Multiplatformní (Windows / Linux) GUI knihovna napsaná v jazyce C++

Multiplatform (Windows / Linux) GUI library written in C++ language

Petr Kobalíček

Bakalářská práce
2007

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2006/2007

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Petr KOBALÍČEK**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Multiplatformní (Windows / Linux) GUI knihovna napsaná v jazyce C++**

Zásady pro vypracování:

*** Proveďte analýzu a srovnání vlastností existujících multiplatformních GUI knihoven, implementovaných v C/C++**
*** Implementujte vlastní GUI knihovnu, která bude obsahovat:**
o Abstrakční třídy pro práci s řetězci, souborovým systémem a základní algoritmy které vytvoří základ pro psaní uživatelského rozhraní
o Podporu vláken (threads)
o Vnitřní (jednoduchou) podporu XML pro načítání / ukládání konfigurace a parsování XML dokumentů
o Kreslící subsystém, který umožní jednotný grafický výstup na všech podporovaných platformách
o Načítání a ukládání jpeg, png, pcx a bmp obrázků

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

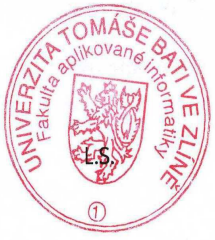
Qt knihovna – http://www.trolltech.com/
Gtk+ knihovna – http://www.gtk.org/
Fltk knihovna – http://www.fltk.org/
Jpeg knihovna – http://www.ijg.org/
Png knihovna – http://www.libpng.org/pub/png/

Vedoucí bakalářské práce: **Ing. Tomáš Dulík**
Ústav aplikované informatiky

Datum zadání bakalářské práce: **13. února 2007**

Termín odevzdání bakalářské práce: **24. května 2007**

Ve Zlíně dne 13. února 2007



prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem bakalářské práce bylo navrhnout a vyvinout efektivní multiplatformní GUI knihovnu v programovacím jazyce C++. Výsledná knihovna byla napsána pro operační systém Windows a GNU/Linux. Základní myšlenkou celého grafického rozhraní bylo použití alternativního způsobu vytváření a vykreslování jednotlivých grafických prvků, což zajišťuje velmi snadnou přenositelnost samotné knihovny na ostatní operační systémy a velmi dobrou odezvu pod X Window System.

Klíčová slova: Multiplatformní, GUI, knihovna, toolkit, C++, X Window System

ABSTRACT

The aim of this bachelor thesis was to design and develop efficient multiplatform GUI library in C/C++ language. Designed library was written for Windows and GNU/Linux operating systems. The base idea of the graphics environment was to use alternative way for creating and painting user components. This way provides very easy portability across operating systems and very good response under X Window System.

Keywords: Multiplatform, GUI, library, toolkit, C++, X Window System

Rád bych poděkoval Ing. T. Dulíkovi za odborné vedení při zpracování bakalářské práce. Děkuji také všem vývojářům, kteří zkoušeli vývojové verze knihovny a posílali připomínky nebo hlášení o chybách.

Prohlašuji, že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

V dnešní době se čím dál více setkáváme s požadavkem na vývoj přenositelných aplikací, které mají jen minimální závislost na konkrétním operačním systému. Pokud je aplikace napsána v interpretovaném jazyce, tvoří přenositelnost samotné jádro interpretu a pomocné knihovny. Aplikace psané v programovacím jazyce C/C++ bývají mnohem častěji označovány za nepřenositelné, protože knihovny operačních systémů se liší a nemůžeme se spolehnout na dodržení standardu jednotlivých překladačů jazyka C/C++ (ISO C99).

Cílem knihovny Wde je usnadnit přenášení aplikací mezi OS Windows a GNU/Linux na úrovni zdrojových kódů. Knihovna je rozdělena do platformní a uživatelské části. V platformní části jsou implementovány rozdíly jednotlivých operačních systémů. Uživatelská část obsahuje standardní grafické prvky a garantuje jednotný vzhled na všech podporovaných operačních systémech. Rozdělení knihovny umožňuje použít vyvinuté řešení i pro konzolovou aplikaci, která může těžit z velkého množství poskytovaných tříd.

Vlastní myšlenka navrhnout GUI toolkit se zrodila ze zkušeností v programování pro X Window System pod operačním systémem GNU/Linux. Komunikace mezi aplikací a X Window System je založena na architektuře klient-server a uskutečňuje se pomocí socketu. Tato architektura je efektivní pro síťovou komunikaci, ale pro desktopové aplikace znamená větší zatížení CPU. Při implementaci knihovny Wde byl kladen důraz právě na minimalizaci komunikace s okenním systémem.

Předmětem teoretické části je popsat některé rozdíly při multiplatformním vývoji aplikací a seznámit s nejpoužívanějšími knihovnami pro jejich vývoj v jazyce C/C++. Praktická část se věnuje dokumentaci vyvinutých knihoven včetně krátkých příkladů napsaných v jazyce C++.

TEORETICKÁ ČÁST

1 KNIHOVNA WDE

Wde toolkit je multiplatformní knihovna pro vývoj konzolových i okenních aplikací. Název vznikl spojením písmena „W“ a slov „Desktop Environment“ (desktopové prostředí). Samotná knihovna měla ze začátku tvořit základ pro Window Manager pod X Window System, ale stalo se z ní něco mnohem víc – multiplatformní toolkit pro vývoj okenních aplikací.

1.1 Použité nástroje

1.1.1 Programovací jazyk C++

C++ je staticky typovaný, objektově orientovaný jazyk, který je z velké části kompatibilní s jazykem C. Obohacuje jej o třídy a výjimky, ale kód je možné psát i procedurálně. Jazyk C/C++ si získal své zastánce v komerční i nekomerční sféře, píšou se v nich operační systémy i samotné aplikace.

Zájem o objektově orientované jazyky v programování okenních aplikací je vysoký, protože objekty a jejich dědičnost jsou velmi přirozené vlastnosti při navrhování uživatelských prvků, kde většinou používáme jako základ nějaký už hotový prvek a rozšiřujeme pouze některé z jeho vlastností.

1.1.2 Cmake – Cross platform make

Cmake[1] je nástroj pro generování projektových souborů pro různé platformy a IDE. Základem je jednoduchý platformě nezávislý konfigurační soubor `cmakelists.txt`, ve kterém jsou obsaženy podrobnosti ohledně projektu a pravidla pro generování projektových souborů. Tento velice jednoduchý nástroj může v budoucnosti nahradit existující Autotools.

Cmake je vhodný pro generování projektových souborů pro tyto IDE:

- KDevelop
- Microsoft Visual Studio
- MinGW

1.2 Modifikovaná licence BSD

BSD licence[8] je licence pro svobodný software, mezi kterými je jednou z nejsvobodnějších. Umožňuje volné šíření licencovaného obsahu, přičemž vyžaduje pouze uvedení autora a informace o licenci, spolu s upozorněním na zřeknutí se odpovědnosti za dílo.

Zkratka BSD označuje „Berkeley Software Distribution“ – obchodní organizaci při University of California, Berkeley, která tuto licenci vyvinula a používala pro práce nad operačním systémem BSD.

BSD licence dovoluje komerční využití, včetně využití v proprietárním software bez zveřejněného zdrojového kódu. Díla založená na dílech licencovaných pod BSD dokonce mohou být zveřejněna pod komerční licencí, pouze musí dodržet podmínky licence (v programu uvést informaci o autorech a zřeknutí se odpovědnosti).

2 ZÁKLADNÍ NÍZKOÚROVŇOVÉ ROZDÍLY MEZI OS WINDOWS A GNU/LINUX

Knihovna Wde vytváří abstrakční mezivrstvu a sjednocuje rozdíly podporovaných operačních systémů na programové i uživatelské úrovni.

2.1 Systém souborů

Operační systém Windows používá k označení disků písmena od A do Z. Písmena A a B jsou z historických důvodů rezervovány pro disketové mechaniky. Každá platná disková jednotka obsahuje adresář, který je pro danou jednotku a proces aktivní. V operačním systému GNU/Linux a na všech ostatních operačních systémech unixového typu jsou disky připojovány jako adresář (většinou v adresáři /mnt, ale není podmínkou) a každý proces má jen jeden aktivní adresář, který je sdílen i mezi vlákny.

Další drobný rozdíl mezi operačním systémem Windows a ostatními systémy unixového typu je způsob psaní adresářového separátoru. Ve Windows se adresář odděluje zpětným lomítkem '\', ale v unixových operačních systémech je to právě obyčejné lomítko '/'. Druhá zmíněná varianta bývá v literatuře označována jako POSIX.

V knihovně Wde se pracuje vždy s obyčejným oddělovačem '/', ale API si dokáže poradit i s druhou variantou pod operačním systémem Windows. Wde obsahuje také pokročilé normalizační funkce, které se ve standardní C knihovně nevyskytují.

2.2 Unikód a rozdíl v implementaci typu wchar_t

Unikód[10] je univerzální znaková sada spravována organizací Unicode Consorcium. Unikód je kompatibilní s normou ISO 10646, která definuje jednotlivé znaky, jejich názvy a pozice v unicode tabulce. Platný rozsah znaků je 31 bitů, což umožňuje definovat až 2^{31} znaků (v současnosti jich je definováno asi 2^{18}).

S univerzálním kódováním se můžeme setkat ve třech variantách:

- UTF-8 – 8 bitové kódování s použitím vícebytového zápisu pro znaky větší než 127. Toto kódování je binárně kompatibilní s první polovinou ASCII tabulky. Kódování UTF-8 se velmi často používá v operačním systému GNU/Linux a je výchozí v jazyce XML.

- UTF-16 – 16 bitové kódování. Pro znaky které se nevlezou do rozsahu 16 bitů se používá dvojice 16 bitových znaků (surrogate-pair). Toto kódování se používá v operačním systému Windows, v prostředí .NET, Java a Python.
- UTF-32 – 32 bitové kódování. Toto kódování je z hlediska paměťové náročnosti nejméně výhodné. Běžný anglický text zabere v paměti 4x více místa než při použití ASCII kódování. Pro vývoj multijazyčné aplikace je toto kódování ovšem nejlepší, protože při práci s řetězci je jeden znak vždy 32 bitové číslo a nemusí se ošetřovat situace variabilní délky, která nastává u kódování UTF-8 a UTF-16.

V operačním systému Windows se setkáme většinou s 8 bitovým kódováním (Win95+) nebo UTF-16 (WinNT, Win2000+). Programátor má možnost ovlivnit kódování při kompilaci zdrojových souborů definováním makra `UNICODE`, což způsobí změnu datového typu `TCHAR`. V operačním systému GNU/Linux se používá výhradně 8 bitové kódování, které zaručuje kompatibilitu při zpracování standardního vstupu a výstupu.

V jazyce C/C++ byl zaveden nový datový typ `wchar_t`, který by měl podle specifikace odpovídat vždy jednomu znaku v unikódu. V praxi závisí velikost tohoto datového typu na operačním systému a ne vždy je splněna specifikace.

Tab. 1. Velikost datového typu wchar_t v různých operačních systémech

	Windows	Linux, BSD	MAC OS X	Embedded Linux
Velikost wchar_t	16 bitů	32 bitů	32 bitů	16 / 32 bitů

Wde knihovna zavádí svůj vlastní datový typ pro znak v unikódu, který je vždy 32 bitový a má schopnost pojmout veškeré definované znaky. Objektový pocit vyvolává i fakt, že tento datový typ je třída, která používá metody k přístupu nebo testování znaku.

2.3 Standardní C knihovna vs. WinAPI

Operační systém Windows se liší od všech unixových operačních systémů tím, že používá vlastní API, kterým nahrazuje velkou část funkcí ze standardní C knihovny. WinAPI funkce nabízí mnohem větší možnosti než standardní funkce z knihovny C pod operačním systémem Windows, a pro většinu aplikací nebo knihoven je jejich použití výhodné

(například pod Windows je mnohem výhodnější použít funkci `GetFileAttributesEx()` než standardní POSIX funkci `stat()`).

Knihovna Wde tvoří abstrakční vrstvu nad nejčastěji používanými funkcemi, které jsou závislé na standardní knihovně C nebo WinAPI a poskytuje třídy, se kterými se pracuje stejně na všech podporovaných platformách. Důvod vzniku těchto abstrakčních vrstev je například systémově závislé kódování řetězců (u C funkcí i WinAPI) a rozdílné IO funkce.

2.4 X Window System vs. WinAPI

X Window System je multiplatformní okenní systém založen na architektuře klient-server. Komunikace mezi klientem a serverem probíhá pomocí socketu nebo sdílené paměti (XShm rozšíření). Tento systém byl vynalezen na MIT (Massachusetts Institute of Technology) už v roce 1984 a je šířen pod stejnojmennou licencí. V současnosti je spravován neziskovou organizací X Consorcium.

X Window System je základní okenní systém pro operační systémy unixového typu (Linux, (x)BSD, Solaris), ale bez problémů lze tento systém provozovat i v OS Windows pomocí projektu Cygwin nebo na Mac OS X. X architektura je velmi otevřená a podporuje různé rozšíření, které umožňují například využít hardwarovou akceleraci pomocí rozhraní OpenGL nebo využít komunikaci mezi klientem a serverem pomocí sdílené paměti. Rozšíření existuje více, ale již zmíněné mají největší přínos pro desktopové aplikace.

Operační systém Windows obsahuje vlastní API pro správu oken, které bylo vyvíjeno od samého počátku pro desktopové aplikace. Obsahuje i vyšší uživatelské prvky jako jsou tlačítka, editor textu a pokročilé komponenty zobrazující stromy a seznamy. Pokud bychom chtěli tyto prvky přenést pod X Window System, nezbývá než si udělat vlastní implementaci.

Rozdílné není jen API, ale i model událostí. X Window System generuje události, které předá do fronty a výsledek těchto událostí už nelze ovlivnit návratovou hodnotou (ze samé podstaty je to v pořádku, protože událost se již stala). WinAPI posílá zprávy (messages), kde výsledek lze ovlivnit návratovou hodnotou a systém usoudí, zda-li zprávu nepřeposlat dál. Zpráva se od události liší tím, že nemusí obsahovat notifikaci něčeho, co se již stalo, ale její výsledek (návratová hodnota) může ovlivnit to, co se teprve stane. Druhý zmíněný model poskytuje vyšší interaktivitu při komunikaci aplikace s uživatelem a snadnější

komunikaci mezi procesy (komunikace mezi procesy ve Windows je založena právě na zprávách).

Knihovna Wde implementuje vlastní jednodušší model událostí, který pro programování okenních aplikací plně postačuje. Událostní model umožňuje posílat uživatelským prvkům zprávy i ve formě třídy C++, které mohou vygenerovat událost (také C++ třída).

2.5 Souhrn

Tyto rozdíly zmíněných operačních systémů nejsou triviální a komplikují multiplatformní vývoj. Řešením je vždy použít knihovnu, která je multiplatformní a skrývá API závislé na konkrétním operačním systému. Pokud tato knihovna umí pracovat i s okny a poskytuje uživatelské prvky, můžeme ji nazvat multiplatformní GUI toolkit.

Wde knihovna vytváří multiplatformní přístup ke všem zmíněným rozdílným vlastnostem operačních a okenních systémů. Při vývoji byl kladen důraz na jednoduchost použití výsledných tříd a možnou rozšiřitelnost kódu i pro ostatní platformy. Knihovna poskytuje mnohem více možností, než které byly rozebrány a mohou přijít případnému vývojáři velmi užitečné.

3 MULTIPLATFORMNÍ GUI TOOLKITY

3.1 Qt Toolkit

Qt[7] je komerční GUI Toolkit vyvíjený firmou Trolltech (www.trolltech.com) z Norska. Jde o multiplatformní, objektově orientovaný framework napsaný v jazyce C++. Zdrojové kódy jsou pod licencí GPL a QPL pro nekomerční účely. Komerční využití je placené a účtuje se počet vývojářů. Qt je jeden z nejstarších toolkitů, který měl stabilní verzi již v roce 1995 a zároveň patří mezi ty nejúspěšnější. Webový prohlížeč Opera, Google Earth nebo použití pro embedded zařízení demonstrují jeho sílu.

Zdrojové kódy Qt jsou napsány velmi kvalitně a obsahují podrobnou dokumentaci. Aplikace jsou snadno lokalizovatelné a přenositelné díky nástrojům, které firma Trolltech dodává k toolkitu (QtLinguist, QtDesigner). Na druhou stranu, penalizace za tento komfort je určitě rychlost a velikost samotné knihovny (Qt4 více než 12MB). Často vyčítanou nutností je MOC (Meta Object Compiler). Tento nástroj předkompiluje třídy odvozené od základní třídy `QObject` a vytvoří některé povinné virtuální metody, které jsou základem pro dynamické vlastnosti tříd v Qt, signal/slot implementaci a dynamické properties.

Qt obsahuje třídy a šablony známé ze standardní C++ knihovny STL (Standard Template Library), které obohacuje o zajímavé programovací techniky. Nejedná se tedy jen o GUI knihovnu, ale rozsáhlý framework, kde nechybí ani síťová vrstva, šifrování nebo vestavěná podpora pro skriptovací jazyk kompatibilní se specifikací ECMAScript 3 (základ pro Javascript 1.5).

Na Qt toolkitu je založeno známe unixové desktopové prostředí KDE.

Tab. 2. Qt Toolkit - přehled

Jméno knihovny a www odkaz	Trolltech Qt (http://www.trolltech.com)
Hlavní programovací jazyk	C++, nutnost použít nástroj MOC a UIC, vestavěná podpora pro skriptování QtScript
Ostatní programovací jazyky	Java, C#, Python, Perl, Javascript, Ruby a další ...
Licence	GPL / QPL pro nekomerční účely

Přenositelnost	Windows, GNU/Linux, Mac OS X, (x)BSD a Embedded Linux (Qtopia)
Nativní vzhled	Windows, Mac OS X, X Window System
Podpora pro změnu vzhledu	Ano
Podpora pro rozšíření widgetů	C++ dědičnost, šablony, zásuvné moduly
Výchozí kódování řetězců	UTF-16
Implicitní sdílení dat a copy-on-write	Ano
Poznámky	Integrovaná lokalizace a pokročilé uživatelské prvky

3.2 Gtk+

Gtk+[4] toolkit vznikl původně pro grafickou aplikaci GIMP (GNU Image Manipulation Program) a dříve byl dodáván jako její součást. Dnes je Gtk+ zcela samostatný soubor C knihoven na jehož vývoji se významným měřítkem podílela firma RedHat.

Základ pro knihovnu Gtk+ je multiplatformní knihovna Glib, která sjednocuje API operačních systémů a implementuje práci s různými algoritmy (binární stromy, hash tabulky a seznamy). Knihovna Glib je dodávána i vyvíjena samostatně a používá ji i mnoho konzolových aplikací pro unixové operační systémy (knihovnu Glib obsahuje snad každá linuxová distribuce).

Na Gtk+ toolkitu je založeno známe unixové desktopové prostředí GNOME.

Tab. 3. Gtk Toolkit - přehled

Jméno knihovny a www odkaz	The GiMP Toolkit (http://www.gtk.org)
Hlavní programovací jazyk	C

Ostatní programovací jazyky	C++, D, Java, C#, Python, Perl, Ruby, Scheme a další ...
Licence	LGPL
Přenositelnost	Windows, GNU/Linux, Mac OS X, (x)BSD a Linux framebuffer
Nativní vzhled	X Window System
Podpora pro změnu vzhledu	Ano
Podpora pro rozšíření widgetů	GtkObject, zásuvné moduly
Výchozí kódování řetězců	UTF-8
Implicitní sdílení dat a copy-on-write	Ne
Poznámky	Obsahuje objektově orientovanou vrstvu implementovanou v jazyce C (Glib), odlišné chování ve Windows

3.3 wxWidgets

Toolkit wxWidgets[9] vznikl v roce 1992 na univerzitě v Edinburgu. Motivace k vývoji tohoto toolkitu bylo umožnit přenositelnost aplikací mezi operačním systémem Windows a UNIX. Pomocí toolkitu wxWidgets je nyní možné vyvíjet i aplikace pro GNU/Linux, Mac OS X a embedded zařízení.

Jedna z upřednostňovaných výhod toolkitu wxWidgets je využití systémových GUI prvků. Většina widgetů obsahuje platformě závislou implementaci, která zaručuje nativní vzhled a chování aplikace na všech podporovaných platformách. Ke knihovně existuje spousta komerčních i nekomerčních nástrojů, které umožňují vytvářet okenní aplikace pouhým přesouváním widgetů – RAD.

Programování pod wxWidgets bývá velice často přirovnáváno k programování pod MFC, protože programovací model je velmi podobný (mapa zpráv ve třídě, makra pro dynamickou identifikaci, ...).

Tab. 4. wxWidgets - přehled

Jméno knihovny a www odkaz	wxWidgets (http://www.wxwidgets.org)
Hlavní programovací jazyk	C++
Ostatní programovací jazyky	Python (wxPython) – silná podpora, Perl, C# (wx.NET)
Licence	LGPL
Přenositelnost	Windows, GNU/Linux, Mac OS X, (x)BSD
Nativní vzhled	Windows, Mac OS X, Gtk, X Window System
Podpora pro změnu vzhledu	Ne
Podpora pro rozšíření widgetů	C++ dědičnost
Výchozí kódování řetězců	ANSI, UTF-8 nebo unicode (podle nastavení)
Implicitní sdílení dat a copy-on-write	Ano
Poznámky	API a hierarchie tříd se podobá programování pod MFC

3.4 Fltk

Fltk[2] je objektově orientovaný toolkit napsaný v jazyce C++. Tento toolkit je rychlý a nenáročný na systémové zdroje. Je navržen pro statické i dynamické linkování a spustitelný soubor aplikace typu „hello world“ má v binární podobě kolem 250kB. Fltk obsahuje i jednoduchý návrhář uživatelského rozhraní Fluid, který z navrženého rozhraní generuje C++ kód. Další příjemnou součástí je multiplatformní podpora pro OpenGL.

Knihovna Fltk je pouze GUI toolkit a neposkytuje žádné jiné vymoženosti jako ostatní zmíněné knihovny. Pro řetězce se používá ukazatel `char*`, jako callback se používá obyčejná C funkce (většinou statická metoda třídy) a model událostí neumožňuje zařadit událost do fronty. I přes všechny tyto nedostatky jde o zajímavé řešení, na kterém je postavená celá řada aplikací a rozšiřujících knihoven.

Fltk toolkit je vhodný pro jednodušší aplikace, kde velmi záleží na velikosti spustitelného souboru a spotřebě paměti.

Tab. 5. Fltk - přehled

Jméno knihovny a www odkaz	Fast Light Toolkit (http://www.fltk.org)
Hlavní programovací jazyk	C++
Ostatní programovací jazyky	Python
Licence	LGPL s výjimkou, která umožňuje statické linkování
Přenositelnost	Windows, GNU/Linux, Mac OS X, (x)BSD
Nativní vzhled	Ne, výchozí vzhled napodobuje Win95
Podpora pro změnu vzhledu	Jen ve verzi 2.x
Podpora pro rozšíření widgetů	C++ dědičnost
Výchozí kódování řetězců	UTF-8 (starší verze ANSI)
Implicitní sdílení dat a copy-on-write	Ne
Poznámky	Velmi nízká spotřeba paměti, k dispozici rozšířené verze EFltk a Sptk

3.5 Souhrn

Zmíněné GUI toolkity jsou dlouho ve vývoji a mají velmi dobrou uživatelskou základnu. Komerční podporu poskytuje pouze firma Trolltech k toolkitu Qt. Ostatní toolkity jsou na druhou stranu zdarma i pro komerční využití díky licenci LGPL.

Gtk+ spolu s Qt jsou dominantní toolkity pro unixové operační systémy. Qt si vede velmi dobře i na všech ostatních podporovaných platformách. Gtk+ má problémy s uživatelskou vrstvou zejména pod operačním systémem Windows, kde se chování tohoto toolkitu výrazně liší od návyků z ostatních Windows aplikací. Toolkit wxWidgets patří mezi knihovny, které nejsou binárně tak veliké jako Gtk+ nebo Qt. Nejvíce je ovšem rozšířený ve světě Windows, kde knihovna wxWidgets nabízí velmi zajímavou alternativu k MFC.

Programátor, který potřebuje vyvinout multiplatformní aplikaci má v dnešní době velmi široké možnosti. Pro ty nejjednodušší aplikace lze použít i javascript, který běží ve webovém prohlížeči, ale pro složitější aplikace se zmíněné řešení zrovna nehodí. Existují rozhodovací faktory, které ovlivní výběr toolkitu pro konkrétní účel.

3.5.1 Základní rozhodovací faktory při výběru toolkitu

- programovací jazyk a API
- licence – při komerčním vývoji nejlépe LGPL nebo BSD
- přenositelnost mezi operačními systémy
- množství základních i přídavných nástrojů / komponent
- stabilita a podpora
- vzhled – nativní vzhled, skinování aplikace
- binární velikost knihovny a počet závislostí na ostatních knihovnách (nebo interpretu dynamického jazyka)
- rychlost (běhu i spuštění) i paměťové nároky

Pro rozhodování mnohdy stačí jen znalost programovacího jazyka a znalost konkrétního toolkitu. Zkušený Qt programátor jen velmi zřídka použije pro svou aplikaci třeba Gtk+.

3.6 Motivace k vývoji Wde toolkitu

Knihovna Wde vznikla původně pro operační systém GNU/Linux. Výchozí okenní systém pro tento operační systém je založen na architektuře klient-server, která není až tak vhodná pro desktopové řešení (X Window System). Komunikace mezi tímto okenním systémem a aplikací je řešena prostřednictvím socketu a každý požadavek má mnohonásobně vyšší režie než u jiných desktopových okenních systémů (například WinAPI).

3.6.1 Top-level okna

Hlavní myšlenkou při vývoji knihovny Wde bylo maximálně omezit komunikaci s okenním systémem. Knihovna používá okenní systém jen k vytvoření top-level oken. K vytvoření dětských oken používá toolkit vlastní implementaci okenního systému, který není absolutně závislý na X Window System nebo WinAPI. Tento způsob umožnil vysokou optimalizaci překreslovacího procesu a maximálně zamezil komunikaci s okenním systémem.

3.6.2 Překreslování widgetů

Efektivní překreslování widgetů bylo jednou z priorit při tvorbě knihovny Wde. Byl implementován vnitřní proces, který se nazývá „update“ a zajišťuje překreslení všech částí top-level okna, které to potřebují. Při změně obsahu kteréhokoliv widgetu se dotyčný widget nejprve označí (bitové proměnné) a v update procesu se překreslí. Tento systém zajišťuje, že do žádného widgetu nebude vykreslen vícekrát stejný obsah. Proces update se pravidelně volá z hlavního cyklu aplikace.

K překreslování widgetů byl implementován vlastní rastrový kreslící subsystém. Wde knihovna je schopná vykreslit vyhlazený text i barevné přechody, které umožňují, aby aplikace vypadala opravdu moderně. Ke změnění celkového vzhledu aplikace stačí pouze napsat nové téma, ve kterém budou implementovány postupy vykreslení všech prvků. Implementace nového tématu je založena na C++ dědičnosti.

3.6.3 Výhody Wde toolkitu

- objektově orientovaná knihovna napsaná v jazyce C++
- možnost volat nízkoúrovňové API z jazyka C

- BSD licence
- rozdělení na abstrakční (WdeCore) a grafickou (WdeGui) vrstvu
- malá velikost v binární podobě a minimální závislosti na ostatních knihovnách
- řetězce používají 32 bitový unikód
- minimální komunikace s okenním systémem (rychlost)
- platformě nezávislé uživatelské prvky s možností skinování (témata)
- zvýšený důraz na efektivní překreslování oken a odstranění „flicker“ efektů

3.6.4 Nevýhody Wde toolkitu

- nová a nevyzkoušená knihovna (může obsahovat chyby)
- v současnosti malý počet uživatelských prvků i pomocných tříd (ve srovnání s Qt, Gtk+ nebo wxWidgets)

PRAKTICKÁ ČÁST

4 DOKUMENTACE KNIHOVNY WDECORE

WdeCore je abstrakční systémová knihovna. Snaží se sjednotit odlišnosti operačních systémů na úrovni API a poskytuje soubor tříd pro efektivní multiplatformní vývoj. Návrh knihovny WdeCore umožňuje i experimentální použití mnoha částí knihovny v čistém jazyce C, ovšem doporučené a očekávané je používání WdeCore knihovny v jazyce C++.

4.1 Wde::Object - Dynamické vlastnosti objektů

Knihovna WdeCore nevyžaduje pro svůj běh RTTI ani výjimky. Dynamické funkce objektů zajišťuje třída `Wde::Object`. Při odvození z třídy `Wde::Object`, se používají makra `WDE_OBJECT(class, inherit_class)` a `WDE_OBJECT_INIT(class, string)`.

Příklad odvození třídy z `Wde::Object`:

```
// deklarace třídy, která dědí z Wde_Object
struct MyObject : public Wde::Object
{
  // Použití makra pro definici typu třídy a typu předka.
  // Každá takto definovaná třída obsahuje typ 'base', což
  // je v našem případě předek 'Wde::Object'
  WDE_OBJECT(MyObject, Wde::Object)

  // Konstruktor a destruktor
  MyObject() {}
  virtual ~MyObject() {}
};
// Inicializace dynamických vlastností objektu, první parametr
// je typ třídy, pro který inicializujeme vlastnosti, druhý
// parametr je typ, pod kterým tuto třídu zaregistrujeme (řetězec)
WDE_OBJECT_INIT(MyObject, "MyObject")

// Vstupní bod programu
int main(int argc, char* argv[])
{
  // Vytvoření instance třídy MyObject
  MyObject myObject;
  // Dynamická identifikace
  if (myObject.isClassOf("MyObject")) {} // true

  return Wde::Exit_Success;
```

```
}
```

4.2 Wde::ByteArray a Wde::String - Bytové pole a řetězce

Řetězce patří mezi velmi důležitý datový typ. Ve standardní C knihovně se používají 8 bitové nulou ukončené řetězce (ve skutečnosti je řetězec pouze ukazatel na pole typu `char` nebo `wchar_t`). WdeCore knihovna rozděluje řetězce na bytové pole (`Wde::ByteArray`) a řetězce v unikódu (`Wde::String`). Obě zmíněné třídy poskytují podobné metody a používají implicitní sdílení dat.

Pro maximální využití rychlosti CPU a potenciálu dočasných řetězců byly vyvinuty šablony (templates), které uloží data třídy na zásobníku a vyhnou se dynamické alokaci paměti. Pro větší bezpečnost jsou tyto šablony schopné i dynamické alokace v případě překročení stanoveného limitu při práci s daty nebo nové alokace v případě zneplatnění odkazu na jejich obsah (způsobeno návratem z funkce).

Příklad použití bytového pole a řetězce:

```
Wde::ByteArray b1;  // vytvoření bytového pole 1 (8 bitové data)
Wde::ByteArray b2;  // vytvoření bytového pole 2

Wde::String s1;     // vytvoření unicode řetězce 1 (32 bitové data)
Wde::String s2;     // vytvoření unicode řetězce 2

b1 = "ByteArray 1"; // nastavení obsahu na "ByteArray 1"
b2 = "ByteArray 2"; // nastavení obsahu na "ByteArray 2"

s1 =  "String 1";   // konverze z ASIII řetězce "String 1"
s2 = L"String 2";   // konverze z wchar_t řetězce "String 2"

// kopírování řetězců nebo bytových polí nevytváří hluboké kopie
b1 = b2;
s1 = s2;
// ale změna obsahu ano
b1 += "New Appended Content";
s1 += "New Appended Content";
```

Příklad použití řetězce jako šablony:

```
void Wde::String fn(void)
{
  // vytvoření dočasného řetězce o kapacitě 128 znaků
```

```
  Wde::StringT<128> s;

  // příklad operace s řetězcem
  s = "${0} Text ${1}";
  s.replace("${0}", "Prefix");
  s.replace("${1}", "Suffix");

  // návrat 's' způsobí realokaci na přesnou velikost, protože jsou
data
  // řetězce uložené na zásobníku a návratem dojde k zneplatnění
adresy
  // (zneplatnění chápeme tak, že se nemůžeme spolehnout na
bezpečnost dat
  // na dané adrese, protože mohou být přepsána při příštím volání
  // jakékoliv funkce)
  return s;
}
```

Řetězce a bytové pole obsahují přes 100 metod pro manipulaci s jejich daty. Mezi standardní metody `append()`, `prepend()`, `insert()`, `remove()` a `replace()` patří i vyhledávací `indexOf()`, `lastIndexOf()`, `startsWith()` a `endsWith()`. Konverzi z číselných typů zajišťují metody `setNum()`, `appendNum()` a `to[I|U][32|64]()`. Implementované jsou i další užitečné metody pro odstraňování bílých znaků `trim()` a `simplify()`.

4.3 Wde::TextCodec – konverze textu z / do unicode

Konverze textu se provádí pomocí třídy `Wde::TextCodec`. Třída je navržena tak, aby byla schopná dekódovat nebo enkódovat text z / do 32 bitového unicode. Obsahuje informace o kódování, popřípadě i konverzní tabulky (pro 8 bitové kodeky). Pro nastavení textového kodeku se používají metody `setFromType()`, `setFromMime()` a `setFromBom()`. Pro samotnou konverzi se používají metody `appendFromUnicode()` a `appendToUnicode()`. Cílový objekt je vždy typu `Wde::ByteArray` nebo `Wde::String` a zaručuje automatické zvětšování paměťového bufferu při konverzi mezi dlouhým textem.

Příklad konverze textu:

```
Wde::TextCodec codec; // Vytvoření instance textového kodeku
Wde::String s1;       // Vytvoření instance prázdného řetězce

// Řetězec v kódování CP1250
static const uint8_t text[] =
```

```
{
  'T', 'e', 'x', 't', ' ', 'v', ' ',
  0xE8, 'e', 0x74, 't', 'i', 'n', 0x2E, 0
};

// Nastaví textový kodek pro kódování CP1250
if (codec.setFromMime("CP1250").ok())
{
  // Dekóduje text z kódování CP1250. Wde::DetectLength znamená
  // že řetězec je ukončený nulou.
  codec.appendToUnicode(s1, text, Wde::DetectLength);
  // s1 == "Text v češtině"

  // alternativní a rychlejší zápis by byl:
  codec.appendToUnicode(s1, text, sizeof(text) - sizeof(uint8_t));
  // s1 == "Text v češtiněText v češtině"
}
else
{
  // chyba: v případě, že požadovaný kodek není podporován
}
```

Z příkladu je vidět, že `Wde::TextCodec` nečistí cílový objekt, jen přidává text na konec (append).

4.4 Wde::Value – variabilní typ

`Wde::Value` je implementace variabilního datového typu v jazyce C++. Knihovna WdeCore používá tento typ u metod, které mohou skončit chybou a vrací chybový kód. Variabilita tohoto typu umožňuje vrátit kód chyby v podobě řetězce. Třída `Wde::Value` není vytvořena jen pro návrat chybových stavů a v kombinaci se seznamem nebo hash tabulkou můžeme vytvořit silnou kombinaci známou s vysokoúrovňových jazyků.

třída `Wde::Value` je schopna pojmout následující datové typy:

- Null
- 32 bitový integer
- 64 bitový integer
- double

- unicode řetězec
- chybový stav (dvě 32 bitové čísla)

Příklad pro nastavení a testování variabilních typů:

```
Wde::Value a; // vytvoření instance Wde::Value, a.isNull() == true
Wde::Value b; // vytvoření instance Wde::Value, b.isNull() == true
Wde::Value b; // vytvoření instance Wde::Value, c.isNull() == true

a = 10;       // nastavení 'a' na typ 'Wde::Value::Type_Int32'
b = 10.5;     // nastavení 'b' na typ 'Wde::Value::Type_Double'
c = "10";     // nastavení 'b' na typ 'Wde::Value::Type_String'

// porovnání hodnot je možné i přes rozdílný typ
if (a == c) {} // true
if (a < b) {} // true
if (a.type() == c.type()) {} // false

// kopírování hodnoty vytváří implicitní kopie
a = b;
if (a == b) {} // true
if (a.type() == b.type()) // true

// nastavení na Null
a.setNull();

// konverze na jiný typ
a.getString(); // "(Null)"
b.getString(); // "10.5"
c.getInt32();  // (int32_t)10
```

Z příkladu je vidět, že datový typ je opravdu univerzální a zvládá i konverzi do jiných typů. Přiřazovací a testovací operátory jsou přetíženy a je tedy možné psát kód tak jak jsme zvyklí z dynamicky typovaných programovacích jazyků.

Chybový stav je možné nastavit metodami `setError()`, `setError_errno()` a `setError_WinLastError()`. Metoda `ok()` testuje, zda není typ nastaven na chybovou hodnotu a výsledná hodnota je rovna výrazu `type() == Wde::Value::Type_Error`. Typ Null není považován za chybovou hodnotu.

4.5 Wde::Hash – Hash tabulka

Hash tabulka je asociativní pole, kde se přiřazuje vždy jedna hodnota pro daný klíč (vzniká pár klíč + hodnota). Implementace využívá šablony v C++, což umožňuje, že typ klíče a jeho hodnota mohou být libovolné. Podporované jsou jednoduché datové typy i komplexní třídy. Hashovací funkce se vytváří přetížením `uint32_t Wde_hash(T& t)`.

Hash tabulka je specializovaná pro některé typy. Při použití klíče `Wde::String` garantuje, že hash 8 bitového řetězce v ISO 8859-1 kódování vrátí stejnou hodnotu jako hash 32 bitového řetězce v unicode kódování ISO 10646.

Hash tabulka implementuje metody `put()` a `set()` pro vložení klíče a nastavení jeho hodnoty. Odstranit položku je možné metodou `remove()`. Přetížení operátoru [] umožňuje zapsat přiřazení i vrácení hodnot jako při práci s polem.

Příklad vkládání, čtení a odstranění prvků z hash tabulky, kde datový typ klíče je `Wde::String` a datový typ hodnoty je 32 bitové číslo:

```
// vytvoření instance hash tabulky
Wde::Hash<Wde_String, uint32_t> hash;

// vložení páru hodnot do hash tabulky
hash["A"]->value() = 1;
hash["B"]->value() = 2;

// výpis
Wde_debug("%d", hash["A"]->value());
Wde_debug("%d", hash["B"]->value());

// odstranění prvků
hash.remove("A");
hash.remove("B");
```

4.6 Wde::Thread a Wde::Mutex – vlákna a mutexy

WdeCore knihovna je vytvořena pro vícevláknové aplikace a je kompilována vždy s podporou vláken. Pro vytváření vlákna se používá jako základ třída Wde::Thread, ve které je třeba implementovat pure-virtual metodu `void Wde::Thread::_entry(void)`. Existuje také dobrovolná metoda `void Wde::Thread::_cleanup(void)`, která je zavolána při ukončení běhu vlákna.

Pro změnu stavu vlákna slouží metody `run()`, `terminate()` a `wait()`. Současný stav lze zjistit metodou `status()` a může nabývat těchto hodnot:

- `Wde::Thread::Status_NotRunning` – vlákno neběží a nejsou alokovány systémové zdroje
- `Wde::Thread::Status_Running` – vlákno běží
- `Wde::Thread::Status_Suspended` – vlákno bylo suspendováno
- `Wde::Thread::Status_Zombie` – vlákno je na konci svého běhu, zatím nebyly uvolněny systémové zdroje

Příklad vytvoření vlákna:

```
// Vytvoření třídy, která dědí Wde::Thread
struct MyThread : public Wde::Thread
{
  // Povinná virtuální metoda, která představuje vstupní bod vlákna
  virtual void _entry(void)
  {
    Wde_debug("Entry in thread %p\n", Wde::Thread::instance());
  }
}

int main(int argc, char* argv)
{
  // vytvoří 10 instancí třídy MyThread
  MyThread thread[10];

  // spustí všech 10 instancí
  for (ulong i = 0; i != WDE_TABLE_SIZE(thread); i++)
  {
    thread[i].run();
  }

  // při návratu z funkce počká na ukončení všech vláken
  // (stejné jako zavolat metodu wait() pro každé vlákno)
  return 0;
}
```

Třída Wde::Thread obsahuje i metody pro vrácení instance současného nebo hlavního vlákna. Jsou to tyto metody:

- `Wde::Thread* Wde::Thread::main()` – vrátí instanci hlavního vlákna
- `Wde::Thrrad* Wde::Thread::instance()` – vrátí instanci současného vlákna

Pro konkurenční přístup ke sdíleným datům mezi vlákny existuje třída `Wde::Mutex` a `Wde::RWMutex`. Tyto synchronizační třídy mají implementované metody pro zamykání a odemykání. Výchozí implementace je rekurzivní a umožňuje vícenásobné zamykání ve stejném vlákně.

Třída `Wde::Mutex` implementuje metody `lock()`, `unlock()` a `tryLock()`. Třída `Wde::RWMutex` implementuje metody `readLock()`, `readUnlock()`, `tryReadLock()`, `writeLock()`, `writeUnlock()` a `tryWriteLock()`.

4.7 Wde::Library – run-time načítání dynamických knihoven

Třída `Wde::Library` se používá k multiplatformnímu dynamickému načítání knihoven. V Linuxu začínají dynamické knihovny většinou prefixem „lib" a mají příponu „so". Ve Windows nemají sdílené knihovny žádný doporučený prefix a většinou končí příponou „dll". Pro zjednodušení přenositelnosti kódu je možné specifikovat při otevírání dynamické knihovny, zda použít platformě závislý prefix, suffix nebo obojí.

Pro otevření a zavření knihovny poskytuje třída `Wde::Library` metody `open()` a `close()`. Pro vrácení adresy symbolu je třeba zavolat metodu `symbol()`.

Krátký příklad otevření knihovny „mylib" a načtení symbolu „fn":

```
// instance třídy Wde::Libary
Wde::Library lib;

// otevření knihovny "mylib"
if (lib.open("mylib").ok())
{
  // deklarace symbolu Fn a jeho instance fn
  typedef void (*Fn)(void);
  Fn fn;

  // načtení symbolu "fn"
  if ((fn = (Fn)lib.symbol("fn")) != NULL)
  {
    // v případě úspěchu zavolání načtené funkce
    fn();
```

```
    }
  }
```

4.8 Wde::File, Wde::Directory a Wde::Path – filesystém

`Wde::File` a `Wde::Path` jsou statické třídy. Třída `Wde::File` se používá k práci se soubory, umožňuje zjistit typ souboru nebo přístupová práva pomocí metod `test()`, `isFile()` a `isDirectory()`. `Wde::Path` je třída pro testování, vytváření a normalizaci řetězců, které obsahují cestu k souboru (nedotýká se fyzicky souborového systému).

Třída `Wde::Directory` slouží k iterování mezi adresáři a soubory nacházející se na zadané cestě. Při každé iteraci vrací ukazatel na třídu `Wde::DirectoryEntry`, která obsahuje podrobné informace ohledně současné položky (soubor, adresář, symbolický odkaz, ...).

Příklad průchodu domácího adresáře uživatele:

```
Wde::Directory dir;
// pomocí metody open() se otevře adresář, vrátí hodnotu Wde::Value
if (dir.open(Wde::UserInfo::homeDirectory()).ok())
{
  const Wde::DirectoryEntry* entry;
  // klasická iterační smyčka, next() vrátí NULL až po projití
  // poslední položky. Adresáře '.' a '..' jsou filtrovány.
  while ((entry = dir.next()) != NULL)
  {
    // 'entry' nyní obsahuje kompletní informace o současné položce:
    // 'entry->name()' - jméno položky
    // 'entry->type()' - typ
    // 'entry->size()' - velikost, pokud se jedná o soubor
  }
}
```

4.9 Wde::Stream – datový proud

Knihovna WdeCore implementuje vlastní datové proudy. Jde pouze o sjednocení do jednoho rozhraní, které může být použito v libovolné části knihovny. Třída Wde::Stream umožňuje používat soubory větší než 2^{32} bitů (4GB).

Datový proud používá systémově závislé prostředky a umožňuje otevřít tyto typy:

- `HANDLE` – WinAPI file descriptor
- `FILE*` – struktura standardní knihovny C (získaná většinou zavoláním `fopen()`)

- unixový file descriptor (`int`) – prostředek standardní C knihovny na unixových OS
- paměťový rozsah (`void*`) – ukazatel na určitou adresu paměti a její velikost

Pro otevření datového proudu jsou implementovány metody `openFile()`, `openFd()`, `openHANDLE()`, `openFILE()` a `openMemory()`. Pro pohyb poskytuje třída metody `seek()` a `tell()`. Čtení a zápis zajišťují metody `read()`, `readAll()` a `write()`. Metoda `readAll()` se používá k načtení celého obsahu souboru do paměti, pokud je takové množství paměti k dispozici.

4.10 Wde::XmlDocument a Wde::XmlNode – podpora Xml

Knihovna WdeCore obsahuje vestavěnou podporu pro značkovací jazyk Xml. Na Xml data se můžeme dívat jako na dokumenty se stromovou strukturou. Dokument reprezentuje třída `Wde::XmlDocument` a jednotlivé větve `Wde::XmlNode`. Třída `Wde::XmlNode` je odvozena od třídy `Wde::Node`, ve které jsou implementovány obecné algoritmy pro práci se stromovou strukturou dat.

Pomocí třídy `Wde::XmlDocument` lze načíst data ze vstupu nebo uložit data na výstup. Pro vstup může sloužit třída `Wde::Stream` nebo `Wde::String`. Načíst Xml strukturu je možné i z paměti. Pro načítání a ukládání lze použít metody s prefixem `read` a `write`. Metody `constRoot()` a `mutableRoot()` slouží k vrácení kořenového prvku Xml dokumentu.

Třída `Wde::XmlNode` reprezentuje jeden element (značku), která má svůj název (tag) a obsah (content). Pro navigaci mezi stromovou strukturou lze použít metody `next()`, `prev()`, `parent()`, `children()` a `last()`. Metody `nextSibling()` a `prevSibling()` slouží k navigaci mezi elementy se shodným tagem. Pokud kterákoliv metoda pro navigaci vrátí neplatný ukazatel NULL, požadovaný element neexistuje.

Pro práci s Xml atributy se používají metody `hasAttributes()`, `attributes()`, `insertAttribute()`, `removeAttribute()`, `findAttribute()` a `useAttribute()`.

Každý Xml element obsahuje také 32 bitový hash, který odpovídá jeho tagu. Hledání mezi elementy tedy probíhá tak, že se porovnává nejprve hash, poté tag.

Příklad načtení Xml struktury a hledání elementů s tagem „Element”:

```
// statické Xml data
static const char xmlData[] =
  "<?xml version='1.0' ?>\n"
```

```
  "<root>\n"
    "<element attribute='content'>1</element>"
    "<element attribute='content'>2</element>"
    "<element attribute='content'>3</element>"
    "<element attribute='content'>4</element>"
    "<element attribute='content'>5</element>"
  "</root>\n";

// vytvoření instance Xml dokumentu
Wde::XmlDocument document;
// načtení dokumentu z paměti
if (document.readMemory(xmlData, sizeof(xmlData)).ok())
{
  // kořenový element lze získat metodou constRoot() nebo
  // mutableRoot()
  Wde::XmlNode* node = document.mutableRoot();
  // nyní máme kořenový element, který není nikdy NULL. V našem
  // případě reprezentuje element 'root' z 'xmlData'.

  // průchod mezi elementy 'element'
  for (node = node->findFirst("element"); node;
       node = node->nextSibling())
  {
    // node->tag() - "element"
    // node->content() - "1" "2" "3" "4" "5"
    // node->attributes() - atributy
  }
}
```

4.11 Wde::Memory – alokace paměti

Statická třída `Wde::Memory` obsahuje funkce pro dynamickou alokaci a uvolnění paměti. Tyto funkce jsou totožné ze standardními funkcemi z C knihovny, ale při zapnutí interního memory-debuggeru je schopen toolkit při korektním ukončení aplikace odhalit neuvolněnou paměť a vypsat podrobné informace. Tyto funkce nelze kombinovat s funkcemi z C knihovny.

Pro alokaci paměti poskytuje třída `Wde::Memory` metody `alloc()`, `realloc()`, `calloc()`, `xalloc()`, `xrealloc()` a `xcalloc()`. Pro uvolnění paměti se používá metoda `free()`. Suffix „x“ znamená, že při selhání alokace se vypíše hláška na standardní chybový

výstup a aplikace se ukončí. Tyto metody se používají na místech, kde selhání alokace znamená kompletní selhání aplikace nebo knihovny.

4.12 Ostatní pomocné třídy pro multiplatformní vývoj

Knihovna WdeCore obsahuje mnohem více tříd než bylo popsáno, ale zbývající třídy jsou už jen pomocné a nevytváří pevný základ samotné knihovny. To že jsou jen pomocné ovšem neznamená, že nejsou užitečné.

4.12.1 Wde::CpuInfo

`Wde::CpuInfo` obsahuje jen metodu `uint32_t Wde::CpuInfo::features()`, která vrací dostupné rozšíření procesoru (MMX, 3dNow!, SSE, SSE2, ...). Ke zjištění dostupných rozšíření se na architektuře x86 používá instrukce `CPUID`.

4.12.2 Wde::Swap

`Wde::Swap` je třída jen se statickými metodami, která se snaží využít platformě závislých prostředků k co nejrychlejšímu vyměnění bytů v 16, 32 nebo 64 bitovém datovém typu. Třída `Wde::Swap` obsahuje metody `i16()`, `u16()`, `i32()`, `u32()`, `i64()` a `u64()`. Každá metoda reprezentuje počátečním písmenem zda se jedná o typ znaménkový či neznaménkový. Číslo 16, 32 nebo 64 udává počet bytů vstupní a výstupní hodnoty. Případný suffix „le“ a „be“ říká v jakém bytovém pořadí požadujeme cílovou hodnotu a může nahradit funkce `htonl()`, `htons()`, `ntohl()` a `ntohs()` ze standardní C knihovny.

Příklad použití a výsledky:

- `Wde::Swap::u16(0x0102U) == 0x0201U`
- `Wde::Swap::u32(0x01020304U) == 0x04030201U`
- `Wde::Swap::u64(0x0102030405060708ULL) == 0x0807060504030201ULL`

4.12.3 Wde::UserInfo

`Wde::UserInfo` je třída pro zjištění informací o vlastníkovy běžícího procesu. V době psaní této práce obsahuje pouze metodu `homeDirectory()`, která vrátí domovský adresář uživatele. V operačním systémem GNU/Linux lze zavolat ještě metody `userId()` a `groupId()` pro zjištění identifikačního čísla uživatele a skupiny.

4.12.4 Wde::Environment

Statická třída `Wde::Environment` slouží k zjištění proměnných hodnot prostředí. Metoda `get()` vrací hodnotu pro určitou proměnnou v unicode řetězci. K vrácení hodnoty se uvnitř knihovny používá standardní C funkce `getenv()`.

4.12.5 Wde::System

Statická třída `Wde::System` se používá k zjištění informací ohledně operačního systému, ve kterém je aplikace právě spuštěna. Metody jsou platformě závislé a programátor by se měl ujistit, že nebude volat například metodu `windowsDirectory()` pod operačním systémem GNU/Linux (kompilace by neproběhla úspěšně).

5 DOKUMENTACE KNIHOVNY WDEGUI

WdeGui je GUI knihovna masivně využívající technologie implementované v knihovně WdeCore. Jedná se o knihovnu, která se snaží být co nejméně závislá na ostatních knihovnách zejména v operačním systému GNU/Linux, kde vyžaduje jen X Window System a freetype2[3] knihovnu pro renderování písma.

5.1 Obsah knihovny WdeGui

- hlavní cyklus aplikace
- systém událostí a zpráv
- časovače
- kreslící subsystém včetně načítání obrázků ze souboru nebo z paměti
- grafické uživatelské prvky – widgety

5.2 Instance aplikace a hlavní cyklus programu

WdeGui knihovna obsahuje třídu `Wde::Application`. Tato třída representuje instanci aplikace a lze si k ní dostat pomocí globální proměnné `Wde_application` nebo pomocí statických metod třídy `Wde::Application`. Tato instance je vytvořena knihovnou WdeGui ještě před vstupem do funkce `main()` nebo `WinMain()`. Makro WDE_MAIN() slouží pro korektní deklaraci vstupního bodu aplikace a rozvine se jako `main()` nebo `WinMain()`.

Hlavní cyklus programu se spustí metodou `Wde::Application::run()` a ukončí metodou `Wde::Application::quit()`. Všechny zprávy a události jsou vždy zpracovány z hlavního cyklu (i časovače `Wde::Timer`).

5.3 Kreslící subsystém

WdeGui knihovna obsahuje vlastní API pro kreslení dvojrozměrné rastrové grafiky. Vše je založeno na přímému přístupu k jednotlivým pixelům. Kreslení se realizuje pomocí třídy `Wde::Painter`, která kreslí většinou do instance třídy `Wde::Image`. Vykreslování bylo maximálně optimalizováno podle možností CPU a je velmi efektivní i na starších platformách díky technologii MMX (pokud je k dispozici).

Kromě vykreslování geometrických primitiv je podporováno také vykreslování vyhlazeného písma a čtyřbarevných gradientů, pomocí kterých je možné vykreslit profesionálně vypadající uživatelské prvky.

5.3.1 Geometrické prvky

- `Wde::Point` – třída obsahující kartézské souřadnice (x, y) popisující polohu pixelu
- `Wde::Rect` – třída reprezentující obdélník, obsahuje souřadnice (x1, y1) a (x2, y2). Výška a šířka se počítá pomoci vzorce (x2 – x1) a (y2 – y1). Je patrné že krajní souřadnice x2 a y2 nenáleží obdélníku (tento model se běžně používá v počítačové grafice)
- `Wde::Size` – třída obsahující velikost obdélníku ve formě šířky (width) a výšky (height)

5.3.2 Wde::Image – obrázek

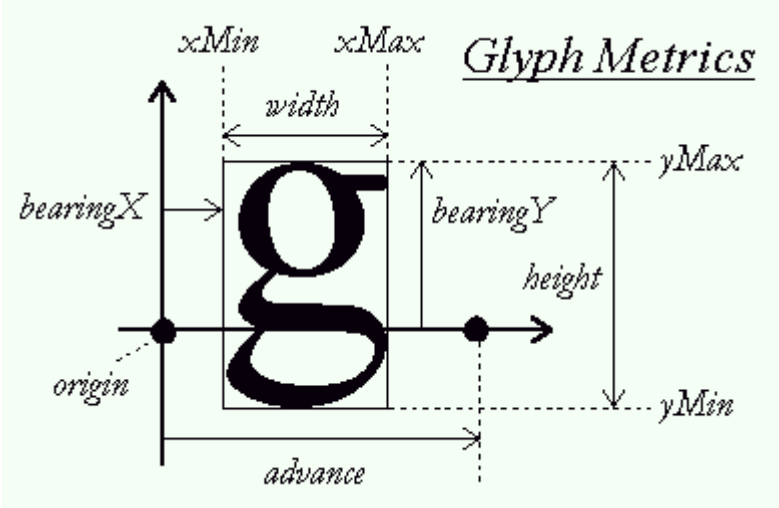
Třída `Wde::Image` reprezentuje rastrový dvojrozměrný obrázek. Obsahuje velikost obrázku, formát pixelů a obrazové data (pixely). Tato třída umožňuje přímý přístup k paměti, kterého využívá zejména kreslící nástroj `Wde::Painter`.

5.3.3 Wde::Font – písmo

Třída `Wde::Font` obsahuje detailní informace, které se týkají typu, velikosti a stylu písma. Font subsystem používá v operačním systému GNU/Linux knihovnu freetype2, pod Windows nativní font rendering (lze zapnout i freetype2).

Třída `Wde::Font` obsahuje metody `set()`, `setSize()`, `setBold()`, `setItalic()` a `setUnderlined()` pro nastavení rodiny a stylu písma. Pro vrácení údajů o písmu nebo textu se používají metody `family()`, `metrics()`, `size()`, `ascent()`, `descent()` a `height()`. Třída `Wde::FontMetrics` obsahuje podrobné informace o písmu.

Metrika jednotlivých znaků vychází z metrického modelu knihovny freetype2.



Obr. 1. Metrický model knihovny freetype2

5.3.4 Wde::Painter – kreslení

Třída `Wde::Painter` slouží ke kreslení rastrové grafiky a používá přímý přístup k obrazovým datům (pixelům). Nejčastější použití je spolu s třídou `Wde::Image`, ale lze použít i ukazatel na pole s pixely a základní informace o něm.

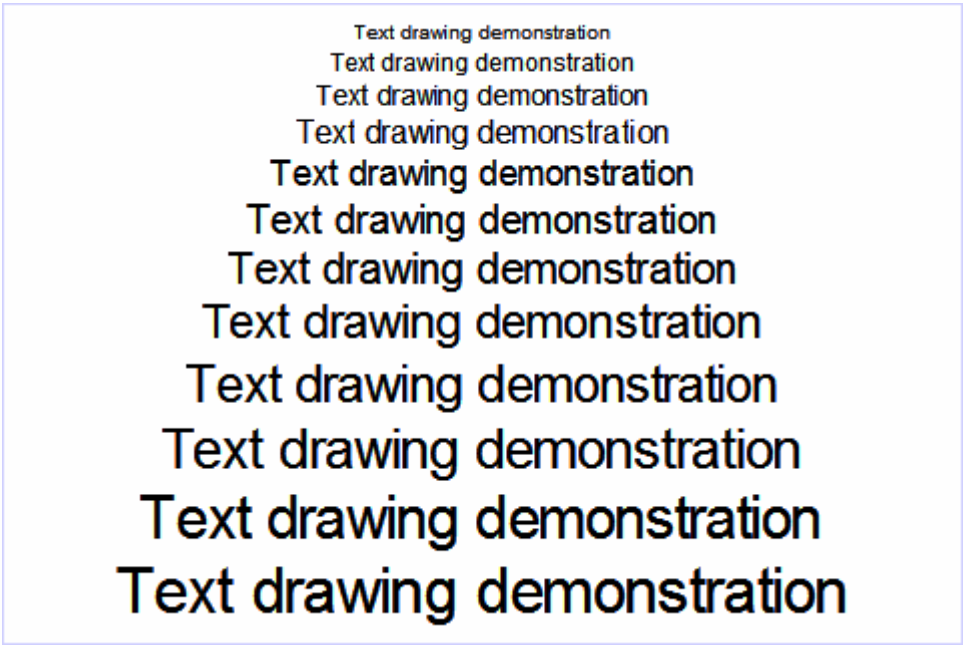
Vlastnosti třídy `Wde::Painter`:

- přímý přístup k paměti
- lze nastavit uživatelský ořezávací region i meta region
- písmo, barva nebo štětec se vždy zadává jako parametr

Příklad kreslení písma:

```
// vytvoření instancí obrázku, písma a kontextu pro kreslení
Wde::Image image;
Wde::Font font;
Wde::Painter p;
int y = 8;

// změna velikosti obrázku na 480x320 pixelů
image.resize(480, 320, Wde::PixelFormat::ID_XRGB32);

// vyčištění obsahu a nakreslení rámečku
p.begin(image);
p.clear(WDE_RGB(255, 255, 255));
p.drawRect(0, 0, p.width(), p.height(), WDE_RGB(208, 208, 255));
```

```
// vykreslení textu v různých velikostech
for (uint size = 10; size <= 32; size += 2, y += font.height() + 1)
{
  font.setSize(size);
  Wde::Rect rect = { 0, y, p.width(), y + font.height() };
  p.drawText(rect,
    "Text drawing demonstration", font, Wde_TextAlign_Center,
    WDE_RGB(0, 0, 0));
}
// ukončení kreslení
p.end();
```

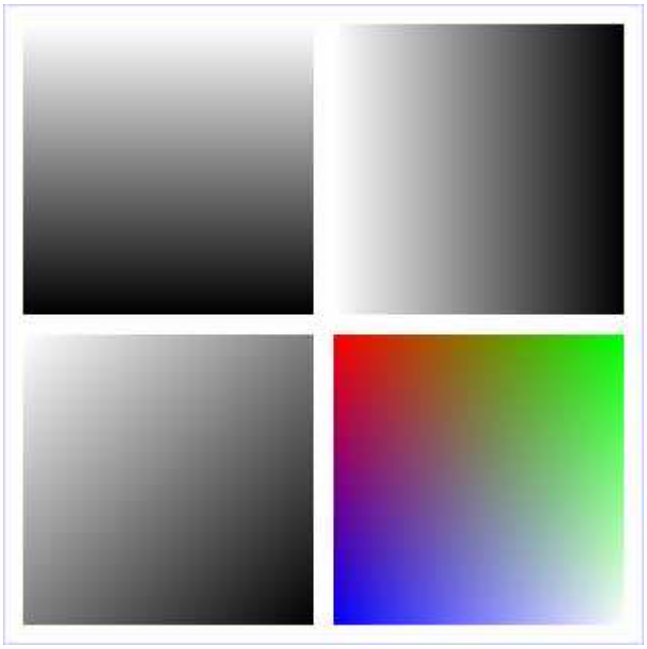


Obr. 2. Výsledek příkladu kreslení písma

Příklad kreslení přechodu:

```
// vytvoření instancí obrázku a kontextu pro kreslení
Wde::Image image;
Wde::Painter p;

// změna velikosti obrázku na 320x320 pixelů
image.resize(320, 320, Wde::PixelFormat::ID_XRGB32);

// vyčištění obsahu a nakreslení rámečku
p.begin(image);
p.clear(WDE_RGB(255, 255, 255));
p.drawRect(0, 0, p.width(), p.height(), WDE_RGB(208, 208, 255));

// vykreslení 4 gradientů
```

```
Wde::RGBA c1 = WDE_RGB(255, 255, 255);
Wde::RGBA c2 = WDE_RGB(0, 0, 0);

p.fillQGradient(10, 10, 165, 155, c1, c1, c2, c2);
p.fillQGradient(170, 10, 310, 155, c1, c2, c1, c2);
p.fillQGradient(10, 165, 165, 310,
  c1, Wde_RGBA_half(c1, c2), Wde_RGBA_half(c1, c2), c2);
p.fillQGradient(170, 165, 310, 310,
  WDE_RGB(255, 0, 0), WDE_RGB(0, 255, 0),
  WDE_RGB(0, 0, 255), WDE_RGB(255, 255, 255));
// ukončení kreslení
p.end();
```



Obr. 3. Výsledek příkladu kreslení přechodu

5.4 Načítání a ukládání obrázků

Knihovna WdeGui obsahuje pokročilou podporu pro načítání a ukládání obrázků různých typů.

5.4.1 Podporované typy obrázků:

- Windows / OS2 Bitmapa (bmp) – načítání / ukládání
- Z Soft Paintbrush (pcx) – načítání / ukládání

- Graphics Interchange Format (gif) – pouze načítání
- Joint Photographic Experts Group[5] (jpeg, jfif) – načítání / ukládání
- Portable Netword Graphics[6] (png) – načítání / ukládání

5.4.2 API pro načítání / ukládání obrázků

Obrázky lze načítat a ukládat pomocí metod třídy `Wde::Image`. Načítání je realizováno pomocí metod `readFile()`, `readStream()` a `readMemory()`. K ukládání se používají metody `writeFile()` a `writeStream()`.

Formát jednotlivých typů obrázků se liší a někdy je potřeba specifikovat parametry načítání nebo ukládání. Přesně k těmto účelům slouží třídy `Wde::ImageEncoder` a `Wde::ImageDecoder`. Pomocí těchto tříd je možné načíst nebo uložit obrázek a specifikovat kvalitu, úroveň komprese nebo například použití RLE.

Příklad uložení JPEG obrázku a nastavení kvality na 75%:

```
// vytvoření obrázku, obsah teď není důležitý
Wde::Image image;
image.resize(100, 100);

// otevření souboru pro zápis
Wde::Stream stream("./image.jpeg",
  Wde::Stream::Open_Write |
  Wde::Stream::Open_Create |
  Wde::Stream::Open_Truncate);

if (stream.isOpen())
{
  // vytvoření instance enkodéru
  Wde::ImageEncoder encoder;
  if (encoder.getById(stream, Wde::ImageFormat_JPEG))
  {
    // nastavení kvality
    encoder.options.quality = 75;
    // zápis
    encoder.encodeImage(image);
  }
}
```

```
// zavření souboru
stream.close();
```

5.5 Uživatelské prvky

WdeGui knihovna obsahuje standardní uživatelské prvky – widgety. Widget slouží k interakci s uživatelem, přijímá události od uživatele nebo toolkitu a reaguje na ně.

5.5.1 Wde::Widget

`Wde::Widget` je základní třída pro všechny widgety. Jedná se o kontejnerový objekt, který může mít svého rodiče (parent) a děti (children). Widget bez rodiče je většinou top-level okno (`Wde::Window`). Každý widget má souřadnice relativní ke svému rodiči a klientské souřadnice. Pouze v klientské části widgetu se mohou vykreslit potomci a pouze na tuto část okna se vztahuje origin.

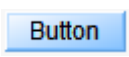
`Wde::Widget` je třída založená na `Wde::Object` a při dědění je třeba použít makro `WDE_OBJECT()` a `WDE_OBJECT_INIT()`. Tyto makra jsou popsány v dokumentaci ke třídě `Wde::Object`. Díky vlastnostem třídy Wde::Object je možné kterýkoliv widget identifikovat pomocí funkce `isClassOf()` nebo `castTo<type>("Type")`.

Pro zobrazení widgetu se používají metody `show()` a `hide()`, pro změnu pozice `move()` a `resize()` a kontejnerovou funkci obstarávají metody `add()` a `remove()`. Většina zmíněných funkcí vrací referenci na objekt, takže zápis `window.add( widget.show() )` je platný i dobře čitelný.

Většina dále popsaných widgetů používá metodu `setText()` pro nastavení textového obsahu. Jde o obecnou metodu implementovanou už ve třídě `Wde::Widget` a má za úkol sjednotit rozhraní.

5.5.2 Wde::Button

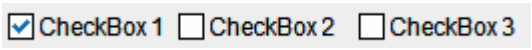
Widget `Wde::Button` patří mezi standardní uživatelské prvky. Je založen na třídě `Wde::AbstractButton`, která implementuje základní chování. Kliknutí vyvolá signál `Click`.



Obr. 4. Widget Wde::Button

5.5.3 Wde::CheckBox

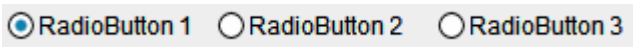
Widget `Wde::CheckBox` se ovládá podobně jako obyčejné tlačítko, ale kliknutí způsobí zaškrtnutí nebo odškrtnutí. Widget tedy obsahuje hodnotu v binární formě, kterou lze vrátit metodou `checked()` a nastavit metodou `setChecked()`.



Obr. 5. Widget Wde::CheckBox

5.5.4 Wde::RadioButton

Widget `Wde::RadioButton` je velmi podobný prvku `Wde::CheckBox`. Při zaškrtnutí se vždy odškrtnou ostatní prvky ve stejné skupině. Skupina lze vrátit a nastavit metodou `radioGroup()` a `setRadioGroup()`.



Obr. 6. Widget Wde::RadioButton

5.5.5 Wde::Label

Widget `Wde::Label` slouží k zobrazení statického textu. Jedná se o pasivní prvek, text nelze nijak ovládat nebo měnit pomocí událostí od uživatele.

Obr. 7. Widget Wde::Label

5.5.6 Wde::GroupBox

Widget `Wde::GroupBox` slouží k vizuálnímu oddělení určité skupiny prvků. Jedná se opět o pasivní prvek, jehož funkce je jen vizuální.

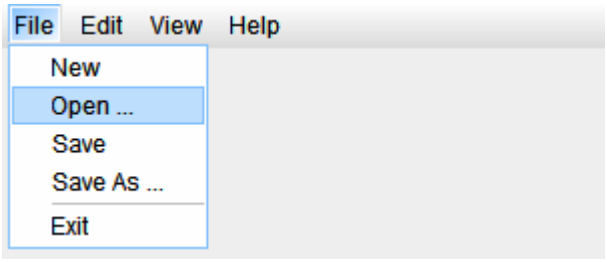
Obr. 8. Widget Wde::GroupBox

5.5.7 Wde::Menu a Wde::MenuPopup

Menu je standardní součást většiny okenních aplikací. Je implementováno ve třídách `Wde::Menu` a `Wde::MenuPopup`. Obě třídy mají velmi podobné vlastnosti (stejné datové struktury), ale liší se v zobrazení. `Wde::Menu` zobrazuje klasické horní menu v aplikaci, `Wde::MenuPopup` se používá jako vyskakovací menu. Jednotlivé položky menu reprezentuje třída `Wde::MenuItem`.

Příklad vytvoření a zařazení menu položek:

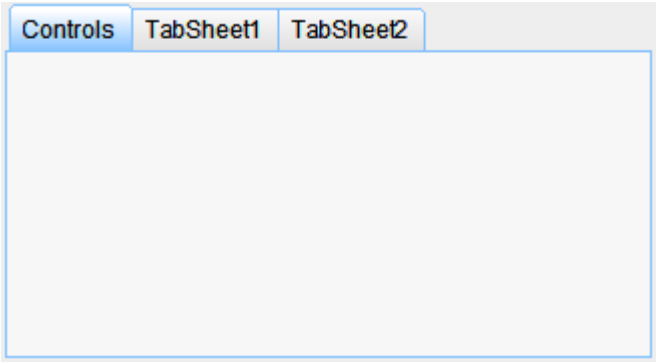
```
Wde::Menu menu;
Wde::MenuItem menuFile;
Wde::MenuItem menuFileExit;

menu.append( menuFile.setText("File") );
menuFile.append( menuFileExit.setText("Exit") );
```



Obr. 9. Widget Wde::Menu

5.5.8 Wde::TabWidget a Wde::TabSheet

`Wde::TabWidget` je standardní uživatelský prvek, do kterého se skládají widgety typu `Wde::TabSheet`. Vždy je aktivní jen jeden a uživatel má možnost přepínat mezi nimi. Pro vkládání widgetů se používají standardní metody `add()` a `remove()`, které jsou deklarovány ve třídě `Wde::Widget`.



Obr. 10. Widget Wde::TabWidget

5.5.9 Wde::ComboBox

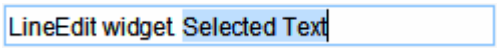
Ve widgetu `Wde::ComboBox` je implementována vyskakovací nabídka. K vložení položek slouží metody `setItems()`, `addItem()` a `insertItem()`. Odstranění položek ze seznamu je implementováno v metodě `removeItem()` a `removeItems()`. V případě změny výběru se volá signál `Change`.

Vyskakovací část je implementována samostatně ve widget `Wde::ListPopup`. Jde o obecný widget, který může být použit nezávisle na prvku `Wde::ComboBox`.

Obr. 11. Widget Wde::ComboBox

5.5.10 Wde::LineEdit

Widget `Wde::LineEdit` patří mezi pokročilejší prvky. Umožňuje editaci nebo prohlížení jednořádkového textu. Pro uchování pozice kurzoru a výběru se používá třída `Wde::LinePosition`, která obsahuje proměnné `begin` (začátek výběru) a `cursor` (pozice kurzoru nebo konec výběru). Metoda `position()` vrací referenci na instanci třídy `Wde::LinePosition`.



Obr. 12. Widget Wde::LineEdit

5.5.11 Wde::SpinBox

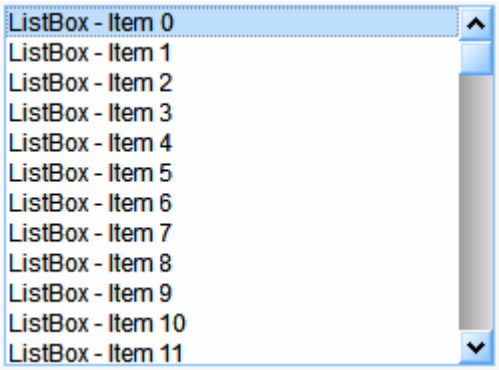
Widget `Wde::SpinBox` rozšiřuje `Wde::LineEdit` o navigační šipky. Pomocí šipek lze jednoduše „naklikat“ požadovanou hodnotu v daném rozsahu. Pro nastavení minima a maxima slouží metody `setMinimum()` a `setMaximum()`. Pro vrácení a nastavení hodnoty slouží metody `setValue()` a `value()`. Hodnota je vždy 32 bitové znaménkové číslo.

Obr. 13. Widget Wde::SpinBox

5.5.12 Wde::ListBox

Widget `Wde::ListBox` zobrazuje jednoduchý seznam položek. Aktivní je vždy jen jedna položka (nebo žádná). Základní implementace je založená na seznamu řetězců `Wde::StringList` a widget obsahuje podobné operace jako zmíněná třída. Při každé modifikace je automaticky widget označen k překreslení.

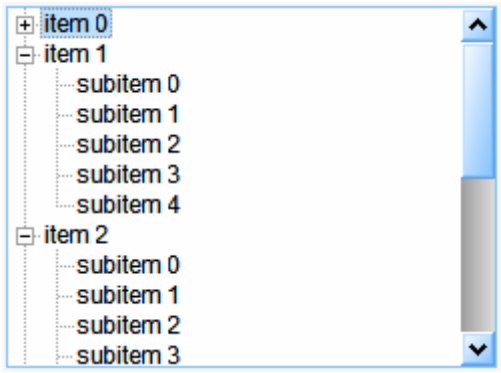
Widget `Wde::ListBox` obsahuje standardní metody `setItem()`, `addItem()`, `insertItem()` a `removeItem()`. Pro přístup k položkám se používá `items()`, a pro změnu všech položek je vhodné použít `setItems()`.



Obr. 14. Widget Wde::ListBox

5.5.13 Wde::TreeBox

Widget `Wde::TreeBox` je jeden z nejpokročilejších prvků implementovaných v knihovně WdeGui. Zobrazuje stromovou strukturu, která se skládá z instancí třídy `Wde::TreeItem`. Pomocí funkcí `Wde::TreeBox` nijak nelze modifikovat obsah stromu. Pro modifikaci se používají přímo metody stromových dat (`Wde::Node` nebo `Wde::TreeItem`), které automaticky vyvolají notifikaci.



Obr. 15. Widget Wde::TreeBox

5.6 Podpora změny vzhledu

Knihovna WdeGui má vestavěnou podporu pro změnu vzhledu aplikace. Veškeré funkce, které kreslí uživatelské prvky jsou implementovány ve třídě `Wde::Theme`. Pomocí C++ dědičnosti je možné vytvořit zcela nový vzhled vybraným uživatelským prvkům (virtuální metody).

Třída `Wde::Theme` obsahuje také výchozí barvy, které by měly být použity ke kreslení. Pozadí je vždy definováno jako 2 barvy, které většinou slouží jako základ pro kreslení barevných přechodů (zároveň ale nenutí tyto přechody používat).

ZÁVĚR

Cílem bakalářské práce bylo vyvinout multiplatformní GUI knihovnu pro operační systémy Windows a GNU/Linux. Knihovna byla rozdělena na dvě části. V první části je implementována abstrakční vrstva mezi operačními systémy, druhá část umožňuje samotnou tvorbu okenních aplikací. Abstrakční vrstva se nazývá WdeCore a uživatelská vrstva WdeGui.

Od samého počátku vývoje byl kladen důraz na efektivitu a binární velikost obou knihoven. Programovací jazyk C++ posouvá psaní aplikací o krok dále, protože při správném použití šablon lze navrhnout opravdu velice výkonné API, se kterým by se například v jazyce C velmi špatně zacházelo. Vytváření dočasných řetězců (nebo i jiných dočasných objektů) pomocí C++ šablon a zajistit jejich alokaci na zásobníku je preferovaná metoda knihovny Wde, kterou bych velmi rád viděl i u ostatních knihoven.

Abstrakční knihovna WdeCore tvoří negrafický základ celého toolkitu. Jedná se o kompletní knihovnu, která se může použít i pro konzolovou aplikaci. V této knihovně je implementována podpora pro vícevláknové aplikace, unikód a značkovací jazyk XML. Obsahuje mimo jiné i velké množství tříd, které velmi usnadňují multitplaformní vývoj a lokalizaci.

Uživatelská knihovna WdeGui obsahuje kompletní uživatelské rozhraní a silnou podporu pro práci s rastrovou grafikou. Umožňuje vykreslovat uživatelské prvky pomocí vlastního kreslícího subsystému, který dokáže využít technologie MMX a SSE2. Knihovna umí načítat obrázky různých typů a převádět je mezi sebou. Bylo implementováno i vykreslování barevných přechodů a renderování vyhlazeného písma. Jednotný subsystém pro kreslení umožňuje, aby aplikace vypadala stejně na všech podporovaných operačních systémech.

Implementace vlastního okenního systému, který běží v top-level okně ukázala, že i pod X Window System lze napsat velmi efektivní okenní aplikace, ve kterých se nesetkáme s nežádoucím „flicker“ efektem při překreslování. Napsáním Wde knihovny jsem ukázal, že existují i jiné cesty při implementaci uživatelského rozhraní.

EDUCT

The target of my bachelor thesis was to create multiplatform GUI library for both Windows and GNU/Linux operating systems. The library was divided to two parts. First part deals with abstract layer for operating systems and second part enables to create applications with graphics user interface itself. Abstract layer is called WdeCore and second layer is called WdeGui.

Since very beginning of library creation efficiency and binary size have been emphasized. C++ language is big step in process of creating applications, if templates are used in right way it is possible to design very powerful API (language C itself does not allow such a pretty possibility). Allocating of temporary strings (and other temporary objects) with C++ templates and moving them on the stack is preferable method of Wde library, which should be used in other libraries as well.

Abstract library WdeCore is non-GUI base for whole toolkit. It is complex library and it allows to create console applications too. Supports multithreading, unicode and XML language. It contains rich set of classes which can be very useful for multiplatform development and localization.

User library WdeGui contains complex user interface and strong support for raster graphic. It allows to draw user interface entities through its own graphic subsystem, which is able to use both MMX and SSE2 technologies (when possible). Various types of image files can be imported by library itself and exported in different formats. Support for both color gradient and antialiased text rendering was implemented into library as well. Advantage of unique graphic subsystem is the same user interface appearance on all supported operating systems.

Demonstration of this library shows, that it is possible to create very effective window application running under X Window System in top level window without "flicker" effect, which can appear during redrawing. By creating this library, I wanted to demonstrate alternative ways in user interface implementation.

SEZNAM POUŽITÉ LITERATURY

[1] Cmake [online], dostupný s www:
http://www.cmake.org

[2] Fltk knihovna [online], dostupný s www:
http://www.fltk.org

[3] Freetype knihovna a dokumentace [online], dostupný s www:
http://www.gtk.org

[4] Gtk+ Toolkit [online], dostupný s www:
http://www.gtk.org

[5] Jpeg knihovna [online], dostupný s www:
http://www.ijg.org

[6] Png knihovna [online], dostupný s www:
http://www.libpng.org/pub/png

[7] Qt Toolkit [online], dostupný s www:
http://www.trolltech.com

[8] Wikipedia [online], dostupný s www:
http://cs.wikipedia.org/wiki/BSD_licence

[9] wxWidgets [online], dostupný s www:
http://www.wxwidgets.org

[10] Unicode [online], dostupný s www:
http://www.unicode.org

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API Application programming interface – soubor struktur a funkcí, které program nebo knihovna poskytuje.

ASCII American Standard Code for Information Interchange – základní kódování pro latinskou abecedu.

BSD Berkeley Software Distribution – unixový operační systém a stejnojmenná licence používaná pro free-software.

DLL Dynamic linked library – dynamické knihovna.

Flicker Nežádoucí efekt, který způsobuje špatné překreslování oken (nejdříve je vidět vykreslení pozadí, poté vykreslení ostatních prvků).

GNOME GNU Network Object Model Environment – desktopové prostředí pro unixové operační systémy využívající Gtk+ toolkit.

GPL GNU General Public Licence – druh licence používaný pro free-software.

Gtk GIMP Toolkit – GUI knihovna napsaná v jazyce C.

GUI Graphics User Interface – grafické uživatelské rozhraní.

IDE Integrated Development Environment – integrované vývojové prostředí.

IO Input / Output – vstup a výstup.

KDE K Desktop Environment – desktopové prostředí pro unixové operační systémy využívající Qt toolkit.

LGPL GNU Lesser General Public Licence – druh licence používaný pro free-software, benevolentnější než GPL (umožňuje použití dynamicky linkovaných knihoven v proprietárním produktu).

MinGW Minimal GNU for Windows

MMX Matrix Math Extension (v literatuře se uvádí i MultiMedia Extension) – rozšíření rodiny procesorů x86 o vektorové celočíselné operace.

MFC Microsoft Foundation Classes – knihovna nad WinAPI.

MOC Meta Object Compiler – nástroj firmy Trolltech k toolkitu Qt.

Motif	Komerční toolkit pro X Window System.
OS	Operační systém
POSIX	Portable Operating System Interface – speficikace pro unixové operační systémy, obsahuje specifikace i pro C knihovnu, vlákna, ...
QPL	Q Public Licence – nekomerční licence pro Qt toolkit.
RAD	Rapid application development – vizuální nástroj pro vývoj GUI aplikací
RLE	Run Length Encoding – jednoduchý způsob komprese obrázků založen na principu spojení stejnobarevných částí na ose X.
RTTI	Run Time Type Information – dynamické informace o třídě v C++ jazyce
SSE	Streaming Simd Extension – rozšíření rodiny procesorů x86 o vektorové operace s plovoucí desetinou tečkou.
SSE2	Streaming Simd Extension 2 – rozšíření rodiny procesorů x86 o vektorové operace s celočíselnými typy.
STL	Standard Template Library – standardní knihovna jazyka C++
UCS	Universal Character Set – univerzální kódovací tabulka, obsahuje znaky všech jazyků
UTF	UCS Transformation Format – způsob kódování unicode řetězců
XML	Extensible Markup Language – značkovací jazyk, který pochází z rodiny SGML (Standard Generalized Markup Language).
XShm	X Shared Memory Extension – rozšíření pro X Window System

SEZNAM OBRÁZKŮ

SEZNAM TABULEK