

Multiplatformní aplikace

Cross-Platform Applications

Bakalářská práce

Ondřej Tuháček, DiS.

Vedoucí bakalářské práce: Ing. Václav Novák, CSc.

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

2010

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne 4. 1. 2010

Anotace

Tato práce se zabývá multiplatformním programováním. Po obecném teoretickém úvodu do problematiky multiplatformního programování se blíže věnuje projektu Mono. Uvádí přehled vývojových prostředí a dalších nástrojů, se kterými lze při vývoji pracovat a pojednává o některých jejich omezeních. Práce dále porovnává dostupné nástroje pro tvorbu grafického uživatelského rozhraní a nakonec uvádí sadu doporučení pro programátory multiplatformních aplikací.

Abstract

The aim of this thesis is cross-platform programming. After a general theoretical introduction to the cross-platform programming it gives attention to Mono Project. It gives the overview of available integrated development environments and other tools that can be used during development and it also mentions its limitations. The thesis then compares several available toolkits for creating the graphical user interface of applications. After that it presents a set of recommendations for cross-platform programmers in Mono.

Obsah

1 Úvod.....	6
2 Cíle práce.....	8
3 Teoretický úvod do problematiky.....	9
3.1 Typy jazyků.....	9
3.1.1 Překládané jazyky.....	9
3.1.2 Interpretované jazyky.....	9
3.1.3 Hybridní jazyky.....	10
3.2 Multiplatformní aplikace.....	10
3.2.1 Přístupy k tvorbě multiplatformních aplikací.....	11
3.2.1.1 Oddělený vývoj pro jednotlivé platformy.....	11
3.2.1.2 Interpretovaný jazyk.....	11
3.2.1.3 Překládaný jazyk a multiplatformní nástroje.....	12
3.2.1.4 Další způsoby	12
3.3 Microsoft .NET versus Mono.....	13
3.3.1 Microsoft .NET.....	13
3.3.2 Projekt Mono.....	15
3.4 Grafické uživatelské rozhraní.....	16
4 Metodika.....	18
4.1 Prověření vhodnosti Mono pro multiplatformní programování.....	18
4.2 Porovnání GUI toolkitů podle zvolených kritérií.....	18
4.3 Přehled dostupných nástrojů pro vývoj.....	21
4.4 Sada doporučení pro Mono programátory.....	21
5 Prověření vhodnosti Mono pro multiplatformní programování.....	22
5.1 Downloader.....	22
5.2 Grafické „Hello World“.....	24
5.3 Watcher.....	25
5.4 FeedReader.....	26
5.5 Portování stávající aplikace z .NET do Mono.....	29
6 Porovnání GUI toolkitů.....	33
6.1 WinForms.....	33

6.2 GTK+.....	35
6.3 Qt.....	37
6.4 wxWidgets.....	41
6.5 Cocoa.....	42
7 Nástroje pro vývoj.....	44
7.1 Vývojová prostředí.....	44
7.1.1 MonoDevelop.....	44
7.1.2 SharpDevelop.....	45
7.1.3 Microsoft Visual Studio.....	45
7.2 Návrháře grafických rozhraní.....	45
7.2.1 Glade.....	45
7.2.2 Qt Designer.....	47
7.3 Ostatní software.....	47
7.3.1 Mono Migration Analyzer (MoMA).....	47
7.3.2 Virtualizační nástroje (VirtualBox, VMWare Player).....	48
8 Sada doporučení pro Mono programátory.....	49
8.1 Nutná perfektní aktivní znalost angličtiny.....	49
8.2 Sledovat stav implementace tříd a jejich metod.....	49
8.3 Grafický toolkit volit s ohledem na majoritní cílovou platformu.....	50
8.4 Testovat a ladit průběžně na všech platformách.....	51
8.5 Vyhnout se používání platformně závislých přístupů.....	52
8.6 Nespoléhat na System.Drawing.SystemColors.....	52
8.7 Nebát se zdrojových kódů Mona.....	53
9 Závěr.....	54
10 Seznam použité literatury.....	57
11 Seznam příloh.....	59
12 Přílohy.....	60

1 Úvod

Každý, kdo již delší dobu pracuje s počítačem, má vytvořený alespoň pomyslný seznam svých oblíbených aplikací, které používá. V onom programu píše textové dokumenty, v jiném kreslí obrázky, přes některý program přistupuje k internetu, ke své poště a přes nějaký si povídá se svými přáteli. Mnohé z těchto aplikací jsou pevně svázány s operačním systémem, na kterém jsou provozovány. Některé aplikace však mají i své verze schopné běhu na operačních systémech jiných. Tyto nazýváme multiplatformními aplikacemi.

Nejrozšířenějším operačním systémem pro desktopové využití je bezpochyby Microsoft Windows. Uživatelsky je přívětivý, snadný na ovládání, mnohdy však bezohledně vnucený při koupi nového počítače. Pokud chceme pak z nějakého důvodu, ať už kvůli ceně, lepší koncepci či chybějící požadované funkcionalitě, přejít např. z Windows na Linux, narážíme na problém. Tím problémem jsou právě naše používané, nemultiplatformní aplikace. Někdo by mohl namítnout, že některé aplikace sice mají své „ekvivalenty“, ale většinou je potřeba naučit se zcela odlišnému ovládání, než na jaké jsme zvyklí, a to činí mnohým uživatelům nemalé problémy především z důvodu časové náročnosti. Mnohdy tyto „ekvivalenty“ ani nedosahují kvalit aplikací původních. Příkladem takového programu může být třeba grafický editor Adobe Photoshop. Ten je nativně určen pro operační systém Windows a i když existuje kvalitní multiplatformní náhrada v podobě editoru GIMP¹, je opravdu hodně nepříjemné učit se takřka „od nuly“ s novým editorem, nehledě na to, že některé funkce zde oproti Photoshopu chybí.

Právě neexistence kvalitních náhrad aplikací či právě absence přenositelnosti aplikací na různé platformy je jistě jednou z hlavních překážek v přechodu na

1 GIMP je zkratka The GNU Image Manipulation Program

jiný operační systém² a také důvodem, proč jsem toto téma zvolil pro svou bakalářskou práci. Sám jsem při přechodu na Linux postrádal zpočátku spoustu aplikací, proto mne zajímalo, proč vývojáři neprogramují aplikace tak, aby multiplatformní byly již od základu, respektive aby běžely alespoň na nejčastěji používaných operačních systémech, což jsou (ve světě i u nás) Windows, Linux a MAC OS X, jak dokazují statistiky v [4] a také v [5]. Pro některé nativní Windows aplikace je sice možné použít Wine³, ale mohu říci, že jak vizuálně, tak rychlostně to nyní není nejšťastnější řešení.

2 Dalším problémem může být např. neochota výrobců hardwaru vyvíjet ovladače pro své výrobky pro jiný OS

3 Wine Is Not Emulator – software umožňující spouštět nativní Windows aplikace pod operačními systémy Unixového typu.

2 Cíle práce

Hlavním cílem této práce je prověřit vhodnost projektu Mono z hlediska vývoje multiplatformních aplikací. Mono je otestováno postupně na konzolové aplikaci, aplikacemi s jednoduchým grafickým rozhraním a pak na složitější aplikaci s grafickým rozhraním. Zdrojové kódy těchto aplikací jsou dostupné na přiloženém médiu.

Dalším cílem je porovnání nástrojů pro tvorbu grafického uživatelského rozhraní, tzv. GUI toolkitů. Porovnání je provedeno podle předem stanovených kritérií a výstupem by mělo být doporučení vhodného toolkitu pro vývoj aplikací s grafickým rozhraním.

V práci je dále vytvořen přehled vývojových prostředí a jiných nástrojů, které lze použít při vývoji aplikací pro Mono a k usnadnění práce s testováním a portováním aplikací.

Posledním cílem této práce je vypracování sady doporučení začínajícím programátorům multiplatformních aplikací pro Mono. Tato doporučení by měla poukazovat na některá omezení či slepé uličky, kterých by se měli programátoři při vývoji multiplatformních aplikací v Monu vyvarovat.

3 Teoretický úvod do problematiky

Pro snazší pochopení problematiky a cílů této práce následuje stručný teoretický úvod.

3.1 Typy jazyků

Programovací jazyky můžeme dělit z několika různých hledisek. Můžeme je dělit z hlediska úrovně abstrakce (nižší jazyk, vyšší jazyk), dále rozlišujeme jazyky procedurální, neprocedurální. Bližší informace o dělení programovacích jazyků je možné nalézt v [6]. Nás však pro pochopení další části této práce bude zajímat především dělení jazyků na jazyky **překládané** (kompilované) a **interpretované**.

3.1.1 Překládané jazyky

Mezi překládané jazyky patří např. jazyk Pascal nebo jazyk C. Programátor napíše zdrojový kód v tomto jazyce a pomocí překladače neboli kompilátoru ho přeloží do strojového kódu. Takovému kódu již pak rozumí pouze daná platforma, pro kterou byl program přeložen a není ho pak již jednoduchým způsobem možné spustit na platformě jiné.

3.1.2 Interpretované jazyky

U interpretovaných jazyků je situace mírně složitější. Zde se zdrojový kód může přímo interpretovat neboli provádět. Tuto činnost provádí tzv. interpret, který může být k dispozici pro různé platformy. Příkladem může být skriptovací jazyk PHP.

Jiný přístup používá například Java, která zdrojový kód nejprve přeloží do tzv. mezikódu (u Javy se tento mezikód konkrétně nazývá bytekód), který následně interpretuje virtuální stroj JVM (Java Virtual Machine), který je rovněž

dostupný pro různé platformy. Virtuální stroj nebo obecně běhové prostředí pak tento kód přímo za běhu aplikace překládá do strojového kódu a tím program provádí.

3.1.3 Hybridní jazyky

Některé jazyky mohou být jak překládané, tak interpretované. Příkladem může být opět jazyk Java nebo i jazyky pro .NET. Mezikód vzniklý kompilací Javy je možné pomocí tzv. AOT⁴ kompilátoru přeložit do platformně závislého binárního spustitelného souboru, jehož běh pak bývá logicky rychlejší, než interpretování mezikódu.

3.2 Multiplatformní aplikace

Multiplatformními aplikacemi nazýváme ty programy, které je možné spouštět alespoň na dvou různých platformách. Tato platforma může být softwarová nebo hardwarová. Platformou se může rovněž myslet kombinace určitého hardwarového a softwarového vybavení počítače.

Softwarovými platformami bývají obvykle konkrétní operační systémy (tedy Linux, Unix, MAC OS X, Solaris, Windows), ale softwarovou platformou můžeme také rozumět Javu a její virtuální stroj, na kterém pak aplikace vyvinuté v jazyce Java, překompilované do tzv. bytekódu, běží. Hardwarovými platformami myslíme většinou architekturu procesorů, tedy x86, x86-64, IA-64, PowerPC apod. Z hlediska hardwarové platformy je tedy operační systém GNU/Linux rovněž multiplatformní, protože existuje jak ve variantě pro procesory x86, tak pro IA-64 a pro mnoho ostatních architektur.

V této práci se však pod pojmem platforma rozumí operační systém počítače.

4 AOT = Ahead-of-Time compiler

3.2.1 Přístupy k tvorbě multiplatformních aplikací

Vytvářet multiplatformní aplikace je celkem náročné a to už z toho důvodu, že každý operační systém používá jiné API⁵. Operační systémy od Microsoftu využívají Windows API, zatímco UNIX a odvozené systémy používají různé verze POSIX⁶ rozhraní. K tvorbě multiplatformního software můžeme tedy přistupovat několika způsoby.

3.2.1.1 Oddělený vývoj pro jednotlivé platformy

První možností je vyvíjet software odděleně pro jednotlivé platformy a to v překládaném jazyku. Znamená to, že pro každou platformu máme jiný zdrojový kód, který obsahuje platformně závislá volání. Aplikaci pak zkompi-lujeme pro každou platformu zvlášť a distribuujeme jako balíček se spustitelným binárním souborem schopným běhu pouze pod konkrétní jednou platformou. Tento přístup s sebou přináší problém se synchronizací funkčnosti a opravami chyb. Změny je totiž nutné provádět ve všech oddělených vý-vojových větvích zvlášť, což je zdlouhavé.

3.2.1.2 Interpretovaný jazyk

Druhým přístupem může být vývoj jednoho zdrojového kódu v interpre-tovaném jazyce. Aplikaci pak můžeme distribuovat v mezikódu, který je možné provádět na platformách, pro které existuje příslušné běhové prostředí (např. zmíněné JVM nebo CLR pro .NET).

5 Application Programing Interface – soubor funkcí a procedur jádra operačního systému (případně knihoven), který se používá při programování aplikací.

6 Portable Operating System Interface – rozhraní pro operační systémy, standardizováno jako IEEE 1003 a ISO/IEC 9945. jako snaha o sjednocení systémových volání pro snazší por-tování aplikací

3.2.1.3 Překládaný jazyk a multiplatformní nástroje

Dalším způsobem může být třeba použití překládaného jazyku C++, pro který je překladač dostupný pro různé platformy. Je pak ovšem potřeba vyřešit platformně závislé věci, jako např. cesty k systémovým adresářům, k uživatelskému úložišti dat. Tyto problémy za nás mohou vyřešit různé frameworky a stejně tak tvorbu grafického rozhraní k aplikaci můžeme svěřit multiplatformním grafickým toolkitům, např. Qt.

3.2.1.4 Další způsoby

Způsobů, jak docílit vytvoření multiplatformní aplikace, je jistě celá řada a není možné je postihnout všechny. Dalším způsobem může být použití multiplatformního běhového prostředí XULRunner, tedy pojmout náš program jako XUL aplikaci. Na tomto běhovém prostředí od Mozilly běží Mozilla Firefox, přehrávač Sunbird a další software.

Nepodařilo se mi nikde vyhledat úplný seznam metod, jak multiplatformní aplikaci vytvořit. Způsobů je v podstatě více, než kolik je programovacích jazyků a navíc multiplatformních nástrojů. V mnoha případech se totiž nemusíme omezovat na použití pouze jednoho jazyka, jednoho frameworku či dokonce jednoho grafického toolktiu.

V této práci jsem jako metodu tvorby multiplatformní aplikace použil programování v interpretovaném jazyce C#, jehož mezikód je následně spouštěn pod běhovým prostředím Mono Runtime na platformách Windows, Linux a Mac OS X.

3.3 Microsoft .NET versus Mono

3.3.1 Microsoft .NET

V roce 2000 přišel Microsoft s novou koncepcí programování pro Windows, s technologií .NET⁷. Mezi hlavní motivační důvody, proč vlastně vůbec přijít s něčím novým a lepším, patřily hlavně často se opakované chyby programátorů v práci s pamětí, neplatnými konverzemi datových typů, špatná spolupráce knihoven napsaných v různých jazycích a především také tzv. problém „DLL Hell“. Tento problém spočíval v tom, kdy jedna aplikace nainstalovala DLL knihovnu, kterou používala, a pak jiná aplikace tuto knihovnu přepsala při instalaci svou starší verzí. První aplikace pak mohla přestat fungovat, protože ve starší knihovně již nemusela najít ty funkce, které ke svému běhu požadovala. Tyto problémy řeší platforma .NET možností koexistence různých verzí stejné knihovny v systému.

Alokováním a uvolňováním paměti se již programátoři nemusí tolik trápit. V Microsoft .NET je automatická správa paměti, tzv. Garbage Collector, která si udržuje reference na všechny objekty. Pokud je objekt již nevyužíván, paměť uvolní automaticky. Platforma .NET rovněž umožňuje, aby v systému existovalo souběžně více verzí DLL knihoven v tzv. GAC⁸ a psát aplikace můžeme vlastně v jakémkoliv programovacím jazyce, pro který existuje překladač do mezikódu.

Microsoft .NET je tedy souhrnný název pro:

- kompilátory jazyků C#, VB.NET a dalších,

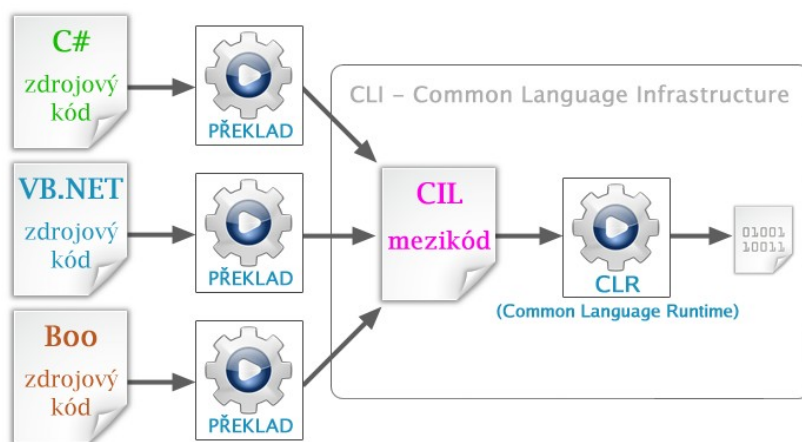
7 Čteme „dot net“

8 Global Assembly Cache – místo pro knihovny .NET frameworku

- řízené běhové prostředí CLR (Common Language Runtime), které spouští program přeložený z těchto jazyků do mezikódu CIL (Common Intermediate Language)
- a knihovnu základních tříd BCL (Base Class Library).

Běhové prostředí a mezikód jsou popsány v otevřené specifikaci CLI (Common Language Infrastructure), kterou Microsoft standardizoval u Ecma⁹, konkrétně ECMA-335. Standardizovaný je i jazyk C# a to pod standardem ECMA-334. Jazyk C# byl vytvořen právě kvůli technologii .NET a vychází z jazyků Java a C++ a přebírá z každého z nich jeho pozitiva [3].

Jak je z obrázku 1 patrné, zdrojový kód .NET aplikace můžeme psát v libovolném jazyce, který nám vyhovuje nebo na který jsme byli doposud zvyklí. Musí však pro něj být dostupný kompilátor pro .NET. Těchto jazyků je již celá řada a nic nebrání psát aplikace pro .NET například i v PHP.



Obrázek 1: Princip práce technologie .NET

Kompletní seznam použitelných jazyků je možné nalézt v přehledu [7]. Každý takový jazyk musí splňovat specifikace CLS (Common Language Specification) a CTS (Common Type Specification), aby uměl správně překládat

⁹ Ecma International se sídlem v Ženevě je nezisková organizace produkující různé standardy v oblasti informačních a komunikačních technologií.

zdrojový kód do mezikódu CIL (Common Intermediate Language). Tento mezikód se konkrétně v Microsoft .NET označuje někdy též MSIL (Microsoft Intermediate Language).

Překladem zdrojového kódu z kteréhokoliv takového programovacího jazyka vznikne tedy vždy stejný mezikód. Tento mezikód je pak řízeným (managed) běhovým prostředím CLR prováděn. To znamená, že až při běhu aplikace je prováděna kompilace do strojového kódu. Používá se tzv. JIT (Just-in-Time Compilation), kdy se překládá pouze ta část mezikódu, která je zrovna potřeba. Tento princip je podobný Javě, kde se zdrojový kód přeloží do tzv. bytekódu, který je pak pomocí JVM (Java Virtual Machine) prováděn.

Jak u Javy, tak u MS .NET nebo Mona je však možné navíc použít tzv. AOT (Ahead Of Time) kompilaci, která předkompiluje aplikaci z mezikódu do platformně závislého strojového kódu. Toto pak může zrychlit běh aplikace.

3.3.2 Projekt Mono

Mono je open-source implementace technologie Microsoft .NET. U jeho zrodu stál mexický programátor Miguel de Icaza a jeho firma Ximian. Miguel de Icaza je mimo jiné autorem linuxového desktopového prostředí Gnome a především pro vývoj aplikací pro toto prostředí plánoval původně Mono použít. Mono bylo již v roce 2001 uvolněno jako open-source projekt, aby se tak urychlil jeho vývoj. V roce 2003 byla společnost Ximian koupena společností Novell a tato společnost nyní vývoj Mona zaštiťuje.

Mono je tedy open-source implementací standardů ECMA-334 a ECMA-335. Skládá se z:

- Mono Runtime, což je implementace řízeného běhového prostředí pro různé operační systémy,

- knihovny základních tříd, (Mono má oproti Microsoft .NET v základní instalaci některé knihovny navíc, jedná se především o knihovny pro práci s GTK+ toolkitem a Gnome prostředím),
- sady nástrojů (disassembler, utilita gacutil pro práci s Global Assembly Cache (GAC), apod.),
- kompilátorů pro různé jazyky (C#, Java, VisualBasic, Boo atd..)
- a vlastně sem můžeme zahrnout i vývojové prostředí MonoDevelop.

Mono je s v současnosti oficiálně provozovatelné na operačních systémech Windows, Linux a Mac OS X. Na některých dalších operačních systémech jako např. Solaris a BSD je možné Mono zprovoznit také, avšak pro tyto systémy není v tuto chvíli distribuováno společností Novell oficiální cestou.

Aplikace, které napíšeme pro Mono, budou tedy (nebo spíše by měly být) multiplatformní. Poběží pod běhovým prostředím na jakémkoliv operačním systému, kde bude Mono Runtime nainstalované.

A jaké tedy aplikace můžeme vlastně pod Monem vytvořit? Klasické konzolové aplikace, aplikace s grafickým uživatelským rozhraním za použití některého z možných grafických toolkitů (GTK+, Windows Forms, Qt ...) nebo webové aplikace v ASP.NET.

3.4 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je klíčovým prvkem snad všech aplikací. Na vzhledu a intuitivnosti ovládání aplikace spočívá rozhodnutí uživatele, zda s aplikací bude vůbec pracovat a zda si aplikaci oblíbí.

K vytvoření intuitivního a pěkného grafického rozhraní slouží nástroje zvané GUI¹⁰ toolkity. Pomocí těchto nástrojů se v aplikaci vykreslují okna, vstupní

10 Graphical User Interface – Grafické uživatelské rozhraní

textová pole, tlačítka, popisky. Nástroje zároveň zprostředkovávají interakci s uživatelem prostřednictvím událostí. Událostí může být například kliknutí na tlačítko, stisk klávesy v textovém poli, změna velikosti okna apod. Na tyto události se pak napojuje obslužný aplikační kód a ten provede nějakou akci.

Existuje bezpočet nejrozličnějších GUI toolkitů. Některé byly vyvinuty pouze za účelem použití v jediné aplikaci (např. toolkit eLiquid použitý v grafickém editoru Pixel), některé naopak slouží pro použití v nejrozličnějších aplikacích napříč různými platformami. Většina grafických toolkitů je napsána v C nebo C++, ale jsou zde i toolkity napsané třeba v Javě (Swing) nebo třeba v Object Pascalu (fpGUI).

Tato práce se dále zabývá grafickými toolkity GTK+, Qt, Cocoa, WinForms a wxWidgets. Abychom však mohli grafické toolkity používat v Monu a tedy jazyku C# či např. VB.NET, potřebujeme, aby jejich nativní podobu napsanou právě v C nebo C++ zaobalovala speciální knihovna. Těmto knihovnám se říká wrapper, někdy též binding. Tyto knihovny poskytují rozhraní pro .NET jazyky a interně pracují s původními knihovnami.

Přehled dostupných obalových knihoven (dále wrapperů) ke zvoleným toolkitům včetně informace o jejich multiplatformnosti uvádí srovnávací tabulka v příloze.

4 Metodika

V této části práce je popsán soubor metod, který vede k dosažení jednotlivých vytyčených cílů zmíněných v části 2. Začneme u prvního z nich.

4.1 Prověření vhodnosti Mono pro multiplatformní programování

Pro splnění tohoto cíle byl zvolen postup vytvoření několika zkušebních aplikací s různou funkcionalitou. Metoda prověřování je tedy následující:

- 1) Tvorba jednoduché aplikace Downloader.
- 2) Tvorba „Hello World“ grafické aplikace v každém toolkitu.
- 3) Tvorba jednoduché grafické aplikace „Watcher“ ve zvoleném toolkitu.
- 4) Tvorba složitější grafické aplikace „FeedReader“ s použitím různých grafických toolkitů pro každou platformu.
- 5) Portování stávající aplikace z MS.NET do Mono.
- 6) Vyhodnocení funkčnosti aplikací.

4.2 Porovnání GUI toolkitů podle zvolených kritérií

Toolkity jsou porovnávány při tvorbě aplikací uvedených v předchozím cíli. Kritéria, která jsem zvolil, vycházejí jak z vývojářských, tak z uživatelských zkušeností. Následuje přehled kritérií.

- 1) Přehled wrapperů použitelných pro Mono.
 - Kritérium určuje, zda daný grafický toolkit má nějaký použitelný wrapper, popř. více wrapperů použitelných v Monu.

- 2) Zda má toolkit multiplatformní wrapper a do jaké míry.
 - Určuje, zda je možné wrapper pro grafický toolkit použít pro všechny platformy (Windows, Linux, Mac OS X).
- 3) Jaké existují grafické návrháře, jakou funkcionalitu poskytují a v jakém formátu ukládají navržené rozhraní.
 - Grafické návrháře slouží k návrhu uživatelského rozhraní v uživatelsky příjemném prostředí. V podstatě pro všechny toolkity je možné vytvořit grafické rozhraní pomocí zdrojového kódu. Mnohem pohodlnější je však využít nějaký návrhář, kde se pomocí myši vytváří rozhraní aplikace a vývojář tak má bezprostřední kontrolu nad tím, jak bude dané rozhraní skutečně vypadat. Takto vytvořené rozhraní se pak do aplikace připojí jako např. XML soubor.
- 4) Jakým způsobem se rozmísťují ovládací prvky na formulář.
 - Kritérium sleduje, zda se ve zkoumaném toolkitu prvky na formulář umísťují pomocí absolutního pozicování, případně spojeným s ukotvením k okrajům nadřazeného prvku, nebo zda je možné v toolkitu použít tzv. layout manažery. Každý způsob má svá pro a proti. Layout manažery umožňují vytvořit rozhraní pružné pro různá nastavení velikosti písma a stylu grafiky na cílové platformě, což u absolutně pozicovaných prvků na formuláři může být problém, protože ovládací prvky se mohou následně zvětšit a překrývat se přes sebe. Absolutní pozicování má naopak výhodu mnohem snazšího návrhu a snadné rozšiřitelnosti, zatímco u layout manažerů je třeba již na počátku návrhu dobře rozvrhnout, jak bude který formulář vypadat a kolik bude mít kde ovládacích prvků, neboť pozdější rozšiřování může být mnohdy složité.

5) Dostupnost dokumentace k wrapperu a toolkitu.

- Pro toto kritérium používám hodnotící stupnici od 1 do 5. Hodnoty přesně znamenají toto:

1. Dokumentace je dostupná online a to včetně ukázkových příkladů použití jednotlivých komponent.
2. Dokumentace je dostupná alespoň offline, bez příkladů použití nebo s velmi málo příklady.
3. Dokumentace je v neúplném stádiu kompletace, chybí tedy zdokumentování funkčnosti některých komponent.
4. Dokumentace je těžko dostupná a/nebo má zdokumentováno pouze malé procento z použitelných komponent.
5. Dokumentace není dostupná.

6) Možnost použít komponentu pro zobrazení HTML obsahu či jinou komponentu pro snadné formátování obsahu.

- Komponenty pro zobrazení HTML obsahu se používají v mnoha aplikacích kvůli své snadné možnosti formátovat např. výstupní data z databáze.

7) Licence, pod kterou nebo pod kterými je grafický toolkit šířen.

- Ne všechny grafické toolkity je možné použít zdarma pro vývoj jak nekomerčních tak komerčních aplikací.

8) Možnosti stylování vzhledu ovládacích prvků.

- Toto kritérium uvažuje jiné možnosti stylování, než je použití HTML komponenty. Některé grafické toolkity mají možnost stylování vzhledu pomocí témat, některé např. použitím vlastního stylovacího jazyka nebo CSS.

9) Jazyk, ve kterém je grafický toolkit vyvíjen.

- Kritérium pouze informuje, v jakém programovacím jazyce je grafický toolkit vyvíjen. Jelikož v Monu používáme wrappery, není příliš potřeba daný jazyk ovládat.

10) Dostupnost wrapperů pro nativní knihovny.

- Některé wrappery jsou již v základní instalaci Mona, jiné je nutné stáhnout z příslušných stránek a popřípadě i zkompilevat.

4.3 Přehled dostupných nástrojů pro vývoj

U tohoto cíle je postup relativně snadný.

- 1) Na Internetu a v literatuře vyhledat dostupná vývojová prostředí, která podporují vývoj aplikací v C#, popř. dalších jazycích pro Mono.
- 2) Dále vyhledat a stručně charakterizovat další možné nástroje užitečné pro vývoj, např. návrháře grafických rozhraní.
- 3) U všech podat stručnou charakteristiku a zmínit jejich klady a zápory.

4.4 Sada doporučení pro Mono programátory

V průběhu vývoje aplikací pro splnění prvního cíle získat zkušenosti s programováním v Monu a na základě těchto zkušeností vypracovat sadu doporučení začínajícím programátorům multiplatformních aplikací. Tato doporučení by měla obsahovat upozornění, čeho se mají programátoři při vývoji v Monu vyvarovat, především pokud zároveň zamýšlejí, aby vyvíjená aplikace byla multiplatformní. Doporučení nebudou podávat kompletní výčet „slepých uliček“, ale poslouží jako jakýsi odrazový můstek k získávání vlastních zkušeností s projektem Mono.

5 Prověření vhodnosti Mono pro multiplatformní programování

Nejprve bylo nutné nainstalovat všechny tři cílové operační systémy. Poté se na každý z připravených operačních systémů nainstalovalo Mono a vývojové prostředí MonoDevelop. Instalaci operačních systémů zde popisovat nebudu, neboť by to bylo nad rámec této práce, stejně tak instalace Mono pro jednotlivé operační systémy. Postup kompilace Mono ze zdrojových kódů je však složitější záležitostí a je proto uveden v příloze C.

Samotné testovací aplikace byly vyvíjeny především pod OS Linux, distribuce OpenSUSE 11.2. Software byl průběžně testován a laděn pod Windows XP SP3, Windows 7 a Mac OS X 10.5.7 (použita byla neoficiální verze iDeneb 1.5.1 schopná běhu i na PC).

Všechny aplikace, které zde nyní budou popsány, jsou dostupné včetně zdrojových kódů na médiu přiloženém k této práci.

5.1 Downloader

Aplikace Downloader byla zvolena jako první testovací aplikace. Jedná se o konzolovou aplikaci vytvořenou namísto obligátního jednoduchého „Hello World“ k otestování, zda je Mono funkční a běhuschopné na všech platformách. Aplikace Downloader je jednoduchý, jednovláknový stahovač souborů z internetu.

Z konzole se spouští s předaným parametrem na URL stahovaného souboru. URL může používat jak protokol http, tak protokol ftp. Dalšími nepovinnými parametry jsou: umístění, kam se soubor po stažení uloží, velikost bufferu, řetězec reprezentující http hlavičku user-agent.

Jádro aplikace tvoří třída `Downloader`, ta je zodpovědná za samotné stahování souborů. K této třídě je vytvořeno konzolové rozhraní `DownloaderConsole`, které registruje posluchače událostí třídy `Downloader`. Nic tedy nebrání pozdějšímu vytvoření grafického rozhraní aplikace namísto konzolového. Činnost tohoto programu je následující (neodpovídá přesně zdrojovému kódu aplikace, pouze znázorňuje její princip):

```
string url = "http://...";           // URL stahovaného souboru
int bufferSize = 32768;              // Velikost bufferu
string destination = "/home/.../";   // Cílové umístění souboru
...

// Vytvoříme WebRequest (http nebo FTP požadavek)
WebRequest request = WebRequest.Create (url);
// Získáme odpověď od serveru
WebResponse response = request.GetResponse ();
// Získáme proud dat z odpovědi od serveru
Stream str = response.GetResponseStream ();
...

// Necháme si vygenerovat dočasný soubor s nulovou velikostí
string temp = Path.GetTempFileName ();
// Pomocí třídy FileStream budeme do dočasného souboru zapisovat
using (FileStream tempStream = new FileStream(temp, FileMode.Create);
{
    // Vytvoříme pole bytů jako buffer pro načtená data
    byte[] buffer = new byte[bufferSize];
    // Proměnná pro množství přečtených dat
    int amount = 0;
    // Postupně načítáme data z proudu str, dokud jsou dostupná
    while ((amount = str.Read (buffer, 0, bufferSize)) > 0)
    {
        ...

        // Přečtená data zapíšeme do temp souboru
        tempStream.Write (buffer, 0, amount);
    }
}
...
File.Move (temp, destination);
```

Ukázka 1: Princip aplikace Downloader

Připomínám, že kompletní česky okomentovaný zdrojový kód je k dispozici na příloženém médiu k této práci.

Soubor se stahuje ze vzdáleného umístění po malých částech o velikosti `bufferSize`. Je tomu tak z následujícího důvodu. Pokud by byl totiž stahován soubor veliký třeba až 2GB, bylo by nutné použít 2GB operační paměti pro načtení souboru a jeho následné přeuložení. Takto se soubor vždy stahuje po částech o výchozí velikosti 32kB a tyto části pravidelně přeukládá do dočasného temp souboru. Až je soubor celý dotažen, je z temp souboru přesunut do cílového umístění.

Dočasný soubor je získán pomocí statické metody `Path.GetTempFileName()`. Tento soubor je vytvořen v každém operačním systému jinde:

Operační systém	Cesta k dočasným souborům
Windows XP	C:\Documents and Settings\<uzivateL>\Local Settings\Temp\
Windows 7	C:\Users\<uzivateL>\AppData\Local\Temp\
Mac OS X	/var/tmp/
Linux	/tmp/

Tabulka 1: Cesty k dočasným souborům

Aplikace byla hned po vytvoření otestována na všech třech cílových operačních systémech a byla plně funkční. Na platformně MS Windows bylo možné ji spustit jak pomocí Mono Runtime, tak pomocí .NET Runtime.

5.2 Grafické „Hello World“

V každém z testovaných grafických toolkitů je vytvořena jednoduchá „Hello World“ aplikace a je k dispozici na přiloženém médiu. Princip je prakticky vždy stejný. Každý toolkit obsahuje třídu řekněme s názvem `Application`, kde se např. metodou `Init()` zinicilizuje, dále se vytvoří hlavní okno aplikace a následně se metodou řekněme `Run()` spustí hlavní smyčka. Ta má za úkol sledovat vyvolávání událostí v hlavním okně a obsluhovat je. Tato smyčka a tedy i aplikace je pak ukončena voláním metody většinou s názvem `Quit()`.

5.3 Watcher

Další testovací aplikací je aplikace Watcher. Jedná se o aplikaci s jednoduchým grafickým uživatelským rozhraním. Jako grafický toolkit byl zvolen GTK+ a to kvůli dostupné dokumentaci, multiplatformnímu wrapperu GTK# a jeho dostupnosti v rámci základní instalace Mona. Grafické uživatelské rozhraní aplikace bylo vytvořeno pomocí návrháře Glade.

Tato aplikace spouští službu protokolu HTTP na zvoleném portu a klientovi, který se prostřednictvím webového prohlížeče na daný socket¹¹ připojí, poskytne snímek obrazovky operačního systému. Tento snímek se zároveň uloží jako soubor do umístění, odkud byla aplikace spuštěna, tedy nikoliv vždy do umístění, kde se nachází spustitelný soubor aplikace.

Aplikace má díky GTK+ toolkitu konzistentní vzhled. Její funkčnost je však omezená a to díky chybějící implementaci metody `CopyFromScreen()` ze třídy `Graphics`. Funkčnost však není implementována pouze pro Mac OS X. Jinými slovy aplikace bez potíží funguje na Windows a na Linuxu. Na Mac OS X se sice spustí, ale nefunguje snímání obrázků z uživatelské plochy. Když se podíváme do zdrojových kódů Mona na implementaci příslušné metody, zjistíme, že je opatřena atributem `MonoLimitation`.

¹¹ Socket je dvojice IP adresa počítače + port, neboli číslo služby

Následující úsek zdrojového kódu byl pro lepší přehlednost zkrácen.

```
[MonoLimitation ("Works on Win32 and ... on X11 but not on Cocoa ...")]
public void CopyFromScreen (int sourceX, int sourceY, ...)
{
    ...
    if (GDIPlus.UseX11Drawable) {
        CopyFromScreenX11 (sourceX, sourceY, ...);
    } else if (GDIPlus.UseCarbonDrawable) {
        CopyFromScreenMac (sourceX, sourceY, ...);
    } else {
        CopyFromScreenWin32 (sourceX, sourceY, ...);
    }
}
...
private void CopyFromScreenMac (int sourceX, int sourceY, ...)
{
    throw new NotImplementedException ();
}
```

Ukázka 2: Chybějící implementace metody CopyFromScreenMac()

Vidíme, že pro každou platformu je volána zvláštní privátní metoda. Konkrétně pro Mac OS X je volána metoda `CopyFromScreenMac()`. V této metodě je však pouze vyhozena výjimka `NotImplementedException`.

Z tohoto důvodu by bylo nutné implementovat příslušnou funkčnost jiným způsobem, což už činí návrh aplikace (nutná znalost, co je a co není pro kterou platformu implementované) a také vývoj samotný (řešit pro danou platformu speciálním způsobem) mnohem složitějším.

5.4 FeedReader

Poslední a zároveň největší testovací aplikací je aplikace FeedReader. Jedná se o čtečku syndikačních kanálů (RSS, Atom, SSS¹²).

Aplikace si v jednoduché „embedded“ databázi SQLite udržuje seznam odebíraných syndikačních kanálů. Periodicky načítá jejich XML obsah z internetu a nově nalezené příspěvky ukládá rovněž do této databáze. Pro každý ode-

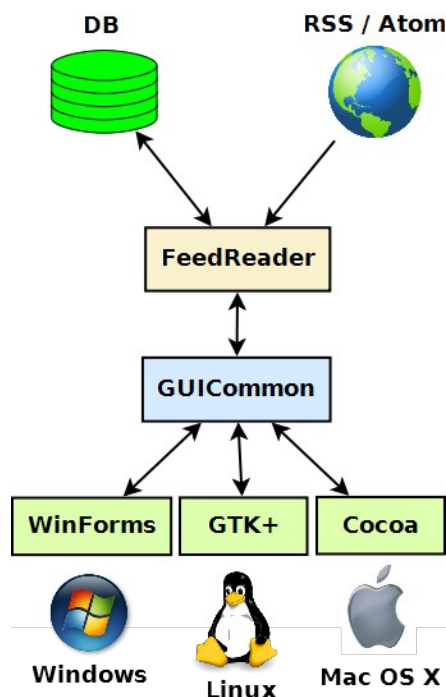
12 SSS = Simple Site Summary – Proprietární odlehčený syndikační formát

bíraný kanál pak poskytuje výpis nejnovějších příspěvků v HTML podobě do grafického rozhraní aplikace, kde je zobrazen v komponentě, která má schopnost HTML obsah vykreslit.

Právě kvůli požadavku pro zobrazení HTML obsahu bylo nutné použít pro každou platformu jiný grafický toolkit. Aplikace byla původně opět vyvíjena v GTK+, ale při pokusu o použití Widgetu pro zobrazení HTML obsahu byla zjištěna jeho nemultiplatformnost. GTK+ má možnost použít buďto widget s renderovacím jádrem Gecko nebo s jádrem WebKit. Příslušné wrappery se nazývají `gecko-sharp`, resp. `webkit-sharp`, ale ani jeden z nich není snadno zprovoznitelný na operačním systému Windows. Wrapper `gecko-sharp` je sice distribuován v základní instalaci Mono pro Windows, ale není automaticky přidán do Mono GAC a navíc je nutné mít na cílové platformě nainstalované ještě GRE for Gecko-Sharp. Dokonce i poté při inicializaci či používání testovací aplikace využívající widget `Gecko.WebControl` pod Windows došlo k vyhození výjimky, kterou se mi nepodařilo odladit. Další vývoj byl tedy koncipován tak, že aplikační jádro zůstalo stejné pro všechny tři platformy. Přibyla tzv. vrstva abstrakce grafického uživatelského rozhraní. Samotná grafická rozhraní v různých toolkittech už pouze zobrazují výstupy od této abstrahující vrstvy a zároveň volají její příslušné metody na základě událostí, které v grafickém rozhraní nastanou.

Pro Windows byl tedy použit grafický toolkit WinForms, který zapouzdřuje nativní ovládací prvky Windows. Tento toolkit používá komponentu `WebBrowser` s renderovacím jádrem prohlížeče MS Internet Explorer. Pro Linux byl použit toolkit GTK+ a komponenta `WebView` s jádrem WebKit a pro Mac OS X byl použit nativní toolkit Cocoa a wrapper `Monobjc`, ve kterém je rovněž použita komponenta s jádrem WebKit.

Obrázek 2 znázorňuje pro představu schéma aplikace FeedReader. Jedná se pouze o schéma ukazující jednotlivé vrstvy aplikace.



Obrázek 2: Schéma aplikace FeedReader

Jádro aplikace FeedReader je zodpovědné za čtení syndikačních kanálů z Internetu. Tyto kanály se zpracují příslušným parserem pro konkrétní kanál. Získají se z nich tak informace o kanálu a o položkách. Informace o kanálu a přijaté položky se překontrolují proti databázi a případné změny či nové položky, které v databázi ještě nejsou, se uloží. Jádro zároveň poskytuje tyto informace z databáze v HTML podobě do vrstvy GUICommon, což je abstrahující vrstva grafického rozhraní. HTML výstup v podobě kanálu a několika nejnovějších položek se vkládá do připravené šablony, která je připojena jako zdroj assembly aplikace. Vrstva GUICommon obsahuje funkcionalitu společnou pro všechna uživatelská rozhraní. Grafická rozhraní si u této vrstvy registrují posluchače událostí a jsou pak notifikována o změnách stavu, tedy například došlo-li k obnovení kanálů, zavolá se posluchač, tedy handler příslušné události a ten pak provede obnovu kanálů. Každé grafické rozhraní tak vlastně funguje

jako observer vrstvy `GUICCommon` a zároveň zprostředkovává interakci s uživatelem, tedy předávají volání pokud se stiskne např. tlačítko v nástrojové liště.

Aplikace si v sobě nese jako zdroj rovněž prázdnou databázi, kterou při prvním spuštění aplikace uloží do uživatelského úložiště dat. To se nachází na každém systému opět jinde. V Linuxu je to `/home/<uživatel>/.config/FeedReader`, stejně jako v Mac OS X, kde je místo `home` adresář `Users`. Na Windows XP je to `C:\Documents and Settings\<uživatel>\Data aplikací\FeedReader`. Nově na Windows 7 je to (ač funguje díky symbolickým odkazům i předešlý způsob) `C:\Users\<uživatel>\AppData\Roaming\FeedReader`. Do tohoto umístění se rovněž uloží konfigurační soubor aplikace. V něm jsou parametry aplikace jako interval aktualizace v sekundách, nastavení barev HTML výpisu apod. Rovněž se v tomto umístění v podadresáři `images` ukládají obrázky odebíraných kanálů.

Aplikace obsahuje hlavní okno, formulář pro přidávání kanálu k odběru, formulář s přehledem odebíraných kanálů a položek a formulář pro nastavení aplikace. Pouze ve verzi pro Mac OS X je aplikace omezena jen na hlavní okno z důvodu obtížné práce s Cocoa toolkitem a verzi Mac OS X pro PC.

5.5 Portování stávající aplikace z .NET do Mona

Cílem tohoto úkolu bylo prověřit možnost portování stávající aplikace, již napsané pro Microsoft .NET a používající grafický toolkit WinForms, do Mona. Projekt Mono nabízí k tomuto účelu nástroj „Mono Migration Analyzer“ (zkráceně MoMA), který má detekovat platformě závislé postupy v kódu a upozornit na případné použití neimplementovaných tříd či metod. Analyzuje již hotové assembly, tedy exe nebo dll soubory. Po zanalyzování je zobrazen protokol s nalezenými problémy. Nástroj však, jak je dále ukázáno, neodhalí vždy všechny možné problémy.

Pomocí tohoto nástroje byla portována aplikace „Movie Database Express“. Jedná se o aplikaci pro vedení záznamů o audiovizuálních dílech, která byla

vytvořena v Microsoft .NET, používala grafický toolkit WinForms a pracovala rovněž s databází SQLite.

Tuto aplikaci je možné najít na přiloženém médiu před portováním i po portování. Sestava v původní verzi obsahovala 2 projekty. Samotnou aplikaci a její instalátor. Jelikož projekt instalátoru byl vytvořen ve vývojovém prostředí Microsoft Visual Studio, nerozezná vývojové prostředí MonoDevelop typ tohoto projektu a neotevře ho. Proto byl ve verzi po portování projekt instalátoru odebrán.

Nástroj MoMA odhalil při analýze assembly `MDExpress.exe` pouze jeden nedostatek, jak ukazuje následující obrázek 3.

Scan Date: 14.12.2009 14:04:21
 MoMA Definitions: Mono 2.4 (Revised)
 For descriptions of issues, see [MoMA Issue Descriptions](#).

Assembly	Version	Missing	Not Implemented	Todo	P/Invoke
⊟  MDExpress.exe	0.1.0.0	0	0	1	0
Calling Method		Method with [MonoTodo]		Reason	
void NastavFormular ()		string EncodingInfo.get_DisplayName ()			
Totals		0	0	1	0

Obrázek 3: Ukázka protokolu aplikace Mono Migration Analyzer

Jedná se o problém v getteru pro název kódování – vlastnost `DisplayName`. Ten je použit v souboru `ParstypForm.cs` na řádku 71. Podíváme-li se do zdrojových kódů Mona na příslušný soubor `EncodingInfo.cs`, zjistíme, že vrací hodnotu vlastnosti `Name`. Kód vypadá následovně:

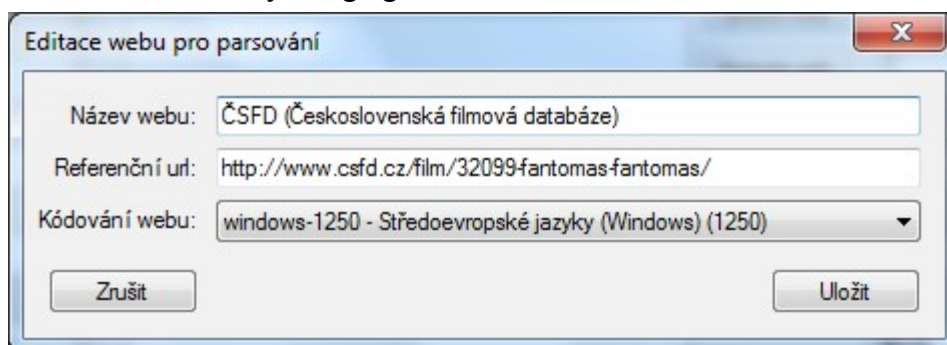
```
[MonoTODO]
public string DisplayName
{
    get { return Name; }
}
```

Ukázka 3: Nesprávné vrácení Name místo DisplayName ve zdrojovém souboru třídy EncodingInfo.cs

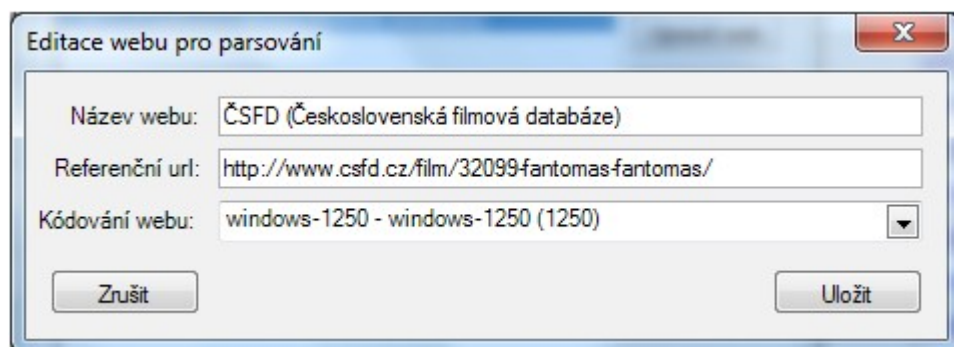
Atribut `MonoTODO` říká, že danou část budou teprve vývojáři implementovat. Naštěstí se zde nejedná o závažný nedostatek. Pouze místo „user-friendly“ názvu kódování bude zobrazen jeho zkrácený název. Funkčnost aplikace se tím však nikterak dramaticky neohrozí.

Nicméně aplikace stejně pod Monem nefungovala, ač MoMA nezjistil žádnou další problematickou část. Bylo to z důvodu použití knihovny pro práci s SQLite databází. Mono má naštěstí vlastní knihovny pro práci s SQLite a tak bylo řešení jednoduché. Namísto ADO.NET provideru `System.Data.SQLite` staženého ze stránek <http://sqlite.phxsoftware.com/>, byl v portované verzi použit provider `Mono.Data.Sqlite`. Bylo pak nutné ve třídě `Database.cs` vyměnit příslušnou řádku s příkazem `using` a zaměnit názvy tříd tak, aby jejich prefix nezačínal `SQLiteXxxx`, ale `SqliteXxxx`. Nakonec bylo ještě potřeba stáhnout a přidat k aplikaci knihovnu `sqlite3.dll` ze stránek <http://www.sqlite.org/>. Po této úpravě a novém překladu již aplikace fungovala správně.

Po portování se tedy aplikace lišila pouze po vizuální stránce. To je dáno rozdílnou implementací použitého grafického toolkitu WinForms v .NET a v Monu. Pro představu je na dalších obrázcích okno se zmiňovaným výběrem kódování. Nejprve před portováním, tedy běžící pod .NET Runtime a dále po portování běžící pod Mono Runtime. Obě dvě verze na operačním systému Windows 7. Za povšimnutí stojí zejména zmíněné chybné zobrazení názvu kódování a mírně odlišný design grafického toolkitu WinForms.



Obrázek 4: WinForms okno pod .NET Runtime (Windows 7)



Obrázek 5: WinForms okno pod Mono Runtime (Windows 7)

6 Porovnání GUI toolkitů

V této části práce jsou ke každému grafickému toolkitu uvedeny základní informace, které jsou pak v závěru použity pro jejich vzájemné porovnání. Nejprve zmíním WinForms a GTK+, neboť tyto toolkity jsou dostupné hned po instalaci Mona a fungují multiplatformně. Poté zmíním i ostatní toolkity, které je nutné doinstalovat zvlášť, případně je multiplatformně provozovat nelze nebo jen velmi složitě.

6.1 WinForms

Grafický toolkit WinForms je v podstatě sám již wrapperem pro nativní Win32 ovládací prvky na platformě Windows. V Microsoft .NET byl k dispozici již od počátku jeho uvedení. V projektu Mono je situace složitější. Neboť se jedná o wrapper k nativním prvkům OS Windows, bylo nutné vytvořit vlastní sadu komponent, které se vykreslují pomocí `System.Drawing` (`System.Drawing` kreslí pod Monem pomocí vektorové grafické knihovny Cairo).

Mono implementace WinForms, která se nazývá MWF (Managed Windows Forms) je tedy napsána od základů znovu. Možnosti grafického stylování prvků jsou tedy zatím (tj. ve verzi Mona 2.4.3) omezené pouze na strohý základní vzhled, pouze na Windows se kreslí jako nativní Windows ovládací prvky podle příslušného stylu. Oproti tomu je WinForms toolkitem, se kterým lze velice snadno a rychle pracovat. Nedrží se návrhového vzoru Model-View-Controller, takže pokud vytváříme jednoduché rozhraní, jeho tvorba a práce s ním je rychlejší např. oproti GTK+, protože nemusíme složitě vytvářet modely dat pro jednoduché komponenty jako `ListBox` a `ComboBox`.

Na webu projektu Mono je informace, že Mono podporuje kompletní WinForms podle .NET verze 2.0. To však není úplně pravda. Jak dokazuje stav implementace Mono frameworku v [9]. K 10. prosinci 2009 cca 30 členů (tří-

dy, metody či vlastnosti) zcela chybí a zhruba 150 jich vyhazuje výjimku `NotSupportedException`, tedy v podstatě chybí také.

WinForms obsahuje komponentu pro zobrazování HTML obsahu. Jedná se o komponentu `WebBrowser`, která v Microsoft .NET používá vykreslovací jádro Internet Exploreru. Jádro Internet Exploreru však není multiplatformní, proto se tato skutečnost v Monu řeší tak, že na každé platformě se použije vykreslovací jádro jiné - takové, které je dostupné, tedy buď Gecko, nebo WebKit. To však vede k zajímavému úkazu, že například funguje metoda `Navigate()` pro načtení URL, ale již nefunguje vlastnost `DocumentText`, která má vykreslit předaný HTML obsah ve formě řetězce. Stejně tak nefunguje ani vlastnost `DocumentStream` a přístup k DOM¹³ načteného dokumentu je pak také omezen.

Velkou výhodou při práci s tímto toolkitem je dostupnost originální dokumentace včetně ukázkových příkladů použití k `System.Windows.Forms` od Microsoft .NET na MSDN [10].

Jeho nevýhodou je absence návrháře rozhraní pro jinou platformu, než je Windows. WinForms mají svůj návrhář rozhraní obsažen ve vývojových prostředích MS Visual Studio a SharpDevelop. Existuje sice pokus o vytvoření návrháře v rámci projektu Mono, ale ten ještě není zcela kompletní a je dostupný pouze přes SVN¹⁴. Pokud tedy budeme potřebovat navrhnout rozhraní, nezbyvá nic jiného, než použít jedno ze zmíněných vývojových prostředí ve Windows nebo kód rozhraní napsat ručně. Návrháře rozhraní pro WinForms generují pouze zdrojový kód. V důsledku toho je třída pro každé navržené okno rozdělena do dvou částí. Je to tedy tzv. „partial class“. Jedna část obsahuje kód generovaný návrhářem a druhá pak uživatelský kód a případné obslužné metody událostí „event handlers“.

13 DOM = Document Object Model – objektový model dokumentu

14 SVN = Subversion – systém pro verzování a spravování zdrojových kódů

Ovládací prvky se většinou na formulář rozmísťujú pomocí absolutní pozice a ukotvením k okrajům nadřazeného prvku. Tento způsob je velmi jednoduchý a rychlý, ale má svá úskalí v podobě rozpadnutí rozhraní v případě např. změny velikosti písma v systému. Lze však použít rovněž `layouty`. K dispozici jsou `FlowLayoutPanel` a `TableLayoutPanel`. Pomocí nich se lépe drží zarovnání a rozložení komponent i v případě změny nastavení písma v systému.

6.2 GTK+

GTK+ je grafický toolkit vytvořený původně za účelem použití pro grafický editor GIMP. Odtud jeho zkratka GTK, která znamená „Gimp Toolkit“. Používá ho například grafické prostředí Gnome v Linuxu. V Monu jsou v něm vytvořeny např. aplikace Banshee, F-Spot nebo vývojové prostředí MonoDevelop. Samotný toolkit je šířen pod licencí LGPL.

Toolkit je napsaný v jazyce C a wrappery má pro mnoho dalších jazyků. Pro Mono jazyky se wrapper nazývá GTK#, někdy též GtkSharp. Je distribuován již v základní instalaci Mona. Je možné ho používat i s Microsoft .NET. Na stránkách Mona je možné momentálně stáhnout instalátor „GTK# for .NET“, který nainstaluje základní GTK knihovny a GTK# wrapper. Momentálně verze wrapperu 2.12 však značně zaostává za nejnovější verzí GTK+ 2.18 oproti wrapperům pro jiné programovací jazyky.

GTK+ se drží návrhového vzoru Model-View-Controller, je s ním tedy práce zdélhavější, ale o to „čistší“. Pro zobrazení dat prakticky slouží jedna komponenta (neboli widget) `Gtk.TreeView`, která zobrazuje buďto jako `ListBox`, `ListView` nebo `TreeView` v závislosti na tom, jaký model dat jí předáme.

Preferovaný způsob rozmístění komponent na formulář je použití `Layoutů`. K dispozici jsou `Gtk.Vbox` pro vertikální rozdělení prostoru, `Gtk.Hbox` pro horizontální rozdělení prostoru a `Gtk.Table` pro tabulkové rozložení. Je zde sice možnost použít jako u WinForms absolutní pozici pomocí widgetu `Gtk.Fixed`,

ale není možné prvky ukotvit navíc k pravému a spodnímu okraji jako u toolkitu WinForms. Navíc, znovu zdůrazňuji, je tento způsob nevhodný kvůli možné změně velikosti písma, změně jazyku a tedy popisků v aplikaci či změny GTK tématu. V tomto případě může snadno dojít k rozboření vzhledu aplikace.

Jak již bylo zmíněno, aplikace v GTK používají témata vzhledu, která je možné nastavit v systému ať už pomocí aplikace „GTK Theme Selector“ ve Windows nebo v Linuxu v příslušné konfigurační sekci pro nastavení vzhledu. V tématech vzhledu je možné nastavit vzhled prvků pomocí obrázků a nastavit jejich různé odstupy textů od okrajů prvků, velikosti písma a jiné parametry. Z vlastních zkušeností s tématy mohu říci, že výchozí GTK téma na Windows, jež se snaží splynout s vzhledem ovládacích prvků ve Windows, je asi nejlépe použitelné téma. Ostatní stažená témata z webu Gnome [12] nefungovala většinou spolehlivě a jejich vzhled tak nebyl příliš vábný.

Výhodou při použití tohoto grafického toolkitu je jeho dokumentace. Je sice pravda, že dokumentace k wrapperu GTK# dostupná pomocí instalovaného dokumentačního programu MonoDoc nebo online [13] není zcela kompletní, ale s výhodou lze používat dokumentaci k samotnému toolkitu GTK+. Navíc na stránkách projektu Mono je dostupné množství kvalitních tutoriálů, které popisují práci se základními prvky grafického rozhraní.

Dlužno dodat, že GTK+ používá k vykreslování svých ovládacích prvků knihovnu GDK (Gimp Drawing kit) a tedy každý widget má k dispozici vlastnost `GdkWindow`, na kterou lze příslušnými metodami kreslit.

K dispozici jsou zde 2 grafické návrháře rozhraní. Jedním z nich je Stetic, což je návrhář integrovaný do vývojového prostředí Monodevelop. Je to jakási obdoba návrháře pro toolkit WinForms ve vývojovém prostředí SharpDevelop. Návrhář si v podadresáři s projektem vytvoří další podadresář `gtk-gui` a zde uloží soubor `gui.stetic`. Jedná se o XML soubor s popisem rozhraní. Při sest-

vení rozhraní a kompilaci projektu však vytvoří ještě pro každé navržené okno nebo dialog soubor s kódem pro popis rozhraní obdobně jako WinForms. Z tohoto důvodu jsou třídy představující jednotlivá okna opět typu „partial class“ a tedy rozděleny do dvou částí - do části popisující rozhraní a do části s uživatelským kódem a handlers událostí.

Komu tento přístup nevyhovuje, může využít druhého editoru Glade. Glade není dodáván společně s Monem, ani není integrován ve vývojovém prostředí MonoDevelop. Je nutné ho doinstalovat zvlášť. Návrh rozhraní probíhá stejně jako v návrháři Stetic jen s tím rozdílem, že výstupem je jediný soubor s příponou `.glade`. Tento soubor má obdobný formát jako soubor `gui.stetic`. Připojení rozhraní k aplikaci se pak provádí pomocí třídy `Glade.XML`, jak je vidno v příslušné „Hello World“ aplikaci na přiloženém médiu.

Stetic má tu výhodu, že je integrován přímo ve vývojovém prostředí MonoDevelop a má velmi pěkně vyřešenou práci s akcemi a událostmi. Oproti tomu Glade má zase tu výhodu, že je při jeho použití možné rozhraní aplikace snadno změnit pouze výměnou Glade souboru ve složce s projektem bez nutnosti dalšího překladu aplikace, což může být vhodné, pokud chceme umožnit uživatelům snadný výběr stylů zobrazení aplikace.

6.3 Qt

Qt (vyslovováno jako cute, tedy „kjuť“), je toolkit vyvíjený společností Nokia, konkrétně její odkoupenou částí Trolltech. Jedná se o jeden z nejpokročilejších grafických toolkitů a to jak po stránce vizuální tak po stránce funkční. Je vydáván ve třech verzích:

- 1) komerční verze pouze pro práci s grafickým rozhraním,
- 2) komerční plná verze (s podporou pro síť, databáze, multithreading...),
- 3) open-source plná verze určená pouze pro vývoj open-source aplikací.

Od své čtvrté verze začal být nabitý nejrůznějšími funkcčnostmi. V současné verzi 4.6 uvolněné 1. prosince 2009 podporuje toolkit snadnou práci s SVG obrázky, PDF soubory, umožňuje integraci OpenGL, disponuje knihovnou Phonon pro snadnou práci s multimédií, frameworkem pro práci s vícevláknovými aplikacemi. Dále od verze 4.6 podporuje multitouch displeje, má framework pro animace, umí číst OpenDocument formát atd.

Stylovat prvky grafického rozhraní je jednoduše možné pomocí CSS stylů. Ke zobrazování HTML obsahu slouží komponenta QWebView s jádrem WebKit.

Qt je napsán v jazyce C++ a je tedy především cílen pro vývoj v tomto jazyce. Existují však wrappery pro nejrůznější jazyky, dokonce i pro PHP. Pro .NET existují v tuto chvíli dva aktuálně vyvíjené wrappery a to Qyoto a Qt4Dotnet.

Qyoto je momentálně zcela bez oficiální dokumentace API. Na stránkách projektu je pouze dostupný základní tutoriál o šesti lekcích datující se do roku 2006. V tutoriálu jsou navíc chyby (např. namísto třídy QObject použita třída Object a volání neexistující metody QPushButton.SetFont()). Chyby jsou zřejmě způsobeny tím, že tutoriál je určen pro jakousi starší verzi knihoven Qyoto. Zdá se tedy, že projekt je již mrtvý, ale podle nejrůznějších diskuzních fór na Internetu však vývoj pokračuje. Příklad aplikace, která Qyoto využívá, může být např. nově vznikající komunikátor Synapse (<http://synapse.im/>).

Qt4Dotnet je založen na wrapperu QtJambi pro Javu a jelikož používá IKVM, což je implementace Javy pro Mono, je možné použít ho i pro vývoj v Monu. Jedná se o nový projekt, jehož první verze byla vydána v únoru 2009. A dokumentací také příliš nedisponuje. Opět je na internetových stránkách dostupných pouze pár příkladů.

K návrhu rozhraní je možné použít aplikaci Qt Designer, která je součástí instalace knihoven Qt. Prvky se na formulář umísťují pomocí layoutů a tzv.

spacerů. K dispozici jsou Vertical layout, Horizontal layout, Grid layout, které více méně odpovídají layoutům VBox, Hbox a Table v GTK+. Navíc je zde Form layout, což je tabulka o dvou sloupcích určená speciálně pro tvorbu formulářů. Zarovnání a ukotvení prvků v rámci layoutů je pak řešeno pomocí horizontálního nebo vertikálního spaceru, který doslova „odpruží“ vše za ním na pravý, respektive spodní okraj nadřazeného layoutu. Pomocí dvou spacerů můžeme navíc ovládací prvky snadno vycentrovat. Uživatelské rozhraní je pak uloženo v XML podobě, kterou je možné opět připojit k aplikaci obdobně jako u GTK+ toolkitu.

Interakce s uživatelským rozhraním je zde řešena jinak než u WinForms a GTK+, kde byla řešena pomocí událostí a handlerů. Zde se interakce řeší způsobem více připomínajícím práci s toolkitem v C++, proto u Qt tříd u obou wrapperů nenajdeme události a možnost registrovat jejich posluchače. U ovládacích prvků je dostupná metoda `Connect()`, pomocí které propojíme tzv. signál ovládacího prvku s tzv. slotem, který signál obslouží.

Následující ukázka č. 4 zobrazí okno s jedním tlačítkem po jehož stisknutí se do konzole vypíše text „Hello World“. Metoda `Connect()` objektu `QWidget` má v příkladu 4 parametry, prvním je objekt, jehož událost (signál) budeme registrovat, druhým parametrem je samotný signál určený metodou `SIGNAL()` ze třídy `QObject`, třetím parametrem je reference na objekt, ve kterém bude handler (slot) a čtvrtý parametr je slot určený metodou `SLOT()` třídy `QObject`. Samotný kód slotu je pak uvozen atributem `Q_SLOT` s příslušnou signaturou. Pokud signaturu neuvedeme, což je také možné (definice atributu je přetížena), musí se slot, tedy obsluhující metoda, jmenovat stejně jako atribut metody `SLOT()`.

```
using Qyoto;
using System;

namespace QyotoTutorial
{
    public class HelloWorldWidget : QWidget
    {
        public HelloWorldWidget() : this(null) {}
        public HelloWorldWidget(QWidget parent) : base(parent)
        {
            QPushButton btn = new QPushButton(Tr("Hello"), this);
            Connect(btn, SIGNAL("clicked()"), this, SLOT("helloWorld()"));
            this.Show();
        }

        [Q_SLOT("helloWorld()")]
        public void HelloWorld()
        {
            Console.WriteLine("Hello World");
        }

        public static int Main(string[] args)
        {
            QApplication app = new QApplication(args);
            new HelloWorldWidget();
            return QApplication.Exec ();
        }
    }
}
```

Ukázka 4: Práce s událostmi v QT

Pokud se naučíme pracovat s tímto přístupem, bude nám již bránit v použití tohoto toolkitu pro tvorbu multiplatformních aplikací pouze jedna zásadní věc. Tou věcí je bohužel multiplatformnost wrapperů. Ač na stránkách projektu Mono je uvedeno, že Qyoto je multiplatformní, jsou s jeho zprovozněním jinde než na Linuxu veliké problémy. Jelikož není poskytována přímo hotová assembly, je ho nutné zkompilevat ze zdrojových kódů. Ani pak však nemáme zaručeno, že vše bude fungovat na 100 procent. Z vlastní zkušenosti mohu říci, že ani Qyoto, ani Qt4Dotnet se bez zásadních obtíží nezdařilo zprovoznit a když už ano, nefungoval například import XML rozhraní z .ui souboru.

Nezbývá, než doufat, že se časem multiplatformní podpora radikálně zlepší a bude možné tento jinak výborný grafický toolkit používat v Monu napříč všemi třemi platformami.

6.4 wxWidgets

Tento grafický toolkit je napsán v C++. Jeho základním rysem je to, že na každé platformě používá její nativní grafický toolkit. Nesnaží se tedy pouze o emulaci jeho vzhledu, jako např. GTK+, pomocí témat. Přímou používá konkrétní API jednotlivých toolkitů a tím docílí správného nativního vzhledu.

Naproti tomu je toolkit omezen faktem, že je možné používat pak pouze prvky, které jsou dostupné v nativních toolkittech pod všemi platformami a může být tak problém vložit nějaký speciální ovládací prvek. Z toho vyplývá i fakt, že mezi ovládacími prvky není použitelný takový, který by uměl plně zobrazit HTML obsah. Obsahuje pouze komponentu pro zobrazení HTML 3.2, která neumí stylovat pomocí CSS a skriptovat pomocí JavaScriptu.

K toolkitu existuje multiplatformní wrapper s názvem wx.NET, který je šířený pod licencí odvozené od LGPL. Balíček, který se stáhne, obsahuje knihovnu „wx-c“, která představuje nativní toolkit pro konkrétní platformu a wx.NET, což je assembly pro Mono nebo .NET.

Výhodou tohoto toolkitu je kvalitní dokumentace jak na stránkách wrapperu, tak na samotných stránkách grafického toolkitu wxWidgets. Dokumentace je včetně tutoriálů a ukázkových příkladů.

K návrhu rozhraní je možné použít návrhář *wxGlade* nebo návrhář *wxForm-Builder*, který je pokročilejší. Je však generován kód C++, který nám není nic moc platný. Zároveň je ještě generován výstup v XML, který pak můžeme opět použít v naší aplikaci. Prvky se na formulář umísťují pomocí layoutů a spacerů obdobně jako u Qt.

6.5 Cocoa

Cocoa je souhrnný název pro API společnosti Apple. Programují se v něm aplikace pro Mac OS X většinou za pomoci vývojového prostředí Xcode v jazyce Objective-C. Částí tohoto API je sada ovládacích prvků uživatelského rozhraní, kterou můžeme použít i pro vývoj aplikací v Monu.

K dispozici máme dvě zapouzdřující knihovny (wrappery). Prvním z nich je Cocoa#, který je distribuován v rámci projektu Mono ve verzi pro Mac OS X. Druhým je wrapper Monobjc, šířený zvlášť pod licencí LGPL. V CocoaSharp je vytvořen jednoduchý grafický HelloWorld a v Monobjc je vytvořeno rozhraní k aplikaci Mac OS X. Ovšem, jak již bylo zmíněno, pouze hlavní okno aplikace, neboť vývoj stěžovala jednak složitost, s jakou je nutné aplikace za použití Monobjc vyvíjet, a jednak nedostupnost dokumentace a příkladů použití přímo pro Monobjc. Existuje sice obsáhlá dokumentace pro Cocoa API v Objective C, nicméně ta pochopitelně neříká nic o tom, jak je funkcionality poskytovaná prostřednictvím Monobjc wrapperu.

K návrhu designu uživatelského rozhraní je potřeba návrhář Interface Builder, který je součástí instalace prostředí Xcode. Xcode je možné zdarma stáhnout z webu společnosti Apple. Z Interface Builderu uložíme rozhraní ve formátu „nib“, což je v podstatě adresář tvářící se jako soubor, který obsahuje další soubory s definicí grafického rozhraní. Část souborů je ve formátu XML a část v proprietárním formátu. Je proto lepší změny v rozhraní provádět pomocí návrháře Interface Builder.

Prvky se na formulář umísťují absolutně pozicované a ukotvují se k okrajům podobně jako ve WinForms, ovšem základní paleta ovládacích prvků je oproti WinForms širší.

Interakce uživatele s grafickým rozhraním je řešena obdobně jako u Qt. Signály (v Cocoa nazývány Actions) ovládacích prvků se napojují na sloty (v Cocoa

nazývané Outlets) jiných ovládacích prvků. Propojování Actions s Outlets v Interface Builderu je obdobné jako propojování signálů se sloty v Qt Designeru, tedy tažením myši.

K zobrazení HTML obsahu slouží komponenta `NSWebView` s jádrem `WebKit`, ostatně jak jinak, když toto jádro má svůj původ právě na platformě Mac OS X.

Na Mac OS X jsou aplikace reprezentovány pomocí tzv. „application bundle“. Je to adresář, který se ve Finderu tváří jako obyčejný soubor. Má pevně danou strukturu a poklepnutím myši se spustí aplikace, kterou zapouzdřuje. Z tohoto důvodu je vhodné použít ještě nástroj `NAnt`, který aplikaci po překladu snadno sestaví. Jak vypadá konfigurační soubor pro `NAnt` je možné vidět na příloženém médiu u projektu `FeedReader` a jeho `Cocoa` rozhraní.

Pro platformu Mac OS X je v zásadě vhodné používat nativní toolkit především kvůli menu. Menu aplikací je v Mac OS X odděleno od okna aplikace a zobrazováno na horní liště. Položky na liště se vždy mění podle toho, jaká aplikace (nebo přesněji jaké okno) má zaměření. Pokud však použijeme `GTK+` nebo `WinForms` toolkit, bude menu součástí aplikačního okna a v příslušné liště se zobrazí pouze položka menu s popiskem „mono“, což může být možná matoucí pro uživatele této platformy.

7 Nástroje pro vývoj

Zde je uveden přehled vývojových prostředí a dalších aplikací, které se mohou při vývoji multiplatformních aplikací v Monu hodit. Není uveden kompletní výčet všech použitelných nástrojů, neboť těch je nepřehledné množství a záleží na vývojáři, v čem raději pracuje. Doporučení jsou tedy odvislá od osobních zkušeností a preferencí.

7.1 Vývojová prostředí

7.1.1 MonoDevelop

MonoDevelop je multiplatformní (od verze 2.2) vývojové prostředí určené právě pro Mono. Vychází ze zdrojových kódů vývojového prostředí Sharp-Develop. Disponuje chytrým doplňováním kódu (tzv. intellisense) pro C# 3.0, modulem pro práci s databázemi, podporou pro lokalizaci projektů, integrovaným debuggerem. Je možné si přizpůsobit zvýrazňování syntaxe a sémantiky a automatické formátování kódu ve stylu vývojového prostředí MS Visual Studio a také umožňuje otevírat projekty v něm vytvořené. Zároveň má MonoDevelop integrovaný návrhář GTK+ rozhraní Stetic a integrovaný prohlížeč assembly. Jeho funkčnost je možné rozšířit pomocí dalších modulů Add-In.

Je šířen pod licencí GPL2 a jeho vývoj se v poslední době velmi urychlil a aplikace se stala celkem stabilní oproti předchozí verzi 1.0. Na stránkách produktu [14] je dostupná dokumentace včetně screencastů, seznamujících s tímto vývojovým prostředím.

Kromě vývoje desktop aplikací v C#, Vala, Boo, VB.NET a C/C++, podporuje rovněž vytváření projektů v ASP.NET a jejich testování na XSP serveru.

7.1.2 SharpDevelop

Vývojové prostředí SharpDevelop doporučím především kvůli tomu, že má použitelný WinForms designer (oproti Mono WinForms Designeru, který je zatím ve stádiu vývoje). Pomocí něho je možné nechat si vygenerovat pouze kód, který vytváří a nastavuje formuláře a ovládací prvky a tento kód pak přenést do vývojového prostředí MonoDevelop a pokračovat ve vývoji zde.

Jelikož se jedná o projekt příbuzný MonoDevelopu disponuje velmi podobnou funkcionalitou. Navíc podporuje programovací jazyky IronPython a F#, obsahuje integrovanou podporu pro systém správy verzí Subversion a umožňuje vytvářet také instalační „Setup“ projekty.

Vývojové prostředí je šířeno pod licencí LGPL.

7.1.3 Microsoft Visual Studio

Pokud má dojít při vývoji na komerční produkty, lze jednoznačně doporučit vývojové prostředí Microsoft Visual Studio 2008 (případně verzi 2010, která je nyní ve verzi Beta 2 a ještě občas při práci zamrzá). Jedná se o bezesporu nejpokročilejší vývojové prostředí pro .NET a doinstalováním přídatku „Mono Tools“ můžeme naše aplikace testovat a spouštět i v Mono Runtime. Nutno říci, že software „Mono Tools“ je rovněž komerční záležitostí.

7.2 Návrháře grafických rozhraní

7.2.1 Glade

Nezmiňuji zde návrhář Stetic, neboť ten je součástí vývojového prostředí MonoDevelop a pracuje se s ním obdobně jako s návrhářem Glade. Osobně jsem při vývoji používal raději návrhář Glade, především z důvodu oddělení rozhraní do jednoho XML souboru, který lze snadno připojit k aplikaci

a kdykoliv podle potřeby obměnit bez nutnosti opětovného přeložení aplikace. Onen soubor může obsahovat i více oken téže aplikace.

Ve své třetí verzi, se kterou jsem pracoval, podporuje dva formáty výstupních XML souborů a to GtkBuilder a libglade. GtkBuilder bohužel zřejmě není možné použít v GTK#¹⁵, ač je tento formát novější. To se snad změní v další verzi GTK#, neboť Mono obsahuje náznak jakýchsi nezdokumentovaných tříd souvisejících s formátem GtkBuilder.

Dlužno podotknout, že XML soubor poskytovaný grafickým návrhářem Glade obsahuje mnohdy vlastnosti některých prvků, které nejsou ve wrapperu GTK# k dispozici. Toto je možné částečně omezit volbou takové verze rozhraní GTK+ v návrhář Glade, aby korespondovala s verzí GTK+, pro níž máme dostupný wrapper GTK#. Některá varování se však odstranit nepodaří ani tak a nezbývá než tyto sporné vlastnosti odstranit z výsledného vygenerovaného XML souboru s rozhraním ručně, nebo se smířit s jejich výpisem do konzole.

```
(Watcher:5700): libglade-WARNING **: unknown property `orientation'
for class `GtkVBox'
```

*Ukázka 5: Varování upozorňující na neexistující vlastnost orientation
pro widget Gtk.VBox*

Dalším omezením návrháře Glade je nemožnost používat jiné ovládací prvky, než ty ze základní sady a proto pokud chceme např. použít `Gtk.WebControl`, nebo `Gtk.WebView` pro zobrazení HTML obsahu, popřípadě nějakou vlastní komponentu, musíme nechat příslušný kontejner v layoutu rozhraní prázdný a tento ovládací prvek vložit až programově v naší aplikaci. Oproti tomu návrhář Stetic umožňuje použití vlastních ovládacích prvků, jak dokazuje screencast „*Creating custom widgets with MonoDevelop*“ na stránkách [13].

¹⁵ Třídy pro import rozhraní ve formátu GtkBuilder jsou sice k dispozici, ale chybí k nim dokumentace.

7.2.2 Qt Designer

Pokud se přeneseme přes nutnost složité kompilace wrapperu Qyoto a rozhodneme se použít grafický toolkit Qt, můžeme k návrhu rozhraní použít velice pěkný návrhář Qt Designer. Ten je distribuován spolu s open-source verzí Qt a funguje tedy pod Windows, Linuxem i Mac OS X. Jeho výstupem je soubor s příponou .ui, což je XML soubor s popisem rozhraní, obdobný jako u návrháře Glade.

Pěkná vlastnost tohoto návrháře je možnost náhledu na navrhované rozhraní v různých motivech (Windows XP, Vista, Motif, CDE, ClearLooks ...). Intuitivní je rovněž práce se signály a sloty. Umožňuje tažením myši spojit signál určitého ovládacího prvku (např. tlačítka) se slotem jiného ovládacího prvku, (např. okna) a bez napsání řádku kódu tak registrovat událost např. pro zavření okna při kliknutí na tlačítko.

7.3 Ostatní software

7.3.1 Mono Migration Analyzer (MoMA)

Tento nástroj usnadňuje portování aplikací z Microsoft .NET do Mono. V aplikaci se otevře jedna nebo více assembly (exe, dll) a ta dále vyhledává platformě závislá volání a přístupy či volání v Monu neimplementovaných metod nebo používání neimplementovaných tříd. Po analýze předaných assembly je zobrazen protokol, který odhaluje nalezené problematické části kódu. Na tento nástroj se však nelze spoléhat ve všech ohledech, jak dokazuje portování aplikace napsané původně pro MS .NET „Movie Database Express“.

7.3.2 Virtualizační nástroje (VirtualBox, VMWare Player)

Při vývoji aplikací je nutné vyvíjenou aplikaci testovat a ladit průběžně na všech platformách. Abychom si tento proces usnadnili, můžeme použít virtualizační nástroje.

Osobně bych doporučil velmi povedený nástroj Sun VirtualBox. Vyvíjíme-li např. na OpenSuse Linuxu pod KDE, můžeme si na virtuální počítač nainstalovat Windows XP, Windows 7 či např. Ubuntu Linux, nasdílet si v těchto virtualizovaných systémech složku s projektem na hostitelském počítači a při kompilaci vyvíjené aplikace pak snadno jedním či dvěma kliknutími spustit aplikaci na různých systémech. Je to rozhodně rychlejší způsob, než vždy startovat jiný systém, tam aplikaci otestovat a pak se pracně přepnout zpět.

Ještě zde uvádím VMWare Player a to z tohot důvodu, že je možné pod ním zprovoznit Mac OS X v oné nelegální verzi pro PC. Osobní zkušenosti s tímto způsobem však dobré nemám, neboť takto virtualizovaný Mac OS X je obrovsky pomalý. V tomto případě je lepší použít druhé PC, nebo ideálně pokud máme počítač Apple.

8 Sada doporučení pro Mono programátory

V této části své práce uvádím přehled doporučení začínajícím Mono programátorům. Doporučení jsou sepsána na základě zkušeností s vývojem třech výše uvedených testovacích aplikací.

8.1 Nutná perfektní aktivní znalost angličtiny

Jakkoliv se toto doporučení může zdát samozřejmé a může znít až jako redundantní, považuji za nezbytné ho zmínit.

Pro mnoho technologií existuje řada českých knih, většinou překladů z anglicky psaných originálů. Příkladem budiž kniha *Inside C#* [1], která vyšla ve svém prvním vydání také v českém překladu pod názvem „Myslíme v jazyku C#“. Většinou k technologiím, které nejsou zatím příliš rozšířené nebo se teprve vyvíjejí a získávají své místo v programátorském světě, knihy v českém jazyce nejsou dostupné. Takové knihy se příliš finančně nevyplácí psát a už vůbec ne překládat. Proto je nabíledni, že k projektu Mono v češtině žádné knihy v tuto chvíli neseženeme a budeme odkázáni pouze na anglicky psanou literaturu.

Zároveň je těžko dostupná nebo i neúplně vypracovaná dokumentace jak k Mono frameworku, tak především ke grafickým toolkitům. Z toho důvodu, ocitneme-li se v úzkých a budeme potřebovat radu, nezbude nám nic jiného, než napsat dotaz do příslušného diskuzního fóra (anglicky), přečíst si odpovědi a umět tak komunikovat na technické úrovni v tomto jazyce.

8.2 Sledovat stav implementace tříd a jejich metod

Zde bych si dovolil ještě jednou upozornit na inkonzistenci mezi tím, co je podle internetových stránek projektu Mono implementováno, a skutečným stavem implementace. Je proto dobré sledovat stav implementace na *Mono*

Class Status Pages [9]. Je zde udáván počet chybějících nebo neimplementovaných členů tříd v porovnání k různým verzím .NET frameworku.

Konzultace s tímto přehledem však vyžaduje velmi přesnou implementační znalost vyvíjeného programu. A nutno říci, že ani informace zde obsažené někdy nemusí zcela postačovat. Jak dokazuje vývoj aplikace *Watcher*, nebyla metoda `Graphics.CopyFromScreen()` implementována pouze pod operačním systémem Mac OS X, ač se v *Mono Class Status Pages* tvářila jako plně implementovaná.

Je tedy možné, že i když budeme před vývojem pečlivě analyzovat implementaci tříd podle *Mono Class Status Pages*, může nás při vývoji překvapit výjimka `NotImplementedException` tam, kde bychom ji nečekali a bude pak nutné požadovanou funkčnost implementovat jiným způsobem. Toto beru jako zásadní nevýhodu *Mona*, neboť je při vývoji pak nutné chybějící implementace různě obcházet. Nezbývá než doufat, že se stav implementace bude rychle měnit k lepšímu.

8.3 Grafický toolkit volit s ohledem na majoritní cílovou platformu

Pokud programujeme nějakou aplikaci, jistě nám musí být zřejmé, jací uživatelé ji budou používat a jaké operační systémy používá tato cílová skupina uživatelů. Pokud u naší cílové skupiny výrazně převažuje použití nějakého operačního systému (tedy nejspíše Windows), můžeme zvolit grafický toolkit s ohledem na tento fakt. Neznamená to však, že ostatním uživatelům znemožníme aplikace zcela používat. Ostatní uživatelé maximálně dostanou ne příliš vábný vzhled.

Vezměme třeba toolkit WinForms, jakožto nativní na majoritní cílové platformě Windows. Jak dokazuje portovaná aplikace „Movie Database Express“, její funkčnost je totožná, pouze grafický toolkit se na jiných platformách vykres-

luje hůře. A i tomuto se dá předejít vytvoříme-li si například vlastní vykreslovací knihovnu pro WinForms prvky. Takový postup se nám však vyplatí nejspíše pouze pokud takových aplikací budeme vyvíjet více.

Jako nejvíce konzistentní po stránce vzhledu se jeví grafický toolkit GTK+. Proto bych pod Monem doporučil používat tento toolkit, pokud to bude možné.

Problém pak nastává při použití komponenty pro zobrazení HTML obsahu. Pak je buďto nutné vyřešit zobrazování jiným prvkem na platformách, kde není tato komponenta dostupná, nebo rozdělit grafická rozhraní pro každou platformu zvlášť. To však vyžaduje provést maximální abstrakci uživatelského rozhraní od aplikační logiky a používat grafické rozhraní pouze jako „observer“ pro sledování stavu této aplikační logiky a pro předávání událostí ovládacích prvků rozhraní zpět aplikační logice.

8.4 Testovat a ladit průběžně na všech platformách

Je velmi důležité vyvíjenou aplikaci testovat průběžně na všech platformách. Právě vzhledem k možným chybějícím funkcionalitám na konkrétní platformě, jako například metoda `Graphics.CopyFromScreen()`. Bylo by nemilé napsat funkční aplikaci pod Linuxem a poté zjistit, že nikde jinde nefunguje a že naše aplikace vůbec není multiplatformní.

K tomuto účelu je dobré používat virtualizační nástroje jako VMWare Player nebo VirtualBox. Trochu problém nastává s Mac OS X. Naštěstí existuje několik verzí tohoto operačního systému, které jsou schopné běhu na běžném PC. Souhrnně se označují jako OS X86. Problémem je však legálnost takto provozovaného operačního systému. Ovšem pro pouhé testování a ladění aplikací je to pořád levnější varianta, než si pořizovat Apple. Tento systém lze nainstalovat na VMWare Player, ale jeho použitelnost se pak blíží k nule, protože virtualizovaně funguje dramaticky pomalu. Je tedy lepší nainstalovat ho na speciální

připravený diskový oddíl a ani tehdy není zaručena 100% kompatibilita s veškerým PC hardwarem.

8.5 Vyhnout se používání platformně závislých přístupů

Je dobré vyvarovat se importování nativních funkcí z `.dll` nebo `.so` knihoven. Pokud už tak učiníme, je třeba příslušnou funkčnost najít v nativním API všech operačních systémů a ošetřit, aby se pro každou platformu zkompiloval příslušný import.

Dále je důležité vzít v potaz, že cesty k různým systémovým adresářům se v různých operačních systémech různí. Obecně se liší vlastně všechny cesty. Například na Windows se k oddělování cest používá zpětné lomítko „\”, zatímco na Linuxu a na Mac OS X se používá dopředné lomítko „/“. Kvůli tomu je k dispozici konstanta `Path.DirectorySeparator`, která na každé platformě vrátí příslušný separátor. Cesty k uživatelským datům, k aplikaci samotné či k temp souborům se rovněž liší.

Na přiloženém médiu je aplikace „Paths“, která vypisuje systémové cesty na dané platformě. Tabulka v příloze B udává přehled důležitých systémových cest na všech třech testovaných platformách.

8.6 Nespoléhat na `System.Drawing.SystemColors`

Třída `SystemColors` využívá hodnoty ve třídě `KnownColors`. Zde jsou napevno nadefinovány základní barvy, které se pak prostřednictvím `SystemColors.Xxxx` vrací. Tyto barvy se správně nastaví pouze na operačním systému Windows, na Linuxu a Mac OS X jsou vždy stejné ať máme v KDE nebo Gnome nastavené jakékoliv barevné téma. Na přiloženém médiu je aplikace `SystemColors`, kterou je možné si barvy otestovat na libovolné platformě.

8.7 Nebát se zdrojových kódů Mona

Někdy je dobré sáhnout po zdrojových kódech Mona. Stáhnout zdrojové kódy je možné v sekci *Download – Mono 2.4.3 Sources* na stránkách projektu. Verze se pochopitelně bude lišit. Archiv má cca 26 MB a po rozbalení má téměř 300 MB. Zdrojové kódy tříd Mono frameworku dělené podle jmenných prostorů do adresářů nalezneme v podadresáři `/mcs/class/`.

A důvody proč do zdrojových kódů nahlížet jsou dva. Prvním důvodem je, pokud nebudeme vědět, proč nám nefunguje použití nějaké třídy nebo metody, můžeme snadno nahlédnout, jak je implementována a hlavně zjistit, zda je vůbec implementována. Druhým důvodem jsou pak komentáře vývojářů. Pokud totiž chybí dokumentace k nějaké metodě v MonoDoc, může k objasnění posloužit klasický komentář, který ve zdrojových kódech zanechali vývojáři Mona.

9 Závěr

Prioritním cílem této práce bylo prověřit vhodnost projektu Mono pro multiplatformní programování. Proověření bylo provedeno tvorbou třech různých aplikací. Připomeňme, že se jednalo o konzolovou aplikaci, aplikaci s rozhraním GTK+ a aplikaci pracující se sítí, databází, vlákny a mající rozdělená grafická uživatelská rozhraní pro každou platformu zvlášť.

Vzhledem k rozdílné implementaci Mono na různých platformách a mnohdy chybějícím implementacím tříd a metod, bylo těžko možné docílit požadované funkčnosti aplikace pod Windows, Linuxem a Mac OS X. Po předchozí zkušenosti s Javou mohou říci, že aplikace napsaná v Javě je ve většině případů ihned plně funkční na systému, kde je dostupná Java Virtual Machine a to včetně aplikací s grafickým uživatelským rozhraním v toolkitu Swing.

Nutno říci, že Java Virtual Machine je dostupná prakticky v repozitářích softwaru všech linuxových distribucí. Mono však může být v repozitářích mnohých distribucí buďto zastaralé, což býval příklad Ubuntu nebo jiných distribucí od něho odvozených. Naštěstí již s novou verzí Ubuntu 9.10 se verze aktualizovala. Na některých distribucích je nutné však Mono kompilovat ze zdrojových kódů, což podstatně zesložituje nasazení aplikací na této cílové platformě.

Java má oproti Monu rovněž k dispozici větší výběr kvalitnějších vývojových prostředí. Zmíním zde alespoň vývojové prostředí Eclipse a výborné vývojové prostředí českého původu NetBeans. Tato vývojová prostředí jsou rovněž multiplatformní.

Přichází tedy logicky na mysl otázka, proč vlastně používat Mono? Je třeba říci, že nejen pro Mono Runtime, ale i pro Java Virtual Machine můžeme programovat a překládat bytekód ve více jazycích (Ada, Scala...). V projektech

lze pak jazyky kombinovat. O tom, zda je Java lepší nebo horší než C#, se vedou spory a je dostupných mnoho srovnání, např. [16]. Pravdou je, že každému bude vždy vyhovovat něco jiného a proto zde tyto dva jazyky nebudu porovnávat. Osobně mohu říci, že mě více vyhovuje jazyk C# a to třeba už kvůli možnosti používat chytré proměnné, neboli vlastnosti (properties) a především kvůli frameworku, který se mi u .NETu potažmo Mona zdá lépe propracovaný.

V Monu je rovněž možné vytvářet aplikace v ASP.NET, které pak můžeme provozovat pod serverem Apache s modulem `mod_mono`. Mono tedy může poskytnout free alternativu k Microsoft IIS a .NET i na serveru. Bohužel editor pro ASP.NET se zatím pro vývojové prostředí MonoDevelop teprve vyvíjí [17].

V Monu je dále možné vytvářet aplikace pro Silverlight. Mono implementace této API se jmenuje Moonlight. Prozatím je však implementováno API Silverlightu pouze ve verzi 1.0.

Druhým cílem bylo porovnání grafických toolkitů se kterými lze v Monu pracovat. Z tabulky v příloze A je možné vyčíst toto porovnání dle zvolených kritérií. S Monem jsou dodávány multiplatformní wrappery pro WinForms a GTK+, proto je jejich použití doporučeno a hlavně pohodlnější než instalovat knihovnu jinou nebo jí ještě navíc kompilovat ze zdrojových kódů. K doporučení vývojářů Mona se tímto přikláním a rovněž doporučím především toolkit GTK+.

Ohledně dalších dvou cílů, tedy doporučení vývojových a dalších užitečných nástrojů a doporučení začínajícím programátorům, snad mohu dodat pouze to, že jsou úměrná mým zkušenostem získaným s Monem během programování schopnosti prověřujících aplikací. Necht' poslouží ostatním jako základ pro další testování Mona.

Závěrem si dovoluji pár subjektivních dojmů. Mono a také vývojové prostředí MonoDevelop se jeví jako velmi rozpracované a rostou prakticky pod rukama

programátora. To je dáno především tím, jak je koncipován vývoj open-source softwaru, kde vychází software prakticky okamžitě, jakmile začne být vyvíjen, zatímco u komerčních aplikací se čeká na první stabilní vydání. S Monem více-méně zkouším experimentovat už asi 2 roky a od té doby bylo učiněno mnoho pokroků, jako např. ve funkčnosti komponenty `WebBrowser` pro WinForms nebo po stránce stability a použitelnosti vývojového prostředí MonoDevelop.

Je trochu škoda, že Mono bude vždy nejméně jeden krok za Microsoft .NET, neboť se snaží o kopii jeho funkčnosti. S nástupem technologie Windows Foundation se však dá očekávat, že se vývoj může i trochu odchýlit jiným směrem, neboť podle internetových stránek projektu se zatím neuvažuje ani do budoucna s implementací Windows Presentation Foundation, který zřejmě dříve či později nahradí starší grafický toolkit WinForms. Nezbyvá tedy než doufat, že na toto vývojáři zareagují například zlepšením podpory pro Qt.

Zda tedy použít, či nepoužít Mono pro vývoj multiplatformních aplikací, záleží na typu a rozsahu aplikace. Pro jednodušší aplikace je možné ho bez problémů použít, u komplexnějších aplikací bych sáhl asi raději po Javě (pokud trváme na interpretovaném jazyku), kde předpokládám menší časové náklady potřebné k odladění aplikací na všech platformách.

Aplikace které používají Mono se však pomalu začínají objevovat s tím, jak roste jeho použitelnost.

10 Seznam použité literatury

- [1] ARCHER, Tom, WHITECHAPEL, Andrew. *INSIDE C#*. 2nd edition. U.S.A : Microsoft Press, 2002. 912 s. ISBN 0-7356-1648-5.
- [2] BORNSTEIN, Niel M., DUMBILL, Edd. *Mono : A Developer's Notebook*. [s.l.] : O'Reilly, 2004. 302 s. ISBN 0-596-00792-2
- [3] MAMONE, Mark. *Practical Mono*. U.S.A : Apress, 2006. 403 s. ISBN 1-59059-548-3.
- [4] *Net Applications : Market Share* [online]. 2009, November 2009 [cit. 2009-12-10]. Dostupný z WWW: <<http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=8>>.
- [5] *TOPLIST : Globální statistiky* [online]. 2009. [cit. 2009-12-10] Dostupný z WWW: <<http://www.toplist.cz/global.html>>.
- [6] *Programovací jazyky* [online]. [2004] [cit. 2009-12-10]. Dostupný z WWW: <<http://k-prog.wz.cz/progjaz/index.php>>.
- [7] *.NET Languages* [online]. c2009, 06.30.2008 [cit. 2009-12-10]. Dostupný z WWW: <<http://www.dotnetlanguages.net/DNL/Resources.aspx>>.
- [8] *Mono - Getting Started* [online]. [2009] [cit. 2009-12-10]. Dostupný z WWW: <<http://mono-project.com/Start>>.
- [9] *Mono Class Status Pages* [online]. [2009] [cit. 2009-12-10]. Dostupný z WWW: <<http://go-mono.com/status/>>.
- [10] *MSDN - System.Windows.Forms* [online]. c2009 [cit. 2009-12-10]. Dostupný z WWW: <<http://msdn.microsoft.com/en-us/library/system.windows.forms.aspx>>.

- [11] *What is GTK+* [online]. c2008 [cit. 2009-12-10]. Dostupný z WWW: <<http://www.gtk.org/>>.
- [12] *Gnome Art - Themes* [online]. c2009 [cit. 2009-12-10]. Dostupný z WWW: <<http://art.gnome.org/themes/gtk2>>.
- [13] *Mono Documentation Library* [online]. [2009] [cit. 2009-12-10]. Dostupný z WWW: <<http://go-mono.org/docs/>>.
- [14] *MonoDevelop Documentation* [online]. 2009 , 25 Mar 2009 [cit. 2009-12-10]. Dostupný z WWW: <<http://monodevelop.com/Documentation>>.
- [15] *SharpDevelop - The Open Source Development Environment for .NET* [online]. c2009 [cit. 2009-12-10]. Dostupný z WWW: <<http://www.ic-sharpcode.net/OpenSource/SD/>>.
- [16] DARE OBASANJO. *A comparison of Microsoft's C# programming language to Sun Microsystems' Java programming language* [online]. 2007 [cit. 2009-12-10]. Dostupný z WWW: <<http://www.25hoursaday.com/CsharpVsJava.html>>.
- [17] *Mono - ASP.NET Visual Designer* [online]. [2009] [cit. 2009-12-10]. Dostupný z WWW: <<http://www.mono-project.com/AspNetEdit>>.

11 Seznam příloh

- A Tabulka porovnání grafických toolkitů podle zvolených kritérií
- B Tabulka s přehledem cest u Windows, Linuxu a Mac OS X
- C Postup kompilace Mona ze zdrojových kódů na distribuci Ubuntu
- D Médium se zdrojovými kódy popsaných aplikací a elektronickou verzí této práce ve formátu PDF.

12 Přílohy

Příloha A

Tabulka porovnává základní rysy grafických toolkitů podle zvolených kritérií.

Grafický toolkit	WinForms	GTK+	Qt	wxWidgets	Cocoa
Wrapper	MWF	GTK#	Qyoto Qt4Dotnet	wx.NET	Cocoa# Monobjc
Multiplatformní wrapper	Ano	Ano	Ano, ale nutná složitá kompilace	Ano	Ne
Grafický návrhář	Sharpdevelop + Visual Studio + Mono WinForms Designer	Stetic + Glade	Qt Designer	wxFormBuilder + wxGlade	Xcode Interface Builder
Umístění ovládacích prvků	Absolutně s ukotvením + 2 Layouty	Layouty + Absolutně bez ukotvení	Layouty v kombinaci se Spacery a absolutním pozicováním	Layouty	Absolutní pozice s ukotvením
Dostupnost dokumentace (1-5)	1	2	4	2	3
Komponenta pro zobrazení HTML (Jádro)	WebBrowser (Gecko nebo Webkit)	WebControl (Gecko) + WebView (WebKit)	QWebView (WebKit)	wxHtmlWindow = Pouze HTML 3.2 bez CSS a bez JavaScriptu	WebView (WebKit)
Licence	MIT X11	MIT X11	GNU GPL (Qyoto) + GNU LGPL (Qt4Dotnet)	wxWindows Licence = odvozená od LGPL	MIT X11 (Cocoa#) LGPL (Monobjc)
Možnosti stylování	Zatím není, plánuje se + má nativní vzhled na Windows	GTK Témata	Qt Style Sheets (CSS)	Nativní vzhled na každé platformě	Nativní vzhled na Mac OS X
Jazyk ve kterém je toolkit vyvíjen	C++	C	C++	C++	Objective-C
Dostupnost	Mono	Mono	Zvlášť + nutná kompilace	Zvlášť	Mono (Cocoa#) Zvlášť (Monobjc)

Tabulka 2: Porovnání grafických toolkitů podle zvolených kritérií

Příloha B

Tabulka udává přehled cest výčtového typu `Environment.SpecialFolder` pro všechny tři testované platformy. Cesty jsou získány z návratové hodnoty...

`Environment.GetFolderPath(Environment.SpecialFolder.Xxxx)`

... kde `Xxxx` je název speciálního adresáře. Aplikaci, která tyto hodnoty vypisuje je možné nalézt na přiloženém médiu pod názvem `Paths`.

V tabulce je vždy uvedena hodnota výčtového typu představující speciální adresář a pak cesta k němu ve Windows, Linuxu a Mac OS X.

Speciální Adresář	Cesta ke speciálnímu adresáři na různých operačních systémech v pořadí Windows 7, Linux Mint, Mac OS X
ApplicationData	C:\Users\<uzivatel>\AppData\Roaming
	/home/<uzivatel>/.config
	/Users/<uzivatel>/.config
CommonApplicationData	C:\ProgramData
	/usr/share
	/usr/share
CommonProgramFiles	C:\Program Files\Common Files
	-
	-
Cookies	C:\Users\Chloris\AppData\Roaming\Microsoft\Windows\Cookies
	-
	-
Desktop	C:\Users\<uzivatel>\Desktop
	/home/<uzivatel>/Plocha
	/Users/<uzivatel>/Desktop
DesktopDirectory	C:\Users\<uzivatel>\Desktop
	/home/<uzivatel>/Plocha
	/Users/<uzivatel>/Desktop
Favorites	C:\Users\<uzivatel>\Favorites
	-
	-

History	C:\Users\<uzivatek>\AppData\Local\Microsoft\Windows\History
	-
	-
InternetCache	C:\Users\<uzivatek>\AppData\Local\Microsoft\Windows\Temporary Internet Files
	-
	-
LocalApplicationData	C:\Users\<uzivatek>\AppData\Local
	/home/<uzivatek>/.local/share
	/Users/<uzivatek>/.local/share
MyComputer	-
	-
	-
MyDocuments	C:\Users\<uzivatek>\Documents
	/home/<uzivatek>
	/Users/<uzivatek>
MyMusic	C:\Users\<uzivatek>\Music
	/home/<uzivatek>/Hudba
	/Users/<uzivatek>/Music
MyPictures	C:\Users\<uzivatek>\Pictures
	/home/<uzivatek>/Obrázky
	/Users/<uzivatek>/Pictures
Personal	C:\Users\<uzivatek>\Documents
	/home/<uzivatek>
	/Users/<uzivatek>
ProgramFiles	C:\Program Files
	-
	-
Programs	C:\Users\<uzivatek>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs
	-
	-
Recent	C:\Users\<uzivatek>\AppData\Roaming\Microsoft\Windows\Recent
	-
	-
SendTo	C:\Users\<uzivatek>\AppData\Roaming\Microsoft\Windows\SendTo
	-
	-
StartMenu	C:\Users\<uzivatek>\AppData\Roaming\Microsoft\Windows\Start Menu
	-
	-

Startup	C:\Users\<uzivate/>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
	-
	-
System	C:\Windows\system32
	-
	-
Templates	C:\Users\<uzivate/>\AppData\Roaming\Microsoft\Windows\Templates
	-
	-

Tabulka 3: Přehled cest ke speciálním adresářům

Příloha C

Překlad Mona ze zdrojových kódů na Ubuntu Linuxu

Zdrojové kódy Mona najdeme na stránkách projektu a to v sekci „Download“ vpravo odkaz „Mono 2.4 Sources“ (číslo verze se pochopitelně může lišit s časem). Tento odkaz nás pošle na adresu <http://ftp.novell.com/pub/mono/sources-stable/>. Stáhneme následující soubory (čísla verzí se opět mohou lišit):

- mono-2.4.tar.bz2
- mono-basic-2.4.tar.bz2
- gluezilla-2.4.tar.bz2
- libgdiplus-2.4.tar.bz2

Dále bude potřeba doinstalovat některé balíčky. Pomocí balíčkovacího software „Synaptic“ (nebo v konzoli příkazy „aptitude“, „apt-get“) doinstalujeme především **make** a **gcc** a dále balíčky, o které si to samo požádá při kompilaci. Zpravidla jsou potřeba následující balíčky: **bison**, **gettext**, **libglib2.0-dev**, **libpango1.0-dev**, **libatk1.0-dev**, **libgtk2.0-dev**, **mozilla-dev**, **libxul-dev**, **libgif-dev**, **libjpeg62-dev**, **libtiff4-dev**, **libpng12-dev**, **libgtksourceview-dev**, **libreadline5-dev**, **libncurses5-dev** + u všech pochopitelně požadované závislosti.

Po nainstalování všech těchto balíčků se můžeme pustit do kompilace. Spustíme si konzoli a přesuneme se do adresáře, kam jsme stáhli zdrojové kódy. Nejprve rozbalíme mono-2.4.tar.bz2. Buď v Gnome správcem balíčků nebo v konzoli pomocí příkazu:

```
tar -xjvf mono-2.4.tar.bz2
```

Pak se přepneme do nově vytvořeného adresáře:

```
cd mono-2.4
```

nyní v adresáři s rozbaleným zdrojovým kódem, napíšeme v konzoli:

```
./configure
```

Tento příkaz provede otestování systému na přítomnost všech potřebných závislostí (pokud není např. nainstalován překladač **gcc**, tak vám to configure skript sdělí právě zde) a vytvoří soubory Makefile. Následuje příkaz:

```
make
```

Ten provede kompilaci a nakonec:

```
sudo make install
```

Tento příkaz soubory nainstaluje na správná místa. Nutno podotknout, že tento krok se provádí pod superuživatelem root, budete tedy vyzváni k zadání jeho hesla. Trojici příkazů „./configure“, „make“ a „sudo make install“ si zapamatujeme jako jakýsi „linuxový kompilační trojchvat“. Bude potřeba ještě dále.

Pokud nenastala žádná chyba, máme v tuto chvíli nainstalované Mono. Dále provedeme ty samé akce s dalšími balíčky. Tedy celý postup zopakujeme pro mono-basic-2.4.tar.bz2 (podpora pro jazyk Visual Basic):

```
tar -xjvf mono-basic-2.4.tar.bz2  
cd mono-basic-2.4  
./configure  
make  
make install
```

Pak to samé s libgdiplus-2.4.tar.bz2, což je GDI+ kompatibilní API pro jiné než Windows systémy:

```
tar -xjvf libgdiplus-2.4.tar.bz2  
cd libgdiplus-2.4  
./configure  
make  
make install
```

A nakonec gluezilla-2.4.tar.bz2. Gluezilla je knihovna k začlenění Mozilla Gecko engine do WinForms WebControl:

```
tar -xjvf gluezilla-2.4.tar.bz2
cd gluezilla-2.4
./configure
make
make install
```

Instalaci Mona ověříme jednoduchým příkazem, který vypíše verzi JIT:

```
mono --version

Mono JIT compiler version 2.4 (tarball Út dub 7
19:41:48 CEST 2009)
Copyright (C) 2002-2008 Novell, Inc and Contribu
tors. www.mono-project.com
  TLS:          __thread
  GC:           Included Boehm (with typed GC)
  SIGSEGV:      altstack
  Notifications: epoll
  Architecture: x86
  Disabled:     none
```

Dále ještě potřebujeme wrapper pro Gtk+ a Gnome knihovny, abychom mohli vytvářet a spouštět grafické aplikace pomocí Gtk+ toolkitu pod Monem.

Stáhneme tedy soubory:

- gtk-sharp-2.12.8.tar.bz2
- gnome-sharp-2.20.1
- gnome-desktop-sharp-2.20.1

a další postup je již zřejmý. Každý soubor příslušným příkazem rozbalit a aplikovat „kompilační trojchvat“.

Dále budeme potřebovat vývojové nástroje, abychom mohli nějak komfortně aplikace vyvíjet. I když pravda, že aplikace lze vyvíjet i v jednoduchém textovém editoru. Tak například pro vývoj webových aplikací v ASP.NET se jistě bude hodit XSP. XSP je jednoduchý webový server napsaný v C# a snadno se

v něm testují ASP.NET projekty. Budeme-li webové aplikace vyvíjet, stáhneme a nainstalujeme ještě:

- xsp-2.4.tar.bz2

Pokud však plánujeme nasadit naše webové aplikace do ostrého provozu, je lepší použít server Apache a přeložit si do něho modul `mod_mono`, který umožňuje provozovat ASP.NET aplikace.

- mod_mono-2.4.tar.bz2

Pro překlad modulu do Apache `mod_mono` bude potřeba doinstalovat ještě balíčky **apache2-threaded-dev** a **apache2-prefork-dev** a další vyžádané závislosti. V podstatě se tímto doinstaluje celý server Apache.

Dále je dobré nainstalovat ještě některé balíčky z části Development tools:

- mono-tools-2.4.tar.bz2
- gecko-sharp-2.0-0.13.tar.bz2
- gtksourceview-sharp-2.0-0.12.tar.bz2
- nant-0.86-beta1-src.tar.gz (pouze make a make install)
- boo-0.9.0.3203-2-src.zip (build pomocí "nant")
- ikvmbin-0.38.0.4.zip (implementace Javy pro Mono)
- mono-debugger-2.4.tar.bz2
- cocoa-sharp-0.9.5.tar.bz2 (wrapper pro Cocoa pro Mac OS X)
- IPCE-r7.zip (IronPython – implementace Pythonu pro .NET)
- mono-addins-0.4.zip

Nejsou pochopitelně potřeba všechny. Příkladně pokud nebudeme vyvíjet v Pythonu, nepotřebujeme IronPython. V Ubuntu dále není třeba cocoa-sharp.

Poslední aplikace, kterou budeme instalovat, bude vývojové prostředí MonoDevelop. Jeho zdrojové kódy jsou na téže stránce opět dostupné.

Stáhneme a nainstalujeme podle potřeby:

- **monodevelop-2.0.tar.bz2**
 - základní instalace prostředí MonoDevelop
- **monodevelop-boo-2.0.tar.bz2**
 - podpora pro jazyk Boo
- **monodevelop-database-2.0.tar.bz2**
 - podpora pro práci s různými databázemi
- **monodevelop-debugger-gdb-2.0.tar.bz2**
 - GNU debugger plugin pro MonoDevelop
- **monodevelop-debugger-mdb-2.0.tar.bz2**
 - Mono debugger plugin pro MonoDevelop
- **monodevelop-java-2.0.tar.bz2**
 - podpora pro jazyk Java
- **monodevelop-vala-2.0.tar.bz2**
 - podpora pro jazyk Vala

Tím by měla být instalace kompletní a v nabídce programů prostředí Gnome by se měl objevit odkaz na MonoDevelop. Postup bude obdobný pro všechny linuxové distribuce. Lišit se bude v podstatě pouze v práci s různými balíčkovacími systémy a jejich obslužnými aplikacemi.