Conceptes avançats en desenvolupament de programari lliure

Jordi Campos Miralles
Ramon Navarro Bosch
Daniel Riera i Terrén (coordinador)

XP06/M2011/01806

UOC
www.uoc.edu

Daniel Riera i Terrén

Doctor enginyer en Informàtica per la Universitat Autònoma de Barcelona. Professor responsable de les assignatures de programació dels Estudis d'Informàtica, Multimèdia i Telecomunicació de la Universitat Oberta de Catalunya.

Jordi Campos Miralles

Enginyer en Informàtica per la Universitat Ramon Llull. Professor de la Universitat de Barcelona en els estudis d'Enginyeria Informàtica. Investigació en informàtica gràfica i biomedicina a la Universitat Politècnica de Catalunya.

Ramon Navarro Bosch

Enginyer en Informàtica per la Universitat Politècnica de Catalunya. Investigació en informàtica gràfica i biomedicina a la Universitat Politècnica de Catalunya. Actualment en estada internacional a la Universitat d'Arhus, Dinamarca.

Primera edició: febrer 2006
Fundació per a la Universitat Oberta de Catalunya
Av. Tibidabo, 39-43, 08035 Barcelona
Material realitzat per Eureca Media, SL

Dipòsit legal: B-49.944-2006

Índex

Introducció 7

Objectius 8

1. Conceptes generals 9
1.1. Mono 9
1.1.1. Introducció 9
1.1.2. Arquitectura 9
1.1.3. Temes jurídics 10
1.1.4. Implementació 11
1.2. GTK+ 11
1.2.1. Introducció 11
1.2.2. Història 13
1.2.3. Qt i WxWindows 13
1.3. Interfícies gràfiques d'usuari 13
1.3.1. Què són les interfícies gràfiques d'usuari? 13
1.3.2. Com s'estructuren? 14
1.4. *Widgets* 15
1.4.1. Contenidors 15
1.4.2. Llistat de *widgets* 15
1.5. Usabilitat 16
1.5.1. Introducció 16
1.5.2. Factors 17
1.5.3. Beneficis 17
1.5.4. Metodologia de treball 18
1.5.5. Guies bàsiques 19

2. Mono 21
2.1. Instal·lació 21
2.1.1. Nadiu 22
2.1.2. Debian 22
2.1.3. Fedora 22
2.2. Eines bàsiques 23
2.3. C# bàsic 23
2.3.1. Sentències 24
2.3.2. Tipus elementals 24
2.3.3. Entrada/sortida 25
2.3.4. Cadenes 27
2.3.5. Vectors i taules 28
2.4. C# mitjà 29
2.4.1. Definició de classes 29
2.4.2. Definició d'atributs interns de classe 30

2.4.3. Mètodes 32
2.4.4. Herència i polimorfisme 34
2.4.5. Delegats i esdeveniments 35
2.4.6. *Structs*, *enums* i *namespaces* 38
2.5. Boo 39
2.5.1. Estructura bàsica de Boo 39
2.5.2. Diferències amb C# 40
2.5.3. Exemples 41
2.6. Nemerle 42
2.6.1. Diferències bàsiques amb C# 43
2.6.2. Ús de funcions locals 44
2.6.3. Macros i disseny per contracte 45
2.6.4. Tipus `variant` i patrons 47
2.6.5. Exemple d'aplicació amb GUI 47

3. GTK# i Gdk# 50
3.1. Instal·lació 50
3.1.1. Exemple GTK# 50
3.2. Widgets GTK# 52
3.2.1. Elements contenidors 53
3.2.2. Elements gràfics 59
3.2.3. *Canvas* 75
3.3. Ús 79
3.3.1. Bucle d'esdeveniments 80
3.3.2. *Drag and drop* 82

4. Glade 85
4.1. Instal·lació 85
4.2. Glade 85
4.3. Glade# 87
4.4. Maneig d'esdeveniments 89
4.5. Alternatives: Gazpacho i Stetic 89

5. *Widgets* avançats 92
5.1. GtkHtml# 92
5.2. Gecko# 96
5.3. GtkSourceView# 99

6. Gnome# 102
6.1. Elements gràficos 102
6.2. Diàlegs 105
6.3. Impressió 108
6.4. *Canvas* 110
6.5. Assistents 114

7. XML 116
7.1. Llegir i escriure XML 116
7.2. Navegar i manipular XML en memòria 117

7.3. Transformar amb XML 119
7.4. Seriar 119
7.5. Restringir XML 120

8. Biblioteca avançada 122
8.1. GnomeVFS 122
8.2. Gconf 125
8.3. Internacionalització 126
8.4. Nant i Nunit 132
8.4.1. Nant 132
8.4.2. Nunit 133

9. Aplicacions 137
9.1. F-Spot 137
9.2. MCatalog 140
9.3. Muine 141
9.4. Beagle 142

Bibliografia 145

GNU Free Documentation License 147

Introducció

En aquesta assignatura es presenten tecnologies avançades per al desenvolupament d'aplicacions mitjançant l'ús de programari lliure. Inclouen eines per a facilitar la programació, interfícies gràfiques d'usuari i alguns elements gràfics existents, entre d'altres. També s'hi inclouen alguns conceptes importants i metodologies per a ajudar en l'esmentada tasca de programació.

En el primer capítol s'introdueixen alguns dels temes en els quals posteriorment s'aprofundeix, i conceptes importants per a l'elecció d'eines. Igualment, es mostren els factors que s'han de tenir en compte en dissenyar aplicacions i els beneficis que obtindrem en aplicar certes metodologies de treball.

Seguidament, es dedica un capítol a la presentació del programari Mono, des de la instal·lació del seu compilador, màquina virtual i llibreries en diferents distribucions de GNU/Linux, fins a la sintaxi bàsica del llenguatge. També s'explica l'ús de classes, les relacions entre classes, delegats, esdeveniments i alguns tipus de dades complexes. El capítol acaba mostrant dos llenguatges alternatius, Boo i Nemerle, i les diferències bàsiques d'aquests llenguatges amb C#.

El tercer capítol presenta una alternativa lliure per a la implementació d'aplicacions gràfiques en Mono: la biblioteca GTK. Es mostra com s'ha d'instal·lar, s'explica com es poden crear *widgets* i quins elements ens ofereixen.

El quart capítol parla de l'eina possiblement més estesa per a utilitzar GTK+ en el desenvolupament d'interfícies gràfiques: Glade. Es mostra que gestiona el codi, però que posteriorment requereix vincular-lo amb la resta de codi de l'aplicació.

Una vegada som capaços de generar interfícies gràfiques, es presenten elements gràfics (*widgets*) avançats en el capítol 5.

Fins al moment, s'ha mostrat com es poden programar interfícies gràfiques d'usuari usant GTK i l'eina de disseny Glade. El capítol sis proposa l'aprofitament dels elements bàsics del projecte Gnome per a desenvolupament d'aplicacions.

El setè capítol proposa l'ús de l'estàndard XML per a la creació i transacció de documents per a la Xarxa. Mono suporta aquest estàndard i presenta una sèrie de mètodes per a manipular documents XML.

El material acaba introduint aspectes de biblioteques avançades portades a Mono i presentant algunes de les aplicacions amb interfície gràfica més conegudes, desenvolupades usant Mono i GTK.

Objectius

Els materials didàctics d'aquesta assignatura aporten a l'estudiant els coneixements necessaris per a treballar amb eines avançades per al desenvolupament de programari. Els objectius que cal haver assolit en finalitzar l'assignatura són els següents:

1. Repassar alguns factors que cal tenir en compte en dissenyar aplicacions i els beneficis que obtindrem en aplicar certes metodologies de treball.

2. Conèixer diferents tecnologies que ens permeten crear aplicacions usant programari lliure. Saber-les instal·lar per poder utilitzar-les en la creació de noves aplicacions.

3. Aprendre a programar en C#, Boo i Nemerle, i saber apreciar les diferències que hi ha entre aquests llenguatges.

4. Ser capaç d'escollir entre les biblioteques lliures disponibles a l'hora de programar una interfície gràfica d'usuari i treure profit dels avantatges que ens ofereixen.

5. Veure algunes de les aplicacions més conegudes fetes amb aquestes tecnologies lliures.

1. Conceptes generals

1.1. Mono

1.1.1. Introducció

El **projecte Mono** es va crear amb l'objectiu d'implementar en programari lliure les especificacions del Common Language Infrastructure (CLI), informalment conegut com **entorn .NET**. Aquestes espedificacions van ser publicades com a estàndard a l'Associació Europea per a l'Estandardització de Sistemes d'Informació i Comunicacions (ECMA-International), inicialment per Microsoft. En concret, es tracta de l'estàndard Ecma-334* per a la definició del llenguatge C# i de l'estàndard Ecma-335** per a la definició de CLI amb les seves correspondències estàndard de la International Organization for Standarization (ISO).

Webs recomanats

Projecte Mono: http://www.mono-project.com

ECMA-International: http://www.ecma-international.org

Microsoft: http://msdn.microsoft.com/netframework

Organization for Standarization (ISO): http://www.iso.org

*** http://www.ecma-international.org/publications/standards/Ecma-334.htm**

**** http://www.ecma-international.org/publications/standards/Ecma-335.htm**

1.1.2. Arquitectura

El CLI de Microsoft va ser dissenyat com a alternativa a l'entorn Java de Sun. Ambdós es basen en l'estratègia d'usar una **màquina virtual**. És a dir, que el codi font dels programes no es compila en el codi màquina d'una arquitectura de computadores existents, sinó que es compila en el codi màquina d'una arquitectura fictícia, virtual. D'aquesta manera, per a executar una mateixa aplicació en diferents arquitectures de computadora, no és necessari tornar a compilar el codi font, sinó simplement disposar d'una màquina virtual que s'executi en cada una de les arquitectures. El programa que simula la màquina virtual és el que interpreta el codi màquina virtual que ha generat el compilador, el codi objecte.

Les tècniques emprades per a implementar els programes que simulen la màquina virtual són també comunes entre el CLI i l'entorn Java. La que més èxti ha tingut ha estat la que tradueix fragments del codi objecte al codi de l'arquitectura de la computadora que allotja la màquina virtual a mesura que els necessita, el *Just in Time* (JIT). De fet, el CLI va ser dissenyat pensant bàsicament per a utilitzar aquesta tècnica que ja estava donant bons resultats en l'entorn Java.

El codi objecte de la màquina virtual, que en l'entorn Java s'anomena *Bytecode*, en el CLI s'anomena Common Intermediate Language (CIL).

Fins aquí les coincidències entre els dos entorns, però justament en la concepció del CIL hi ha la principal diferència. El CLI va ser dissenyat amb l'objectiu

de facilitar la traducció de diferents llenguatges font a aquest codi intermedi. Per aquest motiu, l'estàndard no solament defineix el llenguatge de la màquina virtual sinó també la base comú entre tots els llenguatges que es dissenyen per a aquesta plataforma, el **Common Language Specification (CLS)**. Per permetre la interoperativitat entre aquests llenguatges font, també es va definir la manera en què havien d'estructurar-se els tipus de dades, el **Common Type System (CTS)**. Totes aquestes definicions, juntament amb la inclusió de metadades en el codi objecte, componen l'estàndard Ecma-335 i són les que faciliten l'existència de diversos llenguatges font per a aquest entorn.

Per aquest motiu, en l'actualitat l'entorn Java compta bàsicament amb un sol llenguatge font, el llenguatge Java, mentre que en el CLI podem usar nombrosos llenguatges font, com ara C++, Perl, Python, PHP Fortran, Cobol, Boo, VisualBasic, J# i fins i tot el mateix Java (alguns d'aquests llenguatges amb adaptacions de la seva especificació original). Però no solament és possible usar-los, sinó que també és possible combinar-los entre si. Per exemple, podem declarar una classe en C++ i crear-ne una instància en Python.

A part dels llenguatges ja existents, per als quals hi ha compiladors a CIL, també s'han creat alguns llenguatges nous. Per exemple, en el moment de dissenyar el CLI, també es va dissenyar un llenguatge per a explotar al màxim les possibilitats que ofereix, el **llenguatge C#**. Aquest llenguatge va se definit originàriament per Microsoft i publicat com a estàndard Ecma-334. Es tracta d'un llenguatge de computadora imperatiu de la família dels llenguatges C/C++, igual que el Java. Les semblances, doncs, amb aquest darrer són moltes, tot i que hi ha diferències importants que en el seu moment Microsoft va proposar a Sun d'incloure a Java, però que no van ser acceptades.

1.1.3. Temes jurídics

El projecte Mono bàsicament pretén oferir el CLI i el conjunt d'eines de baix nivell per poder-hi treballar, tot plegat basat en els estàndards definits per l'Ecma-334 i l'Ecma-335.

Malgrat aquest enfocament inicial, actualment també inclou algunes biblioteques d'alt nivell que implementen parts de la plataforma .NET de Microsoft que no estan publicades com a estàndards. Aquestes biblioteques estan sotmeses a patents per part de Microsoft, la qual cosa podria provocar problemes jurídics sobre el codi que les utilitzi i la pròpia posada en pràctica d'aquestes biblioteques.

La postura del projecte Mono ha estat la de fomentar l'ús de la posada en pràctica dels estàndards Ecma-334/Ecma-335 i oferir alternatives a les interfícies de programació (API) que estan afectades per les patents de Microsoft. Així doncs, s'aconsella d'utilitzar solament els estàndards i les API alternatives, i no usar les biblioteques d'alt nivell basades en els models de Microsoft.

Migració al projecte Mono

El motiu per a implementar les API de Microsoft ha estat oferir una menor migració a programadors actuals de la plataforma .NET al projecte Mono, però s'espera que a mitjà termini utilitzin les biblioteques alternatives lliures de patents.

1.1.4. Implementació

Actualment hi ha dues iniciatives principals per a implementar el CLI en programari lliure: el projecte Mono i el Portable.NET*. Aquest darrer està emmarcat dins del projecte DotGNU**, que no solament pretén implementar el CLI, sinó també oferir tota una infraestructura completa a més alt nivell per al desenvolupament d'aplicacions alternatives obertes que treballin en xarxa. Aquesta infraestructura és una alternativa completa a les biblioteques d'alt nivell que utilitza Microsoft, ja que no són estàndard i estan sotmeses a patents.

*** http://www.gnu.org/projects/dotgnu/pnet.html**
**** http://www.gnu.org/projects/dotgnu**

El projecte Mono ofereix el CLI (estàndard Ecma-335) mitjançant la implementació de la seva màquina virtual anomenada **mono**. Aquesta màquina s'acompanya d'un compilador per al llenguatge C# (estàndard Ecma-334) anomenat **Mcs**, i d'un conjunt de biblioteques a baix nivell incloses en els estàndards.

A part d'aquestes peces bàsiques, també s'inclou un conjunt molt ampli de biblioteques per a manipular interfícies gràfies d'usuari, acceleració 3D, components multimèdia, bases de dades, programació distribuïda, etc., totes basades en projectes –existents o de nova creació– de programari lliure.

Per últim, s'inclouen biblioteques basades en les API de Microsoft per tal de facilitar la migració de programadors de la plataforma .NET, tot i que se'n desaconsella l'ús per evitar possibles problemes jurídics.

1.2. GTK+

1.2.1. Introducció

GTK+ és un conjunt d'eines multiplataforma per a la creació d'interfícies d'usuari.

Com a base per al funcionament de les diferents eines, hi ha un conjunt de biblioteques que treballen a baix nivell:

- **Glib** és una biblioteca de baix nivell bàsica per a GTK+ i Gnome. Hi ha implementats els enllaços a escala de C de les estructures de dades, la implementació dels fils, la càrrega dinàmica d'objectes i les interfícies per als bucles d'execució.

- **Pango** és una biblioteca especialitzada en la visualització de les capes de text i el seu renderitzat. Posa una atenció especial en la internacionalització.

- **Atk** és una biblioteca per a crear interfícies amb característiques d'accessibilitat. Es poden usar eines com lupes d'augment, lectors de pantalla o entrades de dades alternatives al clàssic teclat o ratolí d'ordinador.

El conjunt de biblioteques de GTK+ afegeix a aquestes biblioteques un conjunt d'eines per al desenvolupament d'elements visuals, de control d'interacció i esdeveniments, de dibuix de mapes de bits i de dibuix vectorial. Totes s'han dissenyat i implementat desenvolupant vincles a una multitud de llenguatges entre els quals destaquem Perl, Python, Java i C++. A partir d'aquests llenguatges, que es consideren bàsics per al desenvolupament d'aplicacions d'escriptori, s'han desenvolupat vincles per a Ada, Haskell, PHP i C# (que nosaltres utilitzarem). Per poder aconseguir ser una plataforma de treball vàlida per a qualsevol aplicació s'han desenvolupat implementacions sobre plataformes de la família Microsoft i Linux. Actualment, la implementació sobre llenguatges com ara .Net i Java ens permet poder fer aplicacions que es poden portar a més paltaformes: MacOS, Solaris, etc.

Aquest conjunt de biblioteques ha crescut amb el temps i actualment tenim les següents:

- **Gdk**: biblioteca per a enllaçar els contenidors gràfics amb les diferents biblioteques. Gestiona els esdeveniments, *drag and drop*, la connexió amb les X, i els elements d'entrada i sortida de maquinari. Fa de pont entre la biblioteca i els diferents elements externs que intervenen en les aplicacions GTK.

- **GTK**: biblioteca d'elements gràfics de treball. Hi tenim els diferents *widgets* que podem usar, els estils de visualització dels elements, les funcions a escala d'usuari del *drag and drop*, senyals i el portapapers, entre altres. Més endavant detallarem els elements sobre els quals podem treballar.

- **GObject**: biblioteca que proporciona funcionalitats sobre els objectes i una interfície per a treballar-hi des de C. Té una implementació dels tipus bàsics, un sistema de senyals i diferents propietats d'objectes com, per exemple, l'herència.

- **GdkPixbuf**: biblioteca per a treballar amb imatges de mapes de bits. Ens permet tenir un *buffer* de treball i poder fer retocs, redimensions i treballar a escala de píxel.

- **Cairo**: biblioteca per a treballar amb imatges vectorials. Permet utilitzar l'acceleració gràfica de la targeta per a poder visualitzar imatges vectorials i mapes de bits mapats en estructures.

La biblioteca Cairo

Cairo és una biblioteca externa al projecte que ha permès afegir gràfics i textos de gran qualitat a les aplicacions des de la versió 2.4.

Per poder desenvoluar una aplicació d'escriptori, segurament solament ens interessarà usar la biblitoeca GTK. Hi trobem la majoria de les funcions d'alt ni-

vell necessàries per a programar la interacció i visualització. La programació d'esdeveniments, *drag and drop* i control dels diferents elements visuals es poden utilitzar fàcilment des de libGtk.

Un dels aspectes més importants en la biblioteca GTK és la facilitat d'adaptació visual. Tota la biblioteca s'ha implementat de manera modular, amb la qual cosa es pot tenir per serparat l'aspecte que es vol que tinguin els diferents elements visuals. D'aquesta manera, l'usuari pot personalitzar fàcilment el color i la forma dels botons, finestres, llistes, etc.

1.2.2. Història

Inicialment, aquesta biblitoeca es va crear per desenvolupar l'aplicació GIMP amb l'aspecte i funcionalitat de Motif. En formar part del grup de programari de GNU, va començar a ser utilitzada per diferents aplicacions que necessitaven una interfície gràfica d'usuari. Va passar a ser una de les més utilitzades quan l'entorn d'escriptori Gnome va apostar-hi com a base de la seva interfície gràfica, que hi va afegir moltes funcions i nous elements visuals personalitzats a les necessitats de l'entorn. Gràcies a aquesta aportació, totes les aplicacions per a aquest escriptori usen libGtk.

1.2.3. Qt i WxWindows

A més de GTK+, a GNU/Linux hi ha diferents biblioteques gràfiques per al desenvolupament d'aplicacions d'escriptori. Actualment hi ha dues més que destaquen. Una és Qt, de l'empresa Trolltech, sobre la qual es basa l'entorn d'escriptori KDE. Aquest entorn té una llicència GPL però amb una opció a una llicència comercial si es paga a l'empresa. L'altra és Wxwindows, una biblioteca que va néixer amb l'ànim de ser portable de MacOs, Linux i Windows. El 1992, davant la necessitat de desenvolupar una aplicació per a dues plataformes diferents i davant la falta d'una biblioteca portable, es va crear Wxwindows. Amb els anys, i després de guanyar molts usuaris, s'ha portat a GTK per poder-ne utilitzar tota la potència.

1.3. Interfícies gràfiques d'usuari

1.3.1. Què són les interfícies gràfiques d'usuari?

> Les interfícies gràfiques d'usuari són un mecanisme de connexió que facilita la interacció entre l'ordinador i l'usuari mitjançant un conjunt d'imatges i objectes gràfics a més de text.

Hi ha un conjunt d'elements (icones, finestres, caselles de text, etc.) que ens permeten actuar còmodament amb els programes per poder enviar i rebre in-

formació. Aquestes interfícies han evolucionat molt i actualment es divideixen en els tipus següents:

- Orientades a programes d'escriptori. En aquest grup tenim una gran quantitat de programes que es desenvolupen actualment. Finestres, àrees de text, botons, àrees gràfiques vectorials, àrees gràfiques de mapes de bits, menús, etc. Més endavant parlarem d'aquest tipus d'interfícies.

- Orientades a programes web. En aquest tipus podem incloure totes les aplicacions web que treballen sempre amb el mateix conjunt de *widgets*: els definits per HTML. L'avantatge de ser multiplataforma i molt fàcil d'utilitzar li atorga el desavantatge de tenir una interacció reduïda i una dificultat de programació elevada. Avui dia es desenvolupen diferents iniciatives per millorar la interfície d'usuari de la web (AJAX, XForms, etc.), però continua havent-hi un problema d'interacció. En una aplicació que requereix una relació propera amb l'usuari d'intercanvi de dades pot provocar grans pèrdues de temps i funcionalitat.

- Orientades a jocs. En aquest tipus tenim interfícies gràfiques molt artístiques, amb tècniques 3D, efectes, etc. Hi ha una varietat enorme d'entorns en aquesta classe, ja que normalment es programen des de zero cada vegada per poder-les adaptar a les necessitats del joc.

Hi ha tipus més avançats que depenen de sistemes d'interfície maquinari més sofisticats, per exemple entorns 3D. Aquí parlarem de les interfícies gràfiques d'usuari d'escriptori.

1.3.2. Com s'estructuren?

Hi ha una gran quantitat d'interfícies d'usuari d'escriptori i cada una s'estructura d'una manera específica. Per exemple, en una aplicació per a Windows Forms podem definir una aplicació en què podem crear subfinestres dins d'una de principal, de manera que mai no en puguin sortir. En una aplicació amb GTK com a element bàsic tenim el *widget* GtkWindow, que és una finestra. Aquesta finestra es pot estructurar de dues maneres diferents:

- Estructurada: dividint la finestra amb caixes contenidores que ens permetin definir cada un dels objectes visuals amb una referència relativa a tota l'aplicació. Les divisions que permet Gtk són en horitzontal i vertical. D'aquesta manera, aconseguim l'efecte que, en redimensionar l'aplicació, la distribució dels objectes es redimensiona, amb la qual cosa s'obté un resultat visual millor. El problema d'aquest mètode és la dificultat de dividir tota la interfície en divisions horitzontals i verticals.

- Fixa: afegim el *widget* GtkFixed a una caixa contenidora o finestra. Amb aquesta forma afegim els diferents elements visuals en el lloc on volem que

es vegin. El problema rau a dimensionar l'aplicació: els diferents elements queden en la posició original, cosa que produeix un efecte visual normalment no desitjable. És molt més fàcil i ràpid desenvolupar amb posicions fixes, per la qual cosa seria útil per a prototipus i aplicacions petites.

1.4. *Widgets*

Widget és qualsevol element gràfic que puguem usar en el disseny d'una interfície gràfica. Aquests elements definits segons cada API de programació ens permeten generar, combinant-los, qualsevol pantalla.

En algunes API es poden generar elements propis com a combinació d'altres o amb altres biblioteques nadiues com a base de la seva visualització.

1.4.1. Contenidors

Els contenidors són un conjunt de *widgets* que ens permeten dividir i dissenyar la forma de les finestres. Amb els contenidors es poden organitzar, repartir i alinear els diferents elements gràfics que usarem. S'estructuren en forma d'arbre, de manera que el moviment i el canvi de mida d'un element afectarà tots els seus fills. Podem destacar els contenidors següents:

- GtkAlignment: caixa per a controlar l'alineació i la mida de la caixa filla.
- GtkHBox: caixa contenidora horitzontal.
- GtkVBox: caixa contenidora vertical.
- GtkHButtonBox: caixa contenidora per a alinear els botons horitzontalment.
- GtkVButtonBox: caixa contenidora per a alinear els botons verticalment.
- GtkFixed: contenidor per a col·locar els diferents *widgets* en posicions fixes.
- GtkHPaned: caixa amb dues parts horitzontals que l'usuari pot adaptar.
- GtkVPaned: caixa amb dues parts verticals que l'usuari pot adaptar.
- GtkLayout: caixa amb barres que llisquen.
- GtkNotebook: caixa amb diferents pestanyes.
- GtkTable: caixa contenidora dividida en una taula.
- GtkExpander: caixa contenidora que pot amagar els seus fills.

1.4.2. Llistat de *widgets*

Els *widgets* principals són els següents:

- GtkWindow: objecte principal en què podem afegir caixes contenidores. Representa una finestra.
- GtkDialog: finestra emergent.
- GtkMessageDialog: finestra emergent amb un missatge.

El conjunt d'objectes gràfics que podem usar dins les caixes contenidores són els següents:

- GtkAccelLabel: etiqueta amb acceleració de teclat.
- GtkImage: una imatge.
- GtkLabel: etiqueta amb una petita quantitat de text.
- GtkProgressBar: barra de progrés.
- GtkStatusbar: barra d'estat.
- GtkStatusIcon: mostra una icona en la barra d'estat del sistema.
- GtkButton: botó per a polsar.
- GtkCheckButton: botó per a escollir una opció.
- GtkRadioButton: botó per a escollir una opció entre diferents opcions.
- GtkToggleButton: botó de posició fixa.
- GtkEntry: entrada de text en una línia.
- GtkHScale: selector horitzontal d'un valor escalable.
- GtkVScale: selector vertical d'un valor escalable.
- GtkSpinButton: permet obtenir un valor real o numèric.
- GtkTextView: àrea de text amb un *buffer* i diferents opcions de formatació.
- GtkTreeView: arbre o llista d'elements.
- GtkComboBox: selecció d'un element en una llista.
- GtkMenu: menú.
- GtkMenuBar: barra de menú.
- GtkMenuItem: element d'un menú.
- GtkToolbar: barra de botons.
- GtkToolButton: botó de la barra.
- GtkColorSelectionDialog: finestra per a seleccionar un color.
- GtkFileChooserDialog: finestra per a seleccionar un arxiu.
- GtkFileChooserWidget: element empotrable en una caixa per a seleccionar un arxiu.
- GtkFontSelectionDialog: finestra per a seleccionar una font.
- GtkHSeparator: línia separadora horitzontal.
- GtkVSeparator: línia separadora vertical.
- GtkCalendar: mostra un calendari per a escollir una data.
- GtkDrawingArea: element per a dissenyar un *widget* per a l'usuari lliure. Àrea de dibuix.
- GtkTooltips: permet afegir ajudes als elements.
- GtkAccessible: permet afegir accessibilitat als elements.

1.5. Usabilitat

1.5.1. Introducció

El terme *usabilitat*, adaptat de l'original anglès *usability*, rep vàries definicions*, encara que totes aquestes definicions coincideixen en l'estudi del disseny d'interfícies en què l'usuari pugui aconseguir els seus objectius de la manera més efectiva, eficient i satisfactòria possible.

* http://es.wikipedia.org/wiki/usabilidad

Considerarem que una interfície és la connexió que interrelaciona dos sistemes heterogenis.

En el nostre cas, les interfícies permetran la interacció entre una aplicació informàtica i un usuari humà. D'aquesta manera, en el procés de disseny de la interfície d'una aplicació no solament és necessari incloure l'usuari final, sinó també el context en què s'usarà l'aplicació.

Actualment, la usabilitat ocupa un paper molt destacat en el desenvolupament del programari, al mateix nivell que paràmetres com el rendiment i l'escalabilitat. Es va popularitzar amb l'aparició de les pàgines web, amb impulsors tan destacats com Jacob Nielsen*, encara que la seva aplicació a programes d'escriptori s'ha tingut en compte especialment des de l'aparició dels entorns gràfics i el paradigma "What You See Is What You Get" (WYSIWYG). Aquest paradigma emfatitza la relació entre la simbologia usada i el resultat obtingut, com a mitjà per a facilitar a l'usuari la interpretació intuïtiva de les eines abstractes que utilitza. Un dels treballs més destacats vinculats al món del programari lliure és la *Guía para la interfaz*, de Gnome**.

* http://www.useit.com

** http://developer.gnome.org/projects/gup/hig/2.0

1.5.2. Factors

Els factors amb què es pretén mesurar d'una manera objectiva l'ús del programari per part d'un usuari són la seva efectivitat, eficiència i satisfacció.

Com a **efectivitat** entendrem la precisió i plenitud amb què s'arriba a un objectiu. Serem més efectius si aconseguim obtenir uns resultats més propers als objectius que ens havíem plantejat.

D'altra banda, direm que l'**eficiència** és la relació de recursos emprats respecte de l'efectivitat aconseguida. D'aquesta manera, entre dos mètodes igual d'efectius per a aconseguir un mateix objectiu, considerarem més eficient el que requereixi menys recursos per a aconseguir-lo.

Finalment, considerarem **satisfacció** l'absència d'incomoditat i l'actitud positiva en l'ús d'un programa. Un usuari pot millorar la seva satisfacció respecte d'un programa si percep que pot fer la seva feina d'una manera més efectiva i eficient, però també si se sent còmode perquè li resulta intuïtiu.

1.5.3. Beneficis

Els beneficis que es volen obtenir en tenir en compte la usabilitat durant el desenvolupament del programari comencen amb la satisfacció de l'usuari en utilitzar les aplicacions, atès que augmenta la seva eficiència a l'hora d'obtenir els seus objectius.

Aquest augment de la satisfacció i productivitat dels usuaris repercuteix en un nombre menor de peticions de redisseny i manteniment del programari una vegada entra en producció. De la mateixa manera, els costos d'assistència es redueixen i, en conjunt, obtenim uns costos inferiors d'explotació.

A més de reduir els costos d'explotació, els costos de formació i de documentació també disminueixen, atès que la corva d'aprenentatge d'ús del programari també és menor.

En conjunt, la dedicació a factors d'usabilitat durant el desenvolupament del programari redueix els costos en fases posteriors i augmenta la qualitat del producte final.

1.5.4. Metodologia de treball

Les metodologies utilitzades per a incloure els aspectes d'usabilitat en el desenvolupament del programari tenen en compte la participació dels usuaris finals en la fase de disseny de l'aplicació (*user testing*) en la simulació del context en què s'utilitzaran els programes (escenaris).

Aquests elements ens ajudaran a prendre les decisions sobre el disseny de l'estructura subjacent de la informació. Normalment, es vincula la usabilitat al component estètic de l'aplicació, però aquest component està íntimament unit a la concepció de la informació que manipula el programari. Per tant, un canvi en la forma en què cal presentar la informació a l'usuari pot tenir repercussions en les fases inicials d'anàlisi i disseny d'una aplicació.

El procés que s'ha de seguir consisteix bàsicament en el següent:

1) definir les necessitats de la informació,
2) definir l'estructura de la informació,
3) definir la presentació de la informació,
4) contrastar i revisar els punts anteriors amb les proves d'usuari.

Cal fer proves amb usuaris finals (*user testing*) mitjançant el que es coneix com "prototipus de l'aplicació". És a dir, l'estudi de la interacció dels usuaris amb simulacions del comportament final del programari. El disseny inicial es fa normalment mitjançant mètodes heurístics, que inclouen l'anàlisi d'experts en un determinat tipus d'aplicació.

Per a fer les proves amb usuaris finals és molt important determinar en quin context s'han de desenvolupar (escenari), atès que aquest factor influeix notablement en la percepció que en té l'usuari.

La metodologia de treball consisteix a acotar les proves que s'han de fer, detallant les dades que es volen recollir i la seva realització. El model usat a Gnome* es pot considerar vàlid per a la majoria de les aplicacions.

* http://developer.gnome.org/projects/gup/templates/fui_template.html

1.5.5. Guies bàsiques

Com s'ha vist fins al moment, la usabilitat preveu un ventall d'aspectes fortament vinculats a l'usuari i al context d'ús, de tal manera que no és possible donar consells genèrics, sinó solament indicacions molt específiques per a binomis usuari-context determinats.

Malgrat tot, és possible establir una guia bàsica present en gairebé tots els treballs d'usabilitat:

- **L'objectiu sempre és l'usuari**. Per molta lògica que hi trobem com a desenvolupadors, si l'usuari no se sent efectiu, eficient i satisfet amb l'aplicació desenvolupada caldrà replantejar la nostra "lògica".

- **L'aplicació s'ha de vincular amb el seu context.** És necessari usar el vocabulari i les metàfores pròpies del context (i no les relacionades amb l'arquitectura informàtica de l'aplicació).

- **L'aplicació ha de ser consistent.** Les operacions i la manera d'expressar-les han de ser consistents internament i amb el programari ja existent. D'aquesta manera l'usuari pot reutilitzar els conceptes i la lògica ja adquirida.

- **L'usuari ha d'estar informat.** És convenient oferir a l'usuari informació sobre el que està passant en tot moment, usant un llenguatge simple però sense obviar els detalls.

- **La simplicitat ha de predominar**. Sempre que sigui possible, s'ha de buscar la manera més simple d'expressar la informació. Els paràmetres d'estètica poden millorar notablement el factor satisfacció de l'usuari.

- **L'usuari ha de conservar el control**. Per damunt de les operacions automatitzades, l'usuari ha de conservar el control per poder especificar què s'ha de fer exactament en cas que ho consideri convenient.

- **L'aplicació ha de soportar les errades de l'usuari**. Atès que es treballa amb usuaris humans, l'error no ha de ser una excepció, sinó un component més del procés d'interacció.

- **La interacció directa ha de ser-hi present**. Quan sigui possible expressar una acció mitjançant la interacció directa amb un element de l'estructura d'informació, és convenient oferir aquesta opció.

- **Sempre s'ha d'aspirar al major nombre possible d'usuaris.** Els aspectes d'internacionalització (traducció) i localització (adaptació de formats) del programari, juntament amb l'accessibilitat (accés de persones amb discapacitats), s'han de tenir en compte des de les primeres fases de disseny.

2. Mono

2.1. Instal·lació

El paquet de programari de Mono té moltes dependències i subpaquets possibles. Aquí tenim els principals:

- Mono: compilador, màquina virtual i conjunt de classes bàsiques que ens permeten utilitzar les funcions de l'especificació ECMA. En aquest conjunt de biblioteques hi ha algunes d'afegides per Mono i Novell, com ara l'accés a LDAP i una implementació d'XML pròpia.
- Gtk-sharp: conjunt de biblioteques que s'usen en la programació d'aplicacions d'escriptori que utilitzen l'entorn GTK. En aquest conjunt hi ha tot el codi dels enllaços a les biblioteques del paquet GTK, Gnome, Gnomevfs, vte, rsvg, Glade i Gconf.
- XSP: servidor de pàgines web que permet tenir connexions https i http. Aquest programa serveix per apoder executar codi aspx, ashx i asmx sense haver d'instal·lar un servidor de pàgines web de l'estil d'Apache. Robust, molt útil per a depurar i lleuger de càrrega.
- Mod_mono: enllaç des del servidor de pàgines web Apache a les biblioteques d'XSP per poder servir pàgines web en ASP.NET.
- Monodoc: programa amb la documentació de les classes de Mono. Hi ha les realitzades pel projecte Mono més un conjunt de biblioteques extra de gent externa. Quan es fa una biblioteca es pot documentar de manera que aparegui en el Monodoc automàticament i que aquesta documentació es pugui editar des del mateix programa.
- Mono-tools: eines usades per al desenvolupament, com ara gunit.

Gunit

Gunit és el programa per a l'ajuda en la programació d'unitats.

- Gecko-sharp: biblioteca per a la creació de *widgets* de navegació web usant el motor Gecko, el mateix que s'usa en els navegadors de Mozilla.
- Gtksourceview-sharp: biblioteca per a la creació de *widgets* d'edició de codi font. Amb utilitats de reconeixement de llenguatges, colors i números de línia.
- Boo: llenguatge semblant en sintaxi a Python, però amb una connexió directa a les biblioteques de Mono i a tots els llenguatges suportats per aquesta plataforma.
- IKVM: màquina virtual per a poder executar codi i biblioteques fetes en Java dins de la màquina virtual de Mono. Aquest projecte permet cridar des de Java les biblioteques de la plataforma Mono.

- Mono-debugger: versió de la màquina virtual per a depurar el codi. En aquesta versió, es pot veure l'estat de la memòria i variables en moment d'execució. També amb l'ajuda de diferents IDE es poden usar els punts d'anclatge en el codi.

2.1.1. Nadiu

Sense haver de compilar totes les biblioteques i programes, la gent de Mono ha desenvolupat un sistema per a poder instal·lar Mono amb un instal·lador en plataformes Linux. Una vegada ens hem baixat aquest arxiu `.bin` podem executar-lo directament i ens descomprimirà i instal·larà tot el que calgui per executar Mono, Monodevelop i les biblioteques GTK.

Instal·lador

Podeu baixar Mono de la pàgina http://www.mono-project.com/Downloads.

2.1.2. Debian

En la distribució Debian hi ha paquets per a totes les opcions de la plataforma Mono. En el moment d'escriure aquest subapartat, estan en la distribució Etch i es poden instal·lar usant l'aplicació Synaptic o des de consola amb Aptitude.

Els paquets que es requereixen són els següents: mono, mono-assemblies-base, mono-classlib-1.0, mono-classlib-2.0, mono-common, mono-devel, mono-gac, mono-gmcs, mono-jay, mono-jit, mono-mcs, mono-utils, mono-xsp, monodevelop, monodevelop-boo, monodevelop-java, monodevelop-nunit, monodevelop-versioncontrol, monodoc, monodoc-base, monodoc-browser, monodoc-dbus-1-manual, monodoc-gecko2.0-manual, monodoc-gtk2.0-manual, monodoc-gtksourceview2.0-manual, monodoc-manual, monodoc-nunit-manual, libgecko2.0-cil, libglib2.0-cil, libglade2.0-cil, libgnome2.0-cil, libgtk2.0-cil, libgtksourceview2.0-cil, libdbus-cil, libgmime2.1-cil, libevolution-cil, libvte2.0-cil, glade-gnome-2.

2.1.3. Fedora

En el moment d'esciure aquest subapartat, hi ha disponible Fedora Core 4, en què no hi ha inclòs Mono per un problema de llicències.

A Fedora Core 5 ja hi ha instal·lat Mono per defecte.

Per a executar Mono, hi ha diferents maneres de fer-ho: una amb Novell's Red Carpet. Una vegada està instal·lat l'rpm de l'aplicació, podeu descarregar-vos Mono usant:

```
rug sa http://go-mono.com/download
rug sub mono-1.1-official
rug sub gtk-sharp-official
rug in mono-complete gtk-sharp
```

Web recomanada

Podeu descarregar Novell's Red Carpet des de http://www.snorp.net/files/packages/rcd-fc3.

Si voleu instal·lar més programes i biblioteques, hi ha una llista dels rpm disponibles a http://www.go-mono.com/download/fedora-4-i386

2.2. Eines bàsiques

L'entorn de desenvolupament de Mono inclou un gran nombre d'eines bàsiques per al desenvolupament tant des de consola de text com mitjançant interfície gràfica. En general, la qualitat d'aquestes eines és bona, tot i que l'apartat de la documentació és el que està més endarrerit.

Les eines bàsiques que es distribueixen amb la plataforma són les següents:

- mono: màquina virtual que interpreta el llenguatge CIL (inclou profiler).
- mcs: compilador de C#.
- monop: utilitat per a consultar la documentació abreujada mitjançant la línia d'ordres.
- monodevelop: entorn integrat de desenvolupament.
- monodoc: aplicació gràfica per a visualització i edició de la documentació.
- xsp: servidor mínim per a generar pàgines web des de CLI.
- mod_mono: mòdul per a integrar-se amb el servidor Apache.
- mdb: depurador de codi mitjançant la línia d'ordres.
- monodis: desassemblador de CIL.

Exemple de desenvolupament en consola de text

Compilar: `mcs Hola.cs`
Executar: `mono Hola.exe`
Depurar: `mcs -debug Hola.cs ; mdb Hola.exe`
Profiler: `mono --profile Hola.exe`
Desassemblar: `monodis Hola.exe`
Documentació: `monop System.Console`

Exemple de desenvolupament en entorn gràfic

- Monodevelop: eina per al desenvolupament per a poder dissenyar interfícies gràfiques i per a poder implementar solucions basades en C#, Java i Python.
- Gnunits: sistema per a monitoritzar l'execució de tests.
- Monodoc: sistema de documentació. Inclou tota l'API de Mono i podeu afegir la documentació de les vostres pròpies classes.

2.3. C# bàsic

El llenguatge C# està definit en l'estàndard Ecma-334*. Hi ha diverses edicions a partir de l'original publicada el 2000, basada en la proposta de Microsoft, Hewlett-Packard i Intel. Tot i que la definició fa referència a l'entorn Common Language Infrastructure (CLI), C# no hi està vinculat, i per tant hi poden haver implementacions que no el requereixin.

*** http://www.ecma-international.org/publications/standards/Ecma-334.htm**

Aquest llenguatge de programació deriva de la família de llenguatges imperatius que inclou C, C++ i Java. És orientat a objectes, amb suport per a la comprovació de tipus (*strong type checking*), límits dels vectors (*array bounds checking*), ús de variables no inicialitzades i gestió de la memòria no usada (*garbage collection*). L'esperit és proporcionar un llenguatge de propòsit general per

a desenvolupar aplicacions distribuïdes altament portables entre diferents plataformes. També ha d'harmonitzar tant amb arquitectures i sistemes operatius complexos i de gran escala com amb sistemes mínims empotrats.

En aquest punt, revisarem breument les característiques bàsiques del llenguatge i partirem de la base que disposeu de coneixements previs en llenguatges orientats a objectes, com C++ o Java.

2.3.1. Sentències

La sintaxi de les sentències bàsiques és la mateixa que en C++, tot i que inclou alguna sentència nova, com el `foreach`, que normalment estava present en llenguatges considerats de desenvolupament ràpid:

- Comentaris: `//` (per a una sola línia), `/*` (per a múltiples línies) `*/`
- Assignacions: `i = a * 3;` (no permet confondre-les amb la comparació `==`, ja que la comprovació de tipus indica que el resultat no és booleà)
- Alternatives: `if-else`, `switch` (permet usar `switch` amb tipus no bàsics, per exemple `strings`)
- Iteraccions: `while`, `do-while`, `for`, `foreach` (aplicable a tots els objectes que implementin la interfície `IEnumerable`)
- Salts: `break`, `continue`, `goto`, `return`, `throw` (mecanisme d'excepcions)

Exemple `foreach`

```
int[] a = new int[]{1,2,3};
foreach (int b in a)
  System.Console.WriteLine(b);
```

2.3.2. Tipus elementals

Igual que altres llenguatges, C# disposa de tipus simples amb els quals es pot treballar directament amb el valor i tipus compostos als quals normalment s'accedeix mitjançant la seva referència. A Java, tots els paràmetres es passen per valor, tenint en compte que qualsevol variable de tipus compost en realitat és una referència a les dades en memòria. En canvi, a C# és possible passar les dades tant per valor com per referència, siguin del tipus que siguin.

- Tipus simples: contenen literalment el valor i generalment es guarden en la pila.

- `sbyte, short, int, long, byte, ushort, uint, ulong, float, double, decimal, char, bool, structs`

- Tipus compostos: contenen la referència a un objecte i generalment es guarden en el *heap*.

Taula de tipus simples:

Tipus	Nom Mono/.NET	Signe	Bytes	Rang
bool	System.Boolean	No	1	true / false
byte	System.Byte	No	1	0 .. 255
sbyte	System.SByt	S	1	-128 .. 127
short	System.Int16	S	2	-32.768 .. 32.767
ushort	System.Uint16	No	2	0 .. 65535
int	System.Int32	S	4	-2.147.483.648 .. 2.147.483.647
uint	System.Uint32	No	4	0 .. 4.394.967.395
long	System.Int64	S	8	-9E18..9E18
ulong	System.Uint64	No	8	0 .. 18446744073709551615
float	System.Single	S	4	+/-1.5E-45..+/-3.4E38 (7 sig)
double	System.Double	S	8	324 .. +/-1.7E308 (7 sig)
decimal	System.Decimal	S	12	+/-1E-28..+/-7.9E28 (28/29 sig)
char	System.Char		2	cualquier Unicode (16b)

De la mateixa manera que Java, la biblioteca de C# proporciona uns tipus compostos (classes d'objectes) per a encapsular els tipus simples que hi ha en el llenguatge. Per exemple, hi ha la classe `Integer`, capaç de contenir les mateixes dades que el tipus simple `int`. Aquestes classes s'utilitzen per a allotjar les funcions relacionades amb els tipus simples, com ara interpretar el contingut d'un `string` com un valor enter.

A C#, a diferència de Java, les conversions entres els tipus simples i els seus equivalents compostos són implícites, i es coneixen amb el nom de *boxing*. En aquest cas, es copia el valor simple de la pila en el compost que es guarda en el *heap*.

En la concepció de C# es van recuperar algunes característiques de C++ que s'havien rebutjat a Java. Per exemple, hi ha els tipus enumerats, els punters i també les estructures. L'ús de punters s'ha de fer amb precaució, ja que el `garbage collector` pot canviar dinàmicament la ubicació dels objectes. Hi ha els modificadors `unsafe` i `fixed` per a indicar que determinats fragments de dades els gestiona el programador i no la màquina virtual.

2.3.3. Entrada/sortida

La manipulació dels fluxos de dades d'entrada i sortida es fa per mitjà de l'abstracció de les classes `stream`, igual que en Java o C++, encara que a C# s'ha conservat la simplicitat d'aquest darrer i, per tant, la lectura i l'escriptura bàsica resulta molt més senzilla. Un dels factors que més la simplifica és el fet que a C# no és obligatori recollir les excepcions, de manera que el codi més simple pot ignorar-les suposant que no hi haurà errors.

- Treball amb la consola: ús de la classe `Console` amb els mètodes `Read`, `ReadLine`, `Write`, `WriteLine`, `SetIn`, `SetOut` i `SetError`.

- Treball amb arxius: ús de la classe `File` amb els mètodes `Exists`, `Delete`, `Copy`, `Move` i`GetAttributes`.
- Arxius de text: mètodes `OpenText`, `CreateText` i `AppendText`.
- Arxius en binari: ús de les classes `FileStream`, `StreamReader` i `StreamWriter`.
- Treball amb el sistema d'arxius: ús de la classe `Directory` i `DirectoryInfo` amb mètodes com `GetCurrentDirectory` o `GetFiles`.

Exemple de maneig d'arxius de text

```
using System;
using System.IO;

public class ArxiusText {
  public static void Main(string []args){
    Console.Write("Nom de l'arxiu: ");
    string nom = Console.ReadLine();
    Console.WriteLine("\tContingut de l'arxiu {0}:", nom) ;

    StreamReader leer= File.OpenText(nom);
    string lin = leer.ReadLine() ;
    while (lin != null ) {
      Console.WriteLine(lin) ;
      lin = leer.ReadLine() ;
    }
    leer.Close() ;

    StreamWriter escriure= File.AppendText(nom);
    escriure.WriteLine("Afegint contingut");
    escriure.Write("l'arxiu {0}\n", nom);
    escriure.Close();
  }
}
```

Exemple de maneig d'arxius binaris

```
using System;
using System.IO;

public class FitxersStream {
  public static void Main(string []args){
    Console.Write("Nom de l'arxiu: ");
    string nom = Console.ReadLine();

    FileStream llegir = new FileStream(nom,
                        FileMode.OpenOrCreate,
                        FileAccess.Read,
                        FileShare.Read );
    while (llegir.Position < llegir.Length){
      Console.Write(( char )leer.ReadByte()) ;
    }
    llegir.Close();

    FileStream fit = new FileStream(nom,
```

```
                        FileMode.OpenOrCreate,
                        FileAccess.Write,
                        FileShare.None );
    StreamWriter escriure = new StreamWriter(fit);
    escriure.BaseStream.Seek(0, SeekOrigin.End);

    escriure.WriteLine("Afegint contingut");

    escriure.Close() ;
  }
}
```

2.3.4. Cadenes

El tractament de les cadenes de caràcters a C# segueix la tendència a la simplificació que es va iniciar amb `string` a C++, tot i que incorpora més flexibilitat sintàctica normalment present en llenguatges de programació ràpida. Podem observar aquest factor en l'exemple de codi C# per a la inicialització de cadenes, en què, entre altres detalls, veiem la possibilitat d'usar literals de cadenes de caràcters tipus `verbatim`. És a dir, que es respecta el format del contingut fins i tot per a caràcters especials i salts de línia.

Exemple d'inicialització de cadenes

```
using System;

class CadenesDeclaracio {
  public static void Main() {
    String may = "crida automtica al constructor";
    string min = String.Format("{{ Num={0,2:E} }}", 3);

    Console.Write("Sinnimos String({0}) = string({1})\n",
                  may.GetType(), min.GetType());
    Console.WriteLine(min);

    string s1 = "Hola", s2 = "Hola";
    Console.Write("Contenen el mateix:{0}\n", s1.Equals(s2));
    Console.WriteLine("La mateixa referència inicial: {0}",
                  Object.ReferenceEquals(s1,s2));
    s2 = "Adis";
    Console.WriteLine("Han canviat les ref: {0}",
                  Object.ReferenceEquals(s1,s2));
    Console.Write("Operador ==: {0}\n", s1==s2);

    string verba = @"amb ""verbatim"" ens estalviem
escapar determinats codis (\n, \t, \0...)
i podem incloure més d'una línia";

    Console.WriteLine(verba);
  }
}
```

Atès que a C# hi ha les propietats d'objecte i el seu accés indexat, la sintaxi per a accedir a un dels seus caràcters és tan senzilla com si es tractés d'un vector de caràcters: per exemple `s[3]`. De la mateixa manera, consultar-ne els atri-

Les propietats d'objecte a C# es tractaran en el subapartat 2.4.

buts deixa d'expressar-se com una crida a un mètode d'accés i s'expressa com una `property`: per exemple `s.Length`.

La resta de les operacions sobre cadenes es fa per mitjà de mètodes com ara els següents:

- Concatenació:
  - `String.Concat("Hola", "món")`
  - `s1+s2 // a C# és possible sobrecarregar operadors com a C++`
  - `String.Join(" ", {"Hola","món"})`
- Subcadenes:
  - `"Bombolla".EndsWith("uja")`
  - `"Bombolla".StartsWith("Bur")`
  - `"Bombolla".IndexOf("rb")`
  - `"Bombolla".IndexOfAny({'a','o'})`
  - `"Bombolla".Substring(2,4)`
- Creació de cadenes noves:
  - `"Bombolla".Insert(1,"al")`
  - `"Bombolla".Replace('u','o')`
  - `"Bombolla".ToLower()`
  - `"Bombolla".Trim({'a','o'})`
- Expressions regulars: mitjançant els mètodes de la classe `System.Text.RegularExpressions`

De la mateixa manera que Java, els *strings* de C# són immutables, és a dir, una vegada han estat inicialitzats ja no és possible canviar-los el contingut (les operacions `Insert`, `Replace`, `ToLower`, etc. creen nous *strings*). Per aquest motiu, si es necessita treballar amb cadenes que es puguin alterar, hi ha la classe `System.Text.StringBuilder`.

2.3.5. Vectors i taules

El llenguatge C# conserva la distinció entre vectors unidimensionals i taules multidimensionals. Igual que a Java, a C# es poden crear matrius inserint recursivament vectors unidimensionals dins d'altres vectors. Això permet crear matrius regulars, però també matrius irregulars tant en mides com en tipus de dades contingudes. Però si simplement volem una matriu regular amb dades homogènies, podem crear una taula multidimensional, que és més eficient en memòria i en temps.

La sintaxi dels vectors unidimensionals és bàsicament la mateixa que a Java:

- Definició: mitjançant tipus anònims
  - `int [] a;`
- Inicialització:
  - `int [] a = new int[10];`

- `int [] a = {1,2,3}; // 3 elements, índexs del 0 al 2`

• Manipulació:
– longitud: `a.Length`
– còpia: `Array.Copy, a.CopyTo`
– cerca: `Array.BinarySearch, a.IndexOf`
• Manipulació: `a.Reverse`, `Array.Sort`
• Recorregut: usant `foreach`

Si creem matrius basant-nos en vectors unidimensionals, la sintaxi també és la mateixa que a Java:

```
• int [][] matriu;
• matriu = new int [2][3]; // inicialitzem a 2x3 regular
• matriu = new int [3][]; // o inicialitzem solament les
  "primeres" dimensions
• niu[1] = new int [7]; // i posteriorment les "següents"
• niu[1] = {3,6,4,1,8}; // també podem fer-ho amb literals
```

Per últim, si creem les matrius amb les taules multidimensionals la sintaxi se simplifica:

```
• int [,] matriu = new int[2,3]; // declaració de 2 dimensions
• int [,] matriu = {{1, 2, 3}, {4, 5, 6}}; // literals
• matriu[1,1] = 7; // accés als elements
```

A part dels vectors i les taules, la biblioteca de C# també proporciona un conjunt de classes per a treballar amb col·leccions d'elements a l'estil de l'STL de C++ i les Collections de Java.

2.4. C# mitjà

En aquest segon subapartat sobre C# se n'exposen les possibilitats com a llenguatge de programació orientat a objectes. En general, la seva sintaxi simplifica la disponible per a llenguatges com C++ o Java, però sense perdre control, fins i tot afegint alguns modificadors nous. En molts casos, en la definició del llenguatge es va optar per definir modificadors per defecte, de manera que una persona que s'iniciï en la programació, que no coneix tots els conceptes associats amb els atributs, no està forçada a usar-los, com succeeix en altres llenguatges.

2.4.1. Definició de classes

En definir les classes a C# especificarem en primer lloc els atributs i en segon lloc els modificadors d'aquestes classes.

Els atributs no es refereixen als membres interns de la classe, sinó que es tracta d'una forma flexible de poder estendre els modificadors d'aquesta classe. Per exemple, els modificadors de què podem disposar són els següents:

- `public`: accessible des de tota la resta del codi.
- `private`: accessible només des del mateix `namespace`.
- `internal`: accessible només des del mateix `assembly`.
- `new`: per a poder usar el mateix nom que una classe heretada en classes declarades dins d'unes altres.
- `protected`: accés des de la classe contenidora i les seves derivades.
- `abstract`: per a classes que no es poden instanciar, s'han d'heretar des d'una classe nova.
- `sealed`: per a indicar que no es pot derivar.

I, amb els atributs, podem usar els que hi ha disponibles a les biblioteques estàndard, per exemple:

- `Obsolete`: perquè el compilador indiqui que la classe esmentada no estarà disponible a les biblioteques a mitjà termini. De fet, fins i tot podem demanar al compilador que es negui a compilar si s'utilitza alguna classe obsoleta.

- `Serializable`, `NonSerialized`: per a especificar si la classe es pot seriar en una cadena de caràcters per a ser emmagatzemada o transmesa.

O bé, definir atributs nous segons les necessitats de gestió del codi que tinguem derivant directament de la classe: `System.Attribute`.

Les classes es poden declarar dins d'altres classes, de manera que les internes poden accedir als atributs de les seves contenidores. En llenguatge C# és possible instanciar una classe interna sense necessitat que hi hagi una instància de l'externa.

2.4.2. Definició d'atributs interns de classe

En el disseny de C# es va tenir en compte la càrrega que normalment suposa al programador definir els mètodes d'accés a les dades de la classe. Amb aquesta idea, es distingeixen els atributs interns de la classe següents:

- `Fields`: es corresponen als camps interns habituals de la classe pensats perquè siguin accessibles només per mitjà del codi de la classe. Poden dur els modificadors següents:

– `static`: per a indicar que són dades associades a la classe, no a l'objecte, comuns entre totes les seves instàncies i disponibles encara que no hi hagi cap classe.

– `readonly`: solament s'hi pot donar valor un cop; si es torna a intentar, es genera un error.

– `const`: el valor s'assigna en temps de compilació, per tant solament es poden usar amb tipus de valor no referència.

- `Properties`: corresponen a la sintaxi pensada per a la visió dels camps des de l'exterior de l'objecte. Es permet declarar-ne els mètodes d'accés buits en cas que no s'hagi de fer cap acció en especial.

– Declaració mínima: `public int Property{ get{} set{} }}`

– Accés des de l'exterior de la classe: `obj.Property=5; a=obj.Property`

Per tant, a C# els mètodes d'accés no s'utilitzaran com a la resta dels mètodes, sinó que la seva sintaxi d'ús se simplifica com si fossin camps interns públics. Aquesta simplificació és solament sintàctica, ja que la premissa d'aïllar els camps interns de la seva visió des de fora de l'objecte es continua complint.

Un altre detall introduït a C# és l'accés a les dades d'una classe amb una sintaxi equivalent a l'ús de taules:

- `Indexers`: mètodes per a proporcionar un accés a les dades amb una sintaxi de taula.

– Declaració: `public int this[int i]{ get{...} set{...} }`
– Accés des de l'exterior de la classe: `obj[3]=5; a=obj[5]`

```
using System;

public class ClasesDeCampos {
  int field   = 0;
  int []taula = {1, 2, 3, 4};

  public int Property {
      get { return this.field; }
      set {
        if (value > 0)
          this.field = value;
      }
  }
  public int this[int i] {
      get {
        if (i>=0 && i< tabla.Length)
          return this.taula[i];
        else
          return 0;
      }
      set {
        if (i>=0 && i< taula.Length)
          this.taula[i] = value;
      }
  }
  static void Main() {
```

```
        ClasesDeCamps c = new ClassesCamps ();

        Console.WriteLine("Field: " + c.field);

        Console.WriteLine("Property: " + c.Property);
        c.Property = 5;
        Console.WriteLine("Property(5): " + c.Property);
        c.Property = -5;
        Console.WriteLine("Property(-5): " + c.Property);

        Console.WriteLine("Indexer[2]: " + c[2]);
        Console.WriteLine("Indexer[9]: " + c[9]);
        c[2] = 5;
        Console.WriteLine("Indexer[2]: " + c[2]);
        c[9] = 5;
        Console.WriteLine("Indexer[9]: " + c[9]);
      }
    }
```

2.4.3. Mètodes

La declaració de mètodes, igual que les classes, pot dur associada diversos atributs i modificadors:

- Atributs: `Conditional`, `Obsolete`, etc.

`virtual`, `override` i `new` s'expliquen en el subapartat 2.4.4, dedicat al polimorfisme.

- Modificadors:

– `abstract`: mètode buit que s'ha d'implementar en les classes derivades.
– `static`: associat a la classe, no es necessita cap instància per a usar-lo i, per tant, no pot usar les dades internes d'un objecte.
– `extern`: definit externament (generalment en un altre llenguatge).

- `[DllImport(''util.dll'')] extern int met(int s);`

– `virtual`, `override`, `new`.

```
#define DEBUG
using System;
using System.Diagnostics;

public class ExempleAtributs {

  [Obsolete("Useu el mètode Nou", false)]
  static void Antic( ) { Console.WriteLine("Antic"); }

  static void Nou( )   { Console.WriteLine("Nou"); }

  [Conditional("DEBUG")]
  static void Depura() { Console.WriteLine("Depura"); }

  public static void Main( ) {
    Antic( );
    Depura();
  }
}
```

A més, la declaració dels paràmetres comporta algunes novetats respecte dels seus modificadors:

- `ref`: indica que el paràmetre es rep per referència, de manera que si en canviem el contingut canvia el de la variable d'entrada. Dins del cos del mètode, el paràmetre s'usa sense cap modificador especial, com a C++.

- `out`: igual que `ref`, però el compilador s'assegura que en modifiquem el contingut abans d'acabar el mètode.

- `params`: indica que la resta de la llista de paràmetres és un nombre indeterminat d'arguments del tipus indicat.

Com en la resta dels llenguatges orientats a objectes de la mateixa família, podem sobrecarregar els mètodes usant el mateix identificador amb diferent llista de paràmetres.

A diferència de Java, a C# es permet sobrecarregar els operadors tal com es podia fer aC++.

```
using System;

public class ExempleMetodes {
  static void Entrada   (    int e) { e = 1; }
  static void Sortida (out int s) { s = 2; }
  static void Referencia(ref int r) {
    Console.WriteLine("Refer. r="+r); r = 3; }

  static void Sobrecarrega(out int  x){ x=4; }
  static void Sobrecarrega(out char y){y='B';}

  private int f=5;
  static int operator+(ExempleMetodes a,
                       ExempleMetodes b) {
    return a.f+b.f;
  }

  static void Main() {
    int e=10, s=20, r=30, n=40;
    char c='A';

    Console.WriteLine("e({0}) s({1}) r({2})",
                      e, s, r);
    Entrada(e); Sortida(out s); Referencia(ref r);
    Console.WriteLine("e({0}) s({1}) r({2})",
                      e, s, r);

    Console.WriteLine("n({0}) c({1})", n, c);
    Sobrecarrega(out n); Sobrecarrega(out c);
    Console.WriteLine("n({0}) c({1})", n, c);

    ExempleMetodes u = new ExempleMetodes();
    ExempleMetodes v = new ExempleMetodes();
    Console.WriteLine("u+v="+(u+v));
  }
}
```

2.4.4. Herència i polimorfisme

L'esquema d'herència entre classes és el mateix que es va adoptar a Java: no es permet l'herència múltiple, però hi ha el concepte d'interfície, que sí que permet l'herència múltiple. D'aquesta manera, no hi ha el problema d'escollir entre diferents implementacions d'un mateix mètode, ja que les interfícies solament contenen els prototipus dels mètodes.

La paraula reservada per a referir-se a la classe mare és `base`, que es pot emprar directament en el prototipus del constructor igual que a C++.

També és possible definir el destructor, que el `GarbageCollector` cridarà en el moment en què decideixi eliminar l'objecte. Si volem tenir control sobre aquest instant, hem d'usar la paraula reservada `using`, que denota un bloc de sentències en què els objectes que hi ha declarats en el seu interior seran destruïts en el moment en què acabi l'execució del bloc.

Tal com es comentava en la definició de mètodes, és possible especificar-los modificadors per a controlar-ne el comportament en el moment en què es crea una classe derivada:

- `abstract`: mètode sense implementació, que s'ha de fer en les classes derivades.

- `virtual`: la classe mare indica que pot ser sobreescrita per les classes derivades.

- `override`: la classe derivada explicita que está sobreescrivint un mètode de la classe mare (que ha d'estar declarat com a `virtual`, `abstract` o `override`).

- `new`: la classe derivada explicita que sobreescriu un mètode de la classe mare, encara que la classe mare no indiqui que això pot ser possible.

```
using System;

public class ExempleHe {
  static void Main() {
    Filla x = new Filla("Pepa", 44,
                "Cardiologia", "Muntanya");

    Console.WriteLine("Objecte="+x);
    Console.WriteLine("x="+x.ToString());
  }
}
abstract class Mare {
  private string n;
  private int a;

  public Mare(string n, int a) {
    this.n = n;
```

```
    this.a = a;
  }

  public new virtual string ToString() {
    return "nom("+n+") anys("+a+")";
  }
}

class Filla : Mare, Doctora, Ciclista {
  private string e, t;

  public Hija(string n, int a, string e, string t)
         :base(n, a/2) {
    this.e = e;
    this.t = t;
    Console.WriteLine("Missatge de la mare : "+base.ToString());
  }
  public string Especialitat() { return e; }
  public string Tipus()        { return t; }
  public override string ToString() {
    return base.ToString()+" espec("+e+") tipus("+t+")";
  }
}

interface Doctora {
  string Especialitat();
}

interface Ciclista {
  string Tipus();
}
```

2.4.5. Delegats i esdeveniments

El mecanisme per al paradigma de programació orientada a esdeveniments de C# consisteix en l'existència de classes que deriven del tipus `Event` i mètodes que deleguen la seva crida a una col·lecció de mètodes indicats prèviament. D'aquesta manera, una classe pot publicar un esdeveniment i declarar un tipus de delegat perquè les classes interessades se sugscriguin a l'esdeveniment afegint-se a la col·lecció de mètodes del delegat. En el moment en què es produeix l'esdeveniment, la classe cridarà el delegat i aquest s'encarregarà de cridar consecutivament tots els mètodes subscrits a l'esdeveniment.

a) `Delegate`: per a delegar crides a altres mètodes.

- Funció:
  - guarda les referències a un conjunt de mètodes;
  - quan s'invoca, s'encarrega de cridar tots els mètodes que té referenciats;
  - qui crida al delegat no coneix els mètodes subscrits, simplement li delega la tasca de cridar-los.
- Declaració:
  - es poden declarar dins i fora d'una classe;
  - sempre deriven de la classe `System.MulticastDelegate`;
  - només és possible emmagatzemar mètodes amb la mateixa llista de paràmetres i tipus de retorn.

```
using System;

delegate void Recull (string cadena);

class AltresDelegats {
  public static Recull recollidor;

  public static void Main () {
    recollidor = new Recull(Tejat.RecullEstatic);

    Balco b = new Balco();
    // recollidor += new Recull(b.RecullNoValid);

Terra t = new Terra();
    recollidor += new Recull (t.RecullPublic);

    recollidor("aigua");
  }
}

class Teulada {
  public static void RecullEstatic (string cad) {
    Console.WriteLine ("Teulada.RecullEstatic: "+cad);
  }
}

class Terra {
  public void RecullPublic (string cad) {
    Console.WriteLine ("Terra.RecullPublic: "+cad);
  }
}

class Balco {
  public Balco () {
    AltresDelegats.recollidor += new Recull(RecullPrivat);
  }
  void RecullPrivat (string cad) {
    Console.WriteLine ("Balco.RecullPrivat: "+cad);
  }
  public void RecullNoValid (int enter) {
    Console.WriteLine ("Balco.RecullNoValid: "+enter);
  }
}
```

b) `Event`: mecanisme d'execució de mètodes que s'han subscrit a un esdeveniment concret.

- Els pot llançar el mateix codi o l'entorn.
- Normalment, la classe generadora no llança l'esdeveniment si no hi ha cap mètode subscrit.
- La classe que el publica conté:
  - un delegat tipus `void` que rep una referència a l'objecte que genera l'esdeveniment i les dades associades,
  - un esdeveniment del tipus delegat declarat anteriorment,
  - la classe que agrupa les dades associades a l'esdeveniment,
  - el codi que genera l'esdeveniment.
- La classe que subscriu l'esdeveniment conté:
  - el mètode que se subscriurà a l'esdeveniment,
  - el codi de subscripció a l'esdeveniment mitjançant la creació d'una instància del delegat.

```
using System;
using System.Collections;
using System.Threading;

public class EsdevenimentSatellit {
  static void Main () {
    Receptor receptor = new Receptor();
    Satellit satellit = new Satellit(receptor);
    receptor.EnMarxa();
  }
}

public class Receptor {
  public delegate void InterceptaEsdevenimentInfoRebuda(
                       object s, DadesEsdeveniment args);
  public event InterceptaEsdevenimentInfoRebuda
EsdevenimentInfoRebuda;

  public void EnMarxa() {
    Console.WriteLine("Receptor: en marxa esperarà 2 "+
                      "senyals de l'usuario...");
    for (int i=0; i < 2; i++) {
      Console.WriteLine("Receptor: espero usuari entre "+
                        "senyal...");
      string info  = Console.ReadLine();
      string tiempo = DateTime.Now.ToString();
      string datos = "[Rebuda info("+info+") hora("+temps+")]";
     Console.WriteLine("Receptor: genere esdeveniment info_rebuda
"+
                        "dades="+ dades);
     if (EsdevenimentInfoRebuda != null) // si hi ha algu subscrit
        EsdevenimentInfoRebuda(this, new
DadadeEsdeveniment(dades));
    }
  }
  public class DadesEsdeveniment : EventArgs {
    string dades;
    public string Dades { get { return dades; }
                          set { dades= value; }}
    public DadesEsdeveniment(string d){ dades = d; }
  }
}

public class Satellit {
  private Receptor receptor;

  public Satellite (Receptor receptor) {
    this.receptor=receptor;

    Console.WriteLine("Sat: nou");
    Console.WriteLine("Sat: subscribint el meu "+
                      "interceptador de l'esdeveniment "+
                       "info_rebuda del Receptor");
    receptor.Esdeveniment InfoRebuda +=
          new Receptor.InterceptaEsdevenimentInfoRebuda(
                               SatProcessaSenyal
              );
  }

  private void SatProcessaSenyal(object src,
                       Receptor.DadesEsdeveniment args) {
    Console.WriteLine("SatProcessaSenyal: "+
                      "de \"{0}\" "+
                      "dades={1}", src, args.Dades);
  }
}
```

2.4.6. *Structs, enums* i *namespaces*

A C# s'han recuperat les estructures de C++ que s'havien eliminat a Java. De fet, a C# les estructures es poden usar com a C++, però també poden incloure mètodes. En efecte, una estructura a C# és com una classe però de tipus valor. Per exemple, si assignem dues variables de tipus `struct` el contingut d'aquestes variables es duplica en lloc de fer referència al mateix objecte. El resultat és que podem recuperar la funció i simplicitat de les estructures de C++, i fins i tot els podem afegir codi per a manipular-ne el contigut. A diferència de les classes, no serà possible heretar d'altres estructures, però sí d'interfícies.

De la mateixa manera, també s'han recuperat els tipus enemurats de C++ que s'havien descartat a Java. Això és molt útil per a definir conjunts de constants si no ens preocupa el seu valor concret.

Per últim, cal esmentar l'existència dels espais de noms sota els quals es poden agrupar definicions de classes, estructures i enumerats, tal com es feia a C++ o amb els paquets de Java. La paraula reservada `using` ens serveix per a estalviarnos el prefix dels *namespaces* en els elements de les biblioteques que volem utilitzar.

```
using System;
using Mio;

public class ClassesAltres {

  static void Main() {
    // Enums
    FRUITES f = FRUITES.Pera;
    Console.WriteLine("f="+f);
    Array fruites = Enum.GetValues(typeof(FRUITES));
    foreach (FRUITES fruita in fruites)
      Console.WriteLine("FRUITES({1})={0}", fruita,
fruita.ToString("d"));

    // Structs
    Strct s1 = new Strct(1, 2);
    Strct s2 = s1;
    Suyo.Objct o1 = new Suyo.Objct(1, 2);
    Suyo.Objct o2 = o1;

    Console.WriteLine("s1{0} s2{1}", s1, s2);
    s2.x = 5;
    Console.WriteLine("s1{0} s2{1}", s1, s2);

    Console.WriteLine("o1{0} o2{1}", o1, o2);
    o2.x = 5;
    Console.WriteLine("o1{0} o2{1}", o1, o2);
  }
}

public enum FRUITES: byte {Poma, Pera, Platan}

namespace Mio {
  struct Strct {
    public int x;
```

```
    public int y;
    public Strct(int x, int y) {
      this.x = x;
      this.y = y;
    }
    public override string ToString(){
      return "("+x+", "+y+")";
    }
  }
}
namespace El seu {
  class Objct {
    public int x;
    public int y;
    public Objct(int x, int y) {
      this.x = x;
      this.y = y;
    }
    public override string ToString(){
      return "("+x+", "+y+")";
    }
  }
}
```

2.5. Boo

Boo és un llenguatge de guions amb una sintaxi semblant a la de Python però integrat en el Common Language Interface.

Des dels arxius de codi en Boo, es poden compilar en una biblioteca o executable de Mono o es poden executar directament aquests arxius amb l'intèrpret del llenguatge. El principal avantatge és tenir un sistema de *scripting* enllaçat directament amb les biblioteques existents.

Web recomanat

Hi ha molta documentació i exemples sobre el llenguatge Boo en la pàgina web del projecte: http://boo.codehaus.org.

2.5.1. Estructura bàsica de Boo

Aquest llenguatge ens permet fer guions per a administrar sistemes o per a petits programes d'una manera molt senzilla, sense haver de definir variables i funcions principals, amb una gran versalitat i amb tota la potència de les biblioteques GTK, Ldap, Remoting, etc. de Mono.

Hi ha tres executables:

- booc: compilador a CIL del llenguatge per poder vincular posteriorment des d'un altre programa en C#.
- booi: intèrpret d'arxius Boo.
- booish: intèrpret interactiu de Boo.Comparant C# i Boo.

Una pràctica divertida és compilar un programa que faci el mateix (un *Hola món*) en Boo i en C# i comprovar amb monodis que el resultat obtingut a CIL és gairebé el mateix. Això ens demostra que la velocitat d'un programa en C# i en Boo no variarà gaire si aquest darrer es compila.

2.5.2. Diferències amb C#

Boo és un llenguatge amb les mateixes capacitats i forma de programació que C#. Gairebé tot el que podem fer amb C# ho podem programar en Boo amb les característiques següents:

- Boo no té definició de tipus de variables i crea una matriu si és necessari.

```
Per a escriure:
  Class c = new Class (parametres);
  Type[] l = new Type[] { expr_1, expr_1,..., expr_n }
es pot codificar com:
  c = Class(parametres)
  l = (expr_1, expr_1,..., expr_n)
```

- No s'usa el punt i coma.

- S'usen els condicionals al final de l'acció:

```
Per a escriure:
   if (cond) resposta = 42;
 es pot codificar com:
   if cond:
      resposta = 42
 o es pot usar:
   resposta = 42 if cond
```

- S'usa `except` en lloc de `catch` en les excepcions:

```
Per a escriure:
   try { foo (); bar (); }
   catch (Exception e) { baz (); }
   finally { qux (); }
 es pot codificar com:
   try:
     foo()
     bar()
   except e:
     baz()
   ensure:
     qux()
```

- S'usa `raise` en lloc de `throw` en llançar excepcions.

- Per a fer una conversió d'una variable:

```
Es pot codificar com:
   t = cast(tipo, expr)
 o es pot sar:
   t = expr as tipo
```

- Per poder definir un mètode s'usa `def` igual que a Python.

```
Es pot codificar com:
   static def funcion(x as int, i as string) as int:
```

- Per al constructor s'usa una funció anomenada `constructor` i per a l'herència s'especifica la classe pare entre parèntesis.

```
Es pot codificar com:
   class Foo (Bar):
     def constructor(x as int):
       super(x)
       ...
```

Hi ha algunes diferències més, com el possible ús de # per a posar comentaris, l'ús de tres cometes (""") per a definir cadenes de caràcters llargues, el fet de poder deshabilitar el control del desbordament de *buffer* i l'opció d'inicialitzar diferents variables en la seva definició.

El més important és entendre que és un llenguatge molt fàcil d'usar, sense necessitat de precompilar ni definir variables. Boo és molt útil en entorns d'administració de sistemes o en *scripts* de gestió d'aplicacions.

2.5.3. Exemples

L'estructura d'un arxiu Boo és la següent:

- documentació del mòdul,
- declaració de l'espai de noms que s'han d'usar,
- importació dels mòduls que s'han d'usar,
- membres del mòdul: classes/enum/funcions,
- codi principal d'execució de l'*script*,
- propietats de l'*assembly*.

"Hola món" en Boo

Per a veure'n l'estructura, aquí hi ha un exemple comentat d'un programa fet en Boo que escrigui "Hola món" en consola, usant Windows Forms i amb GTK:

- Usant consola:

```
print("Hola món!")
```

- Usant Windows Forms:

```
import System.Windows.Forms

f = Form(Text: "Hola món!")
f.Controls.Add(Button(Text: "Polseu-me!", Dock: DockStyle.Fill))

Application.Run(f)
```

- Usant GTK:

```
import GTK

Application.Init()

window = Window("Hola Món!",
             DefaultWidth:  200,
             DefaultHeight: 150)

# L'aplicació s'ha de tancar quan la finestra es tanqui
window.DeleteEvent += def():
  Application.Quit ()

window.Add(Button("Polseu-me!"))
window.ShowAll()
Application.Run()
```

2.6. Nemerle

Nemerle es va crear a la Universitat de Wroclaw (Polònia) com a llenguatge amb definició de tipus en temps de compilació que ofereix la possibilitat de treballar amb els paradigmes de programació d'orientació a objecte, programació funcional i programació imperativa.

Web recomanat

http://nemerle.org

L'entorn d'execució de Nemerle és el mateix CLR, tant la implementació Mono com la .NET. Però en la majoria dels casos el compilador s'ha d'obtenir per separat. Es tracta de l'executable ncc, que s'usarà per a compilar els arxius de codi font en Nemerle, normalment amb extensió '.n'.

En el subapartat següent es repassaran diversos aspectes bàsics del llenguatge, i posteriorment es tractaran determinades característiques que requereixen una atenció especial.

2.6.1. Diferències bàsiques amb C#

Per a introduir-nos ràpidament en el llenguatge Nemerle, primer repassarem les diferències bàsiques respecte del llenguatge C# exposat anteriorment:

- En declarar les variables i les funcions, el tipus de dades s'escriu al final de la declaració:

```
funcio (x : int, y : string) : int
```

- La inferència de tipus permet no explicitar-los, sempre que quedin definits sense ambigüitat implícitament:

```
def x = 3
funcio (y) { y++ }
System.Console.WriteLine ("resultat"+funcion(x));
```

- Es poden fer definicions de variables que no podran canviar el seu contingut una vegada inicialitzades (paraula reservada `def`), o bé treballar amb variables convencionals usant la paraula reservada `mutable`:

```
def x = 3;
mutable x = 3;
```

- No hi ha la paraula reservada `new`: `def var = Objecte(3);`

- No hi ha la paraula reservada `return`, el resultat de l'última expressió d'un mètode és el valor retornat. Això implica que no es pot interrompre el flux del codi d'un mètode, ja que no s'hi pot introduir cap `return` en mig.

- A part de la construcció condicional `if` habitual, també incorpora altres notacions a l'estil de Perl: `when` i `unless`.

– `when (x < 3) { ... }`: és com un `if` però no pot tenir clàusula `else`.

– `unless (x < 3) { ... }`: és com la clàusula `else` d'un `if` però sense clàusula `then`.

- Un `module` és una classe en què tots els membres són estàtics.

- El constructor d'una classe o mòdul sempre s'anomena amb la paraula reservada `this`, i la crida al constructor de la classe mare es fa explícitament en el cos amb la paraula reservada `base`:

```
class Clase {
  public this (x : int)
  { ... base(x); ... }
}
```

- Es pot alterar l'ordre dels paràmetres si se n'indica el nom en fer la crida al mètode:

```
funcio (etat : int, nom : string, cognoms : string) : int { ... }

Main () : void
{
  def res1 = funcio( 20, "Ramón", "Martínez Soria");
  def res2 = funcio( cognoms = "De la Fuente", edat = 25, nom =
"Marta");
}
```

2.6.2. Ús de funcions locals

Després d'haver vist les diferències bàsiques respecte de C#, en aquest subapartat ens adentrarem en les característiques pròpies de Nemerle que permeten canviar l'estil de programació:

- Es poden declarar funcions locals, l'àmbit de validesa de les quals és el bloc en què estan definides. D'aquesta manera, es poden usar fàcilment construccions recursives per a implementar els bucles en lloc de les instruccions iteratives explícites:

```
module Factorial {
   public Main (): void
   {
     def fact (x: int): int {
       if (x <= 0)
         1
       else
         x * fact(x - 1)
     }
     System.Console.WriteLine ("Factorial(20)="+ fact(20))
   }
}
```

```
Exemple complet sobre la manipulació de taules i l'ús de funcions
locals:

class ArraysTest {
   static reverse_array (ar : array [int]) : void
   {
     def loop (left, right) {
       when (left < right) {
         def tmp = ar[left];
         ar[left] = ar[right];
         ar[right] = tmp;
         loop (left + 1, right - 1)
       }
     }
     loop (0, ar.Length - 1)
   }

   static print_array (ar : array [int]) : void
   {
     for (mutable i = 0; i < ar.Length; ++i)
       Nemerle.IO.printf ("%d\n", ar[i])
   }

   static Main () : void
   {
     def ar = array [1, 42, 3];
     print_array (ar);
     Nemerle.IO.printf ("\n");
     reverse_array (ar);
     print_array (ar);
   }
 }
```

2.6.3. Macros i disseny per contracte

Una de les característiques més interessants de Nemerle és l'existència de macros que permeten estendre el mateix llenguatge. Mitjançant aquestes macros és possible treballar basant-se en la tècnica de disseny per contracte, en què es poden indicar, abans d'iniciar la implementació, quines interfícies es volen i quin comportament han de tenir.

Podem definir macros per a intervenir en el procés de compilació, ja que es tracta de funcions que executarà el compilador. A diferència de les macros habituals de llenguatges com C, a Nemerle les macros no són executades per un precompilador que substitueix el codi que compilarà el compilador, sinó pel mateix compilador. D'aquesta manera, és possible manipular directament les estructures que usa el mateix compilador a mesura que analitza el codi, d'aquesta manera aconseguir estendre el llenguatge:

```
macro miMacro () {
  Nemerle.IO.printf ("compile-time\n");
  <[ Nemerle.IO.printf ("run-time\n") ]>;
}
```

```
ncc -r Nemerle.Compiler.dll -t:dll mimacro.n -o mimacro.dll
```

```
module Modul {
  public Main () : void {
    miMacro ();
  }
}
```

```
ncc -r mimacro.dll miprog.n -o miprog.exe
```

És possible continuar el paradigma de disseny per contracte usant les macros existents per a indicar les regles que ha de respectar cada mètode:

- `requires`: per a indicar els prerequisits que s'han de complir en arribar al mètode.

```
class Cadena
{
  public Subcadena (idxInici : int) : string
  requires idxInici >= 0 && idxInici <= this.Length
  { ... }
}

* ensures: per a indicar els postrequisits que s'han de complir en
abandonar el mètode.

class Llista
{
  public Vaciar() : void
  ensures this.IsEmpty
  { ... }
}
```

- `invariant`: per a indicar les invariants que s'han de complir durant tota l'execució del mètode o existència d'un objecte.

```
class Vector [T]
invariant posicio >= 0 && posicio <= arr.Length
{
  private mutable posicio : int = 0;
  private taula : array [T] = array [];
  ...
}
```

2.6.4. Tipus `variant` i patrons

La manipulació de tipus de dades sobre conjunts de valors enumerats es pot fer amb els tipus de dades `variant` i la construcció `match`:

```
variant Color {
   | Vermell
   | Groc
   | Verd
   | Diferent {
       vermell : float;
       verd: float;
       blau : float;
     }
 }
obtenerNombColor (color : Color) : string
{
  match (color) {
    | Color.Vermell => "vermell"
    | Color.Groc => "groc"
    | Color.Verd => "verd"
    | Color.Diferent (r, g, b) =>
      System.String.Format ("rgb({0},{1},{2})", r, g, b)
    | _ => "error"
  }
}
```

Aquesta combinació es pot usar per a tractar referències genèriques a objectes en programació de GUI:

```
using System.Windows.Forms;

...

match (control) {
  | button is Button => ...
  | listv is ListView => ...
  | _ => ... // null case
}
```

2.6.5. Exemple d'aplicació amb GUI

En aquest exemple final podem observar diverses de les característiques esmentades anteriorment usades per a crear un editor simple capaç de desar i recuperar arxius de text.

```
using System;
using Gtk;

class NMenuItem : MenuItem
{
        public name : string;
```

```
        public this(l : string)
        {
                base(l);
                name = l;
        }

        public this(l : string, e : object * EventArgs -> void)
        {
                base(l);
                name = l;
                this.Activated += e;
        }


        // This property allows us to set submenus of this menu item
        public SubmenuList : list [NMenuItem]
        {
                set
                {
                        def sub = Menu();
                        foreach (submenu in value) sub.Append (submenu);
                        this.Submenu = sub;
                }
        }
}


class MainWindow : Window
{
        /// Text input area of our window
        input : TextView;

        public this()
        {
                // Set caption of window
                base ("Very Simple Editor");
                // Resize windows to some reasonable shape
                SetSizeRequest (300,200);

                def scroll = ScrolledWindow ();
                input = TextView ();
                scroll.Add (input);

                def menu = MenuBar ();
                def mi = NMenuItem ("File");
                mi.SubmenuList =
                [
                        NMenuItem ("Open", OnMenuFile),
                        NMenuItem ("Save as...", OnMenuFile)
                ];
                menu.Append(mi);

                def vbox = VBox ();
                vbox.PackStart (menu, false, false, 0u);
                vbox.PackStart (scroll, true, true, 0u);

                // Place vertical box inside our main window
                Add (vbox);
        }

        // Handler of opening and saving files
        OnMenuFile (i : object, _ : EventArgs) : void
        {
                def mi = i :> NMenuItem;
                def fs = FileSelection (mi.name);

                when (fs.Run () == ResponseType.Ok :> int) match (mi.name)
                {
```

```
                        | "Open" =>
                                def stream = IO.StreamReader (fs.Filename);
                                input.Buffer.Text = stream.ReadToEnd();

                        | "Save as..." =>
                                def s = IO.StreamWriter(fs.Filename);
                                s.Write(input.Buffer.Text);
                                s.Close();
                        | _ => ();
                };
                fs.Hide();
        }
}

module SimpleEditor
{
        Main() : void
        {
                Application.Init();
                def win = MainWindow();

                // Exit application when editor window is deleted
                win.DeleteEvent += fun (_) { Application.Quit () };

                win.ShowAll ();
                Application.Run();
        }
}
```

3. GTK# i Gdk#

Ja hem tractat la biblioteca GTK en el subapartat 1.2.

Per a poder implementar aplicacions gràfiques en Mono hi ha dues opcions majoritàries: usar Windows Forms o usar GTK. Utilitzar la primera biblioteca està subjecte a patents i al control per part d'una companyia. La segona és una biblioteca lliure derivada de l'aplicació GIMP i usada en el món de l'escriptori per l'entorn Gnome. GTK ha millorat molt durant els darrers anys fins a arribar a ser una de les millors en el maneig de *widgets* per a escriptori. Des de la plataforma Mono tenim accés a aquestes biblioteques per crear les nostres aplicacions gràfiques mitjançant gtk-sharp. També hi ha gtk-dotnet.dll, que fa de pont d'unió entre els *widgets* de GTK i les funcions de dibuix de Mono (`System.Drawing`).

3.1. Instal·lació

Per a poder-se instal·lar l'última versió, el millor és obtenir el codi font de la pàgina oficial de Mono. Una vegada el tinguem, podem descomprimir-lo i compilar-lo:

```
 tar -xvzf gtk-sharp-X.X.X.tar.gz
cd  gtk-sharp-X.X.X
./configure
make
make install
```

Si en el moment de configurar-lo ens indica que no compilarà determinats mòduls, haurem d'anar al sistema de paquets del sistema operatiu i buscar els que continguin la informació de desenvolupament de GTK, Glade, rsvg, vte, Gtkhtml, Gnomevfs i Gnome. Quan el sistema detecti que estàn instal·lats, en reconfigurar els podrem recompilar. Una vegada hem instal·lat la biblioteca GTK, podem començar a compilar programes usant gtk-sharp.

Per a poder compilar un programa usant les biblioteques de gtk-sharp, hem d'incloure en la línia d'ordres l'opció `-pkg:gtk-sharp`.

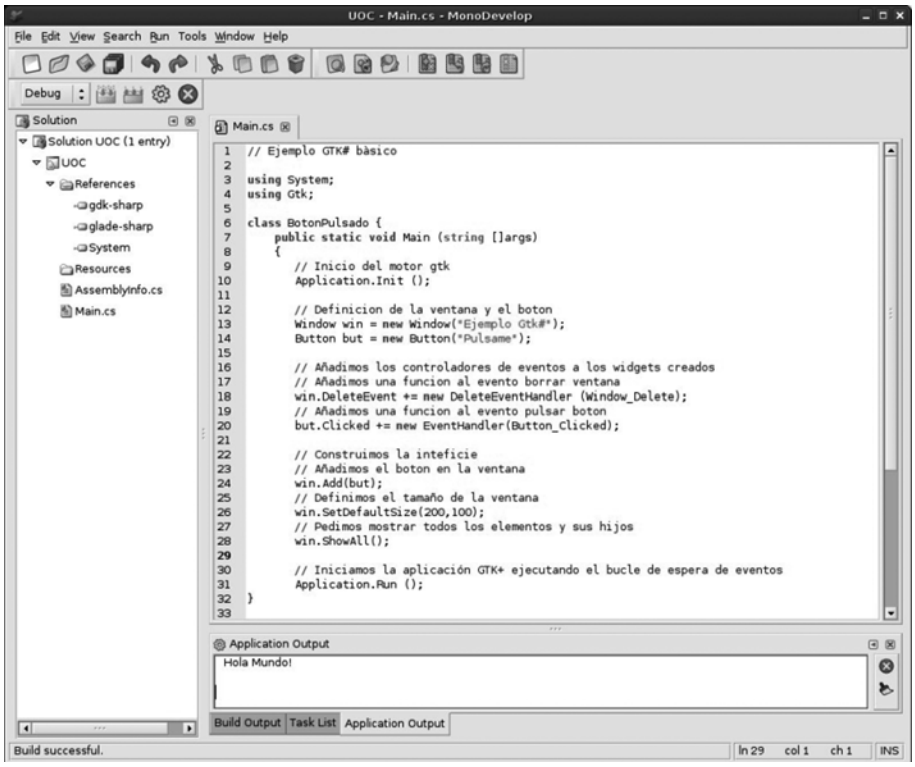
3.1.1. Exemple GTK#

Amb aquest exemple construirem un programa que ens mostrarà un botó. Haurem d'utilitzar gtk-sharp per a dibuixar la finestra, dibuixar el botó i utilitzar un capturador d'esdeveniments per poder saber quan l'hem de prémer.

```
// Exemple GTK# bàsic

using System;
using Gtk;

class BotoPolsat {
public static void Main (string []args)
{
   // Inici del motor GTK
   Application.Init ();

   // Definició de la finestra i el botó
   Window win = new Window("Exemple GTK#");
   Button but = new Button("Polseu-me");

   // Afegim els controladors d'esdeveniments als widgets creats
   // Afegim una funció a l'esdeveniment esborrar finestra
   win.DeleteEvent += new DeleteEventHandler (Window_Delete);
   // Afegim una funció a l'esdeveniment prémer botó
   but.Clicked += new EventHandler(Button_Clicked);

   // Construïm la interfície
   // Afegim el botó a la finestra
   win.Add(but);
   // Definim la mida de la finestra
   win.SetDefaultSize(200,100);
   // Demanem mostrar tots els elements y els seus fills
   win.ShowAll();

   // Iniciem l'aplicació GTK+ executant el bucle d'espera
d'esdeveniments
   Application.Run ();
}
static void Window_Delete ( object o, DeleteEventArgs args)
{
   // Sortim en prémer el botó de tancar finestra
   Application.Quit ();
   args.RetVal = true;
}

static void Button_Clicked ( object o, EventArgs args)
{
   // Escrivim en la consola
   System.Console.WriteLine("Hola Món!");
}
} // Fi classe
```

Per a provar l'exemple, podem usar Monodevelop creant un projecte GTK# 2.0 (figura 1). En aquest projecte, haurem de comprovar que les referències a `GTK` i `system` estan ben posades.

Figura 1. Captura de pantalla de Monodevelop amb l'exemple i les referències



3.2. Widgets GTK#

En el subapartat anterior hem vist un exemple d'alguns dels *widgets* de GTK+ en ús: `Window` i `Button`. Abans de continuar amb aquests elements visuals, hem d'introduir l'ús de les caixes (*boxes*) per al disseny de la interfície. Ja hem parlat d'aquests elements en el subapartat de GTK+, però ara en veurem un exemple i què podem fer amb aquestes caixes.

Aquí tenim un arbre en què podem observar la relació, des del punt de vista dels objectes, dels diferents *widgets*:

```
Gtk.Object
  Gtk.Widget
    Gtk.Misc
      Gtk.Label
        Gtk.AccelLabel
      Gtk.Arrow
      Gtk.Image
    Gtk.Container
      Gtk.Bin
        Gtk.Alignment
        Gtk.Frame
        Gtk.AspectFrame
        Gtk.Button
          Gtk.ToggleButton
            Gtk.CheckButton
              Gtk.RadioButton
          Gtk.OptionMenu
```

```
        Gtk.Item
          Gtk.MenuItem
            Gtk.CheckMenuItem
              Gtk.RadioMenuItem
            Gtk.ImageMenuItem
            Gtk.SeparatorMenuItem
            Gtk.TearoffMenuItem
        Gtk.Window
          Gtk.Dialog
            Gtk.ColorSelectionDialog
            Gtk.FileSelection
            Gtk.FontSelectionDialog
            Gtk.InputDialog
            Gtk.MessageDialog
          Gtk.Plug
        Gtk.EventBox
        Gtk.HandleBox
```

```
    Gtk.ScrolledWindow
    Gtk.Viewport
  Gtk.Box
    Gtk.ButtonBox
      Gtk.HButtonBox
      Gtk.VButtonBox
    Gtk.VBox
      Gtk.ColorSelection
      Gtk.FontSelection
      Gtk.GammaCurve
    Gtk.HBox
      Gtk.Combo
      Gtk.Statusbar
Gtk.Fixed
Gtk.Paned
    Gtk.HPaned
    Gtk.VPaned
  Gtk.Layout
  Gtk.MenuShell
    Gtk.MenuBar
    Gtk.Menu
  Gtk.Notebook
  Gtk.Socket
  Gtk.Table
  Gtk.TextView
  Gtk.Toolbar
  Gtk.TreeView
Gtk.Calendar
Gtk.DrawingArea
```

```
    Gtk.Curve
  Gtk.Editable
    Gtk.Entry
      Gtk.SpinButton
  Gtk.Ruler
    Gtk.HRuler
    Gtk.VRuler
  Gtk.Range
    Gtk.Scale
      Gtk.HScale
      Gtk.VScale
    Gtk.Scrollbar
      Gtk.HScrollbar
      Gtk.VScrollbar
  Gtk.Separator
    Gtk.HSeparator
    Gtk.VSeparator
  Gtk.Invisible
  Gtk.Preview
  Gtk.ProgressBar
Gtk.Adjustment
Gtk.CellRenderer
  Gtk.CellRendererPixbuf
  Gtk.CellRendererText
  Gtk.CellRendererToggle
Gtk.ItemFactory
Gtk.Tooltips
Gtk.TreeViewColumn
```

3.2.1. Elements contenidors

Per a poder dissenyar el conjunt d'elements gràfics visibles, haurem d'utilitzar els diferents contenidors que hi ha disponibles a GTK#. Aquests contenidors són semblants als de la biblioteca en C.

Caixes

Aquests elements ens permeten definir l'estructura en què dividirem les diferents finestres, que ens permetrà escalar fàcilment i internacionalitzar l'aplicació tenint en compte la possibilitat de diferents quantitats de text en els *labels* i altres *widgets*.

Hi ha dos tipus de caixes: `Hbox` i `Vbox`. Representen la divisió horitzontal i vertical respectivament. Aquest sistema ens permet anar dividint en tantes seccions com vulguem fins a obtenir les caixes necessàries. Per a poder utilitzar aquests elements, hem de cridar una funció que dinàmicament adaptarà les diferents caixes a la quantitat de text i a la mida de la finestra. Les dues funcions són `PackStart` i `PackEnd`. En el cas d'una `Vbox`, `PackEnd` començarà a empaquetar per dalt i `PackEnd` començarà per sota. En el cas d'una `Hbox`, `PackStart` empaquetarà d'esquerra a dreta i `PackEnd` de dreta a esquerra. Aquestes funcions redimensionaran l'`Hbox` i `Vbox` per poder visualitzar cor-

rectament la finestra. Aquestes dues funcions, igual que `Vbox` i `Hbox`, tenen algunes opcions interessants que cal comentar:

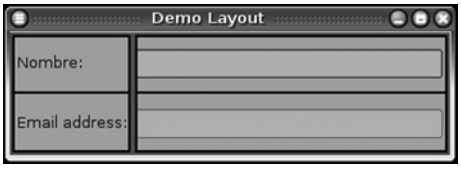
- `Vbox` i `Hbox`:
  - `Spacing`: nombre de píxels que hi ha entre les diferents caixes.
  - `BorderWidth`: nombre de píxels que hi ha entre la vora de la caixa i el contingut.
- `PackStart` i `PackEnd`:
  - `Expand`: s'expandeix per a usar tota la mida possible.
  - `Fill`: fa que el *widget* ocupi tota la mida obtinguda per `expand`; en cas contrari, l'espai extra serà per al `padding`.
  - `Padding`: afegeix espai al voltant del *widget*.

Tots els elements gràfics que es poden alinear de diferents maneres implementen la interfície `Gtk.Misc`. Aquesta inferfície té les propietats `Xalign` i `Yalign`, que prenen valors de `0` (esquerra i a dalt) a `1` (dreta i a sota). L'ús de la funció `add` en una caixa usa internament `PackStart` amb els valors per defecte: `true`, `false` i `0`.

```
Exemple de Packaging

namespace HVoxTutorial {

   using Gtk;
   using System;

   public class MainClass
   {
     public static void Main (string[] args)
     {
       Application.Init();
       SetUpGui();
       Application.Run();
     }

     static void SetUpGui()
     {
       // Creem una finestra
       Window w = new Window("Demo Layout");

       // Creem primer una caixa horitzontal i l'afegim a la
finestra
       HBox h = new HBox();
       h.BorderWidth = 6;
       h.Spacing = 6;
       w.Add(h);


       // Creem una caixa vertical
       VBox v = new VBox();
       v.Spacing = 6;

       // Afegim la VBox a la HBox sense expand ni fill
       h.PackStart(v,false,false,0);


       // Creem una etiqueta i l'alineem a l'esquerra
       Label l = new Label ("Nom:");
       l.Xalign=0;
```

```
        // Afegim el label a la VBox amb expand.
        v.PackStart(l,true,false,0);

        // Creem una etiqueta i l'alineem a l'esquerra
        l = new Label ("Email address:");
        l.Xalign = 0;
        v.PackStart(l,true,false,0);

        // Creem una nova caixa vertical per als entry
        v = new VBox();
        v.Spacing = 6;
        h.PackStart(v,true,true,0);

        v.PackStart(new Entry(), true, true, 0);
        v.PackStart(new Entry(), true, true, 0);

        w.DeleteEvent += Window_Delete;
        w.ShowAll();
      }

      static void Window_Delete (object o, DeleteEventArgs args)
      {
        Application.Quit();
        args.RetVal = true;
      }
    }
}
```

Figura 2



Taules

Per a crear algunes interfícies gràfiques fa falta una estructura quadriculada per a representar els elements gràfics. Per a aquest cas hi ha un contenidor anomenat "taula". A l'hora de crear-la, s'hi poden definir els elements següents:

- `Rows` i `columns`: files i columnes que tindrà la taula.

- `Homogenous`: booleà que ens defineix si tots els elements de la taula tindran la mateixa mida d'alçada i amplada o si s'adaptaran al màxim de la fila i/o columna corresponent.

Per a poder afegir un element gràfic en una casella de la taula, haurem d'usar les funcions `attach`: amb els paràmetres següents:

- `Widget child`: element gràfic que usarà per a posar a la cel·la escollida.

- `leftAttach` i `rightAttach`: columnes en què es col·locarà el *widget*. Amb aquests paràmetres podem ocupar més d'una cel·la amb un element.

- `topAttach` i `bottomAttach`: files en què es col·locarà el *widget*. Amb aquests paràmetres podem ocupar més d'una cel·la amb un element.

- `xOptions`, `yOptions`:

– `Gtk.AttachOptions.Fill`: si la cel·la de la taula és més llarga que el *widget* i `Fill` està seleccionat, el *widget* s'expandirà per a usar tot l'espai disponible.

– `Gtk.AttachOptions.Shrink`: si el *widget* té menys espai que el que requereix (normalment perquè l'usuari ha redimensionat la finestra), els *widgets* normalment desapareixen de la finestra. Si s'especifica `Shrink`, els *widgets* s'encongeixen amb la taula.

– `Gtk.AttachOptions.Expand`: aquesta opció expandirà la taula fins a ocupar tot l'espai disponible a la finestra.

- `xPadding`, `yPadding`: *Padding* en les dues dimensions.

En un cas en què les `xOptions`, `yOptions` i `xPadding`, `yPadding` prenen els valors per defecte podem usar la funció `attachdefaults`, en què no fa falta esmentar-los.

Per a poder definir l'espai que hi ha entre les diferents cel·les de la taula hi ha les opcions següents:

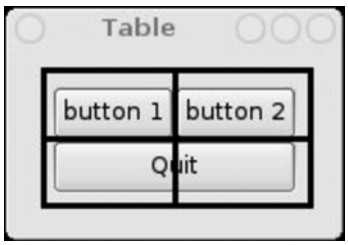
- `SetRowSpacing(int row, int spacing)` i `SetColSpacing(int column, int spacing)`: en què definim la columna o la fila i l'espaiat que volem usar.

- `SetRowSpacing(int spacing)` i `SetColSpacings(int spacing)`: definim l'espaiat per a totes les columnes i files.

```
namespace TaulesTutorial {

    using Gtk;
    using System;
    using System.Drawing;

    public class table
    {

      static void delete_event (object obj, DeleteEventArgs args)
      {
           Application.Quit();
      }

      static void exit_event (object obj, EventArgs args)
      {
           Application.Quit();
```

```
        }

        public static void Main(string[] args)
        {
             Application.Init ();
             Window window = new Window ("Table");
             window.DeleteEvent += delete_event;
             window.BorderWidth= 20;

             /* Creem una taula de 2 per 2 */
             Table table = new Table (2, 2, true);

             /* Afegim la taula a la finestra */
             window.Add(table);

             /* Creem un botó */
             Button button = new Button("button 1");

             /* Posem el botó en la posició de dalt a l'esquerra */
             table.Attach(button, 0, 1, 0, 1);
             button.Show();

             /* Creem un altre botó */
             Button button2 = new Button("button 2");

             /* Posem el botó en la posicio de dalt a la dreta */
             table.Attach(button2, 1, 2, 0, 1);
             button2.Show();

             /* Creem el botó per a sortir */
             Button quitbutton = new Button("Quit");

             /* Afegim l'esdeveniment per sortir al botó */
             quitbutton.Clicked += exit_event;

             /* Posem el botó ocupant els dos espais inferiors */
             table.Attach(quitbutton, 0, 2, 1, 2);
             quitbutton.Show();

             table.Show();
             window.ShowAll();

             Application.Run();
        }
     }
}
```

Figura 3



Frame

Eina que ens permet tenir un contenidor amb un marc al voltant. Dins hi podem posar qualsevol altre contenidor o *widget*. Podem posar un títol en el marc i alinear-lo on vulguem.

Constructor:

```
Frame frame1 = new Frame("Label");
```

Podem definir diferents tipus de marcs amb la propietat `ShadowType`: `Gtk.Shadow.None`, `Gtk.Shadow.In`, `Gtk.Shadow.Out`, `Gtk.Shadow.EtchedIn` (per defecte) o `Gtk.Shadow.EtchedOut`.

```
namespace Frame {
        using Gtk;
        using System;
        using System.Drawing;


        public class frame
        {
               static void delete_event (object obj, DeleteEventArgs
args)
                {
                Application.Quit();
                }

                public static void Main( string[] args)
                {

                        /* Inicialitzar GTK */
                        Application.Init();

                        /* Creem una finestra nova */
                       Window window = new Window ("Exemple Frame");

                        /* Esdeveniment per a tancar la finestra */
                        window.DeleteEvent += delete_event;

                        window.SetSizeRequest(300, 300);
                        window.BorderWidth= 10;

                        /* Creem un frame */
                        Frame frame = new Frame("Frame");
                        window.Add(frame);

                        /* Afegim un títol */
                        frame.Label = "El nostre frame";

                        /* Decidim el tipus de marc */
                        frame.ShadowType = (ShadowType) 4;
                        frame.Show();

                        window.ShowAll();
                        Application.Run();

                }
        }
}
```

Contingut fix

En aquest contenidor podrem posar els *widgets* on vulguem d'una manera fixa. El gran avantatge és la facilitat de veure com quedarà. El desavantatge és que en créixer no es redimensiona.

Les funcions usades són les següents:

```
Fixed fixed1 = new Fixed(); - Constructor
fixed1.Put(widget, x, y); - Per a poder posar un element gràfic en
una posició.
fixed1.Move(widget, x, y); - Per a moure un element gràfic d'una
posició a una altra.
```

Layout

Aquest contenidor és igual que l'anterior, amb la diferència que té una barra desplaçadora que ens permet disposar d'un espai gairebé infinit on es poden col·locar els *widgets*. Aquest contenidor ens permet eludir el límit de 32.767 píxels del que hem presentat anteriorment. Les funcions són les mateixes, llevat que en el constructor definim l'alineació i apareix una propietat per a definir la mida del *layout*.

```
layout1.Size = new Size (width, height)
```

3.2.2. Elements gràfics

Botons

Podem distingir els tipos de botons següents:

1) Botons normals

Són els botons en què tenim un polsador per a activar un esdeveniment. Aquest tipus de *widgets* es poden crear de dues maneres:

```
Button button1 = new Button("text")
Button button2 = new ButtonWithMnemonic()
```

El fet de tenir un botó sense cap text fa que s'obtingui l'esmentat botó amb una caixa contenidora en què podem afegir altres tipus de *widgets*, com imatges i *labels*.

```
namespace Botons1 {

        using Gtk;
        using System;

        public class boton
       {
```

```
 /* Funció del senyal de polsar el botó */
                static void callback( object obj, EventArgs args)
                {
                       Console.WriteLine("Hola, el botó ha estat polsat");
                }

                /* another callback */
                static void delete_event (object obj, DeleteEventArgs args)
                {
                        Application.Quit();
                }

                public static void Main(string[] args)
                {
                        Application.Init();

                        /* Creem una finestra */
                        Window window = new Window ("Botons");
                        window.DeleteEvent += delete_event;
                        window.BorderWidth = 10;

                        /* Creem un botó */
                        Button button = new Button();

                        /* Afegim un senyal de callback */
                        button.Clicked += callback;

                        /* Crem una caixa per a l'etiqueta */
                        HBox box = new HBox(false, 0);
                        box.BorderWidth =  2;

                        /* Creem una etiqueta per al botó */
                        Label label = new Label ("Polsar!");
                        box.PackStart(label, false, false, 3);
                        label.Show();

                        /* Mostra la caixa */
                        box.Show();

                        /* Afegeix la caixa al botó */
                        button.Add(box);

                        /* Mostra el botó */
                        button.Show();

                        window.Add(button);
                        window.ShowAll();
                        Application.Run();
                }
        }
}
```

2) Botons d'estat

La diferència entre aquest botó i l'anterior és que guarda l'estat en una propietat anomenada `Active`. Si premem el botó, queda premut i la propietat obté el valor `true`. En cas contrari, passa a `false`.

```
using Gtk;
using System;

public class Botons2
{
```

```
     public static void Main(string[] args)
     {
          Application.Init();
          Window window = new Window("Bontons");
          window.DeleteEvent += delete_event;

          /* Creem un botó d'estat */
          ToggleButton togglebutton = new ToggleButton("botó");
          togglebutton.Clicked += clickedCallback;

          window.Add(togglebutton);
          window.ShowAll();

          Application.Run();
      }

      static void delete_event (object obj, DeleteEventArgs args)
      {
          Application.Quit();
      }

      static void clickedCallback (object obj, EventArgs args)
      {
          /* Per a saber si està actiu */
          if (((ToggleButton) obj).Active)
               Console.WriteLine ("Estic actiu");
      }
}
```

3) Botons de reconeixement

Són el mateix que els botons anteriors, amb la diferència que es renderitzen amb un petit requadre amb text al costat. Aquest requadre queda marcat quan s'activa. Les funcions de creació són les següents:

```
GtkWidget CheckButton1 = new CheckButton();
GtkWidget CheckButton1 = new CheckButtonWithLabel(string label);
GtkWidget CheckButton1 = new CheckButtonWithMnemonic(string label);
```

4) Botons de ràdio

Aquests elements serveixen per a tenir un conjunt de reconeixement en què els elements que el componen són autoexclusius. Quan escolliu un element d'aquests, es desactiven els altres que hi ha al mateix grup. Per a crear un botó en un grup, modifiqueu la propietat de `Group`.

```
namespace Botons {

   using Gtk;
   using System;

   public class radiobuttons
   {
```

```
      static void delete_event (object obj, DeleteEventArgs args)
      {
           Application.Quit();
      }

      public static void Main(string[] args)
      {
          Application.Init();

          /* Creem la finestra amb els botons */
          Window window = new Window("Botons");
          window.DeleteEvent += delete_event;
          window.BorderWidth = 0;

          VBox box1 = new VBox (false, 0);
          window.Add(box1);
          box1.Show();

          /* Creem un botó de ràdio */
          RadioButton radiobutton = new RadioButton  (null, "button1");
          box1.PackStart(radiobutton, true, true, 0);
          radiobutton.Show();

          /* Creem un segon botó de ràdio i el posem en el mateix grup */
          RadioButton radiobutton2 = new RadioButton(null, "button2");
          radiobutton2.Group=radiobutton.Group;
          /* Activem aquest botó de ràdio */
          radiobutton2.Active = true;
          box1.PackStart(radiobutton2, true, true, 0);
          radiobutton2.Show();

          window.ShowAll();
          Application.Run();
       }
    }
}
```

Inputs

Per a poder recollir dades, tenim diferents tipus de *widgets*:

- **`Entry`** – Element d'entrada d'una sola línia. Podem accedir al text amb la propietat `Text`, decidir si és editable o no amb la propietat `iseditable` i copiar i enganxar del portapapers amb la funció `pasteclipboard`, `copyclipboard` i `cutclipboard`.

- **`Combobox`**, **`ComboboxEntry`** – Elements d'entrada per a poder escollir un opció d'un desplegable. A `ComboboxEntry` podem escollir, escrivint el valor que volem escollir. Per a poder treballar en aquest darrer, es tracta com una `Entry` normal.

```
using System;
using Gtk;

class ComboBoxSample
{
     static void Main ()
     {
```

```
            new ComboBoxSample ();
        }
        ComboBoxSample ()
        {
            Application.Init ();
            /* Creem una finestra */
            Window win = new Window ("ComboBoxSample");
            win.DeleteEvent += new DeleteEventHandler (OnWinDelete);

            /* Crem una combobox */
            ComboBox combo = ComboBox.NewText ();
            /* Afegim 5 elements amb un text */
            for (int i = 0; i < 5; i ++)
                combo.AppendText ("item " + i);
            /* Afegim el controlador de l'esdeveniment */
            combo.Changed += new EventHandler (OnComboBoxChanged);

            win.Add (combo);
            win.ShowAll ();
            Application.Run ();
        }

        void OnComboBoxChanged (object o, EventArgs args)
        {
            /* Recollim el combo escollit */
            ComboBox combo = o as ComboBox;
            if (o == null)
                return;

            TreeIter iter;
            /* Obtenim el combo escollit i mostrem el missatge */
            if (combo.GetActiveIter (out iter))
                Console.WriteLine ((string) combo.Model.GetValue
    (iter, 0));
        }

        void OnWinDelete (object obj, DeleteEventArgs args)
        {
            Application.Quit ();
        }
    }
```

- **`TextView` –** Element per a mostrar un quadre de text amb una barra desplaçadora. En aquest element mostrarem un *textbuffer* que podem obtenir d'un arxiu o que podem crear. Dins de l'element, tenim un atribut `buffer` en què podem usar la propietat de `Text` per a llegir o editar el contingut.

Scrollbars, *Range* i *Scale*

Aquest conjunt de *widgets* serveix per a implementar elements en què hi ha una llista d'opcions per escollir i una barra de desplaçament que permet a l'usuari escollir el valor. El *widget* bàsic és el de `Range`, que associat a un element `Adjustments` pot calcular la mida de l'element de desplaçament i la posició dins de la barra.

Aquests *widgets* tenen una política d'actualització que ens permet definir quan s'ha de canviar el valor associat. L'atribut `UpdateType` pot ser `Continuous`, amb la qual cosa es llança un esdeveniment `ValueChanged` cada vegada que l'usuari mou la barra desplaçadora o l'escalador. Una altra posició és que sigui

`Discontinuous`, amb la qual cosa no es modifica el valor fins que l'usuari ha deixat de polsar el widget. L'última opció és `Delayed`, en què es canvia el valor cada vegada que fa un moment que l'usuari no mou l'element de desplaçament dins de la barra i el manté polsat.

```
namespace Range {
        using Gtk;
        using System;
        using System.Drawing;


        public class ExempleRange
        {
                static HScale hscale;
                static VScale vscale;

                static void scale_set_default_values (Scale s)
                {
                        s.UpdatePolicy = UpdateType.Continuous;
                        s.Digits = 1;
                        s.ValuePos = PositionType.Top;
                        s.DrawValue = true;
                }

                static void create_range_controls ()
                {
                        Window window;
                        VBox box1, box3;
                        HScrollbar scrollbar;
                        Scale scale;
                        Adjustment adj1, adj2;

                        window = new Window ("Range");
                        box1 = new VBox (false, 0);
                        window.Add (box1);
                        box1.ShowAll ();

                        /* Creem un conjunt d'ajustament per al VScale */
                        adj1 = new Adjustment (0.0, 0.0, 101.0, 0.1, 1.0, 1.0);
                        vscale = new VScale ((Adjustment) adj1);

                        /* Posem els valors per defecte */
                        scale_set_default_values (vscale);

                        box1.PackStart (vscale, true, true, 0);
                        vscale.ShowAll ();
                        box3 = new VBox (false, 10);
                        box1.PackStart (box3, true, true, 0);
                        box3.ShowAll ();

                        /* Utilitzem els mateixos ajustos */
                        hscale = new HScale ((Adjustment) adj1);
                        hscale.SetSizeRequest (200, -1);

                        /* Posem els valors per defecte */
                        scale_set_default_values (hscale);

                        box3.PackStart (hscale, true, true, 0);
                        hscale.ShowAll ();

                        /* utilitzem els mateixos ajustos aquest cop per a un scrollbar */
                        scrollbar = new HScrollbar ((Adjustment) adj1);

                        /* Aquí no posem els valors per defecte */
```

```
                    scrollbar.UpdatePolicy = UpdateType.Continuous;

                      box3.PackStart (scrollbar, true, true, 0);
                      scrollbar.ShowAll ();

                      /* Creem ajustaments nous */
                      adj2 = new Adjustment (1.0, 0.0, 5.0, 1.0, 1.0, 0.0);
                      scale = new HScale (adj2);
                      scale.Digits = 0;

                      box1.PackStart (scale, true, true, 0);
                      scale.ShowAll ();
                      box1.ShowAll ();

                      /* Creem ajustaments nous */
                      adj2 = new Adjustment (1.0, 1.0, 101.0, 1.0, 1.0, 0.0);
                      scale = new HScale (adj2);
                      scale.Digits = 0;
                      box1.PackStart (scale, true, true, 0);
                      scale.ShowAll ();
                      window.ShowAll ();
              }

              public static void Main (string [] args)
              {
                      Application.Init ();
                      create_range_controls ();
                      Application.Run ();
              }
      }
}
```

Progress Bars

Widget per a tenir una barra que va incrementant. Amb l'atribut `Fraction` podem triar en quin punt està del procés, un valor de `0` a `1`. Podem escollir si va de dreta a esquerra o a l'inrevés, de dalt a baix o a l'inrevés. També es pot usar per a mostrar que hi ha algun tipus d'activitat amb la funció `Pulse`. L'increment donat per l'acció de polsar un cop es defineix amb la propietat `PulseStep`.

En aquest exemple complet podem veure l'ús de temporitzadors, taules, botons i barres de desplaçament.

```
using GLib;
using Gtk;
using System;

class ProgressBarSample {

    /* Dades que enviem als callbacks */
  public struct ProgressData {
  public Gtk.Window window;
    public Gtk.ProgressBar pbar;
    public uint timer;
    public bool activity_mode;
  }

  static ProgressData pdata;
```

```
  /* Funció que va canviant el valor de l'scrollbar */
  static bool progress_timeout()
  {
    double new_val;
    if (pdata.activity_mode)
        /* En cas d'estar en un mode pulse */
      pdata.pbar.Pulse();
    else {
      /* En cas d'estar en mode fracció*/
      new_val = pdata.pbar.Fraction + 0.01;
      if (new_val > 1.0)
        new_val = 0.0;

      /* Posem el nou valor */
      pdata.pbar.Fraction = new_val;
    }
    return true;
  }

  /* Canviem el text que es mostrarà a l'scrollbar */
  static void toggle_show_text (object obj, EventArgs args)
  {
    if (pdata.pbar.Text == "")
      pdata.pbar.Text = "some text";
    else
      pdata.pbar.Text = "";
  }

  /* Canviem el mode d'activitat de polsos a fracció */
  static void toggle_activity_mode (object obj, EventArgs args)
  {
    pdata.activity_mode = !pdata.activity_mode;
    if (pdata.activity_mode)
      pdata.pbar.Pulse();
    else
      pdata.pbar.Fraction = 0.0;
  }

  /* Canviem l'orientació de l'scrollbar */
  static void toggle_orientation (object obj, EventArgs args)
  {
    switch (pdata.pbar.Orientation) {
      case Gtk.ProgressBarOrientation.LeftToRight:
        pdata.pbar.Orientation =
Gtk.ProgressBarOrientation.RightToLeft;
        break;
      case Gtk.ProgressBarOrientation.RightToLeft:
        pdata.pbar.Orientation =
Gtk.ProgressBarOrientation.LeftToRight;
        break;
      }
  }

  static void destroy_progress (object obj, DeleteEventArgs args)
  {
    app_quit();
  }

  static void button_click (object obj, EventArgs args)
  {
    app_quit();
  }

  static void app_quit() {
      /* Parem el temporitzador */
    GLib.Source.Remove (pdata.timer);
    pdata.timer = 0;
    Application.Quit ();
```

```
  }

  static void Main()
  {
    Gtk.HSeparator separator;
    Gtk.Table table;
    Gtk.Button button;
    Gtk.CheckButton check;
    Gtk.VBox vbox;

    Application.Init ();

    /* Reservem l'espai per a les dades que enviem als callbacks */
    pdata = new ProgressData();
    pdata.activity_mode = false;
    pdata.window = new Gtk.Window(Gtk.WindowType.Toplevel);
    pdata.window.Resizable = true;
    pdata.window.DeleteEvent += destroy_progress;
    pdata.window.Title = "GtkProgressBar";
    pdata.window.BorderWidth = 0;

    vbox = new Gtk.VBox(false, 5);
    vbox.BorderWidth = 10;
    pdata.window.Add(vbox);
    vbox.Show();

    /* Alineem al centre */
    Gtk.Alignment align = new Gtk.Alignment( 1, 1, 0, 0);
    vbox.PackStart(align, false, false, 5);
    align.Show();

    /* Creem la GtkProgressBar */
    pdata.pbar = new Gtk.ProgressBar();
    pdata.pbar.Text = "";
    align.Add(pdata.pbar);
    pdata.pbar.Show();

    /* Creem un temporitzador per poder canviar el valor */
    pdata.timer = GLib.Timeout.Add(100, new GLib.TimeoutHandler
(progress_timeout) );


    separator = new Gtk.HSeparator();
    vbox.PackStart(separator, false, false, 0);
    separator.Show();
    table = new Gtk.Table(2, 3, false);
    vbox.PackStart(table, false, true, 0);
    table.Show();

    /* Creem un check per escollir si volem o no el text */
    check = new Gtk.CheckButton("Show text");
    table.Attach(check, 0, 1, 0, 1,
        Gtk.AttachOptions.Expand | Gtk.AttachOptions.Fill,
        Gtk.AttachOptions.Expand | Gtk.AttachOptions.Fill,
        5, 5);
    check.Clicked += toggle_show_text;
    check.Show();

    /* Creem un check per escollir el mode d'activitat */
    check = new Gtk.CheckButton("Activity mode");
    table.Attach(check, 0, 1, 1, 2,
        Gtk.AttachOptions.Expand | Gtk.AttachOptions.Fill,
        Gtk.AttachOptions.Expand | Gtk.AttachOptions.Fill,
        5, 5);
    check.Clicked += toggle_activity_mode;
    check.Show();
```

```
    /* Creem un check per escollir l'orientació */
    check = new Gtk.CheckButton("Right to Left");
    table.Attach(check, 0, 1, 2, 3,
        Gtk.AttachOptions.Expand | Gtk.AttachOptions.Fill,
        Gtk.AttachOptions.Expand | Gtk.AttachOptions.Fill,
        5, 5);
    check.Clicked += toggle_orientation;
    check.Show();

    button = new Gtk.Button("close");
    button.Clicked += button_click;
    vbox.PackStart(button, false, false, 0);
    button.CanDefault = true;
    button.GrabDefault();
    button.Show();

    pdata.window.ShowAll();

    Application.Run ();
  }
}
```

Labels

Aquest *widget* és el que s'usa en cas que vulguem mostrar un tros de text no editable. Per a poder tenir-hi esdeveniments fa falta posar-lo dins d'un `EventBox` o un botó. Podem escollir la justificació del text de la manera següent:

```
label.Justify = Justification.Left;
```

Si volem marcar un text, podem usar la propietat `Pattern`, en què podem assignar quins caràcters van subratllats, per exemple:

```
label.Pattern ="___ __ ____    __"
```

Tooltips

Els *tooltips* són missatges que apareixen en mantenir el cursor damunt d'un *widget* durant uns segons. Usar-los és fàcil: primer hem de crear l'objecte `Tooltip`, el *widget* en què volem enllaçar-lo, i hem d'usar la funció següent:

```
tooltip1.SetTip(widget, tooltipText, tooltipPrivate);
```

En aquesta funció, el primer argument és l'element gràfic; el segon, el text que es mostrarà, i el tercer, pot ser nul o un enllaç a un element d'ajuda contextual.

Fletxes

Aquest *widget* ens permet crear una imatge d'una fletxa per poder afegir-la a altres *widgets*. Podem escollir-ne la forma en el moment de crear-la o posteriorment:

```
Arrow arrow1 = new Arrow( arrow_type, shadow_type );
arrow1.SetArrow(arrow_type, shadow_type );
```

Aquí podem escollir com a tipus els següents:

- `Gtk.Arrow.Up`
- `Gtk.Arrow.Down`
- `Gtk.Arrow.Left`
- `Gtk.Arrow.Right`

I com a tipus d'ombra:

- `Gtk.Shadow.In`
- `Gtk.Shadow.Out`
- `Gtk.Shadow.Etched.In`
- `Gtk.Shadow.Etched.Out`

Treeview

Aquest *widget* és segurament el més complet i important de tots. Ens permet mostrar dades d'una manera estructurada en una llista o en un arbre, filtrar, editar i mostrar icones d'aquesta llista. Està dissenyat amb un model Modelo Vista Controlador. Aquest sistema ens permet tenir la informació que es vol mostrar, com la veurem i, finalment, com la tractarem.

Per a poder tractar els diferents tipus de dades, hi ha dos tipus d'objectes:

- `ListStore`: serveix per a desar informació en forma de llista.

- `TreeStore`: serveix per a desar informació en forma d'arbre.

Per a poder visualitzar la informació, els `Treeview` es descomponen en:

- `Treeview widget`: responsable de la visualització del *widget* i de la interacció de l'usuari.

- `TreeViewColumn`: responsable d'una columna que conté un `Cell Renderer`.

- `CellRenderer`: element mínim del *widget*. Es poden usar separats de la columna per a poder compondre'ls i se'n poden usar diversos en una co-

lumna per a posar una imatge i un text junts, per exemple. Poden ser de diferents tipus:

- `CellRendererText`: per a mostrar el text.
- `CellRendererPixbuf`: per a mostrar les imatges.
- `CellRendererProgress`: per a mostrar les barres de progrés.
- `CellRendererCombo`: per a mostrar una caixa desplegable.
- `CellRendererToggle`: per a mostrar una opció de reconeixement.

En aquest exemple podem veure com filtrem les dades d'una llista en què afegim la informació des del codi.

```
public class ExempleLlista
{
  public static void Main ()
  {
    Gtk.Application.Init ();
    new EjemploLista ();
    Gtk.Application.Run ();
  }

  Gtk.Entry filterEntry;

  Gtk.TreeModelFilter filter;

  public ExempleLlista ()
  {
    // Creem una finestra
    Gtk.Window window = new Gtk.Window ("TreeView Example");
    window.SetSizeRequest (500,200);

    // Entry per a poder filtrar
    filterEntry = new Gtk.Entry ();

    // Esdeveniment per a actuar en el canvi de l'Entry anterior
    filterEntry.Changed += OnFilterEntryTextChanged;
    Gtk.Label filterLabel = new Gtk.Label ("Artist Search:");

    // Posem la caixa a la part superior
    Gtk.HBox filterBox = new Gtk.HBox ();
    filterBox.PackStart (filterLabel, false, false, 5);
    filterBox.PackStart (filterEntry, true, true, 5);

    // Creem el TreeView
    Gtk.TreeView tree = new Gtk.TreeView ();

    // Posem el Treeview en una caixa
    Gtk.VBox box = new Gtk.VBox ();
    box.PackStart (filterBox, false, false, 5);
    box.PackStart (tree, true, true, 5);
    window.Add (box);

    // Una columna per a l'artista
    Gtk.TreeViewColumn artistColumn = new Gtk.TreeViewColumn ();
    artistColumn.Title = "Artista";

    // Creem la CellRender per mostar el nom
    Gtk.CellRendererText artistNameCell = new Gtk.CellRendererText
();

    // Afego, la Cellrender a la columna
    artistColumn.PackStart (artistNameCell, true);

    // Creem una columna per a la cançó
```

```
    Gtk.TreeViewColumn songColumn = new Gtk.TreeViewColumn ();
    songColumn.Title = "Canción";

    // Fem el mateix que amb l'artista
    Gtk.CellRendererText songTitleCell = new Gtk.CellRendererText ();
    songColumn.PackStart (songTitleCell, true);

    // Afegim les columnes a la llista
    tree.AppendColumn (artistColumn);
    tree.AppendColumn (songColumn);

    // Diem a les columnes quina informació mostrarem
    artistColumn.AddAttribute (artistNameCell, "text", 0);
    songColumn.AddAttribute (songTitleCell, "text", 1);

    // Creem una llista de dos string per fila
    Gtk.ListStore musicListStore = new Gtk.ListStore (typeof
(string),
                                                      typeof (string));

    // Afegim la informació
    musicListStore.AppendValues ("Danny Elfman", "This is hallowen");
    musicListStore.AppendValues ("Danny Elfman", "Chicago
Soundtrack");
    musicListStore.AppendValues ("Eleftheria", "Dinata");
    musicListStore.AppendValues ("Alkistis", "Lava");

    /* En lloc d'assignar el model directament al Treeview, creem un
       TreeModelFilter, que estarà entre el model (la ListStore) i la
visualització
       (el TreeView) filtrant el model que es visualitza.Actuaria com
a
       Controlador. */
    filter = new Gtk.TreeModelFilter (musicListStore, null);

    // Definim quina funció determinarà quina fila es visualitzarà i
quina no
    filter.VisibleFunc = new Gtk.TreeModelFilterVisibleFunc
(FilterTree);

    /* Assignem el model del TreeView com el filtre. En cas de no usar
el
    filtre, assignaríem directamnte la llista */
    tree.Model = filter;

    // Mostra-ho tot
    window.ShowAll ();
  }

  private void OnFilterEntryTextChanged (object o, System.EventArgs
args)
  {
    // Cridem el filtre perquè reactuï
    filter.Refilter ();
  }

  /* Funció per al filtre */
  private bool FilterTree (Gtk.TreeModel model, Gtk.TreeIter iter)
  {
    string artistName = model.GetValue (iter, 0).ToString ();

    if (filterEntry.Text == "")
      return true;

    if (artistName.IndexOf (filterEntry.Text) > -1)
      return true;
    else
      return false;
  }
}
```

Per a poder crear un arbre, en lloc d'una llista tenim:

```
Gtk.TreeStore musicListStore = new Gtk.TreeStore (typeof (string),
typeof (string));
```

Quan creem un valor nou en el model, assignem qui n'és el pare:

```
Gtk.TreeIter iter = musicListStore.AppendValues ("BSO");
musicListStore.AppendValues (iter, "Danny Elfman", "Town meeting
song");
```

Podem assignar una funció a una columna per obtenir les dades. D'aquesta manera, podem tenir una estructura complexa d'informació, i a partir d'aquestes funcions en què tenim l'iterador d'una llista d'objectes podem decidir quina informació mostrarem.

Per a fer-ho, primer hem de crear una estructura de dades per al nostre model:

```
public class Song
{
        public Song (string artist, string title)
        {
                this.Artist = artist;
                this.Title = title;
        }

        public string Artist;
        public string Title;
}
```

Després, hem de decidir quines funcions ens donaran la informació:

```
private void RenderSongTitle (Gtk.TreeViewColumn column,
Gtk.CellRenderer cell, Gtk.TreeModel model, Gtk.TreeIter iter)
{
        Song song = (Song) model.GetValue (iter, 0);
        (cell as Gtk.CellRendererText).Text = song.Title;
}
private void RenderArtistName (Gtk.TreeViewColumn column,
Gtk.CellRenderer cell, Gtk.TreeModel model, Gtk.TreeIter iter)
{
        Song song = (Song) model.GetValue (iter, 0);
        if (song.Artist.StartsWith ("X") == true) {
                (cell as Gtk.CellRendererText).Foreground = "red";
        } else {
                 (cell as Gtk.CellRendererText).Foreground =
"darkgreen";
        }
        (cell as Gtk.CellRendererText).Text = song.Artist;
}
```

Finalment, afegim una llista d'objectes `Song` i la definim com a model del `TreeView`. Una vegada fet això, solament ens cal definir les funcions com a elements per a decidir el contingut de les cel·les.

```
songs = new ArrayList ();
songs.Add (new Song ("Danny Elfman", "Making Christmas"));

Gtk.ListStore musicListStore = new Gtk.ListStore (typeof (Song));
foreach (Song song in songs) {
       musicListStore.AppendValues (song);
}

artistColumn.SetCellDataFunc (artistNameCell, new
Gtk.TreeCellDataFunc (RenderArtistName));
songColumn.SetCellDataFunc (songTitleCell, new
Gtk.TreeCellDataFunc (RenderSongTitle));

tree.Model = musicListStore;
```

Podem definir que una cel·la és editable i assignar una funció com a esdeveniment de la seva edició:

```
artistNameCell.Editable = true;
artistNameCell.Edited += artistNameCell_Edited;
```

Un exemple d'una funció per a rebre aquest esdeveniment és el següent:

```
private void artistNameCell_Edited (object o, Gtk.EditedArgs args)
{
 Gtk.TreeIter iter;
 // Obtenim l'iterador on s'ha editat
 musicListStore.GetIter (out iter, new Gtk.TreePath (args.Path));

 Song song = (Song) musicListStore.GetValue (iter, 0);
 song.Artist = args.NewText;
}
```

Menús

A GTK# hi ha tres tipus de classes per a fer menús:

- `Gtk.MenuItem`: representa un element en un menú.
- `Gtk.Menu`: recull tots els elements en un menú i crea submenús per als ítems.
- `Gtk.MenuBar`: crea una barra amb tots els ítems i crea submenús.

Per poder-ne crear un haurem de fer el següent:

- crear un `MenuBar`,
- crear un `MenuItem` per a cada element del primer nivell:
  – afegim un `Menu` com a submenú per a cada element de primer nivell,

– creem cada element del submenú,
– afegim els elements de primer nivell a `MenuBar`.

Es poden crear claus `Mnemonic` que ens permeten accedir-hi directament des del teclat amb una combinació de la tecla Alt i la lletra escollida.

Per començar un exemple, tenim la creació del `Menubar` i els elements de primer nivell:

```
MenuBar mn = new MenuBar();

Menu file_menu = new Menu();
MenuItem item = new MenuItem ("_File");
item.Submenu = file_menu;
mv.Append(item);

Menu edit_menu = new Menu();
item = new MenuItem("_Edit");
item.Submenu = edit_menu;
mb.Append(item);
```

Podem afegir-hi elements de l'`Stock` i separadors:

```
// Element que s'encarrega de gestionar tots els acceleradors de
teclat de la finestra
AccelGroup agrp = new AccelGroup();
window.AddAccelGroup(agrp);

item = new ImageMenuItem(Stock.Open,agrp);
file_menu.Append(item);
file_menu.Append(new SeparatorMenuItem ());
```

O podem afegir elements manuals:

```
item = new MenuItem("_Transoform");
edit_menu.Append(item);
```

També es poden usar elements del menú més complexos, com `CheckMenuItem`, en què podem tenir un botó de reconeixement, o el `RadioMenuItem` per a incloure una opció de triar.

Per a poder fer accessibles des del teclat les opcions en el cas que no siguin ítems de `Stock`, haurem d'afegir els acceleradors a mà:

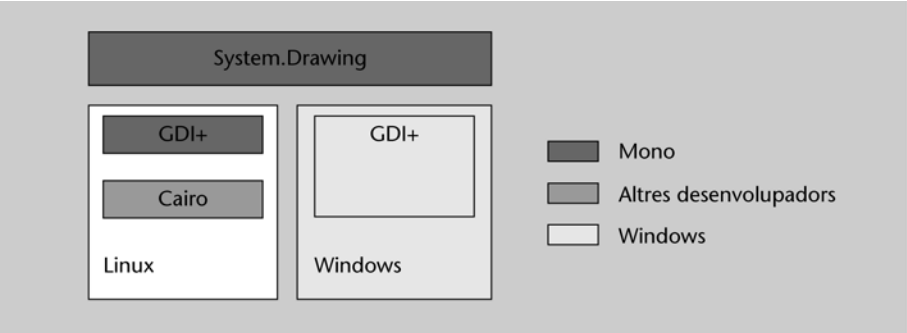
```
/* Escollim la lletra que usarem i el modificador (la R i el
Control) i definim
que l'acceleració sigui visible en el menú */
item.AddAccelerator ("activate",agrp,
   new AccelKey ( Gdk.key.R,
Gdk.ModifierType.ControlMask,AccelFlags.Visible));
```

3.2.3. *Canvas*

A Mono hi ha dues maneres de dibuixar: amb l'API de la libgdi `System.Drawing` i amb la biblioteca Cairo. A part d'aquestes biblioteques, tenim l'opció, més eficient, de la `PixBuf` per a imatges i `Pango` per a text. Aquí explicarem com es poden usar aquestes quatre biblioteques.

Per a entendre com funciona la biblioteca per a dibuixar tenim la biblioteca libgdi nadiua a Windows i portada a Linux pel projecte Mono. Aquesta biblioteca és la base de `System.Drawing`. A Linux disposem de la biblioteca Cairo, que ens permet tenir l'espai de noms `Mono.Cairo` per a accedir als seus *bindings*. Tal com s'ha dit anteriorment, GTK# està implementat sobre la biblioteca Cairo des de la versió 2.8 a baix nivell.

Figura 4



Dibuixar

L'API de `System.Drawing` és semblant a la forma que té PDF 1.4 per a definir l'estructura del document. És una biblioteca que implementa la mateixa API de libgdi, però que hi permet l'accés des de C#. A Linux, la implementació de libgdi es basa en Cairo per a les operacions gràfiques d'alt rendiment.

Imatges i DrawingArea

Per a poder treballar amb àrees d'imatges i gràfics des de GTk#, hem d'usar aquests dos tipus de dades.

- **`Gtk.DrawingArea`**: *widget* de GTK per a definir una àrea de dibuix.

- **`Gdk.Pixbuf`** y **`Gdk.Pixmap`**: objectes de GDK que ens permeten treballar amb imatges de bits. Aquests bits s'usen en qualsevol lloc on necessitem una imatge, ja sigui un botó, un `TreeView` o una `Drawing Area`.

Per a poder carregar una imatge d'un arxiu, haurem d'usar l'objecte `Pixbuf`:

```
Gdk.Pixbuf buffer = new Pixbuf(null,"archiu.png"); # aquest arxiu
pot ser un resource dins o fora de l'Assembly
```

Una vegada tinguem la imatge carregada en un *buffer*, podem escalar-la o treballar-hi a gust nostre amb un conjunt de funcions de la classe.

Per a crear una àrea de dibuix a GTK, tenim el constructor:

```
Gtk.DrawingArea darea = new DrawingArea();
```

Una vegada tenim l'àrea, podem saber-ne la mida amb la propietat `Allocation` i dibuixar-hi un *pixmap* amb la funció `DrawDrawable` del Gdk de la Window de `DrawingArea`. Els *pixmap* ens serveixen per a fer un *doublebuffer* i compondre en aquest espai de memòria la imatge que volem mostrar. Aquest objecte té la funció `DrawPixbuf`, que ens permet mapar un *pixbuf* a un *pixmap*. Cada vegada que canviem un *pixmap*, hem de cridar la funció `QueueDrawArea` de la `DrawingArea` per a llançar l'esdeveniment `Expose`.

```
...
// Creem una àrea
darea = new DrawingArea();
darea.SetSizeRequest(200,300);

// Afegim els esdeveniments
darea.ExposeEvent += Expose_Event;
darea.ConfigureEvent += Configure_Event;
...

// col·loquem en un pixmap un pixbuf i el mostrem en la drawing area
static voig PonerPixbuf (GdkPixbuf buf)
{
  // de la mateixa manera podríem dibuixa línies, polígons, arcs,
...
  pixmap.DrawPixbuf(darea.Style.BlackGC, buf, 0, 0, 0, 0,
buf.Width, buf.Height, RgbDither.None, 0, 0);
  darea.QueueDrawArea (0, 0, buf.Width, buf.Height);
}

// Configurem l'àrea de dibuix
static void Configure_Event (object obj, ConfigureEventArgs args)
  Gdk.EventConfigure ev = args.Event;
  Gdk.Window window = ev.Window;
  Gdk.Rectangle allocation = darea.Allocation;
  // Creem un pixmap de la mida de l'àrea
  pixmap = new Gdk.Pixmap ( window, allocation.Width,
allocation.Height, -1);

  // Posem un quadre blanc en el pixmap
  pixmap.DrawRectangle (darea.Style.WhiteGC, true, 0, 0,
allocation.Width, allocation.Height);
}

static void Expose_Event (object obj, ExposeEventArgs args)
{
  // Esdeveniment que mostrarà el pixmap a la drawing area.
  Gdk.Rectangle area = args.Event.Area;
  args.Event.Window.DrawDrawable (darea.Style.WhiteGC, pixmap,
area.X, area.Y, area.X, area.Y, area.Width, area.Height );
}
```

Cairo

Cairo és una biblioteca d'imatges 2D vectorials que ens permet tenir diferents motors per a visualitzar-la. Hi ha motors implementats sobre OpenGL per poder utilitzar l'acceleració gràfica i disposar d'efectes.

Per a poder treballar amb Cairo des de GTK#, hem de fer-ho amb la classe `Graphics`, que pot estar associada a una `Surface`. Una `Surface` pot ser una finestra, un *buffer* o un arxiu de disc.

Per a poder obtenir l'objecte `Graphics` d'un element `Drawable` (per exemple `DrawingArea`), hem de convertir l'àrea de dibuix en un `Graphics` de Cairo:

```
// Per a poder fer això es requereix la versió superior a 2.8 de
Gdk.
// Hem de comprovar que tenim el mapat de la biblioteca
[DllImport("libgdk-x11-2.0.so")]
internal static extern IntPtr gdk_cairo_create (IntPtr raw);

// Aquesta funció obtindrà el Graphics de Cairo d'un Drawable
public static Cairo.Graphics CreateDrawable (Gdk.Drawable
drawable)
{
    Cairo.Graphics g = new Cairo.Graphics (gdk_cairo_create
(drawable.Handle));
    if (g == null)
        throw new Exception ("Couldn't create Cairo Graphics!");

    return g;
}
// Esdeveniment d'exposed en què dibuixarem el contingut
void OnDrawingAreaExposed (object o, ExposeEventArgs args)
{
    DrawingArea area = (DrawingArea) o;
    // Obtenim el Graphics de Cairo
    Cairo.Graphics g = Graphics.CreateDrawable (area.GdkWindow);

    // Dibuixem

    // Hem d'alliberar la memòria usada per Cairo.
    // A la versió del llibre Mono.Cairo no se sincronitza amb el
Garbage Collector
   ((IDisposable) gr.Target).Dispose ();
    ((IDisposable) g).Dispose ();
}
```

Una vez tenim el `Graphics` de Cairo, podem dibuixar amb les funcions de Cairo.

Les primitives bàsiques són les següents:

- `PointD (x,y)`: objecte que defineix un punt en l'àrea.
- `Graphics.MoveTo(PointD)`: funció per a moure'ns a un punt determinat.
- `Graphics.LineTo(PointD)`: funció per a definir una línia des del punt actual fins al nou.

- `Graphics.CurveTo(PointD, PointD, PointD)`: funció per a fer una corba de Bezier des del punt actual fins al punt 3 usant com a punts de control el punt 1 i el punt 2.
- `Graphics.ClosePath()`: tanca la línia definida.

Una vegada tenim les línies definides, podem omplir d'un color:

```
Graphics.Color = new Color(0,0,0);
Graphics.FillPreserve();
```

I dibuixar la línia definida:

```
Graphics.Color = new Color(1,0,0);
Graphics.Stroke();
```

S'ha de tenir en compte que per a usar un color, primer hem de definir-lo en la propietat `color` i després hem de cridar la funció que l'usarà. L'ús de `Fill` o `FillPreserve` i `Stroke` o `StrokePreserve` es diferencia en el fet de conservar o no el `Path` creat.

Hi ha tres propietats que es poden enregistrar temporalment amb la funció `Save` i `Restore` per a poder modificar temporalment el següent:

- `Color`: el color,
- `LineWidth`: la mida de la línia,
- `LineCap`: el dibuix de la punta de la línia.

A partir d'aquestes funcions bàsiques i de moltes altres que aporta l'API de Cairo, com ara dibuixar gradients o utilitzar transparències, podem dibuixar el que vulguem en la nostra `DrawingArea`.

Pango

Pango és una biblitoeca per a treballar amb text en una àrea `Drawing` de Gdk. Aquesta biblioteca és molt més ràpida que Cairo, però també més complexa. Per a poder mostrar text, hi hem de crear un *layout* i dir a l'`ExposeEvent` que el mostri.

```
using System;
using Gtk;

public class MyWindow
{

  Pango.Layout layout;
```

```
  Gtk.DrawingArea da;
  int width = 400;
  int height = 300;

  static void Main()
  {
    Application.Init();
    new MyWindow();
    Application.Run();

  }

  public MyWindow ()
  {

       Window w = new Window("pango");
    w.SetDefaultSize (width, height);
    w.DeleteEvent += new DeleteEventHandler (OnMyWindowDelete);

         // Creem una àrea de dibuix
    da = new Gtk.DrawingArea();
    da.SetSizeRequest(width, height);
    da.ExposeEvent += Expose_Event;

         // Creem un layout de Pango
         layout = new Pango.Layout(w.PangoContext);
         // Definim la mida amb píxels
         layout.Width = Pango.Units.FromPixels(300);
         // Definim el wrapping en la paraula
         layout.Wrap = Pango.WrapMode.Word;
         // Definim l'alineació a l'esquerra
         layout.Alignment = Pango.Alignment.Left;
         // Definim el tipus de font
         layout.FontDescription =
Pango.FontDescription.FromString("Ahafoni CLM Bold 100");
         // Definim el contingut del layout amb la paraula "hola"
de color blau
         layout.SetMarkup("<span color=" + (char)34 + "blue" +
(char)34 + ">" + "Hola" + "</span>");

    w.Add(da);
    w.ShowAll ();
  }

  void Expose_Event(object obj, ExposeEventArgs args){
         // Funciona per a dibuixar el layout a la drawing area
    da.GdkWindow.DrawLayout (da.Style.TextGC (StateType.Normal),
5, 5, layout);
  }

  void OnMyWindowDelete (object sender, DeleteEventArgs a)
  {
    Application.Quit ();
    a.RetVal = true;
  }
}
```

3.3. Ús

Ja hem vist molts possibles usos de la biblioteca GTK# en els exemples anteriors. Però hi ha un parell de coses que han de quedar clares per a utilitzar-la correctament: el bucle d'esdeveniments principal i el *drag and drop* per a moure informació d'un costat a l'altre de l'aplicació.

3.3.1. Bucle d'esdeveniments

La manera de funcionar GTK# és a partir d'esdeveniments; hi ha tres funcions que controlen qualsevol aplicació que utilitzi aquesta biblioteca:

- **`Application.init`**`()`: aquesta funció permet inicialitzar tota la biblioteca GTK#. A continuació, podem definir tota l'estructura de finestres o usar la biblioteca Glade per a obtenir-la.

- **`Application.run`**`()`: quan ja hem definit tota la interfície gràfica, podem executar aquesta funció. En el moment en què s'executa, s'interromp el programa i ja no s'executa el que pugui venir a continuació. A partir d'aquesta funció, el programa solament està atent als diferents esdeveniments que s'executen per tal d'anar cridant totes les funcions de *callback*.

- **`Application.quit`**`()`: funció que cridem per acabar el bucle d'esdeveniments i sortir de l'aplicació.

Aquest sistema de funcionament té un problema: en el moment d'atendre un esdeveniment, el programa es bloqueja i no redibuixa els diferents elements ni pot atendre esdeveniments nous. En cas que hi hagi esdeveniments que puguin trigar i bloquejar l'aplicació, podem usar *threads* per a redibuixar-la. El problema d'usar *threads* és que la biblioteca no es *threads-safe*, amb la qual cosa gestionar la interfície des d'un `thread` pot fer acabar el programa amb un error. Per a solucionar-ho tenim diferents eines:

Idle

Es pot definir un tros de codi que s'executa quan el sistema no té cap esdeveniment per a atendre.

```
void Start ()
   {
       // Definim la funció que executarem quan no tingui res per fer
       GLib.Idle.Add (new IdleHandler (OnIdleCreateThumbnail));
   }
```

Timeouts

Podem definir un `timeout` per a poder actuar si, en passar un temps, el programa no ha reaccionat davant un esdeveniment.

```
void StartClock ()
    {
        // Cada 1000 mil·lisegons executem la funció update_status
```

```
        GLib.Timeout.Add (1000, new GLib.TimeoutHandler (update_status));
    }

    bool update_status ()
    {
        // Canviem el text d'un label amb l'hora.
        time_label.Text = DateTime.Now.ToString ();

        // Si retornem false, el timeout deixarà de tornar a executar-se
       // Si retornem true, el timeout tornarà a carregar-se i executarà altra vegada la funció

        return true;
    }
```

Invoke

Podem executar una funció per a invocar algun canvi en la interfície gràfica des d'un `thread`.

```
using Gtk;

class TreballaFort {
  static Label label;

  static void Main ()
  {
    Application.Init ();
    Window w = new Window ("Cray en una finestra");
    label = new Label ("Computant");

    // Llancem un thread que farà els càlculs de gran cost
    Thread thr = new Thread (new ThreadStart (ThreadRoutine));
    thr.Start ();

    Application.Run ();
  }

  static void ThreadRoutine ()
  {
        // Càlcul llarg
        LargeComputation ();
      // Invoquem Gtk per a delegar-hi el canvi del text del label
        Gtk.Application.Invoke (delegate {
              label.Text = "Done";
        });
  }

  static void LargeComputation ()
  {
    // Càlculs llargs
  }
}
```

ThreadNotify

Una altra manera de fer el mateix és fent servir `ThreadNotify`. D'aquesta manera, creem una notificació d'un `thread` i des d'aquest `thread` el cridem quan acabem la feina.

```
using Gtk;

class TreballaFort2 {

  static ThreadNotify notify;
  static Label label;

  static void Main ()
  {
    Application.Init ();
    Window w = new Window ("Cray en una finestra");
    label = new Label ("Computant");

    // Creem un thread per a fer un càlcul
    Thread thr = new Thread (new ThreadStart (ThreadRoutine));
    thr.Start ();
    // Creem un threadnotify amb la funció ready
    notify = new ThreadNotify (new ReadyEvent (ready));
    Application.Run ();
  }

  // Funció que s'executarà quan el thread cridi el notify.
  static void ready ()
  {
      label.Text = "Done";
  }

  static void ThreadRoutine ()
  {
        // Fem els càlculs llargs
        LargeComputation ();
        // Aixequem el notify
        notify.WakeupMain ();
  }

  static void LargeComputation ()
  {
    // lots of processing here
  }
}
```

3.3.2. *Drag and drop*

Drag and drop és un mètode efectiu per a permetre a l'usuari manipular les dades, dins de l'aplicació i entre diferents aplicacions. A GTK# tenim un origen de *drag* en què podem definir els tipus que poden proveir i els destins de *drop* que descriuen quin tipus l'accepten. Si es produeixen les dues coses, es pot produir un *drag and drop*.

Per a poder definir que un objecte és destí d'un *drag and drop*, haurem d'usar la funció següent:

```
...
// Definim la taula de tipus d'elements que podem rebre.
private static Gtk.TargetEntry[] target_table =
       new TargetEntry [] {
          new TargetEntry ("text/uri-list", 0, 0),
          new TargetEntry ("application/x-monkey",0,1),
       };
```

```
....

// Definim què podem rebre en quin widget i quina serà l'acció del
drag and drop
Gtk.Drag.DestSet (widget,DestDefaults.All, target_table,
Gdk.DragAction.Copy);
// Creem un esdeveniment per a tractar el que rebem
widget.DragDataReceived += Data_Received;

....

// Funció per a rebre el drag
static void Data_Received ( object o , DragDataReceivedArgs args )
{
  bool success = false

  // Obtenim les dades seleccionades
  string data = System:Text.Encoding.UTF8.GetString (
args.SelectionData.Data );

  // És una uri-list o un x-monkey
  switch (args.Info) {

    case 0: // uri-list
      // Mostrem el contingut de la uri-list
      string [] uri_list = Regex.Split (data, "\r\n");
      foreach (string u in uri_list) {
        if (u.Length>0)
          System.Console.WrilteLine ("Url : {0}",u);
      }
      success = true;
      break;

    case 1: // x-monkey
      // Mostrem les dades enviades
      System.console.WriteLine("Monkey '{0}'", data);
      success = true;
      break;
  }
  //Finalitzem el drag and drop
  Gtk.Drag.Finish (args.Context,success, false, args.Time);
}
```

Per a poder enviar hem d'usar l'estructura següent:

```
....
// Definim el tipus de dades que usarem en enviar
private static Gtk.TargetEntry [] source_table =
   new TargetEntry [] {
       new TargetEntry ("application/x-monkey", 0, 0),
   };
...

// Definim les dades que es poden enviar i el widget d'origen
Gtk.Drag.SrouceSet(widget, Gdk.ModifierType.Button1Mask,
source_table, DragAction.Copy);
// Afegim la funció que definirà les dades que s'enviaran
widget.DragDataGet += Data_Get;
// Afegim la funció que s'executarà quan comencem el drag and drop
widget.DragBegin += Drag_Begin;
 ...

static void Data_Get (object o, DragDataGetArgs args)
{
```

```
  // Obtenim l'espai on enviarem la informació
  Atom [] targets = args.Context.Targets;

  // Afegim la informació que vulguem enviar
  args.SelectionData.Set (targets [0], 8,
System.Text.Encoding.UTF8.GetBytes("Rupert"));
}

static void Drag_Begin (object o, DragBeginArgs args)
{
  // Carreguem un pixbuf carregat amb anterioritat com a icona del
cursor
  Gtk.Drag.SetIconPixbuf (args.Context, dibuix, 0, 0);
}
```

4. Glade

L'eina més estesa per a desenvolupar la interfície gràfica d'usuari en GTK+ és Glade. No es tracta d'un entorn de desenvolupament complet (IDE), sinó d'una eina amb què gestionar fàcilment el codi encarregat de crear la GUI. Posteriorment, hem de vincular la resta de codi de la nostra aplicació a la capa gràfica generada amb Glade.

4.1. Instal·lació

Actualment, Glade està disponible en els respositoris de pràcticament la totalitat de les distribucions, per tant es pot instal·lar còmodament mitjançant el sistema de gestió de paquets.

Com passa amb la resta de les aplicacions amb llicència GPL, en tenim el codi font a disposició, i per tant és possible que vulguem compilar les fons pel nostre compte. En aquest cas, hem de descarregar-nos les fonts des de la pàgina web del projecte i posteriorment les hem de descomprimir en el nostre espai de disc. Com és habitual, primer executarem `./configure` i, en acabar, `make` per a compilar el codi. Si tot ha funcionat correctament, podem passar a mode superusuari i executar `make install` per acabar la instal·lació.

Una vegada hem instal·lat Glade mitjançant el sistema de gestió de paquets o bé a apartir del codi font, podem procedir a executar-lo amb l'ordre `glade-2`.

Web recomanada

Podeu accedir a la pàgina web del projecte des de l'adreça següent <http://glade.gnome.org>.

4.2. Glade

Hi ha dues maneres de traballar amb Glade:

- usar-lo per a generar el codi font en els llenguatges suportats per crear la interfície per a posteriorment incoporar el codi obtingut al nostre projecte i omplir les funcions de gestió d'esdeveniments;
- o bé usar-lo per a crear un arxiu de definició de la interfície en format XML per a posteriorment carregar-lo en temps d'execució des del codi de la nostra aplicació i vincular dinàmicament les nostres funcions de gestió d'esdeveniments.

Observació

En aquest curs no podrem usar la primera opció, ja que a data de juny del 2006 Glade no suporta cap dels llenguatges disponibles per a Mono, en concret el que estem utilitzant, C#. D'aquesta manera, ens centrarem en la segona alternativa de treball.

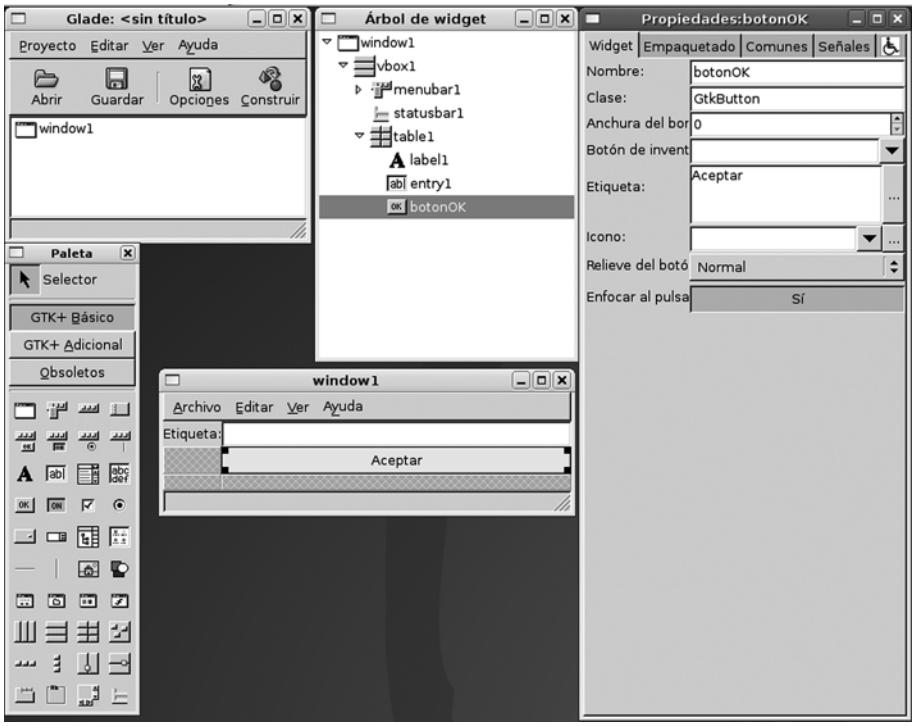
El fet de fer tot el disseny de la interfície amb una eina *ad hoc* com Glade, i poder-ho carregar i vincular dinàmicament en temps d'execució, ens ofereix molta flexibilitat. Entre altres coses, ens serà possible modificar part de la interfície gràfica sense necessitat de recompilar el codi font de la nostra aplicació.

Vegem primer quin és el mode de disseny de Glade i posteriorment com és l'arxiu XML que ens genera.

La filosofia de treball de Glade parteix de la base d'un sistema de finestres independents compostes pel següent (figura 6):

1) la finestra principal, que ens permet crear, carregar i emmagatzemar el projecte actual;
2) la paleta, que conté els elements gràfics (*widgets*) que podem inserir en la interfície;
3) l'arbre d'elements, que ens mostra la jerarquia dels elements que componen la interfície;
4) les propietats, que ens permet visualitzar i editar les propietats de l'element gràfic que tinguem seleccionat.

Figura 5



Primer haurem de crear una finestra sobre la qual podrem disposar altres elements gràfics. Per a fer-ho, és suficient prémer el primer botó de la paleta anomenat "Finestra". Automàticament es crearà un element gràfic de primer nivell en l'arbre d'elements, que serà la finestra creada, i aquesta finestra es mostrarà com una finestra més de l'entorn Glade.

A partir d'aquest punt, haurem d'anar escollint elements gràfics de la paleta i els anirem dipositant en algun altre element tipus contenidor de què disposem. Per exemple, podem seleccionar l'element "Caixa vertical" i inserir-lo a la finestra. En aquest moment, podem escollir el nombre de divisions verticals que vulguem crear. Suposem que en creem tres per a crear les zones de menú, contingut i estat típiques d'una aplicació convencional. Una vegada hem aca-

bat de crear-les, veurem les tres divisions verticals a la finestra de l'aplicació. Serà necessari escollir els elements gràfics "Barra de menú" i "Barra d'estat", i inserir-los en la primera i la darrera divisió vertical. Finalment, en la divisió central situarem la resta dels elements de la nostra aplicació exemple. Creem un contenidor "Taula" de tres files i dues columnes. En la primera cel·la inserirem una "Etiqueta" i en la segona, una "Entrada de text". Per acabar, a la cel·la central dreta col·locarem un "Botó".

Una vegada hem disposat tots els elements gràfics de la nostra interfície, és possible modificar-ne les propietats mitjançant la finestra "Propietats". Per exemple, podem canviar el text que es mostra a l'usuari, la propietat "Etiqueta", o també podem canviar l'identificador intern amb què hem de fer referència a l'element gràfic des del nostre codi de programa, la propietat "Nom".

Després de crear la interfície intuïtivament, com hem vist, la guardem en format XML mitjançant l'opció "Desar" de l'entrada del menú "Projecte".

Si analitzem el contingut de l'arxiu XML, podrem observar l'estructura següent:

```
<?xml version="1.0" standalone="no"?> <!--*- mode: xml -*-->
<!DOCTYPE glade-interface SYSTEM "http://glade.gnome.org/glade-
2.0.dtd">

<glade-interface>

<widget class="{classe}" id="{identificador}">
  <property name="{nom}">{valor}</property>
  ...
  <child>
    {elements gràfics continguts}
 </child>
</widget>

</glade-interfície>
```

A partir d'aquí, farem un pas més, gestionar aquest tipus d'arxiu des del codi de la nostra aplicació, encara que, si en algun moment necessitem retocar-ne el contingut, podrem fer-ho directament sobre l'XML o bé usar altre cop Glade per fer-ho més còmodament.

4.3. Glade#

Una vegada hem definit l'estètica de la interfície gràfica d'usuari amb Glade i, per tant, disposem de l'arxiu XML que en conté la codificació, el pas següent serà manipular-lo des del codi del nostre programa. Per a fer-ho, usarem la biblioteca Glade#. L'ús més senzill consisteix a crear un objecte `Glade.XML` a partir de l'arxiu XML i posteriorment vincular de manera automàtica tots els elements gràfics `[Widget]` que hàgim declarat en el nostre codi. A partir

d'aquest moment, podem treballar amb els objectes vinculats als elements gràfis com si els haguéssim creat explícitament en el nostre codi.

Vegem un exemple senzill d'una aplicació que mostra l'àrea d'un rectangle a partir de les dades de base i alçada que l'usuari introdueix en dues caixes de text:

```
using System;
using Gtk;
using Glade;
public class ExempleGlade
{
  [Widget] Window finestra;
  [Widget] Button bCalcular;

  [Widget] Entry tBase;
  [Widget] Entry tAltura;
  [Widget] Entry tArea;

  public static void Main (string[] args)
  {
    new ExempleGlade (args);
  }

  public ExempleGlade (string[] args)
  {
    Application.Init();

    Glade.XML gxml = new Glade.XML (null, "guiExempleGlade.glade",
"finestra", null);
    gxml.Autoconnect (this);

    bCalcular.Clicked   += EsdevenimentBotoPolsat;
    ventana.DeleteEvent += new
DeleteEventHandler(EsdevenimentTancarFinestra);

    Application.Run();
  }

  public void EsdevenimentBotoPolsat(object obj, EventArgs args)
  {
    int resultado  = Int32.Parse(tBase.Text) *
Int32.Parse(tAltura.Text);
    tArea.Text     = resultat.ToString();
  }

  static void EsdevenimentTancarFinestra(object obj,
DeleteEventArgs args)
  {
    Application.Quit();
  }
}
```

Podem observar com el constructor de la nostra classe aplicació construeix l'objecte `Glade.XML` a partir de l'arxiu `"guiExempleGlade.glade"`, fent servir com a finestra inicial l'etiqueta amb `"finestra"`. Posteriorment, es crida el mètode `Autoconnect` de l'esmentat objecte que vincula tots els membres de l'objecte aplicació que tenen l'atribut `[Widget]`. I, per últim, s'indiquen quins mètodes s'han d'usar per als esdeveniments de botó premut i de tancar l'aplicació.

Per a poder compilar el codi C# que usa Glade# és necessari incloure la referència a l'espai de noms 'glade' que hi ha en el paquet glade-sharp i l'arxiu XML de definició.

```
mcs -pkg:glade-sharp -resource:guiExempleGlade.glade ExempleGlade.cs
```

D'aquesta manera, l'executable final incorpora l'arxiu XML en el seu interior. No obstant això, pot ser-nos interessant vincular-lo però no incloure'l, de manera que haurem de distribuir l'arxiu XML juntament amb l'executable. Si optem per aquesta opció, podrem alterar part de la interfície gràfica modificant l'arxiu XML i sense necessitat de tornar a recompilar l'aplicació.

```
mcs -pkg:glade-sharp -linkresource:guiEsempleGlade.glade
ExempleGlade.cs
```

4.4. Maneig d'esdeveniments

Sobre els delegats podeu veure l'apartat 2.

La programació de la interfície gràfica, seguint el paradigma usat per Glade, està orientada a esdeveniments. D'aquesta manera, el codi de control dels elements gràfics es dedica a reaccionar en els esdeveniments que s'hi produeixen. En treballar amb el llenguatge C#, el mecanisme que s'usa perquè s'esdevingui aquesta funció és el dels delegats. Els delegats ens permeten subscriure més d'un mètode, que es cridarà en el moment en què s'executi el delegat.

En l'exemple del subapartat anterior ja es feien aquests passos de manera explícita en el mètode constructor, però hi ha una manera més senzilla d'enllaçar els mètodes de maneig d'esdeveniments quan es treballa en Glade: es tracta de declarar els senyals que pot rebre cada element gràfic. Per a fer-ho, només cal situar-se a la pestanya "Senyals" de la finestra de propietats de Glade. Des d'aquí podem triar el nom de la funció que s'ha de subscriure a cada un dels esdeveniments possibles. Aquesta informació es desa a l'arxiu XML:

```
<signal name="{esdeveniment}" handler="{mètode}" />
```

Posteriorment, en carregar l'arxiu XML de Glade des del programa i usar el mètode `Autoconnect`, a més de vincular tots els *widgets* declarats en el codi amb els elements gràfics, també subscriurà tots els mètodes amb els noms indicats als esdeveniments corresponents.

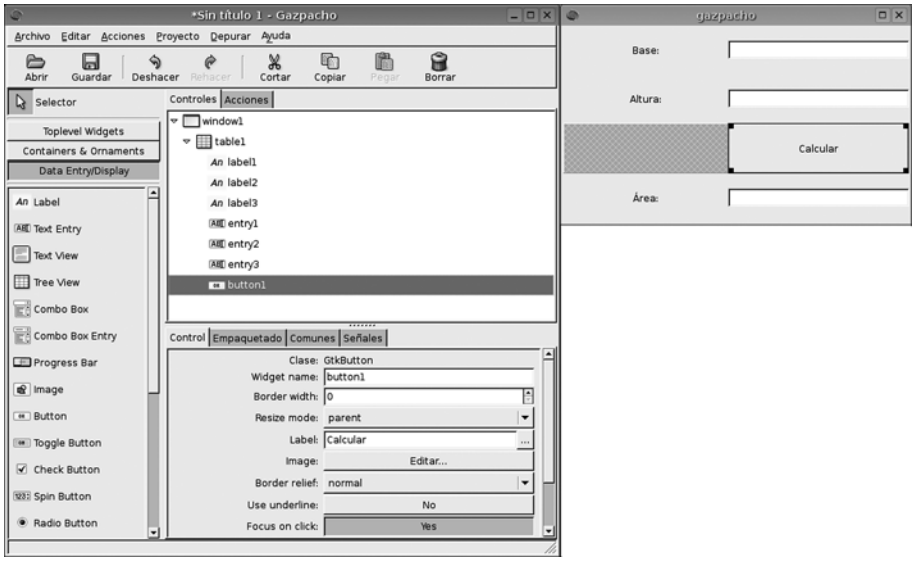
4.5. Alternatives: Gazpacho i Stetic

Hi ha diverses alternatives a Glade per a dissenyar la interfície gràfica d'una aplicació, entre elles Gazpacho i Stetic.

Gazpacho, es tracta d'una aplicació desenvolupada en Python que genera arxius en el format XML de Glade, però amb una interfície lleugerament dife-

rent. En lloc de mantenir per separat totes les finestres del dissenyador, aquestes finetres estan totes integrades en una de sola, com es mostra en la figura 6.

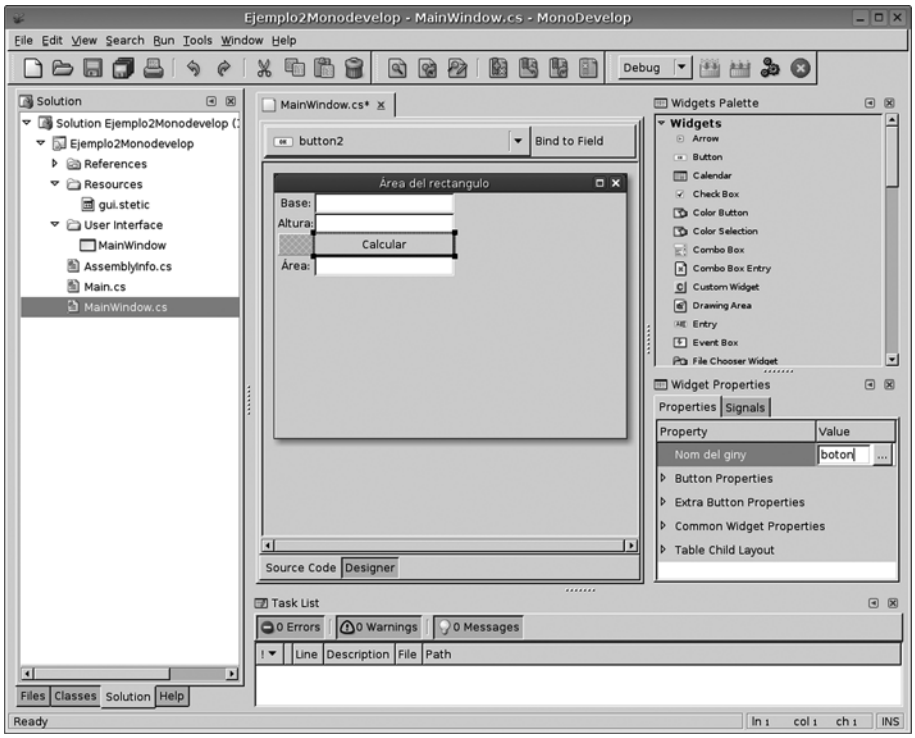
Figura 6



Els seus creadors estan convençuts que aquesta distribució és molt més còmoda. A més, incorpora altres possibilitats amb l'ànim de millorar l'eina, com ara l'ús de plantilles per a elements gràfics. Aquesta eina permet definir un nou element disponible per a ser inserit en la interfície a partir de qualsevol altra que ja tinguem dissenyada, de manera que podem crear la nostra pròpia biblioteca d'elements per a la interfície. Algunes d'aquestes característiques, com les plantilles, s'estan incorporant a versions futures de Glade.

L'aplicació **Stetic** és molt semblant a Gazpacho. La diferència principal rau en el fet que Stetic ha estat empotrada a Monodevelop, fet que facilita el disseny simultani de la interfície i el codi intern de la nostra aplicació. En usar el dissenyador Stetic empotrat, Monodevelop genera automàticament dos arxius: gui.stetic i generated.cs. El primer és un arxiu de format compatible amb Glade, en què s'emmagatzema el disseny creat. El segon arxiu es genera a partir del primer i correspon al codi necessari per a crear la interfície en C#. D'aquesta manera, el funcionament correspon a l'alternativa del que s'ha exposat en els subapartats anteriors. És a dir, en lloc de generar la interfície en temps d'execució, a partir del seu arxiu de definició XML, és generat en temps de compilació. D'altra banda, la integració del dissenyador d'interfície amb la resta de l'IDE permet altres comoditats, com el fet que, en crear "senyals", s'insereixen directament els mètodes corresponents en el codi perquè puguem omplir-los

Figura 7



Web recomanada

Teniu més informació en <http://pixane.net/blog/?p=47>.

5. *Widgets* avançats

5.1. GtkHtml#

El primer element gràfic avançat que veurem és GtkHtml#, que és un element per a visualitzar i editar HTML a les nostres aplicacions. Es tracta d'un element senzill i eficaç amb suport bàsic d'HTML que no inclou altres tecnologies com fulls d'estil (CSS) ni JavaScript. Si necessitem més prestacions, haurem de recórrer al Gecko#, que s'exposa en el subapartat següent.

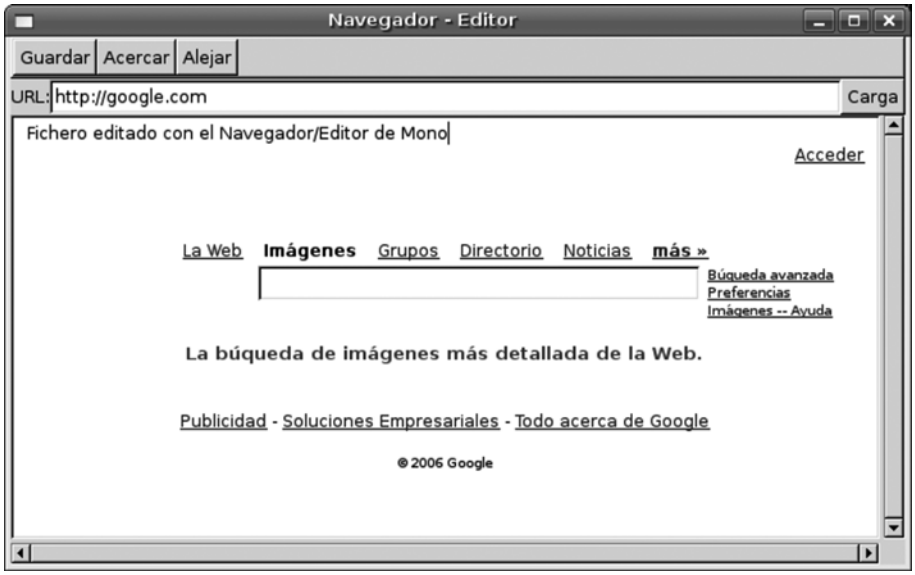
Entre els diferents membres de la classe GtkHtml#, destacamem els següents per a proporcionar una idea de les prestacions que tenen:

- Propietats:
  - `Editable`: commutar a cert per a permetre a l'usuari modificar el contingut.
  - `InlineSpelling`: commutar a cert per a activar la correcció ortogràfica.
- Mètodes:
  - `Begin(), Write(HTMLStream, string, int), End(HTMLStream, HTMLStreamStatus)`: carregar contingut HTML a partir d'un *stream* d'entrada.
  - `Save(HTMLSaveReceiverFn), Export(string, HTMLSaveReceiverFn)`: extraure el contingut HTML del *widget*.
  - `Print(Gnome.PrintContext)`: imprimir el contingut HTML.
  - `ZoomIn(), ZoomOut(), ZoomReset()`: modificar l'escala de visualització.
  - `Copy(), Cut(), Paste(bool), Undo(), Redo()`: operacions bàsiques d'edició.
  - `SelectAll(), SelectLine(), SelectParagraph(), SelectWord(), GetObjectById(string)`: mètodes per a seleccionar fragments d'HTML.
  - `InsertHtml(string), AppendHtml(string)`: per a afegir codi HTML en la posició actual del cursor o al final de tot el contingut.
  - `Command(string)`: execució d'ordres, com ara:
    - `Search`: buscar una cadena dins del contingut HTML.
    - `InsertParagraph`: inserir un paràgraf.
    - `MakeLink`: inserir un enllaç.
    - `BoldOn`, `ItalicOn`, `SizeIncrease`, `AlignCenter`, `ParagraphStyleNormal`, `ParagraphStyleH1`, etc.: modificar l'estil del text.
    - `InsertTable11`, `TableInsertRowBefore`, `TableJoinCellLeft`, etc.: crear i manipular taules.
    - `CursorBod`, `CursorEod`: situar el cursor al principi/final del document.
- Esdeveniments:
  - `CursorMove`: en moure el cursor.
  - `LinkClicked`: en fer clic sobre un enllaç.

– `LoadDone`: en acabar la càrrega del document.
– `Submit`: en enviar les dades d'un formulari.

A continuació, tenim una aplicació d'exemple per a il·lustrar el funcionament bàsic de GtkHtml#. L'aplicació és un navegador de pàgines en HTML d'Internet que permet modificar-les i desar-les en disc, tal com mostra la figura 8.

Figura 8



La interfície consisteix en una caixa de text en què s'entra la URL que s'ha de carregar, uns botons per a desar el document en un arxiu local i per a modificar l'escala de visualització, i finalment l'àrea GtkHtml# en què es visualitza el document i es permet modificar-lo amb les tecles habituals d'edició. Aquesta interfície ha estat creada mitjançant Glade i es carrega a partir de l'arxiu XML que aquest programa de disseny genera.

El llistat següent correspon al codi de l'aplicació:

```
using System;
using System.Net;
using System.IO;
using Gtk;
using Glade;

namespace ExempleGtkHtml
{
  class ExemplGtkHtml
  {
    [Widget] Window windowPrincipal;
    [Widget] Button buttonDesar;
    [Widget] Button buttonApropar;
    [Widget] Button buttonAllunyar;
    [Widget] Button buttonCarrega;
    [Widget] Entry  entryURL;
    [Widget] ScrolledWindow scrolledwindowTreball;
```

```
HTML         html;
string       actualURL;
StreamWriter arxiuSortida;

static void Main (string[] args) { new ExempleGtkHtml(); }

ExempleGtkHtml()
{
  Application.Init ();

  Glade.XML gxml = new Glade.XML (null, "exemplegtkhtml.glade",
                                  "windowPrincipal", null);
  gxml.Autoconnect (this);

  windowPrincipal.DeleteEvent += new DeleteEventHandler(on_windowPrincipal_delete);

  html = new HTML ();
  html.LinkClicked += new LinkClickedHandler (on_html_linkclicked);
  scrolledwindowTreball.Add (html);

  LoadHtml("http://tornatmico.org");
  windowPrincipal.ShowAll();
  Application.Run ();
}

void on_windowPrincipal_delete (object obj, DeleteEventArgs args)
{ Application.Quit(); }

void on_entryURL_activate (object obj, EventArgs args)
{ on_buttonCarga_clicked (obj, args); }

void on_buttonAcercar_clicked (object obj, EventArgs args) { html.ZoomIn(); }

void on_buttonAlejar_clicked (object obj, EventArgs args) { html.ZoomOut(); }


void on_buttonCarga_clicked (object obj, EventArgs args)
{
  actualURL = entryURL.Text.Trim();
  LoadHtml (actualURL);
}

void on_html_linkclicked (object obj, LinkClickedArgs args)
{
  string nuevaURL;

  if (args.Url.StartsWith("http://"))
    novaURL = args.Url;
  else
    novaURL = actualURL + args.Url;

  try { LoadHtml (nuevaURL); } catch { }
  actualURL = novaURL;
}

void on_buttonDesar_clicked (object obj, EventArgs args)
{
  arxiuSortida = File.CreateText("arxiuSortida.html");
  html.Export("text/html", new HTMLSaveReceiverFn( DesaEnDisc ) );
  arxiuSortida.Close();
}

bool DesaEnDisc (IntPtr obj, string data)
{
  arxiuSortida.Write(data);
  return true;
}
```

```
    void LoadHtml (string URL)
    {
      HttpWebRequest web_request = (HttpWebRequest) WebRequest.Create (URL);
      HttpWebResponse web_response = (HttpWebResponse) web_request.GetResponse ();
      Stream stream = web_response.GetResponseStream ();
      byte [] buffer = new byte [8192];

      html.Editable = false;
      HTMLStream html_stream = html.Begin ();
      int count;

      while ((count = stream.Read (buffer, 0, 8192)) != 0){
           html_stream.Write (buffer, count);
      }
      html.End (html_stream, HTMLStreamStatus.Ok);

      html.Editable = true;
      html.InsertHtml("Arxiu editat amb el Navegador/Editor de Mono");
    }
  }
}
```

En primer lloc, podem observar com en el constructor de la classe es carrega dinàmicament la definició de la interfície i posteriorment s'hi afegeix l'objecte GtkHtml#. D'aquesta manera, es mostra com es pot aprofitar Glade per a tot el que proporcioni directament, i com es pot completar el resultat amb la construcció mitjançant codi dels elements d'interfície encara no suportats pel dissenyador.

El mateix constructor carrega una URL inicial usant el mètode `LoadHtml`, que obté la informació per mitjà d'un agent i la diposita en el *widget* GtkHtml mitjançant els mètodes `egin()`, `Write()` i `End()`. En acabar, activa la possibilitat que l'usuari pugui editar el contingut i insereix un fragment d'HTML inicial.

A partir d'aquí, l'usuari pot manipular el contingut de l'àrea GtkHtml amb les tecles habituals d'edició (cursors, inserció, supressió, ratolí, etc.) i modificar l'escala de visualització amb els esdeveniments `on_buttonApropar_clicked` / `on_buttonAllunyar_clicked`.

Per últim, l'usuari pot desar en disc el contingut modificat mitjançant el botó "Desar", que activa l'esdeveniment `on_buttonDesar_clicked`.

Aquesta aplicació senzilla mostra la base per a crear un navegador i/o editor lleuger d'HTML integrat en la nostra aplicació. Per a compilar-la caldrà indicar que estem usant els paquets GTK# i GtkHtml#, tal com s'indica a continuació:

```
mcs -pkg:gtk-sharp -pkg:gtkhtml-sharp -pkg:glade-sharp -
resource:ejemplogtkhtml.glade  ExempleGtkHtml.cs
```

Instal·lació de GtkHtml

Atès que GtkHtml# és solament una capa d'interconnexió amb GtkHtml, cal que tinguem instal·lat aquest darrer element. Normalment, el trobarem en el paquet libgtkhtml en una versió superior o igual a la 3.0.

5.2. Gecko#

El projecte Mozilla ha generat –entre altres– el motor de visualització de pàgines web Gecko. Aquest motor és capaç de visualitzar pàgines web basades en HTML o XML, inclòs CSS, imatges, JavaScript i components que requereixen aplicacions externes, com ara contingut Flash. Per a poder utilitzar-lo des del projecte Mono, hi ha una capa d'interconnexió que permet incorporar-lo a la nostra aplicació com un element gràfic més: es tracta de Gecko#. Aquest element gràfic interactua amb GtkEmbedMoz, submnistrat amb les aplicacions principals del projecte Mozilla.

Gecko# proporciona principalment l'element gràfic `WebControl`, del qual podem destacar els elements següents:

- Propietats:

– `Location`: adreça actual.
– `Title`: títol de la pàgina actual.

- Mètodes:

– `GoBack()`: accedir a la pàgina visitada anteriorment.
– `GoForward()`: accedir a la pàgina des de la qual s'havia retrocedit.
– `LoadUrl(string)`: carregar una nova pàgina des d'Internet.
– `Reload(int)`: recarregar la pàgina actual.
– `StopLoad()`: detenir la càrrega de la pàgina actual.

- Esdeveniments:

– `DomKeyPress`: en prémer una tecla en qualsevol part del document.

– `DomMouseClick`, `DomMouseDblClick`: en prémer el ratolí en qualsevol part del document.

– `LinkMsg`: en situar el ratolí o el focus del teclat en un enllaç (sense necessitat d'activar-lo).

– `LocChange`: en canviar l'adreça actual.

– `NetStart`, `NetStop`: en començar o acabar la càrrega del document actual.

– `TitleChange`: en canviar el títol de la pàgina actual.

A continuació, exposem una aplicació que mostra el funcionament bàsic de Gecko#. Es tracta d'un navegador bàsic amb les opcions de desplaçar-se per la història de navegació, i de recarregar o deternir la càrrega de la pàgina actual.

La figura 9 en mostra l'aparença, que està definida bàsicament mitjançant un arxiu XML usant Glade.

Figura 9

La interfície consta d'un divisió vertical per a situar en la part superior els botons de navegació, seguits de la barra d'adreça, i en la part inferior, una barra d'estat i de progrés. La part central de la finestra està ocupada per l'element gràfic `WebControl` de Gecko#, és a dir, la zona de visualització de les pàgines web.

El llistat de l'aplicació és el següent:

```
using System;
using Gtk;
using Gecko;
using Glade;

namespace ExempleGecko
{
  class ExempleGecko
  {
    [Widget] Window windowPrincipal;
    [Widget] Button bEnrere;
    [Widget] Button bEndavant;
    [Widget] Button bParar;
    [Widget] Button bRecarregar;
    [Widget] Button bCarregar;
    [Widget] Entry  actualURL;
    [Widget] HBox   hboxBottom;
    [Widget] VBox   vboxWebControl;

    WebControl  webControl;
    Statusbar   barraEstat;
```

```
    ProgressBar barraProgres;

    static void Main (string[] args)
    {
       new ExempleGecko ();
    }

    ExempleGecko ()
    {
      Application.Init ();

      Glade.XML gxml = new Glade.XML (null, "ExempleGecko.glade",
                                      "windowPrincipal", null);
      gxml.Autoconnect (this);

      webControl = new WebControl ("/tmp/csharp", "ExempleGecko");
      webControl.OpenUri += new OpenUriHandler(moz_openuri);
      webControl.LinkMsg += new EventHandler(moz_linkmsg);
      vboxWebControl.PackStart(webControl, true, true, 1);
      webControl.Visible = true;

      barraEstat = new Statusbar ();
      barraProgres = new ProgressBar ();

      hboxBottom.Add (barraEstat);
      hboxBottom.Add (barraProgres);

      windowPrincipal.ShowAll ();
      Application.Run ();
    }

    void LoadHtml      (string URL)                  { webControl.LoadUrl (URL); }

    void bCarregar_clk   (object obj, EventArgs args)
       { LoadHtml(actualURL.Text.Trim()); }

    void actualURL_act (object obj, EventArgs args)  { bCargar_clk (obj, args); }

    void bEnrere_clk    (object obj, EventArgs args)  { webControl.GoBack(); }

    void bParar_clk    (object obj, EventArgs args)  { webControl.StopLoad(); }

    void bEndavant_clk (object obj, EventArgs args)  { webControl.GoForward(); }

    void bRecarregar_clk (object obj, EventArgs args)  { webControl.Reload(0); }

    void moz_openuri   (object obj, OpenUriArgs args){ actualURL.Text = args.AURI; }

    void moz_linkmsg   (object obj, EventArgs args)
      { barraEstat.Pop (1);
         barraEstat.Push (1, webControl.LinkMessage);
      }

    void windowPrincipal_delete(object obj, DeleteEventArgs args)
      { Application.Quit(); }
  }
}
```

Com es pot observar, el mètode constructor genera l'estructura principal de la interfície carregant un arxiu XML de descripció amb el format Glade. A continuació, hi afegeix alguns elements gràfics de més, com el mateix `WebControl`. Per últim, es mostren tots els mètodes que respondran als esdeveniments de la interfície, que bàsicament fan crides a mètodes del component `WebControl`.

Per a compilar aplicacions usant Gecko#, necessitarem incloure'l com a paquet, tal com mostra l'exemple següent:

```
mcs -pkg:gecko-sharp -pkg:glade-sharp -
resource:ExempleGecko.glade ExempleGecko.cs
```

Instal·lació de GtkEmedMoz

Atès que Gecko# és solament una capa d'interconnexió amb GtkEmedMoz, cal tenir instal·lat aquest darrer element. Normalment el trobarem en el paquet Mozilla de la nostra distribució.

5.3. GtkSourceView#

Per últim, veurem un element gràfic útil per a crear editors de text, es tracta de GtkSourceView#. Altre cop es tracta d'una capa d'interconnexió per poder incloure des de Mono l'element `SourceView` de les biblioteques GTK. Aquest element ofereix les funcions bàsiques que es poden requerir en una àrea d'edició de codi de programació, i en realitat es compon de diverses classes, entre les quals destaquem les següents:

a) `SourceBuffer`:

- Propietats:
  - `Text`: el text mostrat.
  - `CheckBrackets`: si és cert, es ressalten els parèntesis inicial i final en passar per damunt d'un d'ells.
  - `Highlight`: si és cert, es ressalta la sintaxi del codi mostrat.
  - `Language`: indica el llenguatge del codi editat.
- Mètodes:
  - `ForwardSearch`, `BackwardSearch`: per a fer cerques en el text.
  - `CreateMaker`, `GetMaker`, `GetNextMaker`: per a seleccionar fragments del text.
  - `Undo`, `Redo`: per a desfer o refer les darreres accions.
- Esdeveniments:
  - `HighlightUpdated`: en modificar-se algun paràmetre del ressaltat de codi.
  - `MarkerUpdated`: en modificar-se la selecció del text.

b) `SourceView`:

- Propietats:
  - `AutoIndent`: per a activar la indentació automàtica del codi.
  - `InsertSpacesInsteadOfTabs`: per a usar espais en prémer la tecla del tabulador.
  - `Margin`: per a establir marges des dels límits del text fins al contorn de l'àrea d'edició.
  - `ShowLineNumbers`: per a mostrar els números de línia.
  - `TabsWidth`: per a marcar l'amplada dels tabuladors.
- Esdeveniments:
  - `Redo`, `Undo`: en activar-se els mètodes per a desfer o refer les darreres accions.

Per a exemplificar l'ús de GtkSourceView# s'ha implementat l'aplicació mínima que ofereix les funcions per a editar un arxiu de codi C#. A la figura 10 es mostra la interfície.

Figura 10

L'aplicació s'inicia amb l'àrea d'edició en blanc, amb la qual cosa és possible crear un arxiu nou i posteriorment polsar sobre el botó "Desar" per a emmagatzemar-lo en disc. També hi ha el botó "Obrir" per si es vol editar un arxiu ja existent. L'àrea d'edició permet la interacció mitjançant el teclat i el ratolí, amb les operacions bàsiques d'edició (supressió, selecció, portapapers, etc.).

El codi de l'aplicació és el següent:

```
using System;
using System.IO;
using Gtk;
using Glade;
using GtkSourceView;

class Editor
{
  [Widget] Gtk.Window     windowPrincipal;
  [Widget] ScrolledWindow scrolledwindow;
  [Widget] Button         bObrir;
  [Widget] Button         bDesar;
  SourceBuffer            sourceBuffer;
  SourceView              sourceView;

  string arxiu = "arxiu_exemple.cs";

  public Editor()
  {
    Application.Init();

   Glade.XML gxml = new Glade.XML(null, "ExempleGtkSourceView.glade", "windowPrincipal", null );
    gxml.Autoconnect( this );

    windowPrincipal.Resize(300, 500 );
```

```
    SourceLanguagesManager sl = new SourceLanguagesManager();
    sourceBuffer = new SourceBuffer(sl.GetLanguageFromMimeType("text/x-csharp"));
    sourceBuffer.Highlight = true;

    sourceView = new SourceView(sourceBuffer);
    sourceView.ShowLineNumbers = true;
    scrolledwindow.Add( sourceView );

    windowPrincipal.DeleteEvent += new DeleteEventHandler( Sortir );
    bObrir.Clicked              += new EventHandler( Obrir );
    bDesar.Clicked            += new EventHandler( Desar );


    windowPrincipal.ShowAll();
    Application.Run();
  }

  void Sortir(object o, DeleteEventArgs args) { Application.Quit(); }

  void Obrir (object o, EventArgs args) {
    FileSelection fs = new FileSelection ("Escull arxiu");
    fs.Run ();
    arxiu = fs.Filename;
    fs.Hide ();

    using (StreamReader sr = new StreamReader(arxiu))
    {
        sourceBuffer.Text = sr.ReadToEnd();
    }
  }

  void Desar( object o, EventArgs args )
  {
    using (StreamWriter sw = new StreamWriter(arxiu))
    {
        sw.Write(sourceBuffer.Text);
    }
  }

  public static void Main() { new Editor(); }
}
```

Igual que en els exemples anteriors, la interfície es construeix a partr de la descripció de l'arxiu XML de Glade i es complementa amb elements gràfics creats en el constructor de l'aplicació. Atesa la simplicitat de l'exemple, se subscriuen principalment els mètodes corresponents als dos botons. S'hi usen `StreamReaders` i `StreamWriters` per a treballar amb el sistema d'arxius.

Per a compilar una aplicació que inclogui aquest element gràfic caldrà indicar el paquet. Aquest paquet està disponible en la versió 2, així que haurem d'incloure tot l'entorn en aquesta versió, tal com es mostra a l'exemple:

```
mcs -pkg:gtk-sharp-2.0 -pkg:glade-sharp-2.0 -pkg:gtksourceview-
sharp-2.0 -linkresource:ExempleGtkSourceView.glade
ExempleGtkSourceView.cs
```

6. Gnome#

En apartats anteriors hem mostrat la manera de programar la interfície gràfica d'una aplicació usant GTK i l'eina de disseny Glade. Hem vist diversos elements gràfics de què podem disposar des de GTK i com els hem d'usar. De totes maneres, per a desenvolupar aplicacions podem aprofitar el projecte Gnome. Aquest projecte té com a objectius principals proporcionar un escriptori intuïtiu i fàcil d'usar, i un bon entorn de desenvolupament. En aquest sentit, a més dels elements GTK ja vistos, podem usar els elements bàsics que proporciona Gnome per a desenvolupar aplicacions.

En aquest apartat s'exposen diversos elements que proporciona el projecte Gnome amb què es poden desenvolupar més ràpidament les nostres aplicacions.

- **Elements gràfics**: elements per a inserir en les finestres d'aplicació. Es tracta d'elements complexos creats a partir d'elements gràfics bàsics de GTK.

  Exemple d'element gràfic

  Gnome proporciona, per exemple, un element gràfic per a escollir una font d'escriptura, juntament amb un estil i una mida, tot integrat dins d'un sol element gràfic.

- **Diàlegs**: finestres de diàleg estàndard o personalitzades creades a partir d'elements gràfics GTK o Gnome per facilitar la interacció amb l'usuari.

- **Impressió**: gestió del sistema d'impressió, amb el maneig de dispositius i l'existència dels diàlegs bàsics d'impressió.

- ***Canvas***: element gràfic per a inserir en les finestres d'aplicació en què l'aplicació pot dibuixar fàcilment elements geomètrics bàsics.

- **Assistents**: Gnome proporciona un mètode àgil per a crear agrupacions de diàlegs amb la finalitat de dur a terme tasques complexes.

6.1. Elements gràfics

Gnome proporciona elements gràfics per a inserir en les finestres d'aplicació creats a partir d'elements gràfics bàsics de GTK. D'aquesta manera, en motes ocasions podem usar-los en lloc d'haver de tornar-los a dissenyar a partir de GTK. A més, l'ús dels elements proporcionats per Gnome proporciona més coherència entre la interfície de la nostra aplicació i la resta d'aplicacions Gnome. Aquest darrer punt augmenta la usabilitat del nostre disseny.

Per a il·lustrar l'ús d'aquests elements, a continuació es mostra un exemple que n'usa diversos:

- **`FileEntry`**: per a seleccionar arxius. Permet examinar el sistema d'arxius mitjançant un diàleg estàndard de Gnome i al mateix temps ofereix una llista desplegable amb les darreres seleccions fetes.

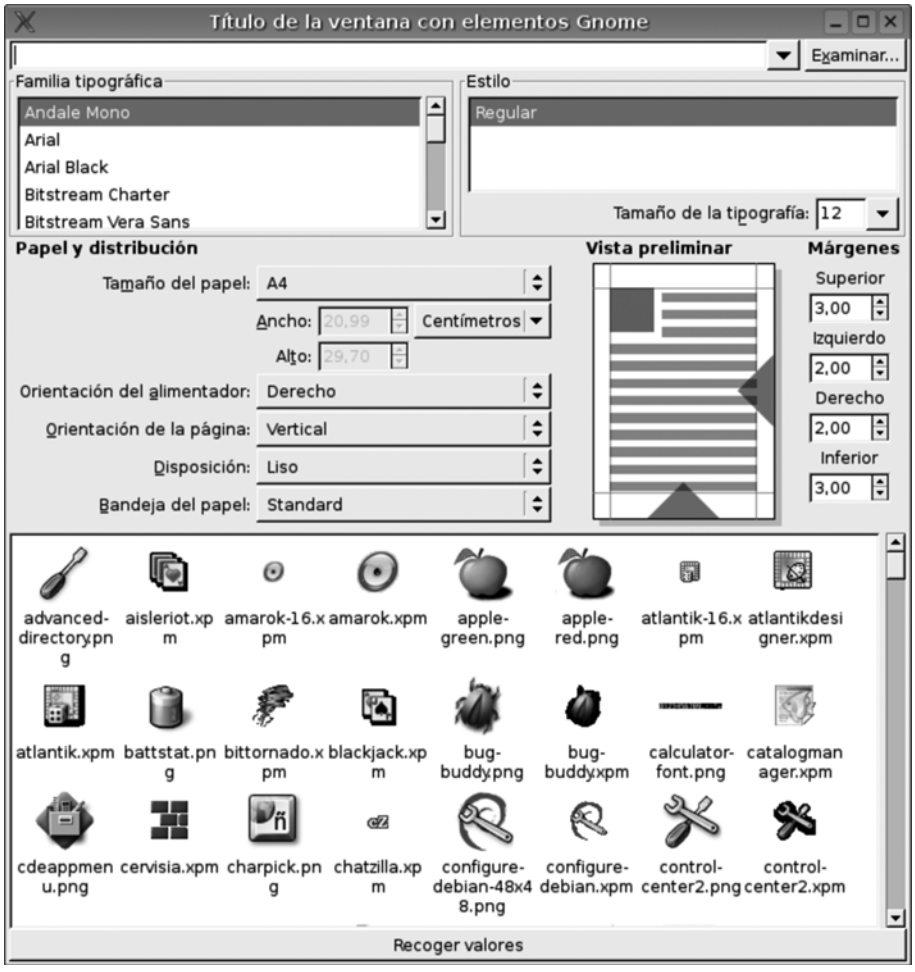
- **`PaperSelector`**: per a seleccionar el paper d'impressió juntament amb la disposició del contingut. Aquest element interactua directament amb el gestor d'impressió de Gnome.

- **`FontSelection`**: per a seleccionar fonts de lletra, tant la família com l'estil i la mida. Aquest element interactua directament amb el gestor de fonts de Gnome.

- **`IconSelection`**: per a seleccionar icones. Aquest element interactua directament amb el gestor d'icones de Gnome.

La figura 11 mostra la interfície de l'aplicació exemple.

Figura 11



S'hi poden veure tots els elements gràfics definits. En general, aquests elements també estan disponibles en les eines de disseny d'interfície com Glade,

encara que en el codi font d'aquesta aplicació es mostra com es poden crear des del mateix codi C#:

```
using System;
using Gtk;
using GtkSharp;
using Gnome;

class ExempleGnomeWidgets
{
  Program program;
  Gnome.FileEntry     selectorArxiu;
  Gnome.FontSelection selectorFont;
  Gnome.PaperSelector selectorPaper;
  Gnome.IconSelection selectorIcones;
  Button              buttonRecollir;

  static void Main(string[] args) { new ExempleGnomeWidgets(args);   }

  ExempleGnomeWidgets (string[] args)
  {
    program = new Program("ExempleGnome", "0.1", Gnome.Modules.UI , args);

    App app = new App("Elements", "Títol de la finestra amb elements Gnome");
    app.DeleteEvent += new DeleteEventHandler (on_app_delete);

    VBox vb = new VBox();

    selectorArxiu = new Gnome.FileEntry ("historial", "título");
    selectorFont = new Gnome.FontSelection();
    selectorPaper = new Gnome.PaperSelector( PrintConfig.Default() );
    selectorIcones  = new Gnome.IconSelection();
    buttonRecollir = new Button ("Recollir valors");

    selectorIcones.AddDefaults();
    selectorIcones.ShowIcons();

    buttonRecoger.Clicked += new EventHandler (on_buttonRecollir_clicked);

    vb.PackStart( selectorArxiu );
    vb.PackStart( selectorFont );
    vb.PackStart( selectorPaper );
    vb.PackStart( selectorIcones  );
    vb.PackStart( buttonRecollir   );

    app.Contents = vb;
    app.ShowAll();

    program.Run();
  }
        void on_buttonRecoger_clicked(object o, EventArgs args)
  {
    Gnome.Font  font = selectorFont.Font;

    Console.WriteLine("Arxiu: " + selectorArxiu.GetFullPath(true) );
    Console.WriteLine("Font: "  + font.FamilyName + " " + font.Size);
    // see http://cvs.gnome.org/viewcvs/libgnomeprint/libgnomeprint/gnome-print-
config.h?rev=1.29&view=markup
    Console.WriteLine("Papel: "
         + PrintConfig.Default().Get("Settings.Output.Media.PhysicalSize") +
     "(" + PrintConfig.Default().Get("Settings.Output.Media.PhysicalOrientation") + ")");
    Console.WriteLine("Icona: "   + selectorIcones.GetIcon(true) );
  }

  private void on_app_delete (object o, DeleteEventArgs args) { program.Quit (); }
}
```

La creació dels elements gràfics es fa en el mètode principal i consisteix a crear els objectes passant determinats paràmetres de personalització als seus mètodes constructors. Destaquem el cas constructor del `PaperSelector`, al qual és necessari indicar quina de les configuracions es vol usar, que en l'exemple és la que el sistema tingui definida per defecte. D'altra banda, cada element gràfic té els seus camps i mètodes propis en què s'emmagatzema la informació recollida. El mètode `on_buttonRecollir_clicked` exemplifica com es pot recollir l'esmentada informació.

6.2. Diàlegs

Els diàlegs són finestres o petits conjunts de finestres pensats per a dialogar amb l'usuari i obtenir dades concretes. Gnome proporciona diversos diàlegs estàndard que habitualment s'usen en les aplicacions. El seu ús incrementa la usabilitat de l'aplicació, ja que l'usuari els troba familiars.

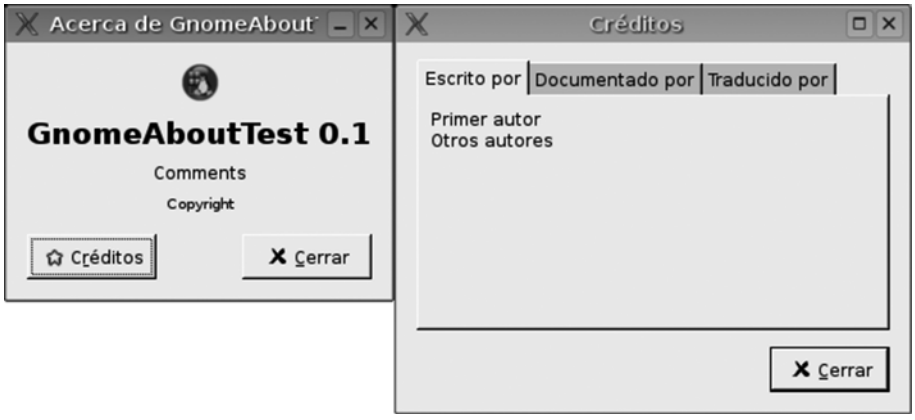
Per a exemplificar-ne l'ús, s'ha creat una petita aplicació que mostra una finestra principal (figura 12) que ofereix diferents botons perquè l'usuari pugui cridar cada un dels diàlegs que es mostren.

Figura 12

Els diàlegs són els següents:

- **Al voltant de** (figura 13): diàleg que mostra els crèdits d'autoria de l'aplicació.

Figura 13



- **Autenticació** (figura 14): diàleg per a demanar l'identificador i la contrasenya a l'usuari. Permet establir valors per als dos camps, així com identificació anònima.

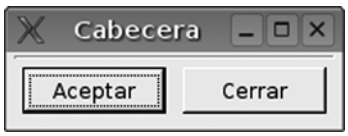
Figura 14

- **Impressió** (figura 15): diàleg perquè l'usuari pugui especificar determinats paràmetres d'impressió.

Figura 15

- **Diàleg personalitzat** (figura 16): a part dels diàlegs estàndard, també és possible crear el nostre propi diàleg personalitzat amb els elements que ens interessin.

Figura 16



El codi necessari per a crear els diàlegs que s'han mostrat anteriorment és el següent:

```
using System;
using GTK;
using GTKSharp;
using Gnome;

class ExempleGnomeDialegs
{
  Program program;

  static void Main(string[] args) { new ExempleGnomeDialegs(args);   }

  ExempleGnomeDialegs (string[] args)
  {
    program = new Program("ExempleGnome", "0.1", Gnome.Modules.UI , args);
```

```
  App app = new App("ExempleGnome", "Exemple Gnome");
  app.DeleteEvent += new DeleteEventHandler (on_app_delete);

  HBox hb = new HBox();

  Button buttonAlvoltantDe       = new Button ( GTK.Stock.DialogQuestion );
  Button buttonPassword       = new Button ( GTKGTK.Stock.Network );
  Button buttonPrintConfig    = new Button ( GTK.Stock.Print );
  Button buttonDialegPropi = new Button ( GTK.Stock.Properties );
  buttonAlvoltantDe.Clicked      += new EventHandler (on_buttonAcercaDe_clicked);
  buttonPassword.Clicked      += new EventHandler (on_buttonPassword_clicked);
  buttonPrintConfig.Clicked   += new EventHandler (on_buttonPrintConfig_clicked);
  buttonDialegPropi.Clicked += new EventHandler (on_buttonDialogoPropio_clicked);

  hb.PackStart(buttonAlvoltantDe);
  hb.PackStart(buttonPassword);
  hb.PackStart(buttonPrintConfig);
  hb.PackStart(buttonDialegPropi);

  app.Contents = hb;

  app.ShowAll();
  program.Run();
}

private void on_buttonAlvoltantDe_clicked (object obj, EventArgs args)
{
  string[] authors = {"Primer autor", "Altres autors"};
  string[] documenters = {"Encarregat de documentació"};
  Gdk.Pixbuf pixbuf = new Gdk.Pixbuf ("ExempleIcona.png");

  About dialegAlvoltantDe    = new Gnome.About ("GnomeAboutTest", "0.1",
                       "Copyright", "Comments",
                       authors, documenters, "translator", pixbuf);
  dialogoAlvoltantDe.Response += new ResponseHandler (OnResponse);
  dialogoAlvoltantDe.Run ();
}

private void on_buttonPassword_clicked (object obj, EventArgs args)
{
  PasswordDialog dialegPwd = new Gnome.PasswordDialog("Títol", "Missatge",
                                "usuari inicial",
                                "clau inicial", false);
  dialogoPwd.Response      += new ResponseHandler (OnResponse);
  dialogoPwd.Run ();
  Console.WriteLine(dialogoPwd.Username +":"+ dialogoPwd.Password);
  dialogoPwd.Destroy ();
}

private void on_buttonPrintConfig_clicked (object obj, EventArgs args)
{
  PrintConfig conf = PrintConfig.Default();
  PrintConfigDialog dialegPrtCnf = new Gnome.PrintConfigDialog( conf );
  dialogoPrtCnf.Response         += new ResponseHandler (OnResponse);
  dialogoPrtCnf.Run ();
  Console.WriteLine("Duplex: " + conf.Get("Settings.Output.Job.Duplex"));
  dialogoPrtCnf.Destroy ();
}

private void OnResponse(object o, ResponseArgs args) { Console.WriteLine (args.ResponseId); }
private void on_buttonDialegPropi_clicked (object obj, EventArgs args)
{
  GTK.Window    win    = new GTK.Window ("Test");
  DialegPropi dialog = new DialegPropi(win, GTK.DialogFlags.DestroyWithParent);
  dialog.Run ();
  dialog.Destroy ();
```

```
  }

  public class DialegPropi : Dialog
  {
    public DialegPropi(GTK.Window w, DialogFlags f) : base("Títol", w, f)
    {
      this.Modal = true;
      this.VBox.PackStart( new Label("Etiqueta") );
      this.AddButton("Acceptar", ResponseType.Accept);
      this.AddButton("Tancar",  ResponseType.Close);
    }

    protected override void OnResponse (ResponseType response_id){
      Console.WriteLine(response_id);
    }
  }
  private void on_app_delete (object o, DeleteEventArgs args) { program.Quit (); }
}
```

Abans de fixar-nos en els detalls relacionats amb els diàlegs, cal destacar com es creen els botons de la finestra principal. En el mètode de creació de l'aplicació es pot observar com els botons que s'afegeixen es creen amb icones estàndard que hi ha en el sistema. Per a referenciar-los, s'usa la classe `Stock` de GTK.

Els diàlegs es creen en els mètodes que se subscriuen a l'esdeveniment `Clicked` de cada botó. L'estructura d'aquests mètodes subscrits és la mateixa per a tots:

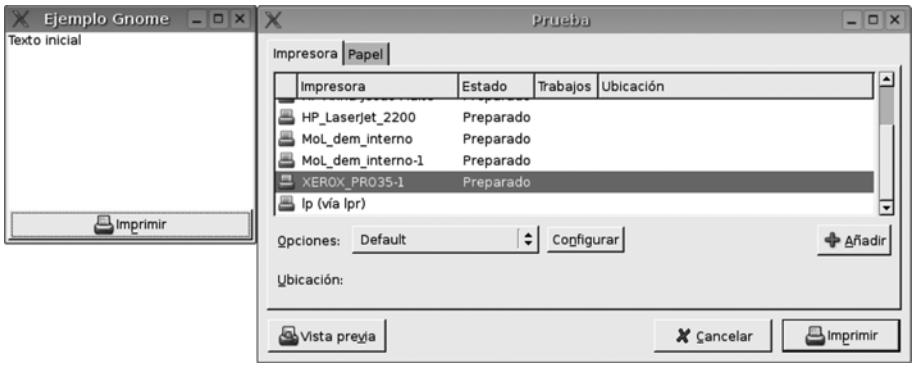
1) creació del diàleg,
2) subscripció a l'esdeveniment `Response`, que es produeix en el moment que l'usuari finalitza el diàleg,
3) execució del diàleg,
4) impressió de les dades recollides pel diàleg i
5) destrucció del diàleg.

Per últim, en el codi trobem la creació del diàleg personalitzat propi en forma de classe imbricada (`DialegPropi`) que deriva de la classe `Dialog`. En el constructor de la classe derivada podem afegir tots els elements que vulguem que tingui el nou diàleg. D'altra banda, la manera de vincular un mètode a l'esdeveniment `Response` no és usant un delegat, com en els casos anteriors, sinó sobrecarregar el mètode `OnResponse`.

6.3. Impressió

El projecte Gnome, amb l'objectiu de crear un entorn d'escriptori complet, inclou prestacions com la gestió del sistema d'impressió. Una manera senzilla d'oferir a l'usuari control sobre aquest sistema de Gnome és usar el diàleg d'impressió que es mostra a la figura 17.

Figura 17



En la imatge es pot observar a l'esquerra la finestra de l'aplicació d'exemple, en què l'usuari pot escriure el text que vol imprimir, i a la dreta el diàleg d'impressió estàndard de Gnome. En aquest diàleg, l'usuari pot escollir quina impressora vol utilitzar i la configuració del paper. A més, des d'aquest diàleg és possible accedir al diàleg de configuració de la impressora o bé a la finestra de previsualització d'impressió.

El codi següent mostra com s'ha creat aquesta aplicació exemple:

```
using System;
using GTK;
using Gnome;

class ExempleGnomeImprimir
{
  Program program;
  TextView areaText;

  static void Main(string[] args) { new ExempleGnomeImprimir(args);   }

  ExempleGnomeImprimir (string[] args)
  {
    program = new Program("ExempleGnomeImprimir", "0.1", Gnome.Modules.UI , args);

    App app = new App("ExempleGnomeImprimir", "Exemple Gnome");
    app.DeleteEvent += new DeleteEventHandler (on_app_delete);

    VBox vb                 = new VBox (false, 0);
    areaText = new TextView ();
    Button buttonImprimir   = new Button (GTK.Stock.Print);

    areaText.Buffer.Text   = "Text inicial";
    buttonImprimir.Clicked += new EventHandler (on_buttonImprimir_clicked);

    vb.PackStart (areaText, true, true, 0);
    vb.PackStart (buttonImprimir, false, true, 0);

    app.Contents = vb;

    app.ShowAll();
    program.Run();
  }

  void ComponerPagina (PrintContext ContextoImp)
  {
    ContextoImp.BeginPage("Demostració");
    ContextoImp.MoveTo(1, 700);
```

```
    ContextoImp.Show(areaText.Buffer.Text);
    ContextoImp.ShowPage();
  }

  void on_buttonImprimir_clicked (object o, EventArgs args)
  {
    PrintJob    treball = new PrintJob (PrintConfig.Default ());
    PrintDialog dialeg = new PrintDialog (treball, "Prova", 0);
    int         resposta = dialeg.Run ();
    Console.WriteLine ("Resposta: " + resposta);

    if (resposta == (int) PrintButtons.Cancel) {
      Console.WriteLine("Impresió cancelada");
      dialeg.Hide (); dialeg.Dispose (); return;
    }
    PrintContext ctx = treball.Context;
    ComponerPagina(ctx);

    trabajo.Close();

    switch (resposta) {
      case (int) PrintButtons.Print:   treball.Print (); break;
      case (int) PrintButtons.Preview:
        new PrintJobPreview(treball, "Prova").Show();
        break;
    }

    dialeg.Hide (); dialeg.Dispose ();
  }

  void on_app_delete (object o, DeleteEventArgs args) { Application.Quit (); }
}
```

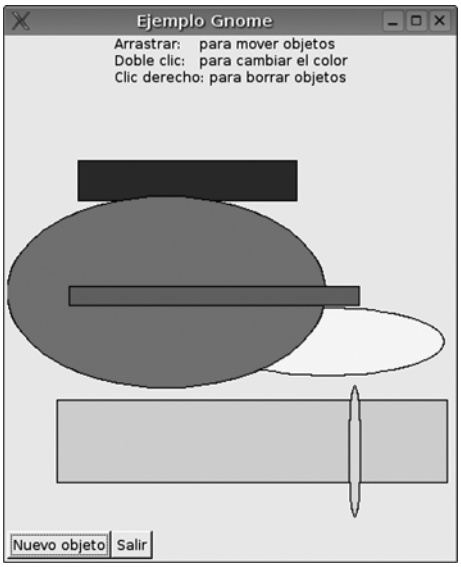
Inicialment es crea la finestra principal de l'aplicació en què s'insereix una àrea de text i el botó d'impressió. En aquest botó se subscriu el mètode `on_buttonImprimir_clicked` a l'esdeveniment `Clicked`, de tal manera que, quan l'usuari el prem, es crea un treball d'impressió (`PrintJob`) a partir de la configuració per defecte (`PrintConfig.Default`). A continuació, es crea el diàleg d'impressió (`PrintDialog`) i s'executa.

El codi que retorna el diàleg (`resposta`) indica quina acció ha iniciat l'usuari. Si l'usuari ha polsat el botó "Cancel·lar", simplement oculta el diàleg i després s'allibera. En cas contrari, s'omple el contingut del treball d'impressió mitjançant el mètode `CompdrePagina` a partir de l'àrea de text. En cas que l'usuari hagi polsat el botó "Imprimir", s'imprimeix directament el treball creat. Alternativament, si el botó que s'ha polsat ha estat "Vista prèvia", es crea una finestra de previsualització d'impressió.

6.4. *Canvas*

En cas que vulguem disposar d'una àrea de dibuix en alguna de les finestres de l'aplicació, podem usar l'esdeveniment `Canvas` de Gnome. Aquest esdeveniment permet dibuixar fàcilment objectes geomètrics bàsics i manipular-ne les propietats. A continuació es mostra el codi de l'aplicació d'exemple que té la interfície que podeu veure a la figura 18.

Figura 18



Es tracta d'una finestra principal en què un conjunt d'etiquetes de text mostren la descripció de les ordres bàsiques de l'aplicació en la part superior de la finestra. En la part central hi ha l'element gràfic `Canvas`, sobre el qual l'usuari pot dibuixar, i en la part inferior, un botó per a crear objectes nous i un altre per a sortir de l'aplicació.

El codi que produeix i gestiona aquesta aplicació d'exemple és el següent:

```
using Gnome;
using GTK;
using Gdk;
using System;

public class ExempleGnomeCanvas {
  Program program;
  private double ini_x = 0.0, ini_y = 0.0;
  private Canvas canvas;
  private Random random = new Random ();
  static void Main(string[] args) { new ExempleGnomeCanvas(args);   }

  ExempleGnomeCanvas (string[] args)
  {
    program = new Program("ExempleGnomeCanvas", "0.1", Gnome.Modules.UI , args);

    App app = new App("ExempleGnomeCanvas", "Exemple Gnome");
    app.DeleteEvent += new DeleteEventHandler (on_app_delete);

    VBox vb = new VBox (false, 0);
    vb.PackStart (new Label ("Arrossegar:    per a moure objetes\n"   +
             "Doble clic:   per a canviar el color\n" +
             "Clic dret: per a esborrar objects"),
        false, false, 0);

    canvas = new Canvas ();
    canvas.SetSizeRequest (400, 400);
    canvas.SetScrollRegion (0.0, 0.0, 400, 400);

    vb.PackStart (canvas, false, false, 0);

    HBox hb = new HBox (false, 0);
```

```
  Button buttonNou = new Button ("Objecte nou");
  Button buttonSortir = new Button ("Sortir");

  buttonNou.Clicked += new EventHandler (ObjecteNou);
  buttonSortir.Clicked += new EventHandler (Sortir);

  vb.PackStart (hb, false, false, 0);
  hb.PackStart (buttonNou, false, false, 0);
  hb.PackStart (buttonSortir, false, false, 0);

  app.Contents = vb;
  app.ShowAll();
  program.Run();
}

void ObjecteNou (object obj, EventArgs args)
{
  double x1 = random.Next (400), y1 = random.Next (400);
  double x2 = random.Next (400), y2 = random.Next (400);
  double aux;

  if (x1 > x2)        { aux = x1; x1 = x2; x2 = aux; }
  if (y1 > y2)        { aux = y1; y1 = y2; y2 = aux; }
  if ((x2 - x1) < 10) { x2 += 10; }
  if ((y2 - y1) < 10) { y2 += 10; }
  CanvasRE item = null;

  if (random.Next (2) > 0)
    item = new CanvasRect (canvas.Root());
  else
    item = new CanvasEllipse (canvas.Root());
  item.X1 = x1; item.Y1 = y1;
  item.X2 = x2; item.Y2 = y2;
  item.FillColor    = "white";
  item.OutlineColor = "black";
  item.WidthUnits   = 1.0;

  item.CanvasEvent += new Gnome.CanvasEventHandler (on_item_event);
}

void on_item_event (object obj, Gnome.CanvasEventArgs args)
{
  EventButton esdeveniment = new EventButton (args.Event.Handle);
  CanvasRE item      = (CanvasRE) obj;
  args.RetVal        = true;

  switch (evento.Type) {
    case EventType.TwoButtonPress: CambiarColor(item);    return;
    case EventType.EnterNotify:    item.WidthUnits = 3.0; return;
    case EventType.LeaveNotify:    item.WidthUnits = 1.0; return;
    case EventType.ButtonPress:
      switch (esdeveniment.Button) {
      case 1: ini_x = evento.X; ini_y = evento.Y;   return;
      case 3: item.Destroy();                       return;
      }
      break;
    case EventType.MotionNotify:
      Gdk.ModifierType estat = (Gdk.ModifierType) esdeveniment.State;
      if ((estat & Gdk.ModifierType.Button1Mask) != 0) {
        double fin_x = esdeveniment.X, fi_y = esdeveniment.Y;
        item.Move (fin_x - ini_x, fin_y - ini_y);
        ini_x = fin_x; ini_y = fin_y;
        return;
      }
      break;
```

```
    }

    args.RetVal = false;
    return;
  }

  void CanviarColor (CanvasRE item)
  {
    string[] colors = new string[] {"red", "yellow", "green", "cyan", "blue", "magenta"};
    item.FillColor = colors[random.Next(colors.Length)];
  }

  void Sortir (object obj, EventArgs args)       { Application.Quit (); }
  void on_app_delete (object obj, DeleteEventArgs args) { Application.Quit (); }
}
```

En primer lloc, es pot observar que el mètode constructor crea els elements comentats en la descripció de la interfície. Posteriorment, hi ha el mètode `ObjecteNou`, que ha subscrit el mètode `Clicked` del botó per a afegir objectes geomètrics nous a l'àrea de dibuix. L'esmentat mètode s'encarrega d'escollir unes coordenades aleatòries i una forma geomètrica a l'atzar entre rectangles i el·lipsis. En crear un ítem d'alguna d'aquestes dues formes geomètriques, es passa al seu constructor la referència del `Canvas` a què s'ha d'afegir. Al mateix temps, s'hi subscriu el mètode `on_item_event` perquè s'executi en qualsevol esdeveniment que es produeixi sobre l'esmentada forma geomètrica. Aquest mètode analitza quin esdeveniment s'ha produït i, en funció de quin sigui, crea la resposta:

- **tipo `TwoButtonPress`**: crida el mètode per a canviar-ne el color (`CanviarColor`).

- **tipo `EnterNotify` /`LeaveNotify`**: modifica la mida de la vora quan el ratolí està damunt de l'ítem per a indicar que està seleccionat.

- **tipo `ButtonPress`**: si es tracta del tercer botó, destrueix l'ítem. Si, en canvi, es tracta del primer botó, es memoritza la posició actual del punter, com a referència inicial del moviment de l'ítem.

- **tipo `MotionNotify`**: si el moviment s'ha produït amb el primer botó que s'ha polsat, es mou l'ítem seguint la relació entre les coordenades actuals del ratolí i les emmagatzemades anteriorment en l'esdeveniment tipus `ButtonPress`.

Com es pot veure en el codi d'exemple, crear i manipular objectes geomètrics amb la classe `Canvas` de Gnome és senzill.

6.5. Assistents

Per últim, en aquest subapartat veurem l'ús d'assistents de Gnome. Es tracta d'un mètode àgil per a crear agrupacions de diàlegs amb la finalitat de dur a terme tasques complexes. L'esquema bàsic és un pas inicial informatiu, una seqüència de passos senzills en què es va recollint la informació necessària i un pas final en què es demana confirmació per a dur a terme les accions oportunes en funció de la informació recollida.

Per a veure com es creen els assistents, s'ha creat l'exemple següent, que té la interfície que podeu veure a la figura 19.

Figura 19

Consisteix en una finestra principal que ofereix el botó mitjançant el qual s'obre el diàleg. El diàleg simplement consta dels tres passos mínims, i recull un nom entrat per l'usuari, per acabar mostrant-lo a la consola de text. A continuació es mostra el codi que el genera:

```
using System;
using GTK;
using GTKSharp;
using Gnome;

class ExempleGnomeAssistent
{
  Program program;

  static void Main(string[] args) { new ExempleGnomeAssistent(args);   }

  ExempleGnomeAssistent (string[] args)
  {
    program = new Program("ExempleGnomeAssistent", "0.1", Gnome.Modules.UI , args);

    App app = new App("ExempleGnomeAssistent", "Exemple Gnome");
    app.DeleteEvent += new DeleteEventHandler (on_app_delete);

    HBox hb = new HBox();

    Button buttonAssistent = new Button ( GTK.Stock.Preferences );
    buttonAssistent.Clicked += new EventHandler (on_buttonAsistente_clicked);
    hb.PackStart(buttonAssistent);

    app.Contents = hb;

    app.ShowAll();
    program.Run();
  }
```

```
  private void on_buttonAssistent_clicked (object obj, EventArgs args)
  {
    Druid druid = new Druid();

    DruidPageEdge pas1 = new DruidPageEdge(EdgePosition.Start, true,
        "Títol de l'assistent",
        "Descripció de l'assistent", null, null, null);

    DruidPageStandard pas2 = new DruidPageStandard("Pas 1",null,null);

    GTK.Entry nom = new GTK.Entry();
    paso2.AppendItem("_Nom:", nom, "descripció de l'ítem");

    DruidPageEdge pas3 = new DruidPageEdge(EdgePosition.Finish, true,
                    "Títol de la última pàgina",
                    "Descripció de l'última pàgina", null, null, null);

    druid.AppendPage(pas1);
    druid.AppendPage(pas2);
    druid.AppendPage(pas3);

    App app = new App("Assistent", "Títol de la finestra");

    pas3.FinishClicked += delegate{
      app.Destroy();
      Console.WriteLine("Dades recollides: {0}", nom.Text);
    };

    pas1.CancelClicked += delegate { app.Destroy(); };
    pas2.CancelClicked += delegate { app.Destroy(); };
    pas3.CancelClicked += delegate { app.Destroy(); };

    app.Contents = druid;
    app.ShowAll();
  }

  private void on_app_delete (object o, DeleteEventArgs args) { program.Quit (); }
}
```

El mètode constructor de l'aplicació simplement crea la finestra principal amb el botó i l'enllaça al mètode `on_buttonAssistent_clicked`. Aquest mètode és l'encarregat de crear l'assistent (`Druid`) amb la pàgina d'informació inicial (`DruidPageEdge` amb `EdgePosition.Start`), la pàgina de recollida d'informació intermèdia (`DruidPageStandard`) i la pàgina de confirmació final (`DruidPageEdge` amb `EdgePosition.Finish`).

A diferència dels diàlegs previs, per posar en marxa l'assistent necessitem crear una finestra nova on puguem inserir-lo com a contingut i mostrar la finestra. Finalment, en l'exemple se subscriuen tots els esdeveniments, des de prémer els botons de cancel·lació en qualsevol dels passos, fins a mètodes anònims que destrueixen l'assistent.

L'esdeveniment que es produeix si l'usuari arriba a l'últim pas de l'assistent i confirma l'operació és `FinishClicked`. En l'exemple, aquest mètode simplement mostra el nom a través de la consola de text.

7. XML

Extensible Markup Language és avui dia el format per excel·lència per a la transacció d'arxius de text. L'ús en la web com a moneda d'intercanvi i la creació d'un conjunt d'estàndards per a la gestió i transformació d'arxius han fet d'aquesta arquitectura la més usada en la programació.

Mono suporta la majoria dels estàndards de W3C al voltant d'XML: XSLT, Xpath, XML Schema, Relax NG i Seriació en XML dels objectes. Aquí descriurem breument com es pot utilitzar aquesta tecnologia.

7.1. Llegir i escriure XML

Vegem com podem llegir i escriure un arxiu de disc amb una estructura XML. Per a fer-ho, necessitem la biblioteca de `System.Xml`, que conté les classes `XmlReader` i `XmlWriter`.

Escriure en consola:

```
using System.Xml;

....

// Definim quin és l'esquema que s'utilitzarà en l'XML
private const string RDF = "http://www.w3.org/1999/02/22-rdf-syntax-ns";

XmlTextWriter writer = new XmlTextWriter(Console.Out); // escrivim a la consola
writer.Formatting = Formatting.Indented; // escrivim indentant
writer.Indentation = 2; // espais en la indentació
writer.IdentChar = ' '; // caràcter per a la indentació
writer.WriteStartDocument(true); // comencem a escriure el document

// Definim el primer element RDF dins de l'esquema RDF amb l'atribut xmlns
// Això crearà:
// <rdf:RDF xmlns="http://purl.org/rss/1.0/"
//                xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns">
writer.WriteStartElement("rdf","RDF",RDF); // comprova que dins d'RDF
hi ha un element rdf
writer.WriteAttributeString("xmlns","http://purl.org/rss/1.0/");
```

```
// Definim un element nou dins del pare anterior:
// <title>prova</title>
writer.WriteElementString("title","prova");

// Creem un fill amb un element li
// <rdf:li rdf:resource="http://www.tornatmico.org"/>
writer.WriteStartElement("li",RDF);
writer.WriteAttributeString("resource",RDF,"http://www.tornatmico.org");
writer.WriteEndElement();

// Tanquem un element
writer.WriteEndElement();

...
```

Per a poder llegir d'un arxiu:

```
using System.XML;

...

XmlTextReader reader = new XmlTextReader("arxiu.xml");
while (reader.Read()) {
  Console.WriteLine("Tipus de node={0}, Nom={1}, Valor=\"{2\"",
                    reader.NodeType, reader.Name, reader.Value);
}
reader.Close();
```

7.2. Navegar i manipular XML en memòria

Les funcions anteriors serveixen per a poder treballar d'una manera seqüencial amb els arxius XML. Els podeu llegir i escriure, però sense tenir constància de tota l'estructura a la vegada. Quan s'escriu, podeu treballar en el punt en què sou, i quan llegiu no podeu navegar per l'arxiu, solament llegir-lo seqüencialment. Per a poder treballar en memòria necessitem l'ús d'un arbre DOM (Document Object Model). Aquesta estructura ens permet mapar un arxiu XML en memòria per poder treballar-hi i navegar de manera no seqüencial. Aquesta manera d'usar els XML va ser desenvolupada per W3C i implementada en tots els llenguatges. Aquests és el sistema que usen els navegadors per a representar internament el codi de les pàgines web XHTML.

Per a poder llegir un arxiu en un document DOM:

```
using System
using System.Xml;
```

```
public class ReadDocument {
  public static void Main (string[] args) {
    string filename = args[0];
    XmlDocument document = new XmlDocument();
    document.Load(fileame);
  }
}
```

Hi ha dos modes de recórrer un arbre XML: en profunditat, seguint un ordre seqüencial, o amb un llenguatge anomenat Xquery que ens permet escollir una part de l'arbre. Per a poder buscar en un arbre DOM:

```
 XmlDocument document = new XmlDocument();
  document.Load(filename);
  // Creem un conjunt de nodes que responen a una xpath query
  XmlNodeList nodes = document.SelectNodes(pregunta);
  foreach (XmlNode node in nodes) {
    ...
  }
```

El problema d'usar aquest mode és que carreguem tot el document en memòria, cosa que en treballar amb documents grans pot suposar una càrrega externa. Per a això serveix el tipus `XpathDocument`:

```
// Creem un document amb referència a xpath, aquest document no es
carrega en memòria
XPathDocument document = new XpathDocument(filename);
// Creem un navegador d'aquest document
XPathNavigator navigator = document.CreateNavigator();
// Creem un conjunt de nodes que responen a una xpath query
XpathNodeIterator iterator = navigator.Select(query);
while (iterator.MoveNext()) {
  ...
}
```

En crear un nou element podem afegir atributs i decidir a quin punt de l'arbre XML volem vincular-lo.

```
 // Creem un element
  XmlElement element =
document.CreateElement("prova","contingut");
  // Creem un atribut
  XmlAttribute attribute =
document.CreateAttribute("info","informació");
  // Afegim un atribut a l'element
  element.Attributes.Append(attribute);
  // Afegim l'element a l'arrel del document
  document.DocumentElement.AppendChild(element);
```

7.3. Transformar amb XML

Podem escollir transformar un document XML amb les classes d'XSLT. En el moment de dur-ho a terme podem triar d'usar la biblioteca nadiua libxslt, per a la qual cosa utilitzarem la variable d'entorn `MONO_UNMANAGED_XSLT`. El fet d'usar la biblioteca nadiua ens obliga a estar en un entorn `no managed`, de manera que el `garbage collector` no actua en els objectes XSLT. Per a poder transformar un document XML:

```
try {
  XslTransform transform = new XslTransform();
  transform.Load(stylesheet);
  transform.Transform(source, target, null);
} catch ( Exception e ) {
  Console.Error.WriteLine(e);
}
```

7.4. Seriar

En Mono es pot seriar un objecte a XML de dues maneres diferents, una amb el *runtime* `serialization`, que ens permet generar d'un objecte un XML amb una sintaxi arbitrària o un format binari, i una altra amb XML `serialization`, que ens permet generar SOAP o un XML arbitrari.

Per a poder seriar un objecte, en el moment de crear-lo hem d'indicar que serà un objecte seriable, de manera que en definir les diferents variables puguem decidir si serà un `xmlelement` o un `attribute`. Vegem-ne un exemple:

```
using System;
using System.Xml;
using System.Xml.Serialization;

// Class main per a crear l'objecte i seriar-lo
public class Persona {
  public static void Main (string[] args) {

    // Creem l'objecte serialitzable
    Persona obj = new Persona();

    // Creem un serialitzador per apoder processar l'objecte
    XmlSerializer serialitzador = new
XmlSerializer(obj.GetType());
    // Mostrem per consola l'XML tornat
    serializer.Serialize(Console.Out, obj);
  }
}

// Escollim que una classe pugui ser serialitzable
[Serialitzable]
public class Persona {
  public string Nombre = "Jesus Mariano Perez";
  // Canviem el nom del tag que contindrà la variable
  [XmlElement("VariableXML")]
```

```
  [XmlElement("VariableXML")]
  public int variable = new Random().Next();

  // Aquesta variable es mostrarà com un atribut en el tag de
l'objecte
  [XmlAttribute]
  public DateTime DataActual = DateTime.Now;

  // Creem un array amb dues adreces per a mostrar dos subelements
  public Adreça [] lamevaadreça =
    new Adreça [] {
      new Adreça(), new Adreça() };
}

[Serialitzable]
public class Adreça {

 [XmlAttribute]
  public string Carrer = "carrer Pepito";

}
```

La sortida seria:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<persona DataActual="2006-04-04T10:01:48.245630-05:00">
  <VariableXML>41231253123</VariableXML>
  <Nom>Jesus Mariano Perez</Nom>
  <lamevaadreça>
    <adreça Carrer="carrer pepito"/>
    <adreça Carrer="carrer pepito"/>

  </lamevaadreça>
</persona>
```

7.5. Restringir XML

Per a poder validar un XML podem carregar un XML Schema i delegar a una funció l'execució de codi en cas d'error de validació. Per a usar-lo:

```
...
// Carreguem un arxiu per a processar-lo
XmlTextReader reader = new XmlTextReader(xmlfile);

// Creem un validador en el reader
xmlValidatingReader validator = new XmlValidatingReader(reader);

// Volem validar amb XMLSchema
validator.ValidationType=ValidationType.Schema;

// Carreguem l'arxiu de l'esquema
validator.Schemas.Add(string.Empty, schemafile);

// Funció que s'executarà quan detecti un error
validator.ValidationEventHandler += new
ValidationEventHandler(Error);

// Llegim l'arxiu amb el validador
```

```
while (validator.Read()) {

}

....

private static void Error(object sender, ValidationEventArgs e) {
  Console.Errors.WriteLine(e.Message);
}

....
```

8. Biblioteca avançada

Hi ha un conjunt de biblioteques que han estat dutes a Mono usant *bindings* o amb implementacions nadiues. N'estudiarem algunes: una per a gestionar arxius, el registre de Gnome; la internacionalització i la compilació automàtica.

8.1. GnomeVFS

Aquesta biblioteca ens permet tenir una interfície per a poder accedir a arxius siguin on siguin. Una vegada obtingut l'arxiu, la biblitoeca ens permet, mitjançant MIME, saber-ne el tipus de contingut i fer-hi operacions. Per a poder fer referència a un arxiu hem d'usar la URI:

Accés a arxius

Podem accedir a arxius de diverses maneres. Per exemple, podem accedir-hi mitjançant un WebDAV, el protocol HTTP, un sistema d'arxius NTFS o el mateix disc dur del sistema.

```
//metode://usuari:contrasenya@adreça/directori/arxiu
http://usuari@contrasenya@maquina/home/usuari/prova.tar.gz
```

```
//uri#metode[/sub_uri]
// Descomprim amb gzip i amb tar per a obtenir un arxiu en el
directori
// /directori1/arxiu
http://usuari@contrasenya@maquina/home/usuari/
prova.tar.gz#gzip#tar/directori1/arxiu
```

Podem monitoritzar els canvis, la creació i l'esborrat d'una URI:

```
// Iniciem el sistema de GnomeVFS
Gnome.Vfs.Vfs.Initialize ();

// Creem un monitor
Monitor monitor = new Monitor ();
// Afegim una funció delegada als canvis
monitor.Changed += OnChanged;
// Afegim una funció delegada a l'esborrat
monitor.Deleted += OnDeleted;
// Afegim una funció delegada a la creació
monitor.Created += OnCreated;
// Afegim una funció al canvi de les metadades de l'arxiu
monitor.MetadataChanged += OnMetadataChanged;

// Afegim un directori per ser monitoritzat
monitor.Add ("directori", MonitorType.Directory);

// Bucle principal d'execució d'esdeveniments
```

```
new MainLoop ().Run ();

// Tanquem el sistema de GnomeVFS
Gnome.Vfs.Vfs.Shutdown ();

public static void OnChanged (string monitor, string uri)
{
         Console.WriteLine ("Uri changed: {0}", uri);
}

public static void OnDeleted (string monitor, string uri)
{
         Console.WriteLine ("Uri deleted: {0}", uri);
}

public static void OnCreated (string monitor, string uri)
{
          Console.WriteLine ("Uri created: {0}", uri);
}

public static void OnMetadataChanged (string monitor, string uri)
{
          Console.WriteLine ("Uri metadata changed: {0}", uri);
}
```

Podem saber el tipus de contingut en un arxiu procedent d'una URI:

```
Gnome.Vfs.Vfs.Initialize ();
// Creem un tipus d'objecte uri
Gnome.Vfs.Uri uri = new Gnome.Vfs.Uri ("/tmp/arxiu");
// Analitzem el tipus d'arxiu que és
MimeType mimetype = uri.MimeType;
// Mostrem el tipus d'arxiu que és
Console.WriteLine ("La uri `{0}' sembla ", uri, mimetype.Name);
Gnome.Vfs.Vfs.Shutdown ();
```

Podem fer operacions (crear, modificar i esborrar) sincronitzades:

```
Gnome.Vfs.Vfs.Initialize ();

Gnome.Vfs.Uri uri = new Gnome.Vfs.Uri ("/tmp/prova");

// Obrim l'arxiu en mode escriptura
Handle handle = Sync.Open (uri, OpenMode.Write);

// Escollim el tipus d'encoding utf8 per a gravar l'arxiu
UTF8Encoding utf8 = new UTF8Encoding ();

Result result = Result.Ok;
Console.WriteLine ("Enter text and end with Ctrl-D");
while (result == Result.Ok) {
    // Mentre hi hagi alguna línia per llegir i el resultat de
gravar sigui correcte, llegir de consola.
    string line = Console.ReadLine ();
    if (line == null)
        break;
    byte[] buffer = utf8.GetBytes (line);

    ulong bytesWritten;
   // Escrivim en l'arxiu. El sistema es bloquejarà fins que acabi
l'escriptura
```

```
    result = Sync.Write (handle, out buffer[0],
        (ulong)buffer.Length, out bytesWritten);
    Console.WriteLine ("resultat escriptura '{0}' = {1}", uri,
result);
    Console.WriteLine ("{0} bytes escrits", bytesWritten);
}

// Tanquem l'arxiu
result = Sync.Close (handle);
Console.WriteLine ("resultat tancar '{0}' = {1}", uri, result);

Gnome.Vfs.Vfs.Shutdown ();
```

Podem fer operacions asíncrones:

```
// D'aquesta manera podem obrir, llegir, escriure, esborrar i tancar arxius de manera que no
bloquegi
// el programa principal

using GLib;
using Gnome.Vfs;
using System;
using System.Text;
using System.Threading;
namespace TestGnomeVfs {

        public class TestAsync {
                private static MainLoop loop;
                private static Handle handle;

                static void Main (string[] args)
                {
                        // Iniciem el sistema de GnomeVFS
                        Gnome.Vfs.Vfs.Initialize ();

                        // Creem un objecte uri de la cadena entrada per consola
                        Gnome.Vfs.Uri uri = new Gnome.Vfs.Uri (args[0]);
                        // Creem un thread perquè obri un arxiu per llegir
                        // amb la prioritat per defecte.
                        // Una vegada que ha estat obert, cridarà l'operació OnOpen
                        handle = Async.Open (uri, OpenMode.Read,
                                             (int)Async.Priority.Default,
                                             new Gnome.Vfs.AsyncCallback (OnOpen));

                        // Comencem l'execució del bucle principal d'esdeveniments
                        loop = new MainLoop ();
                        loop.Run ();

                        Gnome.Vfs.Vfs.Shutdown ();
                }

                private static void OnOpen (Handle handle, Result result)
                {
                        // Quan s'ha obert l'arxiu es crida aquesta funció
                        Console.WriteLine ("Uri opened: {0}", result);
                        if (result != Result.Ok) {
                                // En cas que hagi anat malament, tancarà el bucle
                                // principal d'esdeveniments
                                loop.Quit ();
                                return;

                        }
                        // Definim un buffer per poder llegir l'arxiu
                        byte[] buffer = new byte[1024];
                        // Creem un thread perquè llegeixi l'arxiu
                        // i el desi en el buffer.
```

```
                    // Una vegada llegit, cridarà la funció OnRead
                    Async.Read (handle, out buffer[0], (uint)buffer.Length,
                                new AsyncReadCallback (OnRead));
            }

            private static void OnRead (Handle handle, Result result,
                                        byte[] buffer, ulong bytes_requested,
                                        ulong bytes_read)
            {
                    // Una vegada llegit part de l'arxiu
                    // ( la mida del buffer ) s'executa aquesta funció
                    Console.WriteLine ("Read: {0}", result);
                    if (result != Result.Ok && result != Result.ErrorEof) {
                            // Si ha ocorregut un error sortim del bucle principal
                            loop.Quit ();
                            return;
                    }
                    // Decidim en quina codificació treballarem
                    UTF8Encoding utf8 = new UTF8Encoding ();
                    // Escribim per pantalla el contingut del buffer
                    Console.WriteLine ("read ({0} bytes): '{1}'", bytes_read,
                                       utf8.GetString (buffer, 0,
                                       (int)bytes_read));
                    // Si no hem acabat de llegir l'arxiu ( bytes_read=0)
                    // creem un altre thread
                    // perquè continuï llegint i quan torni a omplir el buffer
                    // torni a cridar
                    // la funció OnRead
                    if (bytes_read != 0)
                            Async.Read (handle, out buffer[0], (uint)buffer.Length,
                                        new AsyncReadCallback (OnRead));
                    else
                            // En cas d'acabar l'arxiu, el tanquem de
                            // manera asíncrona
                            Async.Close (handle,
                                         new Gnome.Vfs.AsyncCallback (OnClose));
            }
            private static void OnClose (Handle handle, Result result)
            {
                    // Quan el thread creat per tancar l'arxiu el tanca,
                    // crida aquesta funció
                    Console.WriteLine ("Close: {0}", result);
                    loop.Quit ();
            }
    }
}
```

8.2. Gconf

Per a desar els paràmetres personalitzats d'un programa podem fer-ho desant un conjunt de variables en el registre de Gnome (gconf). Aquest sistema té forma d'arbre, que és el lloc on desa totes les configuracions de cada programa i de l'escriptori. En aquest arbre hi ha la definició de la seva estructura, que s'anomena GConf Schema. D'aquesta manera ens permet restringir-ne el contingut.

A continuació teniu un exemple per desar quin arxiu s'usa per a fer *log* i si s'activa o no s'activa:

```
using GConf;
```

```
...
GConf.Client client;
// Base on desarem les dues variables
static string KEY_BASE="/apps/monodemo";
// Variable de text en què desarem el nom de l'arxiu
static string KEY_FILENAME = KEY_BASE + "/archivo";
// Variable booleana en què desarem si fem log o no
static string KEY_LOGGING = KEY_BASE + "/log";

// Creem un client del servei gconf
client = new Gconf.Client();
// Afegim un delegate a gconf perquè, en cas que es modifiqui
qualsevol variable
// a partir de KEY_BASE cridi la funció GConf_Changed
client.AddNotify(KEY_BASE,new NotifyEventHandler(GConf_Changed));
...

// Per a obtenir els valors
try {
  NomText = (string) client.Get(KEY_FILENAME);
  Actiu = (bool) client.Get(KEY_LOGGING);
} catch (GConf.NoSuchKeyException e) {
  // No existeix la key
} catch (System.InvalidCastException e) {
  // No és del tipus sol·licitat
}

...

// Per a desar els valors
client.set(KEY_LOGGING, true);
client.set(KEY_FILENAME,"/tmp/prova.log");
```

8.3. Internacionalització

Avui dia, el desenvolupament d'aplicacions n'inclou la traducció en la majoria dels casos. Encara que el públic objectiu d'una aplicació sigui inicialment un col·lectiu amb una llengua determinada, és previsible que a mitjà termini es vulgui distribuir el programa entre col·lectius que utilitzen altres llengües.

Una bona pràctica és desenvolupar les aplicacions en anglès com a idioma bàsic i crear una primera versió traduïda al castellà. Si seguim aquesta metodologia, assegurarem dos punts importants:

1) l'aplicació desenvolupada està totalment preparada per a ser traduïda a altres idiomes, i

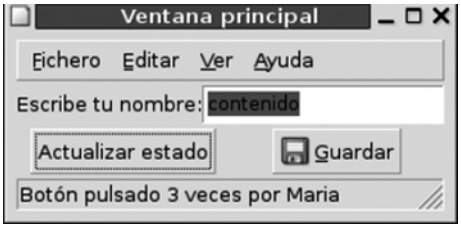
2) l'aplicació ja està disponible en anglès, de manera que hi poden accedir per a fer-ne traduccions altres persones que parlin idiomes diferents.

Normalmente, la traducció dels textos d'una aplicació és solament una part del procés d'**internacionalització** i, a part dels textos, també s'ha de tenir en compte que en les diferents cultures hi ha maneres diferents d'expressar les dates, les quantitats numèriques, el dia inicial de la setmana, etc.

Abreujarem *internacionalització* amb "i18n".

Per a mostrar com es poden traduir fàcilment les aplicacions que fem usant l'arquitectura exposada en aquest mòdul (C#+GTK+Glade), farem servir un exemple. Es tracta d'una aplicació no funcional, simplement amb ànim de mostrar diferents contextos en què podem trobar cadenes de text que s'han de traduir. L'aplicació té la interfície creada amb Glade que podeu veure a la figura 20.

Figura 20



Es tracta d'una interfície que ofereix a l'usuari un menú convencial, una àrea de contingut amb una etiqueta, un camp de text i dos botons; i una barra d'estat. La funcionalitat és mínima, i consisteix a actualitzar el text de la barra d'estat en el moment en què l'usuari prem el botó esquerre. La barra d'estat indicarà el nombre de vegades que l'usuari amb el nom indicat en el camp de text ha polsat el botó esquerre.

En primer lloc veiem el codi de l'aplicació, que ha estat desenvolupada d'entrada en anglès:

```
using GTK;
using System;
using Glade;
using Mono.Unix;


public class ExempleI18Ngui
{
  [Widget] Statusbar barraEstat;
  [Widget] Entry     entryNom;
  int                cont;


  public static void Main (string[] args)
  {
    Catalog.Init("DominiTraduccio","./locale");
    Application.Init ();
    new ExempleI18Ngui();
    Application.Run();
  }


  public EjemploI18Ngui()
  {
    XML glade = new Glade.XML("ExempleI18Ngui.glade",
                              "window1",
                              "DominiTraduccio");
    glade.Autoconnect (this);
    cont = 0;
    barraEstat.Push( 0, Catalog.GetString("Welcome!") );
```

```
  }

  void on_botonBEstado_clicked(object o, EventArgs args)
  {
    cont++;

    string format = Catalog.GetPluralString(
                      "{0} pushed the button once",
                               "{0} pushed the button {1} times",
                                  cont);

    string missatge = String.Format(format, entryNom.Text, cont);

    barraEstat.Pop(  0 );
    barraEstat.Push( 0, missatge);
  }

  void on_my_menu_entry1_activate   (object o, EventArgs args){}
  void on_new1_activate             (object o, EventArgs args){}
  void on_open1_activate            (object o, EventArgs args){}
  void on_save1_activate            (object o, EventArgs args){}
  void on_save_as1_activate         (object o, EventArgs args){}
  void on_quit1_activate            (object o, EventArgs args){}
  void on_cut1_activate             (object o, EventArgs args){}
  void on_copy1_activate            (object o, EventArgs args){}
  void on_paste1_activate           (object o, EventArgs args){}
  void on_delete1_activate          (object o, EventArgs args){}
  void on_about1_activate           (object o, EventArgs args){}

  private void OnWindowDeleteEvent (object o, DeleteEventArgs
args)
  {
    Application.Quit ();
    args.RetVal = true;
  }
}
```

La compilació d'aquest exemple s'ha de fer incloent `Mono.Posix` per a disposar de les eines de traducció, a part dels recursos habituals per a treballar amb Glade:

```
mcs -pkg:GTK-sharp,glade-sharp -r:Mono.Posix -
resource:ExempleI18Ngui.glade ExempleI18Ngui.cs
```

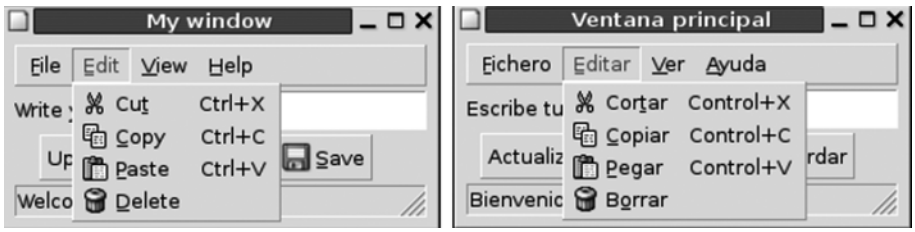
En primer lloc, veiem que el mètode principal indica quin és el nom del domicili de traducció (`DominiTraduccio`) i el directori en què es podran trobar els arxius de traducció de les cadenes de text. Més endavant àmpliarem aquest punt.

1) Elements estàndard del sistema traduïts directament

A continuació, es crida el constructor de l'aplicació que crearà la interfície a partir de l'arxiu de descripció Glade en XML. Noteu que en aquesta ocasió el darrer paràmetre de `Glade.XML` no és `null`, com en els exemples dels apartats anteriors, sinó el domini de traducció. En indicar aquest domini de traducció, el creador de la interfície no solament crearà els elements definits en l'arxiu XML, sinó que en buscarà automàticament la traducció per a tots aquells que siguin elements estàndard del sistema.

Els elements estàndard del sistema són, per exemple, els botons d'accions predeterminades, en aquest cas, el botó "Desar" i les opcions internes dels menús (figura 21).

Figura 21



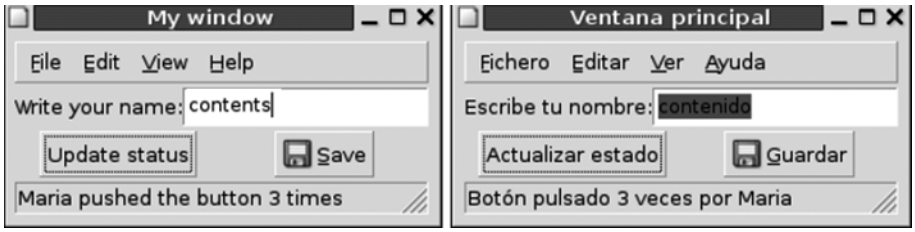
En la figura 21 es veu la mateixa aplicació executada usant l'idioma anglès i l'idioma espanyol. En ambdós casos les etiquetes que apareixen en les entrades del menú d'edició i el botó per a desar es tradueixen automàticament gràcies a la traducció d'elements estàndard que normalment són a:

```
/usr/share/locale/<idioma>/LC_MESSAGES/<dominiTraduccio>.mo
```

2) Catalog.GetString

Una vegada creada la interfície, s'inicialitza la barra d'estat amb el missatge de benvinguda. Per a crear el text que inserirem en la barra d'estat, es crida el mètode `GetString` del catàleg de traduccions (classe `Catalog` de `Mono.Unix`). Aquest mètode busca en el catàleg del domini indicat en el mètode `Main` i en retorna la traducció a l'idioma de l'entorn d'execució.

Figura 22



L'idioma del context d'execució es pot canviar mitjançant les variables d'entorn. Les dues imatges anteriors han estat creades amb les línies d'execució següents:

```
LANGUAGE=en mono ExempleI18Ngui.exe
LANGUAGE=es mono ExempleI18Ngui.exe
```

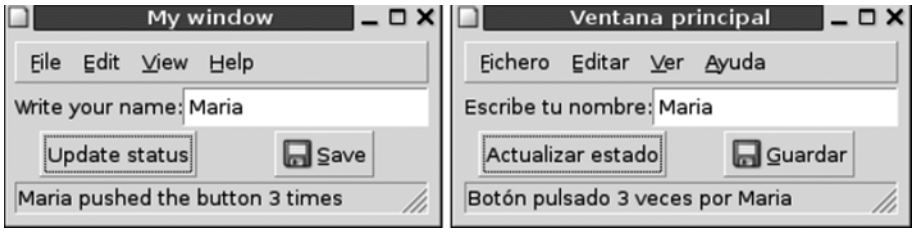
La resta del codi simplement correspon als mètodes subscrits als diferents esdeveniments de la interfície, tant els ocasionats pels botons com els ocasionats per les opcions de menú. L'únic mètode que s'ha implementat internament és el que dóna resposta al botó d'actualitzar la barra d'estat. Inicialment s'assigna

un missatge de benvinguda a la barra d'estat, però quan l'usuari prem el botó d'actualització es modifica el missatge perquè indiqui el nombre de vegades que l'usuari ha polsat el botó. Per a aconseguir un missatge per al singular diferent del missatge del plural, hi ha el mètode Catalog.GetPluralString, que tria entre dos missatges en funció del darrer paràmetre numèric. Si és 1 escull el primer missatge, i si és més gran que 1 escull el segon missatge:

```
string format = Catalog.GetPluralString(
                                    "{0} pushed the button once",
                                "{0} pushed the button {1} times",
                                    cont);
```

Amb aquests dos mètodes de Catalog podem fer manualment la traducció de qualsevol element que no sigui estàndard de l'estoc de Gnome.

Figura 23



3) Generació dels arxius de traducció

Per a generar els arxius de traducció és necessari extraure en primer lloc les cadenes susceptibles de traducció del codi original. Per a fer-ho usarem l'ordre xgettext:

```
xgettext --join-existing --from-code=UTF-8 ExempleI18Ngui.cs
ExempleI18Ngui.glade -o ExempleI18Ngui-es.po
```

A l'ordre xgettext hi indiquem quins arxius volem explorar per a buscar cadenes traduïbles (tant arxius de codi font en C# com de descripció d'interfície Glade) i la codificació de caràcters usada. El resultat és un arxiu amb extensió .po que conté totes les cadenes que es poden traduir. Aquest arxiu el podem modificar amb qualsevol editor de text o bé fent servir alguna eina que s'hi dediqui específicament. El resultat de la traducció a l'espanyol per a l'exemple és el següent:

```
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2006-06-21 19:17+0200\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
```

```
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=ISO-8859-1\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=INTEGER; plural=EXPRESSION;\n"

#: ExempleI18Ngui.glade:8
msgid "My window"
msgstr "Finestra principal"

#: ExempleI18Ngui.glade:35
msgid "_File"
msgstr "_Arxiu"

#: ExempleI18Ngui.glade:99
msgid "_Edit"
msgstr "_Editar"

#: ExempleI18Ngui.glade:148
msgid "_View"
msgstr "_Veure"

#: ExempleI18Ngui.glade:157
msgid "My menu entry"
msgstr "Opció personalitzada"

#: ExempleI18Ngui.glade:170
msgid "_Help"
msgstr "_Ajuda"

#: ExempleI18Ngui.glade:179
msgid "_About"
msgstr "Al_voltant de..."

#: ExempleI18Ngui.glade:218
msgid "Write your name:"
msgstr "Escriu el teu nom:"

#: ExempleI18Ngui.glade:243
msgid "contents"
msgstr "contingut"

#: ExempleI18Ngui.glade:272
msgid "Update status"
msgstr "Actualitzar estat"

#: ExempleI18Ngui.cs:45
#, csharp-format
msgid "{0} pushed the button once"
msgid_plural "{0} pushed the button {1} times"
msgstr[0] "Botó polsat una vegada per {0}"
msgstr[1] "Botó polsat {1} vegades per {0}"

#: ExempleI18Ngui.cs:37
msgid "Welcome!"
msgstr "Benvingut!"
```

A partir de l'arxiu .po generarem l'arxiu .mo per a l'idioma a què hem fet la traducció, usant l'ordre msgfmt:

```
msgfmt ExempleI18Ngui-es.po -o locale/es/LC_MESSAGES/
DominiTraduccio.mo
```

L'arxiu .mo tindrà el nom del domini indicat en cridar l'ordre Catalog.Init, i estarà col·locat en el directori que hem indicat amb l'esmentada inicialització, o bé en el directori compartit de traduccions del sistema (per exemple, /usr/share/locale/es/LC_MESSAGES/).

8.4. Nant i Nunit

8.4.1. Nant

Nant és un sistema per a compilar tot un projecte usant la filosofia d'Ant a Java. Construint un arxiu `.build` podem definir les diferents normes per a poder testar, compilar, executar o instal·lar fàcilment una aplicació. Aquest arxiu funciona a l'estil del típic `Makefile`, de manera que podem definir normes que requereixin altres normes i podem delegar a altres arxius les diferents operacions. Ens permet treballar amb conjunts d'arxius dins d'un zip, un tar, en un cvs, empotrats en l'*assembly* o en un directori. Les funcions bàsiques són les següents:

- **csc:** compila un arxiu `.cs`. En aquesta tasca se li pot demanar que l'objectiu sigui una biblioteca o un executable, que tingui diferents recursos referenciats o encastats a dins, que ometi errors de *warning* de valors concrets, que usi un conjunt de paquets, i que usi un conjunt ampli d'arxius de codi font. Executa diferents compiladors depenent de si usem .NET framework o Mono.

- **copy/delete/move:** ens permet treballar amb els arxius.

- **exec:** executa un programa extern.

Per a poder fer una compilació bàsica, aquí tenim un exemple d'arxiu `.build`:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project name="proyecto1" default="build" basedir=".">
        <description>Projecte de prova</description>
        <property name="project.name" value="Projecte1"/>
        <property name="project.version" value="2.0"/>
        <description>Posem el debug a cert com a variable global</description>
        <property name="debug" value="true"/>
        <target name="clean" description="Netejar">
                <echo message="Netejant"/>
                <delete>
                        <description>Esborrem un arxiu en el directori bin</description>
                        <fileset basedir="./bin/">
                        <include name="Programa.exe"/>
                        <include name="Biblioteca.dll"/>
                        </fileset>
                </delete>
        </target>
```

```
<description>Creem una regla per a compilar el programa. Requereix la
                biblioteca per a poder compilar-la</description>
        <target name="programa" description="Programa" depends="biblioteca">
                <echo message="Compilant programa"/>
                <csc target="exe" output="bin/Programa.exe" debug="${debug}">
                        <sources failonempty="true">
                                <include name="src/Programa.cs"/>
                        </sources>
                        <references>
                                <include name="bin/Biblioteca.dll"/>
                        </references>
                </csc>
        </target>
        <description>Creem una regla per a compilar la biblioteca</description>
        <target name="biblioteca" description="Biblioteca">
                <echo message="Compilant biblioteca"/>
                <csc target="library" output="bin/Biblioteca.dll" debug="${debug}">
                        <sources failonempty="true">
                                <include name="src/libPrograma/*.cs"/>
                                <include name="src/archivobiblio.cs"/>
                                <include name="src/libPrograma2/*.cs"/>
                        </sources>
                        <description>Les referències de dlls </description>
                        <references>
                                <include name="System.Data.dll"/>
                                <include name="System.Xml.dll"/>
                                <include name="Npgsql.dll"/>
                        </references>
                        <description>Recursos que s'afegiran al binari</description>
                        <resources>
                                <include name="config.xml"/>
                        </resources>
                </csc>
        </target>
```

Per a poder compilar-lo executaríem:

```
nant clean; // netegem els directoris
nant biblioteca; // compilem la biblioteca
nant programa; // compilem el programa
```

8.4.2. Nunit

Nunit és un sistema per al desenvolupament d'aplicacions amb la metodologia orientada a proves. Aquesta biblioteca ens permet programar diferents classes que ens permetran testar el funcionament de les nostres aplicacions d'una manera sistemàtica. Amb aquest mètode primer podem desenvolupar les proves i després programar l'aplicació perquè compleixi els diferents tests.

Un test no és res més que una classe que usa les funcions d'`assert` per a comprovar que una variable té un valor en concret després d'usar les funcions programades, i que ens permet verificar si es llancen excepcions o altres comprovacions.

Podem fer *asserts* per a comprovar la igualtat de dos decimals o enters, de dos reals amb tolerància i objectes. Podem comprovar si dos objectes són el mateix, si un objecte està dins d'una llista, comprovar si enters, decimals o reals

són més grans o més petits, si un objecte és una instància d'un tipus, si és nul, no nul, cert o fals, i si una cadena conté un valor.

Creem un test d'una classe que vulguem desenvolupar:

```
// Creem un namespace amb la classe que volem desenvolupar i els
tests
namespace banc
{
  using System;
  using NUnit.Framework;

  // Volem comprovar el funcionament d'una classe que gestiona els
comptes d'un banc

  [TestFixture]
  public class CompteTest
  {
    Compte origen;
    Compte desti;

    // Configuració dels tests
    [SetUp]
    public void Init()
    {
      // Creem un compte de prova
      origen = new Compte();
      // Afegim una quantitat
      origen.Deposit(200.00F);
      // Creem un compte de prova
      destinacio = new Compte();
      // Afegim una quantitat
      destinacion.Deposit(150.00F);
    }

    // Així definim un test
    [Test]
    public void TransferFunds()
    {
      // Transferim una quantitat de diners des de l'origen fins
al destí
      origen.TransferFunds(destination, 100.00f);
      // Amb aquest test comprovem que el destí tingui 250
      Assert.AreEqual(250.00F, destinacion.Balance);
      // Y també comprovem si a l'origen li queda 100
      Assert.AreEqual(100.00F, origen.Balance);
    }

     // Test en què esperem rebre una excepció
    [Test]
    // Esperem aquesta excepció
    [ExpectedException(typeof(NoHihaFonsException))]
    public void TransferWithInsufficientFunds()
    {
      origen.TransferFunds(destinacio, 300.00F);
    }

    // Test que no volem que s'executi
    [Test]
    [Ignore("Ignorem aquest test")]
    public void TransferWithInsufficientFundsAtomicity()
    {
      try
      {
        origen.TransferFunds(destinacio, 300.00F);
      }
```

```
      catch(NoHihaFonsException expected)
       {
       }

       Assert.AreEqual(200.00F,origen.Balanç);
       Assert.AreEqual(150.00F,destinacio.Balanç);
     }
   }
}
```

Una vegada tenim el test, podem fer la classe:

```
namespace banc
{
  using System;

  // Excepció que s'usarà quan no hi ha fons
  public class NoHihaFonsException: ApplicationException
  {
  }

  public class Compte
  {
    // Variable del balanç
    private float balanç;

    // Variable que no es pot modificar des de l'exterior amb el balanç mínim
    private float minimumBalance = 10.00F;
    public float MinimumBalance
    {
       get{ return minimumBalance;}
    }

    // Afegim diners al compte
    public void Deposit(float amount)
    {
      balance+=amount;
    }

    // Retirem diners del compte
    public void Withdraw(float amount)
    {
      balance-=amount;
    }

    // Enviem diners d'un compte a un altre
    public void TransferFunds(Account destination, float amount)
    {
      // Si el balanç menys la quantitat és menor que el mínim, llancem una excepció
      if(balance-amount<minimumBalance)
        throw new NoHihaFonsException();
      // Si no, afegim i retirem diners
      destination.Deposit(amount);
      Withdraw(amount);
    }

    public float Balance
    {
      get{ return balance;}
    }
  }
}
```

Una vegada tenim els dos codis, podem compilar-los amb:

```
mcs -pkg:nunit testCompte.cs Compte.cs -target:library -
out:Compte.lib
```

Una vegada tenim l'assemblatge, podem comprovar els tests amb l'eina de consola:

```
nunit-console Compte.lib
```

També podem usar les aplicacions gràfiques gnunit i monodevelop perquè ens mostrin gràficament l'estat de tots els tests.

9. Aplicacions

Aquest últim apartat del mòdul de creació d'aplicacions amb interfície gràfica en Gnome repassa algunes de les primeres aplicacions d'escriptori desenvolupades amb la tecnologia exposada: Mono i GTK. Es tracta d'aplicacions d'escriptori que posen de manifest l'alt rendiment de la tecnologia Mono, ja que duen a terme tasques sobre gran quantitat de dades de manera molt diferent. Malgrat la gran capacitat i prestacions que tenen, totes les aplicacions tenen una interfície tan simplificada com sigui possible, seguint les línies d'usabilitat ja comentades de Gnome.

Les quatre aplicacions descrites són les següents:

- **F-Spot**: gestor de fotografies, dissenyat per a manipular grans col·leccions. Ofereix opcions de classificació i exportació d'aquestes col·leccions.

- **MCatalog**: catàleg de discos, llibres i pel·lícules. Ofereix múltiples formes de cerca i importació/exportació de dades.

- **Muine**: reproductor de música, ideat per a explorar col·leccions grans d'arxius musicals. Ofereix la descàrrega de caràtules i informació bàsica sobre les pistes reproduïdes.

- **Beagle**: buscador de dades en els nostres mitjans d'emmagatzemament locals. Ofereix temps de resposta mínims per a localitzar qualsevol tipus d'informació relacionada amb les paraules clau.

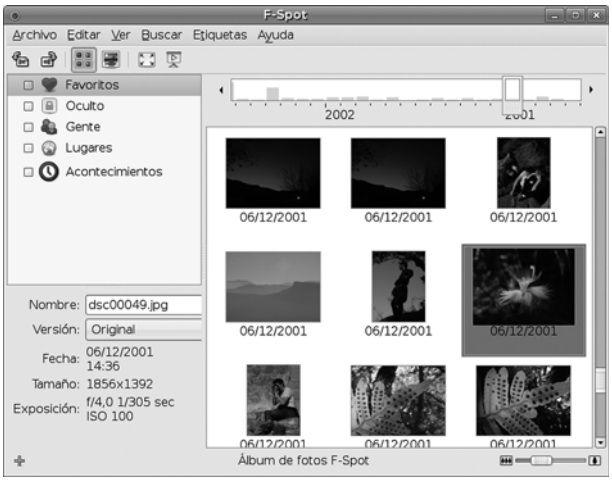
9.1. F-Spot

L'aplicació d'escriptori F-Spot permet gestionar un gran volum de fotografies mitjançant una interfície molt senzilla. La seva capacitat va més enllà de la classificació i localització de fotografies, i arriba fins a funcions bàsiques d'edició d'imatge i publicació.

La interfície bàsica es mostra a la figura 24.

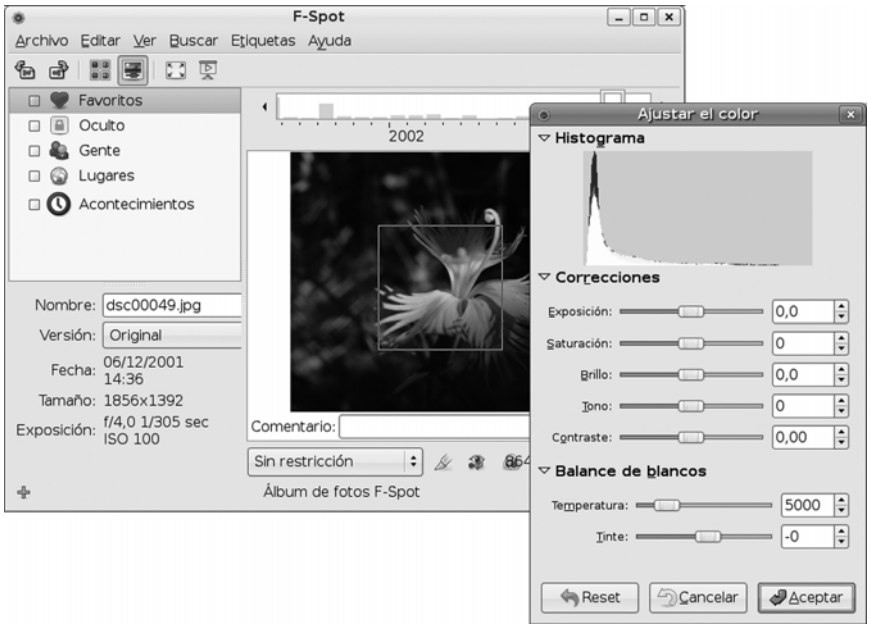
Es tracta d'una finestra dividida en quatre àrees principals: informació sobre la imatge escollida (inferior esquerra), categories (superior esquerra), barra de desplaçament del contingut (superior dreta) i àrea de contingut (inferior dreta). La barra de desplaçament del contingut es pot basar tant en la data de captura de les imatges com en les carpetes que les contenen.

Figura 24



L'àrea de contingut mostra per defecte el conjunt de miniatures de les fotografies corresponents al període seleccionat i les categories seleccionades. La mida de les miniatures es pot graduar fàcilment amb la barra d'escala inferior. En fer doble clic sobre una imatge, la imatge es mostra ampliada, fins a ocupar tota l'àrea de contingut. En el moment en què es mostra ampliada una de les imatges, és possible fer-hi operacions de manipulació bàsiques. Aquestes operacions inclouen la reducció d'ulls vermells, el reenfocament, el canvi a blanc i negre, el canvi a tons sèpia i els ajustaments del color, com mostra la figura 25.

Figura 25



F-Spot desa tota la metainformació referent a les imatges en una base de dades pròpia. Quan fem les operacions de manipulació bàsica descrites anteriorment, també desa automàticament una còpia de la imatge original, de tal manera que en qualsevol moment és possible veure les versions anteriors de la mateixa imatge.

Un altre punt destacat de l'aplicació és la importació i exportació d'imatges. Per exemple, a la figura 26 s'aprecia com importa més de deu mil fotografies.

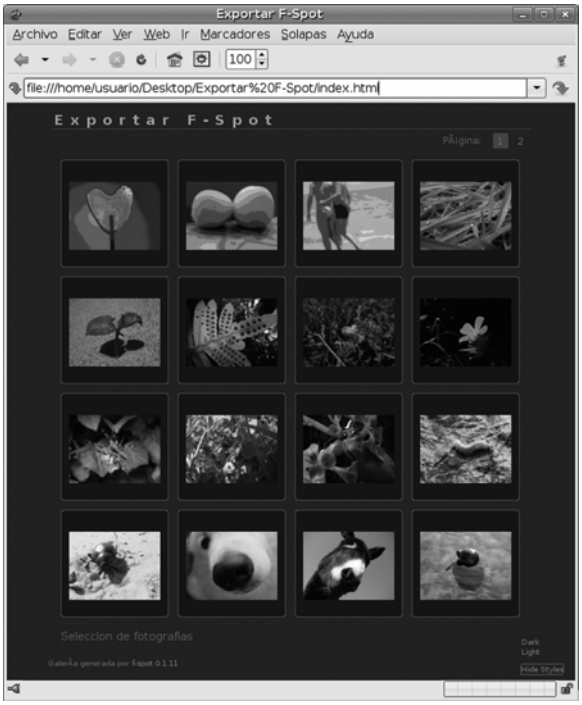
Figura 26

Al mateix temps, és possible exportar qualsevol selecció de les fotografies gestionades per F-Spot a diferents destinacions, com ara la web Flickr o qualsevol web amb el sistema Gallery, així com carpetes locals (figura 27).

Figura 27

L'exportació a carpetes locals permet deixar solament els arxius de les imatges, o crear unes pàgines web que les continguin (figura 28).

Figura 28



El desenvolupament de l'aplicació F-Spot ha resultat interessant per a posar en pràctica la connexió entre el codi C# i les biblioteques ja existents de manipulació d'imatge. El resultat és un rendiment molt alt que no està penalitzat per tenir la interfície creada amb un llenguatge basat en màquina virtual.

9.2. MCatalog

L'aplicació MCatalog està dissenyada per a gestionar col·leccions de llibres, música o pel·lícules. La interfície intuïtiva d'aquesta biblioteca imita la disposició dels volums classificats en una biblioteca virtual, de manera que la seva exploració visual recorda la cerca de volums en una biblitoeca tradicional.

La interfície bàsica de l'aplicació mostra quatre àrees principals (figura 29): la llista de col·leccions (esquerra), el contingut de la col·lecció (centre), la descripció de l'element seleccionat (dreta) i la barra de cerca (inferior).

Figura 29

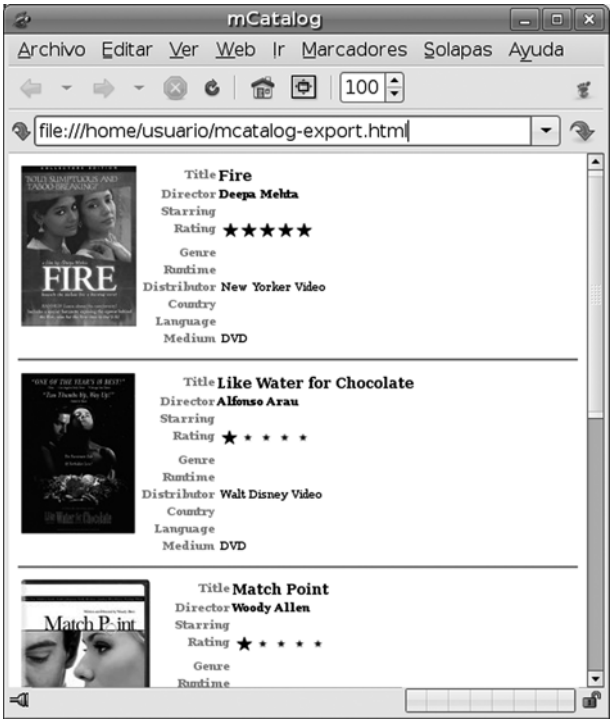
Seguint les línies de disseny de Gnome, usar-la resulta senzill i intuïtiu. En el moment en què volem afegir un volum, tenim la possibilitat d'indicar algunes paraules clau i fer una cerca per Internet per no haver d'omplir la resta dels camps (figura 30).

Figura 30

Automàticament es localitzen les metadades, inclosa la caràtula del volum, que es mostrarà en la visita virtual de la biblioteca.

A part de les possibilitats de gestió internes de la informació, que s'emmagatzema en una base de dades local SQLlite, també és possible exportar la col·lecció a altres formats, per exemple, a una pàgina web, tal com mostra la figura 31.

Figura 31



L'aplicació Mcatalog resulta interessant per a comprovar el bon rendiment de la interacció de Mono amb les bases de dades.

9.3. Muine

Muine és un reproductor de música amb capacitat per a gestionar grans col·leccions. La seva interfície simple proporciona una manera senzilla d'accedir a qualsevol cançó o grup de cançons que formi part de la nostra col·lecció.

La interfície principal mostra tres àrees principals, tal com es pot veure a la figura 32.

La barra de control (superior), l'àrea d'informació sobre la cançó reproduïda (intermèdia) i la llista de reproducció (inferior). La llista de reproducció es pot alterar fàcilment per a afegir-hi títols, esborrar-ne, reordenar-los i buscar-los de manera individual. A part, amb el botó "Reproduir àlbum" de la barra de control, és possible seleccionar totes les cançons pertanyents a un mateix àlbum (figura 33).

Figura 32

Figura 33

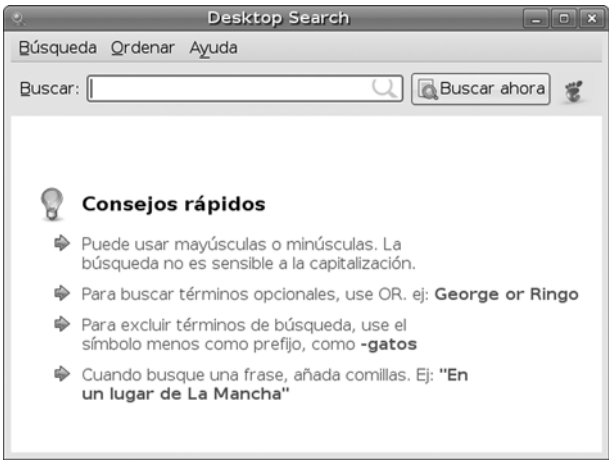
Muine localitza automàticament per Internet la caràtula dels àlbums i la mostra en el moment en què els estem explorant o quan es reprodueix una cançó que hi és inclosa.

El desenvolupament de l'aplicació Muine ha estat acompanyat des dels inicis de la millora dels vincles entre l'arquitectura Mono i el sistema de reproducció multimèdia de Gnome gstreamer.

9.4. Beagle

Per últim, exposem l'aplicació de cerca de contingut en l'ordinador d'escriptori que més ha sorprès pel seu bon rendiment: Beagle. La seva interfície és minimalista, a l'estil del buscador Google o la seva eina equivalent Google Desktop (figura 34).

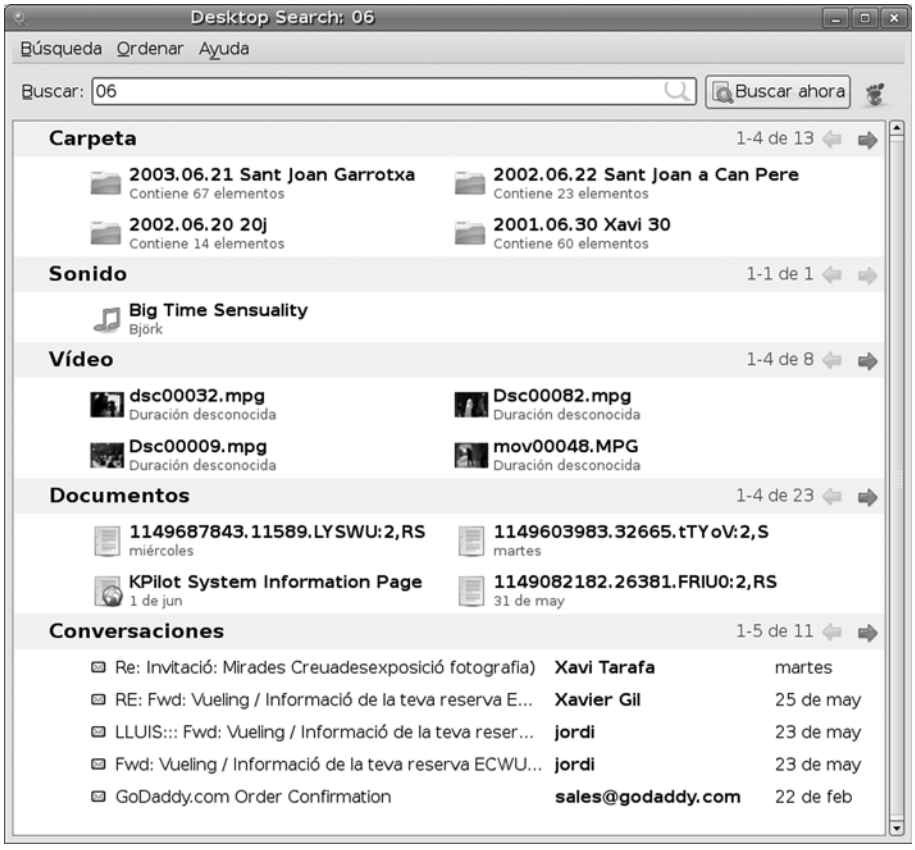
Figura 34



Es mostra una caixa de text en què s'han d'introduir les paraules clau de la cerca, que admet la sintaxi habitual per a expressar criteris compostos. Aquesta interfície és independent del motor que crea l'índex de les dades en disc i que s'excuta en paral·lel i s'actualitza a mesura que es produeixen canvis.

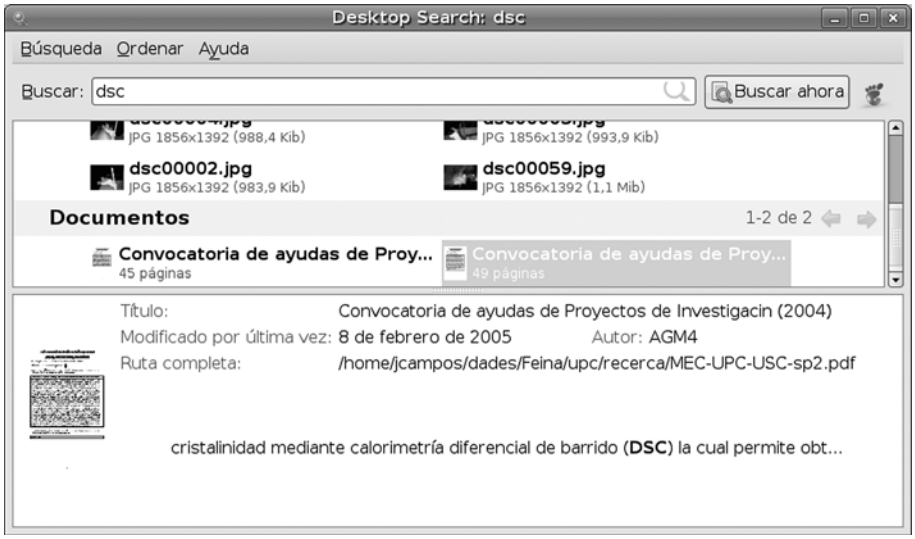
Els elements que pot localitzar Beagle –tant pel seu nom com pel seu contingut– no solament són arxius en si. Beagle també és capaç d'accedir al contingut del nostre correu electrònic, a converses instantànies o a l'historial de navegació. El resultat es mostra de manera resumida segons el tipus d'informació, tal com es pot comprovar en la figura 35.

Figura 35



Un simple clic sobre qualsevol dels elements ens en mostra més informació (figura 36).

Figura 36



I un doble clic executa l'aplicació corresponent per tal de visualitzar l'element seleccionat. Tot plegat amb una velocitat sorprenent que permet que la cerca s'actualitzi a mesura que teclegem les paraules clau.

El desenvolupament de l'aplicació Beagle ha comportat una avenç en la comunicació de Mono amb el sistema operatiu, entre altres, la comunicació del codi C# amb el sistema de monitorització de canvis en el disc del sistema operatiu. D'aquesta manera, el motor d'indexació subjacent de Beagle està al corrent dels canvis que es produeixen per a actualitzar l'índex que ha creat inicialment. La comunicació entre els diferents components es fa per mitjà del nou estàndard Dbus, proposat per l'organització Free-Desktop, que ara també serà adoptat per l'escriptori KDE.

Bibliografia

Dumbill, E.; Bornstein Niel, M. (2004). *Mono: A Developer's Notebook*. (1a ed., juliol, 21). Ed. O'Reilly Media.
ISBN: 0596007922

Mamone, M. (2005). *Practical Mono (Expert's Voice in Open Source)* (desembre, 8). Ed. Apress.
ISBN: 1590595483

H. Schoenig, H.; Geschwinde, E.; Schonig, H. (2003). *Mono Kick Start* (1a. ed., setembre, 15). Ed. Sams.
ISBN: 0672325799

Web del projecte Mono: http://www.mono-project.com

Web sobre Mono en català: http://tornatmico.org

GNU Free Documentation License

GNU Free Documentation License
Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software. We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such

manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent.

An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the

Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material.

If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location

until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any

sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit.

When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form.

Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from

their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.