VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POROVNÁNÍ IMPLEMENTACE GUI V RŮZNÝCH KNIHOVNÁCH V OS LINUX

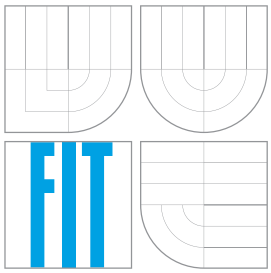
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN ŠELEPA

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POROVNÁNÍ IMPLEMENTACE GUI V RŮZNÝCH KNIHOVNÁCH V OS LINUX

COMPARING GUI IMPLEMENTATION IN DIFFERENT LIBRARIES IN OS LINUX

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN ŠELEPA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN ČERMÁK

BRNO 2009

Abstrakt

Tato práce prezentuje základní vlastnosti a principy grafických uživatelských rozhraní. Dále ukazuje způsob, jakým se s grafickým uživatelským rozhraním pracuje na operačním systému GNU/Linux a popisuje některé knihovny používané při tvorbě aplikací s grafickým rozhraním. V další části práce je uveden návrh uživatelského rozhraní pro aplikaci postavenou na funkčním antivirovém démonovi. Poslední část práce poukazuje na rozdíly při implementaci navržené aplikace.

Abstract

This thesis presents the elemental properties and principles in graphical user interface. It then shows how one can work with a graphical user interface on GNU/Linux operating system and gives some examples of toolkits used for this purpose. The next part shows the design of an aplication based on an existing anti-virus program. The last part shows the differences between the two implementations of the designed application.

Klíčová slova

Linux, grafické uživatelské rozhraní, sada nástrojů, prvek rozhraní, licence, multiplatformnost.

Keywords

Linux, graphical user interface, toolkit, widget, license, cross-platform.

Citace

Jan Šelepa: Porovnání implementace GUI v různých
knihovnách v OS Linux, diplomová práce, Brno, FIT VUT v Brně, 2009

Porovnání implementace GUI v různých knihovnách v OS Linux

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Martina Čermáka. Další informace mi poskytl Ing. Jaromír Smrček spolupracující s firmou ZONER software, a.s.

.....
Jan Šelepa
25. května 2009

Obsah

Kapitola 1

Úvod

Uživatelské rozhraní je velice důležitou součástí návrhu jakékoli aplikace. Jak aplikace vypadá a jak se dá s touto aplikací komunikovat do jisté míry ovlivňuje její použitelnost. S rozvojem uživatelského rozhraní roste i počet lidí, kteří jsou schopni ovládat a používat takto vyvíjené aplikace. Vývoj grafických uživatelských rozhraní je tedy velice důležitý i v komerční sféře, protože každý výrobce proprietárního softwaru se snaží o co největší rozšíření svého produktu.

Na začátku tohoto dokumentu uvedu základní pojmy z uživatelských rozhraní s tím, že se zastavím u samotného pojmu grafická uživatelská rozhraní a jeho vývoje od prvního nápadu až k dnešní podobě, kterou známe z různých platforem.

Dále se v této práci budu soustředit na vývoj grafických uživatelských rozhraní pomocí moderních nástrojů. V dnešní době totiž existuje již celá řada dostupných knihoven, pomocí kterých se dají relativně jednoduše vytvářet sofistikovaná uživatelská rozhraní se standardním vzhledem a chováním. Některé z těchto nástrojů popíši podrobněji a budu se snažit vyzdvihnout jejich vzájemné rozdíly.

Další část tohoto dokumentu bude věnována návrhu aplikace s grafickým uživatelským rozhraním, která bude spolupracovat s již existujícím antivirovým démonem od firmy ZONER software, a.s. V této části také uvedu knihovny, pomocí kterých budu posléze danou aplikaci implementovat.

Velice důležitou sekci práce tvoří část, kde rozdělím vytvořenou aplikaci na jednotlivé grafické prvky či logické části, u kterých uvedu postupy při jejich implementaci ve vybraných knihovnách a vyzdvihnu rozdíly mezi těmito knihovnami.

Kapitola 2

Grafická uživatelská rozhraní

2.1 Definice

Grafické uživatelské rozhraní je způsob, jakým uživatel komunikuje s počítačem. Tento způsob používá grafické prvky jako okna, menu nebo ikony, které mohou být ovládány myší a do určité míry i klávesnicí, připojenou k danému počítači.

V ostrém kontrastu s grafickým uživatelským rozhraním stojí rozhraní s příkazovou řádkou, které prezentuje veškeré informace v textové podobě a lze jej ovládat pouze pomocí klávesnice. Příkladem takového rozhraní je příkazový řádek systémů MS-DOS nebo GNU/Linux (popřípadě grafická konzole).

Ikony jsou v podstatě malé obrázky nebo symboly v grafickém uživatelském rozhraní, které reprezentují určitý program, příkaz, soubor a podobně. Ikony jsou užívány jak na pracovní ploše, tak přímo v aplikacích.

Okno je většinou definováno jako obdélníková část obrazovky, kde je zobrazován jeho obsah (například nějaká aplikace, text či obrázek) nezávisle na obsahu zbytku obrazovky. Velkou výhodou je možnost spravovat více takovýchto oken najednou. V každém z těchto oken je pak možné otevřít jinou aplikaci nebo zobrazovat obsah souborů, které byly otevřeny nebo vytvořeny v jedné aplikaci.

Menu obsahuje jednu či více nabídek, které může nastavit. Hlavní menu aplikace většinou obsahuje všechny možnosti, které může uživatel ovlivnit. Tak si nemusí pamatovat různé parametry a, protože je vidí všechny pohromadě, snadněji si vybere možnost, která nejvíce odpovídá tomu, čeho chce dosáhnout. Menu se skládá z jednodušších prvků jako jsou tlačítka. Existují také různé druhy menu: roletové (pull-down), vynořující se (pop-up) a další.

Formuláře a dialogy jsou okna obsahující tlačítka, seznamy a grafické prvky pro reprezentaci a výběr informace.

Příkazy se v grafickém uživatelském rozhraní udělují většinou pomocí myši, která ovládá pohyb kurzoru po obrazovce. Pro udělení příkazu je nutné najet kurzorem myši na určitý prvek grafického uživatelského rozhraní, tedy ikonu, okno nebo třeba menu. Další akce závisí na tom, co chceme s určitým prvkem dělat. Například aplikace jdou spouštět kliknutím na jejich ikonu, okna lze posouvat tažením za jejich horní okraj a tak podobně.

2.2 Vlastnosti

Asi největší výhodou při používání grafického uživatelského rozhraní je, že usnadňuje práci s počítačem a ta se tak stává více intuitivní. Čím je práce na počítači více intuitivní, tím lépe se dá naučit. Pro nového uživatele je například jednodušší přesunout soubor z jednoho adresáře do druhého tím, že přetáhne ikonu souboru pomocí myši, než pokud by si měl pamatovat určitou formu pro textový příkaz do konzole.

Na intuitivnosti přidává i fakt, že pro každou operaci existuje její vizuální odezva. Pokud například uživatel smaže soubor v grafickém uživatelském rozhraní, pak ikona, která reprezentovala tento soubor, okamžitě zmizí. Pokud oproti tomu v příkazovém řádku uživatel zadá příkaz pro smazání souboru spolu s názvem tohoto souboru, neobjeví se mu žádná odezva, která by potvrzovala úspěšné provedení této operace.

Grafické uživatelské rozhraní také umožňuje plně využít výhod možnosti zpracovávat více úloh najednou u moderních operačních systémů spuštěním několika různých aplikací v samostatných oknech. Výsledkem tohoto přístupu je zvýšení flexibility a zvýšení produktivity práce s počítači.

Grafické uživatelské rozhraní je dnes už daleko více než jen pouhá vymoženost. Z tohoto rozhraní se stal standard při komunikaci člověka se strojem, který ovlivnil práci generace uživatelů. Grafické rozhraní také přispělo k vytvoření úplně nových typů aplikací.

I přes tyto vlastnosti jsou takoví uživatelé, kteří raději používají rozhraní s příkazovou řádkou. Tuto možnost většinou používají správci a pokročilí uživatelé operačních systémů, protože operace v příkazové řádce bývají většinou mocnější, s více možnostmi, než které by se daly vykonat přes grafické rozhraní. V unixových operačních systémech jsou grafická rozhraní jednotlivých programů většinou jen nástavbou nad programy, které fungují v příkazové řádce a veškerou svoji funkčnost přebírají od svých konzolových základů.

Vytvořením grafického prostředí pro aplikaci, která běží v příkazové řádce ovšem lze umožnit začínajícím uživatelům využívat většinu funkčnosti dané aplikace při jednoduchém a intuitivním ovládání, zatímco uživatelé, kteří potřebují využívat plnou funkčnost této aplikace využijí silnější příkazovou řádku.

2.3 Historie

Počátky grafického uživatelského rozhraní je možné najít již v roce 1945, kdy Vannevar Bush, vědec pracující na Technické univerzitě v Massachusetts, navrhl ve svém článku *As We May Think* administrativní nástroj, který by uchovával informace na mikrofilmech a tyto informace by byly snadno dostupné, navzájem provázané odkazy a programovatelné.

V roce 1963 napsal Ivan Sutherland jako svou disertační práci s názvem Sketchpad program, který umožnil přímou manipulaci grafických objektů na obrazovce monitoru pomocí světelného pera. Jeho návrh zahrnoval přibližování a oddalování, opatření v paměti pro ukládání objektů a možnost kreslení přesných čar a rohů na obrazovce (více v [3]).

Douglas Engelbart, který zkoumal interakci mezi člověkem a počítačem na Stanfordské univerzitě, představil v roce 1968 první primitivní myš. Jeho produkt s názvem X-Y Position Indicator byla dřevěná krabička na kolečkách, která pokud se pohybovala po horizontální podložce ovládala kurzor na obrazovce.

Prvním systémem zahrnujícím všechny vlastnosti moderního grafického uživatelského rozhraní byl Alto (viz. [2]). Alto byl postaven firmou Xerox pro výzkumné účely jako malý, ale silný osobní počítač, který uměl vyjadřovat informace v grafické podobě. Tento počítač

měl monochromatickou bitmapovou obrazovku, na které se vykreslovalo velice jednoduché grafické uživatelské rozhraní. První počítač Alto byl vyroben v roce 1973.

Později stejná firma, tedy firma Xerox, vyvinula v roce 1981 systém Star, který jako první používal pracovní plochu v tom smyslu, jak chápeme význam tohoto slova dnes. V tomto systému se pracovalo s okny, ikonami, složkami a tak podobně. Star byl původně pouze název softwaru, který byl dodáván společně s pracovní stanicí 8010, ale postupně se název Star začal používat ve spojení s celou touto stanicí.

V roce 1983 začala společnost Apple Computers vyrábět svůj systém, který byl pojmenován Lisa Office System. Všechno na tomto stroji od hardware, přes operační systém až po aplikace bylo vytvořeno za účelem získání výkonného systému pro zpracovávání dokumentů.

Tentýž rok také firma VisiCorp vydává svůj produkt nazvaný VisiOn. Je to první grafické uživatelské rozhraní pro PC od IBM, které plnohodnotně využívá principu pracovní plochy. Tento systém byl ovšem ve své době v celku náročný na hardware, na kterém běžel. Systém VisiOn byl naprogramován v upravené verzi jazyka C tak, aby bylo možné ho portovat na jiné operační systémy jako je například Unix, nebo i na jiné procesory než 8086. Všechny aplikace, které byly psané pro VisiOn totiž byly napsány tak, že využívaly virtuální stroj, který byl nezávislý na procesoru. Jen jádro VisiOn bylo psáno jako závislé na stroji. I přes tuto svoji portabilitu VisiOn nikdy neběžel jinde, než na IBM PC s operačním systémem MS-DOS. Grafické uživatelské rozhraní VisiOn přinášelo možnost ovládání aplikací myší, konzistentní vzhled všech aplikací, možnost práce s více aplikacemi najednou, sdílení dat mezi aplikacemi, nápovědu a dokonce vestavěný instalátor.

Ve stejném roce firma Microsoft oznámila vývoj svého prostředí Windows. Podobná prostředí v té době, jako VisiOn nebo Lisa, byla na tehdejší dobu velice náročná na hardware. Microsoft sliboval, že Windows budou mít nižší požadavky a budou za nižší cenu. První verze Microsoft Windows vyšla v roce 1985.

V roce 1984 vznikl na MIT projekt s názvem X Window, který se velice rozšířil a je v dnešní době považován za standard pro operační systémy jako GNU/Linux a ostatní Unixové systémy.

2.4 Budoucnost

Vývoj grafického uživatelského rozhraní se díky stále lepšímu hardware (rychlejší procesory, více paměti a větší rozlišení monitorů), vývoji v software a stále větší poptávce ze strany uživatelů stále rozvíjí.

Jednou z velice zajímavých oblastí pro rozvoj grafických rozhraní je takové rozhraní, které poskytuje iluzi ovládání aplikace v trojrozměrném prostoru.

Další možnosti se také nacházejí v umožnění uživateli více kontrolovat grafické objekty, jako například rotovat, měnit velikost nebo průhlednost ikon.

Výzkum také probíhá v oblasti využití řeči jako doplňku ke grafickým rozhraním pro ovládání jednotlivých aplikací.

Kapitola 3

Grafická uživatelská rozhraní v OS GNU/Linux

3.1 X Window System

Pro to, aby vůbec bylo možné vytvořit jakékoli grafické uživatelské rozhraní, je potřeba mít určitý systém, který zajistí plochu, na které by se uživatelské rozhraní mohlo vykreslovat. Na unixových operačních systémech je nejrozšířenější systém X Window.

Tento systém nejenže poskytuje rastrovou plochu pro vykreslování, ale také poskytuje principy a nástroje pro vykreslování a manipulaci se základními primitivy, tedy s okny, a pro jejich interakci se vstupními zařízeními jako je klávesnice a myš.

X Window ovšem nijak nedefinuje, jak by mělo vypadat uživatelské rozhraní každé aplikace, to nechává na samotných klientských aplikacích. To vede k velkým rozdílům ve stylech mezi jednotlivými implementacemi aplikací. Všechny moderní knihovny a toolkity pro tvorbu grafického uživatelského rozhraní staví právě na systému X Window.

3.1.1 X Server

X Window také spravuje komunikaci jednotlivých aplikací s grafickou kartou a ovládacími prvky, které jsou přístupné uživateli. Komunikace mezi těmito instancemi probíhá pomocí X Protokolu, který zajišťuje síťovou transparentnost.

Na stroji s grafickou kartou běží program nazývaný X Server, který komunikuje s různými klientskými programy. X Server přijímá požadavky na vykreslení od klientského programu a vykresluje na monitor uživatele. Dále také X Server zachytává vstupní informace od uživatele (myš, klávesnice) a posílá tyto informace klientskému programu.

V systému X Window běží server na počítači, u kterého je uživatel, kdežto programy, které uživatel spouští a ovládá mohou být na úplně jiném počítači. Toto uspořádání je opačné než na většině architektur klient-server, kde klientská aplikace běží u uživatele a server je na vzdáleném počítači.

3.1.2 Správce oken

Správce oken je další aplikace, která běží nad systémem X Window. Oproti Mac OS a Microsoft Windows, kde se vzhled oken a jejich nastavení řídí podle toho jak to výrobce požadoval, se správa oken schválně drží oddělená od systému, který spravuje grafický systém. Správce oken pro systém X Window si může uživatel zvolit z dlouhé řady produktů

třetích stran, které se od sebe značně liší. Odlišnosti zahrnují i rozsah možností, které uživatel může nastavit v rámci daného správce oken, potřebné prostředky pro běh správce oken nebo stupeň začlenění správce do pracovní plochy.

Pokud na systému běží jakýkoli správce oken, je část komunikace mezi X Serverem a klienty přesměrována přes tohoto správce oken. Pokud například přijde požadavek na zobrazení nového okna je tento požadavek přeposlán ke správci oken a ten určí počáteční pozici nového okna. Moderní správci oken navíc provádějí takzvaný reparenting, což umožní správci oken umístit dekorace (jako horní panel a okolní rámeček) k danému oknu. Takto vytvořené elementy jsou ve vlastnictví správce oken a při interakci s nimi uživatel komunikuje se správcem oken, který provádí veškeré operace.

3.1.3 Vývoj

Projekt X Window vznikl roku 1984 na Technické univerzitě v Massachusetts jako společný projekt Jima Gettyse a Boba Scheiflera. Gettys pracoval na projektu Athena, kde byl navržen protokol, nad kterým běžely lokální aplikace a zároveň bylo možné s jeho pomocí volat vzdálené prostředky. V květnu roku 1984 nahradil Scheifler synchronní protokol z projektu Athena asynchronním protokolem a nazval ho X verze 1. X se stal prvním okenním systémem, který byl úplně nezávislý na hardware a na výrobci.

Na této počáteční verzi začali pracovat společně Scheifler, Gettys a Ron Newman a na začátku roku 1985 už vydali verzi 6. Tato verze byla naportována do MicroVAX od firmy DEC. Ještě v první polovině stejného roku byl projekt X Window upraven pro firmu DEC tak, aby podporoval barvy na jejich novém stroji VAXstation-II/GPX. Z toho se zrodila verze 9 systému X Window.

Skupina vědců na Brown University pozměnila systém tak, aby běžel na RT/PC od firmy IBM, ale určité problémy v kompatibilitě vedly k předělání protokolu na X10 na konci roku 1985. V dalším roce se o systém X Window začaly zajímat i externí organizace. Tento zájem vedl k vytvoření X10R3. Tuto verzi se tvůrci rozhodli licencovat takzvanou MIT licencí, oproti X6, která byla zpoplatněná. Změnou licenční politiky doufali tvůrci ve větší rozšíření.

Ačkoli X10 byl výkonný, proto aby se mohl rozšířit ve velkém měřítku bylo potřeba předělat jeho design na ještě méně hardwarově závislý. Toho se ujala firma DEC. Verze X11 vyšla v září roku 1987 a byla pod stejnou licencí jako verze předcházející. Výsledná podoba tohoto projektu byla diskutována na internetu.

V roce 1988 vzniklo MIT X Consortium jako nezisková organizace, která měla zaručit další vývoj projektu bez komerčních nebo vzdělávacích vlivů. Tato skupina vytvořila několik důležitých revizí X11. V roce 1993 se z MIT X Consortium přetvořilo na X Consortium, Inc., které bylo také neziskové. Tato skupina pokračovala ve vývoji a opět vydala několik důležitých revizí. Koncem roku 1996 se tato skupina však rozpadla, přičemž jejich poslední vydaná revize měla číslo X11R6.3.

Poté se na rok ujímá řízení projektu firma The Open Group, která pro svoji verzi změnila licenční podmínky. Tato politika však dlouho nevydržela a byla opět obnovena stará licence.

Firma XFree86 pracovala na úpravě verze X11R5 z roku 1991, ale časem se jejich snažení rozvíjelo a postupně se v podstatě stala vedoucím projektu X. V roce 1995 se firma XFree86, která vybudovala nejpopulárnější verzi X té doby, spojila s firmou X.org, která vznikla z firmy The Open Group. Tyto dvě firmy poté začaly uvažovat o reorganizaci, která by prospěla vývoji X. Na počátku roku 2004 vydává firma XFree86 verzi X, která má přísnější licenci, což mnoho projektů závisejících na X bralo jako nepřijatelné.

Začátkem roku 2004 vzniká z několika členů X.org a freedesktop.org skupina zvaná X.Org Foundation. Toto znamenalo velkou změnu ve vedení projektu. Zatímco od roku 1988 vedly vývoj prodávající firmy, X.Org Foundation bylo vedeno softwarovými vývojáři, kteří využívali při vývoji komunity kolem projektu. V dubnu roku 2004 vyšla verze X11R6.7. Od té doby je vývoj X Window stabilní a byla vydána řada důležitých verzí. Poslední verze dnes je X11R7.4.

3.1.4 Xlib

Xlib je knihovna k X Window systému napsaná v jazyce C, která obsahuje funkce pro komunikaci s X Serverem. Tyto funkce umožňují programátorům používat funkčnost X Serveru bez přesné znalosti protokolu. Pomáhají také například tím, že programátora odstíní od vlivů latence sítě.

Funkce používané v knihovně Xlib lze rozdělit do tří skupin:

1. Operace připojování/odpojování (XOpenDisplay, XCloseDisplay apod.),
2. Žádosti na server, které se dají ještě rozdělit na:
   - žádosti o operace (XCreateWindow, XCreateGC apod.),
   - žádosti o informace (XGetWindowProperty apod.);
3. Operace na lokálním klientovi, které se opět dají ještě rozdělit na:
   - operace s frontou událostí (XNextEvent, XPeekEvent apod.),
   - operace s lokálními daty (XCreateImage, XSaveContext apod.)

Knihovna Xlib i přes svá zjednodušení obsahuje v podstatě jen volání protokolu. Neobsahuje definici složitějších grafických prvků jako jsou tlačítka, menu a další. Takové definice lze najít ve složitějších knihovnách nebo toolkitech. Některé z těchto toolkitů jsou představeny dále v tomto dokumentu.

3.2 Toolkity

Již od počátku projektu X Window byly paralelně s vývojem knihovny Xlib vyvíjeny nástroje pro pohodlnější tvorbu aplikací. Knihovna Xlib je navržena tak, aby se tato knihovna dala použít při vytváření složitějších nástrojů. Některé z těchto nástrojů jsou zde popsány (viz. také [1]).

3.2.1 X Toolkit

X Toolkit je nástroj pro objektově orientovaný návrh v jazyce C postavený na knihovně Xlib. X Toolkit přináší způsob, jakým lze napsat prvky uživatelského rozhraní, zorganizovat tyto jednotlivé prvky do jednoho celkového designu a spojit takto vytvořené rozhraní s funkčním jádrem vytvářené aplikace. Součástí X Toolkitu jsou také některé již předdefinované prvky uživatelského rozhraní jako například textová pole, posuvníky nebo tlačítka (více v [5]).

Při psaní programu pomocí X Toolkitu si programátor nemusí žádat o každou událost, která přijde od X Serveru, jako je tomu při použití samotné knihovny Xlib, ale pouze zaregistruje daný typ události k proceduře. Pokud přijde ze serveru daná událost, toolkit

pouze zavolá danou proceduru, která se provede. Procedury mohou být svázány například i s určitou posloupností příchozích událostí.

Uživatelské rozhraní tvoří strom instancí jednotlivých prvků tohoto rozhraní. Vnitřní uzly tohoto stromu tvoří prvky uživatelského rozhraní, které samy o sobě nemají žádnou vstupní ani výstupní funkčnost, ale slouží pouze pro správu velikostí a pozic jednotlivých prvků v nich obsažených. Listy tohoto stromu jsou primitivní prvky, které nemohou obsahovat žádné potomky a které se starají pouze o zobrazení potřebné informace. Většina aplikací si tento strom vytváří podle sebe tak, jak přidává jednotlivé prvky uživatelského rozhraní.

Pokud si programátor nevystačí s prvky rozhraní, které nabízí X Toolkit od základu, pak je možné udělat si vlastní prvek, nebo upravit již existující. V prvních verzích X Toolkitu nebylo možné pouze upravit již existující prvek a programátoři byli nuceni psát veškeré prvky od začátku, i když chtěli změnit třeba jen malou část již existujícího. V současné verzi toolkitu je již možné zachovat vlastnosti upravovaného prvku a nadefinovat pouze chování, které je nové, nebo odlišné od již existujícího.

Uživatel může měnit vzhled aplikace napsané pomocí X Toolkitu. Nastavením určité hodnoty v konfiguračním souboru lze například změnit barvy, písmo nebo šířku použitých rámečků. Takto lze měnit vlastnosti pro všechny prvky najednou nebo i pro jednotlivé části uživatelského rozhraní.

Nevýhodou X Toolkitu je, že i přes jeho zjednodušení není toto zjednodušení dostatečné. Pro naprogramování složitějšího grafického uživatelského rozhraní je potřeba kód v řádech několika set řádků. X Toolkit se ale opět využívá jako základ, nad kterým jsou vytvářeny různé sady grafických prvků, například Athena nebo Motif.

Motif

Motif je sada objektů, která je postavena na X Toolkitu. Při vzniku této sady byla v prodeji grafická uživatelská rozhraní jako Windows 3.11. Motif je designován tak, aby co nejvíce připomínal právě prostředí Windows 3.11. I chování jednotlivých prvků Motifu bylo vypracováno podle tehdejších Windows. Oproti samotnému X Toolkitu přináší:

- propracovanější geometrii prvků,
- jejich automatickou konfiguraci,
- možnost používání systému cut&paste a drag&drop,
- nové třídy a objekty,
- vlákna,
- lokalizaci a další.

Motif v sobě také obsahuje takzvanou Style Guide, která určuje jak má vypadat finální aplikace. Tím se v rámci Motifu sjednocuje vzhled všech aplikací. Součástí Motifu je také správce oken, který má stejně zaměřený design jako jednotlivé prvky uživatelského rozhraní v rámci Motifu.

Motif je komerční produkt, ovšem firma The Open Group, která vlastní práva na Motif, vydává verzi s názvem OpenMotif. OpenMotif jsou zdrojové kódy, které jsou stejné jako ty Motifu, rozdíl je však v tom, že OpenMotif lze použít zadarmo pokud platforma, na které aplikace běží je open source.

3.2.2 GTK+

GTK+ je multiplatformní toolkit pro vytváření grafického uživatelského rozhraní. Tento toolkit byl původně vytvořen pro GNU Image Manipulation Program (GIMP), rastrový grafický editor, v roce 1997 Spencerem Kimballem a Peterem Mattisem. GTK+ má LGPL licenci, je to svobodný software a je součástí projektu GNU.

GTK+ bylo vybudováno na čtyřech základních knihovnách (viz. [7]):

- *GLib* - knihovna, která tvoří základ GTK+. Zajišťuje základní potřeby jako je obsluha datových struktur v jazyce C a rozhraní pro funkčnost jako je třeba hlavní smyčka programu (zpracovávání událostí), práce s vlákny, dynamické načítání nebo objektový systém.
- *Pango* - knihovna pro rozložení a vykreslování textů s důrazem kladeným na mezinárodní podporu. Na této knihovně je postavena veškerá práce s textem v GTK+.
- *Cairo* - knihovna pro vykreslování 2D grafických elementů, která podporuje různá výstupní zařízení jako systém X Window nebo Win32. Knihovna se stará o to, aby byl výstup konzistentní nehledě na výstupní zařízení a zároveň využívá hardwarovou akceleraci tam, kde je to možné.
- *ATK* - tato knihovna obsahuje sadu nástrojů, které umožňují lepší přístup k datům aplikace. Podporou této sady nástrojů se aplikacím nabízí možnost použití nástrojů jako zvětšovací lupa nebo čtení textu z obrazovky.

Výhodou GTK+ je podpora velkého množství programovacích jazyků, ve kterých lze psát aplikace pomocí GTK+ toolkitu. Původním programovacím jazykem je jazyk C, ovšem dnes již lze využívat programovací jazyky jako C++, Perl, Ruby, Java, nebo také PHP.

GTK+ bylo původně vyvíjeno pro systém X Window a tento systém dodnes zůstává hlavním cílem projektu. Dnes je možné GTK+ používat také na Microsoft Windows (Windows 2000 a novější) nebo také na grafickém systému Quartz (Mac OS X v10.4 a novější).

GTK+ samozřejmě poskytuje velkou sadu prvků, ze kterých programátor může sestavovat uživatelské rozhraní. Tyto prvky mají jednotný vzhled, čímž se sjednocuje i celkový styl aplikací vyvíjených pomocí GTK+. Uživatel si tento vzhled však může sám snadno změnit. Existují sady nastavení, které se snaží emulovat vzhled některých jiných grafických prostředí jako jsou například Qt, Motif nebo Windows.

GNOME

Na knihovně GNOME je postaveno prostředí pracovní plochy pro unixové operační systémy nazvané GNOME (GNU Network Object Model Environment). GNOME lze ale také chápat jako vývojovou platformu. GNOME jako prostředí pracovní plochy obsahuje sadu aplikací, která se snaží vyhovět základním potřebám uživatelů na prostředí, ve kterém pracují. GNOME obsahuje aplikace pro správu souborů, práci s internetem, práci s multimédii nebo práci s dokumenty. Kromě těchto základních aplikací, které jsou distribuovány společně s GNOME lze do tohoto prostředí zařadit i aplikace, které nebyly vyvíjeny přímo pro GNOME, ale jsou napsány v jedné z mutací toolkitu GTK+. Takovým příkladem může být například GIMP nebo Inkscape.

GNOME také přináší některé funkce, které jsou velmi uživatelsky přívětivé. Jsou to funkce jako náhledy multimediálních souborů (obrázků, videí či audio záznamů), připojování vzdálených souborových systémů, změny vzhledu a celých motivů. Dalšími výhodami jsou použití XML souborů pro uchovávání konfigurací či vykreslování SVG formátů.

3.2.3 Qt

Dalším multiplatformním toolkitem pro tvorbu uživatelského rozhraní je projekt s názvem Qt. Qt je vyvíjeno norskou firmou Qt Software, dříve také známou jako Trolltech. Od června roku 2008 je Qt Software zcela ve vlastnictví firmy Nokia. Qt je možné používat v souladu s licencí GPL v2 nebo v3, pokud výsledný produkt bude svobodný, nebo lze použít komerční licenci Qt.

Qt používá C++ s několika nestandardním úpravami. Při překladu se se nejprve tyto speciální úpravy přeloží do standardního kódu jazyka C++ a poté se již postupuje běžným způsobem. MOC (Meta Object Compiler) je nástroj, který vytváří metainformace o třídách použitých v kódu. Tyto metainformace jsou použity pro dosažení funkčnosti, která není v C++ dostupná. Příkladem takové funkčnosti mohou být signály a sloty.

Qt lze použít také společně s několika dalšími programovacími jazyky jako je C#, Java, Pascal, Perl, ADA, Ruby, Python nebo PHP. Tento toolkit je možné využívat na všech větších platformách (X11, Windows, Mac OS a další) a má velkou podporu pro mezinárodní úpravy.

Qt obsahuje velké množství předdefinovaných tříd a funkcí, pomocí kterých lze sestavit uživatelské rozhraní rychle a efektivně. Tomu také pomáhá přehledná dokumentace, která je distribuována s produktem a je dostupná také na internetu (například zde [6]).

Velkou výhodou oproti jiným toolkitům má Qt ve své nadstandardní funkčnosti. Qt obsahuje řadu funkcí, které nesouvisí zcela s uživatelským rozhraním. Qt umí přistupovat k SQL databázím, parsovat XML dokumenty, pracovat po síti, používat OpenGL rozhraní, či jednotně spravovat soubory na všech platformách.

Qt se ze začátku snažilo napodobovat vzhled a chování platformy, na kterou byl směřován výsledný program, avšak to vedlo k určitým nesrovnalostem, kde se nepodařilo přesně zachytit požadované chování. Pozdější verze Qt používají nativní API jednotlivých platforem pro vykreslování jednotlivých prvků, a proto netrpí žádnými podobnými nesrovnalostmi.

KDE

Stejně jako je GNOME prostředí pracovní plochy postavené nad toolkitem GTK+, tak je KDE prostředí pracovní plochy postavené nad toolkitem Qt. Úkolem projektu KDE je připravit prostředí pro uživatele, které splňuje všechny požadavky na denní práci se systémem a zároveň zprostředkovat nástroje a dokumentaci pro vývojáře samostatných aplikací. Projekt KDE zaštiťuje mnoho menších projektů, které staví své programy na KDE. Příkladem takovýchto projektů může být třeba KOffice nebo KDevelop. Licenční politikou Qt, na kterém je KDE postaveno, bylo KDE omezeno na volné operační systémy. Licence pro nové Qt verze 4, jsou však přívětivější a proto lze nyní KDE distribuovat i na jiné platformy jako je Microsoft Windows a Mac OS.

Projekt KDE vznikl roku 1996 na Univerzitě Eberharda Karla v Tübingenu, kde student Matthias Ettrich přemýšlel jak vylepšit prostředí pracovní plochy unixových systémů. U těchto systémů se mu nelíbilo, že každá aplikace vypadala a chovala se jinak. Navrhl vytvoření nejen sady aplikací, ale celé prostředí pracovní plochy, u kterého by mohli uživatelé očekávat konzistenci jak ve vzhledu tak v chování. Dalším jeho požadavkem bylo udělat ovládání jednodušší a intuitivnější, aby ho mohl ovládat i nezkušený uživatel. V roce 1998 vyšla první verze systému KDE. Aktuální verze je KDE 4.1.3.

3.2.4 wxWidgets

Projekt wxWidgets se zrodil na Univerzitě v Edinburgu, kde na něm pracoval Julian Smart. Z počátku byl projekt zaměřen na vytváření aplikací pro platformy Windows a Unix, postupem času se ale projekt rozrostl a přibyla podpora i pro další platformy. wxWidgets poskytuje jednoduché prostředí pro vytváření aplikací s grafickým uživatelským rozhraním pro různé platformy. Při tvorbě aplikací je možné využít GTK+ toolkitu, Motifu, nebo samotného systému X11.

wxWidgets je svobodný software a lze ho použít jak pro vytvoření svobodné aplikace, tak pro komerční aplikaci. Tato licence také zaručuje budoucí vývoj produktu. Vývojem se totiž zabývá skupina zainteresovaných lidí přes internet, kteří budují projekt podle svých požadavků.

wxWidgets obsahuje velké množství tříd, se kterými lze vytvářet podobu výsledné aplikace podle představ programátora. Tyto třídy jsou zdokumentovány a tato dokumentace je dostupná ve formě html stránek či jako pdf.

wxWidgets obsahují funkce a třídy, které umožňují podporu pro spojování událostí s funkcemi, náhledy před tiskem, databáze ODBC, schránku (copy&paste), drag&drop, různé typy konfiguračních souborů, spravování více vláken v aplikaci, síťových operací, OpenGL a další (více v [8]).

3.2.5 FLTK

FLTK je zkratka pro Fast, Light ToolKit a jak již z názvu vyplývá tato sada nástrojů si zakládá na své rychlosti a malé velikosti. FLTK je velice pečlivě organizován a to dává linkeru možnost vynechat vcelku hodně nepoužitého kódu. Hlavní smyčka programů psaných pomocí FLTK je koncipována tak, že není blokující a FLTK program musí žádat o odchytávání událostí.

FLTK toolkit neobsahuje předdefinované celé dialogy, oproti například Qt, kde jsou některé standardní dialogy již připraveny k použití. FLTK za to ovšem poskytuje velice efektivní a přímé rozhraní s OpenGL a obsahuje knihovnu pro kompatibilní přístup ke knihovně GLUT.

FLTK má oproti ostatním toolkitům jedinečný přístup k vlastnostem jednotlivých objektů. Zatímco u ostatních knihoven se pro získání a nastavení určitého atributu používají dvě různé funkce, ve FLTK se používá funkce pouze jedna a její význam, tedy to zda určitý parametr nastavuje nebo pouze čte, se odvodí od kontextu ve kterém se tato funkce nachází. Tento přístup je sice jednodušší na psaní, ale kód se poté stává méně čitelný.

FLTK, jako ostatní sady nástrojů, má vlastní designový editor, který obsahuje všechny třídy předdefinované touto sadou nástrojů. FLTK je napsaný v jazyce C++, ale lze jej použít i ve spojení s jinými jazyky jako Perl nebo Python (více v [4]).

Kapitola 4

GUI pro antivirus

Jedním z výsledků mé práce má být grafické uživatelské rozhraní pro již funkčního antivirového démona od firmy ZONER software, a.s. ve dvou nezávislých knihovnách v systému GNU/Linux.

4.1 Zoner AntiVirus

Současné řešení antiviru se skládá z démona (`zavd`), který běží na pozadí systému a čeká na dotazy od klientské části programu (`zavcli`). Pokud z klientské části programu přijde požadavek na oskenování souboru, zkontroluje démon stav souboru a vrátí klientské části informace, které o daném souboru zjistil. Klientská část programu je dosud implementována pouze v příkazové řádce.

Pro jednoduché oskenování souboru stačí zavolat `zavcli` a jako parametr předat jméno souboru, který má být oskenován. Soubory pro oskenování lze zadávat jednotlivě oddělené mezerou, ale lze zadávat také celé adresáře. Klient všechny soubory postupně oskenuje a vypíše výsledky na standardní výstup.

Použití `zavcli` lze ovšem vhodně nastavit podle přání uživatele. Existují parametry tohoto programu, pomocí kterých lze nastavit skenovací proces. Parametry, které lze nastavit zahrnují například: typ skenování, hloubku zanoření do archivů, velikost skenovaných souborů, typ vypisovaných informací a další. Z důvodu neustálého vývoje tohoto softwaru, není možné uvést všechny aktuální možnosti.

4.1.1 Požadavky na GUI

Grafické uživatelské rozhraní pro skenování souborů pomocí Zoner AntiVirus by mělo být funkčně velice podobné klientovi v příkazové řádce. Je nutné připravit uživatelské rozhraní, které intuitivně a jednoduše umožní práci se skenovanými soubory. Musí existovat způsob jak jednoduše vybrat soubory, které se mají skenovat a dále pak způsob, jak zobrazovat soubory čekající na oskenování a soubory již oskenované s jejich příslušným výsledkem.

Dalším velmi důležitým prvkem je možnost nastavení skenovacího procesu podobně jako u verze v příkazové řádce. Oproti klientovi v příkazové řádce by grafické uživatelské rozhraní mělo obsahovat funkce a ovládací prvky potřebné k pozastavení a opětovnému spuštění skenovacího procesu. Proto, aby bylo možné vůbec soubory oskenovat je nutné aby běžel antivirový démon (`zavd`). V příkazové řádce existuje zvláštní příkaz (který není součástí zavcli) pro práci s démonem, ovšem aby nebylo nutné opouštět grafické rozhraní je

nutné přidat i možnost ovládat i tohoto démona (spouštět, vypínat a aktualizovat) přímo přes grafické rozhraní.

Jako rozšíření je také vhodné implementovat statistiky skenovacího procesu a určitý prvek pro zobrazování textových informací a pro indikaci právě probíhajícího skenovacího procesu.

4.2 Návrh GUI

Výsledný vzhled klienta s grafickým uživatelským rozhraním reprezentuje návrh, který lze vidět na obrázku 4.2. Na tomto obrázku jsou jednotlivé hlavní prvky uživatelského rozhraní označeny římskými číslicemi a jejich význam je následující:

I. Hlavní menu aplikace. Z tohoto menu je možné přistupovat na veškeré funkce, které aplikace má.

II. Panel nástrojů, ze kterého jsou přístupné nejpoužívanější funkce aplikace pro snadnější přístup k nim

III. Statistiky skenovacího procesu. Tento prvek bude ukazovat aktuální skenovaný soubor, stejně tak jako počet již skenovaných souborů a počet infikovaných souborů a další podobné statistiky.

IV. Indikátor běhu skenovacího procesu. Prvek obsahuje animaci, která se spustí zároveň se spuštěním skenovacího procesu.

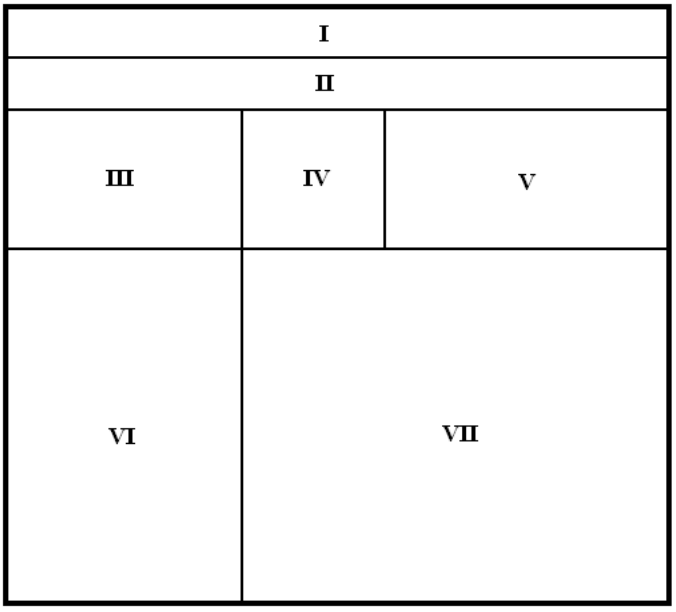
V. Textový výstup aplikace. U tohoto textového prvku je možné jeho obsah kopírovat, vytisknout, vyčistit, či uložit do souboru. Pro snadnější přístup k těmto funkcím je u tohoto prvku vytvořeno kontextové menu, které se zobrazí po stisknutí pravého tlačítka.

VI. Prvek pro procházení souborů na disku. Prvek implementuje možnost označit soubory nebo celé adresáře, které se mají skenovat.

VII. Seznam souborů, které mají být skenovány, nebo které již oskenovány byly, v tom případě se u nich zobrazí výsledek skenování. S tímto seznamem je také možné provádět řadu operací: smazat seznam, uložit seznam do souboru, načíst seznam ze souboru, vytisknout seznam a filtrovat zobrazené položky.

Z menu je přístupný dialog, pomocí kterého lze nastavit možnosti pro skenovací proces. Rozsah těchto možností je velmi podobný tomu, který má klient v příkazové řádce.

Soubory, které jsou přidávány pomocí prvku na procházení souborů jsou řazeny do fronty. Po spuštění skenovacího procesu se spustí druhé vlákno, které vybírá postupně soubory z fronty. U každého souboru se spojí s démonem antivirového softwaru, předá jméno a adresu skenovaného souboru společně s nastavením skenovacího procesu a počká na odpověď. Poté zapíše příchozí výsledek k příslušnému souboru do fronty a postoupí k dalšímu souboru. Tento proces je v samostatném vlákně, aby nezablokoval grafické rozhraní, což by bylo nežádoucí.



Obrázek 4.1: Návrh grafického uživatelského rozhraní pro klientskou aplikaci k antiviru

4.3 Výběr knihoven

Pro aplikaci s grafickým uživatelským rozhraním postavenou pro již existujícího antivirového démona jsem vybral knihovny GTK+/GNOME a Qt/KDE. Jelikož Zoner AntiVirus je komerční aplikace a grafické rozhraní má zvýšit jeho použitelnost, pak je jen logické vybrat toolkity, na kterých stojí v dnešní době dvě, mezi uživateli unixových systémů nejrozšířenější, prostředí s pracovní plochou a to GNOME a KDE. Použitím GTK+/GNOME a Qt/KDE se dosáhne v těchto dvou prostředích nativních vzhledů, které jsou konzistentní jak ve vzhledu tak v chování s ostatními aplikacemi z těchto prostředí.

Obě tyto knihovny mají dobrou podporu v různých distribucích operačního systému GNU/Linux a existují u nich přehledné dokumentace. U knihovny Qt se uvádí jako nevýhoda její duální licence, ze kterou se musí platit při jejím použití pro komerční účely. Avšak aplikace s grafickým rozhraním se bude distribuovat jako volný software a pouze pro operační systém GNU/Linux a tím pádem lze použít licence bez poplatku.

4.4 Implementace GUI

V této části dokumentu rozepíšu postupy používané při tvorbě grafického uživatelského rozhraní pro antivirus. Uvedu zde také rozdíly v obou vybraných knihovnách.

4.4.1 Vývojové prostředí

Vývojové prostředí je důležitou součástí vývojového procesu aplikace. Výběrem správného vývojového prostředí si lze velice usnadnit a zpříjemnit vývoj jakékoli aplikace.

Pro vývoj aplikací v knihovnách GTK+/GNOME je v současné době nejlepší volbou vývojové prostředí *Anjuta*. U knihoven Qt/KDE je v současnosti jasná volba *KDevelop*.

Tyto produkty, ač každý zaměřený na vývoj aplikací v jiných knihovnách, jsou si velmi podobné. Oba totiž poskytují:

- Rozšiřitelnost pomocí pluginů,
- šablony projektů,
- správce projektů,
- editor zdrojových kódu se zvýrazněním syntaxe a dalšími funkcemi,
- integrovaný editor grafického rozhraní,
- integrovaný systém nápovědy a další.

Díky rozšiřitelnosti obou produktů přes systém pluginů, které může napsat v podstatě kdokoli, je skoro nemožné vyjmenovat veškerou funkčnost obou produktů. Moje zkušenost osobně je ovšem taková, že projekt KDevelop je vyspělejší a lépe se s ním zvládá vývoj celého projektu. Nabízí více možností pro nastavení projektu přesně podle přání vývojáře. Celkový vzhled a nastavení grafického rozhraní, nebo spíše jeho jednotlivých částí, je jednodušší a rozmístění jednotlivých prvků se zachová i po restartování aplikace.

4.4.2 Hlavní smyčka

Hlavní smyčka programu s grafickým uživatelským rozhraním představuje hlavní prvek pro komunikaci mezi uživatelem a programem. Uživatel zadává příkazy většinou pomocí ukazovacího zařízení jako jsou například myš nebo klávesnice. Jednotlivá stlačení kláves či pohnutí ukazovacím zařízením generuje v systému různé události, které jsou předány aplikaci, kde se podle potřeby řádně ošetří nebo ignorují, pokud na ně není potřeba reagovat. O naslouchání a přijímání takových zpráv se stará právě hlavní smyčka programu. Programátor tak může odchytávat jednotlivé události a ve svém programu je odpovídajícím způsobem zužitkovat.

GTK+/GNOME

V knihovnách GTK+/GNOME se hlavní smyčka spouští pomocí funkce `gtk_main()`. Hlavní smyčka programu čeká na vstupní události a tyto události poté přepošle jednotlivým prvkům grafického uživatelského rozhraní, kterých se daná událost týká. Tyto prvky dále zpracují vstupní události a podle jejich typu vygenerují určité signály. Každý prvek má své specifické signály. Tyto signály jsou již srozumitelné a přístupné pro programátora, který může spojit tyto signály s funkcemi, které se mají provést při každém vygenerovaní takového signálu. Spojení signálu s funkcí lze provést pomocí `g_signal_connect()`. Funkce, které reagují na vygenerované signály se často nazývají zpětná volání. Po spuštění hlavní smyčky programu se chování aplikace řídí výhradně reakcemi na jednotlivé události generované uživatelem, proto se hlavní smyčka spouští až úplně na konci vstupní funkce programu `main()`, po vykreslení veškerých prvků grafického uživatelského rozhraní.

Qt/KDE

U knihoven Qt/KDE je princip hlavní smyčky, generovaných signálů a na ně reagujících zpětných volání v podstatě totožný. Veškerou správu událostí zajišťuje objekt *QApplication*, který kromě tohoto také:

- Inicializuje aplikaci podle globálního nastavení systému (barevná paleta, font, atd.) a nadále udržuje informace o tomto nastavení pro případ změny například přes kontrolní panel,
- zpracovává vstupní parametry a nastavuje podle toho vnitřní stav aplikace,
- zajišťuje překlad řetězců použitých v aplikaci, i když se tato funkce v praxi nepoužívá přímo, ale přes `QObject::tr()`,
- uchovává objekt pro komunikaci se schránkou,
- udržuje informace o oknech aplikace, dokonce i jednotlivých prvcích grafického rozhraní,
- obstarává aktuální vzhled kurzoru ukazovacího zařízení,
- poskytuje sofistikovanou podporu pro správu sezení.

4.4.3 Hlavní menu

V programech s grafickým uživatelským rozhraním je jednou z nejdůležitějších součástí hlavní menu. Je to jeden z ustálených prvků grafického prostředí. Již v počátcích se využívalo hlavní menu jako nástroj pro přístup k funkcím programu. Drtivá většina programů s grafickým uživatelským rozhraním má k dispizici hlavní menu.

Hlavní menu prezentuje úplnou skupinu funkcí, které daná aplikace může vykonávat. Dále se v hlavním menu mohou objevit některá nastavení, která se často mění a není vhodné nutit uživatele otevřít jiné okno pokaždé, když chce toto nastavení upravit.

Ve většině aplikací se řádkové menu nachází pouze v hlavním okně. Ostatní vedlejší okna aplikace by měla být natolik jednoduchá, aby se jejich funkce daly poskytovat pomocí tlačítek a jiných běžných prvků grafického rozhraní, které jsou umístěny uvnitř tohoto okna. V určitých případech lze řádkové menu umístit i do vedlejších oken, v tomto případě by ovšem toto okno nemělo obsahovat tlačítka, která ho zavírají, jako *OK*, *Zavřít* nebo *Zrušit*, taková funkce by měla být obsažena v řádkovém menu v části *Soubor*.

Jednotlivé položky v menu by měly být vyjádřeny jasně a krátce. Pro položky vyjadřující určitou funkci programu se používají slovesa, oproti tomu pro položky jejichž funkce je měnit nastavení programu by měly být přídavná jména. Položky menu, které právě nejsou dostupné, nebo by v danou chvíli neměly žádný význam v dané aplikaci, by měly být neaktivní. Příkladem takové situace může být položka *Kopírovat*, pokud není označen žádný text. Řádkové menu by nemělo obsahovat menu, které má méně než tři položky, výjimkou je ovšem menu *Nápověda*. Pokud se při tvorbě softwaru programátor dostane do takovéto situace měl by tyto položky přesunout o úroveň výše. Pokud jsou tyto položky již v nejvyšším menu, pak je vhodné tyto položky přesunout do jiného menu.

Pokud uživatel klikne v řádkovém menu otevře se rozevírací menu. V tomto menu jsou umístěny položky, které jsou spojeny s názvem tohoto menu. Počet položek v žádném z těchto menu by neměl překročit 15. V takovém případě by se některé položky měly buďto přesunout do jiného menu, nebo je také možné pro skupinu položek s podobnou funkcí vytvořit podmenu, do kterého se tyto položky umístí. Během chodu aplikace by se z menu neměly ztrácet položky ani by se neměly nové objevovat. Toto chování by bylo pro uživatele velice matoucí a těžko by získal přehled o celkové funkčnosti programu.

Podmenu je menu, které se objeví při kliknutí na jeho titulek, nebo při posunutí kurzoru na tento titulek, přičemž se otvírané menu objeví s malým zpožděním, aby nedocházelo

k nechtěnému otevírání při pouhém přejíždění kurzorem. Podmenu by ovšem mělo být používáno s rozvahou. Jeho položky jsou hůře zapamatovatelné a nepřispívají tak k lehké orientaci v hierarchii menu. Pro lidi s tělesným postižením může být také velice obtížné navigovat ukazovací zařízení pro vybrání položek z podmenu. Obecně všechna menu v aplikacích by neměla mít více než dvě úrovně, jinak jsou těžko zapamatovatelná a špatně se v nich naviguje.

Většina aplikací má mnoho funkcí společných s ostatními již používanými aplikacemi, například *Kopírovat*, *Otevřít* nebo *Ukončit*. Pro lehčí zapamatování mají tyto menu a jejich položky v každé aplikaci stejné popisky a jsou umisťovány na stejná místa. Jejich chování je také udržováno konzistentní mezi jednotlivými aplikacemi. Standardní položky menu se neobjevují všechny v každé aplikaci, ale pokud daná aplikace obsahuje položku nebo celé menu, které je používáno jinde a uživatelé jsou zvyklí jej používat z jiné aplikace, pak by tato položka nebo toto menu mělo být umístěno v každé nové aplikaci na stejné místo, se stejným názvem a stejnou funkcí.

Jedním z takových standardních menu je například *Soubor*. Toto menu obsahuje funkce, které operují nad aktuálním dokumentem. Toto menu se umisťuje úplně vlevo na řádkovém menu, protože je důležité a je velmi často využíváno. Postupem času, protože většina aplikací již měla takové menu a protože rozdíl mezi zavřením dokumentu a zavřením aplikace se překrýval, se do menu *Soubor* přidala i položka *Ukončit*, která se nachází na konci tohoto menu. Pokud aplikace nepracuje s dokumenty, pak se toto menu přejmenovává podle typu objektu, který reprezentuje. Například ve hrách je menu *Soubor* přejmenováno na *Hra*. Avšak položka *Ukončit* se umisťuje stále jako poslední.

Dalším standardním menu jsou *Úpravy*. Toto menu obsahuje položky spojené s úpravou dokumentu (ovládání schránky, hledání a nahrazovaní v dokumentu atd.) a změnou nastavení. Nastavení lze umístit i do vlastního menu, pokud by ovšem v tomto menu byla pouze jedna položka otevírající dialog s nastavením, pak je samostatné menu neefektivní a doporučuje se umístit tuto položku právě do menu *Úpravy*.

Menu *Zobrazit* se nacházejí pouze položky spojené se prohlížením aktuálního dokumentu. Do tohoto menu se neumisťují položky, které jakýmkoli způsobem mění obsah aktuálního dokumentu. Jedinou výjimku tvoří položka *Obnovit*, která může měnit obsah dokumentu.

Formát je další standardní menu, ve kterém lze najít příkazy, které mění vizuální stránku aktuálního dokumentu, například změnou fontu, barvy atd. Rozdíl mezi funkcemi u tohoto menu a menu *Zobrazit* je v tom, že funkce z menu *Formát* mění trvale aktuální dokument a jsou uloženy jako součást tohoto dokumentu.

Menu *Okno* zahrnuje položky, které ovládají všechna okna patřící aplikaci. Toto menu lze využít pouze u aplikací, které mohou mít otevřeno více oken najednou, ovšem tato okna jsou přímo závislá na rodičovském okně a mohou se posouvat pouze po jeho ploše.

Menu, které je umisťováno v aplikacích a jejich řádkových menu úplně vpravo se nazývá *Nápověda*. Toto menu obsahuje položky, které by měly uživateli pomoci, pokud si neví při práci s programem rady. Dávají se sem položky odkazující na dokumentaci aplikace a také položka *O programu*, kde je uveden název programu, jeho verze, stručný popis, jména autorů atd.

GTK+/GNOME

Přestože v knihovnách GTK+/GNOME lze využít stejného způsobu tvorby celé struktury řádkového menu jako v knihovnách Qt/KDE (tento způsob je popsán později v této práci),

rozhodl jsem se pro variantu vytvoření této struktury ručně, z důvodu porovnání výhod a nevýhod obou přístupů.

Ručně vytvořené řádkové menu zahrnuje vložení odpovídajícího prvku grafického uživatelského rozhraní na správné místo v hlavním okně aplikace a přidávání jednotlivých položek menu se správným nastavením. U každé položky je třeba nastavit zpětné volání, které zařídí správné chování této položky v hlavním menu. Naštěstí pro vývojáře GTK+/GNOME aplikací existuje v GUI editoru *Glade* dialog, ve kterém lze menu a jeho položky velmi rychle přidávat, měnit a odebírat. Pomocí této funkce editoru *Glade* si lze výrazně zjednodušit nastavení menu podle představ tvůrce. V grafickém rozhraní si programátor akorát vybere jaký typ položky chce, zda chce u této položky obrázek, pokud ano, tak si snadno vybere z nabídnutých možností, přiřadí položce klávesovou zkratku a zpětné volání, které ošetří chování položky.

Tím že jsou všechny položky přidávány ručně je zaručeno, že všechny položky budou na pozicích které si programátor sám určil. Žádné položky nejsou do menu přidávány automaticky jako je tomu u přístupu, který jsem zvolil v knihovnách Qt/KDE. Nevýhodou ruční tvorby menu je nutnost starat se o každou položku z menu zvlášť a později v této práci ještě uvedu výhody spojené s panelem nástrojů, při použití vygenerování řádkového menu.

Hlavní řádkové menu v mé aplikaci grafickým uživatelským rozhraním pro antivirus osahuje tyto menu: *Skener*, *Fronta* a *Nápověda*. Menu *Skener* zastává funkci standardního menu *Soubor*, jelikož antivirový program nepracuje s dokumentem, ale se skenerem. Toto menu zahrnuje standardní položku pro ukončení aplikace (*Ukončit*) a mimo to položky pro spuštění skenovacího procesu (*Spustit Sken*), jeho pozastavení (*Pozastavit Sken*) a jeho úplné zastavení (*Zastavit Sken*). Do tohoto menu jsem byl také donucen umístit položku pro nastavení aplikace, jelikož samostatné menu *Nastavení* s jednou položkou je neefektivní a stejně tak tato aplikace nemá menu *Úpravy*. Do menu *Fronta* jsem umístil položky s funkcemi prováděné na aktuální frontě souborů pro skenování. Položky v tomto menu jsou následující: *Resetovat Frontu* pro vyresetování stavu všech položek ve frontě, *Vyčistit Frontu* pro vyjmutí všech souborů z fronty pro skenování, *Načíst Frontu* pro načtení dříve uložené fronty ze souboru na disku, *Uložit Frontu* pro uložení aktuální fronty souborů na disk pro pozdější znovunačtení a poslední položkou je *Smazat Soubory* pro fyzické vymazaní ve frontě označených souborů z disku. Posledním menu je samozřejmě *Nápověda*, které obsahuje pouze položku *O programu*.

Qt/KDE

Druhým možným způsobem jak vytvořit menu je použití akcí. Akce je objekt, který definuje abstraktní reprezentaci položek v menu, a jak později v této práci ukážu, i v panelu nástrojů.

V mnoha aplikacích je implementováno více než jedno menu a položky v těchto menu se opakují. Při ručním vytvářením menu je nutné vytvořit každé menu zvlášť a nastavit v nich každou položku znovu. Pak také může nastat situace, kdy je potřeba některou položku změnit na neaktivní nebo naopak a u ručně vytvářeného menu se musí položky ve všech menu zvlášť upravit. To vede k znepřehledňování kódu a také se může stát, že se na některou položku zapomene. Toto řeší právě objekt akce. Každá akce reprezentuje jednu funkci programu. U akce se nastaví odpovídající titulek, popis, klávesová zkratka a typ akce. Existují tři základní typy akcí:

- Standardní akce spustí určitou funkci programu,

- přepínací akce vybírá mezi dvěma stavy určitého nastavení aplikace,
- výběrová akce je jednou z více podobných akcí, které společně tvoří možnosti pro nastavení aplikace.

Tyto objekty akcí se přidávají do jednotlivých menu a pokud má dojít ke změně u položky v menu, změní se vlastnost akce a tato změna se projeví všude, kde je příslušná akce použita, tj. ve všech menu a panelech nástrojů. Nevýhodou ovšem může být to, že neexistuje žádný nástroj pro vytvoření akcí přes grafické rozhraní jako tomu je u ručního vytváření menu. Akce se musí vytvořit a nastavit přímo v kódu programu.

Při použití akcí se naskytují dvě možnosti jak vytvořit řádkové menu a jak později ukážu i panel nástrojů. Je možné vytvořit prvek menu v editoru s grafickým rozhraním a přidávat do něj místo položek přímo akce definované v kódu. Další možností je nechat si celou strukturu vygenerovat podle XML kostry, která je vytvořena v externím souboru. Do tohoto souboru jsou v požadované struktuře vypsána jména akcí, které se mají umístit v menu. Soubor je v operačním systému uložen na místo, ze kterého je možné přečíst jeho specifikaci. Při spuštění dané aplikace se struktura menu vygeneruje právě podle tohoto souboru. Při použití tohoto způsobu se dají přidávat nejenom tvůrcem definované akce, ale také standardní akce definované v knihovnách KDE a GNOME. Lze tak například přidat celé menu *Nastavení*, do kterého se automaticky vygenerují položky jako nastavení klávesových zkratek nebo nastavení pro panely nástrojů. Nevýhodou ovšem je, že vygenerované položky se umístí do menu vždy nakonec a tak programátor nemá plnou kontrolu nad rozložením menu. Ovšem toto rozložení bývá většinou velice rozumné a proto není třeba do něj zasahovat. Také se může stát, že se ve vygenerovaném menu objeví položka, kterou daná aplikace nevyužije. V systému KDE tento problém není příliš vyřešen a neexistuje čistý způsob jak se takovýchto položek z menu zbavit.

Zbůsob s generováním řádkového menu z XML souboru jsem zvolil i pro svoje grafické rozhraní pro antivirus v knihovnách Qt/KDE. Stejně jako u knihoven GTK+/GNOME se v řádkovém menu objeví menu *Skener* a *Fronta*, ovšem z menu *Skener* je odebrána položka pro nastavení aplikace, jelikož ta se přesunula do vlastního menu *Nastavení*. V tomto novém menu se kromě nastavení samotné aplikace vygenerovaly položky pro skrytí a zobrazení panelu nástrojů, skrytí a zobrazení stavového řádku, nastavení panelu nástrojů (typ zobrazování, výběr položek na něm zobrazených) a nastavení klávesových zkratek pro tuto aplikaci. Do menu *Nápověda* také přibyly některé vygenerované položky, ovšem většina z nich je bohužel nefunkčních.

4.4.4 Panel nástrojů

Panel nástrojů je řada ovládacích prvků, které umožňují snadný přístup k často používaným funkcím programu. Většina aplikací obsahuje v panelu nástrojů jednoduchá tlačítka. Ve složitějších aplikacích však lze nalézt i komplexnější ovládací prvky jako rozevírací seznamy. V panelu nástrojů lze zobrazovat buď pouze ikonu nebo ikonu s textem. Pokud je obsazení panelu nástrojů vybráno s rozvahou, pak výrazně ulehčuje uživateli používání celé aplikace a jejích funkcí, které by jinak byly schovány ve struktuře hlavního menu. Je ale důležité, aby v panelu nástrojů byly pouze ty nejdůležitější a nejpoužívanější funkce jinak panel ztrácí na přehlednosti a tím pádem také na efektivitě. Hledat funkci na panelu nástrojů, kde je příliš mnoho položek se stává obtížnější a panely nástrojů, které mají více než jeden řádek zabírají místo jiným prvkům aplikace.

Efektivita panelu nástrojů se zvyšuje při dodržení konzistence s ostaními již vyvinutými a používanými aplikacemi. Panel nástrojů je první věc, které si uživatel všimne při prvním spuštění aplikace. Panel nástrojů, který obsahuje známé položky, které se používají obecně ve více aplikacích, uživateli připadá přívětivější a cítí se pohodlněji při používání této, pro něj nové, aplikace.

Aplikace může obsahovat více panelů nástrojů pro různé skupiny funkcí, které daná aplikace může provádět. Ať již aplikace obsahuje jakékoli množství panelů nástrojů je vhodné vytvořit jeden hlavní, obsahující podmnožinu funkcí, které definují hlavní funkčnost dané aplikace. Například hlavní panel nástrojů pro jakýkoli textový editor bude obsahovat položky *Nový*, *Otevřít* a *Uložit jako...*. Podobně pak v jakémkoli prohlížeči lze na panelu nástrojů najít položky *Zpět*, *Vpřed*, *Zastavit* a *Aktualizovat* v tomto pořadí.

Všechny funkce, které jsou vyneseny na panely nástrojů musejí být přístupné také z hlavního menu buďto přímo, nebo přes další dialog vyvolaný z menu. Rozmístění položek v panelu nástrojů by mělo být ve stejném pořadí a rozdělení do skupin jako je tomu v hlavním řádkovém menu. Konkrétně je vždy třeba umístit do zvláštní skupiny oddělené dělícími liniemi ty položky, které se navzájem vylučují. Na panel nástrojů by se neměly přidávat položky jako *Nápověda*, *Zavřít* nebo *Ukončit*, jelikož tyto položky jsou málo používány a zabíraly by zbytečně místo, na které je možné umístit jiné ovládací prvky.

Existuje také možnost umístit panel nástrojů vertikálně, což se ovšem obecně nedoporučuje. Lidské oko nemá tak dobré vertikální zpracování jako horizontální. Při vertikálním rozložení není skupina vylučujících se položek tak výrazná jako při horizontální variantě. Taktéž panel nástrojů zobrazující jak ikonu tak text je na pohled nepřirozený a méně prostorově efektivní. Některé ovládací prvky jako rozbalovací seznamy ani nedokáží správně fungovat při umístění na vertikální panel nástrojů. Ovšem i vertikální panely nástrojů mohou najít své uplatnění, pokud:

- Rozložení aplikace vede k tomu, že by při použití horizontálního panelu nástrojů bylo vynecháno po straně zbytečné místo, ve kterém by mohl být vertikální panel nástrojů a tím ušetřit místo v horní nebo spodní části aplikace,
- v panelu nástrojů je tolik položek, že by zabíral tři řádky pod hlavním řádkovým menu(v takovém případě je ale také potřeba zvážit možnost odebrání některých položek z panelu nástrojů).

GTK+/GNOME

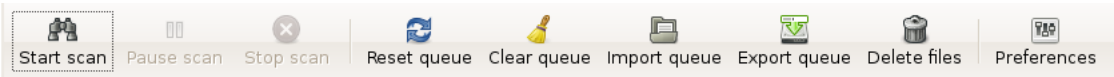
Tvoření hlavního řádkového menu a panelů nástrojů spolu úzce souvisí, proto jsem zachoval stejný způsob rodělení vytváření panelu nástrojů jak pro knihovny GTK+/GNOME, tak i pro knihovny Qt/KDE. Panel nástrojů v knihovnách GTK+/GNOME jsem tedy vytvořil ručně v grafickém editoru *Glade*. Stejně jako u řádkového menu totiž v tomto grafickém editoru existuje dialog, ve kterém lze panely nástrojů snadno naplnit položkami a nakonfigurovat je podle vlastních představ. U každé položky lze opět nastavit zobrazovaný text, klávesovou zkratku a zpětné volání pro ošetření funkčnosti položky. Při ruční tvorbě panelu nástrojů je také nutné starat se zlášť o každou položku při jakékoli její změně, stejně jako je tomu u řádkového menu.

Pro aplikaci komunikující s antivirovým démonem jsem zvolil pouze jeden panel nástrojů, který bohatě postačí pro pojmutí veškerých potřebných položek pro tuto aplikaci. Tento

panel je umístěn horizontálně hned pod hlavním řádkovým menu, tak jak je doporučeno pro většinu aplikací. Tento panel nástrojů obsahuje následující položky:

- *Spustit Sken*,
- *Pozastavit Sken*,
- *Zastavit Sken*,
- *Resetovat Frontu*,
- *Vyčistit Frontu*,
- *Načíst Frontu*,
- *Uložit Frontu*,
- *Smazat Soubory*,
- *Nastavení*.

Panel nástrojů v prostředí pracovní plochy GNOME lze nastavit přes globální nastavení vzhledu a upravovat tak, zda je zobrazen pouze text, ikona nebo obojí najednou. Pokud je nastaveno obojí najednou, lze také určit, zda bude zobrazovaný text pod ikonou nebo vedle ní.

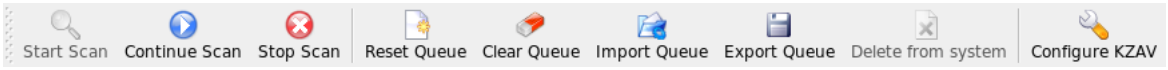


Obrázek 4.2: Panel nástrojů v GTK+/GNOME

Qt/KDE

Stejné položky jsou v základní podobě panelu nástrojů použity i v knihovnách Qt/KDE. Zde jsou ovšem tyto položky přidávány přes akce předem definované a použité již při vytváření hlavního řádkového menu. Celý panel nástrojů je vygenerován ze souboru ve formátu XML, ve kterém je zapsána společná definice řádkového menu a panelů nástrojů. Rozdíl mezi generovaným panelem nástrojů a řádkovým menu je v tom, že do panelu nástrojů se nepřidávají žádné automaticky vygenerované položky. U každého takto vygenerovaného panelu nástrojů lze ovšem nastavit:

- Pozici panelu,
- zda je panel ukotven či volně pohyblivý,
- zda obsahuje pouze text, pouze ikony nebo obojí a v tomto případě i jejich vzájemnou pozici,
- zda je viditelný nebo schovaný,
- položky, které jsou na panelu zobrazeny.



Obrázek 4.3: Panel nástrojů v Qt/KDE

4.4.5 Stavový řádek

Stavový řádek je prostor ve spodní části okna, který lze využít pro sdělování krátkých informací o aktuálním stavu aplikace. Stavový řádek se umisťuje vždy do spodní části hlavního okna. Není vhodné stavový řádek umístit do dialogových oken. Stavový řádek by měl sloužit pouze pro sdělování informací, které nejsou kritické pro běh celé aplikace. Informace, které lze sdělit pomocí stavového řádku jsou například tyto:

- Obecné informace o aplikaci nebo aktuálním dokumentu, například současný stav připojení k síti v aplikaci, která komunikuje se vzdálenými počítači, nebo velikost aktuálního dokumentu v textovém editoru,
- informace o právě probíhající operaci, například *Stikněte Shift pro rozšíření vybrané oblasti* v aplikaci, která se zabývá kreslením,
- informace o operaci, která probíhá na pozadí, například *Odesílání na tiskárnu*, *Tisk stránky 3 z 5* nebo *Tisk dokončen*,
- popis ovládacího prvku nebo oblasti, která se nachází aktuálně přímo pod kurzorem.

Stavový řádek bývá většinou uživatelem vnímán pouze periferně nebo jej lze vypnout úplně, je tedy možné že uživatel nezachytí informaci, která je mu podávána pouze přes stavový řádek.

Pokud v aplikaci zrovna není adekvátní informace k zobrazení ve stavovém řádku je lepší jej nechat prázdný, než ho vyplnit něčím jako je například *Připraven*. Pokud bude stavový řádek prázdný a přijde nová informace, uživatel si jí spíše všimne díky větší změně.

Na stavový řádek lze také umístit aktivní ovládací prvky jako tlačítko nebo indikátor průběhu. Zde je ovšem nutné zajistit, aby veškerá funkčnost poskytnutá ve stavovém řádku byla rovněž dostupná přes hlavní řádkové menu.

GTK+/GNOME

Stavový řádek v knihovnách GTK+/GNOME je poskytován objektem *GtkStatusbar*. Tento objekt udržuje zásobník zpráv pro zobrazení ve stavovém řádku. Zpráva, která je aktuálně na vrcholu tohoto zásobníku je ta, která je právě zobrazena. Každá zpráva, která má být přidána do zásobníku zpráv musí mít přiřazený *context_id*. Toto identifikační číslo kontextu v podstatě určuje zdroj, ze kterého byla zpráva přidána. Pro odstranění zprávy ze zásobníku se používá funkce `gtk_statusbar_pop()`, které je nutné předat jako jeden z parametrů právě *context_id*, tak se odstraní zpráva s daným *context_id*, která se aktuálně nachází v zásobníku nejvýše. Lze tedy říci, že *GtkStatusbar* spravuje zásobník zpráv pro zobrazení, přičemž je možné že tento zásobník obsahuje několik podzásobníků, které se rozlišují svými identifikačními čísly kontextu. Při zobrazování zpráv ovšem tento *context_id* nehraje roli a ve stavovém řádku se zobrazí zpráva, která je na vrcholu zásobníku.

Stavový řádek v knihovnách GTK+/GNOME zobrazuje vždy to, co je na vrcholu zásobníku. Pokud tedy ve stavovém řádku nemá být zobrazen žádný text, musí tento zásobník

být vyprázdněn. Pokud by měla informace ve stavovém řádku zůstat pouze po určitou dobu, je nutné využít funkce `g_timeout_add()`, která zavolá po určitém časovém intervalu jinou funkci, ve které se odstraní aktuálně zobrazovaná zpráva ze zásobníku.

Qt/KDE

V knihovnách Qt/KDE je stavový řádek realizován objektem *QStatusBar*. Na rozdíl od implementace v knihovnách GTK+/GNOME objekt *QStatusBar* neposkytuje zásobník pro zobrazované zprávy. Zprávu do stavového řádku lze vložit třemi způsoby:

- Dočasná zpráva - při vytváření této zprávy se zadává, jako jeden z parametrů, jak dlouho má daná zpráva zůstat viditelná na stavovém řádku. Zpráva se také zruší pokud se zavolá metoda `clear()`, která smaže obsah stavového řádku, nebo pokud je vytvořena jiná zpráva, která tuto dočasnou zprávu přepíše.
- Normální zpráva - tento druh zprávy je v podstatě totožný s předešlým typem, jediný rozdíl je v tom, že tato zpráva nemá omezené trvání. Tuto zprávu lze tedy vymazat pouze metodou `clear()` nebo přepsat jinou zprávou.
- Trvalá zpráva - název zpráva není u tohoto typu tak úplně přesný termín. Tento typ vyjadřuje určitý ovládací prvek, který je přidán na stavový řádek a zůstává tam po celou dobu běhu aplikace. Takové prvky mohou být například tlačítka.

4.4.6 Statistiky

Součástí uživatelského rozhraní pro funkční jádro antivirového systému jsou také statistiky, které uživateli sdělují statistické informace o skenovaných souborech. Data zobrazovaná ve statistické části uživatelského rozhraní lze rozdělit na dvě hlavní kategorie, které jsou dále rozvětveny:

- Informace o právě probíhajícím nebo posledním skenování:
  - úplná cesta k právě skenovanému souboru,
  - uplynulý čas skenování,
  - počet oskenovaných souborů,
  - počet infikovaných souborů,
  - počet pravděpodobně infikovaných souborů,
  - počet podezřelých souborů,
  - počet nestandardních souborů;
- informace získané od antivirového jádra:
  - verze antivirového démona,
  - verze jádra antivirového systému,
  - verze antivirové databáze,
  - datum poslední aktualizace databáze.

Informace o skenovacím procesu se mění po dokončení oskenování každého souboru, zatímco informace o antivirovém jádře se mění pouze při spuštění aplikace nebo při spuštění skenování.

V poli pro zobrazení úplné cesty k právě skenovanému souboru může dojít k tomu, že řetězec s cestou bude delší než vyhrazené místo pro toto pole, pak se tato cesta zleva a pokud je to nutné i zprava zkrátí tak, aby tento řetězec nepřekračoval vyhrazené místo.

Při načítání informací z antivirového jádra se zjistí jak je stará aktuální verze databáze a při překročení určitého stanoveného prahu se na samostané okno se zprávou vypíše upozornění, které doporučuje aktualizovat databázi.

GTK+/GNOME

Veškeré vypsané statistické údaje jsou v knihovnách GTK+/GNOME zobrazeny každý zvlášť pomocí objektu *GtkLabel*. Pro zobrazení data poslední aktualizace ve spravné lokalizované formě jsem použil funkci `g_date_strftime()`, která pracuje podobně jako klasická funkce `strftime()` s tím rozdílem, že jako parametr, pro vytvoření lokalizovaného řetězce data či času, se zadává datum nastavené v objektu *GDate*.

Current object:	none		
Tested objects:	23	**Elapsed time:**	00:00:09
Infected objects:	20	**ZAVd version:**	1.1
Probably infected objects:	0	**Core version:**	1813
Suspicious objects:	0	**Database version:**	1082728
Nonstandard objects:	0	**Last DB update:**	04/28/2009

Obrázek 4.4: Statistiky v GTK+/GNOME

Qt/KDE

Stejně jako u knihoven GTK+/GNOME je statistická část uživatelského rozhraní složena z objektů *QLabel*, které zobrazují jednotlivé statistické údaje. O zobrazení data ve spravně lokalizovaném formátu se stará objekt *KLocale*. Tento objekt, inicializovaný podle aktuálního nastavení systému, lze získat pomocí metody `KGlobal::locale()`. Objekt *KGlobal* spravuje objekty, které existují pro každou aplikaci pouze jednou a jsou závislé na aktuálním nastavení celého systému s plochou KDE. Objekt *KLocale* poskytuje podporu pro překlad, určení formátu čísel, měny, data a času podle globálního nastavení v systému KDE.

Current object:	none		
Tested objects:	8	**Elapsed time:**	00:00:00
Infected objects:	7	**ZAVd version:**	1.1
Probably infected objects:	0	**Core version:**	1785
Suspicious objects:	0	**Database version:**	159937
Nonstandard objects:	0	**Last DB update:**	10.3.2009

Obrázek 4.5: Statistiky v Qt/KDE

4.4.7 Animace

Hned vedle části uživatelského rozhraní se statistikami se nachází grafický prvek s animací. Tento prvek informuje uživatele, zda je právě spuštěn skenovací proces. Pokud je v prvku s animací pouze statický obrázek, pak je skenování vypnuto, pokud v tomto prvku běží animace, pak právě probíhá skenování souborů.

Animaci zobrazovanou při spuštěném skenovaní sestavuje sada pěti obrázků, které tvoří stále se opakující. Tyto obrázky je potřeba nainstalovat spolu s aplikací na správné místo do systému (podobně jako ikony).

GTK+/GNOME

V knihovnách GTK+/GNOME je tento grafický prvek vložen pomocí objektu *GtkImage*. Při spuštění aplikace se tomuto objektu přiřadí první ze série obrázků tvořících animaci. Pokud je skenovací proces spuštěn, spustí se i časovaná událost funkcí `g_timeout_add()`. Tato časovaná událost zavolá každých 50 milisekund jinou funkci, která byla stanovena při spouštění časové události, a tato funkce pokaždé změní aktuální zobrazovaný obrázek v grafickém prvku na další v řadě. Časovaná událost se opakuje do té doby, než zavolaná funkce vrátí *FALSE* a tím se také zastaví animace. Objekt *GtkImage* již v sobě obsahuje ošetření blikání při rychlé změně obrázku, není proto nutné zavádět žádná další opatření.

Pro to, aby aplikace našla správné obrázky, je nutné mít tyto nainstalovány správně v systému a aktuálně zobrazovaný obrázek měnit funkcí `g_image_set_from_icon_name()`. Tak je zaručeno, že aplikace bude obrázky hledat ve správných adresářích v systému.

Obrázek 4.6: Animace v GTK+/GNOME

Qt/KDE

U knihoven Qt/KDE je přístup odlišný. Oproti variantě v knihovnách GTK+/GNOME totiž nelze zobrazit obrázky přímo pomocí objektu *QImage*. Tento objekt je pouze reprezentací pixmapy a nikoli grafickým prvkem, který lze vložit do grafického rozhraní. Proto je na místo, kde má být zobrazena animace, vložen objekt *QLabel*, na který se posléze vykresluje načtený obrázek. Bohužel při časté změně obrázků, která je nutná pro animaci, se projeví fakt, že se zobrazovací plocha objektu *QLabel* nejprve smaže a poté se na ni vykreslí obrázek nový. Výsledná animace pak bliká při výměně obrázků. Proto je nutné překreslovat obsah tohoto *QLabel* pomocí metody `QPaintDevice::bitBlt()`, která společně s použitým obrázkem ve formátu *QPixmap* zaručí výměnu obrázků bez blikání.

Obrázky pro animaci jsou načítány pomocí objektu *KIconLoader*, který zprostředkovává načítání ikon z aktuálního zvoleného motivu v prostředí s pracovní plochou KDE.

Obrázek 4.7: Animace v Qt/KDE

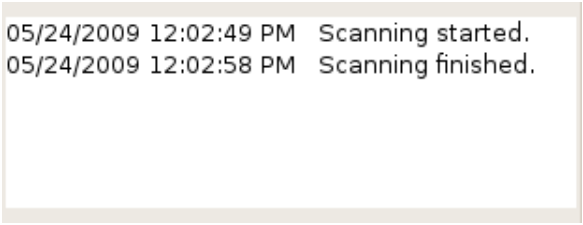
4.4.8 Log

Grafický prvek reprezentující logovací součást grafického rozhraní je využíván pro zaznamenávání důležitých událostí při běhu aplikace s grafickým rozhraním pro antivirového démona. V současné verzi se do tohoto logu zaznamenávají pouze události spojené s během skenovacího procesu, konkrétně start, pozastavení, restart, zastavení a dokončení skenování. Součástí každého záznamu je datum a čas události v lokalizované podobě. V budoucím vývoji projektu se zde budou zobrazovat i výsledky komunikací s ostatními procesy jako je manipulace s démonem antiviru.

GTK+/GNOME

V knihovnách GTK+/GNOME je prvkem reprezentujícím logovací objekt *GtkTextView*. Tento prvek zobrazuje víceřádkový textový výstup. Text, který má být zobrazen, je spravován pomocí objektu *GtkTextBuffer*. Každý prvek *GtkTextView* zobrazuje právě jeden *GtkTextBuffer*. Text je uchováván, spolu s případnými formátovacími znaky, právě v tomto bufferu a je zobrazen tam, kam byl příslušný buffer přiřazen.

Pro získání aktuálního času v lokalizované formě jsou opět použity funkce z knihovny *time.h*. Konkrétní použité funkce jsou `time()` pro získání aktuálního času a `strftime()` pro převod času a data na lokalizovanou podobu podle nastavení sytému.

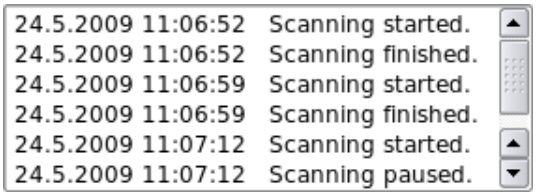


Obrázek 4.8: Log v GTK+/GNOME

Qt/KDE

Knihovny Qt/KDE používají pro zobrazování víceřádkového textu objekt *QTextEdit*. Zobrazovaný text se uchovává přímo v tomto prvku, není tedy třeba používat žádný další prvek pro jeho úpravu.

Pro získání data a času se používají objekty *QTime* a *QDate*. K převodu získaných hodnot na lokalizovanou podobu se využije objekt *KLocale*, který z uložených hodnot vytvoří řetězce s časem a datem v lokalizované podobě podle aktuálního nastavení systému.



Obrázek 4.9: Log v Qt/KDE

4.4.9 Prohlížeč souborů

Prohlížeč souborů je součást grafického uživatelského rozhraní, která implementuje možnost procházení adresářovou strukturou systému a výběr souborů či adresářů pro skenování.

Pro reprezentaci adresářové struktury jsem zvolil stromovou strukturu. V této struktuře každý uzel stromu reprezentuje adresář a každý list reprezentuje soubor. Každý řádek zobrazovaný ve stromové struktuře reprezentuje jeden soubor či adresář na disku a skládá se z následujících částí:

- Rozbalovací prvek - pokud daný řádek reprezentuje adresář, pak je úplně vlevo v tomto řádku zobrazen rozbalovací prvek, na který lze kliknout myší a tak rozbalit jeho obsah.
- Zaškrtávací pole - každý řádek obsahuje toto pole, aby bylo možné přidat soubor nebo adresář, který tento řádek reprezentuje, do fronty pro skenování.
- Ikona - dává jednoznačnou vizuální informaci o tom, zda daný řádek reprezentuje adresář či soubor.
- Název souboru či adresáře - tento údaj poskytuje název pro daný záznam na disku. Cestu k danému souboru či adresáři reprezentují rodičovské uzly daného stromu.

V této stromové struktuře se zobrazují pouze adresáře a soubory, které lze číst. Je to proto, aby mohl soubor být oskenován. Antivirové jádro, které skenování provádí, totiž vyžaduje přístup k tomuto souboru s právy pro čtení. Bylo by tak zbytečné zobrazovat v této struktuře soubory, které nelze skenovat.

GTK+/GNOME

V knihovnách GTK+/GNOME je grafický prvek zobrazující stromovou strukturu objekt *GtkTreeView*. Tento prvek se používá pro zobrazení stromových struktur nebo seznamů (speciální případ stromu, kdy existuje pouze jedna úroveň), které mají jeden či více zobrazovaných sloupců.

Nejdůležitější koncept, který je základní myšlenkou při používání *GtkTreeView*, je kompletní oddělení dat a toho jak jsou tyto data zobrazena na obrazovce. Tento koncept se nazývá model/zobrazení/ovládání (Model/View/Controller - MVC). Data různých typů, například řetězce, čísla, obrázky, atd., jsou uložena v modelu, který je implementován objektem *GtkTreeModel*. Zobrazení je poté sděleno, která data má zobrazit, kde je má zobrazit a jak je má zobrazit. Jednou z výhod tohoto přístupu je možnost mít více zobrazení, která čerpají data z jednoho modelu. Tato data však může každé zobrazení interpretovat rozdílně nebo i stejně ovšem s tou výhodou, že tato data nemusejí existovat ve více kopiích. Toto zabraňuje duplikaci dat a programovací činnosti, pokud jsou použita stejná data pouze

v jiném kontextu. Další výhodou je také to, že pokud dojde ke změně dat v modelu, pak všechna zobrazení, která čerpají data z tohoto modelu se aktualizují najednou.

Zatímco *GtkTreeModel* je použit pro uložení dat, existují jiné komponenty, které určují, jaká data budou zobrazena a jak se tato data zobrazí. Tyto komponenty jsou sloupce (implementovaný v objektu *GtkTreeColumn*) a vykreslovací jednotky (objekt *GtkCellRenderer*). *GtkTreeView* je složeno ze sloupců. Toto jsou sloupce, které uživatel vidí jako sloupce na obrazovce. Sloupce mají hlavičku, na kterou je možné klikat, dále obsahují titulek a lze u nich měnit velikost a řadit je podle nadefinované funkce. Sloupce samy o sobě nezobrazují žádná data, jsou použity pro reprezentaci uživatelské části prvku pro zobrazování stromové struktury. Sloupce jsou také využívány jako obalovací rám, do kterého se skládají jednotlivé vykreslovací jednotky, které se starají o zobrazování dat uložených v modelu na obrazovku. Každý sloupec musí obsahovat alespoň jednu zobrazovací jednotku, ale může jich obsahovat i více.

Vykreslovací jednotky jsou objekty, které mají určité vlastnosti a tyto vlastnosti určují jak bude daná buňka vykreslena. Buňka je část řádku, která zobrazuje určitý typ dat jedním způsobem. Každá buňka má svoji vykreslovací jednotku.

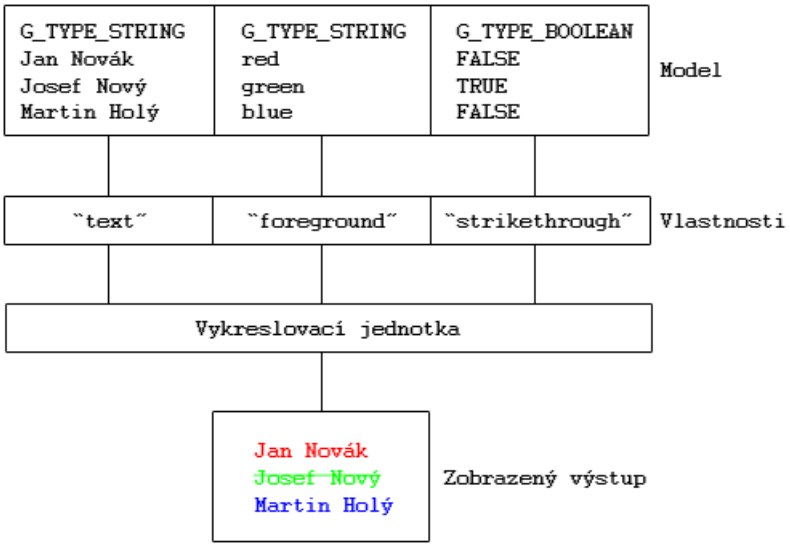
Pro to, aby buňky na různých řádcích byly vykreslovány s jiným obsahem, je nutné nastavit vykreslovací jednotku dané buňky. Toto se provádí pomocí atributů nebo funkcí, které pracují nad daty buněk. Při nastavení atributů se v podstatě spojí jeden sloupec z modelu s určitou vlastností vykreslovací jednotky. Před vykreslením každého řádku se poté přečte z modelu hodnota pro daný řádek, která se posléze nastaví pro vlastnost vykreslovací jednotky a ta podle těchto vlastností vykreslí obsah buňky. Druhou možností je nastavit funkce, které se budou volat pro každý řádek, který má být vykreslen. Tak je možné nastavit hodnoty vlastností vykreslovacích jednotek manuálně před tím, než jsou použity pro vykreslení dat. Oba přístupy lze kombinovat a používat najednou. Také je možné nastavit vlastnosti vykreslovacích jednotek při jejich vytváření. Tak se změněné vlastnosti použijí pro všechna vykreslování, ovšem pouze do té doby než budou jinak změněna.

Pro různé účely existují různé vykreslovací jednotky:

- *GtkCellRendererText* vykresluje řetězec, čísla nebo pravdivostní hodnoty jako text (například *Jan*, *1337*, *true*). Pokud tedy v modelu bude sloupec s čísly a tento sloupec se spojí s vlastností *text* vykreslovací jednotky *GtkCellRendererText*, pak se toto číslo převede na řetězec a zobrazí jako text.
- *GtkCellRendererPixbuf* se používá pro zobrazení obrázků. Tyto obrázky mohou být definované uživatelem nebo ty, které jsou poskytovány spolu s GTK+.
- *GtkCellRendererToggle* zobrazuje pravdivostní hodnoty ve formě zaškrtávacího pole nebo výběrového tlačítka.
- *GtkCellEditable* je speciální buňka, která umožňuje vložit editovací ovládací prvky jako například *GtkEntry* nebo *GtkSpinButton*.

Vykreslovací jednotka nevykresluje pouze jednu buňku z celého prvku pro zobrazení stromu, ale je zodpovědná za vykreslení části či dokonce celého sloupce v každém řádku. V podstatě začne v prvním řádku a vykreslí jeho část, za kterou je zodpovědná. Poté pokračuje na dalším řádku, kde opět vykreslí svoji část a takto pokračuje až k poslednímu řádku.

V příkladu na obrázku 4.10 je vidět proces při použití atributů. Vlastnost *text* textové vykreslovací jednotky byla v tomto příkladu spojena s prvním sloupcem v použitém modelu. Vlastnost *text* obsahuje řetězec, který má být vykreslen. Vlastnost *foreground*, která



Obrázek 4.10: Příklad vlastností vykreslovací jednotky a jejich vliv

ovlivňuje použitou barvu písma, je spojena s druhým sloupcem z modelu. Konečně pak vlastnost *strikethrough*, která určuje zda je použitý text přeškrtnut horizontální čarou, je spojena s třetím sloupcem z modelu. Výsledek takto nastaveného zobrazení je vidět na obrázku úplně vpravo. Datový typ vlastnosti, která se má nastavit podle modelu musí odpovídat datovému typu ve sloupci tohoto modelu.

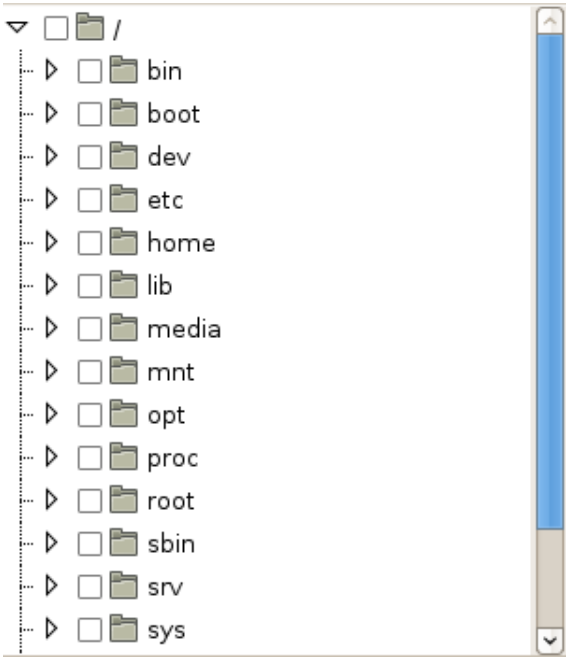
Pro aplikaci implementující grafické rozhraní pro funkční jádro antiviru jsem vytvořil model, který má 6 sloupců:

- Jméno souboru nebo adresáře - typ *G_TYPE_STRING*,
- cesta k souboru nebo adresáři - typ *G_TYPE_STRING*,
- zda daný řádek reprezentuje adresář nebo soubor - typ *G_TYPE_BOOLEAN*,
- zda je daný řádek vybraný pro oskenování - typ *G_TYPE_BOOLEAN*,
- zda je adresář konzistentní - typ *G_TYPE_BOOLEAN*,
- název ikony použité pro daný řádek - typ *G_TYPE_STRING*

V knihovnách GTK+/GNOME neexistuje jednoduchý způsob jak propojit chování potomků při výběru rodičovského uzlu a naopak. Proto je nutné toto chování naimplementovat ručně odchycením signálu pro kliknutí myší na daný řádek a manuálním nastavením všech dotčených řádků. Je tak nutné si v modelu vyhradit sloupec pro ukládání vybraného či nevybraného stavu. Dalším důležitým údajem, který je nutno uchovávat v modelu, je skutečnost, zda daný řádek a jeho synovské uzly jsou konzistentní, tedy zda se má uzel zobrazovat jako vybraný, nevybraný nebo vybraný jen z části. Toto nastavení má ovšem význam pouze u adresářů, jelikož souborové řádky nemají žádné potomky v prohlížeči souborů.

Z takto nadefinovaného modelu se zobrazí pouze první sloupec jako jediný zobrazovaný sloupec v celém *GtkTreeView*, který reprezentuje prohlížeč souborů. Ostatní sloupce z modelu jsou použity jako hodnoty vlastností pro zobrazování jednotlivých řádků.

Plnění modelu daty, tedy zjišťování obsahu potřebného adresáře, vlastností souborů a adresářů v něm obsažených zajišťuje *GFile*. Funkce `g_file_enumerate_children()` získá seznam souborů a adresářů, které jsou přímými potomky daného *GFile*. Tento seznam se posléze projde a vyberou se relevantní prvky pro zobrazení v prohlížeči souborů, tedy všechny prvky, které lze číst, kromě prvků reprezentujících aktuální adresář a adresář o úroveň výše, které se do stromové struktury neuvádějí.



Obrázek 4.11: Prohlížeč souborů v GTK+/GNOME

Qt/KDE

V knihovnách Qt/KDE je implementován objekt *QListView*, který posykytuje možnost zobrazení a ovládání stromové struktury nebo seznamu s jedním nebo více sloupci. Narozdíl od knihoven GTK+/GNOME zde neplatí systém model/zobrazení/ovládání. Prvek, který reprezentuje grafické zobrazení taktéž udržuje potřebná data, která se budou zobrazovat.

Stejně jako v knihovnách GTK+/GNOME se nejprve vytvoří prvek *QListView*, poté se do něj přidají sloupce, které budou zobrazovány uživateli. Zde ovšem podobnost končí, protože v knihovnách Qt/KDE se v tomto okamžiku již jen přidávají jednotlivé řádky. Jeden řádek může být reprezentován dvěma různými prvky:

- *QListViewItem* - u tohoto prvku lze nastavit zobrazovaný text, ikonu a zda je možné jej otevřít a zobrazit tak jeho potomky,
- *QCheckListItem* - tento prvek dědí veškerou funkčnost z předchozího a navíc přidává zaškrtávací pole pro výběr tohoto prvku.

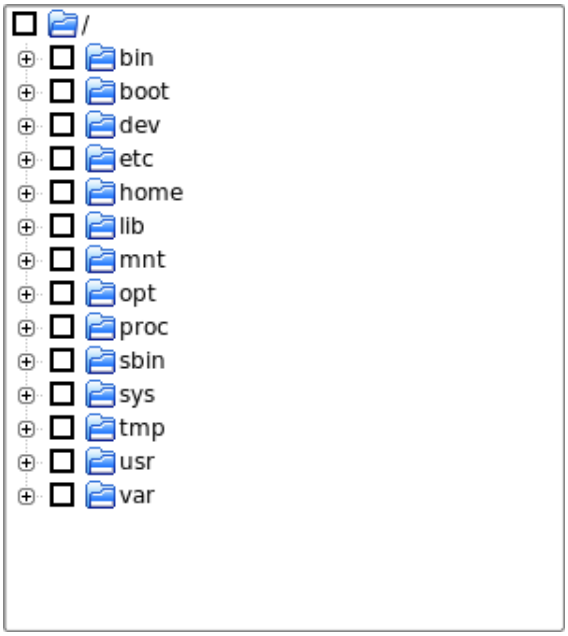
Nevýhodou tohoto přístupu, oproti tomu z knihoven GTK+/GNOME, je neupravitelné pořadí jednotlivých částí těchto prvků. Vždy bude úplně vlevo symbol pro rozkliknutí prvku, pak eventuelně zaškrtávací políčko či výběrové tlačítko (pouze pokud je zvolen prvek

QCheckListItem), poté přiřazená ikona a jako poslední zobrazovaný text. Každý prvek obsahuje data pro každý ze sloupců zobrazovaný v *QListView*.

Výhodou u knihoven Qt/KDE je již naimplementovaná závislost mezi synovskými a rodičovskými uzly stromu. Pro to, aby se uzly v hierarchii mezi sebou správně ovlivňovaly, stačí při vytváření každého prvku určit, zda se jedná list stromu nebo o uzel, který pod sebou bude mít další uzly a listy. Při vytvoření správné struktury se pak jednotlivé uzly a listy navzájem ovlivňují. Pokud je například vybrán určitý uzel, jsou vybráni i všichni jeho potomci. Není potřeba toto chování obsluhovat manuálně jako u knihoven GTK+/GNOME.

Pro aplikaci s grafickým uživatelským rozhraním pro antivirového démona jsem použil *QListView*, do kterého jsem přidal dva sloupce: Jeden pro jméno souboru nebo adresáře, druhý pro cestu k tomuto souboru nebo k adresáři. Sloupec s cestami je ovšem skrytý a je používán pouze pro vnitřní interakci fronty pro skenování a prohlížeče souborů. Prvek pro zobrazení stromové struktury má tedy ve výsledku pouze jeden viditelný sloupec, což je ovšem dostačující, protože uživatel cestu k danému souboru či adresáři může vysledovat z rodičovských uzlů.

Pro naplnění *QListView* potřebnými daty se používá *QDir*. Pomocí jedné z jeho metod `QDir::entryInfoList()` se získá seznam souborů a adresářů, které patří pod daný adresář a po odfiltrování nedůležitých prvků se tyto přidají do stromové struktury vytvořením *QCheckListItem* prvků správného typu.



Obrázek 4.12: Prohlížeč souborů v Qt/KDE

4.4.10 Fronta souborů

Asi nejdůležitějším prvkem celého grafického rozhraní pro antivirového démona je prvek zobrazující frontu souborů pro skenování. Tento prvek totiž zobrazuje soubory, vybrané uživatelem přes prohlížeč souborů, které mají být oskenovány. Tento prvek ale také zobrazuje výsledky skenování tak, jak jsou vráceny jádrem antiviru.

Soubory vybrané pro skenování jsou zobrazovány ve formě seznamu, který má následující sloupce:

- Číslo - pořadové číslo ve frontě,
- jméno - název souboru či adresáře,
- status - označuje zda byl prvek již skenován a pokud ano, tak jeho výsledek,
- typ - pokud prvek nebyl skenován, pak se zde zobrazuje zda je prvek adresář nebo soubor, jinak jsou zde doplňující informace k výsledku skenování ke každému prvku,
- cesta - úplná cesta k vybranému souboru nebo adresáři.

Sloupec status může v současné verzi aplikace zobrazovat 10 různých hodnot: Neoskenovaný, infikovaný, pravděpodobně infikovaný, podezřelý, nestandardní, neznámý, čistý, chyba, vypršení skenovacího časového limitu a nebo vypršení časového limitu na soubor. Všechny tyto hodnoty jsou kromě textového zobrazení ve sloupci status odlišeny barvou pozadí a barvou textu celého řádku. Použité barvy lze natavit přes nastavovací dialog aplikace.

Ve sloupci typ se alternuje informace o tom, zda je daný prvek soubor či adresář s dodatečnými informacemi dodanými skenovacím procesem. Do fronty je možné přidávat celé adresáře zaškrnutím patřičného políčka v prohlížeči souborů, ovšem skenovacímu jádru je možné posílat pouze jednotlivé soubory. Proto pokud skenovací proces dojde k prvku, kterým je adresář, je nutné jej nejprve rozbalit na jednotlivé soubory a poté až oskenovat. Tak je také mimo jiné zaručeno, že oskenovaný prvek nemůže být adresářem a je tak možné využít sloupec *typ* nejen pro dodatečné informace od antivirového jádra, ale také pro identifikaci adresářů ve frontě. Dodatečnými informacemi od jádra antiviru se v tomto případě myslí typ viru, kterým je daný soubor nakažen. Uživatel tak například vidí, že daný soubor je infikovaný a typ viru, kterým je tento soubor nakažen, je 'VBS.Lovelorn.B'.

Fronta má většinou tvar seznamu, ovšem pokud má uživatel zapnuté rozbalování zabalených souborů nebo spustitelných souborů (obě možnosti jsou v základní konfiguraci povoleny), pak se může stát, že se virus bude nacházet hlouběji uvnitř těchto souborů. Pak se tyto prvky stávají kořenovým uzlem pro stromovou strukturu, ve které je vidět jaké části tohoto souboru jsou nakaženy a čím. Jednotlivé úrovně zanoření do daného souboru jsou odděleny odsazením od kraje stejně jako v prohlížeči souborů, navíc je zde ovšem přidáno rozlišení jednotlivých úrovní zanoření pomocí barevných odstínů základních barev zvolených v nastavení aplikace. Čím je zanoření hlubší, tím je barevný odstín tmavší.

GTK+/GNOME

Stejně jako pro prohlížeč souborů je pro vytvoření prvku zobrazujícího frontu použit objekt *GtkTreeView*. Model použitý pro uchovávání dat pro frontu obsahuje 6 sloupců:

- Pořadové číslo daného prvku ve frontě - typ *G_TYPE_INT*,
- jméno souboru nebo adresáře - typ *G_TYPE_STRING*,
- status prvku - typ *G_TYPE_STRING*,
- dodatečné informace k prvku - typ *G_TYPE_STRING*,
- cesta k souboru či adresáři - typ *G_TYPE_STRING*,

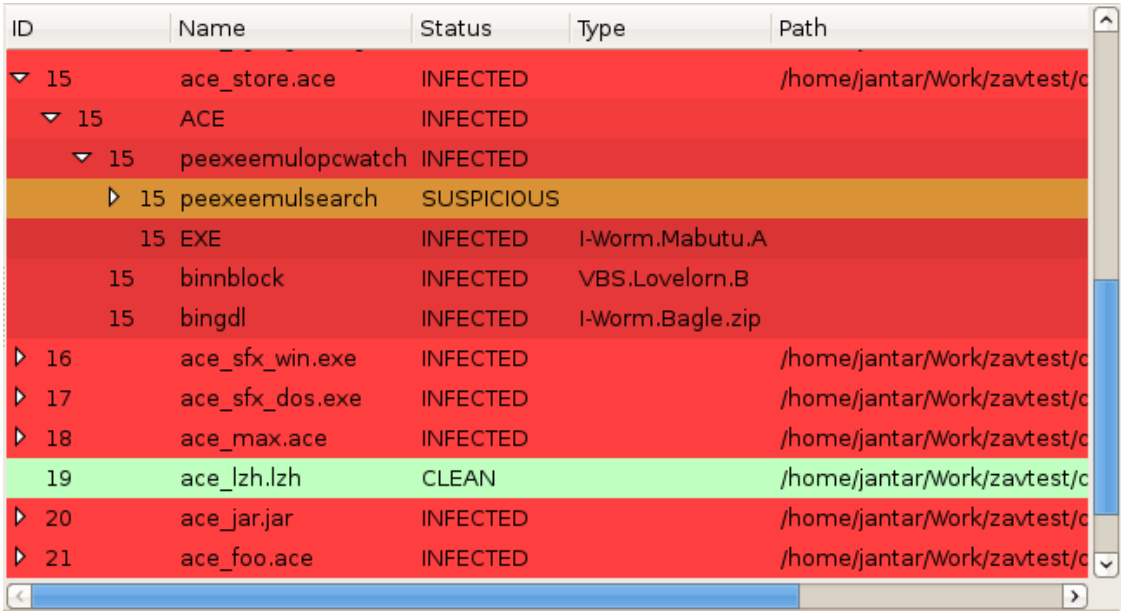
- číselná reprezentace statusu - typ *G_TYPE_INT*.

Prvních 5 sloupců modelu je spojeno se zobrazovanými sloupci v prvku představujícím frontu souborů jak byly definovány výše. Poslední šestý sloupec modelu je použit pro definování barev pro každý prvek ve frontě.

Každý zobrazovaný sloupec vložený do *GtkTreeView* je tvořen pouze jedinou vykreslovací jednotkou textového typu, které si berou informace z datového modelu a jeho jednotlivých sloupců. Jedná se tedy o jednoduché spojení modelu a zobrazení.

Pro změnu barev pozadí a písma u jednotlivých prvků podle jejich statusu je nutné definovat tyto barvy pro každou vykreslovací jednotku v řádku zvlášť. Proto je nutné nastavit pro každý zobrazovaný sloupec zvlášť funkci, která se postará o nastavení správné barvy pro všechny vykreslovací jednotky obsažené v tomto zobrazovaném sloupci. V tomto případě je nastavování barev u všech vykreslovacích jednotek o to jednodušší, že každý sloupec obsahuje pouze jednu takovou vykreslovací jednotku. Volaná funkce pro změnu jakýchkoli vlastností vykreslovacích jednotek daného sloupce se definuje pomocí funkce `gtk_tree_view_column_set_cell_data_func()`. Takto definovaná funkce se poté volá pro každou buňku ve sloupci před tím, než je tato buňka vykreslena. V případě aplikace s grafickým uživatelským rozhraním pro antivirového démona se mění vlastnosti *foreground-gdk* pro změnu barvy písma a *cell-background-gdk* pro změnu barvy pozadí. Hodnoty těchto vlastností se nastaví podle čísla reprezentujícího status prvku v posledním sloupci datového modelu a podle příslušných hodnot v nastavovacím dialogu aplikace.

Pro dosažení ztmavení podle aktuálního zanoření bylo nutné implementovat převod barvy do HSV systému, v tomto systému zvětšit patřičně V složku a převést barvu zpět na RGB reprezentaci, se kterou umí knihovna GTK+ pracovat.



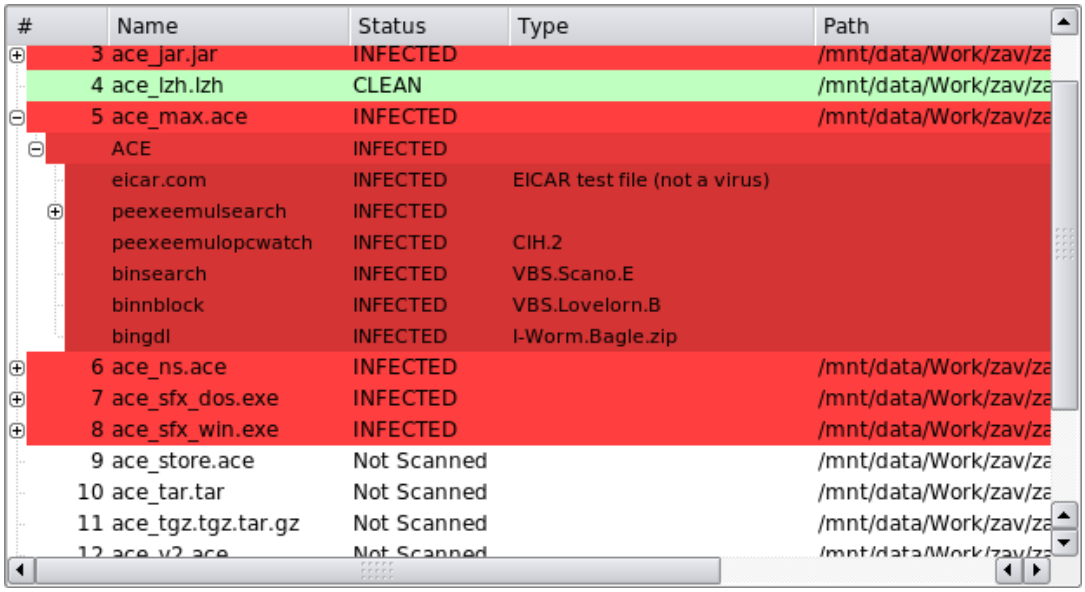
Obrázek 4.13: Fronta souborů v GTK+/GNOME

Qt/KDE

V knihovnách Qt/KDE se stejně jako u prohlížeče souborů použije objekt *QListView*. Do tohoto objektu se přidá 5 sloupců, které odpovídají definici uvedené výše.

Rozdílem oproti prohlížeči souborů je nutnost zdědit základní prvek *QListViewItem* a tento upravit podle potřeb tak, aby zobrazení fronty souborů bylo korektní. V knihovnách GTK+/GNOME se problém s uchovávání číselné hodnoty statusu vyřešil přidáním dalšího sloupce k datovému modelu. V knihovnách Qt/KDE ovšem tento princip není zaveden a proto jsem přidal do definice upraveného *QListViewItem* vlastnost, která uchovává právě číselnou hodnotu tohoto prvku. Dalším důvodem pro zdědění a úpravu základního prvku je nutnost upravit funkci, která se stará o vykreslování jednotlivých buněk a implementovat tak změnu barev u každého prvku. Funkce, která je upravována se jmenuje `QListViewItem::paintCell()`. V této funkci se nastaví hodnoty správně vybraných barev podle nastavení aplikace a podle statusu prvku a takto nastavené hodnoty se předají neupravené funkci `QListViewItem::paintCell()`, která se postará o správné vykreslení se správně vybranými barvami pozadí a textu.

Pro vybrání správného odstínu barvy podle zanoření se použije funkce `QColor::dark()`, která převede barevnou reprezentaci na HSV model, vynásobí V složku zadaným koeficientem a převede zpět na RGB reprezentaci. Není tedy třeba algoritmus implementovat znovu jako v knihovnách GTK+/GNOME.



Obrázek 4.14: Fronta souborů v Qt/KDE

4.4.11 Nastavení

Nastavením je v tomto případě myšlen dialog, který slouží pro nastavení vzhledu částí aplikace a skenovacího procesu. Dialog použitý v aplikaci s uživatelským rozhraním pro antivirového démona se dá rozdělit do tří hlavních sekcí, ve kterých se nastavují preference uživatele pro:

- Nastavení skenovacího procesu,

- nastavení fontu,
- nastavení barev.

V sekci pro nastavování skenovacího procesu je možné nastavit cestu, kde se nachází socket, přes který aplikace komunikuje s antivirovým démonem a cestu kde je démon nainstalovaný. V další části nastavení skenovacího procesu lze upravovat, jaké metody budou použity nebo zakázány při skenování souborů, zda se mají rozbalovat archivy a spustitelné soubory a pokud ano, do jaké hloubky se mají rozbalovat nebo jakou maximální velikost lze rozbalit z jednoho souboru. Také zde lze nastavit maximalní časové limity pro skenování jednoho souboru, po jejichž překročení se přejde na skenování dalšího souboru.

Sekce nastavení fontu slouží pro vybrání libovolného fontu, který se použije pro všechny prvky grafického rozhraní, kromě hlavního řádkového menu, panelu nástrojů a stavového řádku. V této části nastavení lze definovat rodinu použitého písma, styl, tedy zda je písmo tučné, kurzívou nebo jejich kombinace a v neposlední řadě lze nastavit i velikost použitého písma.

Poslední sekcí nastavení je změna barev pozadí a písma u logovacího textového pole, prohlížeče souborů a fronty souborů pro skenování. U fronty souborů pro skenování lze upravit nejen barvy v základní podobě, ale také barvu písma a pozadí pro každý ze statusů jednotlivých prvků.

GTK+/GNOME

Pro uchovávání nastavení i po vypnutí aplikace se v knihovnách GTK+/GNOME používá systém *GConf*. *GConf* je mechanismus, který se používá pro uchovávání konfiguračních dat. *GConf* lze využít i ve spolupráci s jinými knihovnami jako je Xlib nebo KDE. Pro prostředí s pracovní plochou GNOME je to ovšem základní forma ukládání takovýchto dat a aplikace vyvíjené pro toto prostředí by za účelem jednotnosti měly tento systém respektovat.

GConf umožňuje použití různých architektur pro uchovávání konfiguračních dat. V základním nastavení ukládá tato data do textových souborů ve formátu XML, avšak je možné použít databázi podobnou registrům Windows, kdy jsou data uložena v binární podobě, či plně funkční SQL databázi. U každého klíče je možné uvést jeho dokumentaci. Aplikace mohou spolu s klíčem a hodnotou uložit také popis možných hodnot nastavení a jejich efekt. Existuje nástroj, kterým lze tuto dokumentaci přečíst a změnit hodnotu klíče podle informací v dokumentaci k tomuto klíči, aniž by uživatelé museli hádat možná nastavení. Databáze *GConf* také uchovává určitá metadata, například: která aplikace klíč vlastní nebo kdy byla hodnota klíče naposledy upravena.

Velmi významnou vlastností *GConf* je systém zpráv o změně hodnoty klíče, které jsou zasílány všem aplikacím, které by mohla takováto změna zajímat. Tento systém zpráv funguje i přes síť, tudíž je-li uživatel přihlášen na více místech, pak se zprávy pošlou i aplikacím spuštěným v ostatních sezeních stejného uživatele. To znamená, že pro aplikace složené z různých komponent, které mohou běžet v různých procesech, je jednoduché spravovat jejich nastavení. Pokud jedna část aplikace změní určité nastavení, ostatní části aplikace budou uvědoměny pomocí zpráv a mohou na to patřičně zareagovat. Systém zpráv je také výhodný při vícenásobném běhu jedné aplikace. Prostředí s pracovní plochou GNOME je implementováno tak, aby se veškerá nastavení projevila ihned, bez nutnosti restartování aplikace, a aplikace vyvíjené pro toto prostředí by toto měly respektovat. Je nutné, aby při změně nastavení v jedné instanci aplikace se toto nastavení projevilo i v ostatních aktuálně spuštěných instancích. Se systémem zpráv je toto zaručeno. Pokud totiž jedna instance

změní určité nastavení, vyšle se zpráva ostatním instancím a ty se musejí podle nového nastavení zařídit.

GConf uchovává jednoduchý pár klíč-hodnota. Klíč má podobu obyčejného unixového jména souboru. Klíče jsou organizovány do adresářů stejně jako je tomu u unixových souborů. Například adresář `/desktop` uchovává nastavení pro uživatelskou plochu a klíč ve formátu `/desktop/standard/background-color` bude použit pro uchovávání nastavení barvy pozadí pro uživatelskou plochu. V dokumentaci pro *GConf* je uvedeno několik základních jmen adresářů, které používá systém a pokud chce programátor použít některou z těchto vlastností, pak musí pracovat s těmito základními jmény. Hodnoty spojené s každým klíčem jsou různého typu, mezi základní typy patří celé číslo, booleovská hodnota, řetězec a číslo s pohyblivou desetinou čárkou. Jednotlivé typy nemají omezenenou velikost, ukládané řetězce mohou mít tedy libovolnou délku, avšak nepřiměřeně velké řetězce způsobí určité výkonostní problémy.

GConf je implementován jako démon zvaný *gconfd* a každý uživatel využívá služeb právě jednoho tohoto démona. Démon *gconfd* se také stará o zasílání upozornění o změně hodnoty aplikacím, kterých by se změna mohla týkat. Jelikož veškeré přístupy do databáze hodnot jsou prováděny právě přes *gconfd* není problém spravovat přístup k těmto datům z více aplikací naráz.

Aplikace používající *GConf* by měly mít své preference a konfigurační kód rozděleny podle již výše vysvětleného systému model/zobrazení/ovládání. Podle tohoto systému by se nastavení aplikace mělo odehrávat pomocí třech oddělených objektů:

- Model - je tvořen daty, která jsou prezentována uživateli a která je uživatel schopen měnit podle svých preferencí. V tomto případě je model tvořen databází systému *GConf* přesněji jednotlivými klíči, které se týkají aplikace s grafickým rozhraním pro démona antiviru.

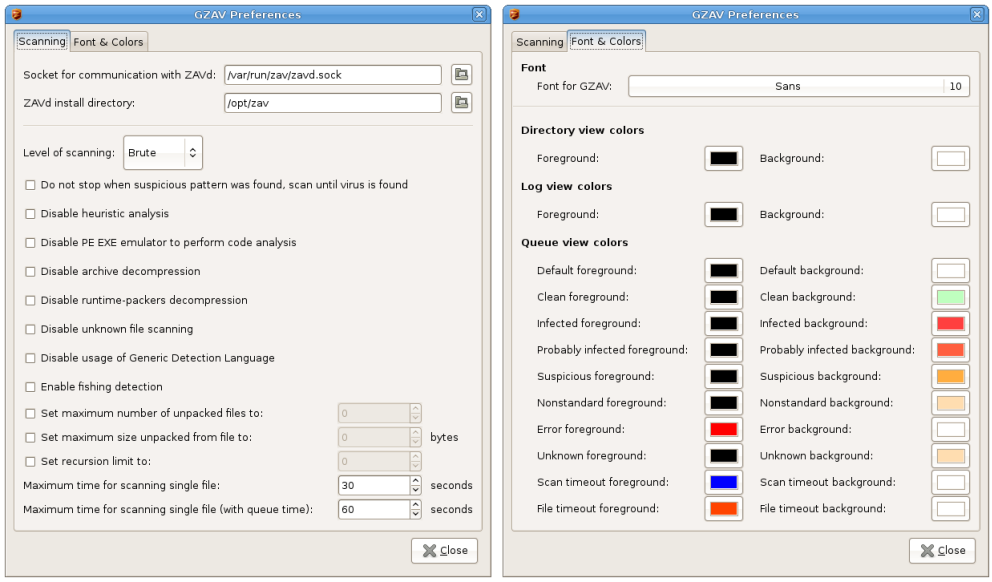
- Zobrazení - určitým způsobem zobrazuje data, která jsou uložena v modelu. Zobrazení dostává určitý typ upozornění, že se obsah modelu změnil a grafická reprezentace modelu by se tedy měla aktualizovat. V případě aplikace s grafickým rozhraním pro démona antiviru lze zobrazení chápat právě jako tuto aplikaci, přesněji tedy jednotlivé grafické prvky, které mění svůj vzhled podle nastavení uloženého v modelu.

- Ovládání - je určené pro modifikaci dat v modelu. V implementované aplikaci pak lze jako ovládání definovat dialog s nastavením, kde uživatel právě tato nastavení může libovolně upravovat a tím měnit obsah modelu.

V knihovně GNOME ještě existuje rozšíření *GConfClient*, které oproti základnímu *GConf* implementuje vyrovnávací paměť, kam se mohou předem načíst adresáře, které jsou pro danou aplikaci zajímavé a tak urchlit přístup k těmto adresářům (ať už kvůli změně hodnot nebo pouze přečtení aktuálně platné hodnoty). Další výhodou oproti základní verzi *GConf* je možnost zapnout automatické obsluhy chyb, které se mohou vyskytnout při komunikaci s databází *GConf*. V neposlední řadě také *GConfClient* posílá signál *value_changed*, který se vygeneruje, pokud přijde zpráva od *gconfd*, že se změnila určitá část datového modelu.

Konfigurační dialog pro nastavení aplikace je, pro udržení konzistence s většinou aplikací v prostředí GNOME, tvořen jednoduchými záložkami. Výběr záložek je v horní části dialogu a titulek každé záložky má pouze textovou formu. Každý prvek ovládající jednotlivé nastavení musí mít při změně okamžitý účinek ve spuštěné aplikaci. Tento způsob je

propagován prostředím GNOME. Jakákoli změna se tedy musí ihned projevit a proto také konfigurační dialog obsahuje pouze tlačítko *Close* pro zavření tohoto dialogu.



Obrázek 4.15: Záložky konfiguračního dialogu v GTK+/GNOME

Qt/KDE

V knihovnách Qt/KDE se pro uchovávání uživatelských preferencí používá objekt *KConfig*. Čtyřmi základními částmi celého systému udržování uživatelského nastavení jsou:

- *KConfigSkeleton*, což je třída zprostředkovávající jednodušší přístup ke konfiguračním datům,
- XML soubor uchovávající jednotlivé vlastnosti, či nastavení (přípona tohoto souboru je `.kcfg`),
- INI soubor obsahující obecné informace potřebné pro vygenerování kódu (tento soubor má příponu `.kcfgc`),
- *kconfig_compiler*, který umožňuje vygenerování C++ kódu z `.kcfg` a `.kcfgc` souborů. Tento překladač vygeneruje třídu, která je založena na třídě *KConfigSkeleton* a přes tuto třídu se z aplikace přistupuje na uložená nastavení.

Je tedy nutné vytvořit soubor s příponou `.kcfgc`, který obsahuje data potřebná pro vygenerování třídy, přes kterou se bude aplikace dostávat k uloženým preferencím. Tento soubor musí obsahovat jméno souboru, kde jsou uvedena všechna nastavení a jejich vlastnosti, tedy soubor s příponou `.kcfg`. Dále pak je nutné uvést jak se má pojmenovat vygenerovaná třída. Toto jméno je velice důležité, protože se bude používat v aplikaci pro přístup k uloženým datům. Tyto dva údaje jsou nejdůležitější a každý `.kcfgc` soubor je musí obsahovat pro úspěšné vygenerování potřebné třídy. V souboru mohou být uvedeny ještě další informace, které ovlivní vlastnosti výsledné vygenerované třídy.

Druhým potřebným souborem je ten s příponou `.kcfg`. Tento soubor obsahuje jednotlivé prvky nastavení, které jsou v aplikaci použity. U jednotlivých prvků je nutné uvést jejich jméno a typ. Uvedené jméno je velice důležité a slouží pro vytvoření spojení s grafickými prvky přímo v aplikaci. Postup spojení je popsán níže. Lze rovněž uvést další doplňující informace jako například popis, implicitní hodnota, minimální a maximální hodnota, atd. Jednotlivé prvky lze sdružovat do skupin podle jejich logické příslušnosti. Soubor je ve formátu XML.

Poté, co jsou vytvořeny tyto dva soubory a podle těchto souborů je vygenerována třída přistupující na jednotlivé preference, zbývá už jen vytvořit grafické prvky, které budou ovládat jednotlivá nastavení přímo z aplikace a spojit tyto grafické prvky s prvky uvedenými v XML souboru. Je třeba vytvořit dialog, který bude obsahovat veškerá nastavení aplikace, což se provede vytvořením objektu *KConfigDialog*. Tomu se při vytváření zadají parametry, které určí, jak má tento dialog vypadat. Pro zachování jednotnosti v systému jsem pro aplikaci s grafickým rozhraním pro démona antiviru vybral typ *IconList*, který je v podstatě dialogem se záložkami, pouze záložky jsou uvedeny po straně a zobrazují ikonu reprezentující jejich funkci. Dalším parametrem při vytváření konfiguračního dialogu je jméno třídy pro přístup k uloženým datům preferencí. Je to jméno již dříve vygenerované třídy. Po vytvoření prázdného dialogu se vytvoří zvlášť grafické prvky, které budou ovládat jednotlivá nastavení. Proto aby bylo možné ovládat nastavení pomocí nově vytvořených grafických prvků je nutné použít při pojmenovávání těchto grafických prvků určité konvence. Prvek ovládající určitou vlastnost se musí pojmenovat následujícím způsobem: Jméno souboru musí začínat prefixem `kcfg_` a zbytek jména je tvořen jménem vlastnosti, kterou má grafický prvek spravovat. Jméno vlastnosti musí odpovídat jménu uvedenému v XML souboru, jinak by se grafický prvek nespojil s daným nastavením. Jediné, co tedy spojuje vlastnost a grafický prvek je jejich jméno. Jednotlivé grafické prvky se poté umístí na prázdný objekt *QWidget* a tento objekt se přidá jako samostatná stránka do konfiguračního dialogu.

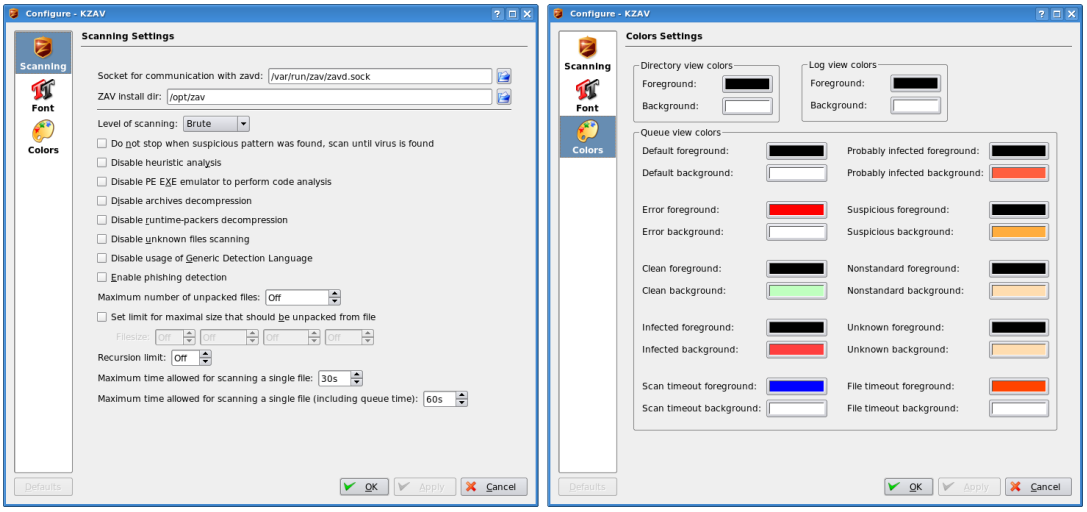
Konfigurační dialog se poté stará o ukládání nastavených hodnot automaticky. Pokud uživatel změní kteroukoli hodnotu na konfiguračním dialogu, tak se tato změna neprojeví v aplikaci ihned. Je nutné stisknout tlačítko *OK* nebo *Apply*, pro provedení změn. Pro zrušení provedených změn lze požít tlačítko *Cancel*. Konfigurační dialog dále může také obsahovat tlačítko *Default*, které vrátí všechna nastavení do základních hodnot, definovaných v souboru ve formátu XML, kde jsou vypsaná jednotlivá nastavení aplikace.

4.4.12 Skenovací vlákno

V aplikaci s grafickým uživatelským rozhraním jsem pro skenovací proces vyhradil speciální vlákno, aby skenování neovlivňovalo plynulost běhu grafické části programu. Při skenování větších souborů se může stát, že skenování tohoto souboru bude trvat delší dobu. Pokud by skenovací proces nebyl ve zvláštním vláknu aplikace, pak by po dobu skenovaní souboru nereagovalo grafické rozhraní a uživatel by si mohl myslet, že aplikace přestala fungovat.

Vlákno, ve kterém probíhá skenovací proces, je spuštěno ihned po spuštění aplikace a ukončeno je až s koncem aplikace. Ihned po startu vlákna se ovšem toto uvede do blokujícího stavu dokud není probuzeno z vlákna hlavního, tedy dokud uživatel nespustí skenování. Do stejného blokujícího stavu se vlákno vrátí pokaždé, když uživatel skenování přeruší nebo pokud skenovacím procesem projdou všechny soubory ve frontě pro skenování.

Pokud je skenovací proces spuštěn a ve frontě jsou stále prvky, které se mají oskenovat, pak se vezme jeden prvek z fronty a zjistí se, zda je tento prvek soubor či adresář. Pokud je daný prvek adresář, pak se tento adresář musí nahradit jeho obsahem. Adresář



Obrázek 4.16: Záložky konfiguračního dialogu v Qt/KDE

se nahradí ve frontě podobným způsobem jako se do prohlížeče souborů přidávají položky. Poté, co je adresář ve frontě nahrazen jeho obsahem, se přejde na další kolo cyklu. Pokud je aktuální prvek soubor, tak se soubor přepošle antivirovému jádru a čeká se na odpověď. Po přijetí odpovědi se informace zpracuje a ve frontě se upraví právě oskenovaný soubor podle výsledku obdrženého od antivirového jádra. Tento cyklus se opakuje do té doby, než ho uživatel přeruší nebo než jsou oskenovány všechny soubory.

GTK+/GNOME

Knihovna GLib, která tvoří základ knihoven GTK+/GNOME je zcela bezpečná vzhledem k používání vláken. Všechna globální data jsou automaticky zamykána, ovšem individuální datové struktury se z výkonostních důvodů nezamykají a programátor je odpovědný za koordinaci přístupů k těmto datům. Hlavní smyčka programu poskytovaná právě knihovnou GLib je také bezpečná pro použití v aplikaci s více vlákny.

Zbytek knihovny GTK+ již není úplně bezpečný pro použití s více vlákny. Ovšem používání této knihovny s více vlákny je velice usnadněno poskytnutým globálním zámkem, který lze ovládat pomocí funkcí `gdk_threads_enter()` a `gdk_threads_leave()`. Tento systém globálního zámku chrání použití jakékoli části GTK+. Správné použití tohoto zámku umožní přístup k funkcím GTK+ pouze z jednoho vlákna v každém časovém momentu.

Pokud aplikace má správně pracovat s více vlákny najednou je nutné zavolat funkce `g_thread_init()` a `gdk_threads_init()`. Tyto dvě funkce je nutné zavolat jako úplně první po spuštění aplikace jinak je chování aplikace nedefinované a může docházet k zamrzání a padání aplikace.

Pro vytvoření vlákna pro skenování je v knihovnách GTK+/GNOME použit objekt *GThread*. Pro ochranu dat sdílených mezi vlákny je využit *GMutex* a pro uvedení skenovacího vlákna do blokujícího stavu je použit *GCond*. Veškeré operace, které vyžadují úpravu grafického uživatelského rozhraní jsou uzavřeny mezi volání funkcí `gdk_threads_enter()` a `gdk_threads_leave()`, čímž je zaručena správná synchronizace obou vláken.

Qt/KDE

V knihovnách Qt/KDE vždy pouze jedno vlákno ovládá veškeré grafické rozhraní. Je to vždy to vlákno, ze kterého byla spuštěna hlavní smyčka programu. Je to také to vlákno, které je spuštěno ze vstupního bodu aplikace, funkce `main()`. Toto vlákno je jediné, kterému je dovoleno provádět operace s grafickým uživatelským rozhraním, včetně generování a přijímání událostí. V knihovnách Qt/KDE nelze spustit hlavní smyčku z jiného než hlavního vlákna programu.

Vlákna, která chtějí modifikovat obsah či vzhled grafických prvků, jsou nucena použít systém zasílání zpráv mezi vlákny a tak v podstatě sdělit hlavnímu vláknu, co má být změněno na kterých grafických prvcích. Zpráva z jednoho vlákna do jiného se posílá metodou `QApplication::postEvent()`. Většinou je nutné k této zprávě ještě přidat další data, která jsou důležitá pro správnou změnu grafických prvků. To je možné při použití *QCustomEvent*.

Pro odchycení zprávy poslané jiným vláknem je potřeba reimplementovat virtuální funkci `QObject::customEvent()`, která obslouží událost, která se vygenerovala, když daný objekt přijal zprávu. Každá zpráva by měla mít definované identifikační číslo, které určí typ posílané zprávy. Podle tohoto identifikačního čísla a dat poslaných společně se zprávou se pak provede úprava grafického rozhraní.

Pro vytvoření vlákna se v knihovnách Qt/KDE používá objekt *QThread* a pro ochranu sdílených dat se využívá *QMutex*. Je-li nutná synchronizace mezi vlákny, v aplikaci s grafickým rozhraním pro antivirového démona je synchronizace potřeba při určitých změnách grafických prvků, například je-li nutné nahradit adresář ve frontě soubory před tím, než se přejde na skenování dalšího prvku, tak se použije objekt *QWaitCondition*, který zastaví jedno vlákno dokud nedostane signál od jiného vlákna, že se může pokračovat dále.

Kapitola 5

Závěr

V úvodu této práce jsem zopakoval některé ze základních pojmů, které se týkají grafického uživatelského rozhraní, jako je okno, menu dialog a další. Připomněl jsem, že základní myšlenkou a vlastností uživatelského rozhraní je vytvářet aplikace jednodušší a intuitivnější pro koncového uživatele. Ovšem používání grafického rozhraní s sebou přináší určitá omezení vůči rozhraní přes příkazovou řádku. Také jsem se snažil zachytit vývoj uživatelského rozhraní od dob, kdy bylo takové rozhraní pouze myšlenka, až do podoby, jak jej známe dnes.

Ve druhé části tohoto dokumentu jsem probral zprostředkování základní funkčnosti pomocí X Window a jeho způsob komunikace typu klient-server. Na to jsem navázal popisem základní knihovny Xlib, která umožňuje komunikovat s X Serverem bez přesné znalosti X Protokolu. Tato knihovna je ovšem příliš složitá na to, aby se v ní dalo efektivně implementovat grafické rozhraní. Proto se společně s touto knihovnou již od začátku vyvíjí i další nástrojové sady, které mají zjednodušit návrh a implementaci uživatelského rozhraní. Příkladem může být například X Toolkit, který přináší další zjednodušení. I přes tato zjednodušení je X Toolkit stále příliš složitý a proto i nad ním existují nástavby jako je Motif, které představují více předdefinovaných prvků a další zjednodušení, takže vývoj aplikací s těmito nástroji je již v celku efektivní. Kromě X Toolkitu existují další toolkity jako GTK+ nebo Qt, které jsou základem dnešních velice úspěšných prostředí s pracovní plochou, které se snaží o jednotné chování a vzhled svých aplikací. Qt navíc obsahuje celou řadu funkcí, které nesouvisí pouze s uživatelským rozhraním, jako je práce po síti či komunikace s databázovým systémem. Podobnou rozšířenou funkčnost má i sada wxWidgets, která ovšem nepodporuje různá grafická témata. Za zmínku také stojí FLTK, který je koncipován tak, aby byl velice rychlý a nenáročný na místo na disku.

V další části jsem popsal antivirus, který je základem pro aplikaci s uživatelským rozhraním. Dále jsem popsal požadavky na toto grafické rozhraní vyplývající ze zamýšleného používání této aplikace. Poté jsem navrhl vzhled aplikace a rozložení jednotlivých prvků rozhraní. Mezi hlavní části rozhraní patří prvek, který zajišťuje procházení souborů a možnost přidání těchto souborů do fronty skenovaných souborů. Neméně důležitou roli hraje prvek, který uchovává tuto frontu souborů, kde se nachází nejen soubory pro skenování, ale také soubory, které již prošly procesem skenování. U těchto souborů se poté ukáže výsledek oskenování. Další důležitou součást tvoří dialog pro nastavení parametrů pro skenovací proces. V tomto dialogu lze nastavit vlastnosti velmi podobné těm z příkazové řádky. Také jsem uvedl výběr knihoven, které nejlépe splňují požadavky na tvorbu takovýchto doplňkových aplikací ke komerčním produktům, jsou to GTK+/GNOME a Qt/KDE.

V části, která patřila srovnávání implementace aplikace s uživatelským rozhraním pro

démona antiviru jsem rozdělil aplikaci na jednotlivé logické části, u kterých jsem důkladně probral rozdíly v implementaci těchto částí ve vybraných knihovnách. Je zde ukázáno, že mezi knihovnami jsou velké rozdíly při záměru prezentovat data ve stromové struktuře. Zatímco knihovny GTK+/GNOME používají systém model/zobrazení/ovládání, který odděluje data od zobrazovací části, v knihovnách Qt/KDE žádné takové rozdělení není. Velké rozdíly jsou i při používání více než jednoho vlákna v aplikaci. Knihovny GTK+/GNOME jsou buďto bezpečné pro použití v aplikaci s více vlákny, nebo poskytují globální zámek, který zaručí správnou funkčnost svých funkcí volaných z jiného než hlavního vlákna. Rozdíly lze také najít v přístupu k nastavování uživatelských preferencí, které aplikace nabízí. Knihovny GTK+/GNOME používají systém okamžitého projevování změn a posílání upozornění o změně preferencí, které daná aplikace využívá. U Qt/KDE knihoven se změna nastavení musí ještě potvrdit, aby bylo možné je uplatnit.

5.1 Budoucí vývoj

Jelikož v nedávné době vyšla nová verze knihoven Qt/KDE, již verze 4, která obsahuje velké rozdíly oproti původní verzi 3, bude nutné se tomuto přechodu přizpůsobit a upravit aplikaci s grafickým rozhraní pro démona antiviru v těchto knihovnách podle daných změn a využít tak plný potenciál, který nabízejí.

Vývoj je možný také přidáním dalších funkcí, které by uživatel mohl využívat při práci s implementovanou aplikací, jako například tisk fronty. Některé funkce, které byly prezentovány v návrhu aplikace, také ještě nejsou implementovány, protože firma Zoner Software, a.s. ještě nedodala adekvátní rozhraní použitelné pro přidání těchto funkcí do grafického rozhraní. Funkce, které nejsou ještě implementovány zahrnují spouštění, zastavování a aktualizaci antivirového démona přímo z grafického rozhraní.

Literatura

[1] The GUI Toolkit, Framework Page [online]. poslední revize 3.3.2002 [cit. 7.1.2009]. Dostupné na URL: http://home.pacbell.net/atai/guitool/

[2] Graphical User Interface Timeline [online]. [cit. 7.1.2009]. Dostupné na URL: http://toastytech.com/guis/guitimeline.html

[3] GUI Definition [online]. poslední revize 1.10.2004 [cit. 7.1.2009]. Dostupné na URL: http://www.linfo.org/gui.html

[4] GUI Toolkits for The X Window System [online]. poslední revize 27.7.2003 [cit. 7.1.2009]. Dostupné na URL: http://freshmeat.net/articles/view/928/

[5] An overview of the X toolkit, Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software., s.46-55. 17-19.10.1988. Alberta, Canada. Dostupné na URL: http://home.pacbell.net/atai/guitool/ [cit. 7.1.2009]

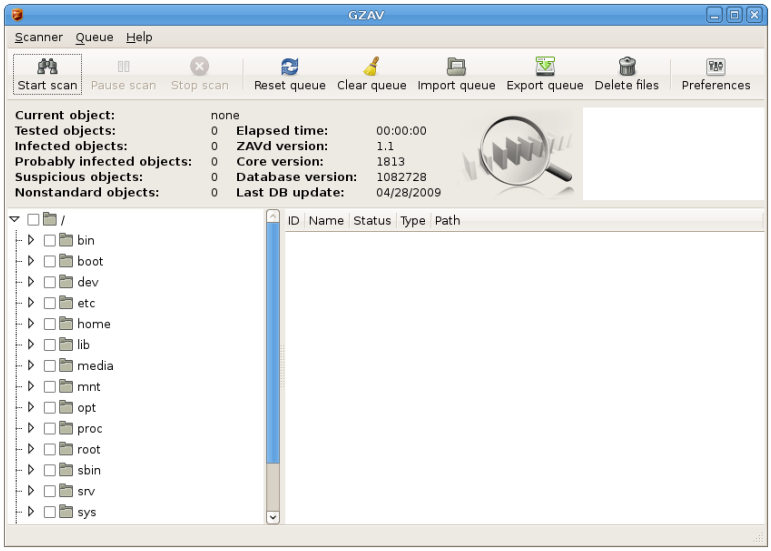
[6] Qt Online Reference Documentation [online]. [cit. 7.1.2009]. Dostupné na URL: http://doc.trolltech.com/

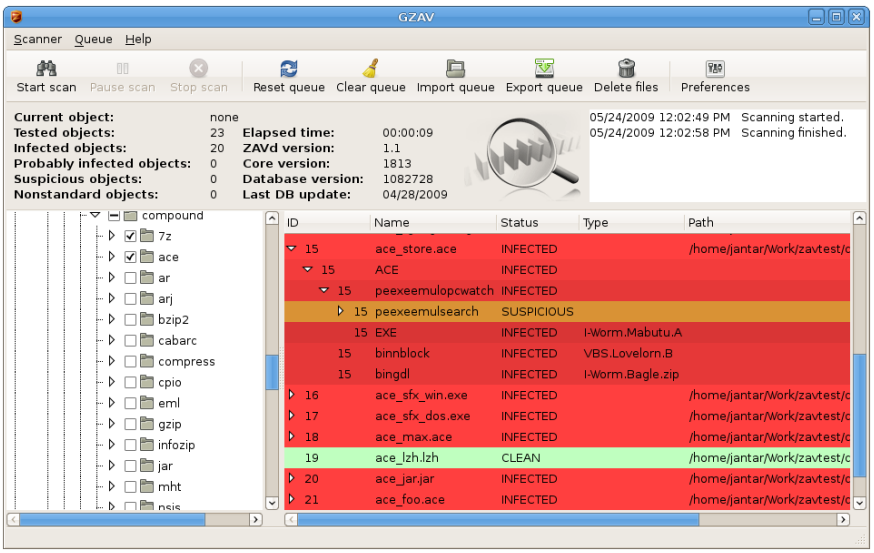
[7] The GTK+ Project [online]. [cit. 7.1.2009]. Dostupné na URL: http://www.gtk.org/

[8] wxwidgets [online]. [cit. 7.1.2009]. Dostupné na URL: http://www.wxwidgets.org/

Příloha A

Aplikace pro prostředí GNOME





Příloha B

Aplikace pro prostředí KDE