Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Vývoj aplikací s využitím knihovny Qt

Václav Boháč

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačního inženýrství

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Boháč Václav

Informatika

Název práce

Vývoj aplikací s využitím knihovny Qt

Anglický název

Application development using Qt library

Cíle práce

Bakalářská práce se zabývá problematikou vývoje software s využitím knihovny Qt. Hlavním cílem práce je představení knihovny Qt a jejích vlastností a demonstrace zjištěných poznatků formou vytvoření ukázkové aplikace.

Metodika

Metodika řešené práce je založena na studiu a analýze odborných informačních zdrojů. Na základě syntézy teoretických poznatků budou shrnuty základní vlastnosti knihovny Qt a popsáno její využití při tvorbě aplikací. Následně bude vytvořena ukázková aplikace demonstrující tyto poznatky.

Harmonogram zpracování

05/2013-12/2013 - studium a analýza informačních zdrojů
01/2014 - kontrola postupu práce (zápočet)
01/2014-03/2014 - tvorba aplikace
03/2014 - odevzdání práce

Rozsah textové části

35-40 stran

Klíčová slova

Objektové programování, C++, Open source, KDE, Linux, GUI, XML, QML, Threading

---

Doporučené zdroje informací

Stroustrup Bjarne. C++ programovací jazyk. Praha: BEN-technická literatura, 1997, 686 s. ISBN: 80-86056-20-1.

Bradley Neil. XML kompletní průvodce. Praha: Grada Publishing, 2000, 540 s. ISBN: 80-7169-949-7.

Summerfield Mark. Rapid GUI Programming with Python and Qt. Westford USA: Prentice Hall PTR, 2007, 625 s. ISBN: 978-0-13-235418-9.

Qt library documentation. Norway: Oslo. Digia - Qt Projetct [online]. Dostupné z WWW <http://www.digia.com>, <http://www.qt-project.org/doc>.

Blanchette Jasmin, Summerfield Mark. C++ GUI Programming with Qt4. Westford USA: Prentice Hall PTR, 2008, 625 s. ISBN: 978-0-13-235416-5.

Vedoucí práce

Brožek Jiří, Ing., Ph.D.

Termín odevzdání

březen 2014

Ing. Martin Pelikán, Ph.D.
Vedoucí katedry

prof. Ing. Jan Hron, DrSc., dr. h. c.
Děkan fakulty

V Praze dne 12.9.2013

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj aplikací s využitím knihovny Qt" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu bakalářské práce Ing. Jiřímu Brožkovi Ph.D. za odborné rady, pozornost a péči, kterou věnoval mé bakalářské práci.

Vývoj aplikací s využitím knihovny Qt

Application development using Qt library

Souhrn

Bakalářská práce se zabývá analýzou vývoje software s využitím frameworku Qt. Předestírá historii jeho vzniku a různé koncepce, kterými knihovna procházela. Uvádí čtenáře do základních principů objektového programování a nabízí základní přehled nejdůležitějších komponent frameworku Qt a jejich použití. Popisuje principy jeho fungování a práci v prostředí jazyka C++. Obsahuje popis SDK, které je možno využívat při vývoji.

Summary

Bachelor thesis analyzes the development of software using Qt framework. It suggests the history of its origin and different concepts of developing the library. It introduces readers to the basic principles of object-oriented programming and provides a basic overview of the most important components of the framework Qt and their use. It describes the principles of its functioning and working environments of C + +. It contains a description of the SDK that can be used during development.

Klíčová slova: Objektové programování, C++, Open source, GUI, XML, QML, Threading, Linux, KDE

Keywords: Object oriented programming, C++, Open source, GUI, XML, QML, Threading, Linux, KDE

Obsah

1 Úvod

Vývoj software byl ještě do nedávné doby doménou specialistů, kteří při své práci byli detailně obeznámeni s hardware, na kterém bude vyvíjený software provozován. Právě díky hardwarovým omezením byli nuceni maximálně hospodárně využívat jeho možnosti. Toto zvláště platilo v dobách prvních generací počítačů. Teprve příchodem tranzistorů, integrovaných obvodů a mikroprocesorů třetí a čtvrté generace počítačů se toto pravidlo mění. S nástupem stolních počítačů na bázi architektury IBM PC a jejich postupnou cenovou dostupností ve společnosti dochází k masovému používání výpočetních systémů nejen v korporátní sféře. Dochází k širokému rozšíření do segmentu domácností. Tento segment se ještě více penetruje současnou pátou generací počítačů, která je charakteristická signifikantní miniaturizací. Počítače se stávají automatickou součástí domácností a jsou integrovány do prostředků běžného použití, jako jsou mobilní telefony, tablety nebo zařízení zábavného charakteru, jako televize, domácí kina nebo herní konzole. Tento vývoj byl předznamenán zvýšenou potřebou tvorby softwaru. Dochází k rozvoji stávajících relačních, ale zvláště pak k vývoji nových, převážně objektových, programovacích jazyků.

2 Cíl práce a metodika

Práce je rozdělena do dvou částí - teoretické a praktické. V první teoretické části je nastíněn přehled použité technologie. Historie frameworku Qt a jeho současnost. Následuje popis jednotlivých samostatných vývojových nástrojů a dále popis IDE vývojového prostředí, které je možno použít při tvorbě aplikací. Zmíněn je také ekosystém obklopující tento framework.

Druhá část obsahuje praktický příklad s popisem jednotlivých kroků při vývoji aplikace s použitím Qt frameworku. Příklad je postaven na několika vybraných komponentech knihovny Qt.

2.1 Hlavní cíl

Hlavním cílem je poskytnutí informací týkajících se vývoje software s využitím knihovny Qt. Práce má za cíl poskytnout vhled do používaných technologií a koncepci celého frameworku Qt, popsat vývojové prostředí a nakonec vytvořit ukázkovou aplikaci.

2.2 Dílčí cíle

- nastínit základy objektového programování a upozornit na jeho výhody a nevýhody
- použití programovacího jazyka C++, jeho historii koncepce a současnost
- popis vývojového prostředí SDK Qt či jednotlivých nezávislých komponent, které SDK do sebe integruje

2.3 Metodika práce

Vývoj aplikace se skládá z následujících kroků:

- Definice požadavků a potřeb uživatele/zadavatele (uživatelská specifikace)
- Úvodní a podrobná analýza
- Návrh designu pokud se jedná o GUI aplikaci
- Mapování bussiness požadavků do jednotlivých aplikačních tříd
- Propojení GUI (uživatelského rozhraní) a aplikačního rozhraní
- Otestování aplikace

3 Teoretická východiska

V této části práce se budeme zaobírat historií fameworku Qt, základními principy objektově orientovaného programování, aspekty multiplatformního programování a v poslední části této kapitoly bude nastíněna struktura instalace a popis frameworku Qt a jeho vývojového prostředí Qt Creator.

3.1 Historie frameworku Qt

Zakladatelé a první vývojáři projektu byli Haavard Nord a Erik Chambe-Eng. První verze byla dostupná v květnu roku 1995 (1). Cílem projektu bylo vytvořit multiplatformní objektově orientovaný Graphical User Interface (dále jen GUI) framework napsaný v jazyce C++. Společnost dostala název Trolltech. Framework bylo možno použít jak na platformě OS Windows, tak Unix. V této době zároveň vzniká na linuxové platformě projekt desktopového prostředí KDE (K Desktop Environment) (2). Jeho zakladatel Matthias Ettricht později přechází do firmy Trolltech a rozhoduje se implementovat KDE s využitím prostředků frameworku Qt. Tímto se Qt stává de facto standardem pro vývoj C++ GUI na platformě Linux. Framework je vyvíjen jak pod svobodnou licencí GPL tak pod licencí komerční. V roce 2001 byla vydána verze 3.0, která již podporovala OS Window, Mac, Unix a Linux. Následná, verze 4.0, znamenala technologický skok oproti předchozí řadě s postupnou do široka rozvíjenou funkcionalitou. Bylo kompletně přepsáno aplikační jádro. Následkem byla ovšem zpětná nekompatibilita s předchozí verzí a bylo nutno vyvinutý software portovat do verze řady 4.0, což byl pro mnoho firem zdlouhavý a bolestný krok. Přes tento handicap se framework Qt stával stále populárnějším.

Firmu Trolltech koupil v roce 2008 mobilní gigant firma Nokia. Ta nasměrovala framework směr k mobilním zařízením. Firma Nokia, vzhledem ke svému rozhodnutí orientovat se na mobilní OS firmy Microsoft, nakonec zakládá komunitní projekt qt-project a v roce 2012 prodává framework Qt firmě Digia. Digia společně s komunitou qt-projekt na konci prosince roku 2012 vydává verzi 5.0. I tato verze znamená generační skok zaměřený na poslední trendy hardwarových zařízení, jako jsou chytré telefony, tablety a přenosná zařízení vůbec. V současnosti (březen 2013) je stále doporučována, zvláště pro korporátní vývoj, stabilní řada verze 4 frameworku Qt.

3.2 Programovací jazyky

Programovací jazyky prošly především v posledních dekádách bouřlivým vývojem. Tento vývoj předznamenaly především možnosti a potřeby, které plynou z uplatňování nových softwarových technologií a především potřeb internetových aplikací. Dnes to jsou velmi populární jazyky Java, C# či Python a nad nimi vystavěné frameworky jako např Spring, .NET či Ruby on Rails. Ve všech těchto případech se jedná o interpretované jazyky, které se v první fázi překládají do mezikódů překladových objektů. Ke svému běhu potřebují běhové prostředí, které teprve ve fázi zavádění tohoto mezikódu buď plně tento mezikód interpretuje, nebo lépe „just in time“, překládá do spustitelného formátu konkrétní aktuální běhové hardwarové platformy.

Přes nepřehlédnutelné výhody těchto jazyků se v popředí zájmu programátorů stále drží jak původně UNIX-ový „systémový jazyk“ C, tak zvláště jeho rozšíření o objektové rysy jazyk C++. Autor tohoto jazyka, Bjarne Stroustup, ale tento příměr vyvrací (3). Podle něho nejde o pouhé rozšíření jazyka C. Určité konstrukce jazyka C nejsou v C++ dovoleny a některé konstrukce, byť lexikálně stejné, mohou mít v každém jazyce jiný význam. Jazyk C++ je nový jazyk, který pouze zachovává částečně lexikální kompatibilitu s jazykem C. Zvláště posledně přijatý standard ISO C++11 značně rozšiřuje možnosti C++. Jazyk C++ je přísně typový jazyk (3). Překlad zdrojových kódů se provádí přímo pro určitou hardwarovou platformu. Pokud se dodržují zásady přenositelnosti při kódování lze dosáhnout velmi dobré přenositelnosti jak mezi hardwarovými, tak i softwarovými architekturami – operačními systémy. Tím, že kód je překládán přímo do spustitelného formátu, tak i běh aplikace je rychlejší, než u interpretovaných jazyků. Jako relativní nevýhodu lze ale uvést to, že při kódování v C++ je je každý programátor zodpovědný za tzv. memory management, nebo-li správu paměti. To klade na programátory zvýšené nároky. Při špatné správě paměti dochází během běhu aplikace k jejím únikům, což vede k postupnému odčerpávání prostoru ve sdílené paměti a může vést v nejhorších případech až k pádu aplikace. Druhým velkým „nebezpečím“ je špatné používání ukazatelů, kdy v případě jejich špatného použití (snaha o čtení nealokované paměti, čtení z paměti přidělené jinému procesu, vícenásobné uvolnění paměti atd.) dochází k okamžitému ukončení programu operačním systémem.

3.3 Vývojové nástroje

Při vývoji software se velmi osvědčily, tzn. Vývojové nástroje, či Vývojová prostředí. Druhý jmenovaný většinou pouze integruje jednotlivé nástroje a zastřešuje je

grafickým rozhraním tzv. IDE (*Integrated Development Environment*) (4). Většina z těchto prostředí integruje také nástroj pro rychlý vývoj tzv. RAD (*Rapid Application Development*) (5). Některé z programovacích jazyků již v době svého uvedení přicházejí s vlastním IDE (jako např. .NET), které se přizpůsobuje programovacím paradigmatům daného jazyka. Na druhou stranu např. pro jazyk C++ neexistují takto postavené IDE nástroje. Jazyk C++ je koncipován jako univerzální vývojová platforma, která není pevně spojena s žádným grafickým IDE rozhraním.

Pro vývoj v prostředí frameworku Qt je možno použít více variant. Vývoj lze uskutečňovat buď mimo jakékoliv vývojové prostředí pouze za použití nutného minima, což je textový editor a překladač jazyka (někteří Linuxovývývojáři stále preferují textový editor vi, či jeho novější verzi vim (6) a konzolový příkazový řádek). Pro efektivní vývoj je ale dnes již nutnost použít některé z dostupných vývojových prostředí, které obsahují minimálně pokročilý editor zvýrazňující syntaxi jazyka, poskytující on/off-line nápovědu, doplňování kódu a kontrolu syntaxe (Intelli sence), ladící nástroj (debuger), překladač (compiler), správce projektů či nástroj pro týmovou spolupráci (např. Subversion, CVS).

Při výběru vývojového prostředí se lze rozhodovat, na jakém operačním systému (dále jen OS) bude vývoj probíhat, nebo zda je potřeba multiplatformního vývojového prostředí. Na platformě OS Linux je možno volit například mezi prostředím KDE s aplikací Kdevelop, či Anjuta pro prostředí Gnome. Na platformě OS Windows jednoznačně dominuje prostředí Microsoft Visual Studio. K dokončení výčtu je potřeba také zmínit multiplatformní (a multijazyková) vývojová prostředí jako Eclipse či Netbeans, které jsou designované primárně pro vývoj Java aplikací, přesto obsahují rozšíření - možnosti pro vývoj v jazyce C++.

Doporučovaným vývojovým prostředím pro framework Qt je nástroj Qt Creator (7). Toto vývojové prostředí vyvíjí a, stejně jako celý Qt framework, zastřešuje společnost Digia Plc., Finsko (8).

3.4 Vývoj aplikací ve frameworku Qt

3.4.1 *Multiplatformní vývoj*

Základní myšlenka frameworku Qt byla právě multiplatformní vývoj. Znamená to možnost běhu aplikace na různých typech OS a různých typech hardwarových platforem. Framework Qt přináší vývojáři jednotné API, které zjednodušuje a zrychluje vlastní vývojový cyklus. I přes tyto neoddiskutovatelné výhody je vývojář multiplatformních

aplikací nucen být obezřetný při použití technik a postupů, které jsou aplikovatelné a použitelné pouze na jedné konkrétní cílové platformě (OS).

Framework Qt aktuálně podporuje desktopové/serverové platformy: Windows, Linux/X11, Max OS X a Solaris. V segmentu embedded pak platformy: Embedded Linux, Windows CE, Windows Mobile, Windows Embedded Compact 7, Integrity RTOS, QNX RTOS a VxWorks RTOS a mobilní platforma Symbian(9).

3.4.2 *Objektové programování*

Objektové programování (3) je, dá se říci, fenomén dnešních programovacích jazyků a technik. Základní stavební jednotka modelované reality je objekt. Ten je nositelem jak vlastností, tak i dat. Díky vlastnosti zapouzdření, kde ke vnitřním (skrytým) atributům a metodám není zvenčí objektu přímý přístup, jsou (mohou, tedy spíše návrhově musí být) data ochráněna od nepovolených či nechtěných modifikací z vně objektu. Programátor může pak abstrahovat od jejich „skrytých“ vlastností, které jsou plně v interní režii daného objektu. Objekt pracuje jako „černá skříňka“ a pouze očekává vstupy a vrací na výstupu výsledky své práce. Velkou výhodou a základním stavebním kamenem objektového programování je dědičnost, která umožňuje skládat objekty dle jejich logického hierarchického uspořádání na „předky a potomky“. Každý potomek je vždy schopen zastoupit svého předka. Předek naopak vždy dokáže „skrýt“ potomka. Pokud do výčtu možností přidáme použití virtuálních metod, dostáváme se k technologii polymorfismu (3). Odkazovaný objekt se chová podle toho, jaké skutečné třídy – úrovně dědičnosti – je instancí. To znamená, že při volání metody na typovém ukazateli na předka, dochází k vlastnímu volání na příslušné metodě konkrétního potomka, který je za ní skryt.

Při aplikování dědičnosti je potřeba vždy vážit, zda se skutečně jedná o dědičnost, nebo o vlastnost objektu. Lze použít základní pomůcku, kdy se ptáme, zda prvek „je“ objekt nebo zda „je atributem“ objektu. Pokud si můžeme odpovědět že „je“ tzn. např. člověk „je“ živočich pak použijeme dědičnost. V opačném případě při použití „je atributem“ např. člověk není ruka, ale ruka „je atributem“ člověka se jedná nikoliv o dědičnost, ale o vlastnost objektu.

3.4.3 *Instalace frameworku Qt*

Na instalaci je možné se dívat ze dvou pohledů. První je příprava běhového prostředí a druhý vlastní instalace dle typu instalačního balíku (10).

3.4.3.1 *Běhové prostředí*

Pro běh frameworku Qt je potřeba mít k dispozici běhové prostředí tzv. runtime. Ten je většinou zdarma distribuován výrobcem OS a již ho obsahuje placená verze Qt SDK (Windows, Mac). Další nutný nástroj je překladač pro překlad zdrojového kódu, linker pro sestavení programu a debugger pro ladění programu. I tyto nástroje jsou buď součástí OS, nebo je výrobce nabízí zdarma ke stažení ze svých webových stránek či repozitářů softwaru. Překladač se stará o přeložení kódu psaného programátorem do spustitelného formátu (do strojového kódu) konkrétní platformy (OS) a linker spojuje jednotlivé runtimové knihovny s knihovnami Qt a vlastním aplikačním kódem.

3.4.3.2 *Instalace*

První možností je využít placenou verzi. K dispozici je kompletní SDK, které obsahuje IDE vývojové prostředí Qt Creator a nástroje pro překlad a ladění programu (možno je také využívat pouze jednotlivé nástroje, z kterých se SDK skládá – viz další kapitoly popisující jednotlivé komponenty IDE). Dále obsahuje kompletní dokumentaci k celému SDK. Toto SDK vydává firma Digia pouze v binární podobě pro OS Linux, Windows a Mac. Vzhledem k tomu, že celý framework je open-source (11) jsou dostupné také jeho zdrojové kódy. Pokud binární forma neexistuje, je možno si pro konkrétní platformu a konkrétní C++ runtime Qt framework přeložit (zkompilovat).

Druhou možností je neplacená verze. Jak již bylo zmíněno výše celý framework Qt je open-source a tudíž jeho použití je zdarma a zároveň jsou dostupné jeho zdrojové kódy. Ze stránek projetku www.qt-project.org je možno stáhnout balík samostatné knihovny Qt, nebo balík vývojového IDE prostředí Qt Creator. Pro tyto distribuce jsou připraveny předkompilované balíky pro Linux, Windows a Mac a zdrojové kódy pro možnost vlastní kompilace. Nástroje pro překlad a ladění programu je nutno explicitně doinstalovat. Na platformě Linux, jsou všechny tyto nástroje již připravené v repozitářích softwaru příslušné linuxové distribuce.

Pro použití překladu a ladění programu je potřeba instalovat na OS Windows Windows SDK, pro Mac OS balík Xcode a gcc, gdb pro OS Linux. Detailní informace ohledně instalace viz (10).

3.4.4 Struktura frameworku Qt

3.4.4.1 Qt Quick

Qt Quick (12) je novější součást frameworku Qt. Je to kolekce technologií navržených pro tvorbu moderního uživatelské prostředí pracující s grafickými efekty. Jedná se o aplikace vyvíjené pro mobilní zařízení, multimediální centra, tablety a jiná přenosná zařízení. Qt Quick se skládá z vlastních vizuálních prvků, deklarativního jazyka a běhového prostředí. Toto běhové prostředí interpretuje deklarativní kód a zároveň slouží jako interface mezi prostředím Qt Quick a standardním C++ API. Toto je možno využít při implementaci aplikací, které požadují moderní uživatelské prostředí a výkon aplikačního podloží napsaného v jazyce C++.

Jazyk QML (12) je skriptovací deklarativní interpretovaný jazyk, který syntaxí velmi připomíná jazyk JavaScript. Díky využití, dále popisované části Qt Webkit, je možno psát kód nejen v nativním jazyku QML ale také přímo JavaScript kód.

Prostředí Qt Quick je také integrováno do vývojového IDE. Jedná se o grafický návrh, editor a možnosti použití ladění programu. Integrace grafických komponent bohužel není zdaleka vyčerpávající a úplná. Prostředí Qt Quick frameworku Qt prochází značným vývojem, a tak se dá očekávat v dalších verzích větší a větší integrace Qt Quick komponent do stávajících vizuálních nástrojů IDE prostředí.

Vzhledem k tomu, že obliba přenosných zařízení stále stoupá, je a bude vývojáři této části frameworku Qt v budoucnu věnována stále větší pozornost a důležitost.

3.4.4.2 Qt C++ Framework

Qt C++ framework (13) je základní stavební kámen celého Qt. Dnes již velmi rozvětvené a stabilní API umožňuje plnohodnotnou a efektivní práci při vývoji software. Je základem jak jednotlivých vývojových nástrojů, tak celého IDE (Qt Creator viz dále). Detailní popis C++ části Qt frameworku popisují následující kapitoly.

3.4.4.3 Qt Webkit

Vykreslovací systém založený na open source projektu WebKit podporující standardní internetové technologie. Modul implementuje internetový prohlížeč. To umožňuje plnou interakci Qt aplikace s prostředím internetu. Zároveň zajišťuje vykreslování značkovacích

jazyků Hyper Text Markup Language (HTML), Extensible Hyper Text Markup Language (XHML), Scalable Vector Graphics (SVG), zpracování stylů Cascading Style Sheets (CSS) a běh skriptů v jazyce JavaScript.

3.4.4.4 *Qt Creator*

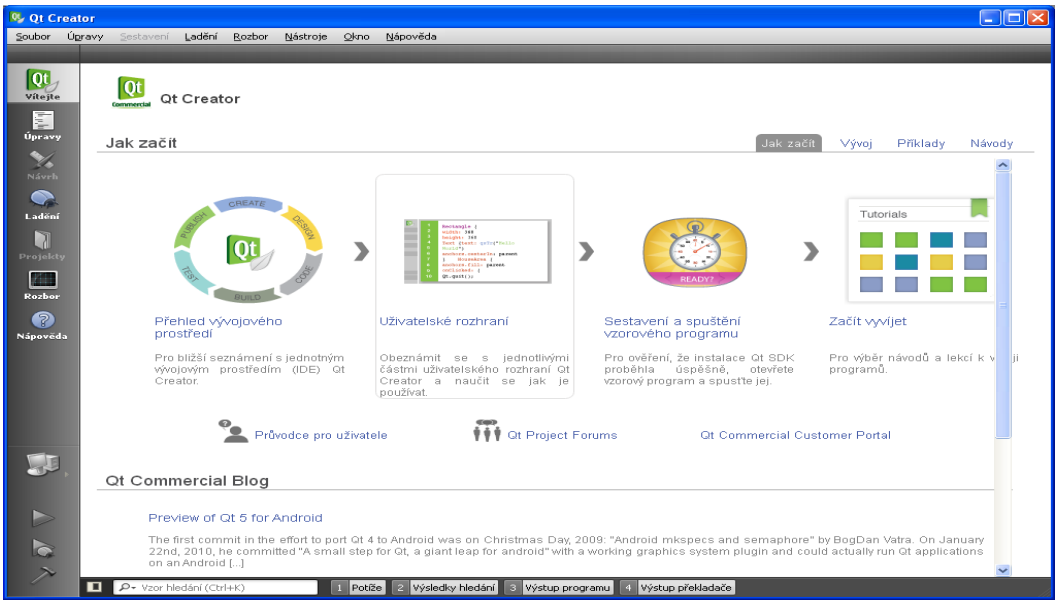
Je integrované vývojové prostředí IDE pro Qt framework (14). Zahrnuje podporu vývoje na platformě desktopové i serverové. Umožňuje tvorbu projektů - aplikací s i bez GUI rozhraní. Podporované jsou projekty založené na Qt C++ knihovně a na knihovně Qt Quick. Je zde ale možnost také vytvoření projektů na vývoj C/C++ bez použití Qt knihoven. IDE podporuje také distribuované systémy správy verzí jako Git, CVS, či Subversion.

IDE je multiplatformní, takže jej lze provozovat na všech, výše uvedených, podporovaných operačních systémech.

Jako celý framework Qt prochází i jeho komponenta Qt Creator rychlým vývojem. IDE bylo ve svých prvních verzích doporučováno pouze jako doplněk k již zavedeným IDE prostředím (viz kapitola 3.3). S přibývajícími vlastnostmi IDE Qt Creator „dospělo“ a v současné verzi je schopno do určité míry konkurovat nativním IDE jednotlivých OS a na poli multiplatformního IDE se stává významným „hráčem“.

3.4.4.5 *Tvorba projektů v prostředí IDE Qt Creator*

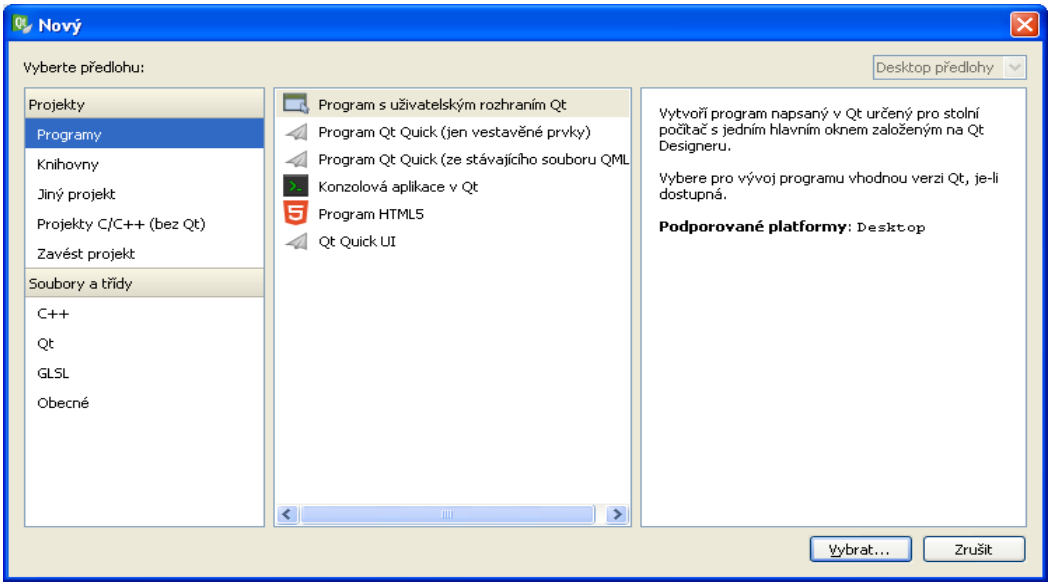
Úvodní obrazovka nabízí základní rozcestník. Přehled dokumentace a návody pro práci s vývojovým prostředím, vzorové příklady a záložka Vývoj, kde jsou rychlé odkazy



Obrázek 1: Úvodní strana IDE Qt Creator, zdroj: Digia Plc.

na vytvoření projektu a naposledy otevřené projekty (viz Obrázek 1).

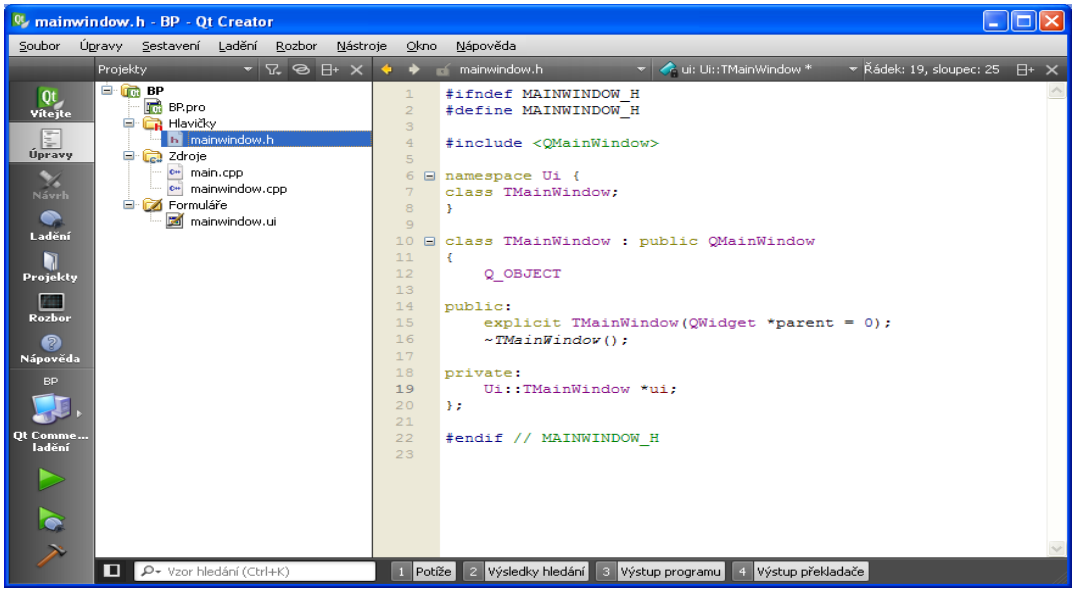
Z nabídky Soubor a výběru položky „Nový soubor nebo projekt“ lze vybrat z nabídky různých typů projektů (viz Obrázek 2).



Obrázek 2: Výběr nového projektu, zdroj: autor

Editor kódu

Po vytvoření projektu (dále jen projekt s uživatelským rozhraním Qt), a kliknutí na ikonu „Úpravy“ v levé části vertikální nástrojové lišty, lze v okně Projekty (viz Obrázek 3) volit mezi jednotlivými předgenerovanými zdrojovými soubory. Pokud se jedná o hlavičkové soubory, či o soubory implementační, zobrazuje se jejich obsah v pravé části editoru kódu. Editor podporuje zvýrazňování klíčových slov, doplňování kódu,

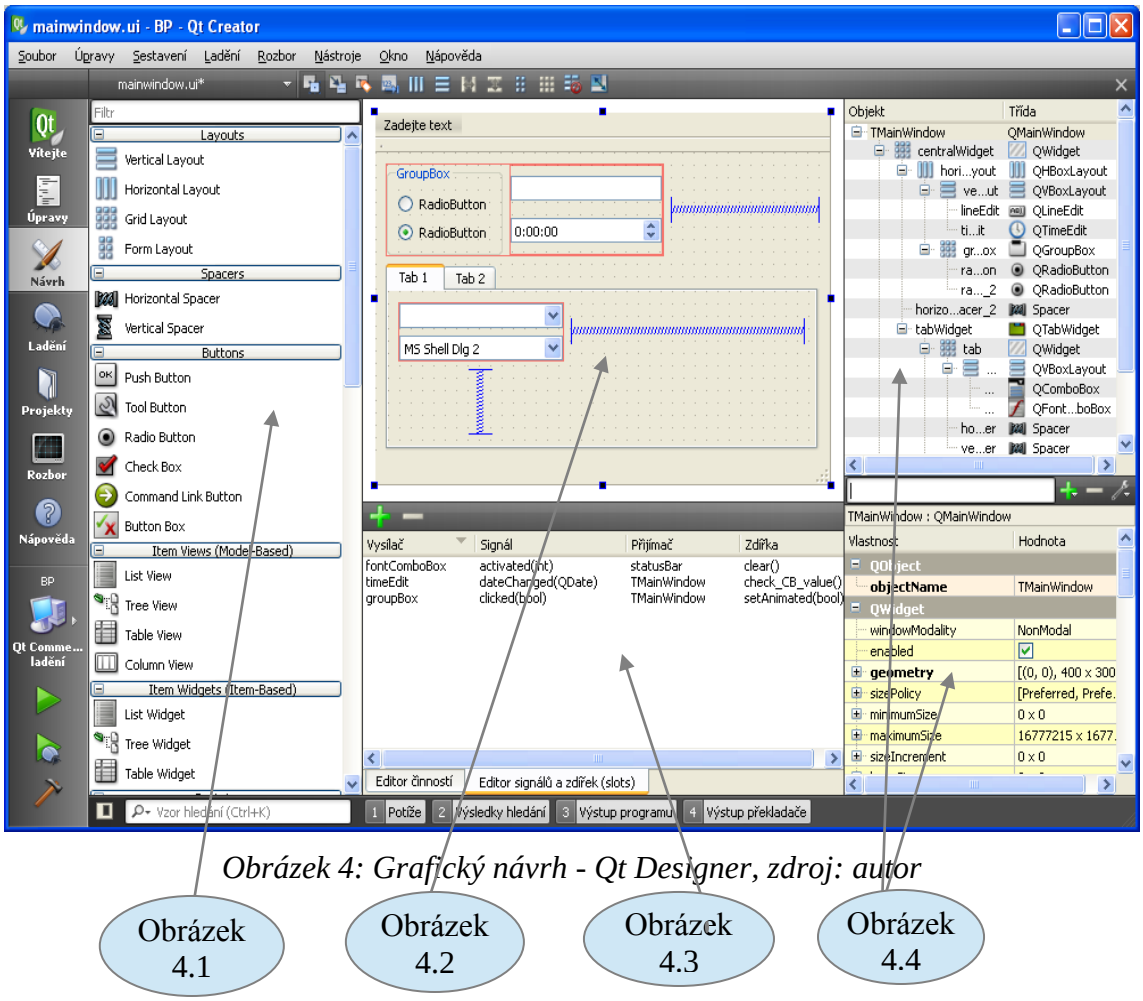


Obrázek 3: Strom projektu a editor kódu, zdroj: autor

upozorňování na chyby a další možnosti, které jsou známé pod zkratkou IntelliSense (15). Pomocí klávesy F1 lze kdykoliv vyvolat nad jednotlivými Qt třídami on-line (nebo off-line) nápovědu.

Grafický návrh

Grafický návrh v prostředí frameworku Qt se definuje XML popisnou strukturou, která se ukládá do souboru s příponou ui. Pokud v okně „Projekty“ poklikáme na soubor s touto příponou, v levé části nástrojové lišty dojde k automatické volbě - ikona „Návrh“ a celé IDE se přenastaví do módu pro modelování uživatelského rozhraní (dále jen designer) (možno spouštět i nezávisle jako komponentu Qt Designer) (viz Obrázek 4).



Obrázek 4: Grafický návrh - Qt Designer, zdroj: autor

Qt Designer se skládá ze čtyř základních částí. V centrální části designeru je plocha, na kterou se nanáší jednotlivé vizuální komponenty knihovny Qt a kde se modeluje vlastní návrh UI *(*viz Obrázek 4.2*)*.

V levé části se nachází *Widget manager* - okno s přehledem grafických komponent *(*viz Obrázek 4.1*)*. Ty lze metodou chytni a táhni (drag and drop) přetáhnou do centrální části pro konkrétní umístění daného grafického prvku. Je možno také definovat vlastní

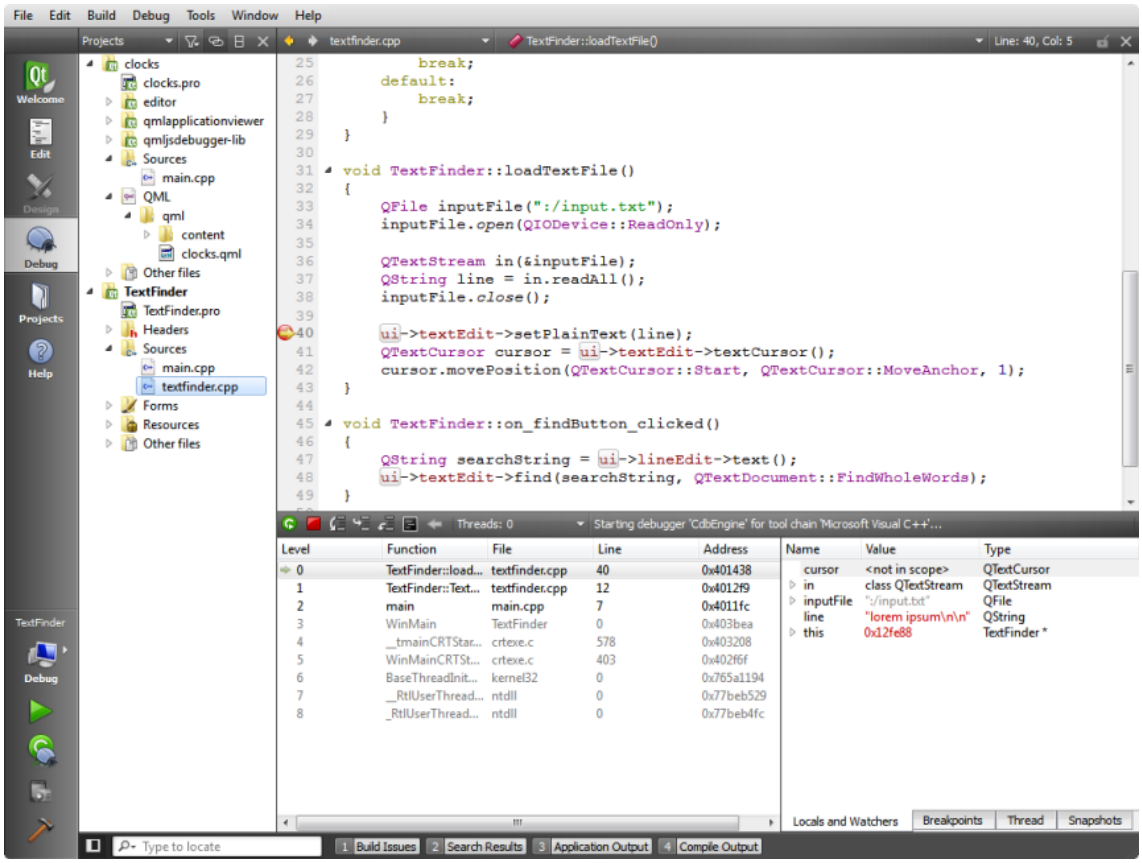
grafické prvky, které jsou potomky tříd Qt. S těmi pak lze pak pracovat jako s nativními Qt komponentami.

V pravé části designeru jsou dvě okna *(*viz Obrázek 4.4*)*. První je *Object Ispector*. Zobrazuje hierarchické uspořádání objektů aktuálního/navrhovaného UI. Pod tímto oknem se nachází *Property Editor*. Používá se pro zobrazení a modifikaci aktuálně vybraného grafického objektu.

Poslední částí je *Slot & Signal manager.* Slouží k projování – zasílání zpráv mezi jednotlivými grafickými komponentami (více viz Signály a Sloty) *(*viz Obrázek 4.3*)*

Ladění

Velmi důležitou částí IDE je komponenta „Ladění“ (debugger). Jedná se o modul, který umožňuje trasování- tzn. umožňuje analýzu běžícího aplikačního kódu. V okně editoru zdrojového kódu lze, v tomto módu, na každé řádce nastavit tzn. breakpoint. Breakpoint označuje místo, kde se běžící aplikace zastaví a je zde možno sledovat aktuální hodnoty lokálních i globálních proměnných, počet a stav běžících vláken a aktuální výpis zásobníku zvoleného vlákna *(*viz Obrázek 5*)*.



Obrázek 5: Ladění programu - debugger, zdroj: Digia Plc.

Nápověda

Další komponentou IDE je systém nápovědy a dokumentace (možno spouštět i nezávisle jako komponentu Qt Assitant). K Aktivaci je možno využít klávesu F1, kdy se zobrazí nápověda vedle okna editoru. Nebo je možno spustit celý modul kliknutím na ikonu „Nápověda“ ve vertikální nástrojové liště nebo přes nabídku Nápověda.

Rozbor

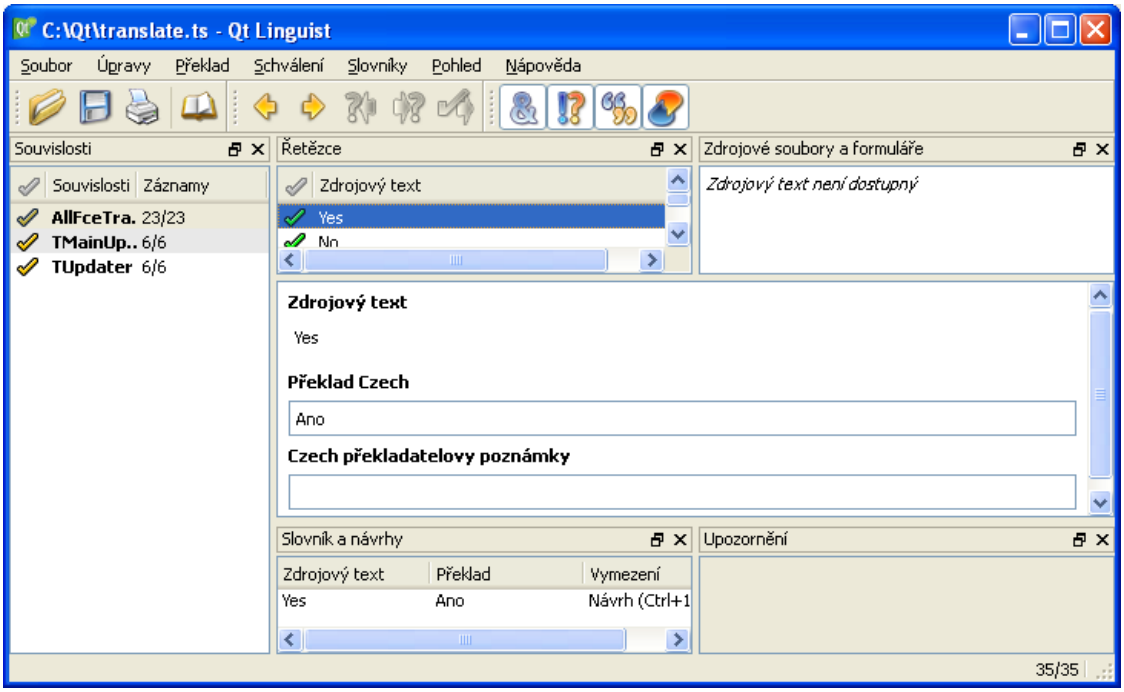
Ikona „Rozbor“ ve vertikální nástrojové liště umožňuje analyzování běhu programu. Analyzuje spojení uživatelských signálů a slotů a vykreslovací operace při běhu QML kódu. Pomocí externího nástroje Valgrind (16) je možno analyzovat úniky paměti, volání funkcí a chování vícevláknových aplikací (tento nástroj je nutno instalovat samostatně a je dostupný pouze na platformě OS Linux (na OS Windows již existuje neoficiální port v podobě balíku valgrind4win)).

Internacionalizace

Nástroj pro podporu jazykových překladů aplikací se nazývá Qt Linguist *(*viz Obrázek 6*)*.

Tento nástroj je možno spouštět samostatně, nebo z menu Qt Creatoru „Nástroje“ „Vnější“ a poté „Linguist“ (nebo samostatný nástroj Qt Linguist). Vyhledání řetězců k překladu ve zdrojových kódech slouží nástroj lupdate. Ten vytvoří soubor s příponou ts. Lexikálně hledá volání metody „tr“ což je statická metoda na třídě QObject.

Tím, že QObject je základní třída Qt hierarchie (více viz Popis Qt tříd) je tato metoda dostupná ze všech tříd frameworku Qt a zároveň se dostává (děděním) do každé nové uživatelské třídy, která je potomkem jakéhokoliv potomka třídy QObject.



Obrázek 6: Nástroj pro internacionalizaci - Qt Linguist, zdroj: autor

Soubor s příponou ts obsahuje XML kód *(*viz Obrázek 7*)*, kde je originální řetězec, který je ve zdrojovém kódu a k němu ekvivalentní jazykový překlad.

```
<?xml version="1.0" encoding="utf-8"?>
<TS version="2.0" language="cs_CZ">
<context>
    <name>TMyForm</name>
    <message>
        <source>Enter the system</source>
        <translation>Vstup do systému</translation>
    </message>
    <message>
        <source>User</source>
        <translation>Uživatel</translation>
    </message>
```

Obrázek 7: Zdrojový kód Internacionalizačních překladů, zdroj: autor

3.4.5 Popis Qt modulů a tříd

3.4.5.1 Moduly

Qt třídy je možno rozdělit do modulů určujících jejich použití. Lze je rozdělit do čtyř kategorií.

První kategorie nese název Moduly pro základní vývoj a obsahuje 16 modulů. Uvedu zde pouze ty nejdůležitější. Jsou to: QtCore, obsahující základní a negrafické třídy, dále QtGui obsahující komponenty grafického uživatelského rozhraní (GUI), QtXml pro

práci s XML, QtNetwork pro programování síťových aplikací a QtDeclarative obsahující komponenty pro práci v deklarativním prostředí QML.

Druhou kategoirí jsou moduly pro práci s Qt nástroji. QtDesigner (viz. kapitola QtDesigner 3.4.4.5), QtLinguist pro internacionalizaci, QtHelp pro online nápovědu a QtTest nástroje pro testování programových jednotek.

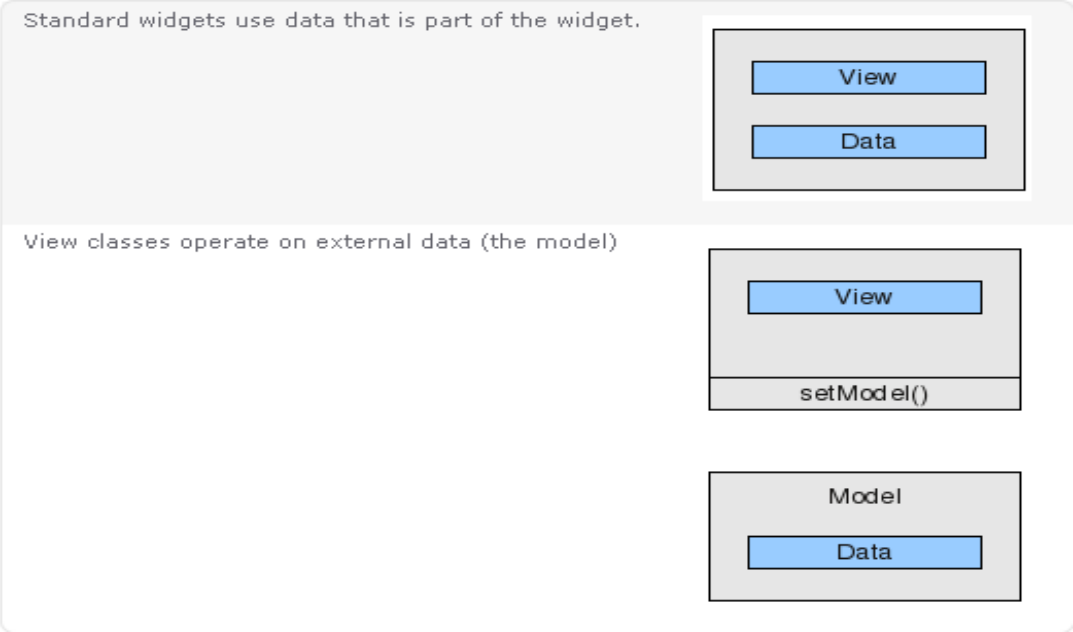
Třetí a čtvrtou kategorií jsou OS platformově závislé moduly pro vývojáře OS Windows a to QAxContainer a QAxServer pro práci s prvky ActiveX. Pro vývojáře platformy OS Linux je to modul QtDBus pro meziprocesovou komunikaci používající rozhradí D-Bus.

3.4.5.2 Třídy

V aktuální popisované vezi frameworku Qt (4.8.4) je cca 850 tříd. Jejich základní kategorizování bylo nastíněno v kapitole (viz kapitola 3.4.5.1 Moduly). Není účelem této práce detailně rozebírat funkční náplň těchto tříd. Přesto bych rád několik z nich, nebo technologií které pokrývají, zmínil.

Qt třídy nabízejí podobnou šíři použití jako STL (Standard Template Library) knihovna (17). Jsou navržené tak, aby právě s touto standardní knihovnou dokázali účelně a rychle pracovat. Implementačně je to formou přetížených konstruktorů v kontejnerových třídách (jako např. QList, QVector, QMap či QSet).

Další velmi užitečnou a moderní technikou je používání konceptu Model-View-Controller (MVC) programování (1). MVC je návrhový vzor převzatý z jazyka Smalltalk. Nejčastěji se používá, pokud vytváříme GUI. Tato technologie odděluje data od jejich vizuálních prezentací *(*viz Obrázek 8*)*. Tím umožněna fexibilnější implementace a zároveň případné vícenásobné použití kódu při změně prezentační nebo datové vrstvy. Objekty v modelu MVC lze rozdělit do tří částí. Model představuje zdrojová data a metody. Je to neprezentační/nevizuální část. View naopak představuje uživatelské rozhraní, neobsahuje žádná zdrojová data a slouží pouze k prezentaci dat z vrstvy Modelu. Třetí část je Controller, který v knihovně Qt pomocí mechanismy signálů a slotů, zajišťuje spojení/notifikaci mezi Modelem a View (například pokud dojde ke změně dat Modelu je nutno změnit i jejich prezentační vyjádření neboli View vrstvu).



Obrázek 8: Schéma MVC modelu, zdroj: Digia Plc.

Nelze nezmínit podporu vícevláknového programování. Tyto možnosti primárně pokrývá třída QThread a pomocné třídy zajišťující serializovaný přístup k datům jako QWaitCondition, QMutex, QSemaphore, QRunnable a další. Zde je vhodné zmínit možnost využití předávání meziprocesových zpráv pomocí standardní interní komunikace mezi Qt objekty – tedy pomocí signálů a slotů (viz kapitola 3.5.4 Události).

Jako poslední bych rád zmínit knihovnu pracující se značkovacími jazyky. Jazyk HTML je již přímo podporován Qt grafickými objekty, které se nazývají widgety. Pomocí CSS stylů je jim možno nastavovat grafický vzhled (barva, rozměry atd). Dalším moderním značkovacím jazykem, který Qt framework podporuje je XML. XML je možno zpracovávat metodami SAX i DOM (18), validovat dokumenty vůči DTD či Schematům. V neposlední řadě implementuje (ne úplně) specifikaci dotazovacího jazyka XQuery 2.0 a Xpath (18). Přesto použití XSLT transformací je i s tímto omezením široce použitelné. Se značkovacími jazyky také souvisí modul QtWebKit, který s využitím renderovacího jádra WebKit nabízí plnou podporu pro webové vývojáře.

3.5 Meta-Object system

Meta-Object systém představuje způsob získávání a použití metadat. Umožňuje použití mechanismu signálů a slotů pro interní meziobjektovou komunikaci a management run-time informací o vestavěných či uživatelských typech.

Skládá se ze tří částí. První část je základní třída QObject. Druhou částí je makro Q_OBJECT a třetí je Meta-Objet compiler. Součinnost všech těchto částí dává dohromady možnost plného využití metadat jádrem Qt frameworku.

3.5.1 *QObject*

QObject je základní třída tj. předek všech Qt tříd. Poskytuje standardní rozhraní, které využívají všechny poděděné uživatelské třídy. Je to základní stavební prvek (implementuje prvky, vlastnosti a schopnosti) Meta-Object modelu.

Jak již bylo zmíněno výše, při programování v nemanage kódu (kdy se o práci s pamětí plně stará programátor) dochází k problémům s memory managementem ve všech jeho aspektech. Qt framework tento „nedostatek“ částečně řeší tím, že všechny konstruktory Qt objetků obsahují jako jeden z možných parametrů parametr parent typu QObject. Pokud vytvořím dynamicky alokovaný objekt, který je potomkem třídy QObject a ukazatel na něho předám v konstruktoru dalšího (druhého) potomka třídy QObject, pak v případě uvolňování (dealokaci) druhého objektu je zároveň uvolněn i první objekt bez nutnosti programátora starat se o uvolnění prvního objektu.

3.5.2 *Makro Q_OBJECT*

Abychom mohli plně využívat interního sytému meta dat, je nutno v deklaraci nové třídy deklarovat makro Q_OBJECT. Touto deklarací se při následném generování kódu zpřístupní meta-objektová funkcionalita, jako jsou dynamické vlastnosti objektů či mechanismus signálů a slotů (viz. Kapitola 3.5.5. Signály a Sloty).

3.5.3 *Meta-Object compiler (moc)*

Mechanismus signálů a slotů by nefungoval bez „prostředníka“, kterým je Meta-Object compiler (dále jen moc). Moc je nástroj, který čte hlavičkové (deklarační) soubory. Pokud v deklaraci třídy nalezne makro Q_OBJECT vygeneruje nový C++ zdrojový soubor pojmenovaný jako hlavičkový soubor pouze s předponou „moc_“. Ten obsahuje Qt meta-object kód pro danou třídu. Soubor obsahuje implementaci potřebnou pro chod mechanismu signálů a slotů, informace o run-time typech (uživatelsky definovaných) a dynamických vlastnostech dané třídy.

Tento soubor se stává součástí aplikace (projektu). Je přeložen a slinkován spolu s ostatním implementačním kódem do podoby spustitelného souboru či knihovny.

3.5.4 *Události*

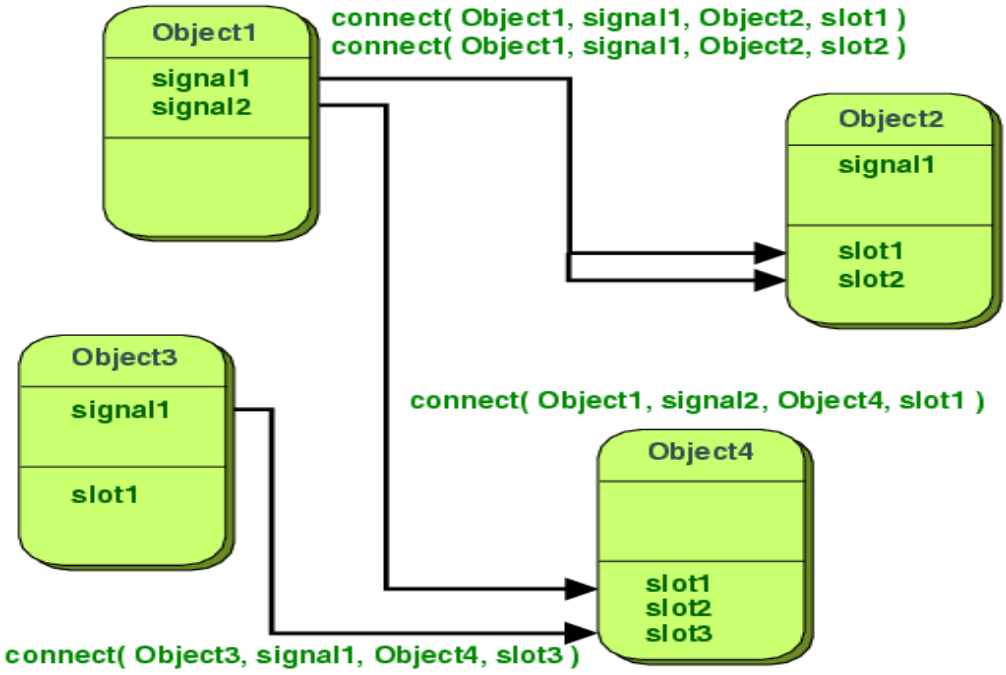
Události se v prostředí frameworku Qt šíří pomocí zasílání zpráv interním objektem typu QEvent. Při startu aplikace se inicializuje jádro knihovny, které zpracovává příchozí události.

Ty mohou nastat dvojího typu: Interní, což jsou události generující vlastní framework a nebo uživatelsky (programově) definované (generované). Druhý druh událostí jsou externí. Tyto události generuje okolí frameworku Qt. Framework oba druhy událostí zpracovává a podle potřeby na ně reaguje příslušným způsobem ošetřením. Ošetření se děje vyhodnocením události a následným zasláním zprávy do příslušných vrstev frameworku, kde se finálně zpracují anebo se dále propagují do uživatelského rozhraní (GUI), kde na ně může reagovat uživatel.

3.5.5 *Signály a sloty*

Signály a sloty slouží ke komunikaci mezi Qt objekty. Mechanismus signálů a slotů je základní vlastností Qt frameworku (19). Tento mechanismus zasílání zpráv/komunikace se zásadně liší od způsobů implementovaných ostatními frameworky. Ty řeší události, např. typu zmačknutí tlačítka a reakcí na ně, komunikací typu callback kdy ukazatel na funkci je předán do funkce, od které následně očekáváme callback tj. zpětné volání z důvodu vyřízení zprávy.

Qt framework používá ke stejnému účelu mechanismus signálů a slotů (viz Obrázek 9). Signál (událost) se začíná propagovat ve chvíli, kdy se zdrojový objekt rozhodl notifikovat své okolí o nějaké události. Signál je iniciátor události (volající) a slot je obsluha události (volaný). Qt objekty již obsahují signály a sloty, které jsou potřebné k jejich interní vzájemné komunikaci (viz kapitola 3.5.1 QObject). Ty se dědí, takže každý potomek je schopen vyslat signál či „automaticky“ obsloužit slot předka. Signály a sloty jsou typové, takže při pokusu o navázání mezi sebou musí souhlasit počet i typy parametrů.



Obrázek 9: Schéma mechanismu signálů a slotů, zdroj: Digia Plc.

Stejný způsob komunikace jaký používají interní objekty frameworku Qt lze využít také při implementaci vlastního aplikačního kódu. Při deklaraci tříd je nutno novou třídu podědit od třídy QObject, nebo jakéhokoliv jeho potomka (např. QWidget, QThread atd.). Dále použít makro Q_OBJECT a při deklaraci metod (signály a sloty jsou metody) použít klíčová slova slots a signals (viz Obrázek 10).

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT

public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }

public slots:
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

Obrázek 10: Deklarace třídy s použitím makra Q_OBJECT, zdroj: Digia Plc.

Definice slotu se ničím neliší od deklarace standardní instanční metody (viz Obrázek 11) a je ho možno také, jako běžnou metodu, volat přímo.

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

Obrázek 11: Definice slotu (metody), zdroj: Digia Plc.

Spojení signálu a slotu se děje pomocí třídní statické metody předka tj. QObject::connect(). Rozpojení opět třídní metodou QObject::disconnect(). Tato metoda je přetížená a existuje tedy v několika verzích.

Zde na obrázku (viz Obrázek 12) je prvním parametrem ukazatel na objekt odesilatele. Druhým je signál, který bude emitován. Třetí je ukazatel na objekt příjemce a čtvrtým je slot příjemce, který bude použit k ošetření události v případě emitování signálu.

```
Counter a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));

a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);     // a.value() == 12, b.value() == 48
```

Obrázek 12: Spojení signálu a slotu, zdroj: Digia Plc.

Jako dalším parametrem metody QObject::connect(), na obrázku neprezentován, je informace jakým způsobem má být spojení provedeno. Synchronně, asynchronně a nebo, zda způsob volání nechat na rozhodnutí frameworku Qt (což je defaultní hodnota). Slot je standardně volán po emitování signálu synchronně, tzn. dochází k přímému volání metody, což se děje v rámci běhu jednoho subprocesu (vlákna/threadu). Asynchronní volání je použito tehdy, pokud dochází k volání mezi vlákny nebo pokud z nějakého důvodu potřebujeme asynchronní volání i v rámci jednoho vlákna.

4 Vlastní práce

4.1 Analýza aplikace

Druhou částí této práce je vytvoření multiplatformní aplikace v prostředí frameworku Qt s ukázkou využití a použití jednotlivých nástrojů prostředí Qt Creator. Jak již bylo nastíněno výše možnosti použití frameworku Qt jsou široké. Rozhodl jsem se blíže představit jeho možnosti při práci s XML a jeho transformací do HTML. Projekt nese název XSLT Processor.

Jedná se transformační XSLT processor jehož výstupem je transformovaný soubor ve formátu HTML. Zadání nepřímo vychází z potřeby transformací, které byly obsahem předmětu „Značkovací jazyky“ vyučovaném ve IV. semestru.

4.1.1 *Základní analýza a návrh uživatelského rozhraní*

Uživatelské rozhraní je koncipováno dle minimálního počtu vstupních dat, která jsou nutná pro chod transformačního procesu, resp. které jsou nutné pro správné volání příslušných instančních metod tříd frameworku Qt. Na vstupu bude od uživatele požadován odkaz na „datový“ soubor XML a transformační soubor XSL. Výstupem je pak transformovaný soubor ve formátu HTML, který bude zobrazen na centrální ploše aplikace. Případné chyby běhu transformace (a nejen ty) budou zobrazovány do samostatného chybového okna. O transformaci se bude starat speciální třída, jejíž transformační metoda obdrží na vstupu výše uvedené cesty ke zdrojovým souborům a výstupem bude transformovaný dokument. Vlastní transformace bude spouštěna tlačítkem k tomu určenému. Aplikace se bude ukončovat samostatným tlačítkem (či standardním způsobem ukončování vizuálních aplikací). Bude psána pro multiplatformní použití, anglickém jazyce a pomocí internacionalizačních nástrojů přeložena do českého jazyka.

4.1.2 *Popis zdrojového kódu*

Aplikace se skládá z grafického návrhu uloženého v souboru tmainwindow.ui. Grafický návrh je reprezentován popisným dokumentem XML. Obsahuje grafické komponenty frameworku Qt tzn. jejich typy, umístění na ploše aplikace, velikosti, identifikátory, uživatelsky definované signály a sloty atd (viz Obrázek 13).

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>TMainWindow</class>
 <widget class="QMainWindow" name="TMainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>708</width>
    <height>475</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>XSLT Processor</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QGridLayout" name="gridLayout_4">
    <property name="leftMargin">
     <number>7</number>
    </property>
```

Obrázek 13: Zdrojový kód grafického návrhu, zdroj: autor

Soubor resource.qrc obsahuje ikony, které se v aplikaci používají. Ikony v aplikaci mohou být uloženy buď jako součást distribuce aplikace (jako fyzické soubory), nebo jak bylo zvoleno v tomto případě, ikony se jako datový zdroj stávají součástí spustitelného souboru a není je tudíž nutné explicitně distribuovat.

Soubory s příponou „ts“ a „qm“ jsou soubory, které obsahují internacionalizační překlady (viz kapitola 4.2.2.3 Internacionalizace).

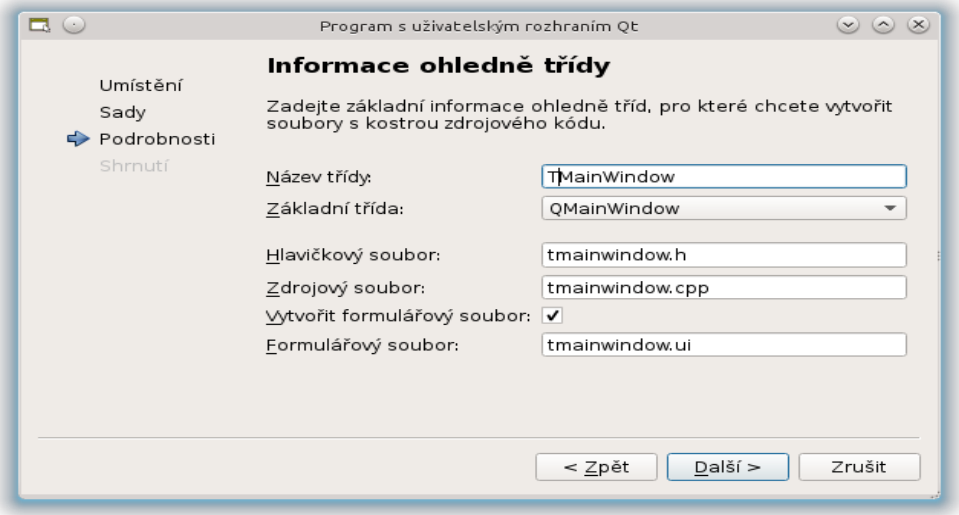
Projekt dále obsahuje zdrojové C++ kódy aplikace. Jsou v podobě tzv. hlavičkových souborů s příponou „h“, které obsahují deklarace tříd, metod a funkcí. K nim patří soubory stejných jmen s příponou „cpp“ obsahující definice tzn. vlastní implementace/definice deklarací uvedených v hlavičkových souborech. Speciálním případem deklarace je soubor main.cpp. Ten obsahuje pouze jednu funkci s názvem main(), která slouží jako entry point programu, nebo-li touto funkcí se začíná po spuštění aplikace provádění uživatelského kódu.

4.2 Aplikace

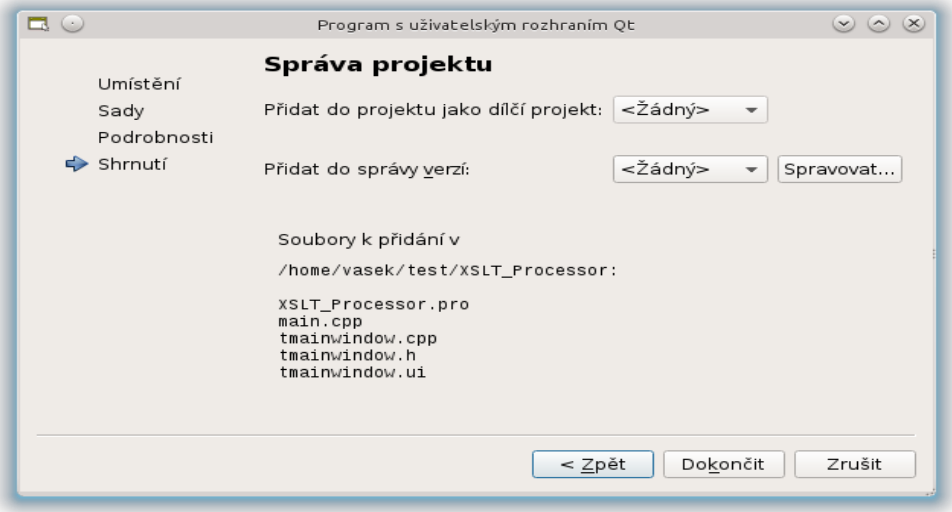
V předchozí kapitole byl předestřen účel vzorové aplikace, provedena základní analýza a popis návrhu uživatelského prostředí. Byly přehledově představeny hlavní komponenty aplikace a popsány zdrojové soubory, ze kterých se skládá. V této kapitole budou popsány jednotlivé kroky vývoje aplikace.

4.2.1 *Založení nového projektu*

Aplikace byla vyvíjena v prostředí IDE QtCreator s českou lokalizací (viz kapitola 3.4.4. Qt Creator). Z nabídky Soubor a poté „Nový soubor nebo projekt“ byl vytvořen nový projekt. Vybrán byl typ projektu „Program s uživatelským rozhraním Qt“. Poté zadáme jméno a umístění projektu. Projekt nese název XSLT Processor. Vybereme dále název námi nově definované třídy (TMainWindow) a k ní Qt základní třídu, která bude předkem této námi definované (ponecháme QMainWindow). Dále zkontrolujeme, zda je zaškrtnuta volba „Vytvořit formulářový soubor“ (viz Obrázek 14).



Obrázek 14: Vytvoření projektu vzorového příkladu, zdroj: autor



Obrázek 15: Finální dialog vytvoření nového projektu, zdroj: autor

Všechny tyto volby následně potvrdíme a projekt je tímto založen (viz Obrázek 15).

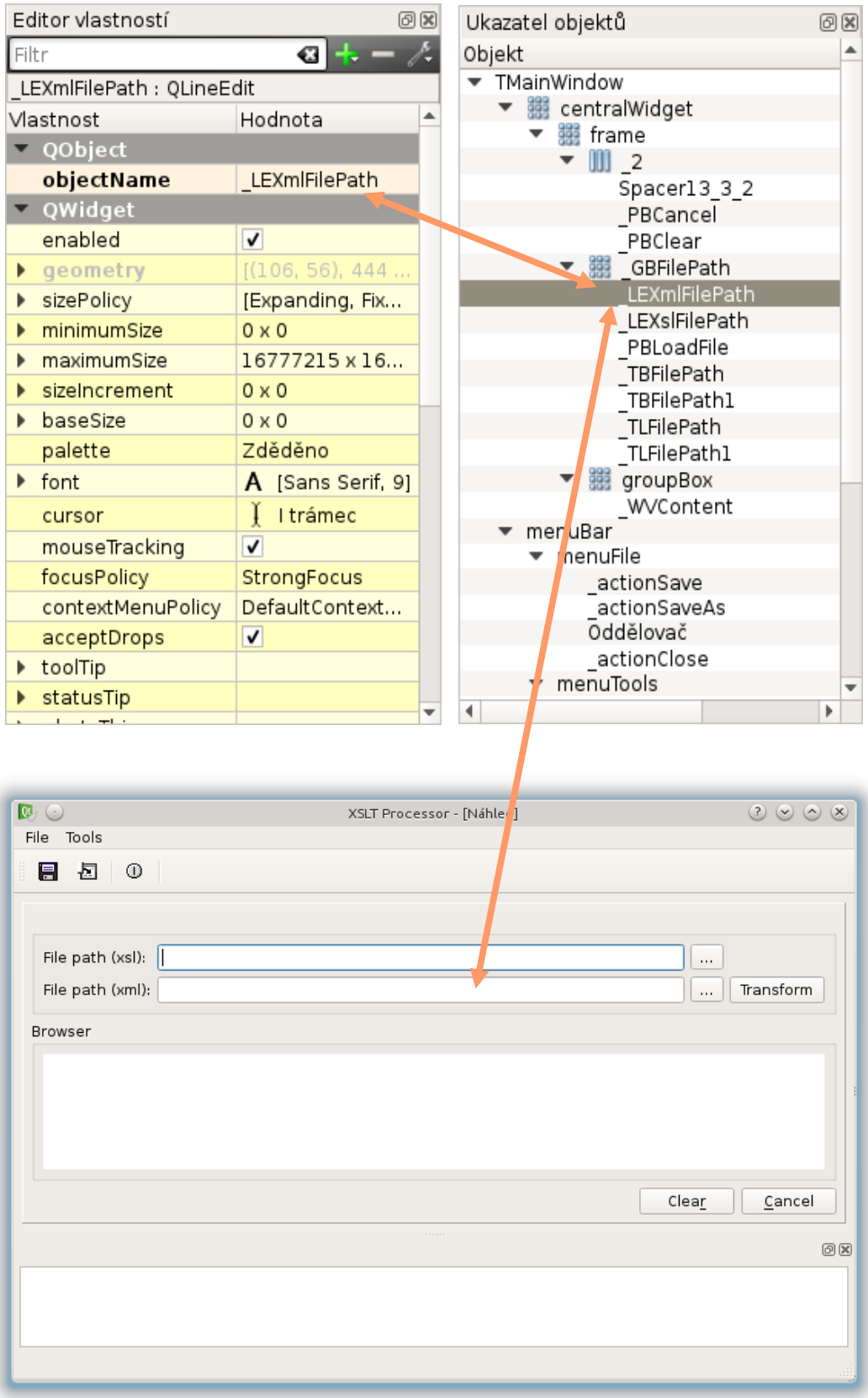
4.2.2 *Modelování grafického rozhraní aplikace (GUI)*

Nově vytvořený projekt obsahuje složky: Hlavičky, Zdroje, Formuláře, Prostředky. V nich se předgenerují jednotlivé deklarační a definiční soubory a GUI návrhový soubor. Po poklikání (či kliknutí v závislosti na OS, na kterém je aplikace vyvíjena) se otevře pracovní plocha Qt Designeru.

Z levé části vybíráme jednotlivé požadované grafické komponenty a metodou táhni a pusť (drag and drop) je postupně přesouváme na grafický panel. Zde lze vybrané objekty pozičně umisťovat zvětšovat a zmenšovat či seskupovat pomocí myši. Každému takovémuto objektu je možno nastavit hodnotu jeho atributů. Množina modifikovatelných atributů se mění v závislosti na zvoleném typu grafického objektu (např. textové pole vz. menu). Okno s atributy tříd je standardně umístěno v pravé dolní části okna designeru (viz Obrázek 16).

S přidávanými grafickými prvky se zároveň jednotlivé objekty stromově zobrazují v na pravé straně v horní části. Zde se z nich vytváří strom dle vrstev, jak byly umisťovány na grafický panel (viz Obrázek 16).

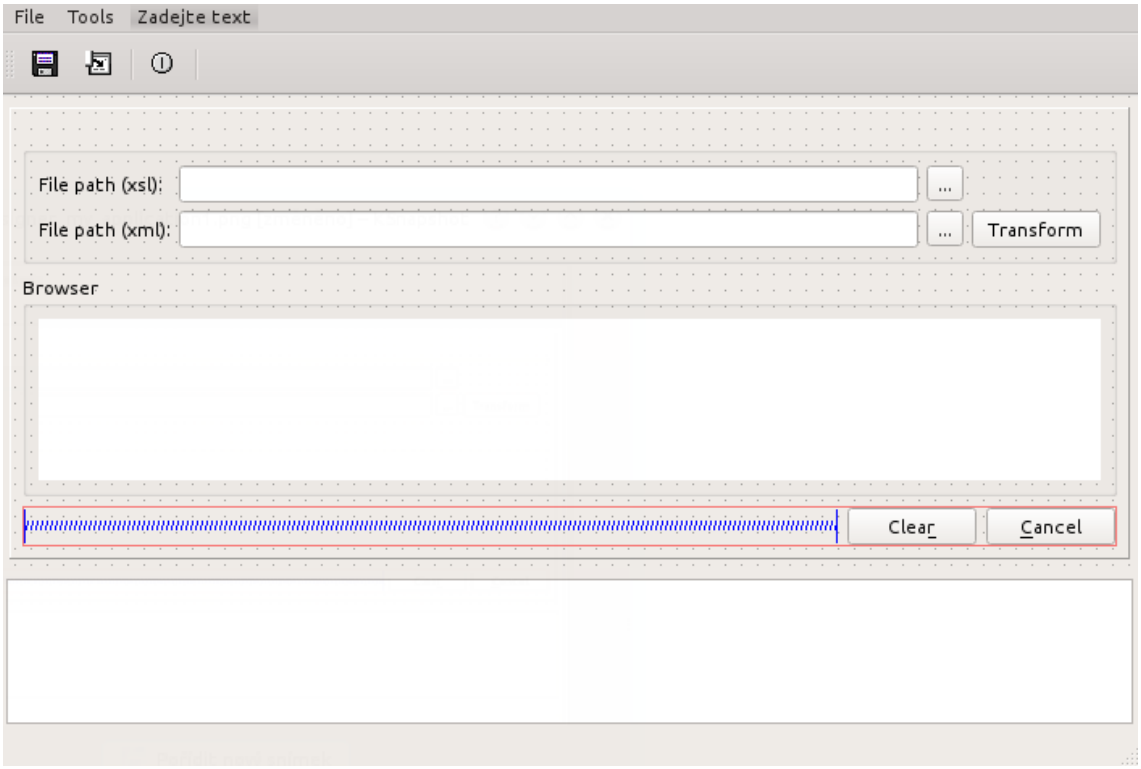
Výsledný návrh GUI je na obrázku (viz Obrázek 16). Šipkou je zde naznačeno projení mezi jednotlivými komponentami designeru. Od atributového popisu grafické komponenty, přes její hierarchické umístění až po výsledné umístění na grafickém panelu.



Obrázek 16: Propojení atributů objektu v grafickém návrhu, zdroj: autor

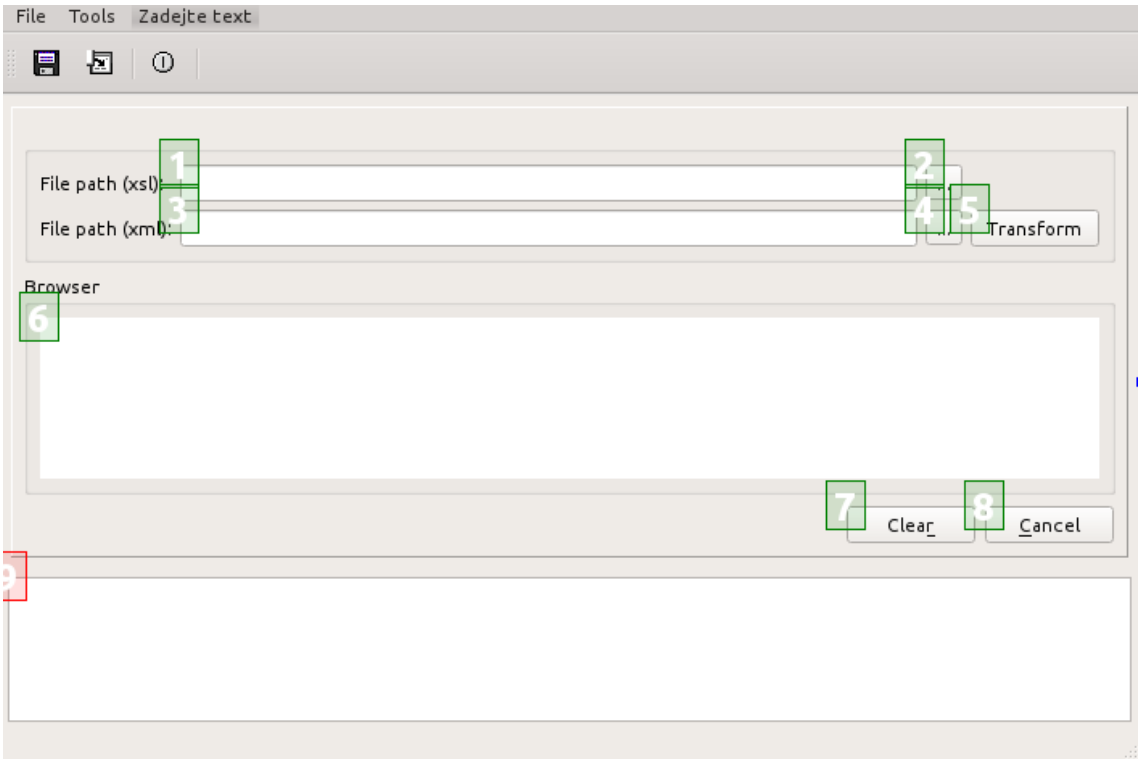
4.2.2.1 *Layout a pořadí tabulátorů*

Pomocí tzv. layoutů (rozvržení) lze objekty seskupovat horizontálně, vertikálně, formulářově, či tabulkově (na střed). Takto seskupené objekty při změně své velikosti respektují daný layout a v rámci něho upravují svoje velikosti a pozice. Ne každý layout je graficky reprezentován. V návrhu vzorové aplikace je viditelný pouze horizontální layout v centrální oblasti formuláře kdy jsou spojené tlačítka Clear, Cancel a horizontální spacer (viz Obrázek 17). Ostatní objekty jsou zarovnány do tabulkového layoutu.



Obrázek 17: Použití layoutů v grafickém návrhu, zdroj: autor

Každý grafický objekt je schopen přijmout takzvaný focus (zaměření). Tím je uživateli umožněn, či omezen, vstup do vstupních polí. Focus může být několika druhů. (Tab, Mouse, Wheel, atd.). Tzv. Tab focus umožňuje pohyb uživateli po formuláři, pokud k pohybu použije klávesu Tab. Pořadí posouvání se po formuláři lze ovlivnit v návrhu graficky. Po přepnutí do příslušného módu Qt Designeru postupným klikáním na jednotlivá čísla objektů určujeme zároveň Tabulátorové pořadí (viz Obrázek 18).



Obrázek 18: Grafická editace tab orderu v grafickém návrhu, zdroj: autor

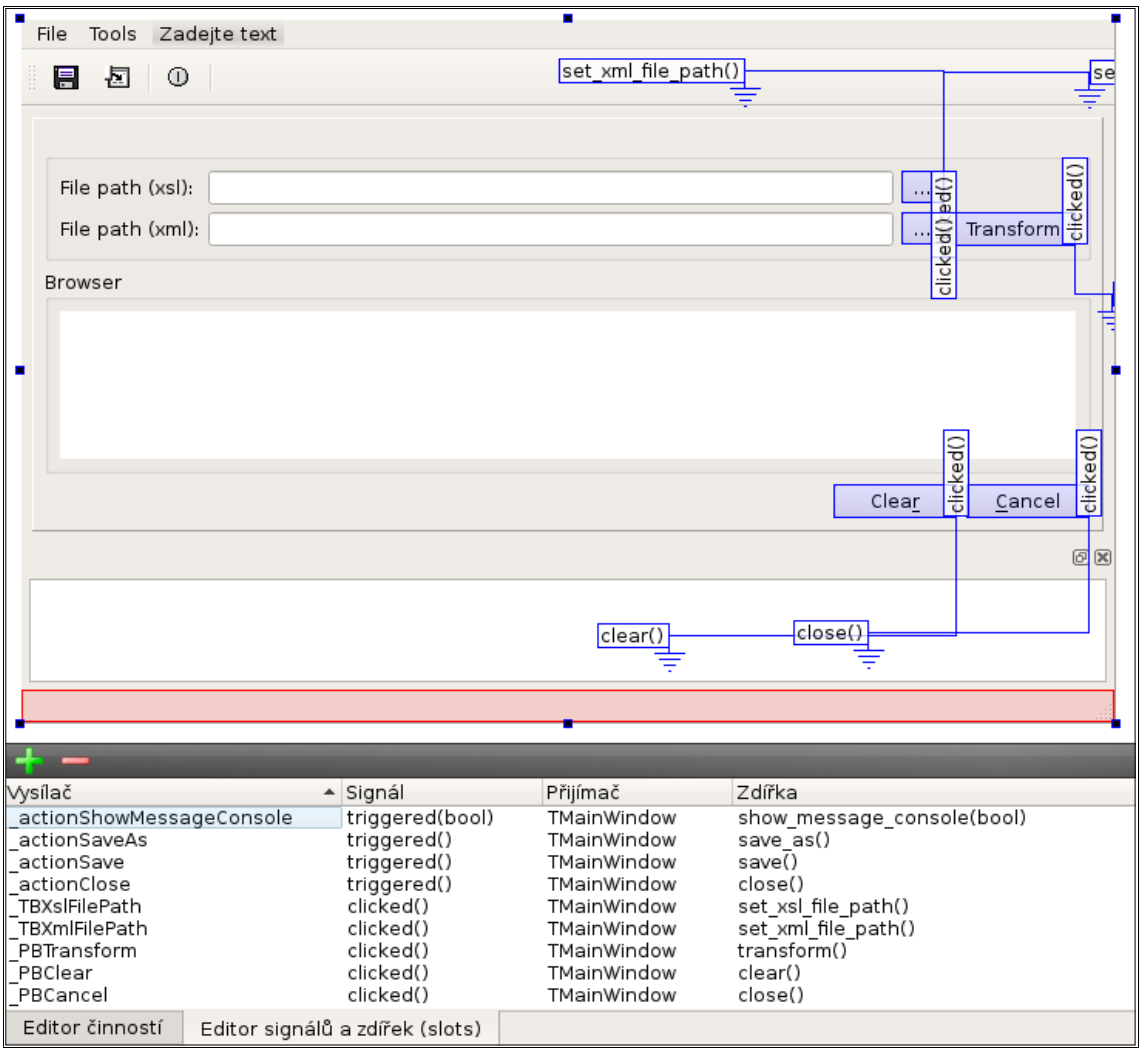
4.2.2.2 *Signály a sloty při návrhu GUI*

V kapitole (viz kapitola 3.5.5 Signály a Sloty) o signálech a slotech byl již význam mechanismu signálů a slotů popsán a prezentován ve formě C++ kódu. Signály a sloty je možno také využívat v prostředí návrhu GUI. Spojovat lze signály a sloty již definované, nebo uživatelsky přidané. Qt Designer přepneme do režimu spojování signálů a slotů, kdy metodou táhnutím ze zdrojového do cílového objektu (graficky prezentovaného čarou) určíme objekt zdroje (signálu) a objekt cíle (slotu). Na těchto objektech pak vybereme příslušné metody (signály a sloty), které spojíme (viz Obrázek 19).

V našem vzorovém příkladě byly přidány uživatelské sloty na třídě TMainWindow. Jsou to například metody set_xsl_file_path(), set_xml_file_path(), transform() a další. Tyto metody byly napojeny na signály vycházící z konkrétních objektů. Například tlačítko Transform po stisknutí vysílá vestavěný signál clicked(), který je napojený na uživatelský slot transform(). Tento slot (metoda) poté již spustí validaci vstupních dat a provádí vlastní transformaci.

Uživatelsky přidané sloty jsou pouze grafickou reprezentací budoucích spojení mezi instancemi objektů. Je tedy nutno tyto sloty (metody) v příslušných třídách deklarovat a definovat (provést implementaci).

Další signály a sloty, které aplikace využívá, jsou již ekvivalentní k výše uvedenému mechanismu spojení slotu transform(). Na obrázku (viz Obrázek 19) je vidět finální grafickou i textovou podobu spojených signálů a slotů.



Vysílač	Signál	Přijímač	Zdířka
_actionShowMessageConsole	triggered(bool)	TMainWindow	show_message_console(bool)
_actionSaveAs	triggered()	TMainWindow	save_as()
_actionSave	triggered()	TMainWindow	save()
_actionClose	triggered()	TMainWindow	close()
_TBXslFilePath	clicked()	TMainWindow	set_xsl_file_path()
_TBXmlFilePath	clicked()	TMainWindow	set_xml_file_path()
_PBTransform	clicked()	TMainWindow	transform()
_PBClear	clicked()	TMainWindow	clear()
_PBCancel	clicked()	TMainWindow	close()

Editor činností | Editor signálů a zdířek (slots)

Obrázek 20: Spojení signálů a slotů v grafickém návrhu, zdroj: autor

```
<connection>
 <sender>_TBXslFilePath</sender>
 <signal>clicked()</signal>
 <receiver>TMainWindow</receiver>
 <slot>set_xsl_file_path()</slot>
 <hints>
  <hint type="sourcelabel">
   <x>604</x>
   <y>95</y>
  </hint>
  <hint type="destinationlabel">
   <x>692</x>
   <y>35</y>
  </hint>
 </hints>
</connection>
```

Obrázek 19: Zdrojový kód designu grafických objektů, zdroj: autor

Grafický návrh ukládaný do souboru s příponou ui tato spojení obsahuje v textové podobě XML, která popisuje jednotlivé signály a sloty a jejich propojení mezi budoucími instancemi objektů (viz Obrázek 20).

4.2.2.3 *Internacionalizace aplikace*

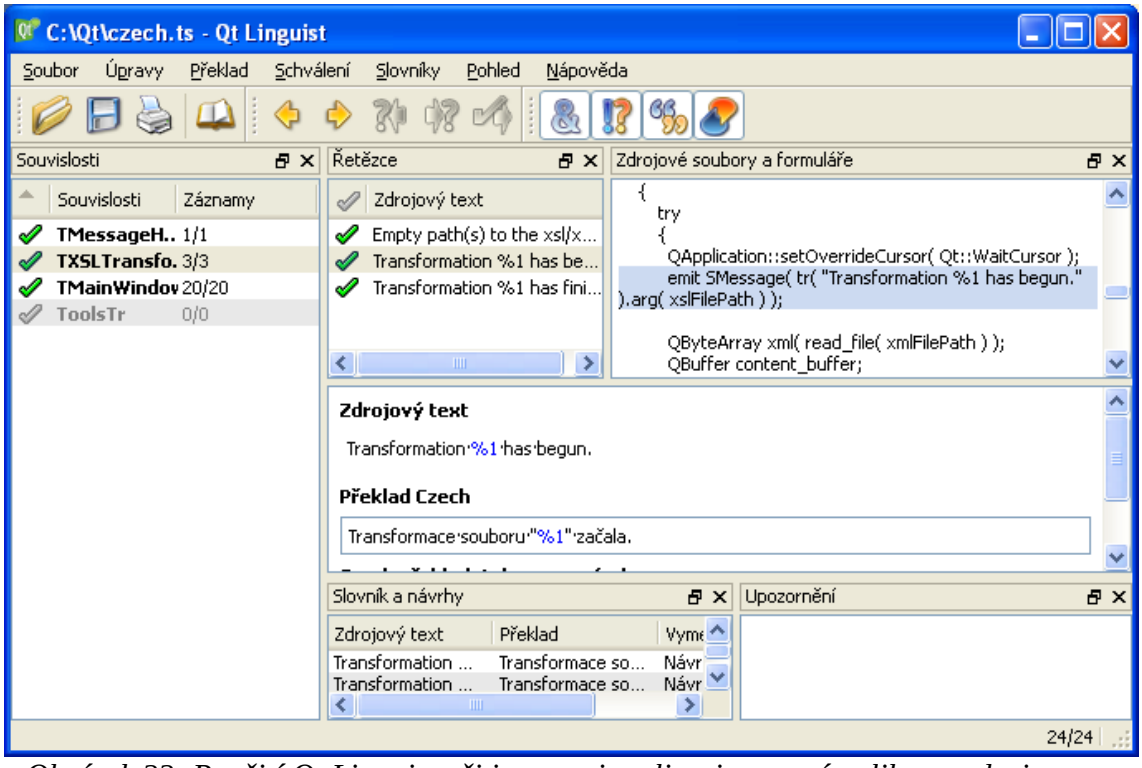
Framework Qt poskytuje nejenom možnost multiplatformního vývoje ale také podporu a nástroje pro vývoj vícejazyčných aplikací.

Všechny konstantní řetězce, které budou vygenerovány z grafického návrhu aplikace jsou automaticky „obaleny“ statickou metodou třídy QApplication::translate(). Stejného výsledku dosáhneme, pokud ve své implementaci použijeme metodu tr(), která je zděděná ze třídy QObject (viz Obrázek 21).

```
try
{
  QApplication::setOverrideCursor( Qt::WaitCursor );
  emit SMessage( tr( "Transformation %1 has begun" ).arg( xmlFilePath )
```

Obrázek 21: Použití internacionalizace v kódu, zdroj: autor

Do konfigurace projektu přidáme řádku TRANSLATIONS = czech.ts. Poté z nabídky Nástroje->Vnější->Linguist vybereme „Aktualizovat překlady (lupdate)“. Tím se ze všech zdrojových kódů vyberou řetězce, které jsou určené pro překlad a budou uloženy do námi zvoleného souboru czech.ts. Následně spustíme Qt Linguist (viz Obrázek 22) a doplníme český překlad k anglickému. Nakonec znovu z menu Nástroje->Vnější->Linguist spustíme „Vydat překlady (lrelease)“. Tímto XML soubor czech.ts přeložíme na jeho binární podoby a tím vznikne soubor czech.qm.



Obrázek 22: Použití Qt Linguist při internacionalizaci vzorové aplikace, zdroj: autor

Tímto způsobem je vzorová aplikace přeložena do českého jazyka. Abychom mohli lokalizovat celé prostředí Qt (např. dialogy otevírání souborů, informační zprávy Qt XSLT procesoru atd.) potřebujeme do naší aplikace přidat také systémové překlady, které jsou již připraveny a jsou součástí distribuce Qt frameworku. Jedná se o soubory qt_cs.qm, qtbase_cs.qm a qtxmlpatterns_cs.qm. Instalace překladů se provádí na objektu třídy QApplication ideálně v metodě main(), v souboru main.cpp, při startu aplikace (viz Obrázek 23).

```
//install all kinds (library's) translations
//this app czech translations
QTranslator *translator = new QTranslator();
translator->load("../XSLT/czech.qm");
a.installTranslator(translator);
//qt czech translations
QTranslator *qttranslator = new QTranslator();
qttranslator->load("../XSLT/qt_cs.qm");
a.installTranslator(qttranslator);
```

Obrázek 23: Instalace internalizačních překladů, zdroj: autor

Výsledkem instalace všech překladových souborů je plná jazyková lokalizace do českého jazyka. V případě potřeby jiné/další jazykové lokalizace je pouze potřeba v prostředí Qt Linguist provést překlady z anglického do dalšího požadovaného jazyka. Následně provést stejnou posloupnost kroků jako s výše popisovaným českým překladem. Proces změny jazykového prostředí je možno v aplikaci provádět i dynamicky mezi více jazykovými lokalizacemi.

4.2.3 *Zdrojový kód*

Zároveň s návrhem GUI je potřeba implementovat navržené třídy a funkce. Zdrojový kód lze rozčlenit na tři části: Třídu grafického rozhraní, transformační třídu a ostatní obslužné třídy a funkce.

4.2.3.1 Třída grafického rozhraní

Třída TMainWindow je základní třída našeho grafického rozhraní. Je potomkem třídy QMainWindow. Tato třída je jakýsi grafický rámec umožňující vytvořit komplexní prostředí uživatelského rozhraní - hlavní aplikační rozhraní.

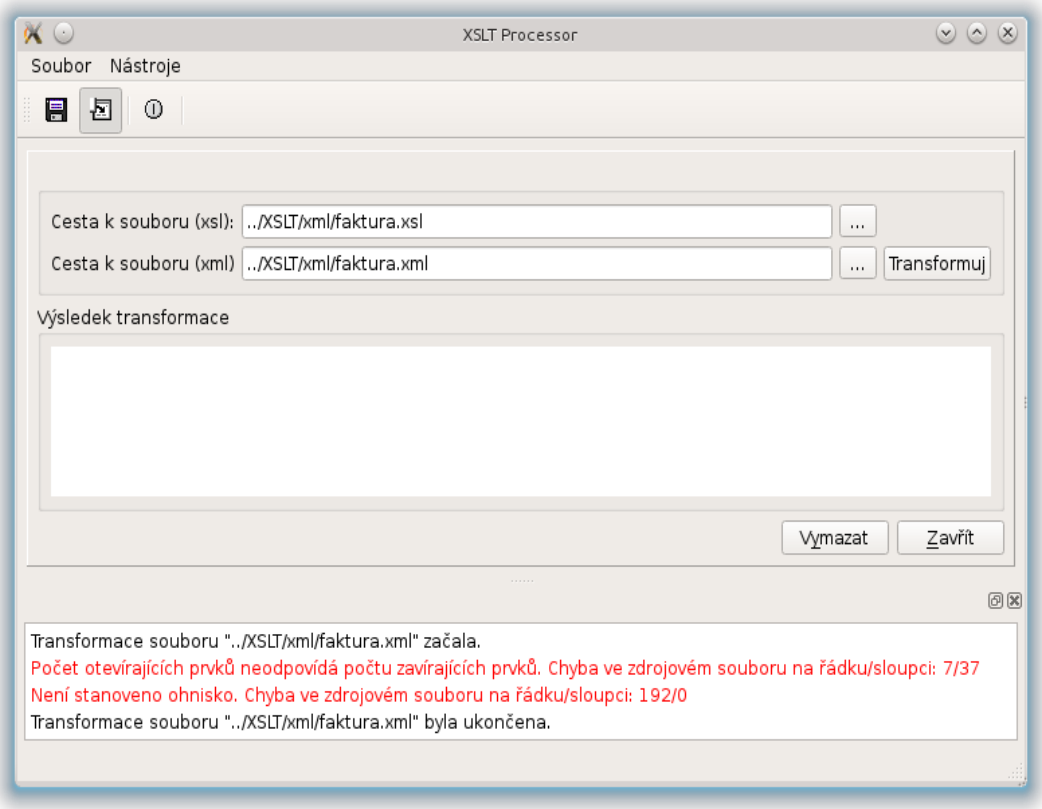
Uživatel komunikuje s grafickými prvky naší uživatelské třídy TMainWindow. Zadává cesty ke zdrojovým souborům transformace, spouští transformaci, ukládá její výsledek a vymazává či ukončuje formulář. K tomuto účelu jsou na této třídě implementovány metody, které ukazuje následující obrázek (viz Obrázek 24).

```
protected slots:
  void clear() throw();
  void clear_statusBar() throw();
  void set_xsl_file_path() throw();
  void set_xml_file_path() throw();
  void transform() throw();
  void show_message_console( bool show ) throw();
```

Obrázek 24: Deklarace slotů hlavní grafické třídy aplikace, zdroj: autor

Formulář pomocí slotu show_message() zobrazuje stavové informace ve stavové liště. Informační okno dále zobrazuje, výstup z metody show_handle_message(), informační či chybové hlášení při vlastním transformačním procesu (viz Obrázek 25).

Okno zpráv je standardně schované. Pokud ovšem dojde kritická zpráva okno se automaticky zobrazí. Uživatel může informační okno pomocí menu, nebo v nástrojové liště, ponechat vždy zobrazené, nebo stejným způsobem skrýt.



Obrázek 25: Detail okna zpráv, zdroj: autor

4.2.3.2 *Transformační* třída

Vlastní XSL transformaci zajišťuje třída TXSLTransformer. Konkrétně se jedná o metodu transform(), která má dva parametry. Prvním je cesta k souboru XSL, druhým je cesta k souboru XML (viz Obrázek 26).

```
QString TXSLTransformer::transform( const QString& xslFilePath,
                                    const QString& xmlFilePath )
                                        throw( const TException )
```

Obrázek 26: Deklarace metody zajišťující XSL transformaci, zdroj: autor

Nejprve dochází k načtení XML souboru (viz Obrázek 27).

```
QByteArray xml( read_file( xmlFilePath ) );
QBuffer content_buffer;
content_buffer.open( QBuffer::ReadWrite );
content_buffer.write( xml );
content_buffer.reset();
```

Obrázek 27: Načtení XML souboru, zdroj: autor

Pokud je soubor úspěšně načten, jeho obsah se stává prvním argumentem instance třídy QXmlQuery, Druhým parametrem je pak soubor s definicí XSL. Vlastní transformace se na instanci třídy QXmlQuery provede zavoláním metody evaluateTo(), která do vstupně výstupního argumentu metody vrátí transformovaný HTML dokument (viz Obrázek 28).

```
QXmlQuery myQuery( QXmlQuery::XSLT20 );
myQuery.setMessageHandler( _msgHandler );
myQuery.setFocus(&content_buffer);
QFile fQuery( xslFilePath );
fQuery.open( QIODevice::ReadOnly );
myQuery.setQuery( &fQuery );
myQuery.evaluateTo( &transform_buffer );
```

Obrázek 28: Vlastní XSL transformace, zdroj: autor

Třída definuje signál SMessage(), který je pomocí konektoru signálů a slotů spojen s metodou show_message() na třídě TMainWindow. Tímto dochází k předávání uživatelských zpráv do okna hlavní aplikace (viz Obrázek 29).

```
emit SMessage( tr( "Transformation %1 has finished" ).arg( xmlFilePath )
QApplication::restoreOverrideCursor();
```

Obrázek 29: Emitování (call back) zprávy, zdroj: autor

4.2.3.3 *Ostatní obslužné třídy a funkce*

Důležitou obslužnou třídou je Třída TMessageHandler. Instance této třídy je atributem transformační třídy TXSLTransformer. Zajišťuje pomocí signálů a slotů, implementací abstraktní virtuální metody handleMessage() (viz Obrázek 30), callback rozhraní pro zasílání zpráv. Tím jsme schopni získávat zprávy, v našem případě, ze třídy QXmlQuery, která provádí XSL transformaci.

```
void TMessageHandler::handleMessage(QtMsgType type, \
                                    const QString & desc, \
                                    const QUrl&, \
                                    const QSourceLocation & sourceLocat
```

Obrázek 30: Deklarace metody zpracovávající transformační události, zdroj: autor

Takto získané zprávy jsou upraveny a signálem SHandleMessage() emitovány příjemci. Příjemcem je třída grafického rozhraní TMainWindow a její informační okno.

Další obslužnou třídou je třída TException. Třída slouží k předávání informací o výjimkách, které vznikají při běhu programu. Ne všechny třídy a metody ji deklarují jako zachytitelný typ výjimek. Je uvedena pouze tam, kde šíření výjimek dává smysl s ohledem na důležitost vykonávané metody, na lokalizaci zdroje chyb a přesný výpis chybového stavu, který vznikl. Třída ve svém konstruktoru přijímá jako argumenty zprávu k zobrazení a zprávu pro logovací systém (v našem případě se logování provádí pouze na standardní výstup). V místě ošetření výjimky je možno z instance této třídy výše uvedené informace získat metodami disp() a log() (viz Obrázek 31).

```
class TException
{
  public:
    TException( const QString& strToDisp = QString::null,
                const QString& strToLog = QString::null );
    virtual ~TException();

    virtual const QString name() const;
    virtual const QString disp() const;
    virtual const QString log() const;

  protected:
    QString _strToDisp;
    QString _strToLog;
};
```

Obrázek 31: Deklarace třídy zpracovávající chybové události. , zdroj: autor

Obslužné funkce umožňující načíst a uložit soubor, zdrojový soubor tools.cpp, jsou poslední výkonnou částí aplikace. Jedná se o obecně použitelné funkce ke čtení a zápisu souboru. Funkce read_file() má jeden parametr a tím je cesta k souboru. V aplikaci je použita pro načítání XML dokumentu (viz Obrázek 32).

```
QByteArray read_file( const QString& fileName )
                    throw ( const TException ) {...}
```

Obrázek 32: Definice funkce zajišťující čtení souboru, zdroj: autor

Funkce save_file() je primárně použita pro uložení výsledného transformovaného HTML souboru a jejími parametry jsou vlastní ukládaná data a cesta k souboru, který bude uložen (viz Obrázek 33) .

```
void save_file( const QByteArray& data,
                const QString& fileName )
              throw ( const TException ) {...}
```

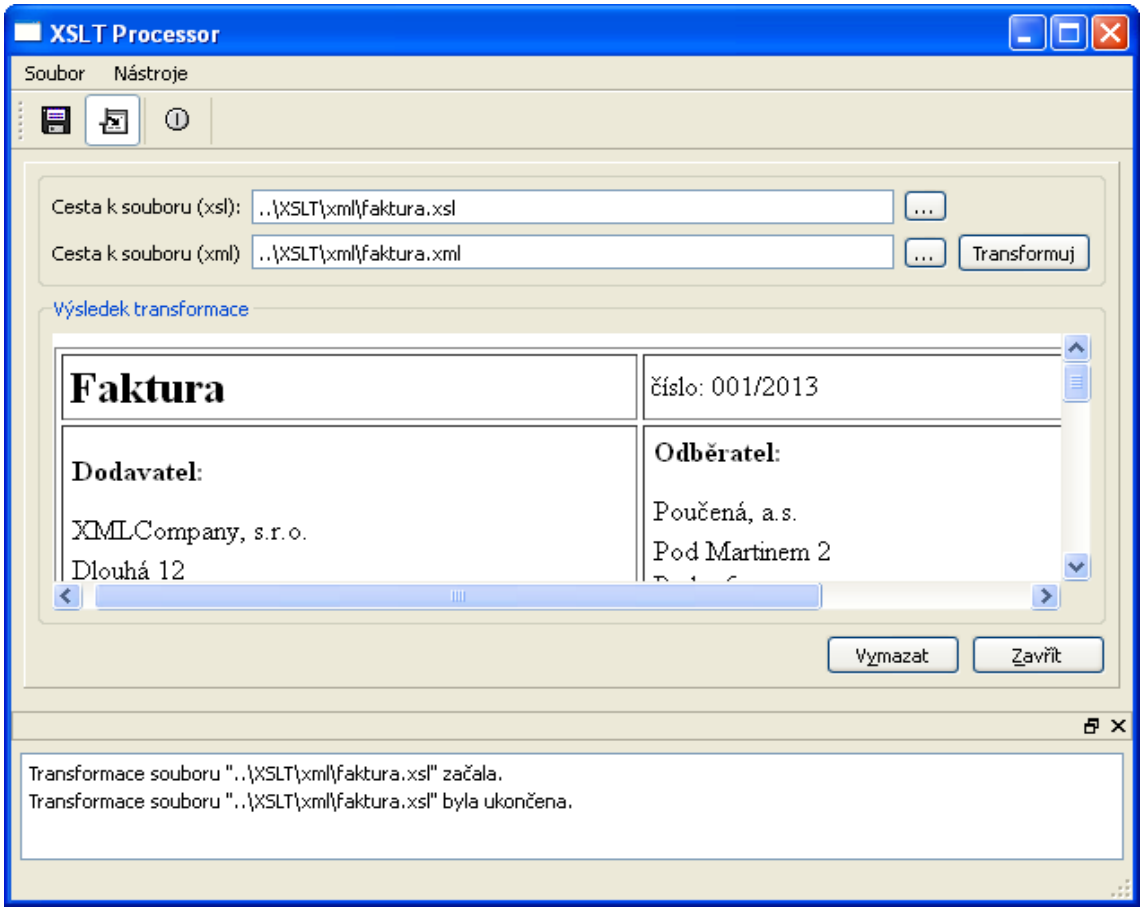
Obrázek 33: Deklarace funkce zajišťující uložení souboru, zdroj: autor

Obě funkce v případě neúspěchu generují výjimku Texception, která s sebou nese detailní informace o vzniklé chybě.

4.3 Testování

Aplikace byla otestována na příkladu transformace XML, která obsahuje data týkající se fakturace. Test byl proveden na OS Linux a WinXP. Výstupem je formulář faktury připravený k odeslání zákazníkovi (viz Obrázek 34).

Během finální kompilace projektu nebyla kompilátorem nalezena žádná syntaktická chyba, která by znemožňovala úspěšnou kompilaci. Věcné testy ukázaly několik chyb, které byly opraveny a aplikace s testovanými XML a XSL soubory vykazovala stabilitu a očekávané výsledky.



Obrázek 34: Finální podoba aplikace, zdroj: autor

5 Závěr

Jak již bylo uvedeno v úvodu této práce vývoj softwaru se v dnešní době neustále zrychluje. Aplikací přibývá geometrickou řadou. Přispívá k němu stále větší penetrace prostředků výpočetní techniky v populaci. Neméně důležitým prvkem je dostupnost programovacích jazyků a vývojových prostředí, které rychlost vývoje podporují ať už návrhem vlastního jazyka, knihovním (framework) API rozhraním, či právě možnostmi vývojových prostředí (IDE). Jedním z těchto nástrojů je právě multiplatformní vývojové prostředí frameworku Qt se svým IDE Qt Creator.

Tato práce má za cíl nastínit možnosti frameworku Qt. Framework Qt byl představen v první části práce. Byla popsána jeho struktura a moduly ze kterých se skládá. Další část práce byla věnována historii a principům programovacího jazyka C++. Zmíněn byl také open source způsob vývoje a různé způsoby licencování frameworku Qt. V technické části byly představeny možnosti instalace na různých běhových platformách operačních systémů. Detailně bylo představeno vývojové prostředí (IDE) Qt Creator a jeho jednotlivé komponenty (Správce projetků, Editor kódu, Qt Designer, Qt Linguist a debugger).

V druhé části práce byly teoretické znalosti, nabyté v první části, aplikované při implementaci vzorového příkladu. Nejdříve byla zpracována úvodní analýza aplikace a navržen její grafický design. Poté byly navrženy základní programové komponenty, ze kterých se bude aplikace skládat. Vlastní implementace byla provedena s využitím vývojového nástroje Qt Creator. V něm byly implementovány a odladěny jednotlivé programové komponenty aplikace.

Vlastní výsledek praktické části bakalářské práce demonstruje možnosti frameworku Qt při vývoji aplikace pro práci se značkovacími jazyky. Výsledkem je funkční aplikace umožňující transformaci zdrojových XML souborů pomocí technologie XSLT do formátu html.

Výslednou aplikaci je možno dále rozšiřovat o další funkční prvky. Ty mohou tento základní model funkčně kvalitativně i kvantitativně rozšířit.

6 Seznam literatury

1. **Blanchette Jasmin, Summerfield Mark;** *C++ GUI Programming with Qt4*. Westford USA: Prentice Hall PTR, 2008. 625 s. ISBN 978-0-13-235416-5.
2. **KDE Community.** About KDE.[Online] 2013 [Citace: 27.3.2013] http://www.kde.org/community/whatiskde/
3. **Stroustrup Bjarne;** *C++ programovací jazyk*. Praha: BEN – technická literatura, 1997. 686 s. ISBN 80-86056-20-1.
4. **Technopedia.** Integrated Development Environment. [Online] 2013 [Citace: 17.4.2013] http://www.techopedia.com/definition/26860/integrated-development-environment-ide
5. **Summerfield Mark;** *Rapid GUI programming with Python and Qt*. Westford USA: Prentice Hall PTR, 2007. 625 s. ISBN 978-0-13-235418-9.
6. **Vim.org.** Vim the editor. [Online] 2013 [Citace: 17.4.2013] http://www.techopedia.com/definition/26860/integrated-development-environment-ide
7. **Qt Project.** IDE Overview. [Online] 2013 [Citace: 17.4.2013] http://qt-project.org/doc/qtcreator-2.7/creator-overview.html
8. **Digia company.** Digia in brief. [Online] 2013 [Citace: 21.4.2013] http://www.digia.com/en/Company/
9. **Qt Project.** Supported platforms. [Online] 2013 [Citace: 28.4.2013] http://qt-project.org/doc/qt-4.8/supported-platforms.html
10. **Qt Project.** Installation. [Online] 2013 [Citace: 11.5.2013] http://qt-project.org/doc/qt-4.8/installation.html
11. **Qt Project.** Contribute to Qt. [Online] 2013 [Citace: 11.5.2013] http://qt-project.org/contribute
12. **Qt Project.** Qt Quick & QML. [Online] 2013 [Citace: 18.5.2013] http://qt-project.org/doc/qt-4.8/qtquick.html
13. **Digia company.** Product Qt. [Online] 2013 [Citace: 18.5.2013] http://qt.digia.com/product/
14. **Qt Project.** Qt Creator. [Online] 2013 [Citace: 20.5.2013] http://qt-project.org/wiki/Category:Tools::QtCreator
15. **Wikipedia.** Intelli-sense. [Online] 4.5.2013 [Citace: 22.5.2013] http://en.wikipedia.org/wiki/Intelli-sense

16. **Valgrind.org.** About Valgrind. [Online] 2013 [Citace: 10.6.2013] http://valgrind.org/info/about.html
17. **Josuttis M. Nicolai.** *C++ standardní knihovna a STL*. Brno: CP Books a.s., 2005. 743 s. ISBN 80-251-0511-1.
18. **Bradley Neil;** *XML kompletní průvodce*. Praha: Grada Publishing, 2000. 540 s. ISBN 80-7169-949-7.
19. **Qt Project.** Signals & Slots. [Online] 2013 [Citace: 20.5.2013] http://qt-project.org/doc/qt-4.8/signalsandslots.html#signals-and-slots

7 Seznam obrázků

8 Seznam příloh

1. CD se zdrojovými kódy uložené v adresářové struktuře dle projektového stromu vývojového nástroje Qt Creator.