možné šifrování => začátek šifrovacího kanálu.
6) Server informuje klienta o možnostech autentizace

   Server klientovi:

```
<stream:features>
        <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
                <mechanism>DIGEST-MD5</mechanism>
                <mechanism>KERBEROS_V4</mechanism>
                <mechanism>PLAIN</mechanism>
                <mechanism>EXTERNAL</mechanism>
        </mechanisms>
</stream:features>
```

7) Klient si některou možnost vybere

   Klient serveru:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='DIGEST-MD5'/>
```

8) Proběhne příslušná autentizace, případně i napojení klienta k určitému zdroji.
9) Výměna XML stanzas

[37] Port 5222 je standardní port pro připojování klientů k XMPP serveru, byl oficiálně přidělen organizací IANA;

Klient serveru, ten přepošle serveru kontaktu (montague.net):

```
<message
  from='julie@kapulet.com'
  to='romeo@montague.net'
  xml:lang='cs'>
        <body> Nejsi ty Romeo, a nejsi Montek?</body>
</message>
```

Klientův kontakt svému serveru, ten přepošle na můj server (kapulet.com):

```
<message
  from='romeo@montek.net'
  to='julie@kapulet.com'
  xml:lang='cs'>
        <body> Nic z obého, má děvo spanilá, nenávidíš-li jedno nebo
               druhé. </body>
</message>
```

10) Ukončení XML obou streamu

Klient: *</stream:stream>*

Server: *</stream:stream>*

11) Ukončení TCP spojení.

4.5 Knihovny implementující XMPP

Pro téměř každý programovací jazyk, který se používá ve větší míře, existuje nějaká knihovna zapouzdřující komunikaci pomocí protokolu XMPP. Zde uvádím ty nejpoužívanější:

QXmpp[38] je multiplatformní C++ knihovna implementující veškerou XMPP komunikaci, kterou může obdržet/vyslat klient. Je napsána v C++/Qt. Je poměrně intuitivní a snadná na použití. Veškerá komunikace (ať již TCP komunikace, nebo XMPP komunikace definovaná v RFC3920 a RFC3921) je zapouzdřena do tříd. Proto se už vývojář nemusí s těmito detaily zatěžovat a může se plně věnovat vývoji vlastní aplikace. Zkušeným uživatelům je však vždy doporučeno prostudovat si i nízkoúrovňové detaily.
Celá knihovna je napsána tak, aby její „styl“ (implementace tříd a jejich pojmenování) byl shodný se samotným Qt. Když chceme vytvořit XMPP klienta, použijeme třídu *QXmppClient*. Když chceme vytvořit XMPP zprávu, použijeme třídu *QXmppMessage* a podobně.
QXmpp je neustále ve vývoji a to pod licencí LGPL. Poslední verze je 0.2.92 vydaná 12. února 2011.
Tato knihovna byla použita při vytváření prototypu aplikace.

Gloox[39] je robustní plnohodnotná XMPP klientská knihovna, napsaná v čistém ANSI C++. To umožňuje integrovat XMPP funkcionalitu do existujících aplikací. Gloox je vydáván pod licencí GNU GPL. Jsou však dostupné i komerční licence včetně technické podpory.

Jabber.py[40] je modul implementující XMPP protokol pro Python. Slouží pro vytváření jak Jabber klientů, tak serverů. Tento modul je vydán pod licencí LGPL.

agsXMPP[41] je knihovna pro práci s protokolem XMPP napsaná v C# pro technologie .NET a Mono. Implementuje základní komunikaci včetně mnoha rozšíření. Např. vCard, multiuživatelský chat, Chat State Notification, zeměpisnou polohu a mnoho dalších. Pro použití agsXMPP je nezbytná knihovna **MatriX**[42], což je knihovna pro práci s protokolem XMPP napsaná v C# pro technologie .NET a Silverlight.

Smack[43] je otevřená knihovna XMPP knihovna napsaná v čisté Javě, která implementuje většinu funkcí využívaných XMPP klienty.

[38] http://code.google.com/p/qxmpp/
[39] http://camaya.net/gloox/
[40] http://Jabberpy.sourceforge.net/
[41] http://www.ag-software.de/agsxmpp-sdk/
[42] http://www.ag-software.de/matrix-xmpp-sdk/
[43] http://www.igniterealtime.org/projects/smack/

4.6 XMPP klienti pro mobilní zařízení

Slick[44]

Klient dostupný pro Symbian, Windows Mobile a Android. Zvládá komunikační protokoly Jabber, ICQ, Yahoo, Facebook chat, AIM, MSN, Google Talk. Je možné se připojit k několika účtům zároveň. V současné době je ve fázi vývoje. Dostupná je zatím pouze (již delší dobu) alfa verze, proto některé funkce ještě nejsou podporované. Nicméně je to i v současné době stabilní, graficky povedený klient. Slick umí například zobrazovat grafické „smajlíky“, ukládat historii, úpravu vzhledu. Do budoucna by měl umět i odesílat/přijímat soubory. V současnosti (alfa verze) je zdarma dostupný pro každého.

IM+[45]

Komerční aplikace (na 7 dní k vyzkoušení zdarma) pro BlackBerry, PalmOS, Windows Mobile, Symbian OS, J2ME. Kromě protokolu XMPP podporuje i AOL, ICQ, MSN, Yahoo, Google Talk a MySpace chat. Podporuje přenos souborů, ale jenom odesílání, neumí přijímat soubory. Podobně Chat State Notification umí jen zobrazovat, ale neodesílá jej. Umí ukládat historii a zobrazovat grafické emotikony.

Talkonaut[46]

Zdarma použitelný multiprotokolový klient dostupný pro Android, Symbian, Windows Mobile a jako Java aplikace. Podporuje protokoly Jabber, Google Talk, ICQ, MSN, AIM a Yahoo. Podporuje víceuživatelský chat, registrace účtů, přenos souborů, kompresi přenášených dat. Podporuje i SIP.

OctroTalk[47]

Komerční produkt, momentálně ve fázi beta verze, která je s časovým omezením zdarma. Dostupný je pro desktop Windows, Windows Mobile, Symbian. Podporuje protokoly GoogleTalk/Jabber, MSN, AIM, ICQ a Yahoo. U GooglaTalk zvládá i hlasovou komunikaci. Podle stránek výrobce by měl umět i přenos souborů, což jsem ale v nabídce aplikace nenalezl.

[44] http://www.lonelycatgames.com/?app=slick
[45] http://www.shapeservices.com/en/products/details.php?product=im&platform=s60v3
[46] http://www.talkonaut.com/
[47] http://www.octro.com/index.php

	Slick	IM+	Talkonaut	OctroTalk	Moje aplikace
historie konverzace	ANO	ANO	NE	NE	ANO
Chat State Notification	ANO	ANO[1]	ANO	NE	ANO
Přenos souborů	NE	ANO[2]	ANO	NE	NE
Více účtů zároveň	ANO	ANO[3]	ANO	NE	NE
Profily kontaktů	NE	NE	NE	NE	ANO
Placený	NE[5]	ANO	NE	NE[5]	NE
Grafické emotikony	ANO	ANO	ANO	ANO[4]	NE

1) jen přijímá
2) jen odesílá
3) od každého protokolu maximálně jeden účet
4) velmi omezené typy
5) pouze dokud není finální verze

Tabulka 4.1 Srovnání XMPP klientů

5 Popis aplikace

5.1 Přihlašovací obrazovka

V okně pro přihlášení (Obrázek 5.2) je nutné zadat přihlašovací jméno uživatele (Jabber identifier) a jeho heslo. Pokud název serveru, ke kterému se chceme připojit, neodpovídá doméně našeho JID, specifikujeme i doménu. Po stisknutí tlačítka „Connect“ se aplikace pokusí přihlásit k zadanému účtu. Když se to podaří, zobrazí se kontakt list. V opačném případě se zobrazí chybové hlášení (Obrázek 5.3).

5.2 Seznam kontaktů

V seznamu kontaktů (Obrázek 5.1) se zobrazují veškeré kontakty. Řazeny jsou nejprve podle stavu – online/offline a poté podle abecedy. V případě, že nějaký kontakt pošle zprávu, když máme zobrazený kontakt list, tak jednak budeme upozorněni zvukovým efektem (pokud není v nastavení vypnutý), ale navíc se u kontaktu zobrazí růžový čtvereček, informující o nepřečtené zprávě. U každého kontaktu se zobrazuje barevný čtvereček, jež má barvu podle aktuálního stavu kontaktu. V menu si můžeme zobrazit informace „O programu“, okno s nastavením aplikace a nebo se odhlásit. Po kliknutí na některý kontakt se otevře okno chatu s daným kontaktem.

5.3 Okno chatu

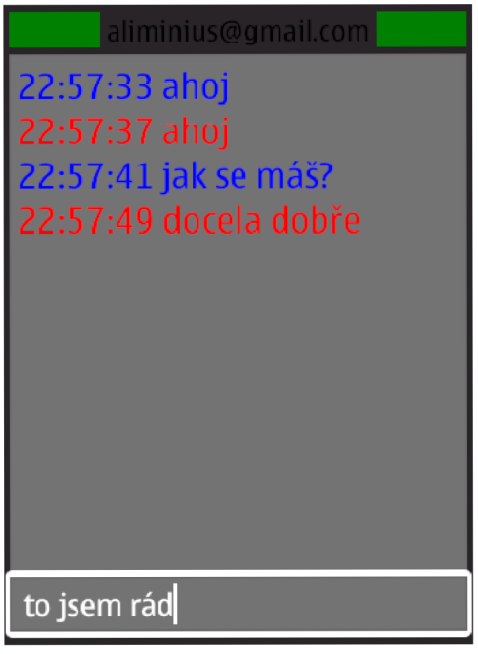
V horní části okna se nachází JID kontaktu, se kterým budeme v tomto okně chatovat. To mění barvu v závislosti na Chat State Notification (když kontakt začne psát zprávu, tak zezelená, když přeruší psaní zprávy, tak zežloutne). Po okrajích JID jsou barevné obdélníčky, zobrazující aktuální stav kontaktu (obdobně jako v seznamu kontaktů). Hlavní část okna zabírá konverzace. Červenou barvou jsou označeny zprávy kontaktu, modrou barvou moje zprávy. Ve spodní části okna se nachází kolonka pro psaní našich zpráv. V menu se můžeme přesunout zpět ke kontakt listu, odhlásit se, zobrazit historii s daným kontaktem, nebo zobrazit jeho profil.

5.3.1 Okno historie

Po zobrazení okna s historií se nám zobrazí seznam dní, ve kterých jsme s daným kontaktem konverzovali. Po kliknutí na některý den se zobrazí konverzace vedená v tomto dni s daným kontaktem. V menu se můžeme vrátit zpět na seznam dní a nebo historii zavřít.

Obrázek 5.2 Přihlašovací obrazovka

Obrázek 5.1 Kontakt list

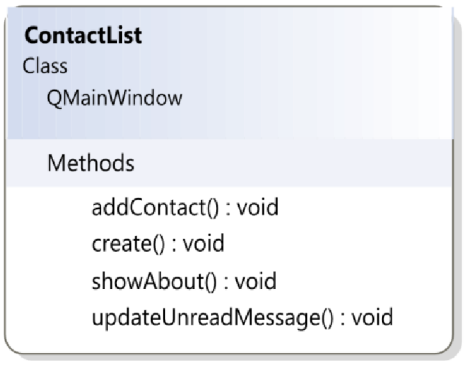


Obrázek 5.4 Okno chatu

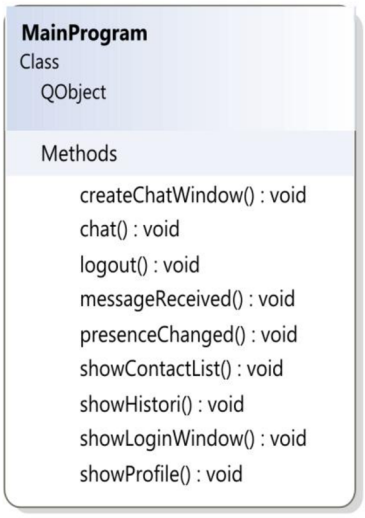
Obrázek 5.3 Chyba při přihlašování

6 Implementace aplikace

Celou funkcionalitu aplikace zajišťuje třída MainProgram. Ta samotná nic nevykresluje, ale zajišťuje komunikaci jednotlivých částí aplikace a zobrazování hlavních oken. Mezi ty patří přihlašovací obrazovka (Obrázek 5.2), seznam kontaktů (Obrázek 5.1), okno chatu (Obrázek 5.4), historie a okno s profilem kontaktu. Samotná komunikace je z větší části zajištěna pomocí signálů a slotů. Po vytvoření instance třídy MainProgram je nutné zavolat metodu *showLoginWindow()*, která zobrazí přihlašovací obrazovku.



Obrázek 6.2 Třída ContactList



Obrázek 6.1 Třída MainProgram

O vykreslení a funkcionalitu přihlašovací obrazovky se stará třída LoginWindow. Kontroluje, jestli jsou vložené přihlašovací údaje validní. U loginu i hesla musí být vyplněna nějaká hodnota. U loginu se navíc kontroluje, jestli je zadané JID validní. Jestliže ano, vyšle se signál přihlašovacími údaji, který by měla zachytit třída MainProgram. Jméno hosta se implicitně bere z doménové části JID. Jestliže se domácí server uživatele jmenuje jinak, než jak je specifikováno v doméně JID, je potřeba ho zadat ručně.

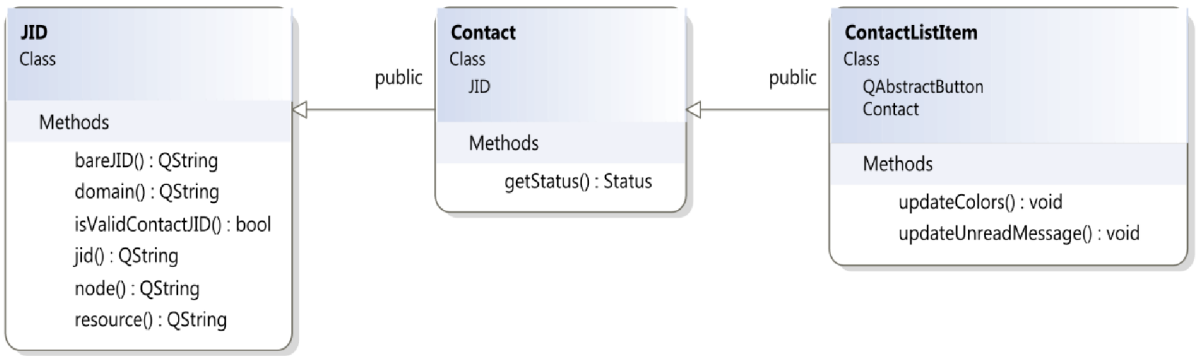
Funkci kontakt listu zajišťuje třída ContactList. Jednotlivé položky kontakt listu jsou instance třídy ContactListItem (Obrázek 6.3). Ta dědí z Qt třídy QAbstractButton, díky čemuž je z ní widget, který reaguje na kliknutí. Informace o tomto kliknutí se předává ContactListu, ten ji předá třídě MainProgram, která zajistí otevření chatovacího okna. Třída ContactListItem zobrazuje 3 jednoduché informace o kontaktu: jeho uživatelské jméno, stav, a jestli od něj nepřišla nová zpráva. Pokud je zavolána metoda *updateUnreadMessage()* třídy ContacListItem, znamená to, že uživatel si buď přečetl nepřečtenou zprávu (a je potřeba se zbavit barevného upozornění na nepřečtenou zprávu), nebo má novou nepřečtenou zprávu a je potřeba informovat uživatele o jeho nové nepřečtené zprávě. To se udělá jednak zobrazením růžového obdélníčku v seznamu kontaktů napravo od uživatelského jména kontaktu a následně se vytvoří instance třídy Notification.

Třída Notification provádí upozornění na novou nepřečtenou zprávu. Na základě uživatelského nastavení aplikace přehrává určený zvukový soubor. Buď přehraje implicitní soubor, který je uložený v binárních datech aplikace, nebo přehraje soubor, který si uživatel

sám zvolil a nebo nepřehraje nic (pokud si uživatel nepřeje být na nepřečtené zprávy upozorňován). K přehrávání zvukového souboru je využita Qt třída QSound.
Ve třídě Notification je implementováno i upozorňování na zprávy formou vibrací mobilního zařízení. Tento způsob je však v nastavení aplikace implicitně vypnutý (a nedá se i z uživatelského hlediska aktivovat). Je to z důvodů nedostatečných oprávnění pro překlad aplikace s tzv. „systémovými“ oprávněními. Konkrétně se v případě vibrací jedná o oprávnění ReadDeviceData a WriteDeviceData.

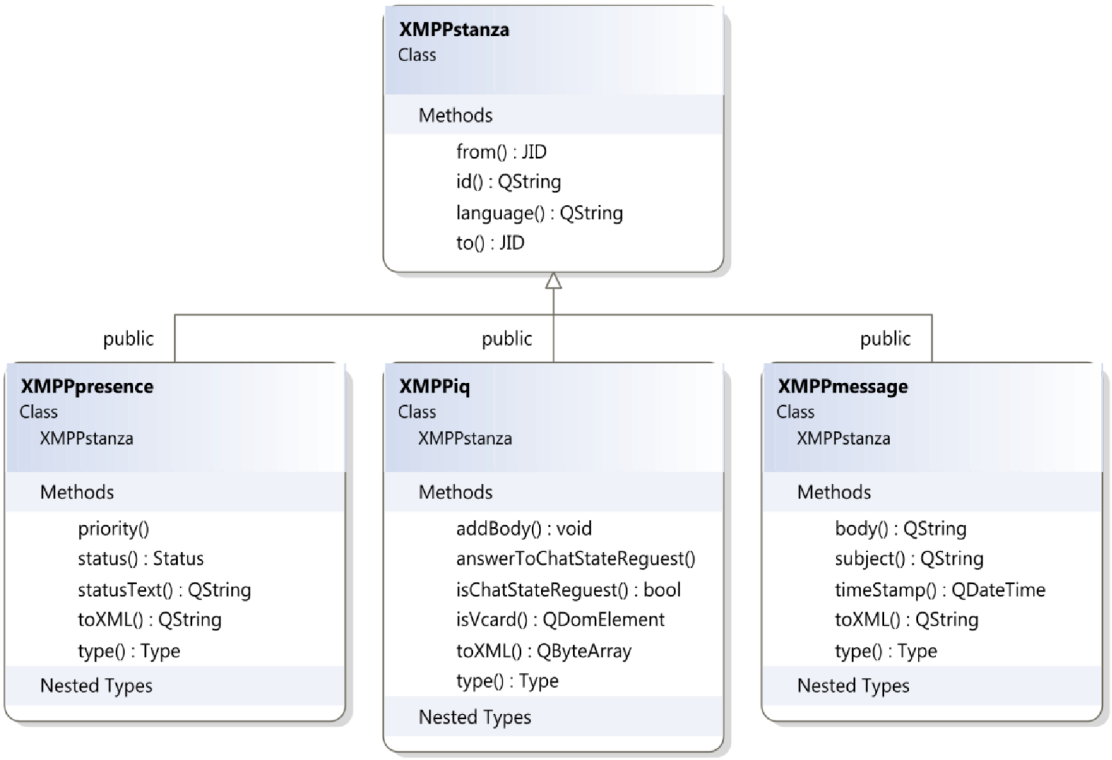
Do třídy ContactList se musí nejprve vložit všechny kontakty pomocí metody *addContact()*. Až jsou kontakty „nahrány“, zavoláním metody *create()* dojde k jejich vykreslení v okně. Seřazeny jsou v pořadí „přítomné“ kontakty a „nepřítomné“ kontakty. Přítomné i nepřítomné kontakty jsou následně řazeny podle abecedního pořadí uzlů (z Jabber identifier).

Chatovací okno vykresluje třída ChatWindow. Zprávy odesílá signálem *sendMessage()*, který zachytává třída MainProgram. Naopak příchozí zprávy jsou přijímány slotem *newMessage()*. Také zpracovává Chat State Notification. Upozornění na přerušení psaní zprávy je odesláno po 7 vteřinách. ChatStateNotification je (stejně jako klasická zpráva) odesíláno signálem *sendMessage()* a přijímáno slotem *newMessage()*.
Pokud do chatovacího okna dorazí zpráva, je nejprve zjištěno, o jaký typ zprávy se jedná. Jestli jde o klasickou textovou zprávu, nebo o zprávu typu Chat State Notification. Pokud jde o klasickou zprávu, je vložena červenou barvou do pole s konverzací a uložena do historie konverzace (pokud není v nastavení aplikace zakázáno ukládání historie). Následně se zjišťuje, jestli je okno právě zobrazeno. Jestliže je zobrazeno, nic víc se neděje. Pokud ale není zobrazeno, je potřeba informovat uživatele, že přišla nová zpráva. To se udělá tak, že se vyšle signál *unreadMessageStateChanged()*, který putuje od ChatWindow, přes MainProgram do ContactList, který zavolá metodu *updateUnreadMessage()* příslušné třídy ContactListItem.



Obrázek 6.3 Hierarchie dědičnosti třídy ContactListItem

Jednou z nejpoužívanějších tříd v aplikaci je třída JID. Ta reprezentuje Jabber Identifier. I pro jednotlivé XMPP stanzy (message, presence, iq) jsou vytvořeny třídy (Obrázek 6.4), které s nimi usnadňují komunikaci. Základní třídou je pro ně XMPPstanza. Ta obsahuje atributy, které mohou mít všechny 3 stanzy. Jedná se o odesílatele, příjemce, id stanzy a jazyk.
XMPPmessage je třída reprezentující message stanzu – stanzu pro odesílání zpráv. Stejně tak XMPPpresence reprezentuje stanzu presence pro odesílání stavu a XMPPiq reprezentuje iq stanzu pro odesílání požadavků.



Obrázek 6.4 Třídy reprezentující jednotlivé XMPP stanzy

XMPP komunikaci zapouzdřuje třída XMPPclient. Ta ale nekomunikuje přímo se serverem, jen vytváří rozhraní pro snazší použití. Samotnou XMPP komunikaci zajišťuje třída XMPPconnection. Ta dědí z třídy QSslSocket, takže umožňuje klasickou i šifrovanou TCP komunikaci. Po navázání TCP spojení se serverem mu automaticky pošle otevírací XMPP stream a pokusí se u něj se zadanými přihlašovacími údaji autentizovat. Pokud server nabízí možnost šifrovaného spojení (TLS), začne s ním komunikovat šifrovaně. V Qt se šifrované spojení aktivuje jednoduše tak, že se zavolá metoda *startClientEncryption()* třídy QSslSocket. O navázání šifrovaného spojení se už postarají Qt knihovny.
Po navázání spojení vyšle informující signál. Stejně tak při přijetí zprávy od serveru, ji zpracuje a pošle ji signálem „nadřazeným“ třídám. Takže například když dorazí změna stavu některého kontaktu, je vyslán signál, který putuje z XMPPconnection do XMPPclient, odtud do MainProgram. Z MainProgram je odeslán jednak do příslušného ChatWindow ale i do ContactList. Ten předá informaci o změně stavu příslušnému ContactListItem.
Jakákoliv příchozí data jsou předána metodě *handleData()*. Její nejdůležitější funkcí je zajistit, aby data předaná dále ke zpracování byla validním XML elementem. Může se například stát, že příchozí data jsou velkého rozsahu (obvyklé u obrázků), proto jich dorazí najednou jen část a musí se čekat na „zbytek“. Jakmile přišel celý validní XML element, je předán metodě *handleStanza()*. Ta zjistí, o jaký element se jedná (odpověď na autentizační požadavek, změna prezence, …) a podle toho zareaguje (odpoví na něj a nebo ho zpracuje a odešle „vyšším“ třídám).
Po úspěšné autentizaci je opět vyslán informující signál. Po jeho přijetí ho třída XMPPclient „přepošle“ dál, ale také zavolá metodu *askForRoster()* třídy XMPPconnection. Tím je serveru odeslán požadavek, aby klientovi poslal seznam kontaktů. Požadavek je odeslán s identifikátorem „xmppBP2011roster“, proto i odpověď je očekávána v iq stanze s tímto identifikátorem.

Přijatý kontakt list je předán třídě XMPProster. Ta jej zpracuje a jednotlivé položky kontakt listu si uloží jako instance třídy XMPProsterItem. Po uložení celého kontakt listu vyšle XMPProster pro každý kontakt požadavek na jeho vCard. Jakmile dorazí jednotlivé vCard, jsou předány XMPProsteru, který je uloží do příslušných XMPProsterItemů. Může se zdát, že takto přijaté profily kontaktů mohou být neaktuální, než kdyby se stahovali až při požadavku na jejich zobrazení. To může být pravda, nicméně pouze ve vzácných případech. Profily kontaktů jsou obecně stálé informace, měněné jen jednou za delší časový úsek. Naproti tomu má tento přístup výhodu, že při požadavku na zobrazení profilu kontaktu nenastane prodleva, vCard je již staženo a může být zobrazeno okamžitě, díky čemuž nemusí uživatel čekat.

Pokud nastane při komunikaci se serverem nějaký problém (spojení je přerušeno, server odmítl klienta autentizovat, SSL handshake se nepovedl,…), je vyslán signál *comunicationProblem()* s informacemi o nastalé situaci.

Zobrazení okna s profilem kontaktu obstarává třída ProfileWindow. Ta si jako parametr v konstruktoru bere třídu XMPPvCard, která reprezentuje vCard[48], což představuje v XMPP profil kontaktu. Z údajů, které jsou ve vCard vyplněny, zobrazí profil kontaktu včetně jeho avatara. Jednotlivé položky profilu jsou instance třídy ProfileItem (kromě avatara).

Nastavení aplikace se provádí v okně, jež vykresluje třída SettingsWindow. Samotnou práci s nastavením pak zajišťuje třída Settings. Ta k ukládání/načítání nastavení využívá Qt třídu QSettings. Samotný soubor s nastavením se pak uloží do adresáře
„{HomeDirectory}\.config\{OrganizationName}\{ApplicationName}.conf\".
Kde {HomeDirectory} je domovský adresář. Na platformě Symbian je to „C:\Data". {OrganizationName} je jméno organizace, která vyvinula aplikaci. {ApplicationName} je název aplikace. V případě této aplikace bude tedy soubor s nastavením v adresáři:
„C:\Data\.config\Tomas Vanak\Jabber.conf\". V Qt se jméno organizace a aplikace nastavuje metodami *QCoreApplication::setApplicationName()* a
QCoreApplication::setOrganizationName(). Soubor *.conf je formátován jako INI soubor.

Zobrazení historie zajišťuje třída HistoryWindow. Samotnou práci s historií ale zajišťuje třída History. Historie se ukládá do adresáře „{HomeDirectory}\JabberBP2011\history". Zde se pro každého uživatele, který se přihlásí do aplikace, vytvoří vlastní podadresář s názvem takřka shodným s bareJID, ale s vypuštěním zavináče. Pro každý kontakt, se kterým budeme chatovat, se vytvoří také separátní adresář. Do tohoto adresáře se bude ukládat konverzace z každého dne do zvláštního souboru.

[48] vCard představuje obecně elektronickou podobu klasické vizitky;

7 Závěr

V této práci jsem se seznámil s vývojovým prostředím Nokia Qt SDK a poprvé si vyzkoušel vytvořit aplikaci pro mobilní zařízení. Protože vlastním mobilní telefon s operačním systémem Symbian, zaměřil jsem se na zvláštnosti, se kterými je potřeba se potýkat u této platformy. Protože mám rád instant messagingovou komunikaci, rozhodl jsem se vytvořit klienta pro decentralizovanou a otevřenou síť Jabber.
Výsledná aplikace je tedy XMPP klient, který se umí připojit k serveru, navázat s ním šifrované spojení (pokud ho server podporuje), autentizovat se a začít komunikovat s jinými klienty. Z rozšíření XMPP (XEP[49]) jsem si vybral Chat State Notification. Díky tomu v klientovi vidím, když mi někdo začne psát zprávu, když přeruší psaní zprávy nebo když uzavře okno konverzace. Stejně tak vidí i můj kontakt, že mu píši apod.
Jako největší přínos této práce vidím to, že jsem se blíže seznámil s propracovaným, multiplatformním frameworkem Qt a prakticky si ověřil získané znalosti na větším projektu. Další pozitivum s prací na větším projektu bylo získání zkušeností, plánování postupu práce dopředu.
Zdrojové kódy mé práce jsem uvolnil pod GNU GPL licencí a uveřejnil je na internetové stránce http://jabberclientinq.sourceforge.net/, odkud si je může každý stáhnout.
Nevytvořil jsem robustní knihovnu, která by podporovala veškeré XMPP možnosti, ani aplikaci s množstvím funkcí, které by stejně drtivá většina uživatelů nevyužila. Vytvořil jsem jednoduchého klienta, který má jednoduché ovládání a plní dobře základní funkce, na které je každý zvyklý.
Do budoucna bych se chtěl ve volném čase dále věnovat práci na tomto klientovi. I když je aplikace docela dobře použitelná, je i nadále co vylepšovat. Například lépe ošetřit možné chybové stavy při XMPP komunikaci, doplnit některá užitečná XEP rozšíření nebo vylepšit vzhled.

[49] Rozšíření XMPP vydaná nad rámec RFC; http://xmpp.org/xmpp-protocols/xmpp-extensions/

8 Literatura

[1] YOUNG, Michael J. XML krok za krokem. Vydání druhé. Brno : Computer Press, 2006. 471 s. ISBN 80-251-1070-2.

[2] BLANCHETTE, Jasmin; SUMMERFIELD, Mark. C++ GUI Programming with Qt 4. Stoughton (Massachusetts) : Trolltech, 2006. 537 s. ISBN 0-13-187249-4.

[3] SAINT-ANDRE, Peter; SMITH, Kevin; TRONÇON, Remko. *XMPP:* The Definitive Guide : Building Real-Time Applications with Jabber Technologies. First edition. Sebastopol : O'Reilly Media, 2009. 285 s. ISBN 978-0-596-52126-4.

[4] FITZEK, Frank H.P.; TORP, Tony; MIKKONEN, Tommi. Qt for Symbian. [s.l.] : John Wiley & Sons, 2010. xxi, 185 s. Dostupné z WWW: <http://www.ebooksdownloadfree.com/Programming-general/Qt-for-Symbian-BI11252.html>. ISBN 978-0-470-75010-0.

[5] SAINT-ANDRE, Peter. Extensible Messaging and Presence Protocol (XMPP): Core [online] c2004. <http://www.ietf.org/rfc/rfc3920.txt>. [cit. 2004-10]

[6] SAINT-ANDRE, Peter. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence [online] c2004. <http://www.ietf.org/rfc/rfc3921.txt>. [cit. 2004-10]

[7] Qt Reference Documentation [online]. c2010 [cit. 2011-05-09]. Dostupné z WWW: <http://doc.trolltech.com/latest/index.html>.