O vrtnih palčkih, površno in od daleč - razvojna platforma GNOME

Jaka Močnik
ActiveTools d.o.o.
e-pošta: jaka@gnu.org

Povzetek

Razvojna platforma GNOME predstavlja množico vmesnikov in njihovo izvedbo, ki omogočajo programerju kar najbolj učinkovit razvoj kompleksnega namiznega programja. V pričujočem članku si bomo ogledali celotno arhitekturo razvojne platforme, pa tudi posamezne knjižnice in njihove medsebojne odnose ter navedli vire, kjer se lahko bralec o njih podrobneje pouči.

1 Uvod

Do leta 1997 si je prosto programje pridobilo pomemben delež med strežniškimi sistemi: mnogi spletni in datotečni strežniki, požarni zidovi, usmerjevalniki so tekli na prostih operacijskih sistemih, večinoma Unixu podobnih, njihove storitve pa je nudila prosta programska oprema. Velik del ostalih strežnikov je tekel na lastniških Unix operacijskih sistemih.

Med operacijskimi sistemi računalnikov, ki so služili kot namizne delovne postaje, je bilo stanje popolnoma drugačno: po eni izmed raziskav je več kot 75% namiznih računalnikov poganjalo eno od oblik operacijskega sistema Windows: Windows 95 je tekel na kar 63% namizij. 7% uporabnikov

- vezanih na strojno opremo Appla - je poganjalo MacOS, ostanek pa je bil enakomerno razporejen med lastniške Unix sisteme, Linux, *BSD, DOS in OS/2 [1].

Razmerje na trgu namiznih operacijskih sistemov je bilo pripisati predvsem neobstoju uporabnega namiznega okolja in namiznih aplikacij za Unix in Unixu podobne operacijske sisteme: temeljni namen projekta GNOME - the GNU Network Object Model Environment - je bil in je še vedno vzpostaviti infrastrukturo, ki omogoča razvijalcu hiter in učinkovit razvoj kompleksnih namiznih aplikacij, uporabniku pa enostavno uporabo le-teh in izkušnjo enotnega računalniškega namizja, podobno tisti, ki so je deležni uporabniki Microsoftovih Oken ali Applovega MacOS-a. Pri tem naj bi se - v najboljši maniri prostega programja - izognili pregovorni nestabilnosti in hroščatosti, ki pesti uporabnike Oken.

2 Problem namizja

Temeljni problem postavitve prostega operacijskega sistema na namizni računalnik je bilo pomanjkanje osnovnih aplikacij, primernih za povprečnega uporabnika: hekerji, ki so večinoma pisali prosto programje, pač niso premogli primernih znanj za načrtovanje profesionalnih uporabniških vmesnikov in pogosto tudi niso imeli posluha za želje uporabnikov namiznih računalnikov.

Vzrok za takšno stanje pa je bila predvsem stagnacija razvojnega okolja na Unixih in Unixu podobnih operacijskih sistemih: tehnologije, ki bi omogočale lažjo pouporabo kode (ang. code reuse), pravzaprav niso obstajale. Stanje je bilo v protislovju s tradicionalnim "Unixovim načinom početja stvari": odkar je Dennis M. Ritchie iznašel koncept pip, so uporabniki Unixa kompleksna opravila izvajali tako, da so povezovali množico preprostih orodij, specializiranih za eno samo opravilo [2]. Tiskanje seznama datotek v abecednem vrstnem redu na tiskalniku opravimo takole:

```
ls | sort | enscript -o- | lpr
```

Prvi ukaz izpiše datoteke, drugi jih uredi, tretji naredi iz ASCII besedila PostScript datoteko, primerno za tiskanje, četrti pa to pošlje tiskalniku.

Pipe so zadoščale, dokler je uporabnik živel v asketskem okolju ukazne vrstice: sodelovanje komponent, ki tvorijo kompleksno aplikacijo z grafičnim uporabniškim vmesnikom (GUI), pa je neprimerno bolj zapleteno - linearen cevovod nenadoma ne zadošča več.

Dotični problem je Microsoft elegantno rešil s tehnologijo (D)COM(+), ki je omogočala kompleksno interakcijo med množico programskih objektov, ki lahko živijo v ločenih naslovnih prostorih ali celo na različnih računalnikih v omrežju. Z uporabo COM-a je razvoj distribuirane aplikacije postal močno podoben (preprostemu) razvoju monolitne aplikacije - programerji v okolju Windows so na ta način naenkrat dobili neslutene možnosti za pouporabo obstoječe kode: razvijalec aplikacije, ki je morala tiskati grafe, je v ta namen s kar najmanjšim trudom mirno uporabil Excelove komponente!

Izvedbe sorodnih sistemov so sicer obstajale tudi na Unixu - omenimo le standard CORBA [3], ki ga določa skupina OMG [4]. Težava je bila predvsem v pomanjkanju proste izvedbe in neobstoju standardnih višjenivojskih vmesnikov, ki bi poenotili in poenostavili pogoste vrste interakcij med objekti. Kot posledica te dolgoletne stagnacije je bila pouporaba obstoječe kode prej izjema kot pravilo: če si ogledamo tipično namizno aplikacijo iz tistega časa - Netscape ali StarOffice -, ugotovimo, da si navkljub mnogim skupnim elementom, kot so komponente uporabniškega vmesnika ali prikazovanje slikovnih datotek, zapisanih v istih grafičnih formatih, ti aplikaciji nista delili praktično nobene kode razen standardne C knjižnice in osnovnih knjižnic okenskega sistema X Window. Vsaka aplikacija je bila monoliten mastodont, sam zase!

3 Cilj projekta

Projekt GNOME je nastal z namenom ponuditi uporabniku celotno, enovito namizno okolje z aplikacijami, ki so podobnega videza, ki se odzivajo na enak način in ki ponujajo enoten vmesnik do skupnih storitev. Pot do tega posvečenega cilja pa je vodila preko izvedbe infrastrukture, ki nudi razvijalcu vmesnike do skupnih storitev, ki jih potrebujejo namizne aplikacije: komponent grafičnega uporabniškega vmesnika, upravljanja s sejami (session management), podatkovne baze za konfiguracijo, prikaza besedila, branja slikovnih datotek, pogostih ikon ipd. S ponudbo takšnih vmesnikov dosežemo kar največjo pouporabo kode v namiznih aplikacijah in poenostavimo njihov razvoj, obenem pa jih prisilimo v konsistentno podobo uporabniškega vmesnika.

Glavni cilji, ki jih upoštevamo med razvojem, so naslednji:

- GNOME naj podpira paradigmo komponentnega programiranja: kompleksne aplikacije se gradijo iz množice neodvisnih komponent, ki sodelujejo preko dobro definiranih vmesnikov (npr. z metodami medprocesne komunikacije)
- razvojno platformo sestavlja izključno prosto programje [5]
- platforma mora biti uporabna iz poljubnega programskega jezika: vse osnovne knjižnice morajo ponujati vmesnik za jezik C, ki ga je enos-

tavno zaviti v poljuben programski jezik, vmesniki komponent pa morajo biti opisani na od jezika neodvisen način

- uporabljajo naj se obstoječi odprti standardi za kar največjo mero sodelovanja z ostalim programjem, ki podpira te standarde; morebitni novi standardi morajo biti odprti in po možnosti predani v skrb organizaciji, neodvisni od projekta GNOME

4 Zunanje knjižnice

Razvojno okolje GNOME potrebuje za svojo osnovo množico knjižnic, ki pa so potrebne le za njegovo izvedbo. Njihove podrobnosti so skrite pred programerjem - uporabnikom razvojnega okolja. Te knjižnice so:

- za prikaz teksta potrebuje GNOME razširitvi okolja X Window *Xft* in *Xrender*; prva je odvisna še od knjižnice za konfiguracijo in delo s tipografijami *fontconfig* [6]
- za branje in pisanje slikovnih datotek uporablja knjižnice *libpng* [7], *libjpeg* [8] in *libtiff [9]*
- *libpopt* za delo s parametri ukazne vrstice
- knjižnice okenskega sistema X Window [10] [11]
- *libbz2* in *zlib* za stiskanje podatkov
- *libfam* za nadzor nad spremebami datotek
- *libxml2* [14] in *libxslt* vsebujeta vmesnike za branje XML datotek in predelavo le-teh

5 GLib - programerjev švicarski nož

Na samem dnu razvojnega okolja GNOME je knjižnica glib [12], sprva razvita kot pomožna knjižnica za program GIMP [13]. Vsebuje bogat nabor podatkovnih struktur: povezane sezname, razpršeno tabelo, polja in drevesa, ipd. Poleg teh nudi še mehanizme za delo s pomnilnikom in nizi, prevajanje med kodnimi nabori in delo z Unicode znaki, izvajanje zunanjih programov, delo s časom in koledarsko aritmetiko ...

Poleg osnovne knjižnice libglib vsebuje paket GLib še tri:

libgthread vsebuje od platforme neodvisen vmesnik za rokovanje z nitmi

libgmodule nudi vmesnik za delo z moduli, ki jih povezujemo v program med samim izvajanjem

libgobject je izvedba objektnega sistema v jeziku C: programer z njeno pomočjo dela z informacijo o tipu objektov med izvajanjem in na preprost način v jeziku C uporablja koncepte, kot so dedovanje, agregacija vmesnikov, virtualne funkcije, štetje referenc, ipd. Večina razredov objektov, ki jih uporablja programer, posredno ali neposredno deduje iz razreda GObject, ki ga priskrbi ta knjižnica.

6 V očeh opazovalca: grafični uporabniški vmesnik

Grafični uporabniški vmesnik v okolju GNOME je sestavljen iz osnovnih komponent (*ang. widgets*), ki so izvedene v knjižnici GTK+ [17]. GNOME omogoča lokalizacijo (*l10n*) in internacionalizacijo (*i18n*). V ta namen podpira prevajanje nizov s pomočjo orodja gettext [15] ter prikaz teksta v različnih pisavah s pomočjo knjižnice Pango [16]. GNOME omogoča uporabo namizja tudi invalidnim osebam, slabovidnim ali nezmožnim rokovanja z miško ali tipkovnico - knjižnica ATK [18] namreč omogoča podporo orodjem, kakršen je bralec zaslona (*ang. screen reader*), povečavo zaslona in uporabo alternativnih načinov vnosa - ta funkcionalnost je na voljo brez posebnega truda s strani programerja, ki uporablja razvojno platformo GNOME.

6.1 Pango

Knjižnica Pango [16] omogoča stavljenje in prikaz internacionaliziranega teksta. Vsi nizi, s katerimi dela Pango, so zapisani v kodnem naboru UTF-8. Prednost tega je kompatibilnost z obstoječimi funkcijami, ki delujejo z običajnim ASCII tekstom, in neodvisnost od strojne opreme (UTF-8 kodiranje je predstavljeno enako na *Big-Endian* ali *Little-Endian* CPE). Knjižnica je zasnovana modularno: od jezika ali sistema tipografij odvisne posebnosti so pri stavljenju teksta v ločenih modulih, ki se nalagajo po potrebi.

6.2 ATK

ATK [18], *Accessibility Toolkit*, opisuje množico vmesnikov, ki jih morajo izvesti komponente GUI-ja, da bi bile invalidnim uporabnikom dosegljive (accessible) s pomočjo pomožnih naprav (assistive technologies), kot so bralci zaslona, Braillova vrstica, tipkovnica na zaslonu, ipd. Vmesniki so neodvisni od izvedbe komponent GUI-ja: lahko jih podpira tako GTK+ kot Motif, QT ali kakšna druga GUI knjižnica.

Primera vmesnikov ATK sta `AtkText` in `AtkSelection`; prvega mora izvesti vsaka komponenta, ki vsebuje tekst (npr. tekstno polje), slednjega pa komponente, ki omogočajo izbiro vsebovanih komponent (npr. menu).

Sistem, namenjen invalidnemu uporabniku, na osnovi nastavitev ob vsaki pognani GNOME aplikaciji samodejno naloži še modul, ki vsebuje izvedbo teh vmesnikov za vse standardne GUI komponente, ki jih nudijo GTK+ in višjenivojske GNOME knjižnice. Ta izvedba komunicira s ponudnikom storitev za pomožne naprave (AT-SPI, *Assistive Technology Service Provider Interface*), demonom, ki upravlja s prijavljenimi pomožnimi napravami. Taistega ponudnika storitev na enak način uporabljajo tudi Java (s pomočjo vmesnikov *Java Accessiblity Framework*), OpenOffice.org in Mozilla.

Dokler uporabnik sestavlja GUI le iz standardnih komponent, ki jih nudi okolje GNOME, dobi podporo za pomožne naprave žastonj".

6.3 GTK+

GTK+ [22] definira abstrakten razred komponente GUI-ja (*widget*) in iz njega izpelje najrazličnejše vrste komponent, ki zadoščajo za večino splošnih potreb namiznega programja: okna, sezname, menije, ipd.

Risanje komponent poteka preko vmesnika GDK (*The GIMP Drawing Toolkit*), ki predstavlja abstrakcijo okenskega sistema: implementacije GDK so trenutno na voljo za X Window System, Win32 API in Linuxov framebuffer. Slednja je še posebej primerna za uporabo v vgradnih (*ang. embedded*) sistemih, ki ne premorejo zadostnih virov za sistem X Window. GDK vsebuje tudi funkcije za branje in pisanje slikovnih podatkov v datoteke v pogostih formatih zapisa (JPEG, PNG, ipd.) ter njihov prikaz v okenskem sistemu.

GTK+ poskrbi za implementacijo ATK vmesnikov za vse standardne komponente in prikaz teksta s pomočjo Panga.

6.4 Glade

Knjižnica libglade [19] omogoči programerju, da ustvari GUI svoje aplikacije med izvajanjem aplikacije tako, da naloži opis uporabniškega vmesnika, podan v XML datoteki, namesto da bi ga zapisal v obliki programske kode že v času razvoja; opis lahko spreminjamo neodvisno od same programske kode bodisi z urejevalnikom teksta ali pa s posebnim orodjem Glade [20], ki omogoča oblikovalcu, da "nariše"podobo uporabniškega vmesnika ter jo shrani v XML datoteko. Z orodjem `glade2c` lahko tak opis pretvorimo tudi v kodo v jeziku C, ki jo lahko prevedemo, namesto da bi opis nalagali med izvajanjem.

6.5 Libart

Libart [22] ponuja programerju močan vmesnik za 2D grafične operacije in izjemno optimizirano izvedbo teh operacij: podpira vse operacije, ki jih ponuja PostScript ter doda še *anti-aliasing* in prozornost (*ang. alpha channel*).

6.6 GnomeCanvas

Palčje platno, GnomeCanvas [23], je avtomat (*ang. engine*) za prikaz strukturirane 2D grafike; za dejansko risanje uporablja bodisi klice GDK, ki so izjemno hitri, a nekoliko omejujejo funkcionalnost, ali pa knjižnico libart, ki ponuja nadmnožico operacij PostScriptovega prikazovalnega modela.

7 V začetku je bila Beseda: medprocesna komunikacija in povezovanje komponent

Interakcijo med komponentami v okolju GNOME omogoča lastna implementacija specifikacije CORBA 2.2 [3] [24], posrednik zahtev objektov (*ang. Object Request Broker, ORB*) ORBit2.

Operacije, ki jih je komponenta sposobna izvajati, so definirane z vmesnikom komponente: vmesnik je opisan v jeziku IDL (*Interface Definition Language*) in ga sestavljajo operacije, ki jih odjemalci smejo prožiti na vmesniku (proženje operacije je v CORBA terminologiji ekvivalentno klicu metode v terminologiji klasičnih objektnih jezikov). CORBA razširi paradigmo objektnega programiranja iz monolitnih aplikacij, kjer vsi objekti živijo v istem naslovnem prostoru (v enem samem procesu), na porazdeljene aplikacije, kjer sodelujejo objekti v različnih procesih in tudi na različnih računalnikih v omrežju. V klasičnih objektnih jezikih dostopamo do objektov s pomočjo preproste objektne reference, ki je npr. kazalec na objekt v jeziku C++, v CORBA sistemih pa preko CORBA reference, posebnega podatkovnega tipa, katerega vsebina je programerju skrita in vsebuje tudi omrežno lokacijo objekta.

Raba vmesnika je mogoča na enega od dveh načinov:

1. pri statični uporabi programer z IDL prevajalnikom generira kodo v željenem jeziku za strežniški proces, ki izvede vmesnik, in odjemalca, ki uporablja operacije tega vmesnika. Programer mora le dodati kodo, specifično za posamezne operacije, v strežniški del. Trenutno sta podprta C in C++.

   Tako zgenerirana koda skrije pred programerjem vse umazane podrobnosti medprocesne komunikacije: dejansko lokacijo strežniškega procesa na omrežju (*network transparency*), strojno arhitekturo in OS, na katerem teče strežnik (*platform transparency*), ter klicne konvencije jezika, v katerem sta napisana strežnik in odjemalec (*language transparency*). Programer se lahko osredotoči le na osnovni problem, s katerim se ukvarja aplikacija.

2. pri dinamični uporabi, ki je pogosta predvsem pri skriptnih jezikih, kot sta npr. Perl ali Python, tako odjemalec kot strežnik v času izvajanja pridobita informacijo o željenem vmesniku od skladišča vmesnikov, ki je del ORB-a, in na osnovi te informacije poskrbita za ustrezno komunikacijo.

ORBit2 omogoča prenos po različnih komunikacijskih kanalih, ki vključujejo TCP/IPv4, TCP/IPv6, SSL preko TCP/IPv4 in IrDA, ter se lahko spričo mnogih optimizacijskih prijemov pohvali z izjemnimi performansami : v primeru, ko komunicirajoči komponenti živita v istem naslovnem prostoru (sta del enega procesa), ORBit2 obide prenos podatkov po mreži, njihovo pakiranje in razpakiranje, s čimer doseže, da so stroški (časovni in pomnilniški) proženja operacije takšni kot pri klicu virtualne metode v jeziku C++.

8 Planet opic: Bonobo

Na temeljih CORBE je zgrajen objektni model GNOME, imenovan Bonobo, ki si ga bomo ogledali v treh delih:

8.1 Knjižnica libbonobo

Ta poskrbi za lažjo uporabo CORBE: neposredna uporaba CORBA objektov je, še posebej v jeziku C, za programerja nerodna. CORBA kodo zato zavijemo v objektni sistem knjižnice GLib, ki smo ga omenili v razdelku 5: uporaba tako pripravljenih CORBA objektov je za programerja mnogo preprostejša.

Knjižnica nudi mehanizme za upravljanje z življenjskim ciklom objektov (*ang. lifecycle management*): poskrbi za štetje referenc in avtomatsko uničenje objektov, ko števec referenc doseže vrednost 0. Poleg tega preverja stanje objektov na omrežju, saj je znano dejstvo, da s TCP/IP skladom ni mogoče zanesljivo ugotoviti prekinjene povezave in je za to potrebno poskrbeti na aplikacijskem nivoju.

Libbonobo [25] vsebuje tudi deklaracijo in izvedbo nekaterih osnovnih vmesnikov, ki jih potrebuje objektni model GNOME:

- vmesnik `Bonobo::Unknown` je podoben vmesniku `IUnknown`, znanem iz Microsoftove tehnologije COM: ponuja operacije za dodajanje in odvzemanje reference ter pridobivanje ostalih vmesnikov, ki jih objekt podpira.

- vmesnik `Bonobo::GenericFactory` je vmesnik, ki ga implementirajo "tovarne objektov": ta vmesnik pozna eno samo operacijo, `create_instance`, ki ustvari primerek nekega drugega objekta.

- vmesniki za vhod in izhod: `Bonobo::Stream`, `Bonobo::Storage` in `Bonobo::Persist`

8.2 Knjižnica libbonoboui

Elemente grafičnega uporabniškega vmesnika, ki jih nudi nek objekt, lahko njegov odjemalec vstavi v svoj GUI z mehanizmom, oblikovanim po vzoru Microsoftove tehnologije OLE (*Object Linking and Embedding*). Knjižnica libbonoboui [26] doseže ta cilj tako, da del funkcionalnosti komponente GUI-ja, ki jo ponuja razred `GtkWidget` iz knjižnice GTK+, ponudi preko CORBA vmesnikov in omogoči odjemalcu, da v svoj GUI vstavi elemente, za prikaz katerih skrbi drug objekt, ki je seveda lahko tudi v ločenem naslovnem prostoru ali celo na oddaljenem računalniku. To funkcionalnost podpirata dva vmesnika:

- `Bonobo::ControlFrame` uporablja odjemalec. Ta vmesnik omogoča vstavljanje komponent GUI-ja, zavitih v vmesnik `Bonobo::Control`
- `Bonobo::Control` uporablja strežniški objekt - vanj so zavite komponente GUI-ja in vsebuje metode, s pomočjo katerih objekt, ki podpira vmesnik `Bonobo::ControlFrame`, prikaže te komponente integrirane v njegov GUI.

8.3 Strežnik za aktivacijo objektov

V CORBA terminologiji "aktivacija objekta" pomeni pridobitev CORBA reference do objekta in po potrebi tudi stvarjenje nove instance objekta. Sama specifikacija CORBA [24] ne določa načina, kako poteka aktivacija objekta na strani odjemalca: kako pognati strežnik in pridobiti reference objektov, ki jih strežnik vsebuje, je prepuščeno aplikaciji sami.

GNOME poskrbi za enostavno aktivacijo objekta s pomočjo demona `bonobo-activation-server` [27]: ob namestitvi se vsak objekt registrira pri tem demonu z namestitvijo XML datoteke, ki opisuje njegove lastnosti s standardiziranimi prilastki (*ang. attributes*). Odjemalske aplikacije smejo s pomočjo preprostega povpraševalnega jezika, oblikovanega po vzoru SQL select stavka, zahtevati aktivacijo objekta, ki ustreza povpraševanju. Pri povpraševanju lahko navedemo kompleksne pogoje, ki jih mora objekt izpolnjevati: vmesnike, ki jih mora implementirati, podporo za branje določenih tipov podatkov ipd.

9 Palčji register: konfiguracijska podatkovna baza GConf

Podatkovna baza GConf [28] je namenjena konfiguracijskim podatkom komponent, ki tvorijo okolje GNOME. V hierarhični drevesni strukturi hrani pare (ključ, vrednost), zgrajena pa je na podlagi paradigme *Model-View-Controller*. Za trenutno stanje skrbi poseben program, gconfd, ki predstavlja vstopno točko za aplikacije, ki želijo brati ali pisati konfiguracijske podatke, omogoča pa tudi obveščanje aplikacij o spremembah vrednosti, ki aplikacijo zanimajo. Za shranjevanje podatkov lahko uporablja različne "back-end"module, ki skrbijo za hrambo v LDAP imeniku, XML datotekah na lokalnem disku ali podatkovni bazi z SQL vmesnikom.

10 Kako do podatkov: Gnome-VFS

Knjižnica Gnome VFS [29] je abstrakcija datotečnega sistema, ki omogoča uporabniškemu programju, da z enotnim vmesnikom dostopa do datotek na različnih "pravih"datotečnih sistemih, od lokalnih diskov do digitalnih kamer, FTP strežnikov in datotek na deljenih diskih na računalnikih z operacijskim sistemom Windows (*shares*). V ta namen ponuja funkcije za dostop do datotek, pregledovanje njihovih lastnosti, ugotavljanje MIME [30] tipa podatkov v datoteki in ugotavljanja, katere komponente so zmožne ravnati z določenim tipom podatkov.

11 Zaključek: kje smo in kam zdaj?

S platformo GNOME 2, katere zadnjo inkarnacijo, GNOME 2.4, smo izdali 10. septembra 2003, se je po več kot petih letih razvoja, pri katerem je sodelovalo več tisoč razvijalcev, prevajalcev, piscev dokumentacije, preizkuševalcev in drugih, doseglo večino ciljev, ki so bili zastavljeni leta 1997: razvojna platforma GNOME danes omogoča hiter in učinkovit razvoj kompleksnega programja za namizje s pomočjo povezovanja neodvisnih komponent, celotno razvojno okolje ustreza definiciji prostega programja, uporabljati pa ga je mogoče iz mnogih programskih jezikov: iz C-ja in C++-a, Perla in Pythona ter v omejenem obsegu celo iz Haskella [31]. Vsi standardi, uporabljeni v okolju GNOME, so odprti; nekateri, ki so med razvojem nastali, pa so doživeli tudi širši sprejem: definicijo EWMH (*Extended Window Manager Hints*) danes uporablja tudi namizje KDE.

Cilj je bil dosežen tudi s stališča uporabnika namizja: danes namizje GNOME ponuja množico popolnoma integriranih aplikacij, ki omogočajo njegovo udobno rabo. Med aplikacijami, razvitimi izključno za okolje GNOME, omenimo le PIM (*Personal Information Manager*) Evolution, preglednico GNumeric, urejevalnik teksta AbiWord, H.323 videokonferenčno aplikacijo GnomeMeeting ter množico aplikacij, razvitih na osnovi multimedijske infrastrukture GStreamer. Z namizjem GNOME so integrirane tudi mnoge druge aplikacije, omenimo le Mozillo in OpenOffice.org. Namizje je skoraj popolnoma prevedeno v 29 jezikov, med njimi tudi v slovenščino, še dodatnih

13 jezikov pa ima prevedenih več kot pol teksta. Z namizjem lahko po zaslugi odlične podpore pomožnim napravam delajo tudi invalidni uporabniki.

Bodoče različice namizja GNOME naj bi poleg novih in izboljšanih aplikacij na namizju prinesle tudi izboljšano ravnanje s podatki: namesto klasične predstavitve s hierarhičnim datotečnim sistemom bo uporabnik do podatkov dostopal preko njihove vsebine - s tem se ukvarja projekt GNOME Storage [32]. V namizje GNOME pa bomo s pomočjo projekta Mono [33] integrirali tudi podporo za okolje .NET.

Literatura

[1] Desktop Operating Systems, http://newsport.org/archive/s97/ms/mshare/opsys.html

[2] Ritchie, D. M., The Evolution of the Unix Time-sharing System, http://cm.bell-labs.com/cm/cs/who/dmr/hist.html

[3] OMG, CORBA FAQ, http://www.omg.org/gettingstarted/corbafaq.htm

[4] Object Management Group, http://www.omg.org

[5] What Does Free Mean?, http://www.debian.org/intro/free

[6] Fontconfig Project Page, http://x2.freedesktop.org/tiki/tiki-index.php?page=fontconfig

[7] libpng Home Page, http://www.libpng.org/pub/png/libpng.html

[8] Independent JPEG Group, http://www.ijg.org/

[9] TIFF Software, http://www.remotesensing.org/libtiff/

[10] X.Org Homepage, http://www.x.org/

[11] XFree86 Project Homepage, http://www.xfree86.org/

[12] GLib Reference Manual, http://developer.gnome.org/doc/API/2.0/glib/index.html

[13] The GIMP Homepage, http://www.gimp.org/

[14] The XML C parser and toolkit of Gnome, http://www.xmlsoft.org/

[15] gettext - GNU Project, http://www.gnu.org/software/gettext/

[16] Pango - Introduction, http://www.pango.org/

[17] GTK+ - The GIMP Toolkit, http.//www.gtk.org

[18] GNOME Accessibility for Developers, http://developer.gnome.org/projects/gap/guide/gad/index.html

[19] libglade, http://www.daa.com.au/ james/software/libglade/

[20] Glade: A GTK+ User Interface Builder, http://glade.gnome.org/index.html

[21] Art of Code Libart Page, http://www.artofcode.com/libart.html

[22] GTK+ - The GIMP Toolkit, http://www.gtk.org

[23] The Gnome Canvas Whitepaper, http://developer.gnome.org/doc/whitepapers/canvas/canvas.html

[24] CORBA/IIOP 2.2 Specification, http://www.omg.org/cgi-bin/doc?formal/98-07-01

[25] Libbonobo API Reference Manual, http://developer.gnome.org/doc/API/2.0/libbonobo/index.html

[26] Libbonoboui API Reference Manual, http://developer.gnome.org/doc/API/2.0/libbonoboui/index.html

[27] Bonobo Activation API Reference Manual, Bonobo Activation API Reference Manual

[28] GConf Manual, http://developer.gnome.org/doc/API/2.0/gconf/index.html

[29] GnomeVFS - Filesystem Abstraction library, http://developer.gnome.org/doc/API/2.0/gnome-vfs-2.0/index.html

[30] RFC 2045 - 2049, Network Working Group, http://www.rfcs.org/

[31] Chakravarty, M. T. M., Functional Programming of Component-based Software, University of New South Wales, Sydney, Australia, 2000, http://www.cse.unsw.edu.au/ chak/haskell/gnome/gnome-small/gnome-small.html

[32] GNOME Storage, http://www.gnome.org/ seth/storage/index.html

[33] The Mono Project Homepage, http://www.go-mono.com/