GUI aplikace pro správu Linuxových serverů/routerů

Jan Cholasta

Bakalářská práce
2009

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

ABSTRAKT

Cílem této práce bylo vytvořit softwarovou aplikaci pro vzdálenou správu zařízení používajících operační systémy na bázi Linuxu. V teoretické části se seznámíme s možnostmi konfigurace inicializačního procesu systému, hardwarových zařízení, sítí a také se současnými metodami vzdálené konfigurace systému. V praktické části je popsán návrh a implementace aplikace, včetně popisu použitých technologií.

Klíčová slova: Linux, administrace, vzdálená administrace, Ice, Qt.

ABSTRACT

The aim of this work is to create a software application for remote administration of devices that use Linux-based operating systems. In the theoretical part you'll be introduced to the means of configuration of the initialization process, hardware devices, networking and also to current methods of remote administration. There is a description of the design and implementation of the application in the practical part, as well as a description of the used technologies.

Keywords: Linux, administration, remote administration, Ice, Qt.

Chtěl bych poděkovat především mému vedoucímu práce, Ing. Tomáši Dulíkovi za jeho podporu a připomínky. Mé díky také patří mé přítelkyni Věře Budíkové za podporu a motivaci.

OBSAH

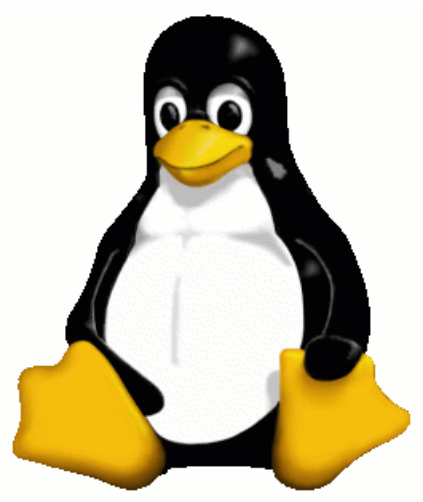
ÚVOD

Operační systém Linux se v současné době těší popularitě zejména v nasazení na serverech či v jednoúčelových embedded zařízení, jako jsou routery. Bohužel ale neexistuje žádný univerzální grafický nástroj pro pohodlnou vzdálenou administraci takových systémů, a tak správcům nezbývá, než systém spravovat vzdáleně pomocí příkazového řádku (pozn.: součástí této práce není polemika nad tím, jestli si správci s příkazovým řádkem vystačí nebo nikoliv – budeme předpokládat, že existují lidé, kterým tento způsob ovládání nevyhovuje). Tento přístup má své nevýhody (mimo řady výhod), což si uvědomují i výrobci konkurenčních produktů, kteří ke svým systémům většinou nějaký konfigurační software dodávají (příkladem budiž v České republice tolik populární MikroTik se svým WinBoxem [1]). Výsledkem této práce by měl být návrh a počáteční implementace aplikace, která tuto mezeru zaplní.

I. TEORETICKÁ ČÁST

1 OPERAČNÍ SYSTÉM LINUX

Operační systém Linux (přesněji GNU/Linux v případě většiny běžných linuxových distribucí, v embedded světě to nemusí platit [2]) začal vznikat již v roce 1991, kdy vyšel Linux (jádro systému) 0.01. Od té doby vzniklo mnoho tzv. *distribucí* (jádro + software), které se liší svým zaměřením jak na koncové zařízení, tak na znalosti uživatele.



Obr. 1: Tux, maskot Linuxu [3]

V dnešní době je Linux vyspělý operační systém vhodný k široké škále využití, od miniaturních embedded zařízení až po superpočítače. Rozšířil se zejména na serverech, v současnosti získává podíl na poli chytrých mobilních telefonů (*smartphones*) s projekty jako Maemo či Google Android.

1.1 D-Bus

V následujících kapitolách zmiňuji různé programy; mnoho z nich má jedno společné – nějakým způsobem využívají technologii D-Bus. Tato technologie je nedílnou součástí moderních linuxových OS, proto ji nesmíme opomenout.

D-Bus je jednoduchý IPC a RPC systém určený ke komunikaci mezi aplikacemi. Jeho návrh byl velmi ovlivněn systémem DCOP používaným v KDE 2 a 3, který byl v KDE 4 úplně nahrazen D-Busem. D-Bus je podporován na operačních systémech Linux (a dalších OS unixového typu), Windows a Mac OS X a jeho podpora je zabudovaná mj. v Qt 4, GNOME, Maemo či OLPC XO-1. V GNOME postupně nahrazuje dřívější Bonobo mechanismus. [4]

D-Bus je vyvíjen jako součást projektu freedesktop.org, který zastřešuje mnoho technologií zajišťujících interoperabilitu na poli pracovních prostředí pro X Window System (X11) na

Linuxu a jiných OS unixového typu. [5]

Sběrnice

Komunikace primárně probíhá přes centrální serverovou aplikaci (D-Bus daemon), tzv. *bus* (sběrnice), ale přímá komunikace aplikace s aplikací je také možná. Při komunikaci po sběrnici mohou aplikace získávat informace o tom, které jiné aplikace a služby jsou dostupné a případně je spouštět na požádání. [6]

V D-Busu existují dvě standardní sběrnice: system bus a session bus. System bus (systémová sběrnice) je sběrnice pracující na systémové úrovni, je dostupná všem aplikacím na daném stroji a může být použita pro oznamování systémových událostí (jako je např. přidání/odebrání HW, změny v tiskové frontě apod.) nebo pro komunikaci se systémovými službami. Session bus (sběrnice sezení) je specifická pro uživatelskou relaci – vzniká po přihlášení uživatele a zaniká před jeho odhlášením. [7]

Výpis 1: Instance D-Bus daemonu na linuxovém systému s jedním přihlášeným uživatelem

```
grubber@grubber:~$ ps -C dbus-daemon -o pid,user,command
  PID USER     COMMAND
 2104 dbus     /usr/bin/dbus-daemon --system
 2437 grubber  /usr/bin/dbus-daemon --fork --print-pid 6 --print-
address 9 --session
```

Aplikace

Aby mohla aplikace komunikovat přes sběrnici, je jí přidělen jeden nebo více jednoznačných názvů – povinný je jednoznačný název spojení (začíná dvojtečkou), další názvy jsou v režii aplikace a mají formu obrácených doménových jmen. D-Bus daemon tyto názvy používá k směrování zpráv z jedné aplikace k druhé (analogicky např. s počítačovou sítí).

Pokud spolu aplikace komunikují přímo, názvy se nepoužívají – jedná se o point-to-point komunikaci, takže je zřejmé, jaká aplikace je na druhé straně komunikačního kanálu.

Výpis 2: Názvy aplikací na systémové sběrnici linuxového systému

```
grubber@grubber:~$ dbus-send --system --dest=org.freedesktop.DBus
 --print-reply / org.freedesktop.DBus.ListNames
method return sender=org.freedesktop.DBus -> dest=:1.118 reply_se
rial=2
   array [
      string "org.freedesktop.DBus"
      string ":1.7"
      string ":1.118"
      string ":1.8"
      string ":1.9"
      string ":1.40"
      string ":1.41"
      string ":1.42"
      string "org.bluez"
      string ":1.10"
      string ":1.99"
      string "org.freedesktop.ConsoleKit"
      string "org.freedesktop.DeviceKit"
      string ":1.12"
      string ":1.13"
      string ":1.0"
      string ":1.1"
      string "org.x.config.display0"
      string ":1.37"
      string ":1.15"
      string "org.freedesktop.Hal"
      string "com.redhat.NewPrinterNotification"
      string ":1.38"
      string ":1.16"
      string ":1.39"
      string ":1.17"
      string ":1.90"
      string ":1.18"
      string ":1.5"
      string "org.freedesktop.DeviceKit.Power"
      string ":1.6"
   ]
```

Speciálním případem aplikací jsou služby – jedná se o aplikace, které lze spustit na požádání. Služby se spouští podle jejich názvu na sběrnici. K nalezení spustitelného souboru dané služby slouží tzv. soubory popisu služby – v tomto souboru je vedle názvu služby cesta ke spustitelnému souboru společně s argumenty.

Výpis 3: Ukázka souboru popisu služby

```
[D-BUS Service]
Name=org.openobex
Exec=/usr/bin/obex-data-server --no-daemon
```

Na každé sběrnici existuje standardní služba *org.freedesktop.DBus*, která slouží k připojení aplikací k sběrnici, přidělování názvů a spouštění služeb na požádání.

Objekty

Aplikace poskytují specifické služby jiným aplikacím pomocí objektů. Tyto objekty tvoří hierarchickou strukturu, ne nepodobnou souborům v souborovém systému (s tím rozdílem, že se nerozlišuje mezi soubory a adresáři). Každý objekt má svůj název a k jeho nalezení slouží cesta, která je tvořena názvy objektů oddělenými lomítkem (např. */org/freedesktop/Hal/Manager*).

Rozhraní

Každý objekt implementuje jedno nebo více rozhraní, která definují metody, signály a vlastnosti dostupné aplikacím pro práci s daným objektem.

V D-Busu jsou definovaná 3 standardní rozhraní, která implicitně implementuje každý objekt. Prvním z nich je *org.freedesktop.DBus.Peer*, které definuje 2 metody, *org.freedesktop.DBus.Peer.Ping*, která vrací prázdný výsledek a *org.freedesktop.DBus.Peer.GetMachineId*, která vrací unikátní identifikátor stroje, na kterém se objekt nachází. Dalším standardním rozhraním je *org.freedesktop.DBus.Introspectable*, které definuje jedinou metodu, která vrací XML popis objektu. Třetím a posledním standardním rozhraním je *org.freedesktop.DBus.Properties*, které definuje metody pro práci s vlastnostmi.

Výpis 4: Ukázka výstupu org.freedesktop.DBus.Introspectable.Introspect

```
grubber@grubber:~$ dbus-send --system --dest=com.redhat.NewPrinter
Notification --print-reply /com/redhat/NewPrinterNotification org.
freedesktop.DBus.Introspectable.Introspect
method return sender=:1.38 -> dest=:1.114 reply_serial=2
   string "<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object
Introspection 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node name="/com/redhat/NewPrinterNotification">
  <interface name="com.redhat.NewPrinterNotification">
    <method name="GetReady">
    </method>
    <method name="NewPrinter">
      <arg direction="in"  type="i" name="status" />
      <arg direction="in"  type="s" name="name" />
      <arg direction="in"  type="s" name="mfg" />
      <arg direction="in"  type="s" name="mdl" />
      <arg direction="in"  type="s" name="des" />
      <arg direction="in"  type="s" name="cmd" />
    </method>
  </interface>
  <interface name="org.freedesktop.DBus.Introspectable">
    <method name="Introspect">
      <arg direction="out" type="s" />
    </method>
  </interface>
</node>
"
```

Zprávy

Aplikace spolu komunikují přes D-Bus posíláním zpráv. Jejich pomocí si předávají vzdálená volání procedur a jejich výsledky či chybové stavy. Při komunikaci přes sběrnici mají zprávy daný cíl, což znamená, že jsou směrovány jen relevantním aplikacím – zabraňuje to zahlcování aplikací zprávami, jak by se dělo v případě rozesílání zpráv všem aplikacím na sběrnici.

Speciálním případem zpráv jsou tzv. signály – na rozdíl od běžných zpráv nemají signály definován cíl a nevrací výsledek. Slouží ke komunikaci od jednoho zdroje k mnoha cílům.

2 INICIALIZACE SYSTÉMU

Start typického linuxového systému začíná zavedením jádra a volitelně počátečního RAM disku (*initrd*) některým ze zavaděčů (nejčastěji se setkáme se zavaděči GRUB a LILO). Jádru můžeme ze zavaděče předat různé parametry, jako např. na jakém zařízení se nachází kořenový souborový systém, nebo jestli potlačit textový výstup jádra. [8]

Pokud je přítomen počáteční RAM disk, připojí se a slouží jako dočasný kořenový souborový systém. Počáteční RAM disk obsahuje minimální systém nutný ke spuštění opravdového systému. Jeho hlavními úkoly je nahrát všechny potřebné ovladače HW, připojit kořenový souborový systém (který se může nacházet na zašifrovaném disku, v RAID poli, LVM svazku nebo na síti) a spustit z něj systém. [9]

Spuštění uživatelské části systému má na starost program *init*. Ten spouští další procesy, které se starají o inicializaci HW, připojení systému do sítě a spouštění služeb. Existuje mnoho variací na program init, které se liší svým přístupem k řešení problému inicializace systému. Některé z nich si popíšeme v následujících kapitolách.

2.1 Tradiční metody inicializace systému

V dobách tradičních unixových systémů (80. léta minulého století) se používal init založený na jednoduchém spouštění shellových skriptů, které se staraly o spouštění jednotlivých procesů. V současné době se nejvíce používají dva takové init systémy, *BSD-style init* a *System V init*. [10]

2.1.1 BSD-style init

Jak napovídá název, BSD-style init vznikl v rámci BSD UNIXu – používá se v jeho derivátech a také v některých linuxových distribucích (např. Slackware).

Tento init systém spouští inicializační shell skript „/etc/rc“ a obsluhu textových terminálů a X Window System na terminálech grafických. Na rozdíl od System V initu v BSD-style initu neexistuje koncept runlevelů – všechna zodpovědnost spadá na rc skript.

Výhodou tohoto init systému je jeho jednoduchost, nevýhodou je, že pokud chce nějaký balíček třetí strany ovlivnit průběh inicializace, musí zasahovat do existujících inicializačních skriptů, což může v případě chyby vést až k nespustitelnému systému. V dnešní době je tento nedostatek odstraněn použitím souboru „rc.local“, který se spouští ve vlastním shellu ke konci bootovací sekvence, nebo pomocí adresáře „rc.d“, kam mohou

balíčky instalovat své vlastní nezávislé start/stop skripty (koncept podobný System V initu).

2.1.2 System V init

System V init je založen na konceptu runlevelů. Tento init systém se v současnosti používá ve většině linuxových distribucí.

Runlevel popisuje určitý stav stroje, charakterizovaný spouštěnými procesy. Obecně existuje 8 runlevelů: runlevely 0 až 6 a runlevely *S* a *s* (které ukazují na jeden runlevel). Tři z těchto runlevelů jsou rezervované:

- runlevel 0 – vypnutí stroje,
- runlevel 1 – jednouživatelský mód,
- runlevel 6 – restart stroje.

Další runlevely interpretují různé varianty UNIXu a OS unixového typu různě. Typický linuxový systém interpretuje další runlevely takto:

- runlevely 2 až 4 – víceuživatelský mód,
- runlevel 5 – víceuživatelský mód s GUI (X11).

Nastavení System V initu se nachází v souboru „/etc/inittab“. V tomto souboru je mj. specifikováno, co se bude při kterém runlevelu spouštět a který runlevel je výchozí. Runlevel můžeme změnit za běhu systému příkazem *init*.

2.2 Moderní metody inicializace systému

Moderní varianty programu init se snaží o zrychlení procesu inicializace systému na maximum. Jejich společným znakem je asynchronní spouštění procesů, což umožňuje významné zrychlení inicializace zejména na současných víceprocesorových systémech.

2.2.1 Initng

Initng (init nové generace) je navržen jako náhrada System V initu, ale není s ním zcela zpětně kompatibilní. Ke konfiguraci spouštěných procesů používá vlastní formát souboru, tzv. *iFile*. [11]

Initng je založen na systému pluginů – samotný initng se stará v podstatě jen o načítání pluginů a zajištění komunikace mezi nimi. Existuje mnoho pluginů, od esenciálních např.

pro načítání iFile souborů až po estetické záležitosti, jako je podpora pro grafické ukazatele průběhu inicializačního procesu. [12]

Výpis 5: Ukázka iFile souboru (web server lighttpd)

```
# NAME: lighttpd
# DESCRIPTION: Very high performance web server.
# WWW: http://www.lighttpd.net/

daemon daemon/lighttpd {
    need = system/bootmisc virtual/net;
    use = system/modules system/coldplug;
    exec daemon = /usr/bin/lighttpd -D -f /etc/lighttpd/lighttpd
.conf;
}
```

Initng není v současné době používán jako výchozí init systém v žádné distribuci (i když řada distribucí nabízí hotové balíčky). Největší překážkou v jeho nasazení je patrně omezená zpětná kompatibilita se skripty System V initu.

2.2.2 Upstart

Upstart je init systém založený na událostech. Procesy jsou spouštěny/zastavovány v závisloti na událostech, které mohou vznikat spuštěním/zastavením jiných procesů, nebo být vyvolány na žádost jakéhokoliv jiného procesu. Komunikace s Upstart daemonem je zajištěna prostřednictvím technologie D-Bus.

Upstart je od začátku navržen tak, aby byl plně zpětně kompatibilní se System V initem a aby byl přechod z něj co nejsnadnější – v tomto se liší od většiny ostatních moderních init systémů, které vyžadují úplný přechod a nepodporují kombinaci starých a nových spouštěcích metod. Mimoto Upstart používá ke konfiguraci také svůj vlastní formát souboru, tzv. *job file*. [13]

Výpis 6: Ukázka job file (web server lighttpd)

```
#this is an upstart script that starts lighttpd

start on runlevel 2
start on runlevel 3
start on runlevel 4
start on runlevel 5

stop on runlevel 0
stop on runlevel 1
stop on runlevel 6

respawn
exec sudo -u www-data lighttpd -D -f /etc/lighttpd/lighttpd.conf
```

Upstart má ambice stát se náhradou nejen System V initu, ale také plánovacích utilit jako je *cron* a *atd* – bude toho docíleno možností generovat události v závislosti na čase či časových intervalech. [14]

V současné době je Upstart používán jako výchozí init systém v distribucích Fedora a Ubuntu a balíčky existují pro řadu dalších distribucí.

3 SPRÁVA ZAŘÍZENÍ

V unixových systémech jsou zařízení představována speciálními soubory, tzv. soubory zařízení (*device files*). Soubory zařízení se nachází v adresáři /dev. Označují fyzická či virtuální zařízení pomocí dvojice čísel, tzv. *major number* a *minor number* (obě jsou součástí struktury každého souboru zařízení). V tradičních OS unixového typu jsou soubory zařízení v adresáři /dev uloženy staticky. Není výjimkou, když se v takovém adresáři nachází tisíce souborů zařízení. [15]

V Linuxu řady 2.4 a starších byla major i minor čísla v rozmezí 0 až 255. V Linuxu řady 2.6 jsou major čísla v rozmezí od 0 do 4096, minor čísel může být přes milion. Počet možných kombinací vyrostl natolik, že statické ukládání přestalo být přijatelné a byly vyvinuty sofistikovanější metody správy souborů zařízení.

3.1 devfs a hotplug

devfs je virtuální filesystém, určený k prezentaci souborů zařízení jen pro ta zařízení, která jsou opravdu připojena. Problém, kdy který soubor zařízení zobrazit je řešen na úrovní ovladačů zařízení – ovladače po připojení zařízení zasílají devfs požadavek o přidání souboru zařízení a po odpojení zařízení požadavek o jeho odstranění. Výhodou tohoto přístupu je jeho jednoduchost, nevýhodou provádět složitější úkony.

Nevýhodu devfs v nemožnosti reagovat na připojení zařízení ničím jiným, než vytvořením příslušného souboru zařízení do určité míry vyřešil *hotplug* mechanismus. Tento mechanismus funguje tak, že jádro při připojení či odpojení zařízení zavolá userspace program (ve většině případů */sbin/hotplug*), který se o reakce na tyto události postará. Jádro hotplugu předá parametry zařízení (pomocí proměnných prostředí) a ten pak může reagovat např. připojením/odpojením filesystému na USB zařízení. [16]

3.2 udev

udev je správce zařízení pro řadu 2.6 jádra Linuxu. Je to nástupce starších systémů devfs a hotplug. Jeho primární zodpovědností je správa souborů zařízení v adresáři /dev a obsluha událostí spojených s přidáváním/odebíráním zařízení, jako je např. nahrávání firmwaru. Poslední verze udevu závisí na poslední verzi rozhraní uevent, které se poprvé objevilo v Linuxu 2.6.13. [17]

Oproti výše zmíněným systémům má udev tu výhodu, že podporuje persistentní pojmenovávání souborů zařízení – např. pojmenování souboru zařízení disků nezávisí na

pořadí, v jakém byly připojeny, ale na unikátním identifikátoru filesystému, který se na něm nachází, jménu disku nebo jeho fyzickém umístění na HW, ke kterému je připojen. Další výhodou je, že udev běží jako daemon v userspace, takže může k pojmenování zařízení použít výstup z jakéhokoliv programu.

Co a jak udev pojmenuje je dáno sadou pravidel, která se nachází v konfiguračních souborech v adresáři */etc/udev/rules.d*. Tato pravidla mohou využívat k pojmenování souborů zařízení informace z jádra (např. jaderný subsystém nebo sériové číslo zařízení) nebo od jiných programů (např. pro zjištění názvu zařízení).

Výpis 7: Ukázka souboru s pravidly (device-mapper.rules)

```
ACTION=="add|change", SUBSYSTEM=="block", KERNEL=="dm-[0-9]*", P
ROGRAM="/sbin/dmsetup info -c --noopencount --noheadings -o name
 -j %M -m %m", NAME="mapper/%c", MODE="0600", SYMLINK+="disk/by-
name/%c"
```

3.3 HAL

HAL je projekt poskytující vrstvu abstrakce hardwaru (*hardware abstraction layer*) pro OS unixového typu. Jeho hlavním cílem je umožnit uživatelským aplikacím zjišťovat přítomnost HW a umožnit jeho použití nezávisle na použité platformě. HAL je vyvíjen v rámci projektu freedesktop.org. [18]

HAL poskytuje svoje API přes D-Bus, takže komunikovat s ním může jakákoliv aplikace tuto technologii využívající. HAL exportuje dva druhy objektů: *manager* a *device*. Manager poskytuje metody pro enumeraci zařízení a pro jejich přidávání a odebírání spolu s patřičnými signály. Device představuje zařízení – každý objekt zařízení má jednoznačný identifikátor (*UDI*), sadu vlastností popisujících toto zařízení a implementuje sadu rozhraní odpovídajících jeho funkci. Zařízení jsou organizována v hierarchii, na jejím vrcholu je zařízení */org/freedesktop/Hal/devices/computer* (které představuje počítač) a všechna další zařízení jsou jeho potomky (přímými či nepřímými). [19]

Výpis 8: Vlastnosti bezdrátového síťového rozhraní

```
grubber@grubber:~$ lshal -u /org/freedesktop/Hal/devices/net_00_1
b_77_12_34_56
udi = '/org/freedesktop/Hal/devices/net_00_1b_77_12_34_56'
  info.capabilities = {'net', 'net.80211'} (string list)
  info.category = 'net.80211'  (string)
  info.parent = '/org/freedesktop/Hal/devices/pci_8086_4227'  (st
ring)
  info.product = 'WLAN Interface'  (string)
  info.subsystem = 'net'  (string)
  info.udi = '/org/freedesktop/Hal/devices/net_00_1b_77_12_34_56'
  (string)
  linux.hotplug_type = 2  (0x2)  (int)
  linux.subsystem = 'net'  (string)
  linux.sysfs_path = '/sys/devices/pci0000:00/0000:00:1c.1/0000:0
3:00.0/net/wlan0'  (string)
  net.80211.mac_address = 117964896041  (0x1b77417729)  (uint64)
  net.address = '00:1b:77:12:34:56'  (string)
  net.arp_proto_hw_id = 1  (0x1)  (int)
  net.interface = 'wlan0'  (string)
  net.linux.ifindex = 4  (0x4)  (int)
  net.originating_device = '/org/freedesktop/Hal/devices/pci_8086
_4227'  (string)
```

V současné době se pracuje na následníkovi HALu, *DeviceKit*u. Má odstranit některé nedostatky a chyby v návrhu HALu, jako je nedostatečná modularita nebo příliš velká úroveň abstrakce. [20]

4 KONFIGURACE SÍTĚ

Konfigurace sítě a síťových zařízení je jedna z věcí, které se liší takřka distribuce od distribuce. Některé nejznámější metody konfigurace si popíšeme v následujících kapitolách.

4.1 Konfigurační skripty

Klasickým (a nejběžnějším) přístupem ke konfiguraci síťových zařízení je použití shellových skriptů. Ve většině případů tyto skripty zpracovávají jeden nebo více konfiguračních souborů.

4.1.1 Debian

O nastavení síťových rozhraní se v Debianu a odvozených distribucích starají programy *ifup* a *ifdown* s pomocí skriptů z adresáři */etc/network*. Hlavní slovo má soubor */etc/network/interfaces*, ve kterém se nachází konfigurace všech síťových rozhraní.

Výpis 9: Ukázka souboru interfaces

```
# Used by ifup(8) and ifdown(8). See the interfaces(5) manpage or
# /usr/share/doc/ifupdown/examples for more information.

# loopback
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 192.168.1.12
        netmask 255.255.255.0
        gateway 192.168.1.1
```

4.1.2 Red Hat

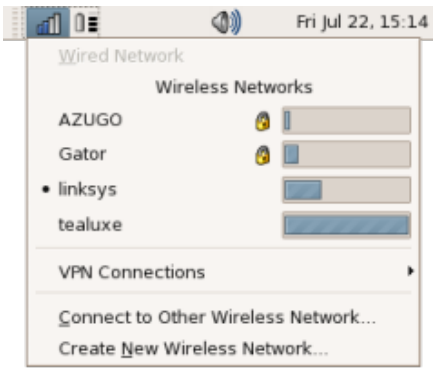
V Red Hatu a odvozených distribucích se o konfigurace sítě stará sada skriptů v adresáři */etc/sysconfig/network-scripts*. Nastavení jednotlivých síťových rozhraní se nachází v konfiguračních souborech */etc/sysconfig/network-scripts/ifcfg-zařízení*. Existuje jeden takový soubor na jedno zařízení.

Výpis 10: Ukázka souboru ifcfg

```
DEVICE=eth0
IPADDR=192.168.1.12
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

4.2 NetworkManager

Moderní desktopové distribuce používají ke konfigurace sítě *NetworkManager*. Cílem tohoto projektu je co nejvíce zjednodušit nastavení sítě (především bezdrátového spojení) běžným uživatelům. NetworkManager podporuje nastavení připojení pro velkou škálu zařízení (ethernet, wifi, bluetooth, modemy, ...), statickou i dynamickou konfigurace těchto zařízení a síťové profily. NetworkManager automaticky přepíná mezi dostupnými připojeními tak, aby bylo vždy aktivní to nejlepší možné. Je postaven na technologiích D-Bus a HAL. Vzhledem k jeho povaze je nasazení na serverech a podobných zařízeních nevhodné. [21]



Obr. 2: NetworkManager [22]

5 VZDÁLENÁ SPRÁVA SYSTÉMU

V této kapitole si popíšeme nejvíce používané metody vzdálené konfigurace linuxových systémů.

5.1 Vzdálený terminál

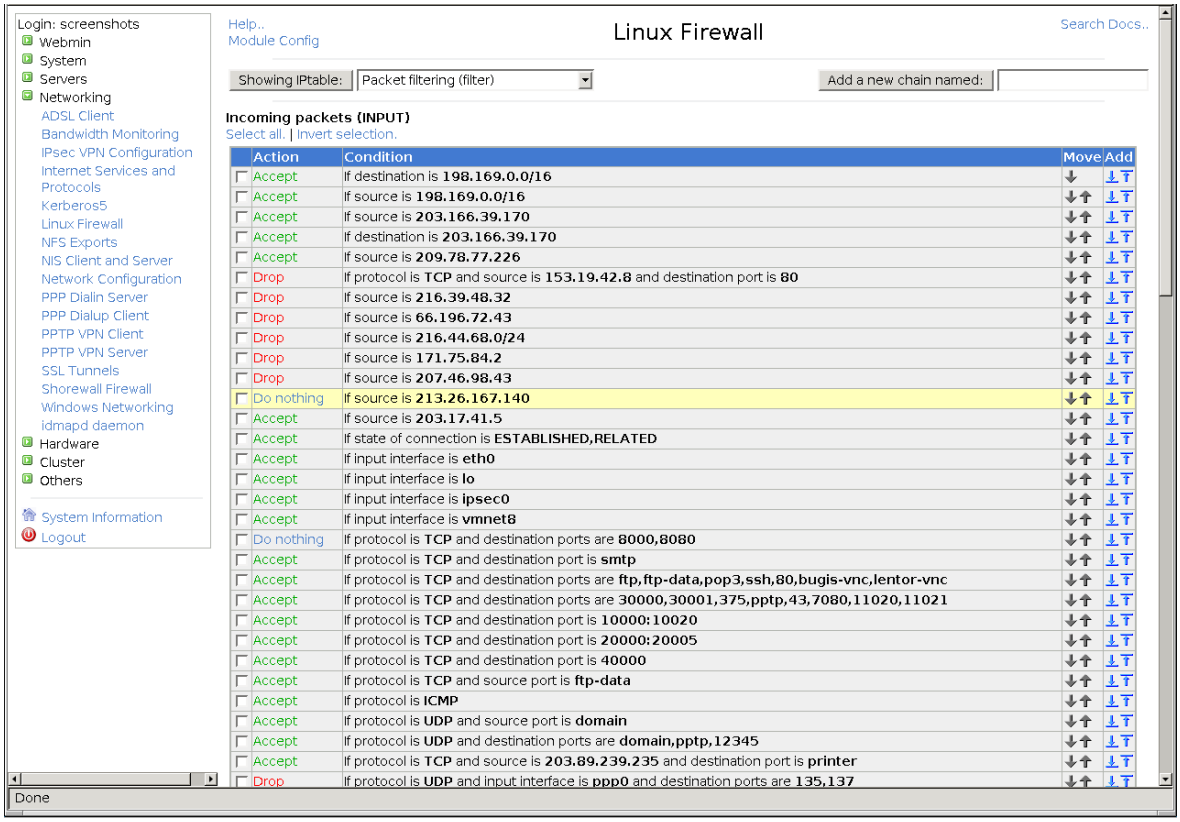
Zřejmě nejstarší metodou vzdálené konfigurace unixových systémů je konfigurace přes vzdálený terminál. Tímto způsobem se se systémem pracuje stejně, jako na fyzickém terminálu – zadáváním příkazů do příkazového řádku (pomineme-li vzdálený přístup ke grafickému terminálu).

Nejznámějšími protokoly pro vzdálený přístup k příkazové řádce jsou telnet a SSH. *Telnet* nabízí jen jednoduchou komunikaci bez šifrování a tak je jeho použití omezeno jenom na bezpečné kanály a lokální sítě (správa serveru přes Internet tady rozhodně nepatří) [23]. *SSH* je daleko sofistikovanější protokol zahrnující šifrovaní, vhodný i pro komunikaci přes Internet. [24]

Nevýhodami tohoto přístupu je malá přehlednost a velká náchylnost k chybám (někdy stačí překlep v příkazu k znefunkčnění celého systému).

5.2 Webové konfigurátory

Do této kategorie spadají projekty jako je např. WebMin nebo konfigurační rozhraní mnoha specializovaných firewallových/routerových distribucí (IPCop, ClarkConnect, Endian Firewall, apod.). Jde o aplikace postavené na webových technologiích – v systému běží webový server, který zpřístupňuje konfigurační aplikaci napsanou ve skriptovacím jazyce (Perl, PHP, apod.), uživatelské rozhraní je tvořeno pomocí webových technologií (HTML, CSS, JavaScript/AJAX, atd.) a zobrazeno ve webovém prohlížeči.



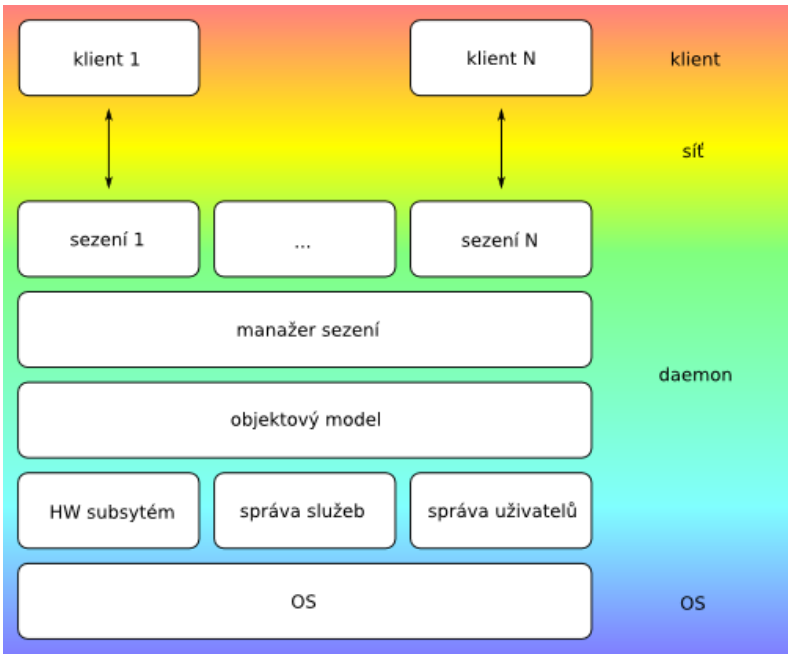
Obr. 3: Webmin [25]

Tento přístup má výhodu automatické portability (uživatelské rozhraní běží všude, kde běží webový prohlížeč), ale naráží na „těžkopádnost“ webových technologií a malou flexibilitu uživatelského rozhraní – nelze např. plynule zobrazovat průběhy statistických dat.

II. PRAKTICKÁ ČÁST

6 NÁVRH APLIKACE

Aplikace je logicky rozdělená na dvě části: serverovou část a klientskou část. Serverová část běží jako daemon na systému, který chceme spravovat. Klientská část je GUI aplikace, kterou pak mohou správci vzdáleně přistupovat k serverové části.



Obr. 4: Blokové schéma architektury aplikace

Serverová část aplikace poskytuje služby klientům prostřednictvím objektů. Objekty představují logické součásti systému, jako jsou uživatelé, síťová rozhraní, služby, atp. Každý objekt je jednoznačně identifikován pomocí UUID (128 bitový náhodný identifikátor [26]). Použití UUID zaručuje velmi nízkou pravděpodobnost kolize identifikátorů i mezi různými servery. Každý objekt může také mít textový název a rodiče (neplatí pro kořenový objekt) – objekty lze tedy kromě UUID identifikovat i názvem a cestou (v tomto případě ale není zaručena jednoznačnost identifikace).

Objekty k OS přistupují pomocí subsystémů. Mezi základní typy subsystémů patří: HW subsystém, který slouží k enumeraci HW a jeho konfiguraci, uživatelský subsystém, který slouží ke správě uživatelských účtů a subsystém služeb, který se stará o spouštění a zastavování služeb. Subsystém jsou implementovány pomocí konkrétních backendů, např. k uživatelskému subsystému mohou existovat backendy pro použití běžných systémových účtů nebo pro použití jednoduchých účtů uložených v databázi. Nové typy subsystémů a backendy lze do serveru přidávat formou pluginů.

Operace s objekty lze provádět pomocí metod. Každý typ objektu může mít definovanou jednu nebo více metod. Metody mají název a sadu vstupních a výstupních parametrů

daných názvem a datovým typem.

Ke správné funkci aplikace je potřeba mnoho typů objektů, mezi ty nejzákladnější patří:

- *Object* – je kořenový typ všech typů objektů. Definuje pseudometodu (pseudometodu proto, že se s ní nezachází jako s běžnou metodou) *View*, která se používá v oprávněních k omezení viditelnosti objektu.
- *ServerRoot* – představuje kořenový objekt serveru. Objekt tohoto typu existuje na každém serveru pouze v jediné instanci.
- *Host* – představuje kořenový objekt HW. Jejich počet není omezen, ale ve většině případů bude na jednom serveru pouze jediná instance představující stroj, na kterém server běží. Tento typ objektu má definovány 2 metody: *Shutdown* pro vypnutí stroje a *Reboot* pro jeho restart.
- *Property* – je objekt představující vlastnost nadřazeného objektu – např. objekt typu *Host* může mít vlastnost *hostname*, představující jméno daného stroje. Tento typ objektu má definovány 2 metody: *Read* pro přečtení hodnoty vlastnosti a *Write* pro nastavení hodnoty vlastnosti.

Další typy objektů závisí na implementaci. Nové typy lze do serveru přidávat formou pluginů.

Komunikaci s klienty diriguje manažer sezení. Když se klient připojí k serveru, tak se musí identifikovat – pokud identifikace proběhne úspěšně, manažer sezení vytvoří pro tohoto klienta nové sezení. Klient potom se serverem komunikuje skrz toto sezení. Pomocí sezení se aplikují oprávnění daného uživatele – pokud se klient pokusí provést operaci, k níž nemá oprávnění, bude tato operace odfiltrována v rámci sezení a ke konkrétnímu objektu nikdy nedorazí.

Oprávnění uživatelů jsou řešena formou kontroly přístupu na bázi rolí (*Role-based access control* [27]). Každý uživatel má přiřazen seznam rolí, které určují jeho pravomoce. Každá role má definovánu sadu oprávnění k provádění operací s objekty nebo množinami objektů. Role mohou dědit oprávnění od jiných rolí.

Vyhodnocení způsobilosti uživatele k provedení konkrétní operace probíhá role po roli, oprávnění po oprávnění. Výchozím stavem je zákaz provedení jakékoliv operace. Existují 3 typy oprávnění:

- *allow* – povol operaci a pokračuj ve vyhodnocování oprávnění,

- *deny* – zakaž operaci a pokračuj ve vyhodnocování oprávnění,
- *always deny* – zakaž operaci a ukonči vyhodnocování oprávnění.

Veškeré informace o objektech včetně jejich stavu se ukládají do databáze, ze které se po restartu serveru obnovují.

Klientská část slouží jenom jako koncový bod pro uživatele. Jejím úkolem je přehlednou formou zobrazit dostupné objekty na serveru a umožnit na nich provádět operace. Hlavním požadavkem na klientskou část je podpora více platforem.

7 POUŽITÉ TECHNOLOGIE

K usnadnění a zrychlení implementace aplikace je vhodné (v některých případech dokonce nutné) použít existující technologie.

Použité technologie musí splňovat tyto podmínky:

- podpora pro více platforem – důležitá je podpora nejen pro Linux, na kterém poběží serverová část programu, ale také pro desktopové platformy, na kterých poběží klientská část programu (především Linux a Windows),
- podpora C++ – celá aplikace bude implementována v jazyce C++,
- open-source licence – budoucí přijetí programu do distribucí bude ve většině případů ovlivněno použitou licencí (ideálně GNU GPL).

7.1 Ice

Internet communications engine (zkráceně *Ice*) je objektově orientovaná middleware platforma, která podporuje vzdálené volání procedur (RPC), grid computing a publish/subscribe funkcionalitu. Ice je vyvíjen společností ZeroC a duálně licencován pod GNU GPL a proprietární licencí. Podporuje programovací jazyky C++, Java, .NET, Visual Basic, Python, Ruby a PHP na řadě operačních systémů, jako např. Linux, Solaris, Windows a Mac OS X. Existuje také odlehčená varianta, *Ice-e*, která běží na malých zařízeních, jako jsou mobilní telefony. Ice může být použit pro internetové aplikace bez potřeby použití HTTP protokolu a na rozdíl od většiny ostatních middleware řešení je schopný procházet skrz firewally. [28]

Ice byl zvolen jako prostředek pro komunikaci zejména z těchto důvodů:

- splňuje podmínky podpory více platforem a open-source licence,
- jednoduchost – základy práce s Ice si lze osvojit velmi rychle (v porovnání s jinými middleware řešeními) a celková čistota návrhu napomáhá vyhnout se chybám,
- rychlost – Ice používá rychlý binární protokol, díky čemuž ho lze použít pro komunikaci i na velmi pomalých linkách (např. vytáčené spojení),
- podpora malých zařízení – možnost v budoucnu portovat klientskou část na tento typ zařízení,
- další možnosti – potenciálně užitečné pro další verze programu (např. podpora pro grid computing),

- kvalita dokumentace – rozsáhlá a úplná dokumentace včetně názorných příkladů,
- vyvíjen komerční společností – zajištění životnosti projektu i do budoucna.

7.1.1 Architektura

Ice je objektově orientovaná middleware platforma. V základu to znamená, že Ice poskytuje nástroje, API a knihovny pro tvorbu objektově orientovaných klient – server aplikací. Ice aplikace jsou vhodné pro použití v heterogenních prostředích, klient a server mohou být napsány v odlišných programovacích jazycích, mohou běžet na odlišných operačních systémech na odlišných HW platformách a mohou komunikovat pomocí různých síťových technologií. [29]

Klient a Server

Termíny klient a server nejsou pevným určením funkce částí aplikace, ale popisují úlohu části aplikace v době zpracování požadavku. Klient je aktivní entita, která zasílá požadavky na služby serveru. Server je pasivní entita, která poskytuje služby odpovědí na požadavek klienta.

V reálném světě server většinou není jenom server (ve smyslu, že pouze odpovídá na požadavky klienta), ale sám je klientem jiných serverů, aby mohl splnit požadavky svého klienta. Podobně klient není většinou jenom klient, ale také odpovídá na požadavky jiných klientů. Tato výměna rolí je v mnoha systémech běžná – takové systémy lze lépe popsat jako *peer-to-peer* systémy.

Objekty

Objekt je entita v lokálním nebo vzdáleném adresovém prostoru, která odpovídá na požadavky klienta. Instance objektu může existovat na jednom nebo redundantně na více serverech – v takovém případě se stále jedná o jeden objekt. Každý objekt implementuje jedno nebo více rozhraní.

Rozhraní (*interface*) je soubor pojmenovaných operací, které jsou objektem podporovány. Klienti vytváří požadavky voláním operací. Operace může mít parametry a návratovou hodnotu. Parametry a návratové hodnoty jsou dány typem, parametry jsou pojmenované a mají směr: vstupní parametry jsou inicializovány klientem a předány serveru, výstupní parametry jsou inicializovány serverem a předány klientovi (návratová hodnota je jednoduše speciální případ výstupního parametru).

Každý objekt má jedno hlavní rozhraní (*main interface*) a může mít více (nebo žádné) doplňujících rozhraní (*facets*). Klienti si mohou vybrat, se kterým z dostupných rozhraní budou pracovat.

Každý objekt má unikátní identitu (*object identity*). Identita objektu slouží k rozeznání objektu od ostatních objektů. Ice předpokládá, že identita objektu je globálně unikátní v rámci jedné komunikační infrastruktury.

Proxy

Aby klient mohl kontaktovat objekt, musí vytvořit *proxy* pro tento objekt. Proxy je entita lokální klientovi, která reprezentuje objekt (ať už lokální či vzdálený). Když klient zavolá na proxy operaci, Ice se postará o aktivaci serveru, na kterém se objekt nachází, stejně jako o aktivaci samotného objektu a zavolá na něm požadovanou operaci. Proxy do sebe integruje všechny potřebné informace o objektu, jako adresu serveru, na kterém se objekt nachází a identitu objektu.

Sluhové

Objekty jsou konceptuální entity, které mají typ, identitu a adresovací informace. Požadavky klientů ale musí končit u konkrétní entity na straně serveru, která požadavek zpracuje. Touto entitou je sluha (*servant*).

Sluha je instance třídy, která implementuje metody korespondující s operacemi, které jsou daným objektem podporovány. Sluha může vyřizovat operace jednoho nebo i více objektů. Naopak k jednomu objektu může být přiřazeno více sluhů, pokud tento objekt existuje na více serverech (konkrétní operaci pak vždy vyřizuje jeden z těchto sluhů, podle toho, na který server byl požadavek zaslán).

7.1.2 Součásti

Ice nabízí řadu nástrojů a služeb – bez některých z nich se žádná aplikace neobejde, jiné slouží specifickým účelům.

Slice

Slice je IDL jazyk/kompilátor pro Ice. Slouží k popisu všech rozhraní, operací a datových typů, které si klient a server mezi sebou vyměňují. Slice umožňuje definovat klient/server kontrakt způsobem nezávislým na programovacím jazyce. Slice definice jsou potom kompilátorem převedeny do podoby API pro konkrétní programovací jazyk.

Výpis 11: Ukázka Slice definice

```
module Filesystem {
    interface Node {
        idempotent string name();
    };

    exception GenericError {
        string reason;
    };

    sequence<string> Lines;

    interface File extends Node {
        idempotent Lines read();
        idempotent void write(Lines text) throws GenericError;
    };

    sequence<Node*> NodeSeq;

    interface Directory extends Node {
        idempotent NodeSeq list();
    };
};
```

Freeze

Freeze je služba Ice pro persistenci objektů – s Freeze je jednoduché uložit stav objektu do databáze. Stav, který se bude pro daný objekt ukládat, se definuje ve Slice a Freeze překladač z této definice vygeneruje kód, který se o vlastní ukládání a načítání postará. Freeze jako databázi používá Berkeley DB.

Ice také poskytuje utilitu *FreezeScript*, která usnadňuje správu databáze a migraci existující databáze na nové schéma, pokud se změní definice objektů.

IceGrid

IceGrid je služba, která umožňuje registrovat servery pro spuštění na požádání, poskytuje nástroje pro snadnou konfiguraci komplexních aplikací s více servery s podporou pro replikaci a rozložení zátěže.

IceBox

IceBox je aplikační server, který obsluhuje spouštění a zastavování komponent aplikace. Jednotlivé komponenty mohou být postaveny jako dynamické knihovny namísto jednotlivých procesů. To snižuje celkové zatížení systému – např. více komponent napsaných v Javě tak může běžet na jednom virtuálním stroji, místo jednoho virtuálního stroje pro každou komponentu, jak by tomu bylo v případě oddělených procesů.

IceStorm

IceStorm je publish/subscribe služba, která odděluje klienty od serverů. IceStorm funguje jako přepínač distribuce událostí – vydavatelé (*publishers*) zasílají události této službě, která je dále rozešle odběratelům (*subscribers*). Tímto způsobem může být událost zaslaná jedním vydavatelem rozeslána mnoha odběratelům. Události jsou rozděleny podle tématu (*topic*) a odběratelé specifikují, která témata je zajímají. Odběrateli jsou tak zasílány jenom události, které odpovídají jeho zájmům. IceStorm je užitečný zejména v případě, kdy chcete distribuovat informaci většímu množství komponent.

IcePatch2

IcePatch2 je služba pro záplatování aplikací. Umožňuje jednoduše distribuovat aktualizace softwaru klientům. Klient se jednoduše připojí k IcePatch2 serveru a požádá o aktualizaci konkrétní aplikace. IcePatch2 zkontroluje verzi aplikace, kterou má klient nainstalovanou a eventuálně odešle klientovi aktuální verzi v komprimované podobě. Záplaty mohou být zabezpečeny službou Glacier2, aby mohli aktualizace stahovat jenom autorizovaní klienti.

Glacier2

Glacier2 je služba pro procházení firewally – umožňuje klientům a serverům bezpečně komunikovat přes firewall bez ohrožení bezpečnosti. Komunikace mezi klientem a serverem může být šifrována pomocí SSL za použití certifikátů a je obousměrná. Glacier2 obsahuje podporu pro vzájemnou autentizaci a správu zabezpečených sezení.

7.2 Qt

Qt je multiplatformní framework pro vývoj aplikací, používaný především pro vývoj GUI aplikací. Mezi nejznámější softwarové produkty využívající Qt patří pracovní prostředí KDE, webový prohlížeč Opera, VoIP aplikace Skype nebo embedded Linux platforma Qt Extended. Qt je vyvíjen společností Qt Software (dříve Trolltech), dceřinnou společností společnosti Nokia, a je licencován pod GNU GPL, GNU LGPL a proprietární licencí. Qt používá C++ s několika nestandardními rozšířeními implementovanými pomocí speciálního preprocesoru, který před kompilací generuje standardní C++ kód. Existují také bindingy (dodávané třetími stranami) pro Javu, .NET, Python, Ruby a další jazyky. Qt je dostupný pro platformy Linux (včetně embedded Linuxu), Windows, Mac OS X, Symbian S60 a Windows CE. [30]

Qt byl zvolen pro tvorbu GUI zejména z těchto důvodů:

- splňuje podmínky podpory více platforem a open-source licence,
- jednoduchost – intuitivní objektově orientovaná tvorba GUI,
- kvalita GUI – dobrá integrace do pracovního prostředí pro všechny podporované platformy,
- rozsáhlá podpora lokalizace – umožní přeložit GUI programu do mnoha jazyků,
- podpora skriptování – modul QtScript implementuje variantu jazyka ECMAScript, možnost změny chování programu bez rekompilace,
- podpora malých zařízení – možnost v budoucnu portovat klientskou část na tento typ zařízení,
- kvalita dokumentace – rozsáhlá a úplná dokumentace včetně názorných příkladů,
- vyvíjen komerční společností – zajištění životnosti projektu i do budoucna.

7.2.1 Moduly

Qt není dodáván jako monolitická knihovna, ale nabízí řadu modulů, poskytujících specifickou funkcionalitu. V této kapitole si uvedeme informace k některým důležitějším z nich. [31]

QtCore

QtCore je základní ne-GUI modul Qt, všechny ostatní moduly na něm závisí. Poskytuje mj. třídy pro základní datové typy (řetězec, seznam, hash tabulka, apod.), práci se soubory nebo vícevláknové programování. [32]

QtGui

Modul *QtGui* je jádrem GUI funkcionality, poskytuje třídy GUI widgetů (tlačítka, stromové pohledy, dialogová okna, apod.) a podpůrné třídy. [33]

QtNetwork

Modul *QtNetwork* poskytuje třídy pro usnadnění síťového programování, od nízkoúrovňových tříd pro práci se sockety po vysokoúrovňové třídy pro práci s protokoly HTTP, FTP a SSL. [34]

QtScript

Modul *QtScript* poskytuje třídy umožňující skriptovatelnost aplikací jazykem na bázi ECMAScriptu (podobně jako např. JavaScript). [35]

QtScriptTools

Modul *QtScriptTools* poskytuje třídy doplňující modul QtScript o možnost debugování skriptů. [36]

QtSql

Pomocí modulu *QtSql* lze aplikace rozšířit o podporu propojení s relační databází. Mezi podporované databázové systémy patří MySQL, PostgreSQL, Oracle, sqlite a databázové systémy podporující ODBC (např. Microsoft SQL Server). [37]

QtWebKit

Modul *QtWebKit* zajišťuje integraci s HTML renderovacím jádrem *WebKit* (toto jádro je také součástí prohlížečů Apple Safari a Google Chrome). [38]

QtXml

Modul *QtXml* poskytuje třídy pro práci s XML soubory a DOM. [39]

QtXmlPatterns

Modul *QtXmlPatterns* rozšiřuje modul QtXml o podporu XPath, XQuery a částečnou podporu XSLT (do budoucna se plánuje úplná podpora této technologie). [40]

QtDBus

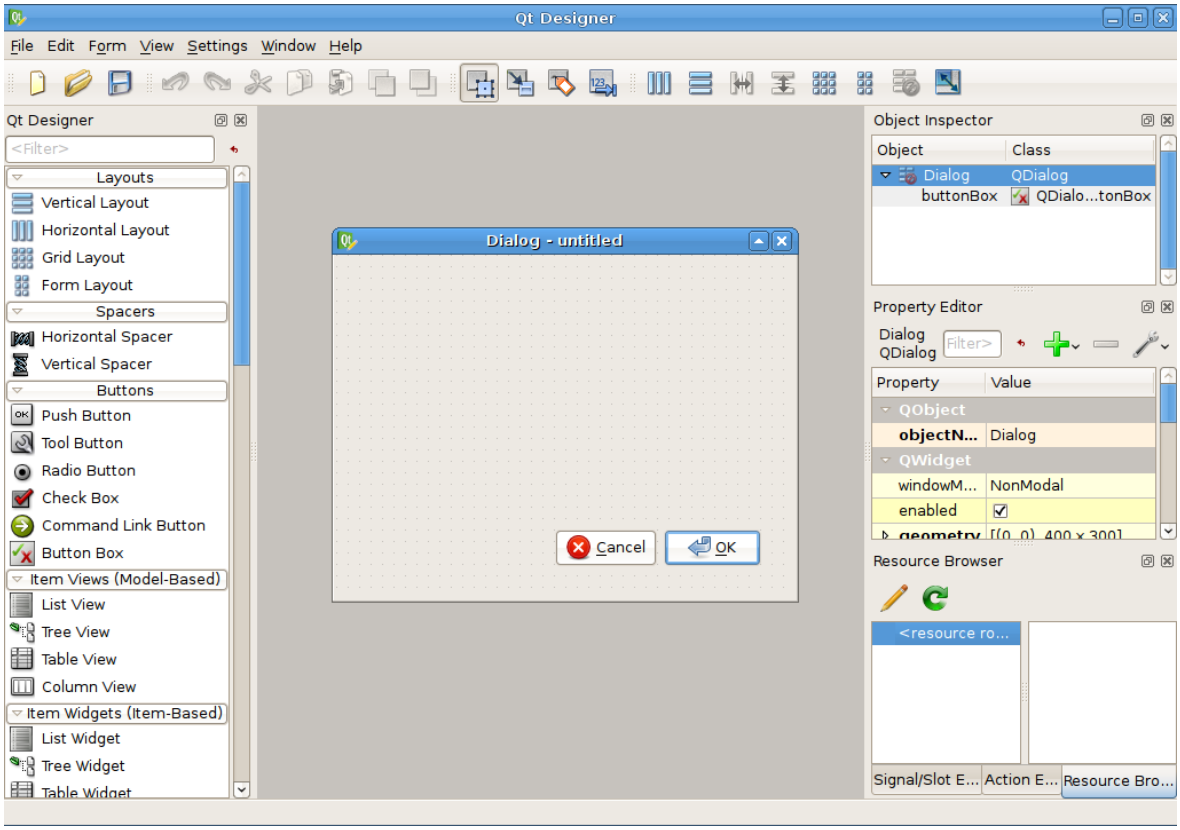
Modul QtDBus zajišťuje integraci IPC mechanismu D-Bus do aplikací. [41]

7.2.2 Nástroje

Qt přichází s řadou nástrojů k usnadnění vývoje aplikací.

Qt Designer

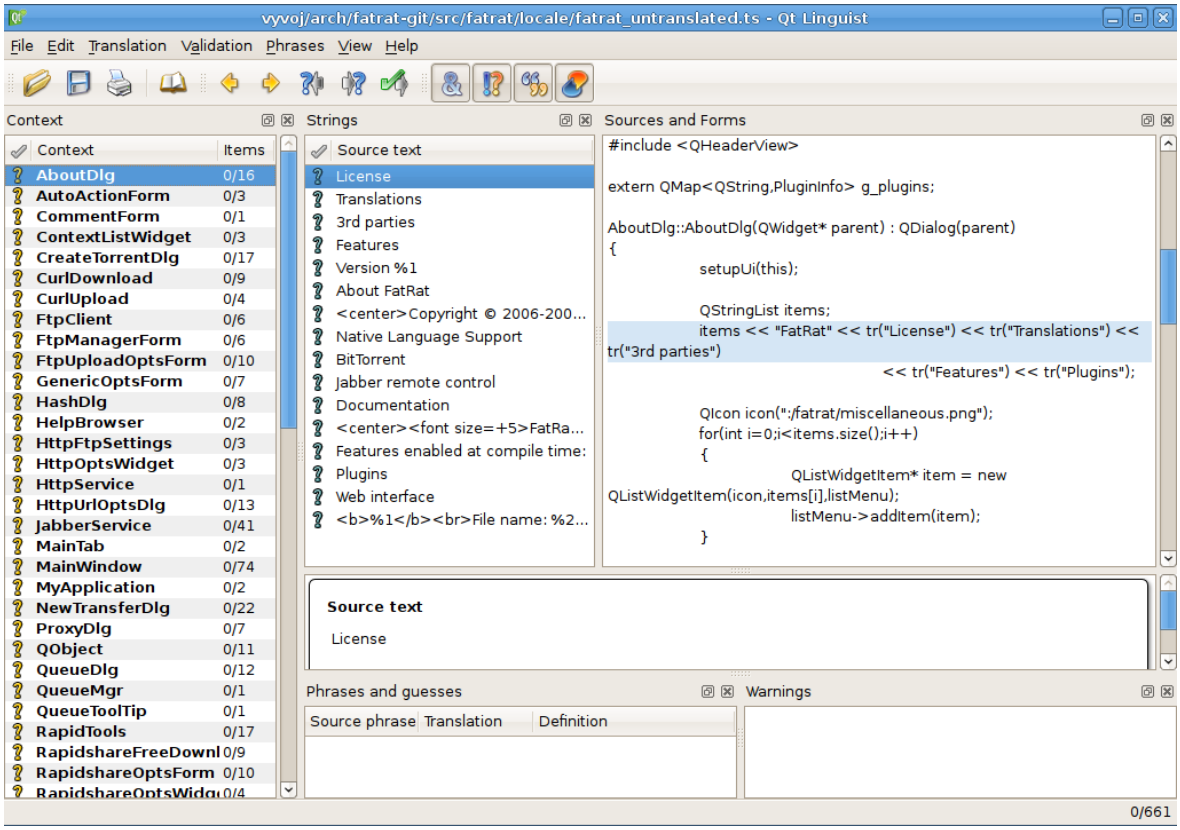
Qt Designer je nástroj na vytváření grafického uživatelského rozhraní pomocí komponent Qt. Jedná se o WYSIWYG nástroj, což zajišťuje okamžité zobrazení výsledku. Widgety a formuláře vytvořené Qt Designerem lze jednoduše integrovat do vlastní aplikace. [42]



Obr. 5: Okno Qt Creatoru

Qt Linguist

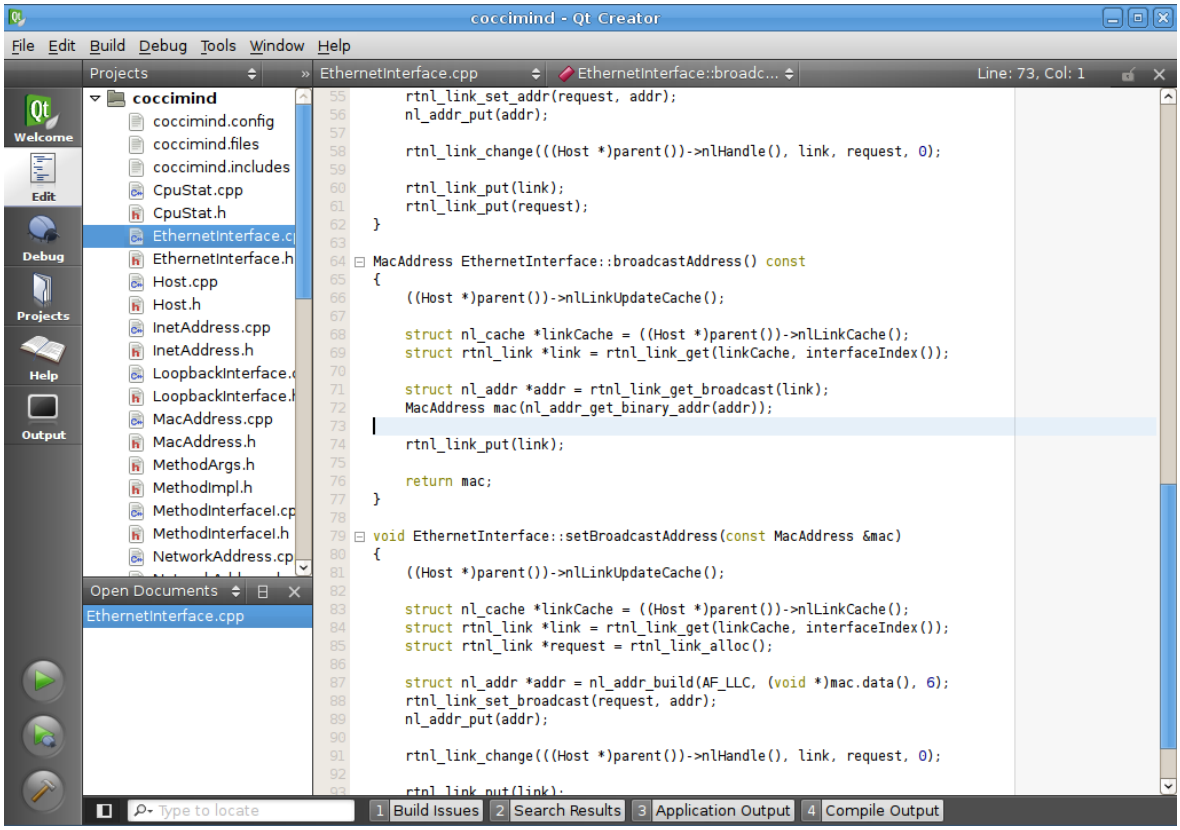
Qt Linguist je nástroj na překlad Qt aplikací. Jeho pomocí lze aplikaci přeložit do mnoha jazyků. [43]



Obr. 6: Okno Qt Linguist

Qt Creator

Qt Creator je multiplatformní IDE pro tvorbu Qt aplikací. Má v sobě mj. zahrnutu podporu debugování, správy revizí kódu pomocí několika SCM systémů (SVN, Git, Perforce) a grafický návrh GUI (pomocí integrovaného Qt Designeru). [44]



Obr. 7: Okno Qt Creatoru

8 IMPLEMENTACE APLIKACE

Přiložený program implementuje jenom podmnožinu navrhovaných vlastností. Chybějící funkcionalita bude doplněna v dalších verzích programu, stejně jako nové vlastnosti.

8.1 Serverová část

Implementace serverové části v rámci jednoduchosti spojuje všechny vrstvy návrhu do jedné vrstvy, která obsluhuje jak komunikaci s OS, tak komunikaci s klienty. Správa sezení a oprávnění není implementována vůbec – připojit k serveru se může kdokoliv bez ověření identity a může provádět jakákoliv operace. Chybí také podpora pro ukládání stavu objektů do databáze (všechny dostupné informace se získají při startu serveru od OS).

Prostředníkem mezi objektovým modelem a klienty je Ice objekt *ObjectManager* – jeho pomocí lze získat kořenový objekt serveru, seznam všech objektů a konkrétní objekt podle jeho UUID.

Server zahrnuje implementaci těchto typů objektů:

- *Object* – bázový typ všech typů objektů.
- *ServerRoot* – kořenový objekt serveru. Vlastní jeden objekt typu *Host*.
- *Host* – objekt představující stroj, na kterém server běží – na serveru existuje vždy jediná instance. Tento typ objektu podporuje operace *Shutdown* a *Reboot*, obsahuje zapisovatelnou vlastnost *hostname* (síťový název stroje), statistiku zatížení procesoru *cpu load* a vlastní skupinu objektů typu *NetworkInterface*.
- *ReadOnlyProperty* – vlastnost jenom pro čtení. Podporuje operaci *Read*.
- *Property* – vlastnost umožňující čtení i zápis. Podporuje operace *Read* a *Write*.
- Stat – objekt, který zaznamenává statistické údaje nadřazeného objektu. Podporuje metodu *ReadOut* pro získání dat pro zvolený časový úsek.
- *CpuStat* – zaznamenává vytížení procesoru stroje.
- *TrafficStat* – zaznamenává provoz na síťovém rozhraní v určeném směru.
- *NetworkInterface* – představuje síťové rozhraní. Tento typ objektu obsahuje vlastnost jen pro čtení *interface index* (číslo síťového rozhraní), vlastnosti *interface name* (název síťového rozhraní) a *state* (stav síťového rozhraní) a statistiky provozu *incoming traffic* a *outgoing traffic* (příchozí a odchozí provoz).

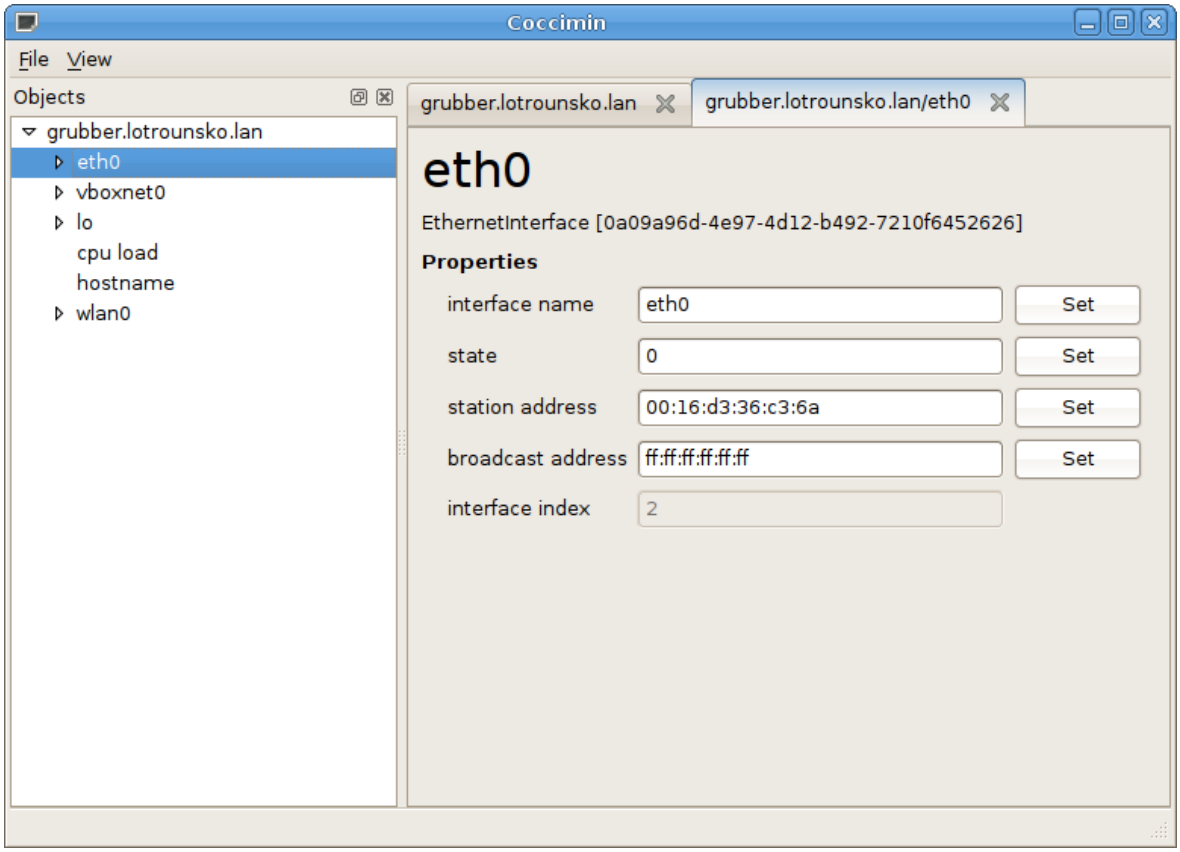
- *LoopbackInterface* – představuje lokální síťové rozhraní *lo*.
- *EthernetInterface* – představuje síťové rozhraní kompatibilní s ethernetem. Přidává vlastnosti *station address* a *broadcast address* (MAC adresa stanice a všesměru).

Server čte nastavení z konfiguračního souboru, který se nachází ve stejném adresáři, jako spustitelný soubor serveru.

8.2 Klientská část

Klientská část je implementovaná jako jednoduchá Qt aplikace s jedním hlavním oknem. Toto okno je rozděleno na dvě části: strom objektů a zobrazovací oblast se záložkami.

Strom objektů zobrazuje hierarchickou strukturu objektů podle jejich názvu. Aktivací objektu (poklepáním, z kontextového menu nebo klávesou Enter) se otevře v zobrazovací oblasti nová záložka s detailním pohledem na objekt. Tento pohled se skládá z hlavičky a oblastí s operacemi, vlastnostmi a statistikami. Hlavička obsahuje název objektu, jeho typ a UUID. Oblast s operacemi obsahuje tlačítka, jimiž lze na objektu volat jednoduché operace. Oblast s vlastnostmi obsahuje seznam názvů a hodnot vlastností, které daný objekt vlastní, s možností nastavit novou hodnotu. Oblast se statistikami zobrazuje grafy průběhů naměřených dat.



Obr. 8: Okno klientské části aplikace

Klient podobně jako server čte nastavení z konfiguračního souboru, který se nachází ve stejném adresáři, jako spustitelný soubor klienta.

ZÁVĚR

Cílem této práce bylo vytvořit softwarovou aplikaci pro vzdálenou správu Linuxových serverů a routerů. Tento cíl byl splněn, nicméně výsledný program je spíše jen ukázkou koncepce (*proof-of-concept*) celé aplikace. Zbývá ještě dořešit spoustu problémů a přidat funkcionalitu jako je kontrola oprávnění uživatelů, podpora pro přiřazování síťových adres (IPv4, IPv6) rozhraním, nastavování routovacích pravidel a síťových služeb (DHCP, Samba, HTTP server, atd.).

Vývoj bude pokračovat dále a ponese se v duchu open-source (podobně jako vývoj samotného Linuxu) a jednoho dne se možná i stane běžnou součástí linuxových distribucí.

CONCLUSION

The aim of this work was to create a software application for remote administration of Linux servers and routers. The aim was accomplished, however the resulting software is mainly a proof-of-concept of the application. There are still many problems to solve and work to be done in areas like user permission management, support for adding network addresses (IPv4, IPv6) to network interfaces, setting up routing rules and network services (DHCP, Samba, HTTP server, etc.).

Development of the application will continue in an open-source fashion (like the development of Linux itself) and some day it might become a regular part of Linux distributions.

SEZNAM POUŽITÉ LITERATURY

[1] *MikroTik* [online]. [cit. 2009-05-25]. Dostupný z WWW: <http://en.wikipedia.org/wiki/MikroTik>.

[2] *GNU/Linux naming controversy* [online]. [cit. 2009-05-25]. Dostupný z WWW: <http://en.wikipedia.org/wiki/GNU/Linux_naming_controversy>.

[3] *Tux* [online]. [cit. 2009-05-25]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Tux>.

[4] *D-Bus* [online]. [cit. 2009-05-23]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Dbus>.

[5] *freedesktop.org* [online]. [cit. 2009-05-23]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Freedesktop.org>.

[6] *Introduction to D-Bus* [online]. [cit. 2009-05-23]. Dostupný z WWW: <http://doc.trolltech.com/4.5/intro-to-dbus.html>.

[7] H. PENNINGTON, A. CARLSSON, A. LARSSON. *D-Bus Specification* [online]. 2007. [cit. 2009-05-23]. Dostupný z WWW: <http://dbus.freedesktop.org/doc/dbus-specification.html>.

[8] M. T. JONES. *Inside the Linux boot process* [online]. 2009. [cit. 2009-05-24]. Dostupný z WWW: <http://www.ibm.com/developerworks/library/l-linuxboot/index.html>.

[9] *initrd* [online]. [cit. 2009-05-24]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Initrd>.

[10] *init* [online]. [cit. 2009-05-24]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Init>.

[11] *Initng* [online]. [cit. 2009-05-24]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Initng>.

[12] *Plugins* [online]. [cit. 2009-05-24]. Dostupný z WWW: <http://www.initng.org/wiki/Documents_Plugins>.

[13] *Upstart* [online]. [cit. 2009-05-24]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Upstart>.

[14] *FAQ* [online]. [cit. 2009-05-24]. Dostupný z WWW: <http://upstart.ubuntu.com/faq.html>.

[15] *Device file system* [online]. [cit. 2009-05-23]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Device_file_system>.

[16] G. KROAH-HARTMAN. *Hot Plug* [online]. 2009. [cit. 2009-05-23]. Dostupný z WWW: <http://www.linuxjournal.com/article/5604>.

[17] *udev* [online]. [cit. 2009-05-24]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Udev>.

[18] *HAL (software)* [online]. [cit. 2009-05-25]. Dostupný z WWW: <http://en.wikipedia.org/wiki/HAL_(software)>.

[19] D. ZEUTHEN. *HAL 0.5.10 Specification* [online]. 2009. [cit. 2009-05-25]. Dostupný z WWW: <http://people.freedesktop.org/~david/hal-spec/hal-spec.html>.

[20] D. ZEUTHEN. *Update on DeviceKit*. 2008. Dostupný z WWW: <http://lists.freedesktop.org/archives/hal/2008-May/011560.html>.

[21] *NetworkManager* [online]. [cit. 2009-05-25]. Dostupný z WWW: <http://en.wikipedia.org/wiki/NetworkManager>.

[22] *NetworkManager* [online]. [cit. 2009-05-25]. Dostupný z WWW: <http://projects.gnome.org/NetworkManager/>.

[23] *Telnet* [online]. [cit. 2009-05-26]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Telnet>.

[24] *Secure Shell* [online]. [cit. 2009-05-26]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Secure_Shell>.

[25] *Webmin* [online]. [cit. 2009-05-26]. Dostupný z WWW: <http://www.webmin.com/demo.html>.

[26] *Universally Unique Identifier* [online]. [cit. 2009-05-21]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Uuid>.

[27] *Role-based access control* [online]. [cit. 2009-05-21]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Rbac>.

[28] *Internet Communications Engine* [online]. [cit. 2009-05-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Internet_Communications_Engine>.

[29] M. HENNING, M. SPRUIELL. *Distributed Programming with Ice* [online]. 2008. [cit. 2009-05-22]. Dostupný z WWW: <http://www.zeroc.com/doc/Ice-3.3.1/manual/>.

[30] *Qt (toolkit)* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Qt_(toolkit)>.

[31] *Qt's Modules* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/modules.html>.

[32] *QtCore Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtcore.html>.

[33] *QtGui Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtgui.html>.

[34] *QtNetwork Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtnetwork.html>.

[35] *QtScript Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtscript.html>.

[36] *QtScriptTools Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtscripttools.html>.

[37] *QtSql Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtsql.html>.

[38] *QtWebKit Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtwebkit.html>.

[39] *QtXml Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtxml.html>.

[40] *QtXmlPatterns Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtxmlpatterns.html>.

[41] *QtDBus Module* [online]. [cit. 2009-05-20]. Dostupný z WWW: <http://doc.trolltech.com/4.5/qtdbus.html>.

[42] *Qt Designer Manual* [online]. [cit. 2009-05-21]. Dostupný z WWW: <http://doc.trolltech.com/4.5/designer-manual.html>.

[43] *Qt Linguist Manual* [online]. [cit. 2009-05-21]. Dostupný z WWW: <http://doc.trolltech.com/4.5/linguist-manual.html>.

[44] *Qt Development Tools* [online]. [cit. 2009-05-21]. Dostupný z WWW: <http://www.qtsoftware.com/products/developer-tools>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application programming interface
BSD	Berkeley software distribution
DCOP	Desktop communication protocol
GNU	GNU's not Unix
GPL	General public license
GRUB	Grand unified bootloader
GUI	Graphical user interface
HAL	Hardware abstraction layer
HW	Hardware
IDL	Interface definition language
IPC	Inter-process communication
KDE	K desktop environment
LILO	Linux loader
LVM	Logical volume manager
RAM	Random access memorz
MAC	Media access control
OLPC	One laptop per children
OS	Operační systém
RAID	Redundant array of inexpensive disks
RPC	Remote procedure call
SCM	Source code management
SSH	Secure shell
UDI	Unique device identifier
UUID	Universally unique identifier
X11	X window systém
XML	Extensible markup language

SEZNAM OBRÁZKŮ

SEZNAM VÝPISŮ

SEZNAM PŘÍLOH

PŘÍLOHA P 1: SEZNAM POUŽITÉHO SOFTWARU.

OpenOffice.org 3.1	sazba této práce
Qt Creator	vývoj aplikace, snímek obrazovky
Qt Designer	snímek obrazovky
Qt Linguist	snímek obrazovky
Inkscape 0.46	tvorba diagramů