



Desenvolupament d'interfícies

CFGS.DAM.M07/0.10

Desenvolupament d'aplicacions multiplataforma

Aquesta col·lecció ha estat dissenyada i coordinada des de l'Institut Obert de Catalunya.

Coordinació de continguts

Miguel Angel Carpintero Rodríguez

Redacció de continguts

Marcel García Vacas

Eduard Latorre Jarque

Primera edició: setembre 2010

© Departament d'Ensenyament

Material realitzat per Eureca Media, SL

Dipòsit legal: DL B 13831-2015



Llicenciat Creative Commons BY-NC-SA. (Reconeixement-No comercial-Compartir amb la mateixa llicència 3.0 Espanya).

Podeu veure el text legal complet a

<http://creativecommons.org/licenses/by-nc-sa/3.0/es/legalcode.ca>

Introducció

Un dels camps de la informàtica que es troba en constant evolució i experimentació és el del disseny i desenvolupament d'interfícies d'usuari. I això no només es degut a l'evolució d'eines de desenvolupament o aparició de nous estàndards, si no també al desenvolupament d'interfícies hardware que donen la volta a la forma d'interacció habitual amb les màquines.

Durant aquest mòdul veurem des de pautes i estàndards de disseny d'interfícies fins el manteniment i la documentació del procés de desenvolupament, passant per l'estudi de les eines i llenguatges més adequats que podem trobar.

En la unitat “Disseny d'interfícies” ens centrarem en els conceptes bàsics sobre interfícies d'usuari, estudiant les seves característiques, els llenguatges més adequats per dissenyar-les i les eines informàtiques que ens poden ajudar.

En la unitat “Generació de documentació” veurem com podem generar dos tipus de documentació molt usuals a l'empresa: els informes, normalment destinats a l'estudi de resultats per part dels usuaris, i la documentació de la mateixa aplicació, com ajuda a l'usuari o al personal tècnic per treballar amb el programari.

Finalment en la unitat “Distribució d'aplicacions” veurem el que és una de les darreres etapes en la fase de desenvolupament: la realització de proves i la distribució de l'aplicació. És cert que les proves s'han de fer al llarg de tot el procés de desenvolupament, però també inclouen una prova final i d'acceptació. Una vegada que tenim l'aplicatiu preparat estudiarem com distribuir-lo als nostres clients, per exemple empaquetat per a un determinat sistema operatiu o en un format instal·lable directament.

Resultats d'aprenentatge

En finalitzar aquest mòdul l'alumne/a:

Disseny d'interfícies

1. Genera interfícies gràfiques d'usuari mitjançant editors visuals fent servir les funcionalitats de l'editor i adaptant el codi generat.
2. Genera interfícies gràfiques d'usuari basades en XML fent servir eines específiques i adaptant el document XML generat.
3. Crea components visuals valorant i emprant eines específiques.
4. Dissenya interfícies gràfiques identificant i aplicant criteris d'usabilitat.

Generació de documentació

1. Crea informes avaluant i utilitzant eines gràfiques
2. Documenta aplicacions seleccionant i utilitzant eines específiques.

Distribució d'aplicacions

1. Prepara aplicacions per la seva distribució avaluant i analitzant eines específiques.
2. Avalua el funcionament d'aplicacions dissenyant i executant proves.

Continguts

Disseny d'interfícies

Unitat 1

Disseny d'interfícies. Confecció d'interfícies d'usuari. XML

1. Confecció d'interfícies d'usuari
2. Generació d'interfícies a partir de documents XML

Unitat 2

Components visuals. Usabilitat

1. Components visuals
2. Usabilitat

Generació de documentació

Unitat 3

Confecció d'informes. Documentació d'aplicacions

1. Confecció d'informes
2. Documentació d'aplicacions

Distribució d'aplicacions

Unitat 4

Distribució d'aplicacions. Proves

1. Distribució d'aplicacions
2. Realització de proves

Disseny d'interfícies. Confecció d'interfícies d'usuari. XML

Marcel García

Desenvolupament d'interfícies

Índex

Introducció	5
Resultats d'aprenentatge	7
1 Confecció d'interfícies d'usuari	9
1.1 Introducció a les interfícies d'usuari	9
1.1.1 Què són les interfícies d'usuari?	10
1.1.2 Evolució de les interfícies d'usuari	15
1.1.3 Tipus d'interfícies d'usuari	19
1.1.4 Elements de les GUI. Propietats i característiques	22
1.1.5 Eines de propietat i eines lliures d'edició d'interfícies	31
1.2 Eines de disseny d'Interfícies	33
1.2.1 Elements de les eines de desenvolupament d'interfícies	33
1.2.2 Components: característiques i camps d'aplicació	36
1.2.3 Finestra de quadre de components	37
1.2.4 Altres contenidors (panells)	39
1.2.5 Afegir/eliminar components a la interfície d'usuari	40
1.2.6 Ubicació i alineació de components. Modificació de propietats	41
1.2.7 Biblioteques de components disponibles per a diferents sistemes operatius i llenguatges de programació	43
1.3 Característiques específiques de les eines de disseny d'interfícies	44
1.3.1 Controls: classes, propietats i mètodes	45
1.3.2 Diàlegs modals i no modals	47
1.3.3 Enllaç de components a orígens de dades. Datagrid	49
1.3.4 Els esdeveniments: concepte, associació d'accions, esdeveniments escoltadors.	52
1.3.5 Edició del codi generat per les eines de disseny	53
2 Generació d'interfícies a partir de documents XML	55
2.1 Introducció a XML	55
2.1.1 Què és l'XML?	55
2.1.2 Àmbit d'aplicació	57
2.1.3 Estructura d'un document XML: etiquetes, atributs i valors	59
2.1.4 Edició d'un document XML	63
2.2 Llenguatges de descripció d'interfícies basats en XML	66
2.2.1 XSLT (eXtensible Style Language Transformation)	67
2.2.2 XUL (eXtensible User interface Language)	70
2.2.3 Els esdeveniments	73
2.2.4 XIML (eXtensible Interface Markup Language)	74
2.2.5 Altres llenguatges	76
2.3 Casos d'ús: generació d'interfícies a partir de documents XML per a diferents plataformes	79
2.3.1 Cas d'ús: disseny estàtic d'una interfície gràfica	80
2.3.2 Cas d'ús: disseny dinàmic d'una interfície gràfica en codi de servidor	80
2.3.3 Cas d'ús: disseny dinàmica d'una interfície gràfica a partir de XML	82

Introducció

Un dels camps de la informàtica que es troba en constant evolució i experimentació és el del disseny i desenvolupament d'interfícies d'usuari. I això no només es degut a l'evolució d'eines de desenvolupament o aparició de nous estàndards, si no també al desenvolupament d'interfícies hardware que donen la volta a la forma d'interacció habitual amb les màquines.

En aquesta primera unitat didàctica farem una introducció a les interfícies d'usuari i a la seva història i els elements que habitualment s'utilitzen dins d'una interfície per aconseguir una bona interacció home-màquina.

També farem un repàs a algunes de les eines que podem emprar per desenvolupar els diferents tipus d'interfície, tant de programari lliure com privat. Dins d'aquest procés de desenvolupament, juga un paper important el llenguatge XML, que ens permetrà exportar, modificar i importar els nostres dissenys entre diferents eines de disseny. Veurem amb exemples com utilitzar aquest llenguatge.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Genera interfícies gràfiques d'usuari mitjançant editors visuals fent servir les funcionalitats de l'editor i adaptant el codi generat.
 - Crea una interfície gràfica fent servir els assistents d'un editor visual.
 - Utilitza les funcions de l'editor per situar els components de la interfície.
 - Modifica les propietats dels components per adaptar-les a les necessitats de l'aplicació.
 - Analitza i modifica el codi generat per l'editor visual.
 - Associa als esdeveniments les accions corresponents.
 - Desenvolupa una aplicació que inclou la interfície gràfica obtinguda.
2. Genera interfícies gràfiques d'usuari basades en XML fent servir eines específiques i adaptant el document XML generat.
 - Descriu les avantatges de generar interfícies d'usuari a partir de la seva descripció XML.
 - Genera la descripció de la interfície en XML fent servir un editor gràfic
 - Analitza el document XML generat.
 - Modifica el document XML.
 - Assigna accions als events.
 - Genera codi corresponent a la interfície a partir del document XML.
 - Programar una aplicació que inclou la interfície generada.

1. Confecció d'interfícies d'usuari

Actualment, teniu al voltant multitud d'exemples d'aplicacions informàtiques, amb les interfícies d'usuari corresponents, que ajuden a la vida quotidiana dels éssers humans. Naturalment, davant un ordinador, es faran servir moltes aplicacions, però davant un caixer automàtic en què es volen treure diners o fer qualsevol altra operació, caldrà interactuar amb un altre tipus d'interfície d'usuari (que serà interactiva, en aquest cas).

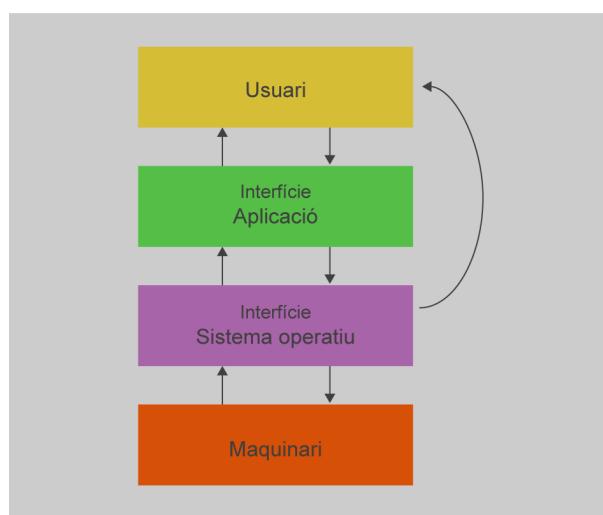
N'hi ha, però, molts més exemples: avui dia veure el televisor, amb el TDT incorporat, representa haver de fer servir una sèrie de menús i opcions. Aquest és un altre tipus d'interacció entre un maquinari electrònic i les persones, mitjançant un altre tipus d'interfície d'usuari. També ho serà un reproductor d'MP3, que portarà una petita interfície molt simple, però necessària per al seu funcionament. I no cal que parlem de les interfícies que porten tots els aparells dedicats a l'oci, com les consoles de videojocs. I es podria parlar també dels telèfons mòbils.

Les interfícies d'usuari estan en contínua evolució i s'han de conèixer una sèrie d'eines que en permetin el desenvolupament.

1.1 Introducció a les interfícies d'usuari

Començarem recordant el funcionament d'un sistema informàtic. Com aconsegueix un usuari manegar com a ell li interessa un ordinador? El conjunt de peces que componen el maquinari d'un ordinador haurà d'arribar a funcionar segons els desitjos de l'usuari. Però la relació entre un i l'altre tindrà alguns actors intermedis, com es pot veure a la figura 1.1.

FIGURA 1.1. Actors en la relació amb el maquinari



En primer lloc es troba el sistema operatiu, encarregat de la gestió i la coordinació de les tasques que porten a terme un intercanvi d'informació entre els diferents recursos. Es compondrà d'un sistema d'entrada/sortida, de la gestió de processos, de la gestió de la memòria principal o del sistema d'arxius, entre d'altres. A la vegada actuarà d'interfície entre el maquinari i les aplicacions utilitzades.

En segon lloc tindrem les aplicacions. Aquestes es troben per sobre del sistema operatiu (el qual necessitaran, adaptant-se a ell) i per sota dels usuaris. Les aplicacions tindran unes funcionalitats concretes, i ajuden els usuaris a aconseguir els seus objectius determinats.

On es troben les interfícies? Les interfícies formen part de les aplicacions. Són la part de les aplicacions amb la qual es relacionaran els usuaris.

CES
L'International Consumer Electronic Show –abreujat com a CES– és la fira mundial anual més important de tecnologies de consum, amb les novetats més importants del mercat.

A la figura 1.2 es pot veure un exemple d'interfície actual d'un microones, presentat al CES 2010, juntament amb una rentadora. Les seves interfícies estan basades en Android, sistema operatiu per a dispositius mòbils basat en una versió modificada de Linux.

FIGURA 1.2. Interfície d'un microones



Hi ha molts tipus d'aplicacions informàtiques. Es poden agrupar segons la finalitat, l'entorn d'ús, si són d'ús genèric o si són fetes a mida, però també es poden agrupar segons si són aplicacions de gestió empresarial, per a l'oci, ofimàtiques, de gestió del maquinari, aplicacions en un entorn client-servidor, en un entorn web... Ens fixarem al llarg dels apartats següents en aquelles interfícies pertanyents a utilitats creades per facilitar les tasques o automatitzar els procediments tant a escala empresarial com a escala massiva d'usuaris.

1.1.1 Què són les interfícies d'usuari?

Hi ha molts tipus d'interfícies d'usuari en molts àmbits diferents. Per aquesta raó és difícil trobar una definició única que deixi clar el concepte.

Una aplicació informàtica tindrà diverses interfícies d'usuari. Una interfície d'usuari és un conjunt d'elements (que poden pertànyer al programari o al maquinari) que ofereixen una informació a l'usuari, i permeten, a més a més, la interacció (física o lògica) entre l'usuari i l'ordinador, per mitjà d'un dispositiu perifèric o un enllaç de comunicacions.

Dintre de les interfícies d'usuari dissenyades per a aplicacions informàtiques, nosaltres ens fixarem en les interfícies d'usuari gràfiques o GUI (*graphical user interface*). Les GUI permetran a l'usuari interactuar amb el sistema informàtic de més maneres que la que històricament ha estat l'única alternativa fins fa uns anys: teclejar instruccions complexes i difícils d'entendre per a l'usuari no expert.

Les interfícies gràfiques d'usuari (GUI) són aquelles que fan servir elements gràfics, com poden ser menús, finestres o diàlegs, a més de l'ús d'altres recursos del sistema informàtic (perifèrics com el teclat, el ratolí o el so) per permetre a l'usuari interactuar amb l'ordinador de manera molt senzilla i intuïtiva.

Una de les novetats més importants de les interfícies gràfiques d'usuari, apareguda a mitjan anys vuitanta, va ser l'aparició del ratolí com a perifèric. Utilitzar-lo permet als usuaris poder precisar l'execució de les instruccions, per poder interactuar amb les interfícies sense necessitat de teclejar una sola instrucció.

Actualment podem trobar un ús generalitzat de les interfícies gràfiques d'usuari. La gran majoria dels sistemes operatius tenen aquest tipus d'interfície, de tal manera que integrar una aplicació al sistema operatiu sigui molt senzill, i només calgui adaptar alguns paràmetres. Fins i tot hi ha dispositius que han desenvolupat el seu sistema operatiu propi, com els dispositius mòbils, amb les seves interfícies pròpies. A la figura 1.3 es pot veure un exemple amb l'iPad, d'Apple.

FIGURA 1.3. Interfície gràfica IPAD



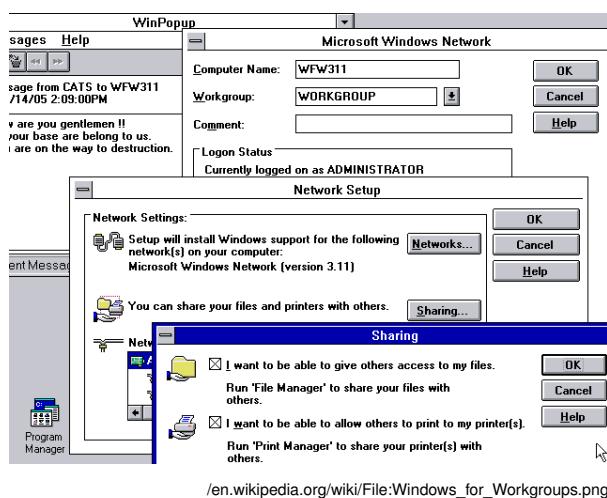
Alguns exemples d'interfícies gràfiques d'usuari els podem trobar en els sistemes següents:

- **Windows:** la interfície d'usuari de Windows ha anat evolucionant juntament amb l'evolució del sistema operatiu. No té res a veure la interfície de l'antic Windows 3.1 amb la del Windows 95, el Windows XP, el Windows Vista o el Windows 7. Totes tenen alguns denominadors comuns, com oferir un menú d'inici, una barra de tasques, un escriptori i algunes característiques com l'execució de diverses finestres de manera simultània o la configuració de moltes de les seves funcionalitats.
- **MAC:** els sistemes informàtics dels Mac porten un sistema operatiu Mac OS X. El nom de la interfície gràfica d'usuari que incorpora el seu sistema operatiu es coneix com a *Aqua* (des de l'any 2000). Abans, amb els sistemes operatius Mac OS 8 i 9 la interfície gràfica d'usuari s'havia anomenat *Platinum*. Es tracta d'una interfície que intenta oferir uns colors i unes textures prou atractives per ajudar a l'èxit del sistema operatiu. Aporta un Dock, una barra d'accisos directes a les aplicacions i a les carpetes de documents, que facilita la navegació pel sistema. Incorpora també un menú sempre a la mateixa ubicació, que anirà canviant en funció de l'aplicació que estigui activa.
- **Linux:** el sistema UNIX/Linux porta una interfície primària o bàsica de tipus text, i era l'únic sistema per poder-hi interactuar fins no fa gaire temps. Actualment hi ha diverses interfícies gràfiques, com:
 - **X Window:** incorpora un model client-servidor que permet l'ús directe per part de les aplicacions dels perifèrics d'entrada/sortida.
 - **KDE (SUSE Linux):** entorn d'escriptori per a diversos sistemes operatius. Permet l'execució automàtica de nous dispositius de dades, incorpora un inici ràpid.
 - **GNOME:** entorn d'escriptori dins el projecte GNU. Va néixer com a alternativa al KDE. Millora la usabilitat i aporta elements útils per a discapacitats.
 - **Open Look:** permet veure i executar diverses aplicacions de manera simultània en diferents finestres.

A continuació es mostren alguns exemples d'interfícies per als tres sistemes operatius que s'han explicat.

A la figura 1.4 és pot veure un exemple d'interfícies en el sistema operatiu Windows 3.11.

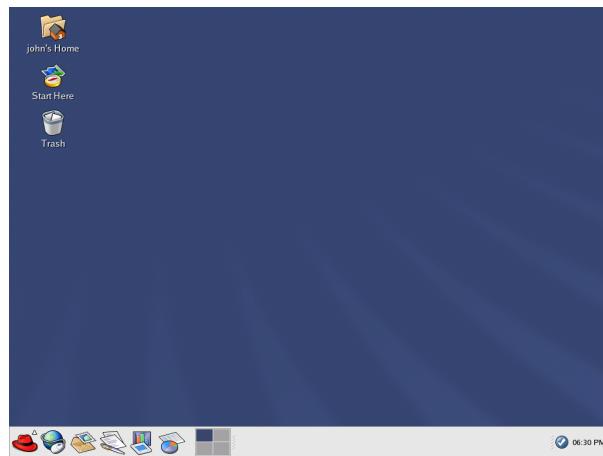
FIGURA 1.4. Interfície gràfica de Windows 3.11|Unitat1 Aparat1|http:



/en.wikipedia.org/wiki/File:Windows_for_Workgroups.png

A la figura 1.5, un de Linux.

FIGURA 1.5. Interfície gràfica de UNIX



A la figura 1.6 un exemple d'interfícies en un sistema operatiu Mac OS.

FIGURA 1.6. Interfície gràfica d'un Mac



Cada interfície tindrà la seva raó de ser i les seves funcionalitats ben definides, però podem establir algunes funcions principals que poden acomplir les interfícies

gràfiques d'usuari. Hi podem trobar:

- Configuració de les interfícies gràfiques d'usuari i de l'entorn de treball.
- Control d'accés a una aplicació informàtica.
- Sistemes d'ajuda interactius.
- Gestió i manipulació de directoris i arxius d'un sistema operatiu.
- Arrancada i tancament d'un sistema informàtic mitjançant un sistema operatiu.
- Intercanvi de dades entre diferents aplicacions.
- Comunicació entre sistemes informàtics.
- Ajuda al desenvolupament d'aplicacions informàtiques.
- Ajuda al disseny i desenvolupament d'interfícies gràfiques d'usuari.
- Gestió i manipulació de les funcionalitats que es puguin configurar en els sistemes informàtics.

Totes aquestes funcions principals més habituals de les interfícies d'usuari es poden encabir dintre de l'objectiu principal d'una interfície d'usuari, que consisteix a comunicar-se de manera senzilla i agradable amb l'usuari per mitjà d'un dispositiu d'entrada/sortida.

Però no totes les interfícies han estat, ni són, senzilles i agradables d'utilitzar. Les interfícies de línies d'instruccions (CLI, *command line interface*), són un tipus d'interfície més arcaic, però amb algunes funcionalitats molt més pràctiques per als seus usuaris, normalment usuaris més experts.

Qui preferirà fer servir una interfície de tipus CLI abans que una interfície gràfica d'usuari? Els usuaris amb molts anys d'experiència, que van haver d'aprendre les instruccions bàsiques per poder treballar amb els sistemes operatius i estan més habituats a aquest tipus d'interfícies, o, simplement, les volen fer servir, ja que una instrucció complexa serà molt complicada de representar mitjançant els elements que ofereixen les interfícies gràfiques.

Altres dispositius de maquinari faran necessari l'ús d'aquestes interfícies. Per exemple, un encaminador o un commutador una mica complex oferirà unes possibilitats de programació que requeriran el treball mitjançant línies d'instruccions.

Tots els sistemes operatius ofereixen la possibilitat de treballar mitjançant un intèrpret d'instruccions, per exemple el *shell* de UNIX, o el símbol del sistema de Windows. Caldrà tenir també en compte que no totes les funcionalitats es poden representar mitjançant interfícies d'usuari. Per exemple, les funcionalitats que requereixen moltes instruccions seran difícilment programables mitjançant interfícies; en canvi, amb una petita programació Batch en Windows o els *shell scripts* en UNIX, es programaran amb facilitat.

CLI

Acrònim de *command line interface*, és a dir, 'interfície d'ús de línia d'instruccions'. Permet als usuaris interactuar amb l'ordinador per mitjà de línies de text simples que contenen instruccions.

Un dels punts crítics serà el disseny de les interfícies d'usuari. Caldrà que acompleixi tots els requisits i que sigui senzilla d'utilitzar. Si en la fase de disseny no s'aconsegueixen aquestes premisses, difícilment s'obtindrà una interfície adequada per ser utilitzada de manera agradable i senzilla per els seus usuaris, sense que aquests necessiten hores de formació o un bon manual al costat mentre interactuen amb la interfície.

1.1.2 Evolució de les interfícies d'usuari

Parlar actualment d'interfícies gràfiques d'usuari és una qüestió que sembla que no tingui gaire raó de ser, ja que les interfícies són part del nostre dia a dia, com un element molt quotidià del que no sembla que sigui necessari parlar. Però no sempre ha estat així. No fa gaire temps no es podia parlar d'interfícies gràfiques, o, si se'n podia parlar, com a mínim no era del mateix tipus d'interfícies de què podem gaudir ara.

Al llarg dels anys les interfícies gràfiques han estat lligades de les mans amb els sistemes operatius. A mesura que aquests anaven evolucionant, les interfícies de les aplicacions que s'executaven sobre aquells sistemes operatius havien evolucionat en la mateixa mesura, o més, si això era possible.

Els sistemes operatius gràfics van apropar la informàtica a molts més usuaris dels que fins aquell moment havien fet servir els ordinadors. Ja no era necessari ser coneixedor de les ordres que calia introduir per la línia d'instruccions o tenir coneixements de BASIC. Per exemple, quan va aparèixer el Windows 3.1, el primer sistema operatiu de gran consum basat en finestres, moltes aplicacions es van adaptar a aquest nou tipus d'interfícies i també es van basar en finestres, i això va fer molt més fàcil i accessible l'ús de la informàtica al gran públic.

Per poder comprendre l'evolució de les interfícies, cal també que ens fixem en l'evolució del maquinari sobre el qual executem els programes i en l'evolució de les tècniques d'enginyeria del programari. A mesura que el maquinari evolucionava en prestacions i baixava en cost hi havia una lluita per aconseguir el negoci de la informàtica de gran consum a escala de programari, és a dir, pel que fa a sistemes operatius. Tot això ha portat a una lluita entre diferents empreses per desenvolupar els entorns més agradables i senzills d'utilitzar, fet que ha provocat una contínua evolució de les interfícies al llarg dels darrers anys.

Empreses com Apple, Xerox, Be o Microsoft han estat les que més temps i diners han dedicat a aquesta tasca durant les darreres quatre dècades. A més, també han intercanviat algunes acusacions i demandes entre elles.

Al llarg del temps es pot dividir l'evolució de les interfícies d'usuari en les fases següents:

- Interfícies d'usuari basades en la línia d'instruccions (fins als anys setanta).
- Naixement de les interfícies d'usuari, cap a l'any 1970.

- Evolució envers les interfícies gràfiques d'usuari (1980-1995).
- Interfícies gràfiques d'usuari no basades en instruccions (1996-2001).
- Interfícies gràfiques d'usuari interactives (2002-ara).

Interfícies d'usuari basades en la línia d'instruccions

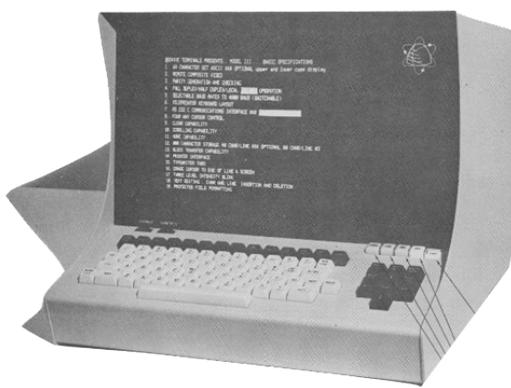
En els inicis dels sistemes informàtics no hi havia cap interfície d'usuari. Si es volia modificar la programació s'accedia directament al maquinari per fer les adequacions pertinents. Posteriorment, amb la programació per lots tampoc no hi havia una interfície que permetés la interacció. L'usuari havia d'indicar un conjunt d'instruccions al sistema que, una vegada processades, oferia uns resultats.

A partir dels anys seixanta, amb els llenguatges d'ordres, es va arribar a la interfície de línia d'instruccions. Aquesta interfície permetia a l'usuari interactuar amb l'ordinador amb una línia de text, que serviria com a text que portaria inclòs una ordre.

Cal tenir en compte que estem parlant d'una època en la qual un únic ordinador central donava servei a molts usuaris, que van enviant instruccions a aquest ordinador central, que distribueix el seu temps entre les sol·licituds.

A la figura 1.7 es pot veure un exemple d'aquell tipus de màquines.

FIGURA 1.7. Interfície basada en línia d'instruccions



Naixement de les interfícies d'usuari (1970)

En la dècada dels setanta es comença a evolucionar envers l'ús d'un ordinador per part d'un únic usuari. Aquests primers ordinadors personals permetran als usuaris emmagatzemar informació, editar-la i compartir-la d'una manera senzilla. Les aplicacions que es desenvolupen segueixen aquests objectius i intenten oferir unes interfícies d'usuari adequades per aconseguir-lo. S'acostuma a fer servir un disseny d'interfícies de dues dimensions amb menús jeràrquics de pantalla completa. Es fan servir les tecles de funció com una manera d'interactuar amb les aplicacions i de completar els menús.

Alhora l'empresa Xerox va desenvolupar, després de molts anys d'investigacions al PARC (Palo Alto Research Center), l'ordinador Xerox Star l'any 1981. Dintre d'aquestes investigacions, l'any 1973 es va desenvolupar la primera interfície gràfica d'usuari. L'objectiu era obtenir un sistema informàtic prou petit per poder ser transportable, per treballar en una oficina, amb un sistema operatiu amb interfície gràfica i poder compartir informació de manera senzilla.

A la figura 1.8 és pot veure un exemple del Xerox Star.

FIGURA 1.8. Xerox Star



Evolució envers les interfícies gràfiques d'usuari GUI (1980-1995)

En la dècada dels vuitanta els ordinadors personals es van fer més habituals per al gran públic, encara que les interfícies gràfiques d'usuari tardarien una mica a arribar. En aquesta fase es comencen a fer servir les interfícies anomenades **WIMP** (*windows, icons, menus and pointer*), és a dir, interfícies que disposen del moviment d'un punter per la pantalla i que tindran finestres, icones i menús. Es pot considerar que fins al moment es treballava en dues dimensions amb les interfícies amb línies d'instruccions i, a partir de l'aparició de les interfícies WIMP, es pot considerar que es fa servir una tercera dimensió que permet treballar amb més d'una finestra (i aplicació) a l'hora, superposant una a sobre de l'altra.

Les GUI permeten als usuaris treballar directament en aquelles funcionalitats que els interessen, i s'han de desplaçar fins a altres interfícies per localitzar i poder fer servir altres funcionalitats. Ofereixen una interacció amb els usuaris que es basa en la manipulació directa, ofereix els elements importants per a l'usuari a cada interfície i en permet la manipulació.

Les primeres interfícies gràfiques hem vist que es van començar a desenvolupar a mitjan anys setanta (Xerox), però fins a mitjan anys vuitanta no les van desenvolupar massivament altres empreses, cosa que va provocar l'ús més generalitzat dels ordinadors. En el quadre següent podem veure l'evolució de les interfícies i alguns sistemes operatius que es van anar desenvolupant:

WIMP

Tipus d'interfície, acrònim de *windows, icons, menus and pointers*, és a dir, 'finestres, icones, menús i punters'.

- **1981:** Xerox Star

- **1984:** Apple Macintosh
- **1985:** Commodore Amiga, Microsoft Windows 1.0
- **1990:** Microsoft Windows 3.0
- **1992:** IBM OS 2 2.0
- **1995:** Microsoft Windows 95
- **1998:** KDE/ Gnome
- **2001:** Apple Mac OS X 10.0, Microsoft Windows XP

Interfícies gràfiques d'usuari no basades en instruccions (1996-2001)

A partir de l'any 1995 hi ha un canvi molt important en l'ús de les interfícies d'usuari. L'aparició de sistemes operatius que interactuen amb els usuaris de manera única i completa amb GUI va convertir les interfícies en un element més de consum i va generalitzar més encara l'ús dels ordinadors personals dintre de les cases i els sistemes informàtics a les organitzacions.

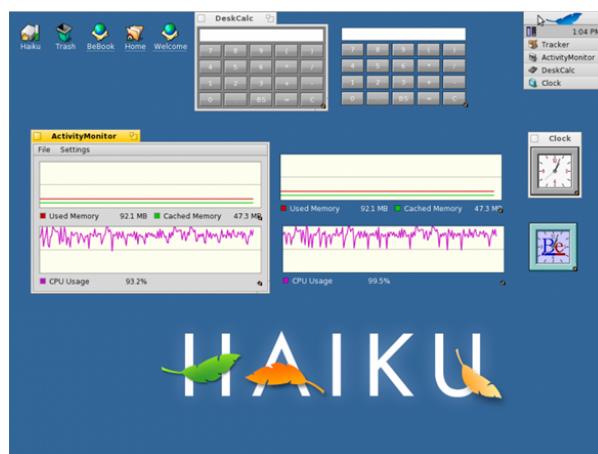
Aquest gran canvi va comportar un canvi en la manera de desenvolupar aplicacions per als nous sistemes operatius gràfics, i va permetre també investigar i evolucionar molt més en aquest camp.

Les interfícies gràfiques d'usuari continuen fent servir elements de les interfícies de l'estil WIMP, però com més va més evolucionats, amb nous elements i funcionalitats. Una de les novetats més importants és l'aparició d'elements multimèdia que es poden afegir a les interfícies, com sons (ja siguin alertes que avisen l'usuari d'una mala interacció o un so de fons que es pot escoltar durant el funcionament de la interfície).

Aquests elements es poden considerar, utilitzant el terme amb cometes, una “dimensió” més per afegir a les dues dimensions que tenien les interfícies d'estil WIMP, encara que no es pot parlar pròpiament d'una tercera “dimensió”, els elements que s'afegeixen de tipus multimèdia, so o interpretació de veu o realitat virtual és cap a on van començar a evolucionar les interfícies.

A la figura 1.9 es pot veure un exemple de sistema operatiu amb un entorn WIMP. El sistema operatiu de l'exemple es diu Haiku, un projecte de codi obert desenvolupat en C++ que vol recrear el sistema operatiu BeOS.

FIGURA 1.9. Exemple d'interfície gràfica no basada en instruccions



Interfícies gràfiques d'usuari interactives (2002-ara)

Les interfícies gràfiques que ens trobem actualment (i l'evolució envers les que ens trobarem en el futur) van encaminades a fer servir encara més dimensions i altre evolucions.

La primera nova “dimensió” que es va incorporar va ser la de fer servir elements multimèdia. En alguns casos per necessitat i en altres per investigar envers una evolució que faci encara més senzilles i útils les interfícies. Un element multimèdia podrà ser la incorporació d’imatges a les interfícies, de sons o d’animacions. Per aquesta raó es podria considerar una tercera “dimensió” el concepte de temps (les animacions faran servir un temps estipulat perquè es pugui completar) o fins i tot una quarta “dimensió” amb el concepte de comunicació verbal. Avui dia les persones amb deficiències visuals poden fer servir una interfície gràfica gràcies a les indicacions verbals que pot oferir cada element de la GUI i a dispositius d’entrada de dades específics per a ells.

Però tampoc no es deixarà de costat en el futur l’aprofitament de les noves possibilitats que pot oferir afegir una tercera dimensió real a les interfícies, gràcies també a l’evolució dels dispositius de sortida més habituals, les pantalles, que després d’evolucionar a la tecnologia de LED (també en les pantalles d’ordinadors), arribaran més aviat que tard a la tecnologia 3D, cosa molt aprofitable per les interfícies per explorar noves possibilitats.

1.1.3 Tipus d'interfícies d'usuari

Les interfícies gràfiques es poden classificar en diferents tipus en funció de les característiques i funcionalitats que tinguem en compte. Si es parla de tipus d'interfícies, en podem trobar algunes que es fan servir en els diferents sistemes operatius.

Per exemple, hi ha un tipus d'interfícies basades en WIMP. WIMP mostra un tipus

Usuaris d'interfícies

Caldrà tenir en compte el nivell de coneixements dels usuaris de les aplicacions i les interfícies a l' hora de dissenyar-les. Hi ha usuaris novells, usuaris intermedis i usuaris experts en l'ús d'aplicacions i interfícies.

de format d'interacció entre les persones o usuaris i els ordinadors (i les seves interfícies).

WIMP és un estil d'interacció que fa servir normalment un ratolí per controlar la ubicació d'un cursor que s'anirà desplaçant per les interfícies. Un altre element important del WIMP és la utilització de finestres per presentar la informació de manera organitzada. També es fan servir icones i menús per oferir les funcionalitats adients als usuaris.

A partir d'aquest conjunt d'elements, les opcions dels menús i les que representen els icones seran activades a partir dels punters que es mouran amb el dispositiu físic (ratolí), i es modificaran les informacions per mostrar a les finestres.

L'estil d'interacció WIMP cerca reduir el temps d'aprenentatge que necessitarà un usuari per utilitzar de manera òptima una interfície gràfica facilitant el record de les opcions que ofereix. A la vegada serà més senzilla de fer servir per part d'usuaris no experts.

Si dividim les interfícies en funció de la manera d'interaccionar amb els usuaris, podem trobar tres tipus d'interfícies:

- Les alfanumèriques, amb les quals caldrà treballar amb instruccions, també conegudes com les **interfícies de línia d'instruccions (CLI)**. Aquest tipus d'interfície serà l'única possibilitat de treballar amb aquests sistemes, i fan servir tota la pantalla per a aquest objectiu.
- Les **interfícies gràfiques d'usuari (GUI)**. Aquest tipus d'interfícies permeten una interacció més ràpida, senzilla i intuïtiva gràcies al fet de representar de manera gràfica els elements de les interfícies.
- Les interfícies desenvolupades per a **pantalles tàctils**. Aquestes interfícies s'han d'adaptar a les necessitats d'un dispositiu d'entrada i de sortida alhora, que serà una pantalla tàctil, i a les seves característiques específiques (fer servir icones més grans, en no poder treballar amb la mateixa precisió que un ratolí, o no utilitzar menús, per exemple).
- Les interfícies desenvolupades per a **dispositius mòbils**. Dispositius com telèfons mòbils, PDA (*personal digital assistant*) o altres eines semblants disposen de sistemes operatius i tipus d'interfícies adequades a les seves necessitats (pantalles d'una mida limitada, pantalles tàctils...).

Si ens fixem en com s'ha desenvolupat la interfície, els elements amb els quals s'ha construït, en podem trobar de tres tipus:

- De **maquinari**: es tracta d'interfícies que permeten la interacció entre l'usuari i el sistema informàtic mitjançant elements de maquinari, dispositius com polsadors o lectors de barres o altres reguladors o instruments.
- De **programari**: aquestes són les interfícies desenvolupades per a ordinadors que basen la seva creació en desenvolupament de programari, mitjançant el qual es durà a terme tota la interacció amb els usuaris.

- **De programari-maquinari:** són interfícies que compartiràn els dos tipus explcats anteriorment. Per una banda interfícies de programari amb algun element o dispositiu de maquinari necessari per recollir algun tipus d'informació específica (per exemple, un lector de codi de barres).

Altres maneres de classificar les interfícies és segons l'arquitectura de les aplicacions i la seva manera de treballar:

- **Aplicacions locals: interfícies Winforms.** Aquestes aplicacions es desenvolupen per treballar en una única màquina un únic usuari a la vegada. S'haurà pogut desenvolupar en diferents llenguatges de programació i es podrà haver vinculat a diferents tipus de bases de dades, però es trobaran instal·lades en una màquina que serà l'únic lloc des d'on es podrà fer servir. Aquest tipus d'aplicacions, habitualment, disposen d'interfícies de tipus **Winforms**. Aquest nom és el que es dóna a les API incloses en entorns integrats de desenvolupament que faciliten la creació d'interfícies basades en finestres i altres elements de tipus WIMP.
- **Aplicacions client-servidor.** Aquest tipus d'aplicacions estan basades en el treball en més d'una màquina, en què una farà de servidor i la resta (una o moltes més) faran de clients que accedeixen remotament al servidor per accedir a les dades o a les funcionalitats. Cal fer una instal·lació a la màquina servidor i una altra instal·lació a cada una de les màquines client. També és podria donar el cas de treballar amb aplicacions client-servidor en una única màquina. El tipus d'interfícies que es fan servir habitualment per a aquest tipus d'aplicacions també seran les **Winforms**, encara que també es podran fer servir **interfícies de tipus Web**.
- **Aplicacions Web.** Són un tipus específic de les aplicacions client/servidor. Aquest tipus d'aplicacions permet accedir a dades i funcionalitat per mitjà de les xarxes telemàtiques, i l'accés més habitual és el que es fa per mitjà d'un navegador d'Internet, que accedeix a un servidor web, on es localitzaran les dades i la implementació de les funcionalitats. Les **interfícies** seran de tipus **Web**, s'executaràn les funcionalitats i la manipulació de dades al servidor i enviaran només les interfícies o dades per mostrar als navegadors que accedeixen de manera remota. En els darrers temps s'està millorant aquest sistema per fer que la transferència d'informació sigui com més va més petita, i implicar llavors la màquina client com més va més.

Una darrera manera de classificar les interfícies és a partir del programari on es desenvoluparan:

- **Plataformes RichClient:** Són eines de desenvolupament de programari que contindran tots els elements necessaris per dissenyar i desenvolupar sense cap altra necessitat una interfície gràfica d'usuari. A més es podranaprofitar de les biblioteques i elements existents. Aquesta manera de treballar facilita el desenvolupament i la integració dels elements i l'aplicació desenvolupada en el seu entorn. Es poden desenvolupar aplicacions locals

API

De l'anglès *application programming interface*, 'aplicació per programar interfícies'. Facilitarà la interacció entre diferent programari.

i client/servidor. Exemples d'entorns RichClient són Visual Studio (de Microsoft) o, en codi obert, Eclipse o NetBeans.

- **Plataformes ThinClient:** Són eines de desenvolupament de programari que necessiten d'altres eines per a poder dissenyar i desenvolupar interfícies gràfiques d'usuari i parts de programari.

1.1.4 Elements de les GUI. Propietats i característiques

Estudiada l'evolució de les interfícies gràfiques d'usuari al llarg del temps, podem observar com el denominador comú ha estat arribar a oferir als usuaris unes interfícies amb una usabilitat òptima. Per aconseguir això s'ha establert que les interfícies han de complir una sèrie de característiques que, entre altres, seran:

Metàfora

Com sabeu, en literatura una metàfora consisteix a suggerir una idea fent referència a una altra cosa que el lector ja coneix.

- **Accessible i intuïtiva:** una interfície ha de ser intuïtiva en el seu ús, ha de mostrar amb claredat les funcionalitats que ofereix i ha de facilitar arribar-hi de manera senzilla i clara.
- **Ús de metàfores:** per identificar les funcionalitats que representen o els objectius que simularan, les interfícies han de fer servir metàfores que vinculin senzillament icona o imatge amb funcionalitat o objectiu.
- **Aprendentatge i ús fàcil:** les interfícies han de ser fàcils d'usar i també d'aprendre per part dels usuaris més novells.
- **Consistència:** les interfícies han de seguir un mateix disseny i estructura entre elles i també amb altres interfícies d'aplicacions anàlogues. També hauran de ser consistent quan s'executin en diferents entorns.
- **Oferir el control de les interfícies:** una interfície ha de saber lliurar el seu control a l'usuari que la farà servir.
- **Anticipació:** s'hauran de preveure els possibles errors que pugui cometre un usuari o les necessitats que pugui demostrar i oferir-hi solucions abans que apareguin o controlant-los i oferint solucions.
- **Llegibilitat:** hauran de ser fàcilment interpretables i oferir una llegibilitat adequada als usuaris.
- **Autonomia:** un usuari no ha de necessitar més informació o ajuda que la que una interfície li ofereix o, la que pot trobar a partir de les indicacions que li indicarà aquesta interfície.
- **Reduir càrrega de memòria:** per poder fer servir una interfície més d'una vegada no caldrà obligar als usuaris a recordar la ubicació de les funcionalitats, sinó que aquestes hauran de ser senzilles de trobar. Això farà que no sigui indispensable memoritzar moltes informacions per aprendre a fer servir una interfície determinada.

- **Internacionalització de la interfície:** ha de permetre ser entesa i utilitzable per usuaris de diferents cultures i idiomes o bé fent-la internacional amb icones fàcilment reconeixibles o amb la possibilitat de seleccionar l'idioma amb el qual es voldrà treballar.
- **Valors inicials:** també coneguts com a valors per defecte o estàndard. Són els valors que en les interfícies que porten un formulari incorporat apareixeran seleccionats inicialment. A més, han de poder ser descartats de manera senzilla.
- **Llei de Fitts:** una interfície ha d'aconseguir optimitzar la llei de Fitts. Aquesta llei, en ergonomia, modelarà el moviment humà, fent una estimació del temps que pot necessitar un ésser humà per moure un punter des d'una zona de la pantalla fins a una altra tenint en compte variables com els objectius, la distància fins a assolir-los i la grandària que tindran.

Les interfícies gràfiques d'usuari (GUI a partir d'ara) disposen d'una sèrie d'elements propis, comuns a moltes de les GUI desenvolupades, que disposaran d'unes característiques i propietats. Amb aquests elements es facilitarà el desenvolupament de les GUI de les aplicacions informàtiques, cosa que ofereix la possibilitat d'acomplir la majoria de les característiques definides.

Perquè l'usuari pugui fer servir aquests elements, haurà de fer servir algun dels dispositius d'entrada/sortida que s'han anat desenvolupant al llarg dels últims anys. El teclat de l'ordinador ha estat el dispositiu indispensable, com ho és avui dia el ratolí. Però n'hi ha d'altres com el *touchpad* (ratolí tàctil), la tableta gràfica, el *trackball* (bola), els *joysticks*, les pantalles tàctils o els micròfons. Com a dispositius de sortida, a banda de la pantalla, seran importants els altaveus o els LED informatius.

Alguns d'aquests dispositius seran indispensables per al funcionament correcte d'alguns dels elements que ofereixen les GUI. Cada interfície gràfica farà servir alguns elements en funció del seu entorn de treball, i no totes tenen el mateixos elements.

Alguns dels elements de les GUI que es poden trobar de manera habitual en moltes interfícies són:

- Finestres
- Quadres de diàleg
- Assistents
- Menús
- Pestanyes
- Barres d'eines
- Icones
- Entorn de treball

- Entorns d'escriptori
- Controls
- Tipografia

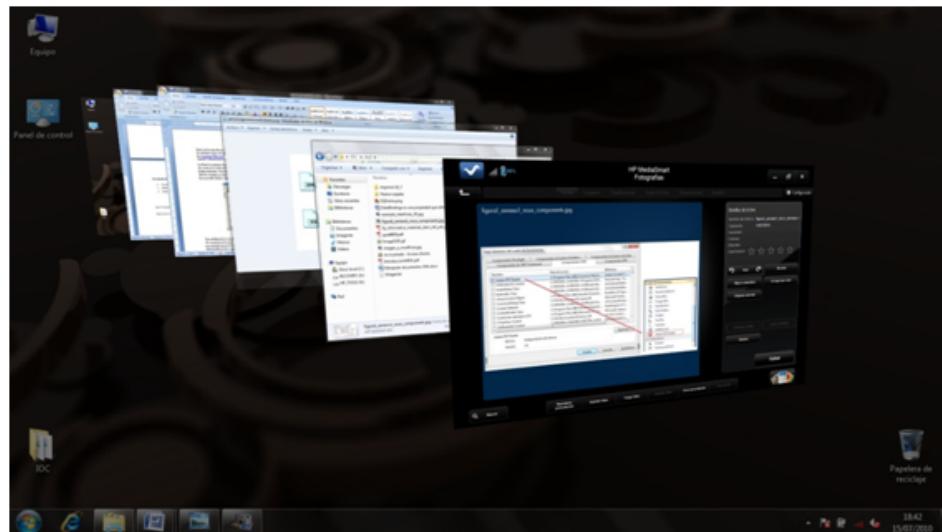
Finestres

Dintre dels diferents elements de les GUI, trobem un dels més importants, que són les finestres (*windows*). Les finestres són, normalment, bidimensionals i de forma rectangular, i s'ubicaran a l'escriptori del sistema operatiu. Quan tenim oberta més d'una finestra ubicada en un mateix escriptori, la que estarà activa serà la que podrem veure, la que estarà més a prop nostre, i la resta quedarán per sota, amagades. Cal dir que en els sistemes operatius més moderns es comencen a fer servir tècniques de transparència per poder identificar alguns elements de les finestres no actives, ubicades per sota de la finestra activa.

Les finestres ens ofereixen la possibilitat de treballar alhora amb diferents entorns, que podran ser de la mateixa aplicació o de diferents aplicacions.

Per exemple, el fet de treballar amb finestres ens ofereix la possibilitat de tenir dos documents ofimàtics oberts (un full de càcul i un document de text) i anar alternant la informació escrita en tots dos documents o anar copiant informació d'un document a un altre. A la figura 1.10 és pot trobar un exemple de treball amb finestres en un entorn de treball amb el sistema operatiu Windows 7.

FIGURA 1.10. Sistema operatiu Windows 7



En un cas d'utilització d'una mateixa aplicació informàtica, el fet de treballar amb finestres ens permetrà tenir més d'una interfície oberta al mateix temps, cosa que ens permetrà comparar l'execució de dues funcionalitats o tenir dues interfícies diferents obertes a la vegada, una per consultar dades i una altra per modificar altres dades.

Les finestres ens permetran oferir informació a l'usuari de manera organitzada. Ens permetrà navegar per les funcionalitats d'una aplicació visualitzant la infor-

mació de manera jerarquitzada. Això permet gestionar i manipular la informació de manera força senzilla. Cada finestra pot ser manipulada com a l'usuari li interessi, maximitzant-la o minimitzant-la, movent-les o tancant-les, navegant-hi o fent-les més grans... Les finestres contindran interfícies gràfiques d'usuari que incorporaran altres elements propis de les GUI, com menús, pestanyes, barres d'eines, objectes gràfics o multimèdia, que permetran tant la sortida com l'entrada de dades perquè l'usuari pugui dur a terme diversos processos.

Al llarg dels anys les finestres han anat evolucionant, i s'ha arribat avui dia al que es podria anomenar un estàndard de disseny de finestres, amb uns elements bàsics, dels quals faran ús totes les finestres, com són: el marc, la capçalera de la finestra, l'espai del contingut, la barra de desplaçament i el peu de la finestra.

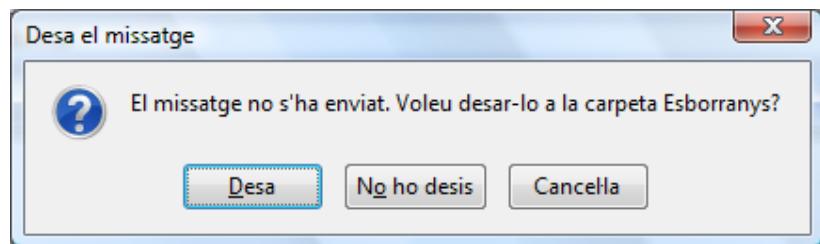
El marc ajuda a distingir què pertany a la finestra i què no. A més, serveix per redimensionar les mides de la finestra. Com pràcticament tots els elements, es podrà configurar en la majoria dels casos. La capçalera de la finestra és el lloc on es trobaran situades les icones que identifiquen i permeten arribar a les funcionalitats principals de l'aplicació que s'executa en aquella finestra. L'espai de contingut variarà en mida en funció de les dimensions de la finestra. Serà el lloc on es mostrerà la informació (imatge, continguts multimèdia, de text...). La barra de desplaçament o *scroll* apareixerà en el cas que l'espai de contingut sigui més petit que la informació per mostrar. D'aquesta manera l'usuari es pot desplaçar pels continguts no visibles inicialment a la finestra. Finalment el peu de la finestra es fa servir per oferir informacions diverses relacionades amb la finestra o amb l'aplicació i els continguts que es mostraran.

Quadres de diàleg

Els quadres de diàleg són un tipus de finestres, també considerades com a finestres secundàries. Els quadres de diàleg apareixen per sobre de la resta de finestres demanant una interacció concreta a l'usuari, que l'haurà de contestar. Aquesta interacció podrà ser només informativa (un missatge d'avís o d'avertència que caldrà confirmar que s'ha llegit per poder continuar) o un requisit concret d'alguna dada o d'alguna acció que caldrà contestar de manera concreta per poder continuar fent servir l'aplicació.

Un exemple de quadre de diàleg el trobem en intentar finalitzar una aplicació. Si hi ha hagut canvis en el document amb el qual s'ha treballat o amb les dades amb les quals s'ha obert una aplicació, en intentar finalitzar aquesta aplicació ens preguntarà si la volem finalitzar sense desar les dades o si ho volem fer. Aquest quadre de diàleg no ens permetrà continuar treballant amb l'aplicació fins que no hàgim contestat el seu missatge.

A la figura 1.11 es mostra un exemple d'un quadre de diàleg generat per l'aplicació Mozilla Thunderbird, per gestionar correu electrònic.

FIGURA 1.11. Exemple de quadre de diàleg

Assistents

Els assistents són finestres que van apareixent una darrere de l'altra fins que s'arriba a un determinat objectiu. Com el seu nom indica, assisteixen l'usuari fins a completar una funcionalitat, preguntant pas per pas totes les informacions necessàries fins a completar-la correctament. D'aquesta manera l'usuari no es podrà oblidar d'introduir una informació o escollirà malament una opció.

Els assistents són habituals en la instal·lació de programari o com a ajuda en la utilització d'una funcionalitat molt concreta.

A la figura 1.12 es mostra un exemple d'un assistent d'instal·lació del programari Mozilla Thunderbird.

FIGURA 1.12. Exemple de wizard

Menús

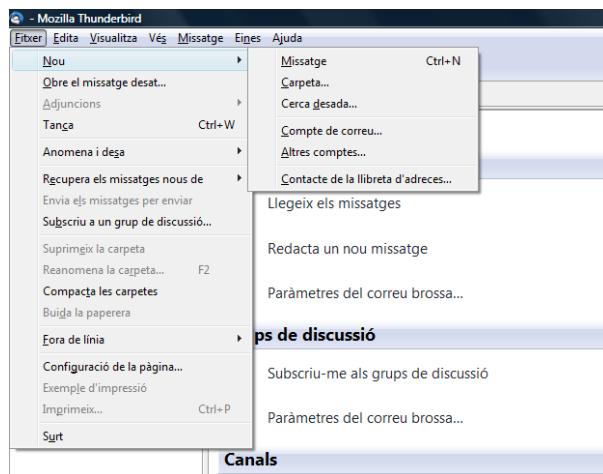
Els menús són un element molt important per a les interfícies d'usuari, ja que seran els encarregats de mostrar totes les possibilitats d'interactuar amb l'aplicació informàtica.

Els menús desplegaran tot el conjunt de funcionalitats que tindrà l'usuari al seu abast. Cada opció d'un menú disposarà d'un subconjunt d'opcions, cosa que converteix els menús en un arbre jerarquitzat amb un nombre de nivells que hauran decidit els desenvolupadors. Com més nivells tingui, més complicat serà que un usuari arribi a una funcionalitat determinada.

Els menús es troben en una àrea determinada de la interfície, normalment just a sota del marc superior. La manera més habitual de seleccionar una opció d'un menú serà fent servir el ratolí. Fent clic a sobre d'un dels textos, s'activarà aquella opció i es desplegaran totes les opcions relacionades. Però les aplicacions han de poder oferir menús accessibles per a usuaris que no disposin de ratolí, amb la possibilitat de moure's pels menús amb un teclat utilitzant la combinació de la tecla *Alt* amb la lletra subratllada per arribar a una opció determinada.

A la figura 1.13 es mostra un exemple d'un menú del programari Mozilla Thunderbird.

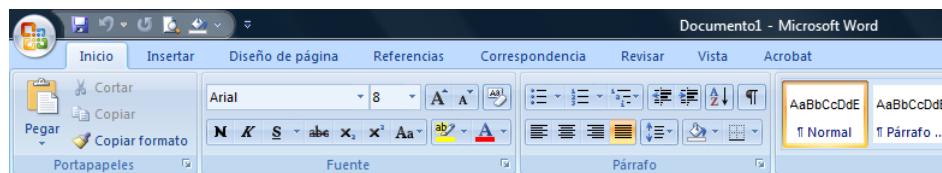
FIGURA 1.13. Exemplo de menú, de Mozilla Thunderbird



Com es pot veure a la imatge, molts menús indiquen les combinacions de tecles que permeten arribar a la mateixa funcionalitat de manera més ràpida. Un usuari habitual d'una aplicació memoritzarà les funcionalitats més indispensables per a ell. De vegades, en comptes de la combinació de tecles hi pot haver altres informacions gràfiques que complementin la informació en format de text que identifica una funcionalitat, o bé indicadors de com es troba una opció (si es pot executar o no o si està activada o no) o l'agrupació a la qual pertany.

Finalment cal tenir present l'evolució dels menús. En les darreres aplicacions, com el paquet d'ofimàtica Office 2007 o l'Office 2010, els menús han evolucionat envers una mostra de funcionalitats per mitjà d'ícones, minimitzant la presència del text. Aquesta evolució pot portar a la confusió als usuaris acostumats a una manera determinada de treballar. Es pot veure un exemple de menú de l'Office 2007 a la figura 1.14.

FIGURA 1.14. Un altre exemple de menú, del Microsoft Office 2007



Hi ha molts tipus de menús, depenent del seu entorn d'execució o de la tipologia d'aplicació a la qual pertany la interfície on es trobarà el menú. El menús més

habituals que ens podem trobar són els menús contextuels, els menús de navegació i els menús jeràrquics:

- **Menús jeràrquics:** menús representats en forma d'arbre, amb un nombre de nivells horitzontal (opcions del menú) i un nombre determinat de nivells verticals (fins on podrà arribar l'usuari navegant pel menú). Les opcions que es troben a un nivell tindran correspondència entre elles. Són utilitzats de manera habitual en sistemes operatius i aplicacions de tota índole.
- **Menús contextuais:** no es trobaran visibles a la interfície fins que l'usuari no en provoqui l'activació. Són menús que s'obriran en una nova finestra flotant, que variaran en funció de la ubicació del ratolí en el moment de cridar-los (normalment fent clic amb el botó dret del ratolí). D'aquesta manera, serà diferent el menú que apareixerà si es fa clic a sobre del marc de la finestra o si es fa clic a sobre de l'àrea de treball.
- **Menús de navegació (*scrolls*):** es poden considerar una evolució dels menús jeràrquics. Es tracta d'un tipus de menú que mostra només les opcions més utilitzades dels menús, i n'oculta la resta, però deixant visible una petita icona que, en ser seleccionada, mostra tota la resta de les opcions del menú. Això ens dóna la sensació de poder navegar pel menú i els seus ítems. Aquest tipus de menú es fa servir quan són moltes les opcions que es volen mostrar en una interfície d'usuari.

Hi ha altres tipus de menús. Un que caldrà tenir en compte per la seva importància és el **menú d'inici**. Aquest menú el va desenvolupar Microsoft per al sistema operatiu Windows 95 i ha evolucionat fins avui dia en tota la família de sistemes operatius de Windows i en altres sistemes operatius que l'han adaptat, com, per exemple, GNU/Linux. És tracta d'un menú jeràrquic que ofereix l'accés a moltes de les funcionalitat i opcions de configuració del sistema operatiu, i arriba a la majoria dels seus elements i aplicacions.

Pestanyes

Les pestanyes de propietats són un element que permet mostrar un conjunt de dades o de funcionalitats relacionades entre si de manera agrupada. En una mateixa finestra es podran mostrar els continguts de tantes finestres com es vulgui, separant-les per pestanyes, que hauran de tenir un títol i contenir informacions relacionades. Se'n pot veure un exemple a la figura 1.15.

FIGURA 1.15. Exemple de pestanyes de propietats



Barres d'eines

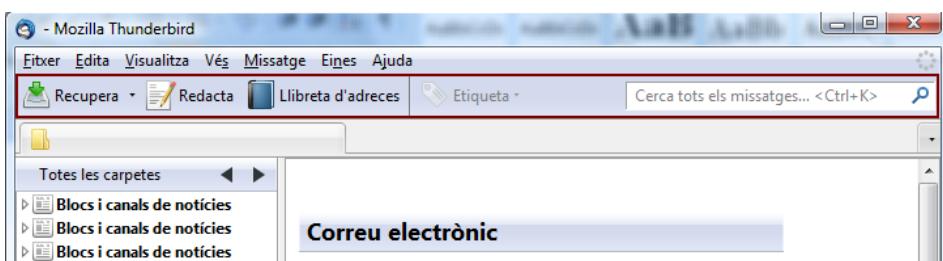
Les barres d'eines faciliten l'accés a algunes funcionalitats mitjançant una icona. Totes les funcionalitats oferides a les barres d'eines es podran trobar desenvolupades als menús, però no totes les opcions que ofereixen els menús es podran trobar a les barres d'eines.

Una vegada seleccionades aquelles funcionalitats que es faran servir més habitualment s'identificarà cada una amb un símbol que representi aquella funcionalitat. Les combinacions de tecles són un altre mètode per poder accedir a aquestes opcions dels menús.

D'aquesta manera es limitarà a una sola acció (un únic clic) l'accés a les principals funcionalitats, i això les fa molt més senzilles i accessibles.

A la figura 1.16 es pot veure un exemple d'una barra d'eines.

FIGURA 1.16. Exemple de barra d'eines, del Mozilla Thunderbird



Icônes

Les icônes són imatges que representen funcionalitats o accions que es podran dur a terme fent un clic a sobre. Entre d'altres llocs, a les interfícies gràfiques d'usuari, són elements que es fan servir a les barres d'eines.

Podran identificar també fitxers, carpetes, altres aplicacions o dispositius d'un sistema informàtic. Cal que les icônes siguin molt fàcilment identifiables per part dels usuaris, o, en el seu defecte, que estiguin molt ben informats de les seves funcionalitats. En cas contrari l'usuari n'haurà de memoritzar el significat, cosa que no compliria un dels principis de les GUI, el de no obligar a memoritzar informació a l'usuari per usar-les correctament.

Es tracta d'un element molt important en les interfícies d'usuari, ja que farà més ràpid, atractiu i senzill l'ús de les aplicacions més que si les informacions que s'ofereixen a l'usuari són en format text. A més, en el cas d'aplicacions internacionals, les icônes poden ser les mateixes en diferents cultures i, en canvi, les traduccions poden ser mal enteses algunes vegades.

Entorn de treball

L'entorn de treball s'ha de considerar com un element més d'una interfície gràfica d'usuari. És un element més de les interfícies, com ho són els menús, les barres d'eines o molts altres.

Es tracta de la ubicació principal de les informacions amb les quals s'interactua amb l'usuari, on es mostrerà el text, les imatges o les dades que l'usuari haurà demanat mitjançant els menús.

A la figura 1.17 es pot veure un exemple d'un entorn de treball amb el sistema operatiu Mac OS.

FIGURA 1.17. Exemple d'un entorn de treball, de Mac



Tipografies

El text que es farà servir en la comunicació amb els usuaris disposarà de diverses opcions tipogràfiques per ser mostrat. L'elecció acurada d'aquestes tipografies també oferirà un missatge específic a l'usuari i és un element més a tenir en compte en el disseny i desenvolupament de les interfícies.

Es coneixen com a *tipografia digital* els tipus de lletra desenvolupats exclusivament per a pantalles, per usar-los en interfícies gràfiques d'aplicacions informàtiques. D'aquesta manera, Microsoft té un sistema propi de tipus desenvolupat, tant per al seu sistema operatiu com per a les aplicacions que funcionaran a sobre. Apple també ha dedicat un esforç important a aquest tema, i va desenvolupar les primeres fonts exclusives per a pantalles de sistemes informàtics.

Altres característiques que cal tenir en compte de les tipografies digitals seran la intensitat i els colors utilitzats, i també les diferents mides de lletra. Una elecció correcta de totes aquestes variables pot influir molt en l'apreciació per part de l'usuari de la interfície desenvolupada.

Controls

Hi ha molts tipus de components que es poden fer servir en les interfícies gràfiques d'usuari. Un tipus de components són els controls. Aquests proporcionen funcions a la interfície d'usuari que permetran moltes més possibilitats en les interaccions entre interfície i usuari.

Un tipus de control són els botons, objectes de control que donen l'opció d'introduir una dada de confirmació al sistema. Hi ha diferents tipus de botons:

- **Botons en forma de radi (radio buttons):** són elements que es fan servir en formularis o en menús per donar a seleccionar a l'usuari una opció entre una llista. Aquests botons en forma de radi són botons rodons que podran ser escollits per mitjà d'accions d'usuari.
- **Botons de confirmació (check box):** botons de forma quadrada molt semblants als de radi, també per poder seleccionar opcions d'una llista d'opcions. La diferència amb els botons en forma de radi serà que en els botons de confirmació es podrà seleccionar més d'una opció entre les mostrades, i en canvi en els botons en forma de radi només se'n pot seleccionar una.
- **Botons en relleu:** aquest tipus de botons imita un botó d'un dispositiu físic que simula un volum, i dóna així la possibilitat de tenir diversos estats (activat o no, seleccionat o no), a més de poder incloure text amb la definició de la funcionalitat que representarà.

A més a més dels botons hi ha molts altres tipus de components, com poden ser els elements d'entrada de text o els elements d'informació de sortida, com la barra de progrés o la barra d'estat, o els elements compostos, com les barres de tasques o el *combo box*.

1.1.5 Eines de propietat i eines lliures d'edició d'interfícies

El desenvolupament d'una interfície gràfica d'usuari despèndrà molt del llenguatge de programació que es faci servir i del seu entorn de desenvolupament.

Serà molt diferent (i es necessitaran eines de programari diferents) desenvolupar una aplicació informàtica amb un llenguatge de programació de **tercera generació**, en què es treballarà amb biblioteques per poder tenir accés a un entorn gràfic, o treballar amb un llenguatge de **quarta generació**, amb components gràfics i d'accés a dades incorporats.

Una altra diferència important serà desenvolupar una aplicació per a un entorn Web o desenvolupar-la per un entorn de finestres (Winforms). Els components, els llenguatges, la metodologia de treball i les possibilitats seran diferents en tots dos casos.

Generacions de llenguatges

Llenguatges de tercera generació:
C, C++, C#, Java, Delphi, ...

Llenguatges de quarta generació:
Visual Basic .NET, PL-SQL,
PHP, ASP, ...

Les eines de creació i edició d'interfícies són un tipus de programari que ajudarà al programador a desenvolupar interfícies que segueixin les indicacions establertes en la fase de disseny a partir de l'anàlisi de les necessitats. Hauran de tenir en compte totes les fases de desenvolupament d'un projecte informàtic.

Aquest tipus de programari haurà de tenir en compte moltes disciplines, ja que la creació d'una interfície d'usuari engloba conceptes de disseny gràfic, de lingüística, d'ergonomia, de sociologia, de programació i tecnologia informàtica i de psicologia cognitiva, entre d'altres.

Hi ha moltes eines diferents al mercat en funció del llenguatge de programació que es faci servir, en funció de l'entorn de desenvolupament o de si l'eina és de propietat o és lliure.

Per aconseguir desenvolupar de manera senzilla les interfícies, les eines incorporen diversos elements preprogramats que faciliten molt la creació d'interfícies gràfiques complexes d'una manera senzilla. Així serà molt senzill afegir a una interfície un menú o una icona, tant senzill com desplaçar aquest element al panel principal on es desenvolupa la interfície, i modificar posteriorment les seves propietats. Aquests elements poden ser per fomentar la iteració amb l'usuari (ajudes, actualització de dades, accions sobre cursors...) o elements per configurar les interfícies (icones, finestres, controladors de dispositius, menús...). A més, es poden gestionar els possibles errors per part de l'usuari en fer servir la interfície, processar excepcions, gestionar i controlar els seus accessos i gestionar els dispositius d'entrada.

Altres característiques que cal tenir en compte de les eines per a la creació i edició d'interfícies gràfiques és que aquestes són tipus d'eines del tipus WYSIWYG, és a dir, és podran observar de manera directa els canvis en el disseny i desenvolupament de les interfícies. El que fan és el que veurà l'usuari final de les aplicacions, i es poden veure els resultats de manera immediata.

A continuació s'enumeren algunes de les moltes eines de desenvolupament d'interfícies existents:

En la secció Activitats del web del mòdul podreu trobar alguns exemples d'interfícies desenvolupades amb diferents eines, tant lliures com de propietat, per dissenyar, desenvolupar i editar interfícies gràfiques d'usuari.

- **Microsoft Visual Studio.** Es tracta d'un entorn de desenvolupament integrat desenvolupat per Microsoft per a sistemes operatius Windows. És de pagament. Suporta els llenguatges de programació següents: Visual C++, Visual C#, Visual J#, ASP.NET i Visual Basic .NET.
- **Eclipse.** Desenvolupat inicialment per IBM. Actualment desenvolupat per una fundació sense ànim de lucre, cosa que fa l'eina gratuïta i de codi obert. Suporta, entre d'altres, els llenguatges Java, C++, Perl i PHP. És multiplataforma.
- **IDE NetBeans.** Desenvolupat inicialment per Sun, que actualment patrocinà els projectes de desenvolupament. Eina de codi obert i gratuïta. Suporta, entre d'altres, els llenguatges Java, JSP, C++ i PHP. És multiplataforma.
- **MonoDevelop.** Entorn de desenvolupament gratuït i de codi obert. Actualment patrocinat per Novell. Dissenyat per a C#, també suporta Java o Boo. És multiplataforma.
- **SharpDevelop.** Entorn de desenvolupament de codi obert. Per als llenguatges de programació C#, Visual Basic .NET i Boo. Per a sistemes operatius

Windows. Per a desenvolupadors que no volen o no poden fer servir el Microsoft Visual Studio.

1.2 Eines de disseny d'Interfícies

Les interfícies gràfiques estan plenes d'elements que ofereixen moltes possibilitats per interactuar amb els usuaris. Però cal donar-hi un cop d'ull des del punt de vista dels desenvolupadors d'interfícies gràfiques. Com podrà un desenvolupador escollir quines són les icones que voldrà fer servir per identificar diferents accions? Com seleccionarà la manera més adequada d'interactuar en cada moment? Quines són les eines que podran servir per aconseguir aquests objectius?

Fins no fa gaires anys desenvolupar una interfície gràfica d'usuari era una tasca força complexa. Actualment hi ha moltes eines que automatitzen aquest procés, i fan, fins i tot, propostes automàtiques d'interfícies a partir de les taules d'una base de dades. Amb petites modificacions el desenvolupador disposarà d'una interfície que acompleixi els objectius marcats.

Però encara hi ha la possibilitat de programar com es feia fa anys, utilitzant la programació per codi, de manera parcial o completa. És molt i molt difícil trobar, avui dia, gent que encara desenvolupi interfícies gràfiques d'usuari a partir d'un llenguatge de programació. En canvi, si que és més habitual fer servir eines de disseny d'interfícies i, a partir d'ubicar manualment els elements a les finestres, manipular posteriorment el codi que configurarà i gestionarà les accions d'aquests elements. La programació visual consisteix a dissenyar les interfícies col·locant els elements als formularis i configurant les seves propietats i accions sense haver d'implementar ni una línia de codi.

Moltes d'aquestes eines de disseny i desenvolupament d'interfícies d'usuari tenen alguns elements comuns (o bé molt semblants), com pot ser disposar d'una àrea de disseny, una paleta de components, editors de propietats o diferents contenidors.

Per explicar alguns d'aquests elements que són similars en les seves característiques i en el seu ús es farà servir com a eina per als exemples la versió 2010 del programari Visual Studio, de Microsoft.

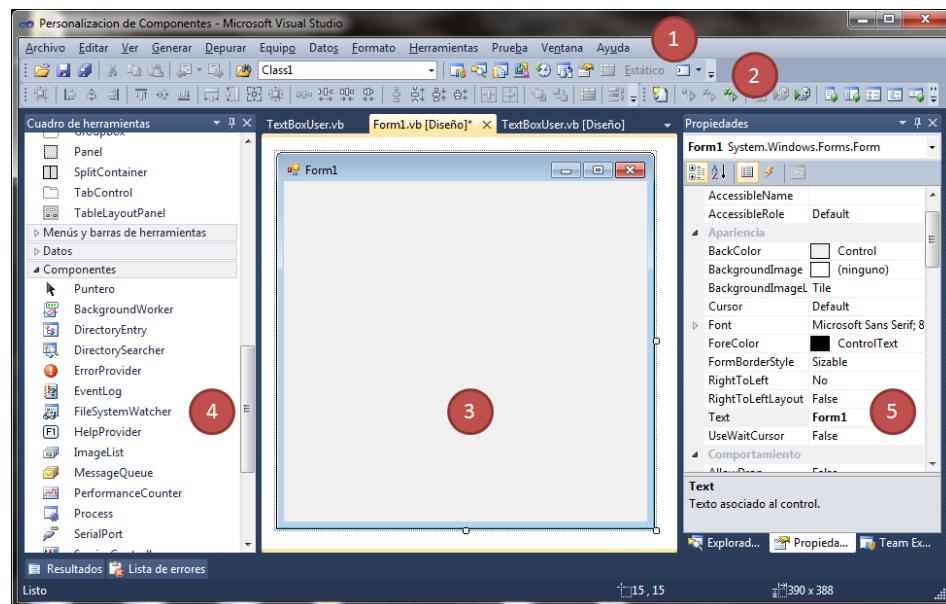
1.2.1 Elements de les eines de desenvolupament d'interfícies

En iniciar el treball amb una eina de disseny i desenvolupament d'interfícies gràfiques cal identificar i entendre molt bé l'entorn de treball, és a dir, sota quin sistema operatiu i condicions s'executarà.

En la figura 1.18 es pot veure l'entorn de treball del Visual Studio 2010 en iniciar-lo amb un projecte en Visual Basic per a una aplicació desenvolupada amb Windows Forms. Aquest entorn de treball és molt similar al que es pot trobar en qualsevol altra eina de desenvolupament d'interfícies, com Eclipse o Mono.

En el moment d'escollir la possibilitat de treballar amb un projecte que permet crear interfícies gràfiques el desenvolupador disposarà de diferents elements que facilitaran molt la seva tasca. Com es pot veure a la figura, a la part central es troba **l'àrea de disseny**, on quedarà desenvolupada la interfície que es dissenyarà. Aquesta àrea és un espai rectangular dintre de la qual s'aniran afegint els elements de la interfície gràfica d'usuari, com poden ser text, imatges o components més complexos com controls o altres panells. L'àrea de disseny disposa de pestanyes que permetran treballar amb més d'una interfície a la vegada. A l'esquerra es disposa de la finestra amb el **quadre d'eines** i a dreta hi ha la finestra amb les **propietats del component** amb el qual s'està treballant en un moment determinat.

FIGURA 1.18. Eina Visual Studio 2010



Cal destacar també d'aquest entorn de treball la part superior, amb el menú i les barres d'eines, que permeten accedir a les funcionalitats que ofereix l'entorn de desenvolupament d'aplicacions i interfícies.

D'aquesta manera, com a resum, es poden identificar els elements següents en l'entorn de treball d'una aplicació de disseny o de desenvolupament d'interfícies d'usuari:

- **Menús.** És on es localitzaran totes les funcionalitats que ofereix el programari. Es podran definir accions per a un conjunt de tecles i configurar quines funcionalitat estaran més accessibles. A la figura 1.18 es pot veure representat pel número 1.
- **Menú de barra d'eines.** Conjunt d'ícones, a la part superior de la interfície, normalment just a sota del menú. Cada icona representa una acció per desenvolupar, a la qual també es podria arribar per mitjà del menú en format de text. A la figura 1.18 es pot veure representat pel número 2.
- **Àrea de disseny.** Ocupa la part central. Contindrà els plafons on s'aniran ubicant els elements que es volen afegir a la interfície. Disposarà de

pestanyes per poder treballar amb més d'un plafó o interfície a la vegada. A la figura 1.18 es pot veure representat pel número 3.

- **Quadre d'eines.** Conté tots els elements que podrem afegir als plafons que s'estiguin desenvolupant a l'àrea de disseny. Seleccionant i arrossegant cada element a l'àrea de disseny es podrà dissenyar la interfície que volem. A la figura 1.18 es pot veure representat pel número 4.

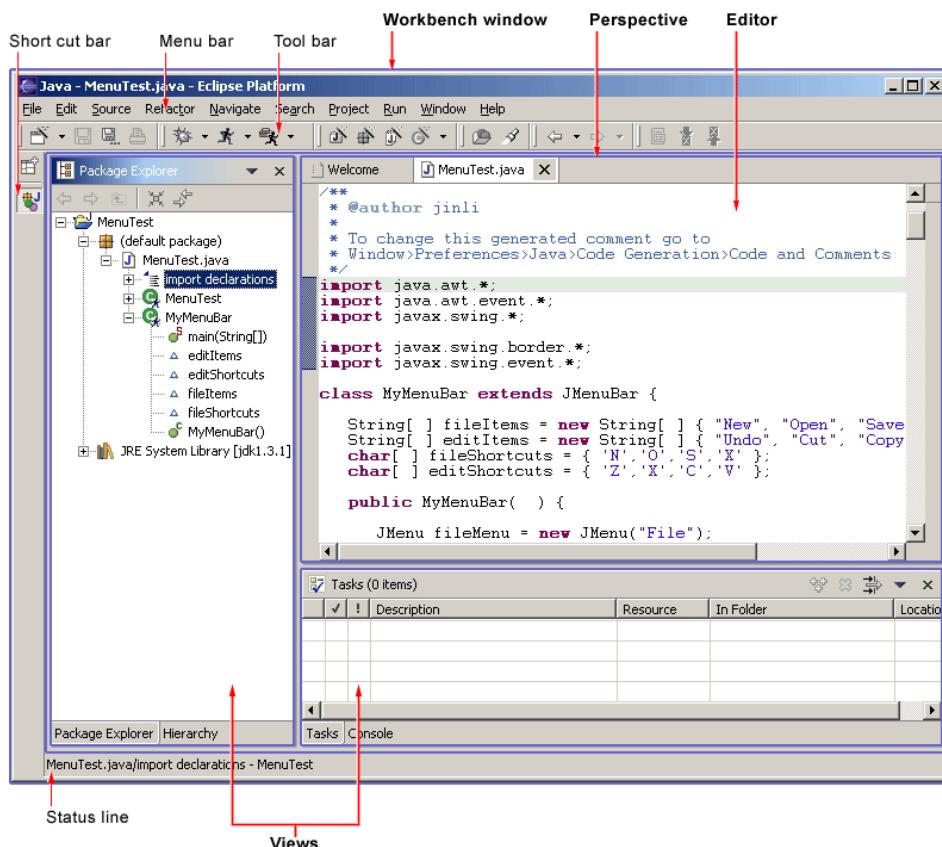
- **Full de propietats.** És una finestra a la part dreta de l'àrea de disseny que mostra les propietats o característiques de l'element que es tingui seleccionat. Aquests valors de les propietats es podran modificar segons interessi per aconseguir l'acompliment dels objectius del disseny de les interfícies. A la figura 1.18 es pot veure representat pel número 5.

- **Assistent.** Ofereixen al desenvolupador de la interfície la possibilitat de crear una interfície a partir de plantilles o d'una sèrie de quadres de diàleg que aniran preguntant les característiques que vol que tingui per oferir-li una proposta d'interfície pràcticament definitiva.

De fet, tots aquests elements que componen una aplicació de desenvolupament d'interfícies són els mateixos que es poden fer servir per crear interfícies per a les aplicacions per desenvolupar.

Un altre exemple d'IDE és l'Eclipse, desenvolupat per IBM per a desenvolupadors de Java. Com es pot veure a la figura 1.19, la seva interfície és molt semblant a la de Visual Studio, pràcticament amb els mateixos components.

FIGURA 1.19. IDE Eclipse



IDE

Acrònim de l'àngles *integrated development environment*, és a dir, 'entorn de desenvolupament integrat'.

1.2.2 Components: característiques i camps d'aplicació

Quan es parla d'interfícies d'usuari normalment és fa servir un vocabulari que inclou paraules com *contenidors*, *controls* o *components*. Cal diferenciar entre aquests termes per poder entendre què serà el més necessari en cada cas, en la creació d'una interfície gràfica.

Temps de disseny: Quan s'estan desenvolupant les interfícies d'una aplicació informàtica.

Temps d'execució: Quan l'usuari està executant l'aplicació i hi està interactuant.

Què és un **component**? És tracta d'un objecte que es podrà reutilitzar, que permet interactuar amb altres objectes i proporciona el control sobre determinats recursos externs en temps de disseny. Els components es podran dissenyar i configurar, cosa que fa que es pugui fer servir en una eina tipus IDE. Un component es podrà afegir al quadre d'eines, es podrà seleccionar i arrastrar cap a un plafó o formulari i es podran modificar les seves característiques. Totes aquestes accions es podran dur a terme en temps de disseny. Els components podran ser visibles o no visibles per a l'usuari a la interfície. Un component que es fa no visible per a l'usuari ofereix algunes possibilitats per al desenvolupador que multipliquen les seves possibilitats a l'hora de crear una interfície.

Alguns exemples de components simples: botons, llistes desplegables, barres de progrés, etiquetes, botons de radi... Alguns exemples de components complexos: menús, taules, arbres, quadres de diàleg...

Què és un **control**? Alguns autors consideren un control com un tipus de component. Altres indiquen que són objectes similars als components, en poder-se dissenyar i configurar. El que sí que és clar és que un control és un objecte que proporciona una interfície a l'usuari, i permet la interacció entre usuari i aplicació per intercanviar informació. Alguns exemples de controls poden ser els botons, els quadres de llistes, els quadres d'edició...

Què és un **panell**? És un control que servirà per fer de contenidor d'altres controls. És una eina adequada per ocultar o mostrar un tipus de controls determinats. Aquest tipus de control podrà tenir barres de desplaçament del control a més de poder personalitzar les seves característiques i propietats de presentació. Un tipus de contenidor serà una barra d'eines.

Què és un **formulari**? És un tipus de component que serà com una finestra que el desenvolupador, en el temps de disseny, podrà utilitzar per ubicar altres components o controls. Un formulari pot existir per ell sol o pot formar part d'un conjunt de formularis. Com a exemples de formulari es tenen les finestres amb marc o sense, els quadres de diàleg...

Una vegada aclarits certs conceptes veureu alguns exemples dels components que formen una interfície gràfica d'usuari.

Hi ha molts elements i de molts tipus, que es podran fer servir per desenvolupar interfícies d'usuari. A més, es podran agrupar, crear-ne de nous i intercanviar-los entre diferents eines de disseny d'interfícies. En la taula 1.1 es mostren alguns dels principals elements (components i controls) que podrem trobar iguals o molt semblants en diverses eines de disseny d'interfícies o entorns integrats de desenvolupament de programari.

TAULA 1.1. Alguns elements del disseny d'interfícies

Component	Nom	Descripció
 Label	Label	Ofereix la possibilitat d'ubicar al formulari una etiqueta amb un text determinat, que no podrà ser manipulat per l'usuari de l'aplicació.
 Button	Button	Són botons que executaran una funcionalitat quan es faci clic o doble clic a sobre. Contindran un text indicatiu de l'accio que representen
 PictureBox	PictureBox	Permet seleccionar una imatge o un gràfic d'un arxiu prèviament existent. Aquest arxiu podrà ser dels diferents tipus de fitxers d'imatges que hi ha (jpg, gif, bmp, ico...).
 ProgressBar	Progress Bar	Ofereix la progressió en l'execució d'una funcionalitat determinada que s'està executant (per exemple, l'actualització de moltes dades en una base de dades). No s'ha de confondre amb HScrollBar i VScrollBar.
 RadioButton	RadioButton	Són un conjunt d'opcions agrupades que ofereixen diferents alternatives a l'usuari per poder escollir-ne una. Les opcions que s'ofereixen seran mítuament exclòents. Si es vol poder escollir més d'una opció a la vegada, cal fer servir un CheckBox.
 CheckBox	CheckBox	Alternativa als RadioButtons. Conjunt d'opcions amb un espai en forma de quadrat a la dreta per poder seleccionar-les. Es podrà seleccionar més d'una opció a la vegada.
 ListBox	ListBox	Aquest control mostra les opcions en format de llista. L'usuari podrà seleccionar una o més de les opcions.
 TextBox	TextBox	Pot mostrar informació en format text escrita en temps de disseny i recollir informació introduïda per l'usuari en temps d'execució.
 ComboBox	ComboBox	Combina un control de tipus TextBox i un de tipus ListBox. Ofereix una llista d'opcions, de les quals se'n podrà seleccionar una o bé introduir un text al quadre d'edició.

1.2.3 Finestra de quadre de components

La **finestra del quadre d'eines** conté els components i altres elements que es poden fer servir a les interfícies que es desenvolupen a l'àrea de disseny. Per afegir un component cal seleccionar-lo amb el ratolí i arrossegar-lo fins a l'àrea

En la secció Annexos del web del mòdul podreu trobar exemples de com cal afegir nous components ja existents a la finestra del quadre d'eines i com es poden crear nous grups d'elements.

de disseny dintre d'un formulari.

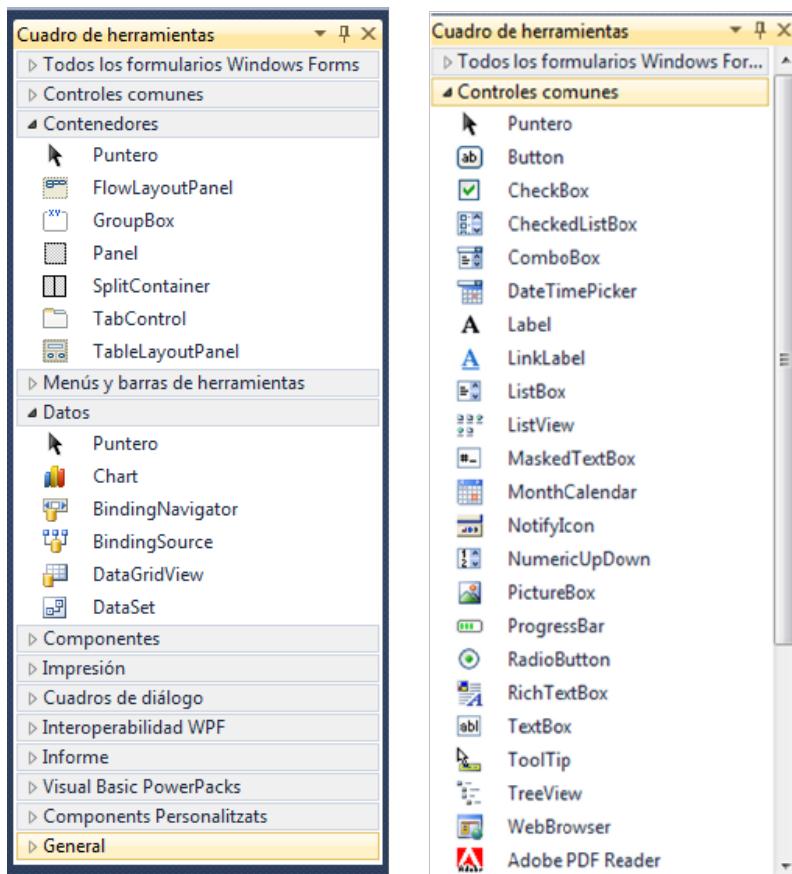
Hi ha diversos grups de controls disponibles; el nombre depèn del tipus de dissenyador actiu en l'editor. Quan s'està dissenyant un formulari de tipus Windows Forms, apareixen les eines necessàries per treballar amb formularis Windows, que seran diferents dels components per treballar amb formularis Web i que, al seu torn, estan agrupats per categories de controls.

A la figura 1.20 (a) es poden observar alguns dels tipus de components més habituals en formularis de tipus Windows Forms. Aquests components s'agrupen en contenidors, menús i barres d'eines, impressió, quadres de diàleg i dades, entre d'altres.

A la figura 1.20 (b) es pot veure el detall d'alguns dels controls més habituals que es fan servir: Els botons, les *checkbox*, les etiquetes, les *listbox*, les imatges...

Aquesta finestra del quadre d'eines serà possible personalitzar-la segons les necessitats del desenvolupador. És possible crear nous grups o afegir components addicionals a un grup existent. Es poden afegir altres components .NET o controls ActiveX dels utilitzats en aplicacions que no són .NET.

FIGURA 1.20. Quadre de Components (a) i Finestra de quadre de controls (b)



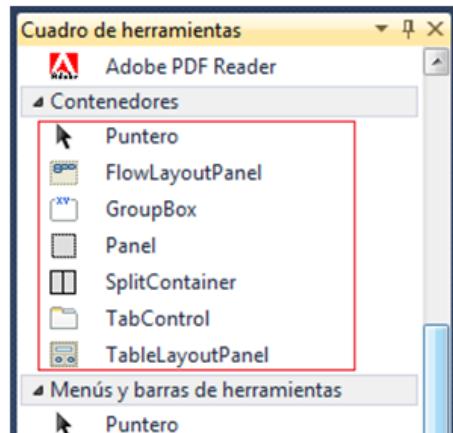
1.2.4 Altres contenidors (panells)

Alguns controls Windows Forms de l'entorn de treball permeten funcionar com a contenidors d'altres controls o components: es tracta dels objectes Form, GroupBox i Panel, que hereten de la classe ContainerControl. Per incloure controls dins d'un contenidor, simplement cal seleccionar un d'aquests elements i dibuixar o arrossegar un control dins del contenidor. A la figura 1.21 es poden observar els contenidors que es troben preinstal·lats en el Visual Studio 2010.

La selecció de diversos controls dins d'un contenidor s'ha de fer sempre utilitzant les tecles *Ctrl* o *Maj*.

Els controls agrupats dins d'un contenidor es mouran, copiaran o s'enganxaran tots junts quan s'apliqui una acció al contenidor. Aquests controls agrupats s'esborraran quan el contenidor s'elimini.

FIGURA 1.21. Altres contenidors



Classe

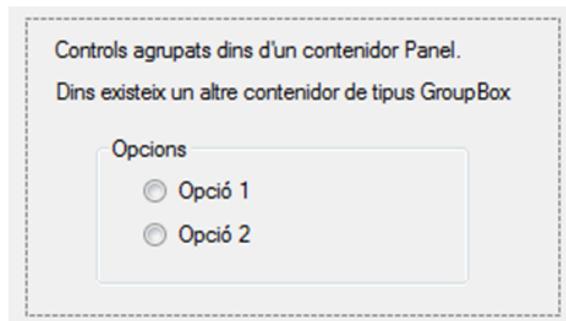
En un context de POO (programació orientada a objectes), una classe és un conjunt de propietats i mètodes relacionats amb un significat propi.

Herència

En un context de POO és un tipus de relació entre classes, en què una deriva d'una segona, cosa que permet la reutilització i extensió del programari. Permet la creació d'una jerarquia de classes.

A la figura 1.22 es pot veure un exemple en el qual tenim un text amb dues opcions creades com a RadioButtons. Tots aquests controls s'han agrupat i s'han convertit en un de sol.

FIGURA 1.22. Controls agrupats



1.2.5 Afegir/eliminar components a la interfície d'usuari

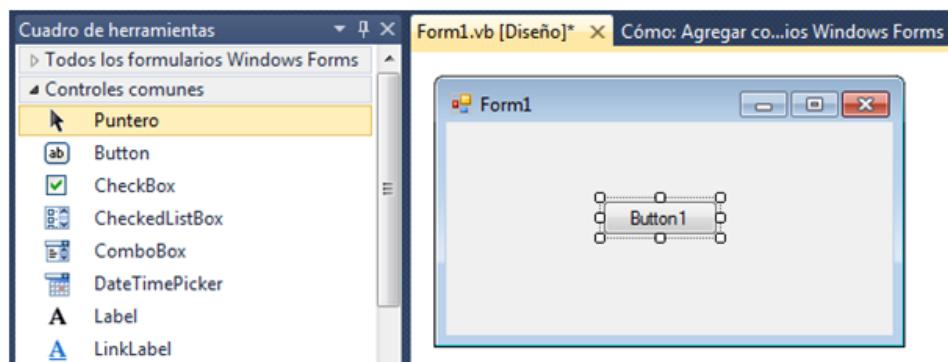
El quadre d'eines permet observar quins són els components que es podran fer servir per afegir-los al formulari que es desenvolupi. En el cas de programació de Windows Forms, la plantilla d'un projecte inicial crea, per defecte, un formulari buit sobre el qual dibuixarem els controls.

La manera d'afegir un nou control a un formulari és:

- Tenir obert un formulari on es vol ubicar el control a l'àrea de disseny.
- Obrir la finestra amb el quadre d'eines.
- Seleccionar el grup i el control o component que volem.
- Fer clic sobre el formulari o arrossegar-lo fins al formulari en el qual es vulgui encabir perquè formi part d'aquella interfície, com es pot observar a la figura 1.23.
- A més, li podrem donar les mides que vulguem (figura 1.23).

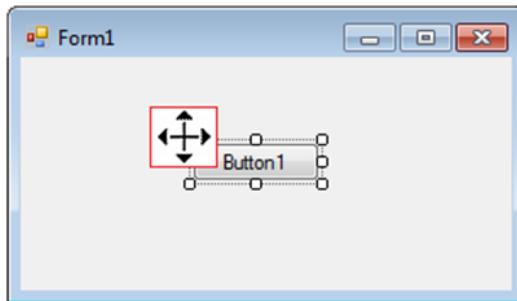
A la figura 1.23 es pot veure com s'afegeix un control de tipus Button al formulari Form1.

FIGURA 1.23. Afegir component



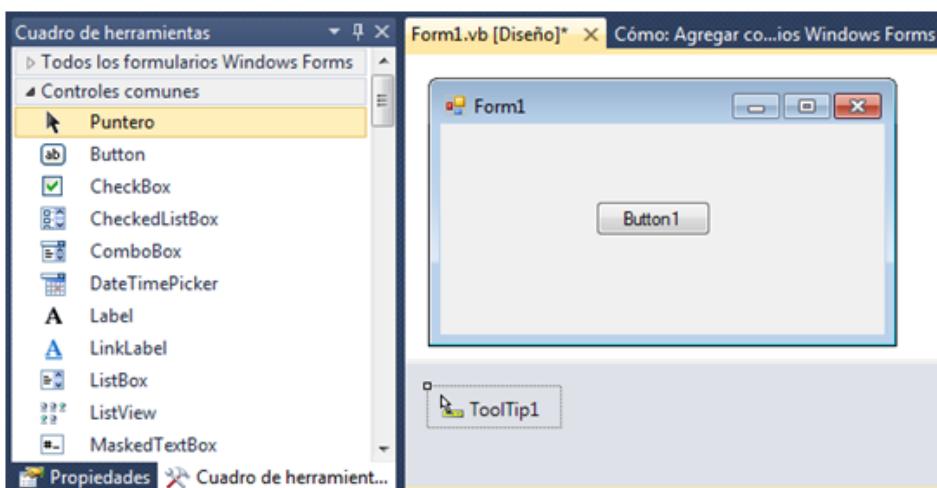
De manera alternativa, és possible incloure un control fent doble clic sobre el control o component que volem a la finestra del quadre d'eines. En aquest cas, si abans de fer el doble clic sobre el control es troba seleccionat un formulari o objecte virtual a l'àrea de disseny, el control es posicionarà a la part superior esquerra d'aquest; si està seleccionat un altre control, el nou control el posicionarà sobre aquest.

Per desplaçar la majoria dels controls simplement cal seleccionar-los i arrossegar amb el ratolí quan el cursor pren la forma d'una creueta. Tanmateix, per desplaçar alguns controls cal utilitzar el tirador que tenen en forma de creu. A la figura 1.24 es pot veure un exemple de l'acció de desplaçament d'un control.

FIGURA 1.24. Desplaçar un component

Per eliminar un control o component ja ubicat en un formulari només cal seleccionar-lo i fer servir la tecla *Supr* per suprimir-lo, i d'aquesta manera es perd el disseny i la configuració de l'objecte.

Hi ha controls que no són visibles. En versions anteriors del Visual Studio, aquests controls també apareixen sobre el formulari. En el Visual Studio .NET, aquests controls (com els quadres de diàleg, el control ToolTip, o el control Timer) es dibuixen sobre el formulari, però apareixen a la safata de components situada sota del formulari.

FIGURA 1.25. Component invisible

A la figura 1.25 es pot veure un exemple de controls invisibles de tipus ToolTip i la manera de treballar-hi.

1.2.6 Ubicació i alineació de components. Modificació de propietats

Una vegada col·locats els components als formularis cal ubicar-los de manera que l'usuari pugui entendre les funcionalitats que ofereixen i disposar-los en una distribució agradable i senzilla. L'entorn de desenvolupament disposa d'eines que permeten donar més precisió a la col·locació dels controls mitjançant les característiques d'alineació, que apareixen en l'opció “Alinear” del menú “Formato” o en la barra d'eines de “Diseño” (es pot seleccionar aquesta barra d'eines o una altra qualsevol fent clic dret en una zona de menús).

La major part d'aquestes opcions d'alignació només es poden utilitzar quan hi ha diversos controls seleccionats. En aquests casos, l'operació es farà sobre l'últim control seleccionat. El primer control seleccionat serà el que actuï com a **control primari** i els altres controls s'alignaran a partir del primer.

Per seleccionar diversos controls es poden anar marcant mentre es manté pulsada la tecla *Ctrl* o *Maj*.

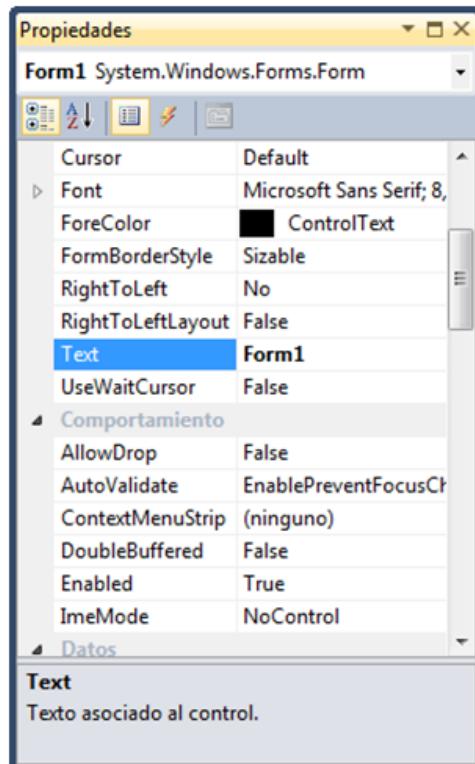
Algunes opcions d'alignació disponibles són:

TAULA 1.2.

Icona	Descripció
	Alinea els controls per sota.
	Centra els controls verticalment.
	Alinea els controls per dalt.
	Alinea els controls a l'esquerra.
	Alinea els controls a la dreta.
	Centra els controls horizontalment.

En la secció Annexos del web del mòdul podreu trobar més exemples i informacions de com cal ubicar i alinear els components en un formulari.

La modificació de les propietats dels components, formularis o controls, per exemple, és una tasca necessària i important per poder configurar aquests objectes segons els objectius del dissenyador de les interfícies. Aquests atributs o propietats es poden modificar mitjançant codi (en temps d'execució) o mitjançant la finestra “Propiedades” (en temps de disseny). La finestra “Propiedades” proporciona un accés a les propietats accessibles en temps de disseny. Com es pot veure a la figura 1.26, aquesta finestra mostra a la part superior una llista de les classes disponibles en un entorn donat (per exemple, si estem en el dissenyador d'un formulari, només mostrarà els controls d'aquest formulari) i a sota una llista de les propietats de la classe, tot indicant aquelles que es poden modificar i les que no, i marcant aquestes últimes en gris.

FIGURA 1.26. Propietats d'un component

Per modificar les propietats d'un control en temps de disseny s'utilitza la finestra “Propiedades”. Primer caldria seleccionar el control de la llista que apareix a la part superior de la finestra o en el formulari mateix, de manera que apareixerien a la finestra totes les propietats de lectura i escriptura modificables en temps de disseny (hi ha altres propietats que només es poden modificar en temps d'execució).

També és possible modificar les propietats de diversos controls, seleccionant-los en el formulari, de manera que a la finestra “Propiedades” apareguin les propietats comunes a tots.

En la secció Annexos del web del mòdul podreu trobar més exemples i informacions sobre les propietats dels components, i també de com modificar-les i altres característiques específiques.

1.2.7 Biblioteques de components disponibles per a diferents sistemes operatius i llenguatges de programació

Els elements (components, controls...) dels quals pot disposar un desenvolupador d'interfícies gràfiques per al disseny facilitaran molt la feina. Això es pot fer aprofitant elements ja existents i que no cal tornar a implementar, i també creant elements amb unes determinades propietats que segueixin un patró semblant o corporatiu, per fer servir només determinats elements propis en el desenvolupament d'interfícies per a una determinada organització.

Aquesta situació porta a molts desenvolupadors a crear components i deixar-los a l'abast d'altres desenvolupadors (ja sigui de manera gratuïta o no). Igual que hi ha molts exemples desenvolupats en qualsevol llenguatge de programació que estan a l'abast dels desenvolupadors o moltes biblioteques amb funcions de codi o exemples, el mateix succeeix amb les biblioteques de components.

En la secció Annexos del web del mòdul podreu trobar algunes referències concretes a biblioteques de components.

Aquests components poden ser d'ús genèric o d'utilització molt concreta per part de determinats desenvolupadors que cerquen una funcionalitat concreta. Poden estar disponibles per a un determinat llenguatge de programació o per a una aplicació determinada o un entorn integrat de desenvolupament d'interfícies. I podran ser vinculats amb determinades funcionalitats (accés a determinades bases de dades, components gràfics per identificar components concrets d'un determinat àmbit...).

Per exemple, hi ha biblioteques de components diferents per a eines de desenvolupament d'interfícies en entorn Web i per a eines de desenvolupament d'interfícies en entorn WinForms.

Widget

El mot *widget* correspon a una abreviatura de les paraules *window* i *gadget*, és a dir, 'finestra' i 'dispositiu'. És una aplicació accessible des d'una finestra petita. Són molt populars en entorns Mac com a accessos a les aplicacions d'ús més freqüent.

Una altra manera d'anomenar als components o controls gràfics és *widgets* (ginys), sempre dins un context de desenvolupament d'aplicacions visuals amb interfícies gràfiques d'usuari. Un giny és un component o control visual que s'haurà programat i es podrà reutilitzar per desenvolupar altres aplicacions o per a altres programadors.

Hi ha ginys per desenvolupar en entorns de sistemes operatius diferents (Microsoft, Mac, Linux...). Els ginys poden ser virtuals o físics, per diferenciar, per exemple, quan es faci referència a un botó físic que podrà polsar l'usuari, o quan ens referim a un botó virtual d'una interfície gràfica, que es podrà seleccionar amb un clic de ratolí.

Els ginys també seran específics de l'entorn de programació, ja sigui que es desenvolupi per a un entorn Web o per a un entorn WinForms.

1.3 Característiques específiques de les eines de disseny d'interfícies

Certs elements d'una interfície gràfica d'usuari es poden utilitzar en diversos àmbits o aplicacions no directament relacionades amb el disseny de GUI.

Altres, en canvi, estan directament relacionats amb les característiques específiques i les necessitats de les interfícies gràfiques d'usuaris. Els esdeveniments, per exemple, permeten associar determinades funcionalitats al fet que succeeixin determinades accions o esdeveniments entre l'usuari i l'aplicació amb la qual treballa.

Una altra característica específica és vincular determinats controls a bases de dades, cosa que permet consultar, modificar, afegir o esborrar dades a un conjunt de dades determinat.

Altres temes per tractar en aquest apartat estan relacionats amb els diàlegs modals i no modals i l'edició de codi generat per les eines de disseny i desenvolupament d'interfícies gràfiques d'usuari.

1.3.1 Controls: classes, propietats i mètodes

Els controls són objectes que permeten un desenvolupament de les interfícies més senzill gràcies a la seva possibilitat de modularitat i reutilització. Els controls tenen propietats i mètodes, i es podran associar a esdeveniments.

Els controls ofereixen un ventall de possibilitats molt gran als desenvolupadors, en possibilitar dotar d'unes mateixes característiques a un conjunt de controls que es faran servir per a interfícies directament relacionades o poder utilitzar un mateix control en diversos projectes de desenvolupament de programari.

Avui dia, es faci servir l'aplicació que es faci servir en un entorn visual amb interfícies gràfiques d'usuari, trobarem un ús generalitzat dels controls. Si es vol obrir un document o una fotografia es fa servir un control que mostra un quadre de diàleg que permet navegar pel nostre sistema de carpetes fins a trobar l'arxiu que busquem. O si es finalitza l'execució d'una aplicació moltes vegades cal interactuar amb una finestra que demana si es vol desar la feina feta.

Tots aquests controls ja estan creats i no caldrà programar-los de nou, només cal fer servir l'aplicació informàtica adequada per dissenyar i desenvolupar la interfície o cercar el control en una de les moltes biblioteques al nostre abast. Una vegada escollit el control que interessa al programador, aquest només haurà de modificar característiques com la mida o el color i afegir-lo al formulari apropiat, associant el codi necessari perquè el control es comporti com interessi quan s'executi l'aplicació.

Els controls disposen de **tres característiques importants**:

- Propietats
- Mètodes (procediments i funcions)
- Esdeveniments

Les **propietats** són les característiques que identifiquen cada control. Aquests són el que són a partir de les parts que constitueixen l'objecte. Per exemple, una persona es podria dir que presenta propietats com podrien ser els seus ulls, les orelles, els llavis, l'alçada, el pes... Aquestes propietats poden tenir uns **valors** (per exemple Ulls = Verds, Orelles = Grans, Llavis = Fins, etc.) que faran que aquella persona sigui identifiable respecte d'una altra per aquelles característiques. De la mateixa manera passa amb els controls.

Un exemple de control: TextBox

Mitjançant aquest control podrem fer tant l'entrada com la sortida de dades en les nostres aplicacions.

Algunes de les propietats de què disposa el control són:

- **Text:** S'indica el text que apareixerà en el control. Podem assignar qualsevol text en temps de disseny o execució

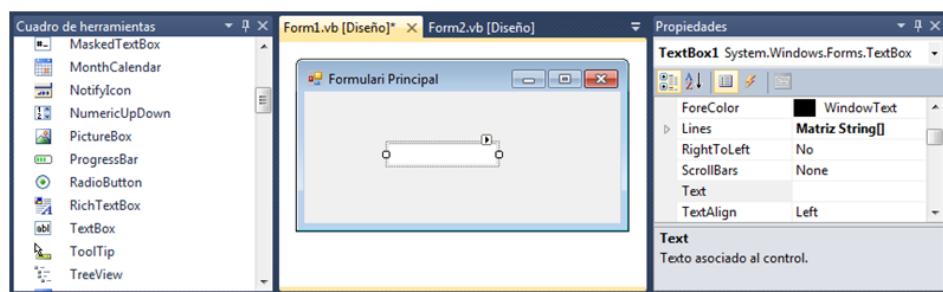
Objecte

En un context de programació orientada a objectes, un objecte és una instància a una classe.

- **Name:** Aquesta propietat la tenen tots els controls: és el nom que ve per defecte, en aquest cas Text1, i és el nom amb què es coneixerà el control quan l'utilitzem en el codi. En un mateix formulari no hi pot haver dos controls amb el mateix nom. Convé posar un nom que representi la funció que té el control en l'aplicació perquè el codi quedi més clar. Exemple, si en el TextBox volem introduir la direcció d'una persona podem assignar a aquesta propietat el valor TextBox_Direcció.
- **Multilínies:** Permet que introduïm diverses línies de text en el control en lloc de només una.
- **Alignment:** Alineació que tindrà el text dins del control: esquerra, centre o dreta. Perquè funcioni la propietat multilínies ha d'estar amb el valor *true*.
- **Locked:** Si el seu valor és *true* bloqueja el control, és a dir, l'usuari no pot introduir ni modificar el text que contingui. Pot servir per utilitzar el control com a sortida de dades sense que l'usuari el pugui modificar per error.

Les propietats del control poden ser modificades en temps de disseny. Com es pot observar a la figura 1.27, una vegada seleccionat el control, podrem accedir a les seves propietats a la finestra que surt a la dreta, visualitzant totes les seves opcions i valors.

FIGURA 1.27. Exemple de propietats



Però les propietats també poden ser modificades en temps d'execució, col·locant el seu nom i el nom de la propietat, com per exemple:

```
1 Text1.Text = "HOLA"
```

Els controls segueixen el principi de l'encapsulació; això significa que es pot tenir un control 50 vegades en un formulari, i si li canviem una propietat a un control només es canvia en aquest i no en els 49 controls restants, és a dir, cada control manté encapsulades les seves propietats.

Els **mètodes** són els procediments o funcions associats a un control, els quals permeten fer certes operacions útils sobre aquest control i obtenir-ne una resposta.

Per exemple, un mètode d'un RadioButton permet ordenar alfabèticament els seus components o bé un altre mètode permet fer una cerca d'una dada concreta.

Els procediments i les funcions són molt similars; la diferència és que **els procediments no retornen cap valor**, mentre que **les funcions sempre retornen un sol valor**.

Un exemple, en codi de Visual Basic, és:

Mètode:

```
1 Sub Nom_del_Metode (Paràmetres)
2     Línies de codi
3 End sub
```

Funcions:

```
1 Function Nom_de_la_Funció (Paràmetres) es Valor_per_retornar
2     Línies de codi
3     Nom_de_la_Funció = valor
4 End function
```

Un **esdeveniment** és una acció determinada que es pot fer vinculada amb un control.

Per exemple, si sobre un control de tipus Button es fa un clic amb el ratolí s'executarà una determinada acció. Es pot dir que l'esdeveniment “clic sobre el control Button” activa una certa funcionalitat. Altres exemples d'esdeveniments poden ser tancar un formulari, seleccionar una opció d'una llista o modificar el contingut d'un control que conté dades.

1.3.2 Diàlegs modals i no modals

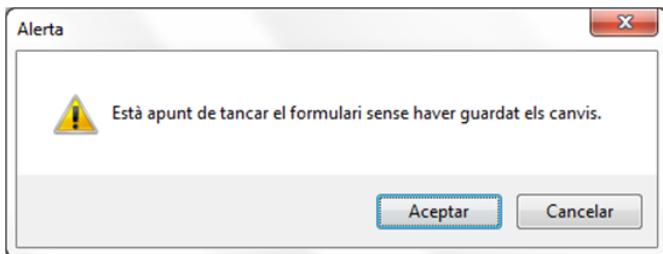
Un quadre de diàleg és una finestra que apareixerà oferint una informació als usuaris en un moment determinat de l'execució d'una interfície.

Un quadre de diàleg és **modal** quan l'aplicació mateixa o el formulari que l'ha cridat no pot rebre cap esdeveniment fins que no s'hagi tancat el quadre de diàleg.

Un quadre de diàleg és **no modal** en cas contrari, quan l'aparició del quadre de diàleg no limita l'usuari de continuar interactuant amb la resta de finestres. En altres paraules, els quadres de diàleg modals són aquells formularis que fins que no es tanquen no deixen continuar amb l'aplicació.

A continuació es mostren tres exemples de diàlegs modals.

Un **quadre de missatge** és un quadre de diàleg modal que es pot utilitzar per mostrar informació textual i permetre que els usuaris prenguin decisions amb botons. La figura 1.28 mostra un quadre de missatge que mostra informació textual, exposa un missatge i proporciona a l'usuari dos botons per respondre al missatge.

FIGURA 1.28. Exemple de quadre de missatge

Per crear un quadre de missatge, s'utilitza la classe MessageBox. A continuació es mostra el codi que crearia el missatge que es mostra en la figura 1.28.

```

1  MessageBox.Show (
2      "Esteu apunt de tancar el formulari sense haver desat els canvis.", _
3      "Alerta", _
4      MessageBoxButtons.OKCancel, _
5      MessageBoxIcon.Warning )

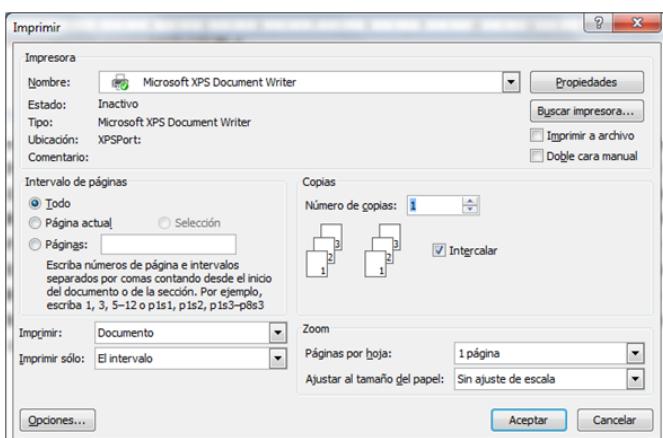
```

En la secció Annexos del web del mòdul podreu trobar més exemples i informacions de com cal crear un quadre de diàleg modal i un de no modal.

Un altre tipus de quadres de diàlegs són els **quadres de diàlegs comuns**, que poden ser tan modals com no modals. L'entorn Windows implementa diversos quadres de diàleg reutilitzables que són comuns per a totes les aplicacions, incloent-hi quadres de diàleg per obrir arxius, desar arxius i imprimir.

Com que és el sistema operatiu el que implementa aquests quadres de diàleg, es poden compartir entre totes les aplicacions que s'executen en el sistema operatiu, fet que contribueix a la coherència de l'experiència de l'usuari; quan els usuaris estan familiaritzats amb l'ús d'un quadre de diàleg proporcionat pel sistema operatiu en una aplicació, no necessiten obtenir informació sobre com cal utilitzar aquest quadre de diàleg en altres aplicacions.

Atès que aquests quadres de diàleg estan disponibles per a totes les aplicacions i que ajuden a proporcionar una experiència d'usuari coherent, es coneixen com a quadres de diàleg comuns. Un exemple de quadre de diàleg seria el creat quan s'intenta imprimir un document. Es mostra a la figura 1.29.

FIGURA 1.29. Exemple d'un quadre de diàleg comú

1.3.3 Enllaç de components a orígens de dades. Datagrid

L'enllaç de dades és el mecanisme proporcionat per algunes plataformes que permeten, en aplicacions amb interfície Windows o en aplicacions web, enllaçar objectes contenidors de dades amb els controls de formulari, per poder fer operacions de navegació i edició.

És possible enllaçar dades (**Databind**) amb els controls de formularis, no solament des d'orígens de dades tradicionals (com DataSets ADO .NET), sinó també a gairebé qualsevol estructura que contingui dades, com estructures, col·leccions, propietats i altres controls.

Hi ha diversos tipus de *data binding* o enllaços a dades, com els *simple data binding* o els *complex data binding*.

Simple data binding

Consisteix en una associació entre un control, que pot mostrar una única dada, i l'objecte que actua com a contenidor de dades, com ara l'enllaç entre un control TextBox o una etiqueta i un objecte DataColumn d'una DataTable d'un DataSet.

Per exemple, podeu vincular el camp *NomColumna* de la taula *NomTaula* a un control de tipus TextBox de la manera següent:

```
1 Me.SqlDataAdapter1.Fill(Me.DataSet1, "NomTaula")
2 Dim bind As Binding
3 bind = New Binding("Text", Me.DataSet1.Tables("NomTaula"), "NomColumna")
4 Me.TextBox1.DataBindings.Add(bind)
```

Tal com s'observa, en l'exemple es fa ús de la classe *Binding*, classe que permet crear un enllaç per a un control indicant alguns paràmetres, en què el primer paràmetre és la propietat de destinació, el segon paràmetre és l'objecte origen que conté totes les dades i el tercer paràmetre és el camp origen en l'objecte origen. Quan hi ha una modificació en la propietat de destinació, el canvi també es fa per a l'objecte origen. Si l'objecte és un DataSet, les dades de la base de dades no són modificades. El DataSet ha de ser sincronitzat amb la base de dades.

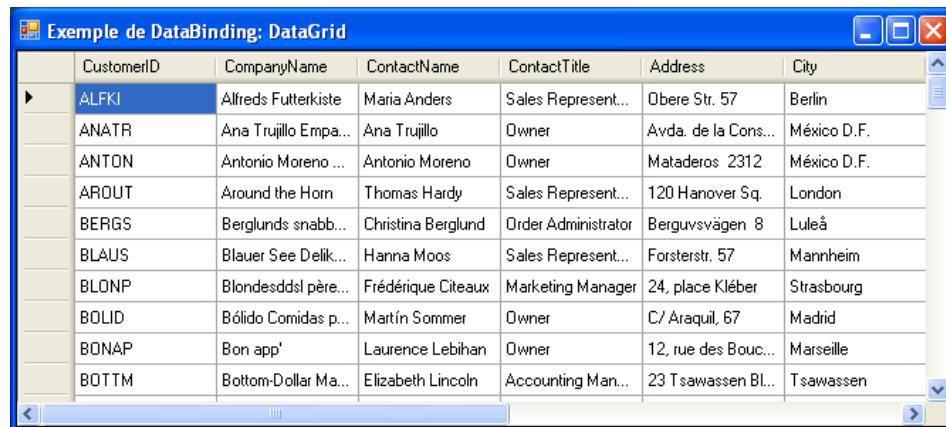
Complex data bindings

Aquest tipus d'enllaç és possible entre un control que pot actuar com a interfície o visualitzador de dades, ja que disposa de la capacitat de mostrar diverses o totes les dades, normalment més d'un registre, de l'objecte contenidor de dades. Això s'utilitza normalment en controls DataGridView (graella), ListBox o ErrorProvider. Un exemple clàssic és l'enllaç entre una graella de dades i un objecte DataTable d'un DataSet. Es pot veure aquest exemple representat a la figura 1.30.

DataSets ADO .NET

Representació d'un conjunt complet de dades (taules, camps, relacions...) que resideix en memòria i ofereix un model de programació independent de l'origen de les dades.

FIGURA 1.30. DataGridView



```

1 Imports System
2 Imports System.Data
3 Imports System.Data.SqlClient
4 Imports System.Windows.Forms
5
6
7 Public Class Form1
8     Inherits System.Windows.Forms.Form
9
10    Private dataGridView1 As New DataGridView()
11    Private bindingSource1 As New BindingSource()
12    Private dataAdapter As New SqlDataAdapter()
13    Private WithEvents RecargarBDButton As New Button()
14    Private WithEvents ActualizarBDButton As New Button()
15
16    <STAThreadAttribute()> _
17    Public Shared Sub Main()
18        Application.Run(New Form1())
19    End Sub
20
21    ' Initialize the form.
22    Public Sub New()
23
24        Me.dataGridView1.Dock = DockStyle.Fill
25        Me.Controls.AddRange(New Control() {Me.dataGridView1, Me.ParentForm})
26        Me.Text = "Exemple de DataBinding: DataGridView"
27
28    End Sub
29
30    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
31        Handles Me.Load
32
33        ' Enllaça el DataGridView a BindingSource i carrega les dades de la BD
34        Me.dataGridView1.DataSource = Me.bindingSource1
35        GetData("select * from Customers")
36
37    End Sub
38
39    Private Sub GetData(ByVal selectCommand As String)
40
41        Try
42            ' Establim la cadena de connexió a la base de dades
43            Dim connectionString As String =
44                "Integrated Security=SSPI;Persist Security Info=False;" + _
45                "Initial Catalog=Northwind;Data Source=localhost"
46
47            ' Creació d'un DataAdapter
48            Me.dataAdapter = New SqlDataAdapter(selectCommand, connectionString
49        )

```

```

50     ' Creació d'un CommandBuilder que ens permetrà fer
51     ' les operació de modificació, inserció i esborrament.
52     Dim commandBuilder As New SqlCommandBuilder(Me.dataAdapter)
53
54     ' Omplim una nova taula de dades i l'enllacem a BindingSource.
55     Dim table As New DataTable()
56     Me.dataAdapter.Fill(table)
57     Me.bindingSource1.DataSource = table
58
59     Catch ex As SqlException
60         MessageBox.Show(ex.Message)
61     End Try
62
63 End Sub
64
65 End Class

```

Si volguéssim actualitzar la informació de la base de dades:

```

1 Private Sub RecargarBDBButton_Click(ByVal sender As Object,
2   ByVal e As System.EventArgs) _
3     Handles RecargarBDBButton.Click
4
5     ' Recarrega la informació de la base de dades
6     GetData(Me.dataAdapter.SelectCommand.CommandText)
7
8 End Sub
9
10 Private Sub ActualitzarBDBButton_Click(ByVal sender As Object,
11   ByVal e As System.EventArgs) _
12   Handles ActualitzarBDBButton.Click
13
14     ' Actualitza la base de dades amb els canvis que ha fet l'usuari
15     Me.dataAdapter.Update(CType(Me.bindingSource1.DataSource, DataTable))
16
17 End Sub

```

ADO.NET és un conjunt de classes que exposen serveis d'accés a dades per al programador de .NET, constitueixen una part essencial de l'entorn de treball .NET, i permeten l'accés a dades relacionals o XML, entre d'altres.

En realitat ADO.NET és una evolució del model d'accés a dades d'ADO que utilitza alguns objectes d'ADO, com Connection i Command, i també afegeix objectes nous, com per exemple DataSet, DataAdapter, DataReader.

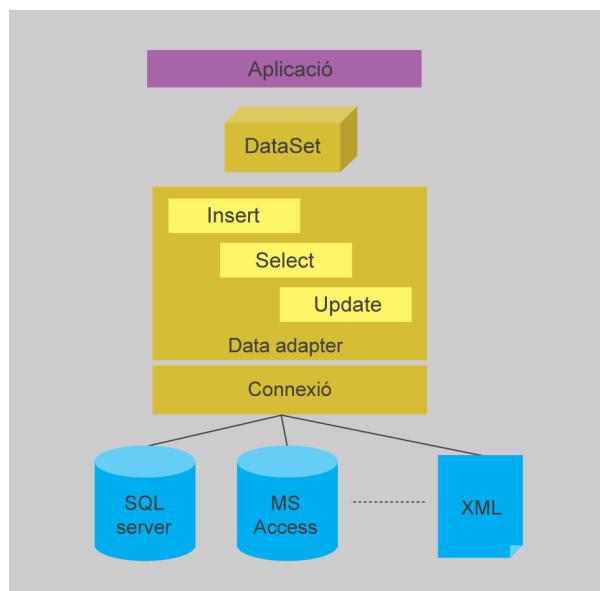
A continuació es farà un breu resum d'aquests objectes:

- **DataSet** és un objecte independent que emmagatzema temporalment les dades. Es pot considerar com un conjunt de registres que sempre està disconnectat i que no sap res sobre l'origen i la destinació de les dades que conté. Dins d'un objecte DataSet, de la mateixa manera que dins d'una base de dades, hi ha taules, columnes, relacions, restriccions, vistes, etc.
- **DataAdapter** és l'objecte que es connecta a la base de dades per omplir l'objecte DataSet. Si és necessari, es torna a connectar a la base de dades per actualitzar les dades de la base de dades a partir de les operacions fetes en les dades contingudes en l'objecte DataSet (operacions d'inserció, modificació o eliminació).

- **DataReader** proporciona una manera de llegir una seqüència de registres de dades només cap endavant des d'un origen de dades SQL Server.
- **Connection** permet connectar amb una base de dades i administrar les transaccions en una base de dades.
- **Command** permet emetre ordres SQL a una base de dades.

A la figura 1.31 es pot veure esquemàticament la relació entre aquests objectes.

FIGURA 1.31. Esquema d'ADO.NET



1.3.4 Els esdeveniments: concepte, associació d'accions, esdeveniments escoltadors.

Una aplicació no pot saber en quin moment l'usuari voldrà dur a terme una determinada funcionalitat. Aquesta interacció serà asíncrona durant l'execució del programa. Els *events* o esdeveniments proporcionen un mecanisme adequat per tractar aquest tipus de situacions que, normalment, són produïdes des de l'exterior de l'aplicació, per exemple quan l'usuari pressiona una tecla.

Quan arriba un esdeveniment, de vegades ens interessa tractar-lo (per exemple la pulsació d'un nombre en una aplicació que presenta la funcionalitat d'una calculadora) i altres vegades no volem tractar l'esdeveniment amb cap acció (per exemple quan l'usuari pressiona amb el ratolí sobre un text al qual no hem assignat cap informació complementària).

Per aclarir el seu significat, s'anomenaran alguns dels esdeveniment més comuns que es poden produir en l'execució d'una aplicació per mitjà de les seves interfícies gràfiques d'usuari:

- **MouseMove:** en moure el ratolí per sobre d'un control.

- **MouseDown:** en prémer qualsevol botó del ratolí
- **Change:** en canviar el contingut del control
- **Click:** en fer clic amb el botó esquerre del ratolí sobre el control
- **GetFocus:** aquest esdeveniment s'activa quan el control rep l'enfocament, és a dir, quan s'activa el control en temps d'execució per introduir-hi dades o fer alguna operació.
- **LostFocus:** és el contrari de l'esdeveniment anterior, s'activa quan el control perd l'enfocament, és a dir, es passa a un altre control per seguir introduint dades.

En la secció “Activitats” del web del mòdul podeu trobar un exercici resolt amb l’ús dels controls bàsics del quadre d’eines.

D'aquesta manera, si volem que un text es posi de color vermell en situar-nos a sobre, i de color gris en sortir-ne (amb el ratolí), hem de crear les instruccions dels esdeveniments *mouseEntered* i *mouseExited*; el primer esdeveniment s'encarregará de posar el text de color vermell i el segon de posar-lo de color gris.

1.3.5 Edició del codi generat per les eines de disseny

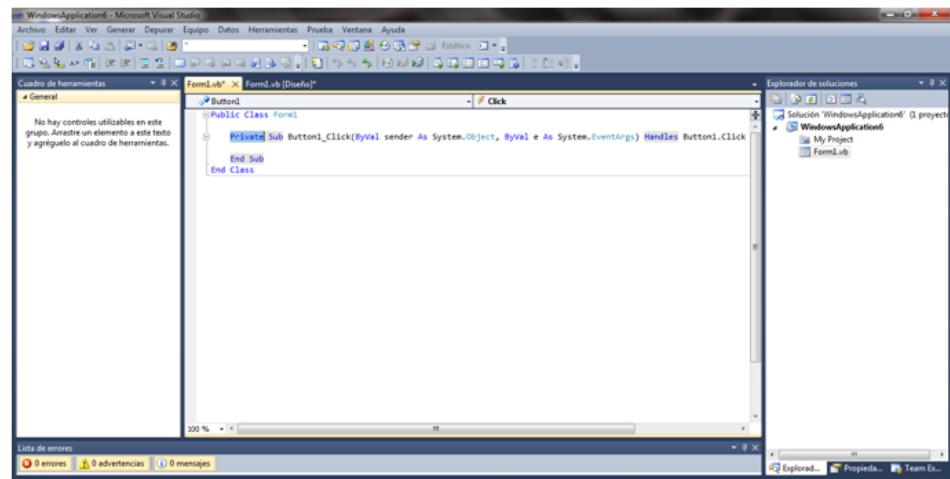
Els entorns integrats de desenvolupament i les altres eines de disseny i desenvolupament d'interfícies són aplicacions que permeten dissenyar interfícies gràfiques d'usuari sense necessitat de programar una sola línia de codi. Però en molts pocs casos un desenvolupador obtindrà un producte final sense la necessitat d'haver de modificar algun detall al qual no haurà arribat l'aplicació de desenvolupament.

Per aquesta raó caldrà tenir accés al codi generat per les eines de disseny, per poder-lo modificar i adequar a les nostres necessitats.

Hi ha aplicacions que integren un editor de codi, cosa que permet accedir-hi sense haver de sortir de l'entorn de desenvolupament, a més de poder veure els resultats de les modificacions fets de manera immediata.

Altres aplicacions generen el codi i els arxius necessaris per poder disposar de les interfícies generades, però no permeten aquesta edició en el seu entorn de desenvolupament, cosa que farà necessari l'ús d'altres aplicacions per poder-lo editar, aplicacions que podran ser especialitzades en aquestes tasques o podran ser molt senzilles, com qualsevol editor de text.

És necessari que el codi generat sigui un codi modular i senzill d'entendre. Si un programador ha de dedicar més temps a entendre i fer seu el codi, potser no li haurà estat de gaire ajuda l'eina de desenvolupament. Es pot veure un exemple d'edició de codi amb el Microsoft Visual Studio a la figura 1.32.

FIGURA 1.32. Eina de disseny

Pràcticament tots els IDE que hi ha avui dia permeten aquesta edició de codi. Alguns exemples d'aquestes eines poden ser **Eclipse**, **MonoDevelop**, **IDE NetBeans** o **Glade**. Algunes d'aquestes eines, fins i tot, generen codi automàticament a partir de les interfícies en desenvolupament en un entorn gràfic, cosa que facilita molt la feina dels programadors.

2. Generació d'interfícies a partir de documents XML

Per dissenyar i desenvolupar una interfície gràfica d'usuari es poden escollir entre moltes opcions d'entorns de desenvolupament i entre molts llenguatges de programació. Si s'escull un llenguatge de quarta generació i un entorn integrat de desenvolupament, es podrà disposar de moltes més eines i elements que facilitaran aquesta feina.

La primera decisió que cal establir pel que fa al disseny d'una aplicació que tindrà diverses interfícies d'usuari és l'entorn d'execució, si serà local o en un entorn client-servidor, si serà per a WinForms o per a un entorn web, si RichClient o ThinClient... Aquesta decisió influirà en el llenguatge de programació de l'aplicació i de les interfícies, per decidir, *a posteriori*, quin entorn de desenvolupament caldrà escollir.

2.1 Introducció a XML

Què és XML? Es veurà amb més detall als apartats següents, però bàsicament és un llenguatge amb una funció principal: descriure dades per mitjà d'etiquetes, sense preocupar-nos com es tractaran aquestes dades ni com es mostraran. Què té de positiu? Aquest llenguatge pot ser desenvolupat pels programadors o dissenyadors de programari des de molts altres llenguatges com Java, C#, .NET, etc.

Per què parlem d'XML en un tema que tracta de la generació d'interfícies gràfiques d'usuari? Un arxiu en XML no permet dissenyar ni configurar una interfície gràfica, només oferirà les dades necessàries per ser mostrades o manipulades. A partir d'altres llenguatges de programació que fan servir les característiques d'XML es poden desenvolupar interfícies de manera molt senzilla, per a aplicacions especialment pensades per a un entorn Web i acomplint els estàndards, criteris i recomanacions definides pel W3C.

W3C

W3C és el World Wide Web Consortium. Es tracta d'una comunitat internacional que desenvolupa estàndards, criteris i recomanacions referents a les tecnologies web que assegurin, a llarg termini, el creixement d'Internet.

2.1.1 Què és l'XML?

XML (eXtensible Markup Language) té com a funció principal la de descriure dades per mitjà d'etiquetes i no es preocupa de la manera de tractar-les o mostrar-les.

XML és un metallenguatge extensible d'etiquetes, és a dir, que a partir de la definició de dades per mitjà d'etiquetes i certes regles serveix com a base per

definir altres llenguatges de marques. Alguns exemples de llenguatges definits sobre XML són:

- XHTML
- RSS (Really Simple Syndication)
- SVG (Scalable Vector Graphics)
- MathML
- GraphML
- MusicXML

Les etiquetes són un conjunt de caràcters alfanumèrics que quedarán escrits entre els símbols > i <. Aquest conjunt de caràcters, el substantiu que representen, serà l'identificador de l'etiqueta. Per exemple, <nom> és una etiqueta amb l'identificador "nom".

Com més va més aplicacions són capaces de definir el seu propi llenguatge específic basat en l'ús d'etiquetes XML. Aquest fet fa que el llenguatge XML sigui com més va més utilitzat i més popular entre els desenvolupadors d'aplicacions i d'interfícies. Tot això és gràcies al seu enfocament genèric, extensible i molt senzill, basat en molts pocs elements bàsics i unes regles de format molt estrictes. A partir de fitxers XML, i algunes eines o llenguatges més, es podran desenvolupar tot tipus d'arxius i formats, com els representants a la figura 2.1.

FIGURA 2.1. Formats de l'XML



Per entendre millor el funcionament de l'XML es mostra a continuació un exemple de les seves característiques i del seu codi:

La X d'XML significa *eXtensible*, és a dir, ampliable. Ser ampliable vol dir que pot definir qualsevol conjunt d'etiquetes addicionals sense que bloquegi l'aplicació. Per exemple, a continuació tenim una part de codi implementat en XML. Per a una assignatura, n'indicarem el nom i les hores que durarà la impartició.

```

1  <assignatura>
2    <nom> Programació </nom>
3    <hores> 150 </hores>
4  </assignatura>
```

En aquest codi les etiquetes són: assignatura, nom i hores.

Ara es voldrà fer més extens aquest exemple amb més informacions, és a dir, ampliar-lo. A més de les dades descrites es vol afegir el nom del professor, però també afegir un nivell més a aquestes dades (un fill més), que són els noms dels alumnes. Per aconseguir això haurem de modificar l'arxiu XML perquè tingui aquest nou aspecte:

```

1 <assignatura>
2   <nom> Programació </nom>
3   <professor> Joan Castells </professor>
4   <alumnes>
5     <alumne> Joan Garcia </alumne>
6     <alumne> Carol Perez </alumne>
7     <alumne> Pep Perez </alumne>
8   </alumnes>
9   <hores> 150 </hores>
10 </assignatura>
```

L'aplicació que llegeixi aquest document XML podrà saber, a més, quins alumnes cursen l'assignatura.

Una arxiu XML és un arxiu que conté a la vegada les **dades** i la seva **estructura** en el **mateix document**. Aquesta és una característica molt important dels fitxers XML. El fet que siguin **autodescriptius** simplifica molt el treball amb aquest tipus de fitxers, ja que són molt senzills d'entendre i de manipular. Aquest fet és molt important tant per al desenvolupament de programes que els utilitzin, com per usar-los amb configuracions o enviament de dades entre diferents entorns o plataformes.

2.1.2 Àmbit d'aplicació

Els documents XML són fitxers de text normal que es poden obrir des de qualsevol editor de text, per molt senzill que sigui. No són com altres tipus d'arxius que necessiten un programari de propietat específic per interpretar-los, com passa amb la majoria dels arxius binaris. Això vol dir que es pot utilitzar un editor tan senzill com el Vi en Linux o el Bloc de notes de Windows per obrir i editar un fitxer XML.

Una de les conseqüències més importants d'aquest fet és la portabilitat que ofereix l'XML. El format d'arxius de tipus text i el fet de ser autodescriptius fan que es puguin manipular i entendre des de qualsevol plataforma.

Cal comentar també altres àmbits d'aplicació de l'XML:

- Quan es necessita intercanviar informació a través de diverses plataformes (compatibles o incompatibles) de maquinari o de programari, o de diverses aplicacions, XML pot ser l'elecció més encertada. Aquest és el cas d'aplicacions d'XML en l'àmbit de tecnologies de la comunicació com, per exemple, el popular WML (llenguatge de format sense fils) i WAP (protocol

d'aplicacions sense fils).

WML

WML és un llenguatge basat en XML que s'utilitza per donar format a aplicacions d'Internet per a dispositius de mà, com telèfons o PDA.

B2B: Business to Business

- En aplicacions de comerç electrònic entre empreses (B2B), en què les empreses necessiten intercanviar una gran quantitat d'informació financerà de manera independent de la plataforma, l'XML també serà una bona elecció. Diverses aplicacions utilitzen SOAP (Protocol senzill d'accés a objectes), un protocol molt popular basat en XML, per intercanviar aquest tipus d'informació a través d'Internet. Aquestes aplicacions basades en XML que s'utilitzen per compartir informació s'anomenen *serveis web*.

En el cas de desenvolupadors que fan servir llenguatges com HTML i CSS per dissenyar les interfícies i la presentació de les dades, fent servir XML per transferir-les es reduirà el risc d'incloure contingut redundant. Si hi ha canvis pel que fa a les interfícies o en la configuració de la presentació, això no afectarà les dades, que s'emmagatzemen per separat en un arxiu XML.

Com a exemple per a aquest darrer cas es pot plantejar un sistema de gestió de continguts. Aquest sistema pot generar documents per a usuaris finals en diversos formats, com poden ser HTML, PDF, etc. No obstant això, no té sentit emmagatzemar múltiples versions (una a cada format) d'un mateix document amb les mateixes dades. El contingut es duplicaria i ocuparia un valuós espai en el disc, la qual cosa ompliria el CMS d'informació redundant i en faria més lent l'ús. Si s'utilitza un motor basat en XML, el contingut es pot emmagatzemar una sola vegada per extreure'l i mostrar-lo posteriorment en el format demanat. Si XML separa el contingut de la presentació, què s'haurà d'utilitzar per presentar les meves dades? Hi ha altres llenguatges basats en els documents XML, que es treballaran en els apartats següents.

En resum, alguns dels àmbits d'aplicació de l'XML podrien ser:

- En aplicacions que manegen gran quantitat de dades i, alhora, necessiten ser flexibles i ampliables. Per exemple, llocs web, llistes d'ofertes de feina o aplicacions financeres.
- En aplicacions en què la presència de continguts redundants sigui un perill, com en sistemes d'administració de contingut, biblioteques de documents o sistemes de seguiment de llocs web.
- En aplicacions en què cal intercanviar gran quantitat de dades a través de diferents plataformes, com les aplicacions B2B, clients de correu electrònic compatibles amb servidors de notícies o dispositius mòbils.
- En aquelles situacions en què la informació ha d'estar disponible per a un gran nombre de clients. Per exemple, titulars de notícies, comunicats de premsa d'empreses, avisos i anuncis importants, marcadors, llistes de reproducció, calendaris d'esdeveniments o llistes de correu.

Actualment, XML s'utilitza habitualment per transferir dades entre diferents aplicacions de bases de dades. La majoria dels SGBD, incloent-hi Microsoft Access i Oracle, permeten exportar taules de bases de dades com a fitxers XML.

2.1.3 Estructura d'un document XML: etiquetes, atributs i valors

Un fitxer XML és un arxiu en format text, que conté etiquetes per identificar els elements i dades que componen el document. El primer que cal tenir en compte a l'hora de definir un fitxer XML és que la primera línia del fitxer ha de ser la següent:

```
1 <? Xml version = "1.0" encoding = "ISO-8859-1"?>
```

La resta del document XML s'escriurà amb etiquetes, i sempre cal obrir-les i tancar-les:

```
1 <etiq1> ..... </etiq1>
2 <etiq2> ..... </etiq2>
3 <etiq3> ..... <etiq4> ..... </etiq4>..... </etiq3>
```

El conjunt format per les etiquetes (obertura i finalització) i el contingut es coneix com a *element* o *node* (en el cas de fer una representació jeràrquica de les dades). Per exemple, el conjunt **<nom>Pere</nom>** és un element o node, amb l'etiqueta “nom” i el contingut “Pere”.

Es pot donar el cas que el contingut d'un element contingui altres elements en comptes de les dades en format de text, però no podrà contenir les dues coses: els altres elements i dades en format de text.

Els comentaris en XML s'escriuran d'aquesta manera:

```
1 <!———— Comentari —————>
```

Aquest tipus d'estructura de document amb les etiquetes i els atributs demana tenir molta cura de la manera d'emmagatzemar la informació, estructurant molt bé el document i ordenant adequadament les informacions.

Un document XML té una estructura imbricada de manera jeràrquica. És obligatori tancar totes les etiquetes, i si un element té vinculats altres elements (*fills*), és a dir, altres elements que en descendeixen, s'han de tancar les etiquetes dels fills abans de poder tancar l'etiqueta del pare.

Un document XML es troba ben format si disposa d'una estructura sintàcticament correcta.

Per aconseguir això caldrà que l'estructura jeràrquica s'acompleixi de manera estricta pel que fa a les etiquetes que es fan servir al document. És a dir, totes

les etiquetes obertes han d'haver estat tancades en l'ordre adequat. Els valors de les dades o continguts dels nodes es troben entre el text que indica l'obertura de l'etiqueta i el que n'indica el tancament.

Etiquetes

Totes les **etiquetes** han d'estar correctament tancades, és a dir, amb una etiqueta de tancament que es correspongui amb la d'obertura.

Les etiquetes han d'estar **sempre** tancades.

Un **exemple incorrecte** seria:

```

1 <menu>
2   <element posicio="15;3">
3     <nom>Menu
4     <element posicio="66;30">
5       <nom>Finestra

```

En canvi, vegeu-ne ara una **versió correcta**:

```

1 <menu>
2   <element posicio="15;3">
3     <nom>Menu</nom>
4   </element>
5   <element posicio="66;30">
6     <nom>Finestra</nom>
7   </element>
8 </menu>

```

Les **etiquetes buides** (és a dir, sense contingut) tenen una sintaxi especial:

```

1 <etiqueta/>

```

Atributs

Els elements poden tenir **atributs**, que són una manera d'inserir característiques o propietats als elements d'un document.

La manera de representar aquesta informació serà a partir d'afegir a l'etiqueta d'obertura del node els atributs (tant el nom com els valors) que volem.

```

1 <gos color= "blanc"> Pastor alemany </gos>

```

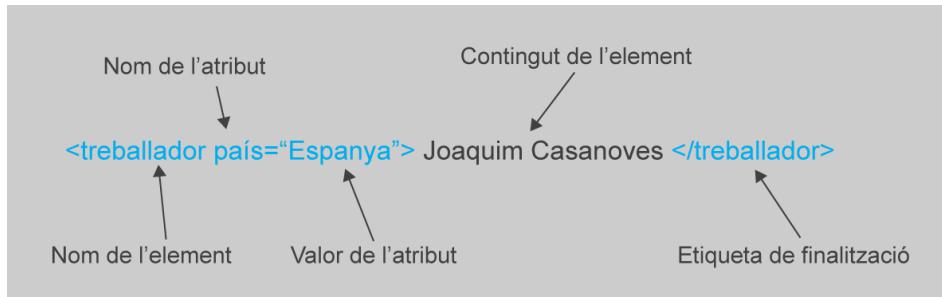
Els valors dels atributs dels elements sempre han d'estar marcats amb les **cometes dobles** ('') o **simples** (').

```

1 <a href = "http://www.ioc.com"> És correcte </a>
2 <a href = 'http://www.ioc.com'> És correcte </a>
3 <a href = http://www.ioc.com> No és correcte </a>

```

Com es pot veure a la figura 2.2, en el cas del treballador de nom “Joaquim Casanoves” s’ha afegit l’atribut *país* per indicar la seva nacionalitat, i s’indica a més el valor d’aquest atribut.

FIGURA 2.2. Estructura d'un XML

El nom de l'etiqueta d'un element, atribut, entitat... comença per una lletra, i continua amb lletres, dígits, guionets, ratlles, punt o dos punts.

El text *XML* (o *xml* o *xMl*, etc.) no es pot utilitzar com a caràcters inicials d'un nom d'element, atribut, entitat, etc.

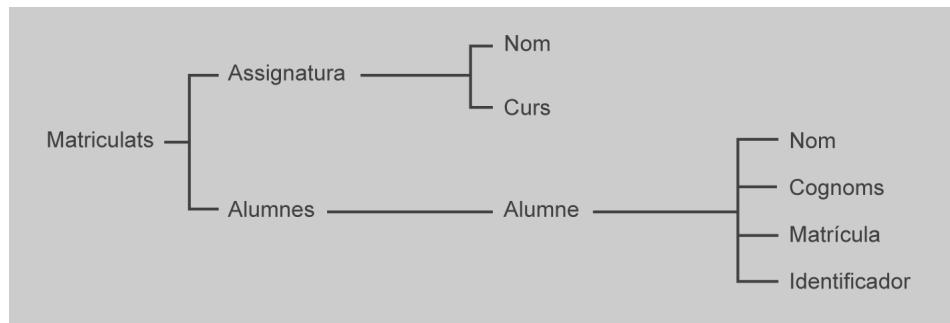
XML és ***case-sensitive*** (sensible a la caixa, l'ús de majúscula o minúscula), és a dir, no és el mateix <**autor**>; que <**Autor**>;.

Un exemple d'*XML* complet i ben format seria:

```

1  <?xml version = "1.0" encoding = "ISO-8859-1"?>
2  <Matriculats>
3      <Assignatura>
4          <Nom> Generacio interficies </Nom>
5          <Curs> 2 </Curs>
6      </Assignatura>
7      <Alumnes>
8          <Alumne>
9              <Nom> Joan </Nom>
10             <Cognoms> Garcia Castellejos</Cognoms>
11             <Matricula> 060000 </Matricula>
12             <Identificador Tipus="DNI"> 4060000 </Identificador>
13         </Alumne>
14         <Alumne>
15             <Nom> Albert </Nom>
16             <Cognoms> Lucas Martinez </Cognoms>
17             <Matricula> 050000 </Matricula>
18             <Identificador Tipus="NIF"> 3060000H </Identificador>
19         </Alumne>
20     </Alumnes>
21 </Matriculats>
```

D'aquesta manera l'estructura jeràrquica de l'*XML* serà la que es mostra a la figura 2.3, amb el node de *Matriculats* al primer nivell, els alumnes i les assignatures al segon nivell, el nom i curs de les assignatures al tercer nivell i les dades de l'alumne com a tercer nivell dels alumnes. El nivell quart, només assolit per la branca de l'*Alumne*, ens indica les dades d'aquest (nom, cognoms, matrícula i identificador).

FIGURA 2.3. Jerarquia de l'XML

Consultes XPath

XPath o XML Path Language és un llenguatge de consultes per accedir a un document XML o per referir-se a parts d'un document XML. Amb les **consultes XPath** podrem accedir, per exemple, a la informació d'un node.

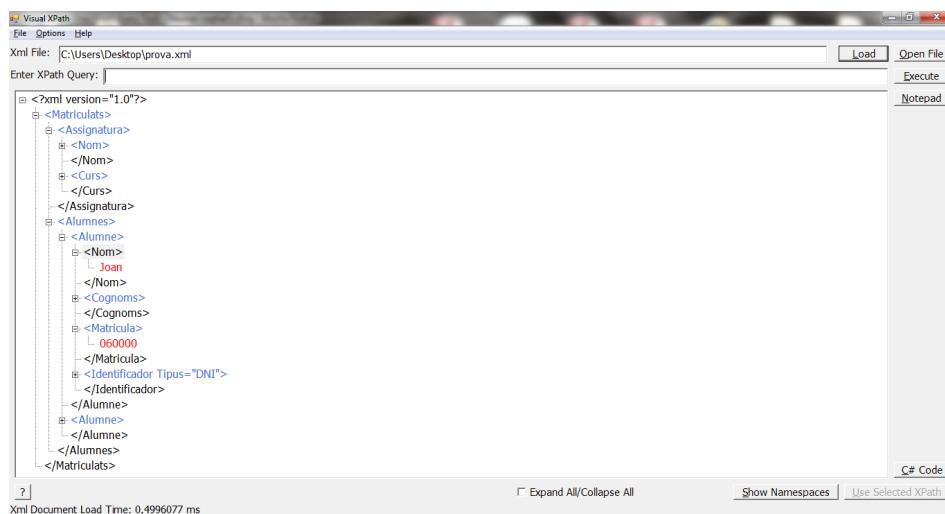
XSLT

Estàndard de l'organització W3C que presenta una manera de transformar documents XML en altres formats.

XPath, definit pel W3C, es pot utilitzar sota diversos entorns, per exemple amb fitxers XSLT, a partir del llenguatge de programació PHP, a partir de Java, etc.

Per veure un exemple, es farà servir l'eina Visual XPath. Amb aquesta eina es pot veure un exemple senzill de com s'executa una consulta i quins serien els resultats.

Si s'agafa l'exemple en XML anterior i s'obre fent servir el programa Visual XPath, s'obté el resultat representat a la figura 2.4.

FIGURA 2.4. XPath

Per exemple, si executem la consulta: **Matriculats / Alumnes / Alumne** en l'XML anterior, el resultat serà un XML que conté tots els nodes d'alumne.

```

1  <?xml version = "1.0" encoding = "ISO-8859-1"?>
2  <Matriculats>
3      <Alumnes>
4          <Alumne>
5              <Nom> Joan </Nom>
6              <Cognoms> Garcia Castellejos</Cognoms>
7              <Matricula> 060000 </Matricula>
8              <Identificador Tipus="DNI"> 4060000 </Identificador>
  
```

```

9      </Alumne>
10     <Alumne>
11       <Nom> Albert </Nom>
12       <Cognoms> Lucas Martinez </Cognoms>
13       <Matricula> 050000 </Matricula>
14       <Identificador Tipus="NIF"> 3060000H </Identificador>
15     </Alumne>
16   </Alumnes>
17 </Matriculats>

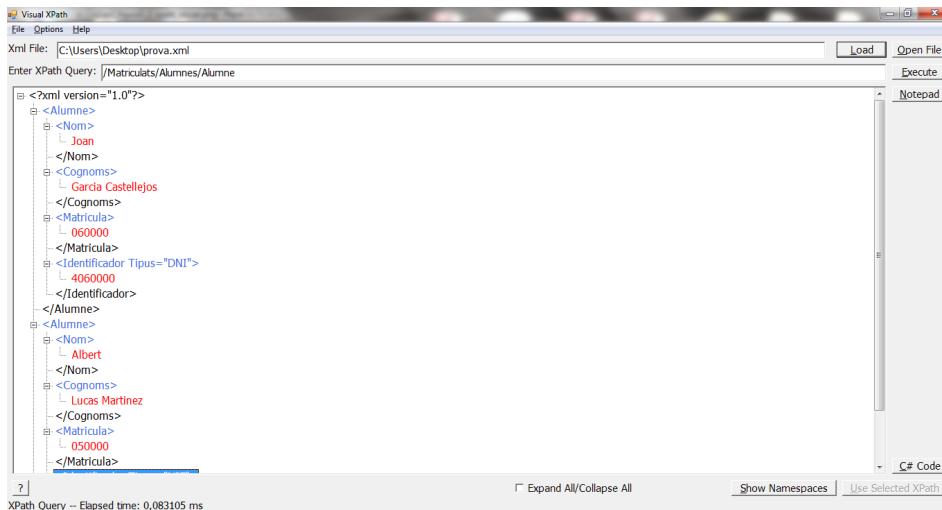
```

És a dir, la solució seria Joan, Garcia Castellejos, 060000, 4060000, Albert, Lucas Martinez, 050000, 306000H.

En el cas de Visual XPath, executar una consulta implica escriure-la en el seu espai corresponent i fer servir el botó “Execute”.

Executant la mateixa consulta amb XPath, s'obté el resultat mostrat a la figura 2.5.

FIGURA 2.5. XPath



Es pot observar com la consulta XPath indica els nodes que cal anar recorrent per arribar al node final. Aquestes consultes tenen en compte la diferència entre majúscules i minúscules, és a dir, no és el mateix **matriculats / Alumnes** que **Matriculats / Alumnes**; la primera no tornaria cap resultat, mentre que la segona sí que tornaria resultats.

2.1.4 Edició d'un document XML

Un document XML és un document de text ASCII, com molts altres tipus de documents que es desenvolupen a partir de diferents llenguatges de programació (per exemple, l'HTML). Aquest llenguatge és interpretat per altres llenguatges o programes específics, però per editar-lo no cal cap eina especial.

Una aplicació tan senzilla com un Edit en MSDOS o un Bloc de notes en un entorn Windows permeten la visualització i edició del document XML.

HTML

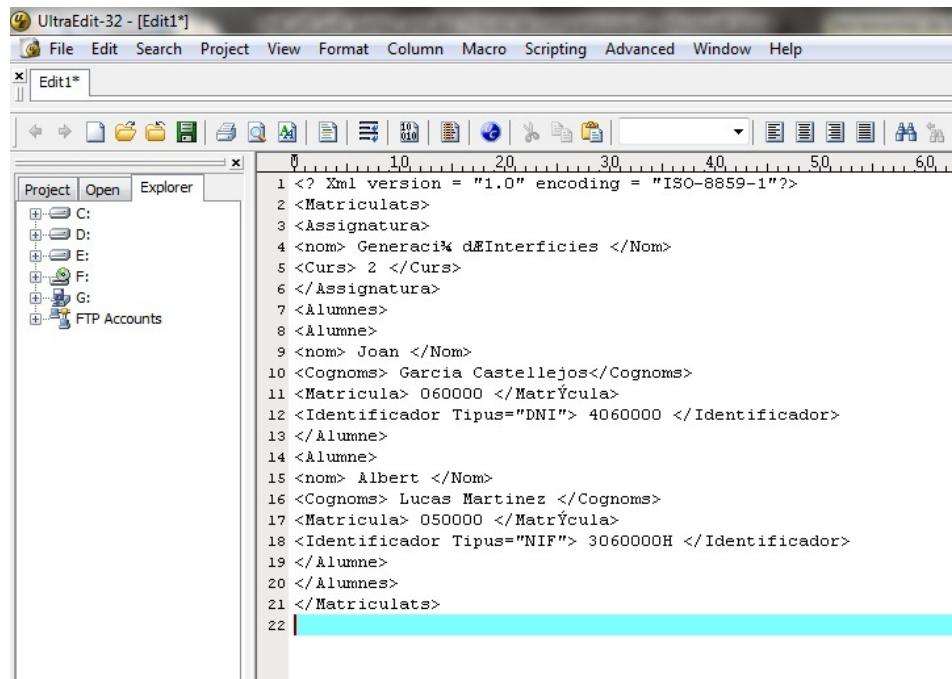
HTML són les sigles d'Hyper Text Markup Language, és a dir, 'llenguatge de marques d'hipertext').

Ultraedit

L'Ultraedit és una eina de propietat d'edició de codi no integrada amb cap entorn de desenvolupament però que permet el reconeixement de molts llenguatges de programació, com l'XML. Per a Windows i per a Linux.

A la figura 2.6 és pot veure com podem editar el codi de l'exemple tractat anteriorment amb una eina d'edició com l'Ultraedit.

FIGURA 2.6. Ultraedit



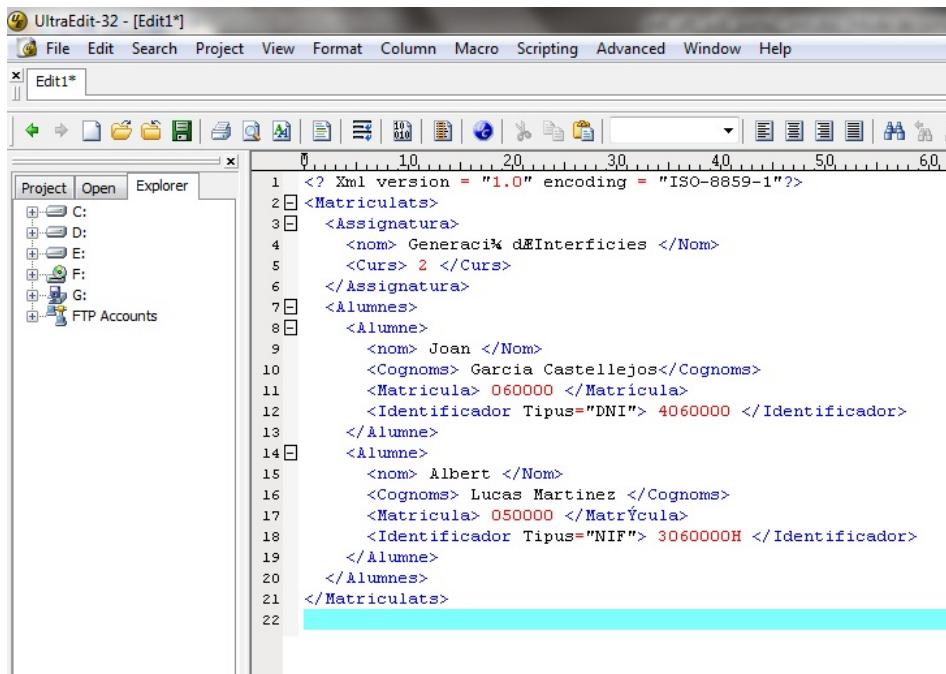
Altres eines permeten el reconeixement de les etiquetes que es fan servir, encara que sigui el desenvolupador el que en decideixi el nom, cosa que permet un desenvolupament més senzill i agradable. Alguns entorns integrats de desenvolupament permetran aquest reconeixement a més de permetre indicar si la sintaxi és la correcta o mostrar com queda el resultat del codi una vegada interpretat o executat, en funció del llenguatge de programació utilitzat.

Però, d'altres vegades, interessa fer servir un editor de codi genèric que ofereix algunes funcionalitats més que les que oferirà un bloc de notes, sense arribar a necessitar un IDE o, senzillament, perquè no existeix per a un llenguatge de programació determinat. Aquestes funcionalitats poden ser el reconeixement de llenguatges de programació o la possibilitat d'escriure en format de columnes.

Una altra característica que permet un tractament millor del codi, tant pel que fa l'edició com a la comprensió, és el fet de sagnar cap a la dreta aquelles etiquetes que formen part d'un mateix nivell jeràrquic. El fet de poder-les agrupar ajudarà molt a la comprensió del codi.

A la figura 2.7 es pot veure l'edició de l'exemple amb l'editor UltraEdit, amb el reconeixement de les etiquetes i la sagnia del codi.

Hi ha moltes altres eines específiques per l'edició de XML. Algunes d'elles són:

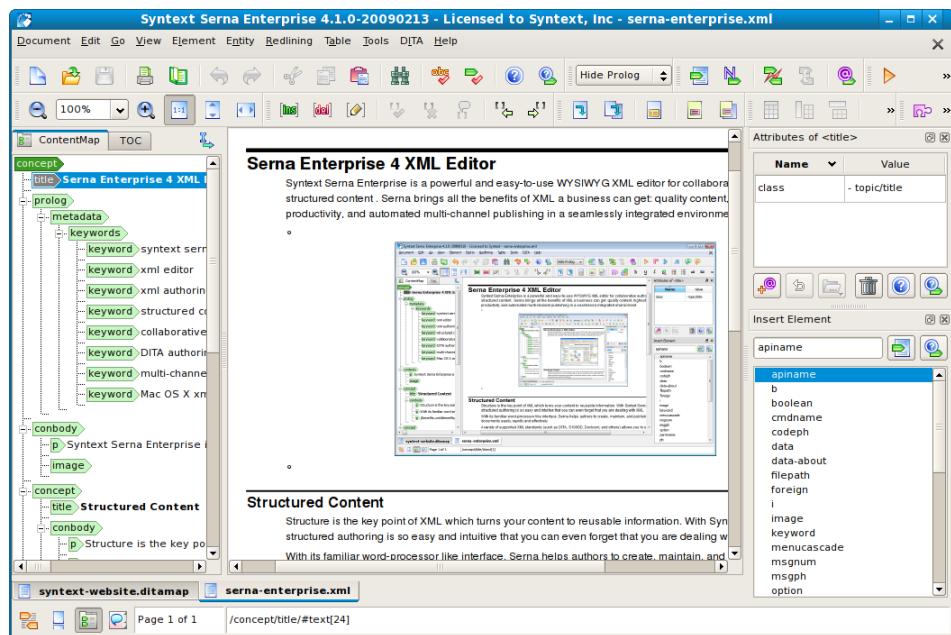
FIGURA 2.7. Ultraedit

- **Syntext Serna.** Editor per a Windows, Linux i Mac, eina amb una versió lliure que permet l'edició de text, l'edició gràfica i l'edició de tipus WYSIWYG.
- **XML Studio.** Editor per a Windows, eina de propietat que permet l'edició de text, l'edició gràfica i l'edició de tipus WYSIWYG.
- **Oxygen XML Editor.** Editor per a Windows, MAC i Linux, eina de propietat que permet l'edició de text, l'edició gràfica i l'edició de tipus WYSIWYG.
- **XML Notepad 2007.** Editor per a Windows, amb llicència pública de Microsoft, que permet l'edició de text, l'edició gràfica i l'edició de tipus WYSIWYG.

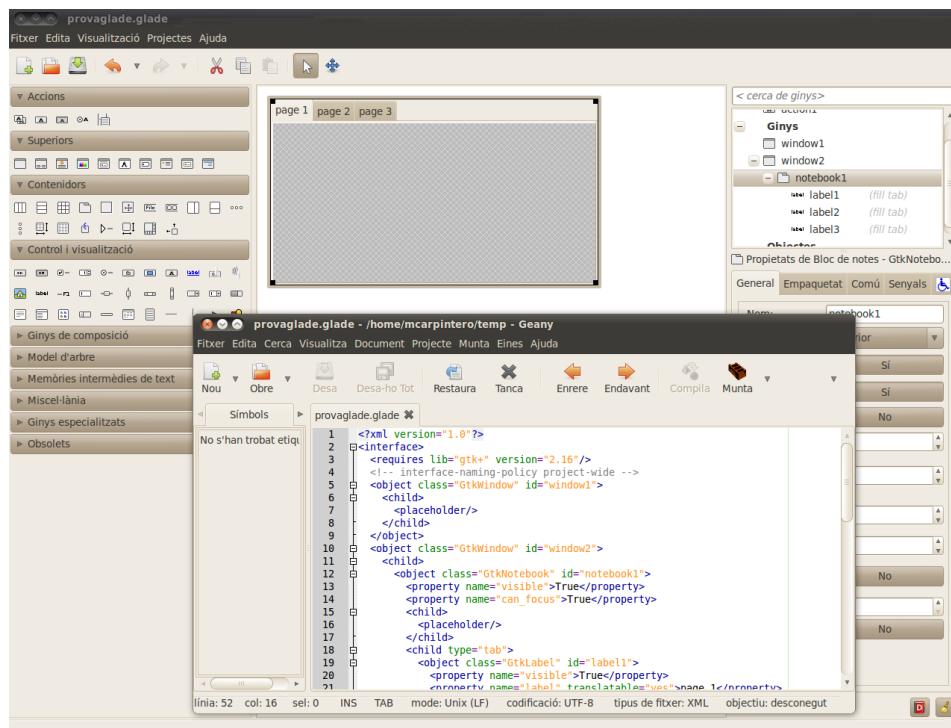
A la figura 2.8 es pot veure un exemple de l'eina Syntext Serna.

Una altra alternativa per editar documents XML és fer-ho a partir de codi generat per eines específiques, com, per exemple MonoDevelop o Glade.

Glade és una eina lliure que ajuda a la creació d'interfícies gràfiques d'usuari. Una vegada dissenyada la interfície, es pot generar el codi corresponent en XML, i aquest es podrà editar per aconseguir modificacions.

FIGURA 2.8. Syntext Serna

A la figura 2.9 se'n pot veure un exemple.

FIGURA 2.9. Glade

2.2 Llenguatges de descripció d'interfícies basats en XML

L'XML és un llenguatge que, per si mateix, no oferirà el disseny d'una interfície gràfica d'usuari. Necessitarà altres llenguatges que aprofitin les seves característiques per oferir el desenvolupament d'una GUI.

És a dir, hi ha llenguatges basats en l'estàndard XML que permeten descriure com ha de ser tractada la informació que conté un document XML per presentar-la en un mitjà com pot ser una pàgina web (HTML) o qualsevol altre document estructurat.

En aquest apartat és durà a terme una descripció dels llenguatges de programació basats en XML:

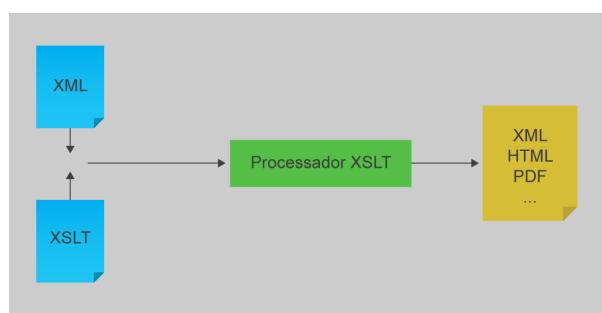
- XSLT (eXtensible Style Language Transformation)
- XUL (eXtensible User interface Language)
- XIML (eXtensible Interface Markup Language)

2.2.1 XSLT (eXtensible Style Language Transformation)

XSLT (o XSL Transformations) és un estàndard de l'organització W3C que presenta una manera de transformar documents XML en altres tipus de documents, fins i tot en formats que no són XML.

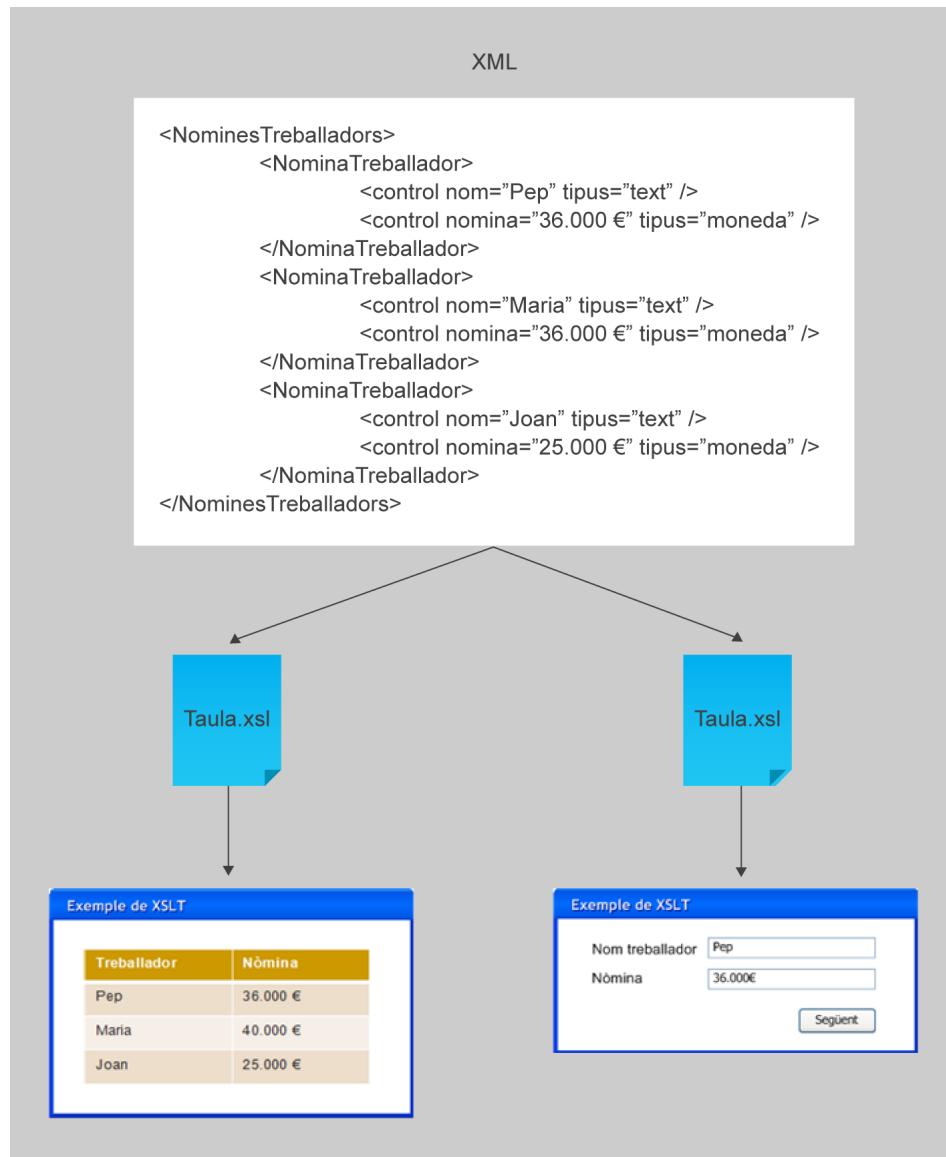
A la figura 2.10 es pot veure la manera de funcionar de l'estàndard XSLT. Els fulls d'estil XSLT fan la transformació del document utilitzant una o diverses regles de plantilla unides al document font que cal transformar.

FIGURA 2.10. Processador XSLT



XSLT està obtenint, actualment, molt reconeixement en l'edició web, i genera pàgines HTML o XHTML. La unió d'XML i XSLT ajudarà a l'augment de la productivitat pel fet de permetre la separació de contingut i presentació gràfica.

D'aquesta manera es pot observar a la figura 2.11 que, a partir d'un mateix XML, aplicant diferents fulls XSLT, es pot representar la informació amb diferents formats. A la figura 2.11 s'observen dues interfícies diferents, amb les dades de diversos treballadors, que tindran el mateix origen, un únic document XML.

FIGURA 2.11. Diferents formats

Per entendre millor el funcionament de l'XSLT es planteja l'exemple següent:

Disposem d'un XML que tracta d'una agrupació de persones:

```

1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <alumnes>
3      <alumne>
4          <nom>Esther Garcia</nom>
5          <dni>38293450S</dni>
6          <notaMitjana>7,5</notaMitjana>
7      </alumne>
8      <alumne>
9          <nom>Carles Aries</nom>
10         <dni>34839593G</dni>
11         <notaMitjana>8,5</notaMitjana>
12     </alumne>
13     <alumne>
14         <nom>Marc Fernandez</nom>
15         <dni>30492839P</dni>
16         <notaMitjana>5,2</notaMitjana>
17     </alumne>
18 </alumnes>

```

Aquest document XML conté les dades (nom, DNI i nota mitjana) de tres alumnes, seguint una estructura jeràrquica amb *Alumnes/Alumne/Nom-DNI-notaMitjana*.

El document XSL de transformació que s'utilitzarà és:

```

1  <?xml version="1.0" encoding="iso-8859-1"?>
2      <xsl:stylesheet version="1.0"
3          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4          <xsl:template match="/">
5              <html>
6                  <body>
7                      <h2>Notes dels alumnes</h2>
8                      <table border="1">
9                          <tr bgcolor="#9acd32">
10                             <th>Nom</th>
11                             <th>Nota Mitja</th>
12                         </tr>
13                         <xsl:for-each select="alumnes/alumne">
14                             <tr>
15                                 <td>
16                                     <xsl:value-of select="nom"/>
17                                 </td>
18                                 <td>
19                                     <xsl:value-of select="notaMitjana"/>
20                                 </td>
21                             </tr>
22                         </xsl:for-each>
23                     </table>
24                 </body>
25             </html>
26         </xsl:template>
27     </xsl:stylesheet>
```

Un document XML es compon d'informació estructurada en nodes en forma d'arbre. Si es té en compte que un XSLT ha de recórrer un document XML, entendrem que la sintaxi bàsica per al desenvolupament d'XSLT són recorreguts, bucles i condicions que naveguen per l'estructura de l'XML. En aquest exemple, es farà un recorregut pel document XML recollint les dades dels alumnes referent als noms i a la nota mitjana, deixant de banda la informació referent al DNI.

La capçalera per a tots els documents serà:

```

1 <xsl:stylesheet version="1.0"
2     xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

En què s'indica un full d'estil XSL mitjançant l'etiqueta *stylesheet* per a tots els documents que fem. Indiquem la versió 1 del full d'estil amb l'atribut *version*. En el cas de complir les recomanacions del W3C s'especificarà amb **xmlns=http://www.w3.org/1999/xhtml** i finalment el seu *namespace* (criteri i restriccions de les recomanacions sobre l'estructura de tipus d'element i noms de l'atribut), amb *xmlns* i *xmlns:xsl*.

En l'exemple s'està transformant el document XML en un HTML compost per una taula amb dues columnes: el nom de l'alumne i la seva nota mitjana.

El resultat de la transformació serà un document HTML com el següent:

```

1 <html>
2     <body>
```

```

3      <h2>Notes dels alumnes</h2>
4      <table border="1">
5          <tr bgcolor="#9acd32">
6              <th>Nom</th>
7              <th>Nota Mitja</th>
8          </tr>
9          <tr>
10         <td>Esther Garcia</td>
11         <td>7,5</td>
12     </tr>
13     <tr>
14         <td>Carles Aries</td>
15         <td>8,5</td>
16     </tr>
17     <tr>
18         <td>Marc Fernandez</td>
19         <td>5,2</td>
20     </tr>
21     </table>
22   </body>
23 </html>

```

A la figura 2.12 es pot veure quin és el resultat d'aquest document HTML si l'obrim amb un navegador. Obtindrem una taula amb 4 files i dues columnes; la primera de les files conté la capçalera i la resta de les files el nom dels alumnes i la seva nota mitjana.

FIGURA 2.12. Exemple d'interfície

Notes dels alumnes

Nom	Nota Mitja
Esther Garcia	7,5
Carles Aries	8,5
Marc Fernandez	5,2

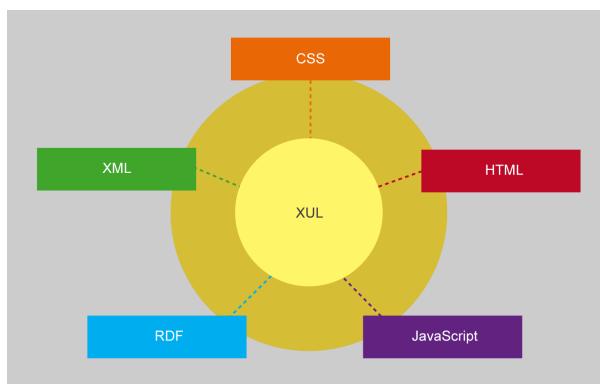
2.2.2 XUL (eXtensible User interface Language)

XUL és l'acrònim d'*eXtensible User interface Language*; traduït de l'anglès significa ‘llenguatge d'interfície d'usuari extensible’. Aquest llenguatge ha estat desenvolupat per crear interfícies d'usuari a partir de documents XML per als entorns Netscape i Mozilla.

XUL fa servir els elements d'un llenguatge de marques per crear interfícies gràfiques d'usuari, com ho fa un altre llenguatge com l'HTML. Es podrà definir l'aparença d'aquesta interfície amb fulls d'estil CSS i usar JavaScript per manipular-ne el comportament. A més, XUL té un conjunt extens de components gràfics usats per crear menús, barres d'eines, caixes de text, entre molts altres components. Aquest darrer punt el diferencia del llenguatge HTML.

XUL, a més de ser un llenguatge amb el qual és possible crear interfícies de manera fàcil, senzilla i ràpida, està disponible per a totes les versions de Windows, Mac, Linux i UNIX. Cal dir que, actualment, encara no és compatible al 100% amb el programari de Microsoft Internet Explorer, cosa que en dificulta l'extensió. A la figura 2.13 es fa una mostra dels diferents elements amb els quals pot interactuar un document desenvolupat en XUL, com són l'XML, el CSS, el JavaScript, l'HTML i l'RDF.

FIGURA 2.13. Elements amb els quals XUL pot interactuar



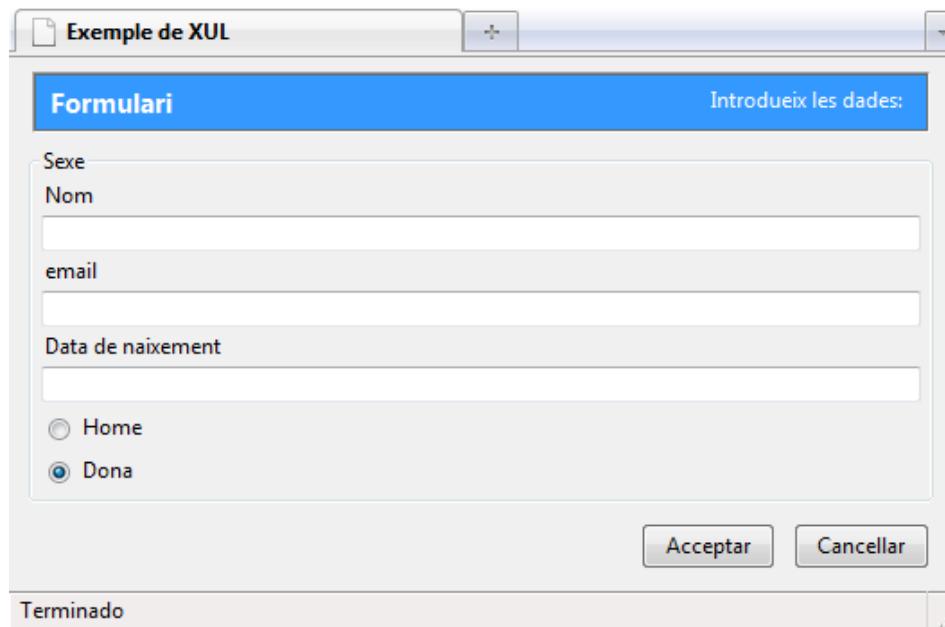
A continuació es desenvolupa un exemple en XUL que implementarà una interfície molt senzilla. Es tracta d'un petit formulari preparat per mostrar les dades contingudes en un document XML.

Per fer això, cal crear un arxiu anomenat *exempleXUL.xul* (amb un editor de text com podria ser el Notepad, l'Ultraedit...) al qual s'afegeix el codi següent:

```

1  <?xml version="1.0"?>
2  <?xml-stylesheet href="chrome://global/skin/global.css" type="text/css"?>
3  <dialog id=" IdentificadorDelFormulari" title=" Exemple de dialeg"
4      xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
5      buttons="accept,cancel"
6      buttonlabelcancel="Cancelar"
7      buttonlabelaccept="Aceptar"
8      ondialogaccept="return doOK();"
9      ondialogcancel="return doCancel();">
10
11      <dialogheader title="Options" description="My preferences"/>
12      <label value="Nom"/>
13      <textbox />
14      <label value="email"/>
15      <textbox />
16      <label value="Data de naixement"/>
17      <textbox />
18      <groupbox>
19          <caption label="Sexe"/>
20          <radiogroup>
21              <radio label="Home"/>
22              <radio label="Dona" selected="true"/>
23          </radiogroup>
24      </groupbox>
25  </dialog>
```

Posteriorment s'obrirà el navegador Mozilla i arrossegarem el fitxer *exemple-XUL.xul* cap al navegador, i veurem el formulari que es mostra a la figura 2.14.

FIGURA 2.14. Interfície gràfica

Arribats a aquest punt, anem a analitzar el codi. Qualsevol arxiu XML, independentment de la seva funció, ha de tenir algunes línies al principi que indiquen la versió d'XML, i un full d'estil si és oportú. En el nostre exemple tenim les dues línies següents:

```

1  <?xml version="1.0"?>
2  <?xmlstylesheet href="chrome://global/skin/global.css" type="text/css"?>
```

Com es pot observar, tenim una línia de versió que indica que la versió XML és la 1.0. La segona línia indica que el nostre full d'estil està en la ruta “chrome”. La ruta “chrome” conté algunes utilitats internes de Mozilla que gestionen habitualment les interfícies d'usuari de Mozilla. Les línies següents contenen les nostres primeres etiquetes:

```

1  <dialog id="IdentificadorDelFormulari" title="Exemple de diàleg"
2    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
3  </dialog>
```

Cada pàgina XUL que es crea necessita una etiqueta de tipus **<window>** o **<dialog>**, entre altres, que puguin ser usades per contenir els components que formen la interfície que volem desenvolupar. Dins d'aquesta etiqueta s'han fet servir 3 atributs.

- El primer *id* (identificador) és una referència única que apunta a aquesta etiqueta en XML. L'atribut *id* és essencial per ser capaços de comunicarnos amb etiquetes i actualitzar amb canvis i informació.
- La segona etiqueta, *title* (títol), conté una sèrie de caràcters llegibles per humans que es mostra a la barra de títol quan llancem l'arxiu XUL a la finestra oportuna.
- L'últim atribut és la part *xmlns*. Aquest atribut especifica el *namespace*

(espai de noms) de l'etiqueta **<window>**, **<dialog>**... en les quals es basaran totes les etiquetes que contingui.

Namespace

Un *namespace* és com un grup especial que podem especificar per determinar d'on ve una etiqueta. Això ajuda en situacions en què es tingui una etiqueta **<window>** d'un altre llenguatge XML i una etiqueta **<window>** del llenguatge XUL: el *namespace* les diferencia.

Ara es podrà omplir amb dades el formulari desenvolupat abans. En el nostre exemple hem creat un petit formulari: Creem un **<label>** (etiqueta) amb el valor “Nom”, en què l'usuari podrà escriure'l dins del camp **<textbox>** (quadre de text)

```
1 <label value="Nom"/>
2 <textbox />
```

De la mateixa manera l'usuari podrà indicar l'adreça electrònica i la data de naixement.

D'altra banda es crea un **<groupbox>** (agrupació de caselles) per indicar el sexe.

```
1 <groupbox>
2   <caption label="Sexe"/>
3   <radiogroup>
4     <radio label="Home"/>
5     <radio label="Dona" selected="true"/>
6   </radiogroup>
7 </groupbox>
```

2.2.3 Els esdeveniments

Moltes vegades, és necessari associar esdeveniments a elements del formulari que s'està desenvolupant. En aquest exemple es farà servir JavaScript per afegir funcionalitat als components. Això es fa d'una manera molt similar a la de l'HTML. En l'HTML, un conductor d'esdeveniments s'associa amb un element i es fa alguna acció quan s'activa el conductor. La majoria dels conductors d'HTML també es troben en XUL, i n'hi ha alguns més que només es troben en XUL.

En la secció Activitats del web del mòdul podreu trobar més exemples de funcionament de XUL i també altres dades d'interès.

Es pot afegir codi dins de l'arxiu de XUL, però una manera més eficient és fer-ho en un arxiu separat. Això permet que la càrrega de la interfície es faci més ràpidament.

Un exemple senzill del JavaScript associat al nostre exemple seria:

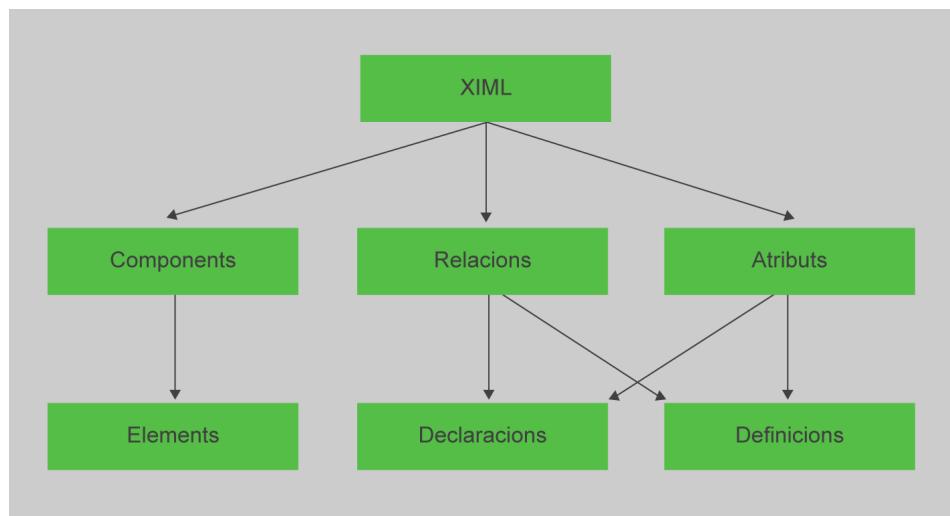
```
1 <script language= "JavaScript" >
2   function doOK(){ alert("Desa el dialeg"); }
3   function doCancel(){ alert("Tanca el dialeg"); }
4 </script>
```

2.2.4 XIML (eXtensible Interface Markup Language)

Un altre exemple de llenguatge per a la creació d'interfícies a partir de documents XML és l'XIML. És un acrònim d'*eXtensible Interface Markup Language* ('llenguatge de marques per a la creació d'interfícies extensibles'). Aquest llenguatge és independent de la plataforma en la qual es farà servir, i s'hi fa un èmfasi especial a desacoblar la representació gràfica de les dades de la interacció amb l'usuari.

A la figura 2.15 es pot veure quina serà l'estructura del llenguatge XIML.

FIGURA 2.15. Estructura del llenguatge XIML



Aquesta estructura està composta per components, relacions i atributs. Els components agruparan els elements de les interfícies. Les relacions definiran les interaccions entre diferents elements i els atributs disposaran de valors que definiran els elements. Cal remarcar la importància de l'estructura jerarquitzada de l'XIML.

Components de l'XIML

El llenguatge XIML està compost per un conjunt d'elements d'interfície d'usuari organitzats que es troben classificats en un o diversos components d'interfícies principals. El llenguatge XIML no limita el nombre i el tipus de components que podran ser definits en un document.

Es defineixen cinc tipus de components d'interfície en la primera versió del llenguatge XIML: presentació, domini, tasca, diàleg i usuari.

- **Presentació.** El component d'interfície *presentació* serveix per definir una jerarquia d'elements. Aquests seran els components que es mostraran a l'usuari. En aquest component no s'especificarà l'aspecte i les propietats que hauran de tenir aquests elements.
- **Domini.** El component *domini* representa un conjunt d'instàncies i classes.

Aquest conjunt estarà organitzat en una jerarquia. Aquesta jerarquia és conceptualment com una ontologia però molt més senzilla: hi haurà una definició dels objectes a partir d'atributs i els seus valors. Els objectes, que podran ser simples o complexos, només es consideraran objectes si l'usuari els ha visualitzat o modificat.

- **Tasca.** Aquest component representa els processos de negoci en els quals intervé un usuari juntament amb les accions que podrà desenvolupar aquest usuari en la seva interacció. Un component *tasca* defineix una descomposició jeràrquica de tasques i subtasques que defineixen el flux de treball d'aquestes tasques i les dades que utilitzen.
- **Diàleg.** Aquest component defineix una col·lecció estructurada d'elements que determinen les accions d'interacció que pot fer l'usuari. Aquest component representarà la implementació de les tasques i les seves accions, i també la navegació de l'usuari per l'aplicació.
- **Usuari.** El component *usuari* representa la jerarquia d'usuaris que haurà de tenir en compte l'XIML. Els elements d'aquest component seran nodes que podran identificar un únic usuari que navegarà per les interfícies de l'aplicació, o diversos usuaris (un grup d'usuaris relacionat, amb característiques comunes).

El llenguatge XIML ofereix la possibilitat d'afegir nous elements i components a la seva estructura. En estar compost per una estructura jeràrquica, està preparat per suportar afegir aquests nous elements.

Relacions

Les relacions en el llenguatge de programació XIML són una definició que vincularan dos o més elements XIML que es trobin dintre un mateix component o distribuïts entre diferents components.

Els elements i els components són molt importants en XIML, ja que ofereixen una informació o coneixement bàsic referent a les interfícies d'usuari. Però les relacions entre els elements també són, en si mateixes, una informació o coneixement també igual de bàsic que el que ofereixen els elements.

Els tipus de relacions que conté XIML són:

- de tipus **definició**, és a dir, relacions que especifiquen la forma canònica de la relació.
- de tipus **declaració**, és a dir, relacions que especifiquen la instància actual de la relació.

Atributs

Dins el llenguatge de programació XIML, els atributs són uns valors que es podran assignar a alguns elements. El significat dels atributs vinculats als elements

n'indicaran les propietats o característiques.

En XIML els valors que es poden assignar als atributs podran ser de dos tipus:

- **Valors de tipus bàsics** de dades (enters, caràcters, cadenes de caràcters...)
- **Instàncies a altres elements** existents en el mateix component o en altres components.

Altres característiques d'XIML

A continuació es presenten altres característiques importants del llenguatge de programació XIML per tenir en compte:

- El llenguatge XIML permet mostrar un mateix element o un component de moltes formes diferents, de tal manera que l'usuari podrà escollir les propietats i l'aspecte més convenient.
- Una altra característica interessant és la possibilitat de modificar l'aspecte de les interfícies d'usuari de manera dinàmica, i llavors dóna la possibilitat d'adaptar-se als canvis que hi pugui haver en els dispositius de visualització. Aquesta és una característica que també es pot trobar en els XForms; això sí, limitats a tres estats. Amb el llenguatge XIML no hi ha límit d'estats per definir. Un exemple per a aquesta característica és la possibilitat d'adaptar les interfícies a un canvi de dimensions de visualització.
- El llenguatge XIML permet la creació de presentacions, una o diverses. Cada presentació és pot vincular a un tipus de dispositiu amb característiques diferents (PC de sobretaula, portàtils de poques polzades, dispositius mòbils, telèfons...). Això s'aconsegueix gràcies a la possibilitat de desvincular completament la definició de les interfícies de la visualització. Cada presentació és farà servir (s'activarà) en el moment que el sistema detecti el dispositiu en el qual es faci servir l'aplicació.
- Una altra característica és la possibilitat de fer servir les múltiples visualitzacions que ofereixen les empreses externes, i que podran fer servir els usuaris de les interfícies sense cap necessitat d'instal·lacions ni descàrregues.

2.2.5 Altres llenguatges

Hi ha altres llenguatges no tan habituals, però també basats en l'estàndard XML, que permeten descriure com ha de ser tractada la informació que conté un document XML.

En aquest apartat es descriuràn breument altres llenguatges de programació basats en XML:

- XAML (Extensible Application Markup Language)
- XForms (neXt generation web Forms)
- AUIML (Abstract User Interface Markup Language)
- UIML (User Interface Markup Language)

XAML (Extensible Application Markup Language)

XAML (Extensible Application Markup Language) és un llenguatge de programació el nom del qual, traduït, vol dir ‘llenguatge de marques ampliable per a aplicacions’. De fet, la *A* inicialment significava *Avalon*. És a dir, Extensible Avalon Markup Language. Avalon és la manera antiga de denominar WPF (Windows Presentation Foundation), un subsistema gràfic per renderitzar interfícies basades en sistemes Windows.

En la secció Activitats del web del mòdul podreu trobar més referències d'aquests llenguatges o d'altres, i també altres dades d'interès.

XAML ha estat desenvolupat per Microsoft. Permet la construcció d'interfícies mitjançant la creació d'una jerarquia d'objectes juntament amb un conjunt de propietats.

XAML està relacionat amb el subsistema de presentació Avalon, i troba actualment al més alt nivell dins de la plataforma .NET (de fet, un document escrit en XAML es compila en classes .NET, normalment en C #).

Hi ha una relació entre cada element XAML i una classe de la plataforma .NET. Els elements de la interfície s'organitzen en forma d'arbre, i l'organització d'aquest arbre és la que determina la visualització i el comportament posteriors de la interfície.

Si es parla de XAML cal també fer referència a **Microsoft Silverlight**. La primera versió va néixer el setembre de l'any 2007, és a dir, és una tecnologia relativament nova que fa la competència a Adobe Flash pel que fa a reproducció de vídeos, animacions i altres recursos interactius.

La base de programació de Microsoft SilverLight és el llenguatge XAML, amb un accés als objectes mitjançant els llenguatges de programació C# i VisualBasic, i és desenvolupat mitjançant l'eina integrada de desenvolupament **Microsoft Expression Blend**.

Adobe Flash

Aplicació multimèdia que ofereix a les pàgines web la possibilitat de tenir animació, vídeo i altres eines interactives. Molt utilitzada pels desenvolupadors web.

XForms (the next generation of web forms)

XForms és un model per a formularis en entorns web per a les especificacions de dades i les interfícies d'usuari desenvolupades en UML

Es tracta d'una proposta del consorci W3C per a l'especificació de formularis per al Web que puguin ser usats en una àmplia varietat de plataformes.

XForms sorgeix com a alternativa per millorar els actuals formularis web que es poden obtenir a partir del llenguatge HTML. Això fa que ofereixi uns components visuals més complets i atractius per aconseguir formar interfícies més usables,

agradables i útils. A més, s'aconsegueix fer una diferenciació entre les especificacions de les dades i el seu propòsit i la representació d'aquestes als formularis.

Precisament, la característica més important que proporciona XForms és que, a diferència dels formularis web habituals en HTML, les especificacions del disseny de la interfície es defineixen per separat de la funcionalitat.

D'aquesta manera el comportament dels formularis no es veu alterat en canviar de plataforma, mentre que l'aspecte de les finestres es determinarà pels recursos de la plataforma on es vulguin mostrar.

AUIML (Abstract User Interface Markup Language)

El llenguatge de programació AUIML és un dialecte de l'XML. *AUIML* significa *Abstract User Interface Markup Language*, és a dir, ‘llenguatge de marcatge per a interfícies d’usuari abstractes’. És un llenguatge desenvolupat per IBM i dissenyat per facilitar la definició semàntica de la interacció entre l’usuari i les interfícies.

AUIML permet una representació de planells, assistents, fulls d'estils, etc. Es tracta d'un llenguatge capaç d'interpretar la informació de posicionament dels elements de les GUI i delegar la construcció de la seva interfície a una plataforma específica. Tota la informació es codifica una sola vegada i es tradueix utilitzant una traducció dependent del dispositiu final. Ha estat dissenyat per ser independent de la plataforma, del llenguatge de programació final i de la tecnologia de desenvolupament.

UIML (User Interface Markup Language)

UIML és un acrònim per a *User Interface Markup Language*, que vol dir ‘llenguatge de marques per a interfícies d’usuari’.

El llenguatge de programació UIML és un metallenguatge que té com a principal objectiu oferir una representació canònica d'una interfície.

El llenguatge UIML és independent de qualsevol altre llenguatge o dispositiu. Una vegada desenvolupat un document en UIML hi ha la possibilitat de convertir-lo a un altre llenguatge de desenvolupament que sigui utilitzat per aquell dispositiu que es farà servir per a la visualització.

UIML proporciona una DTD o XML Schema, en la qual es defineixen les etiquetes permeses (independents de la metàfora d’interfície) i les seves combinacions per a la creació de documents UIML vàlids.

La definició de la interfície es fa en una sèrie de blocs:

- parts que componen la interfície,
- presentació o visualització d'aquestes parts,

- contingut de cadascuna,
- comportament de la interfície,
- conversió de les parts als elements corresponents del llenguatge d'implementació triat,
- connexió amb la lògica d'aplicació.

Aquests blocs faciliten la separació entre els elements que componen la interfície, i es distingeix entre el modelatge estructural, la visualització i el modelatge del comportament.

El llenguatge UIML permet obtenir definicions declaratives d'interfícies d'usuari independents del següent:

- la plataforma,
- el tipus d'interacció, i
- el llenguatge de programació.

Per escriure una interfície en UIML s'ha de fer, d'una banda, una definició de la interfície genèrica, i d'altra banda, un document UIML que representa l'estil de presentació apropiat per al dispositiu en el qual s'arrenca la interfície. D'aquesta manera, una mateixa aplicació només necessitarà un document UIML d'especificació per a qualsevol dispositiu i un document d'estil per a cada dispositiu.

2.3 Casos d'ús: generació d'interfícies a partir de documents XML per a diferents plataformes

Històricament el disseny i desenvolupament d'interfícies gràfiques d'usuari (GUI) són tasques que els dissenyadors i els desenvolupadors de programari han generat de manera estàtica. És a dir, es crea el contingut visual, el disseny de la interfície gràfica, a partir de formularis que contindran controls i components, com poden ser els DataGrids, els PictureBox o altres controls ActiveX. És el que es coneix com a *disseny estàtic d'una interfície gràfica*, que, una vegada lliurat a l'usuari, tindrà el mateix aspecte que el que s'havia dissenyat.

El **disseny dinàmic** d'una interfície gràfica d'usuari consisteix a dissenyar (crear) les interfícies en temps d'execució i no en temps de disseny, de tal manera que la preocupació es trobi més a quadradar la lògica del programari i no tant el disseny estàtic de les interfícies.

Aquest disseny dinàmic és podrà desenvolupar en el servidor o a partir de documents XML que s'hagin transferit al client, implementant el disseny dinàmic al mateix client en temps d'execució.

A continuació es proposen alguns casos pràctics que intenten mostrar les diferències entres aquestes alternatives.

2.3.1 Cas d'ús: disseny estàtic d'una interfície gràfica

La creació d'aquest disseny gràfic es genera mitjançant l'ús dels coneguts entorns de desenvolupament integrats (IDE) que els diferents fabricants llancen al mercat, com ara el Visual Studio.NET de Microsoft o el JBuilder de Borland, entre altres. Per dissenyar i construir aquestes GUI simplement ens cal arrossegar i deixar anar els diferents controls que volem en l'ordre i col·locació que necessitem per completar la interfície segons com la vulguem dissenyar.

A partir d'aquí només cal modificar les propietats dels elements en funció de les necessitats o els gustos del dissenyador i vincular alguns d'aquests elements a esdeveniments planificant les funcionalitats que s'hauran d'executar en cada acció.

Aquest disseny estàtic generat mitjançant un entorn de desenvolupament visual, des del punt de vista pràctic està molt bé, però, què passa si el nostre projecte té, per exemple, 200 formularis i 20 o 30 elements a cada formulari? Doncs que el desenvolupador haurà de dedicar força temps a arrossegar i deixar anar controls per acabar el disseny dels formularis.

La gran pregunta, llavors, és: com podem escurçar aquest temps? La resposta no és tan simple, però tractarem de proposar algunes solucions.

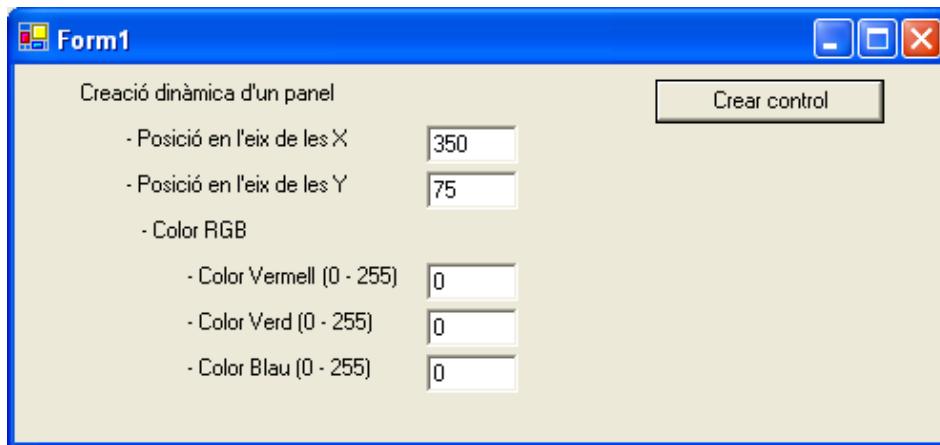
2.3.2 Cas d'ús: disseny dinàmic d'una interfície gràfica en codi de servidor

Per què, en lloc de generar aquestes interfícies d'usuari en temps de disseny, no es creen en temps d'execució? Amb això només cal preocupar-nos de la lògica del nostre sistema, i no hauríem d'estar *dibuixant* la interfície abans que el formulari s'executi i es visualitzi.

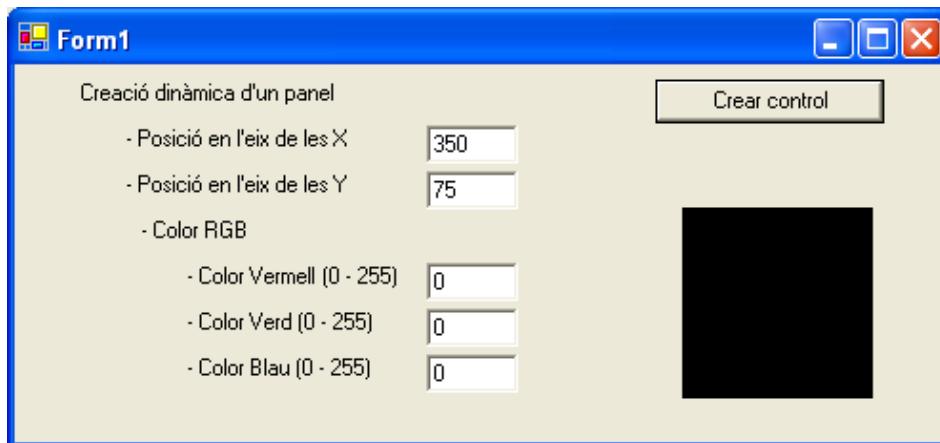
Per aconseguir això hi ha diverses tècniques.

La primera i més simple consistirà en utilitzar la col·lecció de controls que tenen els formularis i que ens permeten, en temps d'execució, mitjançant mètodes com afegir o eliminar, afegir o treure controls de tal manera que sense necessitat de *pintar* res en el disseny, quan el nostre formulari es carregui en memòria, executarà la càrrega o descàrrega de cada un dels objectes que afegiran qualsevol instant a la col·lecció “Controls”.

En l'exemple següent, a la figura 2.16, es crea de manera dinàmica un plafó amb les propietats definides per l'usuari.

FIGURA 2.16. Plafó de propietats

En fer clic en el botó “Crear control”, es crearà un plafó en la posició 350,75 de color negre, com es pot observar a la figura 2.17.

FIGURA 2.17. Creació del plafó

El codi de l'exemple seria:

```

1 Private Sub Button1_Click(ByVal sender As System.Object,
2   ByVal e As System.EventArgs) Handles Button1.Click
3   'Creació del control dinàmic – un panell
4   CrearPicture()
5 End Sub
6
7 Private Sub Form_Load()
8   'Definició dels valors per defecte.
9   TextBox_X.Text = 350
10  TextBox_Y.Text = 75
11  TextBox_ColorR.Text = 0
12  TextBox_ColorG.Text = 0
13  TextBox_ColorB.Text = 0
14 End Sub
15
16 'Procediment que crea el panell en temps d'execució
17 Private Sub CrearPicture()
18   Dim miPanel As Panel()
19
20   'Li establim les propietats del nou panell
21   With miPanel
22     'Característiques predefinides
23     .Visible = True
24     .Width = 100

```

```

25      .Height = 100
26      'Característiques definides per l'usuari
27      .Location = New Point(TextBox_X.Text, TextBox_Y.Text,
28      .BackColor = Color.FromArgb(TextBox_ColorR.Text,
29          TextBox_ColorG.Text, TextBox_ColorB.Text)
30  End With
31
32  'Incorporem el nou control en el formulari
33  Me.Controls.Add(miPanel)
34 End Sub

```

Amb aquesta tècnica es pot generar la interfície d'usuari dinàmicament, en temps d'execució, que és del que es tracta. Però, per què no podem anar més enllà? A més, caldrà també preguntar-nos on s'haurà de dur a terme la creació del formulari, en el servidor que rep la petició i té les dades, o en el client? I si és fa una petita modificació en la sol·licitud de les dades o s'activa alguna funcionalitat del formulari, cal tornar a generar tot el formulari de nou o només enviar les modificacions?

2.3.3 Cas d'ús: disseny dinàmica d'una interfície gràfica a partir de XML

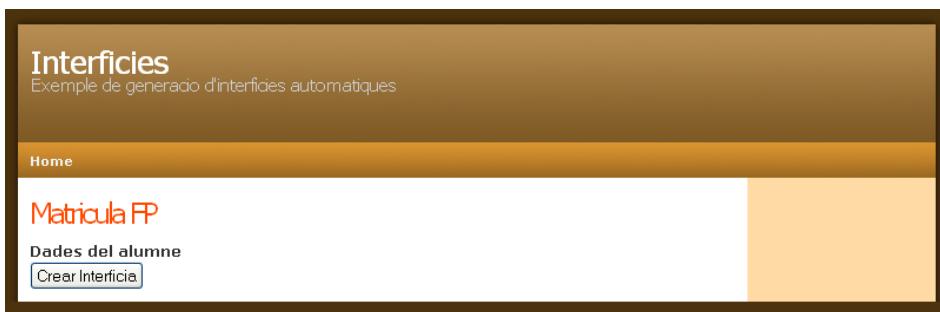
L'objectiu de la generació d'interfícies fent ús d'XML se centra en les dades que interessen del sistema com a contingut i no en el seu aspecte visual. A més, es vol utilitzar aquesta informació afegida com a part principal del desenvolupament i sobre la qual la nostra aplicació s'anirà generant.

Aquesta tècnica de crear aplicacions orientada a les dades i basant-se en l'estructura d'aquestes dades es coneix com a *data driven* i és precisament el que es fa a partir d'un document XML. Les passes que cal seguir són les següents:

1. Establir una relació entre la GUI que es vol desenvolupar, que mostrarà les informacions que ens interessen, i l'estructura de les dades que tenim emmagatzemades al servidor de base de dades.
2. Transformar aquesta informació en controls gràfics perquè siguin carregats i visualitzats per l'aplicació o al navegador.
3. Una vegada que la interfície d'usuari estigui visible, tenir la possibilitat d'interactuar amb aquests controls i enviar la informació introduïda des del navegador a la base de dades o on sigui necessari.

Per fer possible tot això, el primer que s'ha d'obtenir és el document XML que recull la informació que volem, i que servirà com a base per a la creació de la interfície. Aquesta informació es pot obtenir de diverses maneres.

Per explicar el mecanisme de generació d'interfícies a partir d'un document XML ens recolzarem amb l'exemple següent, que es pot observar a la figura 2.18, i que genera de manera automàtica el formulari de matricula per als cicles formatius de la IOC en entorn web.

FIGURA 2.18. Pantalla inicial

En l'exemple, es parteix directament d'un document XML que descriu els controls que tindrà la interfície que es vol desenvolupar; en aquest cas es tracta d'un formulari per a la matrícula d'un cicle formatiu.

```

1  <inscripcion nomIntern="Interfície dinamica">
2      <control nomIntern="PrimerCognom" nomExtern="Primer Cognom:" 
3          requerit="si" tipus="texto"></control>
4      <control nomIntern="SegonCognom" nomExtern="Segon Cognom:" tipus="texto" />
5      <control nomIntern="Nom" nomExtern="Nom:" requerit="si" tipus="texto" />
6      <control nomIntern="DocumentIdentificatiu" nomExtern="Document Identificatiu
7          :"
8          tipus="lista" >
9              <valores valor="DNI (amb lletra)"></valores>
10             <valores valor="NIE (estrangers)"></valores>
11             <valores valor="Passaport"></valores>
12             <valores valor="Altres"></valores>
13         </control>
14     <control nomIntern="NumDocument" nomExtern="Num. Document:" requerit="si"
15         tipus="texto" />
16     <control nomIntern="email" nomExtern="email:" requerit="si" tipus="texto" />
</inscripcion>
```

Una vegada obtingut el document XML que posseeix l'estructura i la informació que es vol utilitzar per generar la GUI, el que es farà ara serà aplicar un full d'estils transformat, XSLT, de manera que la transformació farà possible que aquestes metadades XML es converteixin en controls web ASP.NET.

L'XSLT utilitzat en l'exemple és el que es mostra a continuació.

```

1  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
2      xmlns:asp="remove">
3      <xsl:output method="xml" indent="yes" encoding="utf-8" omit-xml-declaration="
4          yes"/>
5      <xsl:template match="/">
6          <table>
7              <xsl:for-each select="//control">
8                  <tr>
9                      <td valign="top">
10                         <xsl:value-of select="@nomExtern" />
11                         <xsl:if test="@requerit = 'si'">
12                             <asp:RequiredFieldValidator ErrorMessage=" Campo Obligatorio"
13                                 runat="server" ControlToValidate="{@nomIntern}" />
14                         </xsl:if>
15                     </td>
16                     <td>
17                         <xsl:if test="@tipus='etiqueta'">
18                             <asp:Label id="{@nomIntern}" runat="server" />
19                         </xsl:if>
20                         <xsl:if test="@tipus='texto'">
21                             <asp:TextBox id="{@nomIntern}" runat="server" />
22                         </xsl:if>
```

```

22      <xsl:if test="@tipus='radio'">
23          <asp:RadioButtonList id="{@nomIntern}" runat="server">
24              <xsl:for-each select="valores">
25                  <asp:ListItem Value="{@valor}">
26                      <xsl:value-of select="@valor" />
27                  </asp:ListItem>
28              </xsl:for-each>
29          </asp:RadioButtonList>
30      </xsl:if>
31      <xsl:if test="@tipus='lista'">
32          <asp:DropDownList id="{@nomIntern}" runat="server">
33              <xsl:for-each select="valores">
34                  <asp:ListItem Value="{@valor}">
35                      <xsl:value-of select="@valor" />
36                  </asp:ListItem>
37              </xsl:for-each>
38          </asp:DropDownList>
39      </xsl:if>
40      </td>
41  </tr>
42 </xsl:for-each>
43 </table>
44 <asp:button id="submit" runat="server" Text="Enviar" />
45 </xsl:template>
46 </xsl:stylesheet>
```

El que fa el full d'estils XSLT és rastrejar el document XML i, a mesura que va localitzant les diferents etiquetes que fan referència a les metadades descrites, les transforma en els controls web ASP.NET corresponents.

El codi necessari perquè la nostra pàgina ASP.NET carregui i visualitzi el document XML ja transformat estarà dins d'un mètode anomenat *Cargarformulario()*, que s'utilitza des de l'esdeveniment *Page_Init* del WebForm, i que utilitzar els diferents *namespaces* relacionats amb XML, que s'importaran des del principi del codi i es col·locaran abans de la declaració de la classe. El codi de l'exemple seria:

```

1 Imports System.Xml.XPath
2 Imports System.Xml.Xsl
3 Imports System.IO
4 Public Class WebForm1
5
6 Inherits System.Web.UI.Page
7
8 #Region " Código generado por el Diseñador de Web Forms "
9
10    <System.Diagnostics.DebuggerStepThrough()>
11    Private Sub InitializeComponent()
12
13    End Sub
14
15    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As System.
16        EventArgs)
17        Handles MyBase.Init
18            InitializeComponent()
19            CargarFormulario()
20        End Sub
21
22 #End Region
23
24    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.
25        EventArgs)
26        Handles MyBase.Load
27
28        Private Sub CargarFormulario()
```

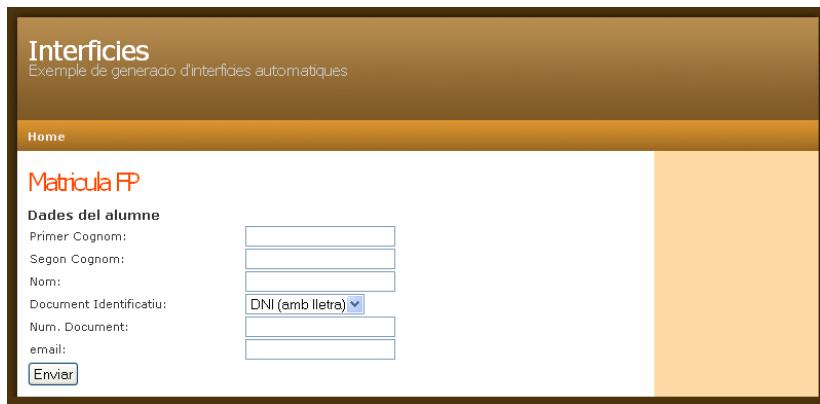
```

29     'Carreguem la informació
30     Dim docInscripcion As New XPathDocument(Server.MapPath("inscripcion.xml"))
31         )
32
33     'Transformem la informació a web controls mitjançant xslt
34     Dim transformacion As New XslTransform()
35     transformacion.Load(Server.MapPath("transf-controles.xslt"))
36
37     'Obtenim els resultats de la transformació
38     Dim sw As New StringWriter()
39
40     'Convertim de l'objecte XML en un string per poder-lo manipular.
41     transformacion.Transform(docInscripcion, Nothing, sw)
42
43     'Obtenim el resultat en un string
44     Dim resultado As String
45     resultado = sw.ToString()
46
47     'Traiem el namespace asp...
48     resultado = resultado.Replace("xmlns:asp=""remove""", "")
49
50     'Afegim els web controls al formulari
51     Dim controles As New Control()
52     controles = Page.ParseControl(resultado)
53     Me.FindControl("Form1").Controls.Add(controles)
54
55 End Sub
56
End Class

```

A la figura 2.19 es pot veure la interfície amb el resultat final.

FIGURA 2.19. Controls generats en temps d'execució a partir d'un XML



Components visuals. Usabilitat

Marcel García

Desenvolupament d'interfícies

Índex

Introducció	5
Resultats d'aprenentatge	7
1 Components visuals	9
1.1 Introducció als components visuals	9
1.1.1 Components visuals: concepte i característiques	9
1.1.2 Propietats i atributs dels components visuals	12
1.1.3 Esdeveniments vinculats	14
1.1.4 Tipus de components: classificacions	15
1.1.5 Alguns exemples de components	18
1.1.6 Plataformes rich client i thin client (client lleuger)	20
1.1.7 Biblioteques de components per a thin i rich clients	24
1.1.8 Utilitat dels components en el desenvolupament d'aplicacions	25
1.2 Creació d'un component visual	26
1.2.1 Eines per a dissenyar i crear components visuals	27
1.2.2 Definició de les propietats i mètodes	28
1.2.3 Esdeveniments als quals ha de respondre un component	32
1.2.4 Proves unitàries i documentació	33
1.3 Persistència i serialització	35
1.3.1 Persistència dels components visuals	35
1.3.2 Serialització	36
1.3.3 La serialització binària	36
1.3.4 La serialització en XML	38
1.4 Empaquetat de components	40
1.4.1 Empaquetat d'un thin client	41
1.4.2 Empaquetat d'un rich client	42
2 Usabilitat	45
2.1 Introducció a la usabilitat	45
2.1.1 Què és la usabilitat?	46
2.1.2 Dimensions de la usabilitat: característiques i atributs	48
2.1.3 Normes ISO relacionades amb la usabilitat	51
2.1.4 Mesura d'usabilitat d'aplicacions: tipus de mètriques	53
2.1.5 Proves d'experts i proves amb usuaris	55
2.2 Pautes de disseny de les interfícies d'usuari	56
2.2.1 Disseny de l'estructura de les interfícies d'usuari	59
2.2.2 Disseny de l'aspecte de les interfícies d'usuari	68
2.2.3 Disseny dels elements interactius de les interfícies d'usuari	75
2.3 Altres pautes de disseny	79
2.3.1 Presentació de les dades	80
2.3.2 Seqüència de control de l'aplicació	81
2.3.3 Assegurament de la informació	82

2.3.4 Específiques per a aplicacions multimèdia	83
-----------------------------------------------------------	----

Introducció

Les aplicacions informàtiques o els llocs web duen a terme una interacció contínua amb els usuaris. Els elements que es podran fer servir per a aquesta interacció poden ser una combinació de dispositius de maquinari i/o tecnologies de programari. Aquests elements ajuden a crear una plataforma amb interfícies que permeten a l'usuari interactuar amb les aplicacions i formen una mena de llenguatge visual que s'ha anat desenvolupant per aconseguir que els usuaris amb poca experiència amb les noves tecnologies i, més concretament, amb els ordinadors comprenguin les interfícies.

En l'apartat “Components visuals” veurem els components visuals que habitualment tenim disponibles, les característiques (propietats i atributs) i com podem fer que interactuin amb nosaltres (esdeveniments). També estudiarem com podem crear els nostres propis components visuals.

Així mateix, en l'apartat “Usabilitat” abordarem un tema tan important com és la usabilitat, que ens permetrà veure com podem adaptar el programari a l'estil de treball real dels usuaris. En aquesta part veurem tant normes ISO que tenen a veure amb la usabilitat com pautes de disseny que afecten aspectes com els colors, les fonts, els menús, etc.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Crea components visuals valorant i emprant eines específiques.
 - Identifica les eines per al disseny i prova de components.
 - Crea components visuals.
 - Defineix les seves propietats i assignar valors per defecte.
 - Determina els events als que ha de respondre el component i se'ls associa a les accions corresponents.
 - Fer proves unitàries sobre els components desenvolupats.
 - Documenta els components creats.
 - Empaqueta els components.
 - Programa aplicacions amb interfície gràfica que utilitzi els components creats.
2. Dissenya interfícies gràfiques identificant i aplicant criteris d'usabilitat
 - Crea menús que s'ajustin als estàndards.
 - Crea menús contextuels, l'estructura i contingut dels quals segueixin els estàndards establerts.
 - Distribueix accions en menús, barres d'eines, botons d'ordres, entre d'altres, seguint un criteri coherent.
 - Distribueix adequadament els controls en les interfícies d'usuari.
 - Utilitza el tipus de control més adequat en cada cas.
 - Dissenya l'aspecte de la interfície d'usuari (colors i fonts entre d'altres) atenent a la seva llegibilitat.
 - Verifica que els missatges generats per l'aplicació són adequats en extensió i claredat.
 - Fer proves per a avaluar la usabilitat de l'aplicació.

1. Components visuals

Les aplicacions informàtiques o molts llocs web duen a terme una interacció contínua amb els usuaris, per mitjà de la qual aquests poden seleccionar i executar funcionalitats per a la consulta o la modificació de dades.

Els elements que es podran fer servir per a aquesta interacció poden ser una combinació de dispositius de maquinari i/o tecnologies de programari. Aquests elements ajuden a la creació d'una plataforma amb interfícies que permeten a l'usuari interactuar amb les aplicacions. Els elements formen una mena de llenguatge visual que s'ha anat desenvolupant per aconseguir l'entesa amb les interfícies per part dels usuaris amb poca experiència amb les noves tecnologies i, més concretament, amb els ordinadors.

1.1 Introducció als components visuals

El nom amb què es coneix el conjunt d'elements visuals d'una interfície gràfica juntament amb la propia interfície, en un entorn d'ordinador personal, és el de WIMP (*windows, icon, menu, pointer device*). Un conjunt de finestres que contindran diverses icones i altres elements com menús són la base per a la creació d'interfícies que oferiran, a més a més, la possibilitat de dur a terme accions a partir del moviment del punter per part dels usuaris. Aquesta serà la base d'aquest estil d'interacció.

El pas següent ha estat la creació d'interfícies per als darrers dispositius que han arribat al mercat, com poden ser dispositius mòbils, *tablet PC* o telèfons. És coneix com a *estil d'interacció post-WIMP*, i un exemple seria el sistema operatiu Android.

Els components visuals són la base de l'estil d'interfícies WIMP. Aquests components tenen propietats i atributs, es podran vincular a esdeveniments, i l'usuari els podrà crear de nou per a completar els que ja té a la seva disposició.

Android

És un sistema operatiu per a dispositius mòbils desenvolupat per Google basat en el nucli (*kernel*) del Linux.

1.1.1 Components visuals: concepte i característiques

Les interfícies gràfiques d'usuari utilitzen elements visuals per a representar la informació i les dades a mostrar. Aquests elements es faran servir per a crear les estructures estàtiques o dinàmiques que es convertiran en les interfícies que interactuaran amb els usuaris.

Classe

Una classe és un conjunt de propietats i mètodes relacionats amb un significat propi, en un context de programació orientada a objectes.

Un component visual és una classe d'ús específic, a punt per a ser arrossegada a un formulari, que es podrà configurar o utilitzar de manera visual, des de l'entorn de desenvolupament. Són elements visuals que permeten la gestió i representació d'informació.

La diferència principal, respecte a una classe normal, és que la major part de la feina de configuració de l'element es pot fer de manera visual, amb el ratolí, i ajustant les opcions i propietats que s'ofereixen en el seu entorn. De fet, es pot crear una interfície gràfica d'usuari sense programar cap línia de codi.

Cada component té associats un conjunt de propietats i mètodes, i un conjunt d'esdeveniments als quals pot respondre. Amb l'inspector d'objectes es poden modelar els components d'una aplicació segons les necessitats que es tinguin, quant a la seva aparença (propietats) i funcionalitat (esdeveniments als quals pot respondre). En definitiva, es poden modificar les propietats dels components i construir els gestors d'esdeveniments a què aquests poden respondre.

IDE

Entorn de desenvolupament d'applicacions integrat.

Si es fa servir una eina tipus IDE per al desenvolupament d'applicacions s'han de tenir a disposició molts components integrats que es podran fer servir per al disseny de les interfícies gràfiques d'usuari. Una vegada es treballi amb un formulari es podran seleccionar i arrossegar cap a aquest formulari un o més components, i se'n podran modificar les característiques i propietats.

Aquest tipus d'acció i algunes altres es poden dur a terme en temps de disseny, és a dir, mentre es dissenya i es programa l'aplicació, al contrari d'altres accions que es duen a terme en temps d'execució, quan l'aplicació ja està en funcionament.

Alguns exemples de components més simples són:

- Botons
- Llistes desplegables
- Barres de progrés
- Etiquetes
- *Radio buttons*
- ...

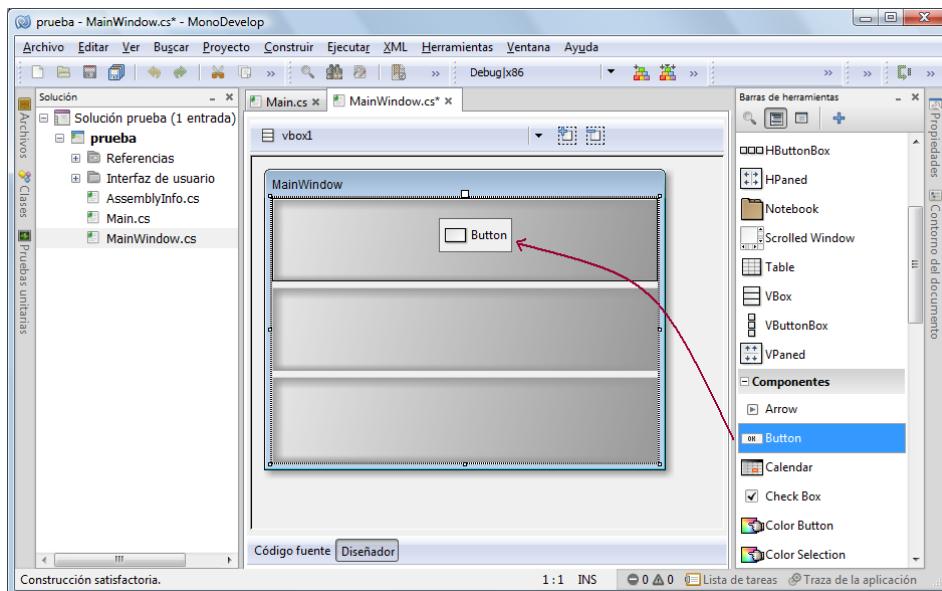
Alguns exemples de components més complexos són:

- Menús
- Taules
- Arbres
- Quadres de diàleg
- ...

Els components tenen, entre d'altres, les característiques conceptuais i de creació següents:

- Permeten la interacció amb altres components.
- Són una unitat de programari compilada reutilitzable, amb una interfície ben definida.
- Proporcionen el control sobre determinats recursos externs en temps de disseny.
- Es distribueixen en un únic paquet que conté en si mateix tot el necessari per al seu funcionament, amb cap o pràcticament cap dependència respecte d'altres components o biblioteques.
- Es poden haver desenvolupat en qualsevol llenguatge de programació (normalment s'utilitzen llenguatges orientats a objectes).
- Es poden utilitzar per al desenvolupament d'aplicacions en qualsevol llençatge de programació.
- Poden ser visibles o no visibles per a l'usuari en temps d'execució, encara que sempre seran visibles per al desenvolupador en temps d'execució.

FIGURA 1.1. Exemple de component-button

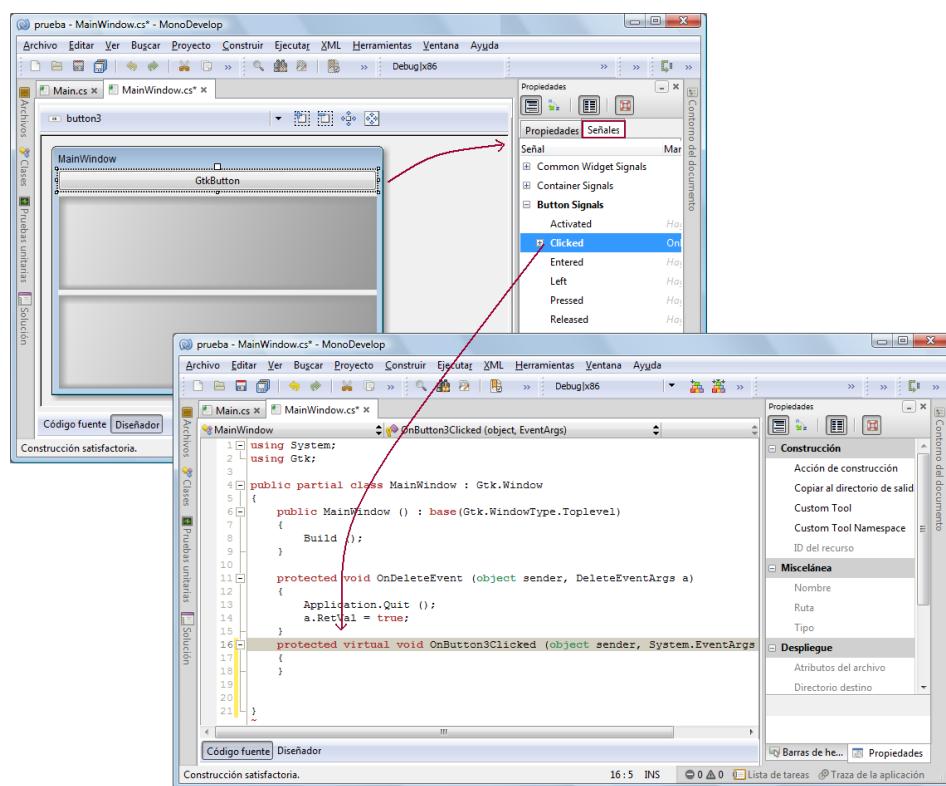


Respecte a la funcionalitat, els components també disposen de les característiques següents:

- Els components instal·lats són accessibles en l'entorn de desenvolupament per mitjà d'una paleta de components i es poden arrossegar a l'aplicació en què s'està treballant. En la figura 1.1 se'n pot veure un exemple.
- En cas de tractar-se de components visuals, és a dir, relacionats amb la interfície d'usuari i visibles per als usuaris finals, és possible manipular-los de manera interactiva.

- Cada component exposa públicament un conjunt de propietats que indiquen els valors de les seves característiques, és a dir, el seu estat intern. La majoria d'aquestes propietats poden ser modificades de manera interactiva pel desenvolupador. En la figura 1.2 se'n pot veure un exemple per al component *button* en què es poden identificar algunes de les seves propietats visuals i d'accions.
- Cada component genera una sèrie d'esdeveniments que es poden capturar. El codi associat a aquests esdeveniments es fa en el llenguatge natiu de l'entorn de desenvolupament: Visual Basic, c++, Java, c#, etc. En la figura 1.2 es poden veure els esdeveniments associats al component *button* i un exemple de codi vinculat al component desenvolupat en Java.

FIGURA 1.2. Propietats i esdeveniments del botó



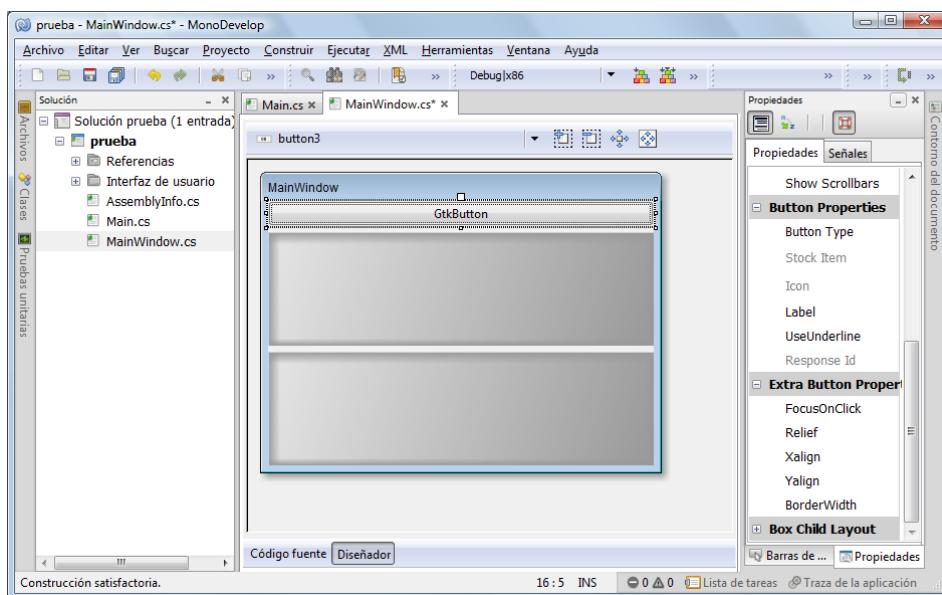
1.1.2 Propietats i atributs dels components visuals

Les propietats i atributs d'un element visual mostren totes les possibles característiques d'aquell component, des de les més senzilles (aquelles que tenen a veure amb els atributs gràfics del component, com el color, les mides, el color quan el ratolí passa per sobre del component...) fins a les més complexes o elaborades, com la forma de la que actuarà quan es faci clic amb el ratolí a sobre o com s'ha de comportar el component quan succeeixi un esdeveniment vinculat amb ell.

Els atributs o propietats d'un component es poden modificar mitjançant codi (en temps d'execució) o mitjançant la finestra de propietats (en temps de disseny). Aquest segon cas és el més habitual en el desenvolupament d'aplicacions en entorns IDE. Es una acció molt important per als dissenyadors d'interfícies i per als desenvolupadors d'aplicacions el fet de poder modificar els valors d'aquestes propietats dels components, formularis o controls, ja que serà una eina bàsica per a complir amb els requeriments inicials del projecte.

En la figura 1.3, es pot veure un exemple corresponent a un entorn de desenvolupament en què s'identifiquen algunes de les propietats relacionades amb el component de tipus botó.

FIGURA 1.3. Propietats d'un component



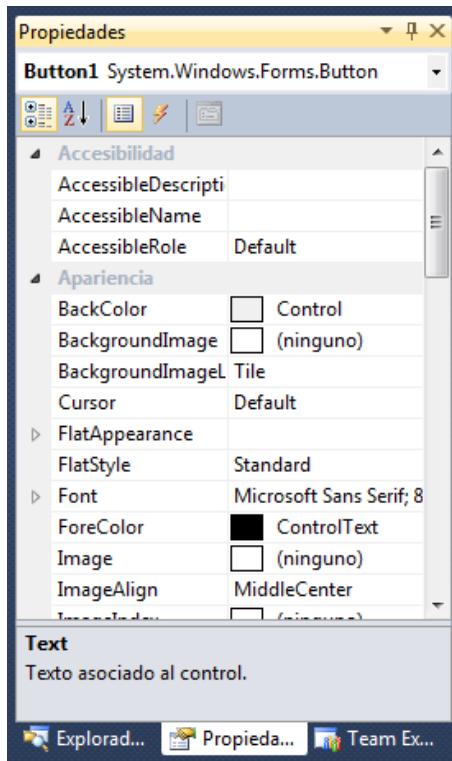
Com es pot veure en la figura 1.4, corresponent a un altre IDE, Visual Studio, la finestra de propietats mostra en la part superior una llista de les classes disponibles en un entorn donat (per exemple, si som en el dissenyador d'un formulari, només mostrarà els elements incorporats a aquest formulari) i sota una llista de les propietats de la classe, i indica les que es poden modificar i les que no, marcant aquestes últimes en gris.

Modificar els valors de les propietats amb l'ajuda de la finestra de propietats és una acció molt còmoda i senzilla per l'accessibilitat d'aquestes en temps de disseny.

A més a més de les propietats, els components també disposaran de mètodes associats.

Un mètode és un tros de codi associat a un component que s'executa després d'una acció o esdeveniment o un missatge per part d'un altre component.

FIGURA 1.4. Propietats d'un component en el Visual Studio



1.1.3 Esdeveniments vinculats

Durant el temps d'execució d'una aplicació es podran executar per part de l'usuari o podran succeir de manera automàtica diferents successos o esdeveniments.

Aquests successos podran ser síncrons (governats per un rellotge que controla els moments o ritmes de determinades accions) o asíncrons (no es pot saber quan poden succeir diferents accions externes com una interacció d'un usuari).

Query

Consulta a executar sobre una base de dades.

Els llenguatges de programació permeten executar parts de codi com a resposta a accions o esdeveniments que, normalment, seran asíncrons, durant l'execució de l'aplicació. Com a exemple, si es fa clic en un component determinat, s'executarà una acció determinada (mostrar, per exemple, per pantalla el resultat d'una consulta o *query*).

Qui podrà desencadenar un esdeveniment? Normalment aquest tipus de situacions es produïxen des de l'exterior de l'aplicació. Un element de maquinari, com ara un ratolí o un teclat o una pantalla tàctil, poden ser exemples d'elements capaços de desencadenar un esdeveniment que aturi l'execució en aquell moment de l'aplicació per executar una altra funcionalitat o part de codi.

Dintre de les seves propietats, els components han de disposar de la possibilitat de vincular parts de codi al fet que es produueixin alguns esdeveniments o, també, la

possibilitat de no vincular res a aquell esdeveniment. Per exemple, en un mateix formulari es podran incloure dos components iguals, i configurar en el primer una determinada acció en accionar-lo l'usuari mitjançant un clic de ratolí, però deixar sense configurar el segon botó per la mateixa acció.

Alguns exemples d'esdeveniments relacionats amb els ratolins per a un component qualsevol podrien ser:

- **Click.** Es podrà indicar la part de codi a executar en fer clic amb el ratolí (botó esquerre) a sobre del component.
- **MouseOver.** Aquest esdeveniment indica que el ratolí és a sobre d'un component determinat.
- **MouseMove.** Aquest esdeveniment té lloc quan l'usuari mou el ratolí per sobre d'un component.
- **MouseDown.** Quan l'usuari fa clic amb el ratolí a sobre de qualsevol botó, el dret o l'esquerre.
- **MouseOut.** Aquest esdeveniment indicarà que el punter del ratolí ha sortit d'una certa zona de la interfície o que deixa d'estar a sobre d'un component.
- **MouseClicked.** És una altra manera d'anomenar a l'esdeveniment *Click* (un clic amb el ratolí a sobre d'un component).
- **MouseDouble-Clicked.** Aquest component es produeix quan es fa clic dues vegades de manera consecutiva a sobre d'un component amb el ratolí.
- **MouseEntered.** Equivalent al *MouseOver*. Per exemple, en passar a sobre d'un component amb un ratolí, el text que aquest contingui en modificarà el color.
- **MouseExited.** Equivalent al *MouseOut*. Per exemple, en sortir amb el ratolí d'un component, el color del text que aquest contingui tornarà al color original.

1.1.4 Tipus de components: classificacions

Hi ha moltes variables que oferiran moltes maneres diferents de classificar els components de les interfícies gràfiques d'usuari. En aquest apartat es farà referència a algunes d'aquestes variables i, en conseqüència, a algunes de les possibles classificacions dels components:

- Visuals / no visuals
- Si són controls
- Si són contenidors

- Components estructurals
- Components d'interacció
- Components avançats per al web 2.0

La classificació dels components en visuals i no visuals es basa en el fet que el component sigui visible o no per a l'usuari final de l'aplicació.

Un component és visual quan té una representació gràfica en temps de disseny i execució (botons, barres de desplaçament o *scroll*, quadres d'edició, etc.), i es diu *no visual* en cas contrari (temporitzadors, quadres de diàleg -no visibles en la fase de disseny-, etc.).

D'altra banda, no hi ha més diferències entre ells, excepte, és clar, les derivades de la visualització del component.

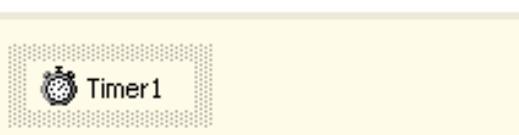
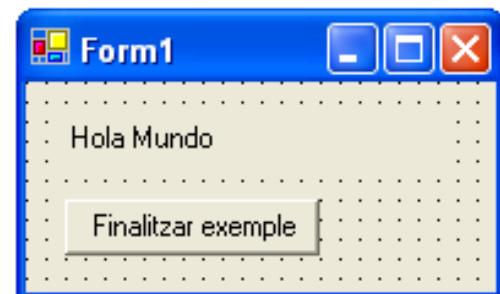
Els components no visuals es poden col·locar en els formularis de la mateixa manera que els controls, encara que en aquest cas la posició és irrellevant.

En aquest tema es farà referència a components visuals.

En la figura 1.5 es mostra un exemple amb les diferències entre components visuals i components no visuals. Es mostra un formulari amb tres components:

- Un primer component mostra una etiqueta de text, un component visual, que dóna un missatge als usuaris.
- Un segon component mostra un botó o *button*, un altre component visual, que dóna l'opció de finalitzar l'exemple (si hi ha un esdeveniment per part d'un usuari que fa clic en aquest botó, es tancarà el formulari).
- Finalment hi ha un tercer component, aquest no visual. És un rellotge. Els usuaris no podran veure aquest component.

El que es mostra en la figura 1.5 és la interfície en temps de disseny, en què el desenvolupador sí que la podrà veure. Aquest component, un rellotge, farà que al cap d'un temps determinat es tanqui sol el formulari.

FIGURA 1.5. Interfície en temps de disseny

Una altra classificació possible dels components serà segons la seva funcionalitat.

Els components de tipus contenidor són components que n'agruparan d'altres, com, per exemple, els formularis o les finestres.

Els components estructurals són els que serveixen per a crear l'estructura estàtica amb què l'usuari podrà interactuar. Exemples d'aquests tipus de components poden ser les finestres, els menús o les icones.

Els components d'interacció permeten als desenvolupadors d'aplicacions intercanviar accions amb els usuaris, oferint-los informacions i recollint-ne les respostes.

Un exemple de components d'interacció són els components de tipus control (conegeuts directament com a *controls*).

Els controls són objectes que proporcionen una interacció entre l'aplicació i els usuaris i l'intercanvi d'informació.

Exemples de components de tipus control són:

- Quadre de llista o *list box*
- *Combo box*
- Icona o *icon*
- *Tree view*
- Botó o *button*
- Casella de selecció o *check box*
- *Radio button*

- Llista desplegable o *drop-down list*
- Menú o *menu*
- Barra de menú o *menu bar*
- Barra d'eines o *toolbar*
- *Grid view*

En la secció Annexos del web del mòdul podeu trobar alguns exemples dels tipus de components enumerats.

Un altre tipus de components són els que ofereixen la possibilitat d'utilitzar nous efectes, més dinàmics, amb més interacció amb els usuaris, components per a interfícies d'usuaris avançades. Són els denominats *components per al web 2.0*. Un exemple de components d'aquestes característiques els ofereix l'eina jQuery, entorn de treball (*framework*) que ofereix moltes oportunitats d'aquests tipus de components.

jQuery estalvia moltes hores de desenvolupament als programadors i haver de baixar a un nivell massa baix de programació. Moltes de les funcionalitats i components necessaris ja estan implementats per aquesta eina o les seves biblioteques.

1.1.5 Alguns exemples de components

Hi ha molts components en el mercat, tant lliures com de pagament i per a molts llenguatges de programació i entorns diferents. N'hi ha molts que són comuns a diferents entorns de treball. Es podrien considerar components universals que tothom coneix i són els més habituals en les interfícies gràfiques d'usuari.

Entre aquests components més habituals, a continuació es fa un petit recordatori d'alguns dels més habituals que es faran servir, sobretot, en desenvolupaments de projectes Winforms:

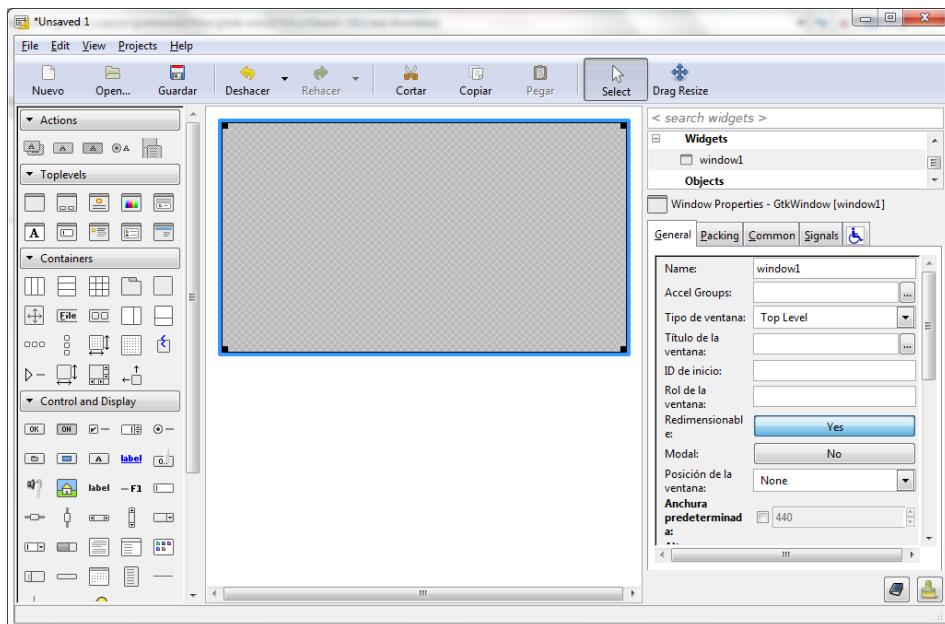
- **Labels (etiquetes)**, que ofereixen la possibilitat d'ubicar en el formulari una etiqueta amb un text determinat, que no podrà ser manipulat per l'usuari de l'aplicació.
- **Buttons (botons)**, que executaran una funcionalitat quan s'hi faci un clic o un doble clic.
- **Picture box (imatges)**, que permeten seleccionar una imatge o un gràfic d'un arxiu prèviament existent.
- **Progress bar (barra de progrés)**, que ofereix la progressió en l'execució d'una funcionalitat determinada que s'està executat.
- **Radio buttons (botons d'opción)**, que són un conjunt d'opcions agrupades que ofereixen diferents alternatives a l'usuari per a poder-ne escollir una.

- **Check box (casella de selecció)**, que són un conjunt d'opcions amb un espai en forma de quadrat a la dreta per a poder-les seleccionar.
- **List box (quadre de llista)**, que mostra una llista d'opcions en format de llista.
- **Text box (quadre de text)**, que mostra informació en format de text escrit en temps de disseny i recull informació introduïda per l'usuari en temps d'execució.
- **Combo box**, que ofereix una llista d'opcions entre les quals se'n podrà seleccionar una o bé introduir un text en el quadre d'edició.

En la figura 1.6 es mostra una finestra de components de l'eina GLADE. Es pot veure com en aquesta eina de disseny d'interfícies gràfiques els components estan agrupats per diferents temàtiques:

- Windows Forms
- De dades
- Components
- XSL-T

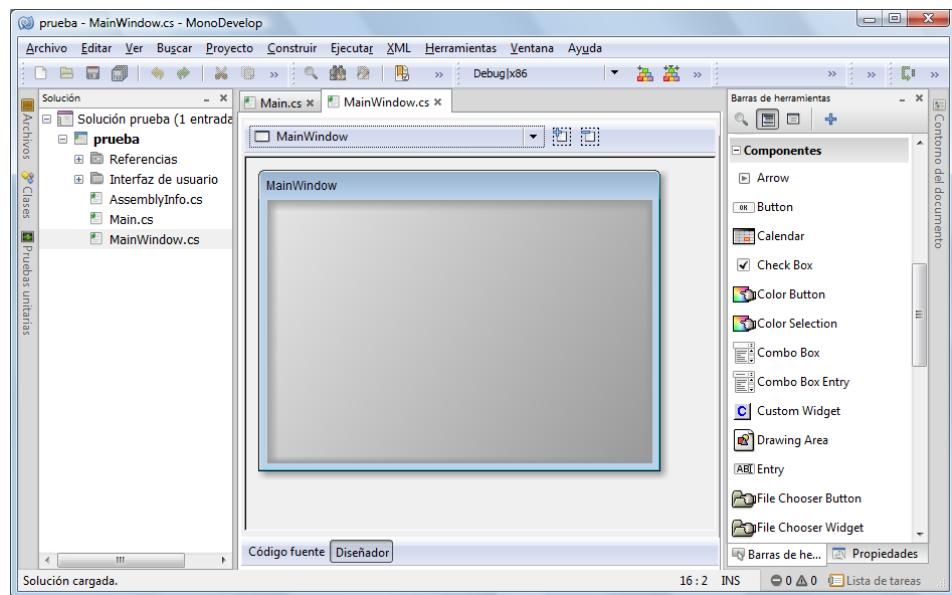
FIGURA 1.6. Components de GLADE



En la figura 1.7 es mostra un altre exemple de components que podem trobar en una altra eina de desenvolupament d'interfícies com MonoDevelop.

GLADE

GLADE és una eina lliure que ajuda a la creació d'interfícies gràfiques d'usuari, molt utilitzada en entorns XML.

FIGURA 1.7. Components de MonoDevelop

1.1.6 Plataformes rich client i thin client (client lleuger)

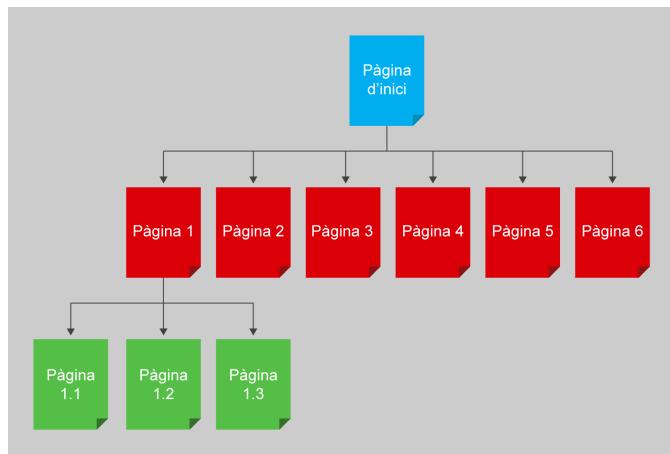
Una altra manera de classificar els components visuals és tenint en compte per a quin tipus d'entorn s'han desenvolupat. Per a això primer cal diferenciar alguns entorns que hi ha actualment en el desenvolupament i utilització d'aplicacions informàtiques amb interfícies gràfiques d'usuari.

D'una banda, hi ha les plataformes anomenades *thin client*, en què els programes i aplicacions desenvolupades depenen en gran mesura d'altres programes per a la seva execució. D'altra banda, hi ha les plataformes anomenades *rich client*, en què els desenvolupadors poden trobar totes les eines per a crear de manera completa una aplicació independent.

Plataformes thin client

En els seus inicis, Internet oferia portals web que simplement eren una gran col·lecció de pàgines estàtiques que oferien documents, imatges, etc. que es podien consultar i/o descarregar.

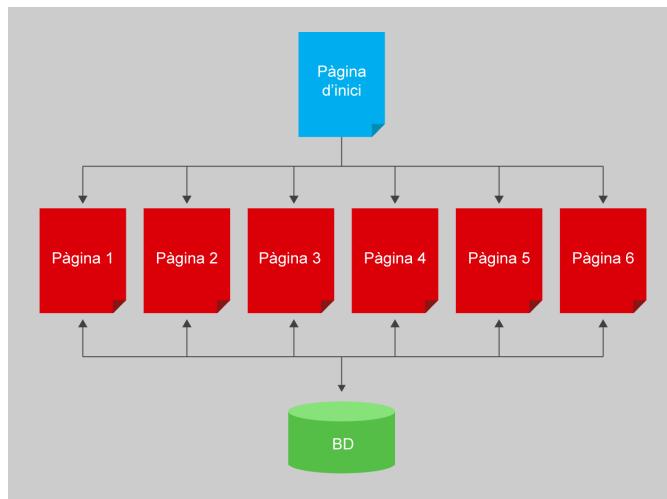
En la figura 1.8 es pot veure una representació d'aquesta situació en què la jerarquia era una característica molt important per a poder fer un manteniment apropiat d'un espai web.

FIGURA 1.8. Esquema d'una pàgina web estàtica

Al llarg dels anys hi ha hagut (i encara hi ha) una evolució contínua d'aquesta estructura de treball. A aquesta evolució també han anat lligats els continguts que s'ofereixen i els serveis que es poden trobar en qualsevol pàgina web que han diferit molt del text i imatges inicials.

El pas següent en la seva evolució va ser incloure un mètode per a la confecció de pàgines web amb continguts dinàmics que permetessin que allò que es mostrava variés en funció de les peticions enviades al servidor web.

En la figura 1.9 és pot veure com a partir d'una pàgina d'inici pengen diverses pàgines dinàmiques que no contindran tots els continguts, sinó que serviran per a accedir a la base de dades per a recollir i mostrar les informacions sol·licitades.

FIGURA 1.9. Esquema d'una pàgina web dinàmica

És en aquest moment quan es pot començar a parlar de thin clients o clients lleugers, és a dir, es comencen a desenvolupar aplicacions de formularis utilitzant un enfocament de client lleuger.

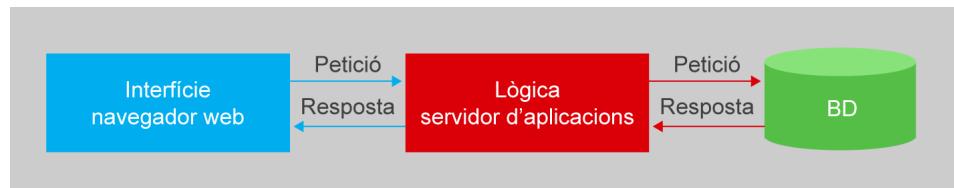
Un client lleuger se sol executar en un navegador que es troba en un ordinador client. El codi dels programes que es poden desenvolupar amb eines com poden ser els llenguatges de marques HTML, llenguatges de *script*, com JavaScript, entre

altres, es troba en un ordinador servidor al qual s'ha d'anar a recollir la informació que es vulgui mostrar.

L'avantatge principal de l'enfocament de client lleuger és la facilitat d'implementació i la possibilitat de treballar de manera independent per capes.

De fet, tenint en compte les múltiples opcions que hi ha per a desenvolupar una aplicació en un entorn web, generalment aquestes estan estructurades seguit un model basat en tres nivells.

FIGURA 1.10. Estructura basada en tres nivells



En la figura 1.10 es poden veure els tres nivells:

- El primer nivell és la interfície o navegador web. Les funcions que es fan en aquest nivell es limiten a enviar les peticions/consultes que genera l'usuari, obtenir les dades generades per l'aplicació web (servidor) i finalment representar aquestes dades en el navegador per a l'usuari.
- En el segon nivell se situaria la lògica. En aquest nivell es troba en el nucli principal de l'aplicació i és l'encarregat de dotar l'aplicació web del contingut dinàmic, interpretant les consultes generades pel client en el nivell de la interfície, executant la lògica necessària per a les consultes que s'enviaran a la base de dades. Una vegada que hagi rebut les dades sol·licitades, les enviarà al nivell de la interfície i les mostrarà per mitjà del navegador.
- En el tercer nivell és on hi ha les dades. Aquest nivell podria estar format per una base de dades, per fitxers XML o per qualsevol altre tipus de sistema d'emmagatzematge de dades dependent de les necessitats de l'aplicació. Només es relacionarà amb el nivell lògic del qual rebrà les seves peticions i al qual lliurarà les seves respostes.

Però, a mesura que passa el temps, es detecta que el fet que la lògica de l'aplicació web s'executi íntegrament en el servidor i que el navegador web només es limiti a la representació de les dades (client lleuger) fa que els programadors vegin limitades les seves capacitats a l'hora de desenvolupar una aplicació web. A més, aquest sistema de treball obliga a una transmissió important de dades de manera contínua, i no optimitza les possibilitats que també ofereixen els clients.

Alguns dels problemes més habituals amb què es poden trobar els clients lleugers

són:

- Efectes com *drag & drop* (arrossegar i deixar anar) i canvi de mida d'elements són impossibles de dur a terme en un ordinador client en un navegador. Passa el mateix amb les actualitzacions de zones específiques de la pàgina sense la necessitat d'haver de renderitzar una nova pàgina HTML.
- Els navegadors interpreten els llenguatges basats en *scripts*, com el JavaScript, de diferents maneres, de vegades incompatibles entre elles. Això provoca que els desenvolupadors hagin d'escriure el codi múltiples vegades acomodant la programació per a cada navegador. L'HTML és un llenguatge estàtic basat en etiquetes molt limitat que no es pot ampliar.
- La serialització (emmagatzemar l'estat d'un objecte en un mitjà) de l'estat de l'aplicació només s'aconsegueix utilitzant galetes (*cookies*), les quals no suporten objectes.

Plataformes rich client

S'està vivint una evolució tant en els desenvolupaments d'aplicacions com en la seva estructura i manera de treballar. Si un dels errors és haver d'enviar molta informació des del servidor fins al client, se cerquen sistemes per a no haver de carregar tanta informació cada vegada i fer més participatiu l'ordinador client.

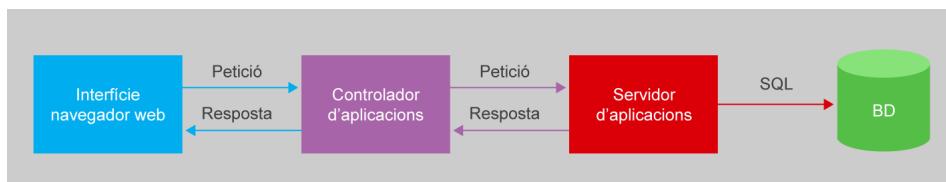
S'ha arribat a un punt en què les aplicacions web no solament han d'ofrir multitud de funcions, sinó que a més han de ser el més òptimes possibles, i les seves interfícies gràfiques han de ser molt més impactants i còmodes per als usuaris. Per a aconseguir aquests objectius s'ha creat un nou tipus de components (anomenats *rich client*) amb l'objectiu d'optimitzar les comunicacions de dades entre client i servidor, i a més ofereix unes interfícies molt més atractives per a l'usuari.

Els clients enriquits fan servir moltes de les característiques i les facilitats del sistema operatiu en què s'executa, la qual cosa soluciona moltes de les mancances dels llenguatges de marques.

Una altra de les característiques dels clients enriquits és la possibilitat d'utilitzar una base de dades local, traslladant part de la càrrega de processament en el dispositiu, i disminuint el nombre de peticions al servidor i les transferències per la xarxa.

D'aquesta manera, es podria dir que l'estructura de les *rich Internet applications* és la que es mostra en la figura 1.11.

FIGURA 1.11. Estructura en quatre capes



En aquest cas s'ha passat de disposar de tres nivells a disposar-ne de quatre:

- Primer tenim el navegador web, que conté el connector o *plug-in (rich client)* necessari per a interpretar les dades que hi arriben de l'aplicació. En aquesta part de l'arquitectura, a diferència de la part de les aplicacions web tradicionals, s'emmagatzema la part gràfica de l'aplicació (esquema XML). Amb això evitem haver de renderitzar una nova pàgina web cada vegada que volem mostrar noves dades en la pantalla.
- En segon lloc, tenim el controlador de l'aplicació i la passarel·la (*gateway*). El controlador de l'aplicació seria la part de l'aplicació que s'encarrega d'interactuar amb el client i viceversa. D'altra banda, la passarel·la d'informació o *gateway* s'encarrega de transformar les dades de què disposa l'aplicació en un format de dades que pugui ser entès pel navegador del client (per exemple, XML).
- L'element següent seria el servidor en què s'allotja l'aplicació. Aquest servidor d'aplicacions compilarà l'aplicació web en cas necessari i permetrà la comunicació de la nostra aplicació amb altres aplicacions o *servlets*. Aquesta part de l'arquitectura també s'encarrega d'obtenir les dades necessàries per a l'aplicació, d'una base de dades o de fitxers.
- Finalment tenim l'emmagatzematge de les dades, que habitualment serà una base de dades en què s'emmagatzemaran les dades relacionades amb el servei que ofereix l'aplicació.

Aquestes dues maneres diferents de treballar comporten que hi hagi components diferents per a cada una, en funció també dels llenguatges de programació utilitzats (que també seran diferents).

1.1.7 Biblioteques de components per a thin i rich clients

A continuació es mostren alguns exemples de components que es poden trobar en el mercat. Hi ha moltes pàgines web que ofereixen components gratuïts per a determinats llenguatges de programació, com també grups de desenvolupadors o comunitats que unifiquen els esforços. Aquests poden anar destinats a programadors individuals o petites empreses que programen amb eines lliures.

Però també es poden trobar moltes altres pàgines que, pagant, ofereixen molts altres components, alguns per a entorns de desenvolupament que també són de pagament. Aquests estan destinats a empreses de desenvolupament de programari més grans que compren aquests components per a agilitzar el desenvolupament de les seves aplicacions.

Moltes altres empreses desenvolupen el propis controls que comparteixen entre els treballadors sense arribar-los a fer públics.

Silverlight

Solució de Microsoft semblant al Flash que permet l'execució d'aplicacions multimèdia en clients.

En la figura 1.12 es pot veure un exemple de components que s'ofereixen en una pàgina web. Se n'hi poden trobar de gratuïts i de pagament. En aquesta figura d'exemple, es poden veure components per a Silverlight, per a ASP.NET (Microsoft) o per a controls activeX, per exemple.

Un altre exemple d'entorn de treball, aquesta vegada del JavaScript, és jQuery, que simplifica la manera de treballar amb els documents HTML i agregar interaccions amb tecnologia AJAX. JQuery ofereix una sèrie de components basats en JavaScript.

Per a més informació sobre aquests components i d'altres, cal consultar la secció "Annexos" del web del mòdul.

1.1.8 Utilitat dels components en el desenvolupament d'aplicacions

El desenvolupament d'aplicacions amb interfícies gràfiques d'usuari és una tasca que necessita molt de temps i, en conseqüència, diners i, tot i així, moltes de les aplicacions desenvolupades acaben necessitant un manteniment que arregli els errors o millori la usabilitat. I tot i que al llarg dels darrers anys hi ha hagut una evolució considerable en el camp de les eines, mètodes i models de programació, la situació actual pel que fa a desenvolupament d'aplicacions amb interfícies gràfiques continua essent la que hem exposat.

Una alternativa que sembla que pot millorar aquesta situació i canviar-la cap a millor és la de modificar el model de desenvolupament actual i portar-lo cap a un model de desenvolupament orientat a components (de fet, hi ha molts desenvolupadors d'aplicacions que ja ho fan).

Aquest tipus de model consisteix a desenvolupar les aplicacions amb interfícies gràfiques d'usuari a partir de l'ús de components de programari, gratuïts o comercials, però que ja existeixen. Aquest nou model de desenvolupament cerca la limitació de la codificació de nou codi de programació al mínim indispensable.

Però serà necessària la creació d'un conjunt molt ampli de components, és a dir, l'existència d'un mercat de components de programari de qualitat i amb un àmbit d'aplicació general, perquè cada vegada més desenvolupadors utilitzin la programació orientada a components.

Els beneficis d'aquest tipus de programació són molts:

- Venda de components per empreses especialitzades (nou camp de negoci).
- Desenvolupament més senzill (quant a temps dedicat i, en conseqüència, quant a costos).
- Taxa menys elevada d'errors si els components que es fan servir passen per un control de qualitat exhaustiu abans de comercialitzar-los.

Els beneficis i alternatives que ofereix aquest model de programació són molts, però s'han explicat com a introducció als components i a les seves possibilitats.

FIGURA 1.12. Components One

.NET Windows Forms Controls		Studio for WPF Controls	
BarCode	List	Accordion	HyperPanel
Chart	Menus and Toolbars	Book	MaskedTextBox
DataExtender	PDF	Chart	MediaPlayer
DataObjects	Print Preview	ColorPicker	NumericBox
DockingTab	Reports	Cube	PropertyGrid
DynamicHelp	Ribbon	Date-Time Editors	RangeSlider
Editor	Scheduler	DockControl	Reports
Excel	Sizer	DropDown	Scheduler
Flash	SpellChecker	Expander	TabControl
FlexGrid	SuperTooltip	Gauges	Windows
Gauges	True DBGrid	Grid	
Input	Zip		
	InputPanel		
ASP.NET AJAX Controls		Silverlight Controls	
Accordion	ProgressBar	Accordion	Image Magnifier
Calendar	ReportViewer	Bitmap	Layout Panels
ComboBox	Scheduler	Book	Maps
Editor	Slider	Chart	Masked TextBox
Excel	Splitter	ColorPicker	MediaPlayer
Expander	SuperPanel	ComboBox	Menu
FormDecorator	TabControl	ContextMenu	NumericBox
Foxy	TabStrip	CoverFlow	PDF
Gauges	Toolbar	Cube	PropertyGrid
GridView	ToolTip	Data	RangeSlider
HeaderContent	TreeView	DataGrid	Reflector
Input	Upload	Date-Time Editors	RichTextBox
Menu	WebChart	DockControl	Scheduler
MultiPage	Window	DragDropManager	SpellChecker
NavPanel	Zip	DropDown	TabControl
PDF		Expander	Toolbar
		FilePicker	TreeView
		Gauges	Uploader
		HtmlHost	Windows
		HyperPanel	Zip
		Image	
ASP.NET iPhone Controls		Studio for ActiveX Controls	
Buttons	NavigationList	Chart	True DBList
Calendar	PickerView	Query	VSFlexGrid
CoverFlow	Slider	SizerOne	VSSPELL
Dialog	TabBarController	True DataControl	VSVIEW Classic
LaunchPad	ViewPort	True DBGrid	VSVIEW Reporting
MultiView	ViewScroller	True DBInput	WebChart
Studio for Mobile Controls			
Chart	MaskedTextBox		
FlexGrid	Zip		

1.2 Creació d'un component visual

Per què s'han d'utilitzar components visuals?

Si es pot desenvolupar una interfície gràfica d'usuari sense escriure cap línia de codi, es justifica de manera molt convincent la utilització de components visuals, sempre que hi hagi els que desenvoluparan les necessitats del programador.

Per què s'han de desenvolupar nous components visuals personalitzats?

Un component visual, una vegada empaquetat, es podrà fer servir en projectes futurs, es podrà compartir amb altres desenvolupadors del mateix projecte que necessitin eines semblants o es podrà compartir de manera altruista amb la resta de desenvolupadors d'arreu del món. Si no es desenvolupa fent servir components o desenvolupant aquest i es programen les aplicacions de manera directa, molt probablement, s'estarà repartint la mateixa feina, o més enllà en el temps per part del mateix desenvolupador, o per part d'altres programadors, la qual cosa incrementarà temps i costos dels projectes.

Cal desenvolupar components visuals encara que només siguin per a una aplicació?

La resposta a aquesta pregunta és un *sí*, seguint la mateixa línia argumental que en la qüestió anterior. Si en el futur es vol modificar l'aplicació desenvolupada, i els canvis afecten la part de codi empaquetada en un component, només caldrà distribuir aquesta biblioteca i no tota l'aplicació. En el cas d'una aplicació client-servidor, si volem modificar un component o hi ha una actualització, potser només caldrà actualitzar el component en el servidor, sense tocar la resta de programaris instal·lats en els clients, que s'actualitzaran automàticament.

1.2.1 Eines per a dissenyar i crear components visuals

Pràcticament la majoria dels entorns de desenvolupament integrats permeten el desenvolupament de nous components visuals, connectors, extensions, *plug-ins* o complements. És a dir, parts de codi que quedaran empaquetades per a la seva utilització posterior des del mateix programari (normalment), i es podran utilitzar per al mateix llenguatge de programació per al qual s'hagin desenvolupat els components.

Una vegada creat un component visual, caldrà empaquetar-lo. Aquesta acció genera un arxiu de tipus dll o msi que es podrà importar al mateix entorn de desenvolupament a altres màquines o, fins i tot, a altres entorns de desenvolupament.

La majoria d'IDE permeten crear components visuals i, alguns menys, empaquetar-los. A continuació, s'enumeren alguns d'aquests IDE:

- Visual Studio .NET, per a sistemes operatius Windows, suporta llenguatges com Visual C++, Visual C#, ASP.NET, i Visual Basic .NET. Permet crear components i empaquetar-los en arxius msi o dlls.
- MonoDevelop, una lliure de codi obert multiplataforma que permet el desenvolupament d'aplicacions amb C# i amb altres llenguatges de programació .NET. MonoDevelop permet crear nous components visuals.
- Eclipse, IDE de codi obert multiplataforma, utilitzada per a crear JDT (*Java development toolkit*) sota el llenguatge Java, encara que també permet el

Add-ons (complements):
millors instal·lables en els programes.

Plug-ins (endollables):
complements per a una aplicació informàtica.

En la secció Activitats del web del mòdul podreu trobar alguns exemples de creació de components i més informació sobre les eines.

treball amb altres llenguatges de programació com C++, PHP o Perl. Permet crear i encapsular nous components visuals.

- NetBeans, eina de desenvolupament d'aplicacions per a desenvolupadors en Java. Permet crear mòduls que es poden agregar a l'aplicació. Permet crear components i empaquetar-los per a NetBeans.
- GLADE, és una eina per a dissenyar interfícies gràfiques d'usuari de manera ràpida i senzilla, i generar un codi XML. No permet la creació de visual de components, però en ser una eina lliure de codi obert, sovint apareixen noves versions i complements.

1.2.2 Definició de les propietats i mètodes

Un objectiu important de molts llenguatges de programació orientats a objectes és permetre encapsular els detalls interns en una classe. Aquests detalls són els que definiran les característiques de l'objecte, la seva aparença visual i el seu comportament.

Una propietat és una barreja entre el concepte de camp d'una base de dades i el concepte de mètode d'una classe. Externament s'hi accedeix com si es tractés d'un camp normal, però internament és possible associar codi a executar en cada assignació o lectura del seu valor.

Aquest codi de programació es pot utilitzar per a comprovar que no s'assignen valors erronis o per a calcular-ne automàticament el valor just en demanar la seva lectura entre moltes oportunitats.

Les propietats són els elements del component visual que en configuren l'aspecte i en controlen el comportament.

Una propietat no emmagatzema dades, però s'utilitza com si les emmagatzemés. A la pràctica, el que se sol fer és escriure com a codi de programació a executar, quan s'hi assigni un valor. Aquest codi controlarà que aquest valor sigui correcte i que s'emmagatzemi en un camp privat si ho és. Amb les funcions *get* i *set* de la programació orientada a objectes es podrà accedir al valor o modificar el seu valor.

Així se simula que té un camp públic sense els inconvenients que aquests presenten perquè no hi poden controlar l'accés.

Vegeu tot seguit la sintaxi de la definició d'una propietat en Visual Basic:

```

1  property <NomPropietat> as <TipusPropietat>
2  {
3      get
4          <CodiLectura>
5      end get

```

```

6   set
7     <CodiEscriptura>
8   end set
9 }
```

Vegeu la sintaxi de la definició d'una propietat en C#:

```

1 <TipusPropietat> <NomPropietat>
2 {
3   get
4   {
5     <CodiLectura>
6   }
7   set
8   {
9     <CodiEscriptura>
10  }
11 }
```

Una propietat pot ser de només lectura, de només escriptura o de lectura i escriptura. Els descriptors d'accés *Get* i *Set* proporcionen accés de lectura i d'escriptura respectivament. Es pot afegir codi a aquests descriptors d'accés per a proporcionar la funcionalitat adequada per a obtenir o configurar la propietat.

Els mètodes són funcions o procediments associats al component que poden cridar perquè aquest faci diferents accions de les quals obtingui una resposta.

Els procediments i les funcions són molt similars. La diferència és que els procediments no retornen cap valor, i les funcions sempre retornen un únic valor.

Per exemple, un mètode d'un *radio button* permet ordenar alfabèticament els seus components o bé un altre mètode permet fer una cerca d'una dada concreta.

Un exemple en codi Visual Basic és:

Mètode:

```

1 Sub Nom_del_Mètode (Paràmetres)
2   Línies de codi
3 End sub
```

Funcions:

```

1 Function Nom_de_la_Funció (Paràmetres) es Valor_a_retornar
2   Línies de codi
3   Nom_de_la_Funció = valor
4 End function
```

Molts components visuals tenen propietats en comú. Per exemple, tots els components visuals tenen les propietats *top* i *left*, que controlen la posició del component en el formulari, tant en temps de disseny com en temps d'execució.

Però, de vegades, ens pot interessar que algunes de les propietats que es defineixen en els components siguin visibles en el quadre de propietats de l'eina que estiguem utilitzant per a desenvolupar el programari, amb l'objectiu que el seu valor es pugui especificar en temps de disseny.

Per tal de demostrar el funcionament de les propietats i els mètodes es farà un petit exemple en la secció Annexos del web del mòdul, en què es definirà la classe cotxe amb les propietats model, marca i preu.

A continuació es mostra un petit exemple que vol explicar com es defineix una propietat visible en temps de disseny.

Un component visual de tipus TextBoxNum mostrarà un nombre. Es vol que el color d'aquest sigui diferent en funció del seu valor, un color si és positiu i un color si és negatiu. Per això caldrà definir dues propietats que permetin indicar a l'usuari els colors que es volen visualitzar.

En visual basic .NET, una propietat es defineix de la manera següent:

```

1  <Category("Behavior"), DefaultValue(GetType(Color), "Black"), _
2      Description("Indica el color del nombre en ser positiu")> _
3  Public Property ColorValorPositiu() As Color
4      Get
5          Return _ColorValorPositiu
6      End Get
7      Set(ByVal Value As Color)
8          _ColorValorPositiu = Value
9      End Set
10     End Property
11
12     <Category("Behavior"), DefaultValue(GetType(Color), "Red"), _
13         Description("Indica el color del nombre en ser negatiu")> _
14     Public Property ColorValorNegatiu() As Color
15         Get
16             Return _ColorValorNegatiu
17         End Get
18         Set(ByVal Value As Color)
19             _ColorValorNegatiu = Value
20         End Set
21     End Property

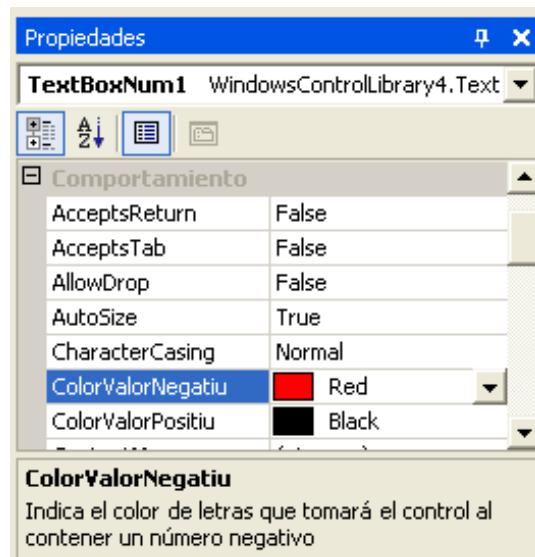
```

La definició de la propietat és igual que la definida més amunt però incorporant un conjunt de metadades.

- **Category.** Especifica la manera de classificar les propietats en un examinador de propietats del dissenyador visual. En l'exemple s'indica que les noves propietats ColorValorPositiu, ColorValorNegatiu es mostraran en la categoria *behavior* (comportament).
- **DefaultValue.** Especifica un valor predeterminat per a una propietat. En l'exemple, el valor per defecte de la propietat ColorValorPositiu serà el negre, mentre que el valor per defecte de la propietat ColorValorNegatiu serà el vermell.
- **Description.** Especifica una descripció breu de la propietat. En un dissenyador visual com Visual Studio, l'examinador de propietats mostra la descripció de la propietat seleccionada, generalment en la part inferior de la finestra.

En la figura 1.13 és poden veure les dues noves propietats afegides al component de tipus TextBoxNum en el Visual Studio 2010.

FIGURA 1.13. Exemple de propietats personalitzades



Al llarg d'aquest apartat s'està fent referència a dos conceptes que, de vegades, es tendeix a confondre. De vegades no queda clar el concepte de propietat i el d'atribut. I són conceptes i termes diferents:

- **Els atributs:**

- Mai no són públics.
 - Són accessibles únicament a nivell de programació.
 - Descriuen l'estat intern i el comportament d'un component visual en temps d'execució.

- **Les propietats:**

- Són característiques públiques dels components.
 - Són accessibles a nivell de programació i a nivell de disseny de les interfícies gràfiques.
 - Descriuen l'estat intern i el comportament d'un component visual en temps de disseny.

En la secció Annexos del web del mòdul podreu trobar alguns exemples més de propietats i mètodes.

Quan s'implementa un component mitjançant una classe d'objectes, cada propietat sol estar associada internament a un atribut, encara que això no passi sempre.

En la creació del component, la definició de l'atribut seria:

1.2.3 Esdeveniments als quals ha de respondre un component

Un esdeveniment és una acció determinada que es durà a terme vinculada amb un component i que activarà una certa funcionalitat. Un exemple és fer un clic a sobre d'un component.

En el Visual Basic, per a definir un esdeveniment només cal escriure la instrucció *event* seguida de la definició que donarem al mètode que rebrà els esdeveniments. Per exemple, si la nostra classe és del tipus *button*, podem definir l'esdeveniment *Click* de la manera següent:

```
1 Public Event Click (ByVal sender As Object, ByVal e As EventArgs)
```

Vegem ara com hauria de definir aquest mateix esdeveniment un programador de C#:

```
1 public delegate void ClickEventHandler (object sender, EventArgs e);
2 public event ClickEventHandler Click;
```

En aquest cas, primer es defineix un delegat i a continuació cal definir l'esdeveniment que ha de ser del tipus d'aquest delegat.

En el Visual Basic:

```
1 RaiseEvent Click (sender, e)
```

En C #:

```
1 if (Click! = null)
2 (
3     Click (sender, e);
4 )
```

En el Visual Basic serà molt més fàcil i simple, i no cal fer cap comprovació.

Un delegat és una referència a una funció, la qual cosa també es coneix com *un punter a una funció*, és a dir, un delegat permet accedir a una funció de manera gairebé anònima, ja que simplement té l'adreça de memòria d'aquesta funció. I, sabent l'adreça de memòria, hi podem accedir.

Però en .NET això ha d'estar controlat, de manera que aquest accés no sigui arbitrari.

Per tant, si es vol accedir a un mètode, s'ha de fer per mitjà d'un punter controlat, i la manera de controlar aquest accés és definint un prototip en el qual s'indiqui de quin tipus és aquest mètode, si rep paràmetres i, si en rep, quants i de quin tipus són.

Aquesta definició del prototip de funció (o mètode) al qual es vol accedir es fa per mitjà d'un delegat.

Un esdeveniment és un missatge (o notificació) que lanza un objecte d'una classe determinada quan alguna cosa ha passat.

L'exemple més habitual és quan l'usuari fa clic amb un dispositiu extern (normalment un ratolí) en un botó d'un formulari. Aquesta acció produeix, entre altres, l'esdeveniment *Click* del botó que s'ha polsat. D'aquesta manera, el botó notifica que aquesta acció ha passat i queda a criteri del programador activar una funcionalitat vinculada a aquest esdeveniment.

En la secció Annexos del web del mòdul podreu trobar alguns exemples més d'esdeveniments vinculats a components.

Si es vol interceptar aquest esdeveniment s'ha de comunicar a la classe que defineix el botó indicant la funció a executar. En cas contrari, no cal indicar res i, deixant en blanc la funcionalitat, s'indicarà que no cal executar cap acció després de l'esdeveniment.

Els esdeveniments es poden definir en qualsevol classe, i no solament en classes que mantinguin algun tipus d'interacció amb els usuaris. Això sí, el més habitual és que es facin servir els esdeveniments amb classes que formen part de la interfície gràfica que interactua amb l'usuari d'una aplicació.

Per exemple, si es vol desenvolupar un esdeveniment en una classe per a fer una notificació cada vegada que es modifica el contingut d'una propietat, s'haurà de definir un delegat que indiqui la signatura que ha de tenir el mètode que hagi de rebre aquesta notificació.

A continuació, caldrà definir l'esdeveniment pròpiament dit, el qual serà una variable especial del tipus del delegat.

Això de variable especial és perquè en realitat es defineix com qualsevol altra variable (només que amb un tipus delegat com a tipus de dades), però afegint-hi la instrucció *event*.

```
1 public delegate void NombreCambiadoEventHandler (string nou, string  
2 anterior);  
3 public event NombreCambiadoEventHandler NombreCambiado;
```

1.2.4 Proves unitàries i documentació

Una vegada creat un component visual en un entorn de desenvolupament integrat cal provar-lo i verificar-lo per a confirmar que desenvolupa les funcionalitats tal com es van plantejar abans del seu desenvolupament.

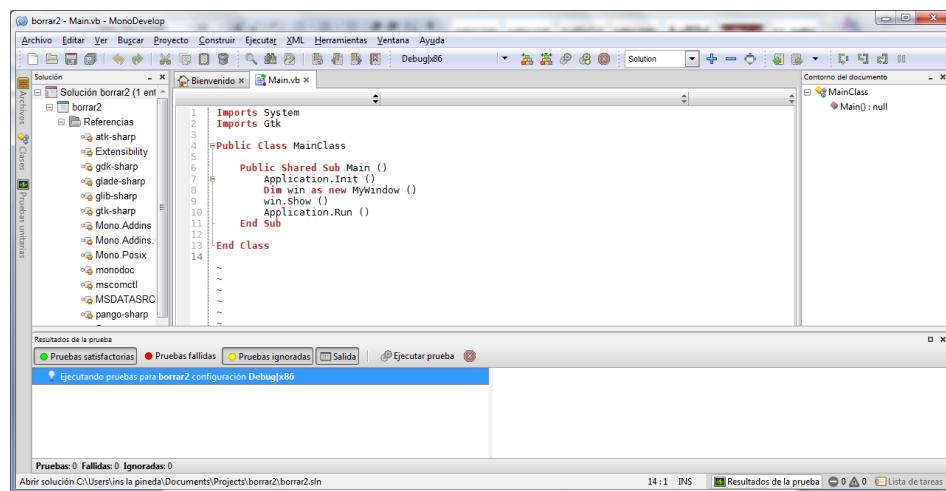
Les proves unitàries també es coneixen com a *proves de components*. Són proves sobre parts del codi de programació de les aplicacions informàtiques amb l'objectiu de validar-ne el funcionament correcte.

Aquestes proves dependran molt de la tecnologia utilitzada per a crear el component visual i es duran a terme en el mateix entorn de desenvolupament. En el cas dels components visuals, aquestes proves seran moltes vegades d'execució i validació visual de les funcionalitats cercades en el seu desenvolupament.

En cas d'aplicar més d'una prova per a un mateix component, aquestes hauran de ser independents les unes de les altres, perquè, si n'hi ha, els errors d'una prova no corregeixin o facin desaparèixer els errors de les altres.

En la figura 1.14 es veu la funcionalitat que ofereix MonoDevelop per a dur a terme les proves unitàries. Algunes IDE afegeixen aquest tipus de funcionalitat, la qual facilita la seva tasca als desenvolupadors.

FIGURA 1.14. MonoDevelop - proves unitàries



Per a aconseguir la independència entre les proves es poden fer servir alguns substituts com els estubs. Aquests ajuden a simular les proves de parts dels components de manera aïllada simulant el funcionament d'altres. Per exemple, si un component s'encarrega d'agafar determinades dades d'una base de dades i fent la mitjana, mostrar-les fent servir un sistema de barres, un estub serà un tros de codi que substituirà l'accés a la base de dades i simularà el lliurament de les dades necessàries.

La documentació associada a un component serà molt semblant a la documentació que es pot crear en el desenvolupament d'una aplicació.

Hi ha molts tipus de documentació per a completar el desenvolupament d'una aplicació informàtica, des de la documentació del codi programat, passant per la documentació tècnica fins a la documentació d'usuari que indica com s'explota l'aplicació desenvolupada.

Al igual que amb les proves d'usuari, determinats IDE ofereixen la possibilitat d'automatitzar la generació de documentació, tant els comentaris en el codi de programació com la generació de la documentació tècnica o d'usuari final.

En el cas d'una aplicació informàtica es pot arribar a trobar documentació referent al següent:

- l'anàlisi i els requisits de l'aplicació
- el disseny i seva arquitectura
- els temes tècnics
- els usuaris
- el màrqueting

En el cas dels components serà important generar la documentació relativa a la creació tècnica del component per a facilitar a futurs desenvolupadors la comprensió i utilització del component. També serà necessària una documentació més d'usuari per a explicar la instal·lació del component i la seva utilització correcta.

1.3 Persistència i serialització

El significat genèric del terme *persistència* és ‘transcendència en el temps i/o en l’espai’.

En el cas de l’execució d’una aplicació informàtica, es parlaria de guardar l’estat de les dades o les variables una vegada finalitzada l’execució del programari.

¿Quin sentit té això si estic executant una aplicació en un entorn local? Té molt de sentit. És el que passa normalment amb les aplicacions de gestió d’informació desenvolupades per a treballar en un únic ordinador. En finalitzar l’execució es desen les dades i això permet continuar en l’execució següent amb la mateixa situació com havia finalitzat.

I en aplicacions en un entorn client-servidor o un entorn web? Si, en accedir a les dades que hi ha en un servidor i mostrar-les per un navegador o un programari client, volem fer un canvi o veure una petita modificació en les dades, no hauria de ser necessari recarregar tota la interfície sencera, amb les dades i els detalls d’aquesta.

1.3.1 Persistència dels components visuals

És força atractiu utilitzar aquesta tecnologia per a emmagatzemar l'estat d'un programa de manera que es pugui restaurar posteriorment, la qual cosa pot tenir moltes aplicacions. Per posar algun exemple, es podria desar un component actiu

i carregar-lo més endavant, desplaçar d'una localització física a una altra portada en un dispositiu de memòria secundària o, simplement, transmetre per una xarxa, etc.

La persistència dels components visuals es defineix com la capacitat d'emmagatzemar els canvis de les propietats d'un component, és a dir, permet que un objecte existeixi més enllà de l'execució d'un programa.

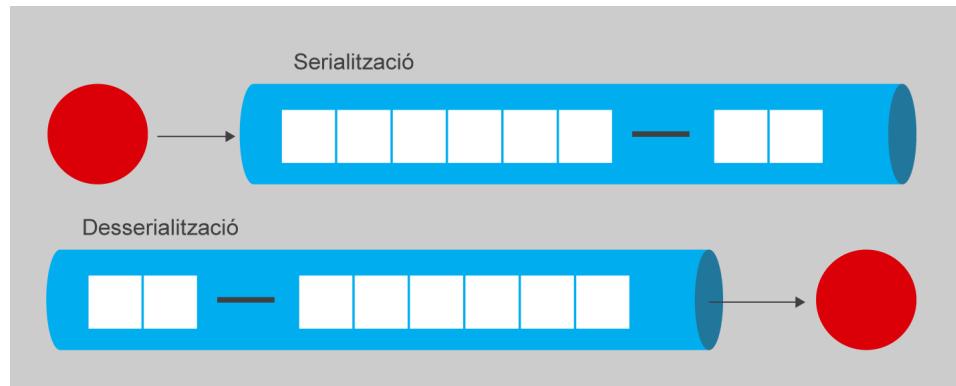
Per a persistir s'ha de serialitzar la informació, i una vegada serialitzada es podrà enviar per xarxa, mantenir en memòria, enviar-la a una impressora, etc.

1.3.2 Serialització

La serialització és la clau per a implementar la persistència. Representa el procés d'emmagatzematge de l'estat d'un objecte en un mitjà. Per a això, es recopila tota la informació necessària de l'objecte, per a convertir-se en una seqüència o flux de dades que, més tard, en permeti la recuperació.

En la figura 1.15 es pot veure una representació de com funciona la serialització i la deserialització.

FIGURA 1.15. Serialització



Podem diferenciar diferents tipus de serialització, per exemple:

- Binària
- XML (*extensible markup language*)

1.3.3 La serialització binària

La serialització binària utilitza la codificació binària per a generar una serialització compacta per a usos com podria ser l'emmagatzemament o les seqüències de xarxa

basades en *sockets*.

La serialització binària es caracteritza per convertir en un flux de bytes tant els membres públics i com privats juntament amb el nom de la classe, incloent-hi el codi assemblador, i després s'emmagatzema en un flux de dades. Quan l'objecte és deserialitzat, s'obté un clon de l'objecte original.

Exemple de serialització binària

Per a entendre el concepte de la persistència i la serialització es mostra un petit exemple codificat en el llenguatge de programació C#. L'objectiu de l'exemple és demostrar com es poden congelar i descongelar les propietats d'un objecte.

```

1 [Serializable]
2     public class Cotxe
3     {
4
5         public string marca;
6         public string model;
7         private float preu;
8
9         public float getPreu()
10        {
11            return preu;
12        }
13
14         public void setPreu(float preu1)
15        {
16            preu = preu1;
17        }
18
19         public override string ToString()
20        {
21            return String.Format("Marca:{0}\nModel:{1}\nPreu:{2}\n",
22 marca, model, preu);
23        }
24    }
25 }
```

```

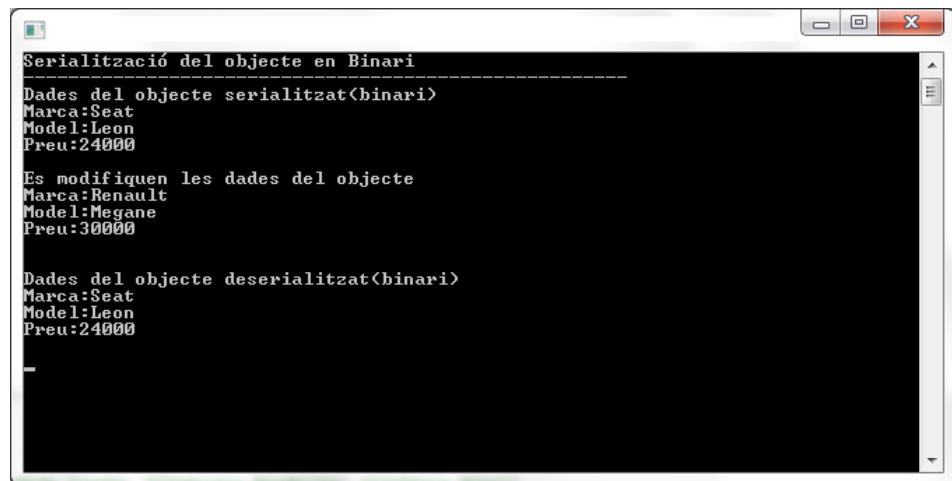
1 using System;
2 using System.IO;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Text;
6 using System.Xml.Serialization;
7 using System.Runtime.Serialization;
8 using System.Runtime.Serialization.Formatters.Binary;
9
10 namespace ConsoleApplication1
11 {
12     class Program
13     {
14         static void Main(string[] args)
15         {
16             //Definició d'un objecte de tipus cotxe
17             Cotxe cotxe1 = new Cotxe();
18             cotxe1.marca = "Seat";
19             cotxe1.model = "Leon";
20             cotxe1.setPreu(24000);
21
22             Console.WriteLine("Serialització de l'objecte en Binari");
23             Console.WriteLine("____");
24             IFormatter formatter = new BinaryFormatter();
25
26             //Serialització de l'objecte
```

```

28         using (FileStream fs = new FileStream(@".\BINCotxe.bin", FileMode.
29             Create,
30             FileAccess.ReadWrite, FileShare.None))
31             {
32                 formatter.Serialize(fs, cotxe1);
33                 fs.Close();
34                 Console.WriteLine("Dades de l'objecte serialitzat(binari)");
35                 Console.WriteLine(cotxe1.ToString());
36             }
37
38             //Modificació de l'objecte
39             cotxe1.marca = "Renault";
40             cotxe1.model = "Megane";
41             cotxe1.setPreu(30000);
42             Console.WriteLine("Es modifiquen les dades de l'objecte");
43             Console.WriteLine(cotxe1.ToString());
44
45             //Deserialització de l'objecte
46             using (FileStream fs = new FileStream(@".\BINCotxe.bin", FileMode.
47                 Open,
48                 FileAccess.Read, FileShare.None))
49                 {
50                     Console.WriteLine("");
51                     cotxe1 = (Cotxe)formatter.Deserialize(fs);
52                     fs.Close();
53                     Console.WriteLine("Dades de l'objecte deserialitzat(binari)");
54                     Console.WriteLine(cotxe1.ToString());
55                 }
56
57             }
58 }
```

En la figura 1.16 es poden veure els resultats d'executar aquestes línies de codi que serialitzen un objecte en binari.

FIGURA 1.16. Serialització d'un objecte binari



1.3.4 La serialització en XML

XML

Llenguatge de marques extensible per a la representació de dades.

Tal com es demostra en l'exemple següent, la serialització XML serialitza les propietats i camps públics d'un objecte. Aquesta és una de les diferències amb la serialització binària que, tal com s'ha vist, guarda tant les propietats públiques

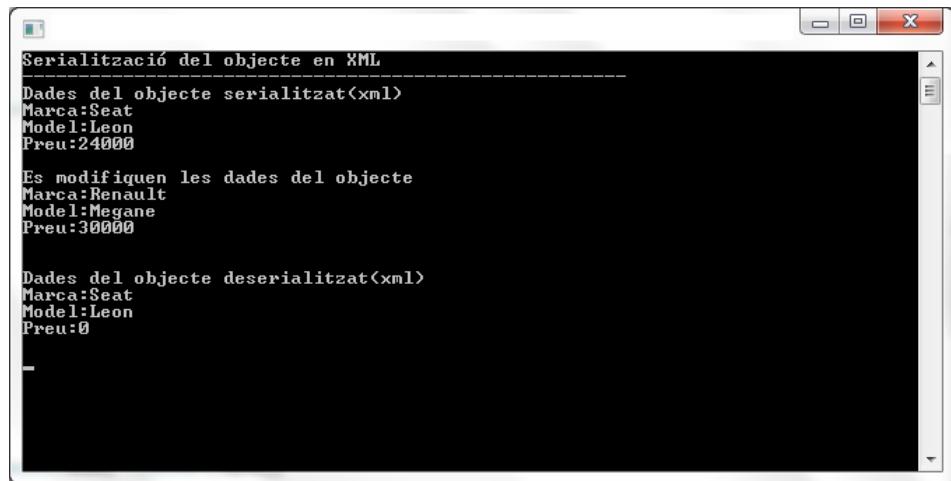
com les privades.

Continuant amb l'exemple anterior, tot seguit es mostra el resultat de serialitzar i deserialitzar utilitzant l'XML.

```
1 static void Main(string[] args)
2 {
3     //Definició d'un objecte de tipus cotxe
4     Cotxe cotxe1 = new Cotxe();
5     cotxe1.marca = "Seat";
6     cotxe1.model = "Leon";
7     cotxe1.setPreu(24000);
8
9     Console.WriteLine("Serialització de l'objecte en XML");
10    Console.WriteLine("____");
11    _____");
12    XmlSerializer xml = new XmlSerializer(cotxe1.GetType());
13
14    //Serialització de l'objecte
15    using (FileStream fs = new FileStream(@"..\XMLCotxe.xml",
16 FileMode.Create, FileAccess.ReadWrite, FileShare.None))
17    {
18        xml.Serialize(fs, cotxe1);
19        fs.Close();
20        Console.WriteLine("Dades de l'objecte serialitzat(xml)");
21        Console.WriteLine(cotxe1.ToString());
22    }
23
24    //Modificació de l'objecte
25    cotxe1.marca = "Renault";
26    cotxe1.model = "Megane";
27    cotxe1.setPreu(30000);
28    Console.WriteLine("Es modifiquen les dades de l'objecte");
29    Console.WriteLine(cotxe1.ToString());
30
31    //Deserialització de l'objecte
32    using (FileStream fs = new FileStream(@"..\XMLCotxe.xml", FileMode.
33        Open,
34        FileAccess.Read, FileShare.None))
35    {
36        Console.WriteLine("");
37        cotxe1 = (Cotxe)xml.Deserialize(fs);
38        fs.Close();
39        Console.WriteLine("Dades de l'objecte deserialitzat(xml)");
40        Console.WriteLine(cotxe1.ToString());
41    }
42
43    Console.ReadLine();
44 }
```

En la figura 1.17 es poden veure el resultats d'executar aquestes línies de codi que serialitzen un objecte en XML.

Podem veure com les dades privades no s'han desat.

FIGURA 1.17. Serialització de l'objecte en XML

1.4 Empaquetat de components

Ja fa un quant temps, quan un desenvolupador volia crear un component visual per a compartir-lo amb els companys o fer-lo servir al cap d'un quant temps, tenia poques alternatives, i la majoria d'aquestes passaven per fer una còpia literal del codi de programació que havia desenvolupat per a crear el component o l'aplicació.

No cal explicar la complicació i dificultat d'aquest procediment que provocava molts errors i equivocacions per la singularitat dels projectes no orientats a objectes. Calia instal·lar altres programes o altres biblioteques perquè aquell codi pogués ser acceptat en un altre entorn i per un altre codi. Era tot un esdeveniment aconseguir que un component creat per un programador funcionés correctament en una màquina diferent (amb el seu programari propi i la seva configuració) essent utilitzat per un altre desenvolupador.

L'empaquetat de components va obrir moltes vies, va facilitar en gran mesura el procés de compartició de components, va facilitar el procés, la distribució i el fet de fer-lo el més transparent possible per als usuaris finals. A més, el procés d'empaquetar un component és un procés automàtic que facilitarà les actualitzacions i manteniments de les aplicacions.

Però el procés d'empaquetat varia en funció del llenguatge i l'entorn de desenvolupament que s'utilitza.

Abans del procés de creació i d'empaquetatge d'un component, caldrà tenir en compte alguns elements com poden ser:

- Empaquetat de fonts, objectes i executables.
- Arquitectura de destinació del paquet.
- Informació relativa al paquet (tècnica, d'usuari, etc.)

- Distribució en la destinació del paquet.
- Signatura digital.

1.4.1 Empaquetat d'un thin client

Per a explicar com s'utilitzen els components creats en una plataforma de tipus client lleuger (*thin client*) es farà servir un exemple implementat en jQuery.

jQuery és una biblioteca de components i funcionalitats implementada en JavaScript, pensada per a interactuar amb els elements d'un web mitjançant el DOM. El que la fa tan especial és la senzillesa i la mida reduïda.

JQuery disposa una pàgina web amb exemples de components desenvolupats que són d'utilització lliure i de codi obert. Aprofitant un d'aquests components es farà un seguiment d'aquest i del seu empaquetatge.

Concretament, s'ha seleccionat un exemple de component que permet ordenar els elements d'una llista arrossegant-los amb el ratolí. Es pot veure un exemple del funcionament d'aquest component en la figura 1.18.

Exemple d'empaquetat en una plataforma thin client mitjançant jQuery

FIGURA 1.18. Exemple d'empaquetat d'un thin component



Per a poder utilitzar la biblioteca jQuery haurem fer referència als fitxers de JavaScript que disposen del codi font dels components.

```

1 <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4/jquery.min.js">
2 </script>
3   <script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8/jquery-ui.min.
4   js">
</script>
```

El codi de la pàgina web seria:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8/themes/base/
5 jquery-ui.css" rel="stylesheet" type="text/css"/>
6   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4/jquery.min.js">
```

DOM

Document Object Model, una interfeïcі de programació d'aplicacions per a representar documents HTML i XML.

```

7   </script>
8     <script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8/jquery-ui.min.
9       js">
</script>
10
11   <script>
12     $(document).ready(function() {
13       $("#sortable").sortable();
14     });
15   </script>
16 </head>
17 <body style="font-size:62.5%;">
18
19 <ul id="sortable">
20 <li>Item 1</li>
21 <li>Item 2</li>
22 <li>Item 3</li>
23 <li>Item 4</li>
24 <li>Item 5</li>
25 </ul>
26
27 </body>
28 </html>

```

D'aquesta manera veiem que, en una plataforma de client lleuger, l'empaquetat d'un component consistiria a fer referència al fitxer JavaScript que el conté.

1.4.2 Empaquetat d'un rich client

En un *rich client* el procés d'empaquetat d'un component no és tan laboriós com era o semblava inicialment. Caldrà tenir en compte certs detalls o paràmetres de configuració en el moment de crear els paquets que dependran de la distribució a la qual estarà destinada el component a empaquetar.

Quan es crea un paquet d'un component, s'ha de considerar cert tipus d'informació que podrà ser comuna a diferents components, com el codi font, la seva arquitectura, la seva descripció, les dependències o els requisits. Tota aquesta informació queda recollida en uns arxius amb les especificacions i els controls. Són arxius que contenen la informació principal del component empaquetat, són la seva font de descripció i especificació per a futures regeneracions del component en sí.

Empaquetar un component és una de les maneres millors i més recomanades de distribuir els components a usuaris de cert entorn, o posar-los a disposició en un mirall perquè es pugui usar en el desenvolupament d'aplicacions.

Una vegada creat un component en una plataforma *rich client*, per a poder-la distribuir caldrà crear un instal·lador o una biblioteca perquè aquest component pugui ser important o pugui ser instal·lat en altres entorns o altres màquines. Les eines que permeten aquesta funció acostumen a incorporar funcionalitats que ajuden a empaquetar el component, amb *wizards* o auxiliars que guien els desenvolupadors en l'empaquetat.

Hi ha diferents tipus d'empaquetat de components en funció de les eines de desenvolupament integrades que es faran servir. Dos dels tipus més habituals són el de creació d'un paquet d'instal·lació i el de creació de dlls:

- Paquet d'instal·lació estàndard. És un tipus d'empaquetat que crearà un arxiu .exe o un arxiu setup.exe o un arxiu .msi que permetrà distribuir el component en altres entorns que facin servir la mateixa IDE per al desenvolupament d'aplicacions. Només caldrà executar l'arxiu autoinstal·lable per a aconseguir que el nou component formi part de les possibilitats del programador. Aquest arxiu es podrà distribuir en un CD, un llapis de memòria o penjant-lo en un servidor web perquè el descarreguin els usuaris o els desenvolupadors.
- Arxiu o biblioteca. Es generarà un tipus d'arxiu dll o biblioteca dependent del llenguatge de programació i l'entorn de desenvolupament. Aquest arxiu també serà necessari en temps d'execució per als usuaris finals i per als usuaris desenvolupadors per a crear les interfícies. Els arxius de dependències d'una aplicació emmagatzemen la informació referent als components (dlls o altres tipus de fitxers) que són necessaris per a executar una determinada aplicació.

En executar la creació del paquet d'instal·lació del component s'haurà d'escol·lir en quina ubicació es crearan els arxius vinculats a l'empaquetat. Es podrà configurar la selecció dels arxius que es deixarà que es creïn, decidir si s'afegeixen automàticament al component arxius d'ajuda o del tipus leame.txt.

Sempre és recomanable comprovar que el component s'ha creat correctament executant el paquet creat i verificant així que funciona.

En la secció Activitats del web del mòdul podreu trobar alguns exemples de creació de component a partir de plataformes *rich client*.

2. Usabilitat

Les interfícies gràfiques d'usuari són les responsables de la interacció que hi haurà entre els homes i les màquines. Una part important és la confecció d'aquestes interfícies i les eines que es faran servir, com, per exemple, els seus elements com els components o els controls. Cal també tenir en compte alguns aspectes concrets d'aquesta interacció quan es fa el disseny de les interfícies.

2.1 Introducció a la usabilitat

Hi ha moltes interfícies gràfiques d'usuari en molts àmbits de la vida quotidiana, i les persones són usuaris habituals de moltes interfícies al llarg del dia. Hi haurà algunes interfícies d'ús obligat, sense opció a escollir si ens agraden o no, o si se'n podrien fer servir d'altres. Per exemple, amb el quadre d'interacció d'una ràdio d'un cotxe, es tindrà poc marge de maniobra. Si agrada, bé. Si és poc amigable, o difícil d'entendre, ens hi haurem d'adaptar, o bé canviar d'aparell de ràdio al cotxe. Passarà una cosa semblant amb els caixers automàtics de les entitats bancàries. Si volem treure diners, ens hi haurem d'adaptar.

En canvi, amb altres interfícies sí que hi ha alternatives. Si volem cercar una informació o una notícia a Internet, accedirem a un portal, i si la seva interfície no és amigable o usable o comprensible, potser intentarem cercar la mateixa informació per altres mitjans o en altres portals d'Internet. Una pàgina web amb una bona usabilitat tindrà molt més èxit que una altra, amb continguts o informacions similars, però una usabilitat no tan bona.

Justament això serà el que ens indicarà l'èxit que tindrà un programari concret un cop estigui en funcionament. No solament haurem de crear programari amb un funcionament correcte (que faci el que els requeriments indicaven que havia de fer) i amb una fiabilitat bona (els resultats que ens ofereix el programari no poden ser dubtosos), sinó que la usabilitat del programari haurà de ser satisfactòria per als usuaris.

"La usabilitat serveix per a adaptar el programari als estils de treball reals dels usuaris, en comptes de forçar els usuaris a adaptar els seus estils de treball al programari."

Per a aconseguir això caldrà tenir en compte aquest aspecte a l'hora de desenvolupar programari de tot tipus, establint d'aquesta manera en la fase d'anàlisi els requeriments d'usabilitat, uns objectius que s'hauran d'assolir en finalitzar el desenvolupament de l'aplicació. Aquests requeriments d'usabilitat guiaran els dissenys de les interfícies que interactuaran amb els usuaris i, a més, els podrem utilitzar per a avaluar la qualitat del programari desenvolupat.

Com a desenvolupadors d'aplicacions es tracta d'un punt a tenir molt en compte, ja que la usabilitat de les aplicacions que es presentaran als futurs usuaris serà el primer que veuran en els prototipus de les aplicacions o de les interfícies i serà

el primer que avaluaran. També cal indicar què serà el que podrà diferenciar una aplicació d'una altra, ja que totes les aplicacions compliran les regles de negoci.

2.1.1 Què és la usabilitat?

En primer lloc, caldrà establir una definició del terme *usabilitat*. Aquesta paraula deriva del terme anglès *usability*, és a dir, indica si una cosa és usable o té un ús senzill.

La usabilitat defineix el grau de facilitat amb què un usuari pot fer servir una eina concreta o qualsevol altre objecte.

Però hi ha moltes maneres d'entendre aquest concepte, ja que s'han de tenir en compte molts matisos per a poder definir o entendre el significat de *facilitat d'ús*. A més, també caldrà parlar d'altres conceptes per a definir la usabilitat, termes com *efectivitat, eficiència i satisfacció de l'usuari*. I tots aquests conceptes variaran l'apreciació en funció de si estan afectats pels usuaris, els seus objectius o les seves expectatives i la situació d'ús de l'eina o aplicació.

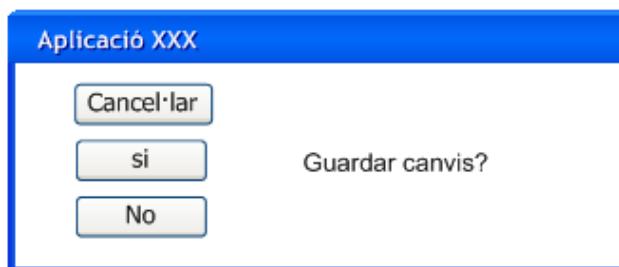
A més, caldrà limitar aquestes definicions del terme *usabilitat* a un context relacionat amb la informàtica i, més concretament, a les aplicacions de gestió que desenvolupareu (encara que aquestes definicions també serviran per a aplicacions o portals web).

Res millor que veure un exemple per a entendre el concepte d'usabilitat.

A continuació us mostrarem tres exemples d'interfícies en què s'espera una interacció de l'usuari amb una aplicació determinada, com podria ser, per exemple, un editor de textos.

En la figura 2.1 podeu veure una interfície que es podria considerar amb una usabilitat no gaire bona.

FIGURA 2.1. Usabilitat no correcta. Aquest diàleg no ens mostra les opcions de manera clara i imparcial.

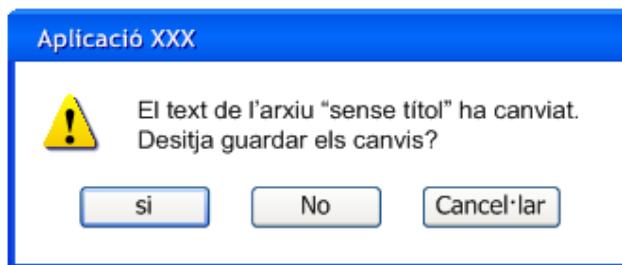


Vegeu com hi ha una pregunta que l'usuari ha de contestar seleccionant l'opció que convingui entre les tres possibles. Com veieu, hi ha tres botons de resposta que

són a l'esquerra, ben alineats, però amb un ordre que costa una mica d'entendre. La pregunta tampoc no ens dóna gaire informació. De tota manera, un usuari amb coneixements mínims d'informàtica podria interactuar amb aquesta interfície.

Vegeu en la figura 2.2 una interfície amb una usabilitat correcta.

FIGURA 2.2. Usabilitat correcta. Aquest diàleg mostra una usabilitat acceptable.



La figura 2.2 mostra una imatge que alerta d'una situació no prevista, com pot ser que vulguem tancar una aplicació sense haver desat els canvis. La pregunta que trobem és força més completa i comprensible, i les tres respostes possibles tenen una ubicació més fàcil d'identificar, amb la primera opció més probable a la dreta.

Vegeu també la proposta de la figura 2.3, que mostra un diàleg per a la interacció d'un usuari amb una aplicació. Aquesta seria d'una usabilitat òptima.

FIGURA 2.3. Usabilitat òptima. Aquest diàleg mostra les opcions d'una manera molt clara i senzilla.



En aquest cas ens trobem amb alguns canvis més. En primer lloc, la imatge que es mostra és més indicativa del tipus d'acció que es requerirà, continua amb el símbol d'alerta, però indicant amb la imatge del disquet que es tracta d'una possible operació de manca de gravació abans de sortir de l'aplicació.

A més, trobem que els botons de resposta per a l'usuari no estan alineats, sinó que tenen una distància irregular entre ells, cercant un tipus de resposta per part de l'usuari.

També cal remarcar el fet que les paraules dels botons de resposta són verbs, que indiquen amb més claredat les opcions a l'hora de seleccionar-ne una.

Per establir una definició més adaptada al nostre àmbit d'estudi, ens fixarem en les definicions que podem trobar en les normes ISO relacionades amb la usabilitat.

Normes ISO

Les sigles ISO corresponen a 'Organització Internacional per a la Normalització'. Es tracta d'una organització no governamental fundada el 1947, composta per representants d'organismes de normalització de més de cent cinquanta països. Podeu trobar més informació en els materials web.]

Entre d'altres ens fixarem en la ISO 9126 i en la ISO 9241. Aquestes normes ISO ens donen models de qualitat i qualitat en mètriques d'ús relacionades amb l'enginyeria del programari i la qualitat dels productes desenvolupats. La ISO 9241 ens ofereix guies d'usabilitat.

A partir d'aquestes normes ISO podem trobar definicions més concretes, com la de la ISO 9241-11, “Guia de la usabilitat” a l'any 1998.

La usabilitat és la part que tracta del grau amb què un producte pot ser usat per usuaris específics per a arribar a objectius específics amb eficàcia, eficiència i satisfacció en un context d'ús concret.

Una altra definició la podem trobar en la norma ISO 9126, que fa referència a la qualitat del producte en l'enginyeria del programari:

La usabilitat és la capacitat perquè, en el desenvolupament del programari, el producte final acompleixi les funcions d'atractiu per a l'usuari, de capacitat de ser comprès, après i usat, segons les condicions especificades d'ús.

Com veiem, podem trobar moltes definicions d'aquest terme, però en termes d'interacció de les persones amb els ordinadors caldrà parlar de la facilitat d'ús, la facilitat amb què nosaltres podrem utilitzar una eina concreta desenvolupada per a assolir un objectiu. Però caldrà afegir altres termes com claredat, elegància amb què s'ha dissenyat l'aplicació o lloc web.

Aquest terme també es podrà fer servir en altres àmbits com els llibres o manuals d'usuari (que seran més o menys fàcils de seguir i tindran més o menys usabilitat), l'electrònica de consum o qualsevol objecte mecànic. En tots aquests àmbits es podrà establir un sistema per a mesurar-ne la utilitat, la facilitat d'ús, la facilitat per a aprendre'n l'ús o l'apreciació par part d'un usuari determinat en un context determinat.

2.1.2 Dimensions de la usabilitat: característiques i atributs

La usabilitat d'una interfície gràfica d'usuari determina el grau d'acompliment dels objectius establerts i permet valorar el resultat. Per a aconseguir això és necessari establir unes característiques i uns atributs que es puguin valorar. Es coneixen com *les dimensions de la usabilitat* i són:

- Eficiència
- Eficàcia/efectivitat
- Satisfacció
- Atractiu

- Facilitat d'aprenentatge
- Facilitat del sistema per a ser recordat
- Tolerància a l'error

Eficiència

En termes econòmics, l'eficiència s'entén com la relació entre els recursos utilitzats i els resultats obtinguts.

Es pot entendre l'eficiència en l'ús de les aplicacions informàtiques. És a dir, una aplicació que obté uns bons resultats en les mesures d'usabilitat aportarà més productivitat a l'usuari i permetrà un ús més ràpid, més eficient de l'aplicació. Es tractarà de poder dur a terme més accions, més interaccions amb el programari en menys temps.

També podríem entendre el terme *eficiència* en l'escenari de desenvolupament del programari. En el cas d'una interfície d'usuari (o d'una aplicació informàtica) els recursos que s'hauran utilitzat seran les hores destinades al desenvolupament de les aplicacions o, més concretament, de les interfícies, juntament amb el cost de les eines de desenvolupament emprades. Els resultats s'entendran com el grau d'assoliments dels objectius marcats al principi del desenvolupament de les aplicacions, és a dir, en què aquesta interfície compleixi els requeriments.

Eficàcia/efectivitat

L'eficàcia o efectivitat és la capacitat d'aconseguir un objectiu planejat o desitjat. És a dir, si una aplicació acompleix els seus objectius, serà eficaç, si no, no ho serà.

Quan parlem de la usabilitat, fem referència a aquest terme, ja que serà el que ens indicarà si les interfícies aconsegueixen tots els objectius establerts al començament del desenvolupament d'una aplicació, essent una interfície que garanteixi l'efectivitat. A més, també caldrà que les funcionalitats establertes en els menús o icones de les interfícies facin el que han de fer.

Satisfacció

La satisfacció es defineix com un estat de la ment per a un ser humà.

Pot un usuari d'una aplicació informàtica obtenir satisfacció per l'ús d'aquesta? Potser depèn de la persona, no ho podem avaluar amb objectivitat, però el que sí que queda clar és que, si aconseguim desenvolupar una aplicació amb la qual un usuari té una actitud positiva quan la fa servir (encara que només no s'hi senti incòmode), haurem assolit l'objectiu de satisfacció. La satisfacció és

l'acompliment dels objectius plantejats inicialment per a obtenir uns resultats amb un cert grau de goig per a l'usuari.

Una vegada definides aquestes tres característiques, cal tornar a recordar el concepte de la usabilitat: part que tracta del grau amb què un producte pot ser usat per usuaris específics per a arribar a objectius específics amb eficàcia, eficiència i satisfacció en un context concret d'ús.

Potser ara aquesta definició ha quedat més clara. Ara cal establir una sèrie de guies i normes per a dur a terme un disseny acurat a les necessitats dels usuaris, amb un grau de la usabilitat més que correcte. Abans, però, afegirem algunes característiques d'usabilitat a la definició que hem fet més amunt i als seus atributs. Aquestes característiques seran:

Atractiu

El significat del terme *atractiu* està molt relacionat amb altres àmbits, però ara cal relacionar aquesta característica amb la usabilitat i les interaccions entre les persones i les màquines.

Una interfície és atractiva per a un usuari quan aquest n'accepta de bon gràt les caracterísques i l'ús, mostrant una predisposició per a utilitzar-la.

Aquesta característica queda molt vinculada a altres dimensions com, per exemple, la satisfacció. Una aplicació atractiva oferirà a l'usuari un treball agradable i satisfactori.

Aquest terme també es podria entendre com el fet de produir una actitud positiva en els usuaris.

Facilitat d'aprenentatge

Les interfícies han de ser prou intel·ligibles i amigables per a permetre l'aprenentatge intuïtiu per part de l'usuari.

Si una interfície segueix les premisses de l'entorn en què s'executarà i fa servir menús i icones semblants, l'adaptació i l'aprenentatge per part de l'usuari seran molt més ràpids.

Aquesta característica està associada a l'atribut d'eficàcia. Una aplicació eficaç s'ha de poder aprendre i utilitzar de manera fàcil i ràpida. Si una interfície requereix un ús continu de la documentació d'usuari, la seva eficiència no podrà ser mai bona.

Facilitat del sistema per a ser recordat

Una aplicació que s'ha d'utilitzar de manera esporàdica, o de manera regular, però, per exemple, una vegada al mes, ha de complir la característica següent: s'ha de recordar fàcilment com es fa servir.

Si les funcionalitats o les icones són difícils d'interpretar, cada vegada que un usuari hagi d'interactuar amb una interfície probablement necessitarà utilitzar el manual d'usuari que indiqui amb claredat com s'arriba a unes determinades funcionalitats.

Aquesta situació determinarà la predisposició d'un usuari a fer servir aquesta interfície i el temps que necessitarà per a utilitzar-la.

Tolerància a l'error

Quan un usuari interactua amb una aplicació, sempre hi ha opcions que no es poden seleccionar en un moment determinat o dades que s'han d'omplir en algun formulari. Hi ha aplicacions que no permeten l'ús d'una opció i no expliquen per què, o que permeten que es faci servir, permeten que l'aplicació falli.

Altres aplicacions no permeten seleccionar una opció si no és vàlida en aquest moment, o indiquen a l'usuari que cal introduir la lletra del DNI, per exemple.

Aquesta característica, com el seu nom indica, mesura el grau en què la interfície evita els errors o ajuda a superar-los.

Algunes d'aquestes dimensions d'usabilitat han estat definides com les 5 E's en l'article "Using the 5Es to Understand Users". Aquestes 5 E's estan referides a *effective, efficient, engaging, error tolerant, easy to learn*, és a dir, efectivitat, eficiència, atracció/satisfacció, tolerància a l'error i facilitat d'aprenentatge.

2.1.3 Normes ISO relacionades amb la usabilitat

Hi ha diverses normes ISO relacionades amb el terme *usabilitat*. La ISO (Organització Internacional per a la Normalització) es va crear amb l'encàrrec de promoure el desenvolupament de les normes internacionals de fabricació, comerç i comunicació, per a totes les indústries tret de l'electricitat i l'electrònica.



ISO (International Organization for Standardization)

Les normes ISO es divideixen per codis que indiquen diferents maneres de classificar les definicions d'estàndards, tant de procediments i processos com requeriments i atributs de productes i serveis.

Pel que fa a la qualitat, tenim les normes 9000, que fan referència als sistemes de gestió de la qualitat, en l'àmbit de fonaments i vocabulari, requisits o directrius.

Si ens fixem en les investigacions referents als aspectes ergonòmics en la interacció home-màquina, es basen en alguns estàndards internacionals, com ara els

següents:

- **ISO 9241.** Aporta requeriments i recomanacions relacionats amb les característiques del programari i el maquinari, com també de l'entorn que ha de millorar la usabilitat i els principis ergonòmics en l'ús de les noves tecnologies amb terminals visuals. Aquesta norma està dividida en moltes parts, afegides al llarg dels anys noranta. Les parts 10 i de la 12 a la 17 són les que estan relacionades amb el programari, i incideixen especialment en els dissenyadors i els responsables de les proves de les interfícies d'usuari.
- **ISO 9126.** Aquesta norma desenvolupa el model de qualitat en el programari, proposant uns atributs de qualitat com la funcionalitat, la fiabilitat, la usabilitat, l'eficiència, la facilitat de manteniment i la portabilitat.
- **ISO 13407.** Aquesta norma explica les activitats requerides per al disseny d'interfícies centrades en l'usuari. Aquestes activitats o requeriments fan referència a tot el cicle de vida del desenvolupament del programari, incident en la fase de disseny d'interfícies. És una norma pensada per als caps de projectes informàtics o per als responsables del disseny d'interfícies d'aquest.

La taula 2.1 mostra un resum de les normes ISO sobre l'orientació dels estàndards.

TAULA 2.1. Resum de les normes ISO sobre l'orientació dels estàndards

Estàndard Internacional	Descripció
ISO 9241	Requisits ergonòmics per a treball d'oficina amb terminals de visualització. Part 1 - Introducció general (1997). Part 2- Guia sobre els requisits de la tasca(1992). Part 10 – Principis de diàleg (1996). Part 11 – Guia sobre la usabilitat (1998). Part 17 – Diàlegs per a emplenar formularis (1998).
ISO 9126	Enginyeria del programari. Qualitat del producte. Part 1 – Model de qualitat. Par t4 – Qualitat en l'ús de les mètriques.
ISO 6385 (1981)	Principis ergonòmics en el disseny de sistemes de treball
ISO 13407 (1999)	Processos de disseny centrat en l'home per a sistemes interactius
ISO 10075 (1991)	Principis ergonòmics relacionats amb la càrrega de treball mental. Termes generals i definicions.
ISO/IEC 14598	Tecnologia de la informació. Avaluació dels productes del programari
ISO 11581	Tecnologia de la informació – Interfícies i símbols de sistemes d'usuari. Símbols i funcions d'ícones.
ISO 11064 (2000)	Disseny ergonòmic dels centres de control. Part 1- Principis per al disseny dels centres de control. Part 2 – Principis per a l'ordenació de les sales de control i els seus annexos.
ISO/DIS 14915-1 (2002)	Principis ergonòmics del programari per a interfícies d'usuari multimèdia. Part 1. Principis de disseny i estructura. Part 2 – Navegació i control multimèdia. Part 3 – Selecció i combinació de mitjans.

2.1.4 Mesura d'usabilitat d'aplicacions: tipus de mètriques

Arribats a aquest punt, trobem que hem conegut la definició d'usabilitat, n'hem analitzat les dimensions i hem vist algunes normes sobre aquesta. Ara cal fer un altre pas endavant i fer-nos les preguntes següents:

Com es dissenya una interfície d'usuari que compleix els requisits d'usabilitat? Com podem saber si una interfície compleix els requisits d'usabilitat que ens hem plantejat?

Precisament aquests són els temes següents a tractar. A partir de la segona qüestió parlarem de disseny d'interfícies, però ara cal parlar de les mètriques que podem fer servir per a mesurar la usabilitat de les aplicacions i, en concret, de les seves interfícies.

Tornem a plantejar-nos la pregunta: una vegada ens trobem davant una interfície, com en podem avaluar la usabilitat?

És una pregunta difícil de respondre. Hi ha molts paràmetres que es poden valorar. Molts són paràmetres subjectius, difícils de mesurar. Potser per a un usuari una interfície és intuïtiva, és satisfactòria, però per a un altre usuari no ho és. És per això que hem de cercar mètriques objectives.

Podem trobar un dels objectius dels tests d'usabilitat segons Kit (1995, pàg. 96), que ens indica que els tests d'usabilitat de les aplicacions es fan “per a adaptar el programari als estils de treball reals dels usuaris, en comptes de forçar els usuaris a adaptar els seus estils de treball al programari”.

A més a més, ens hem de plantejar altres preguntes, com, per exemple: en quina part del desenvolupament d'aplicacions informàtiques haurem de dur a terme les mesures d'usabilitat?, en la fase de finalització i transferència?, durant el desenvolupament?, o cal fer proves en l'anàlisi i durant el disseny? Aquest punt també és important. En funció del moment en què apliquem les mesures ens podrem trobar amb uns resultats o uns altres i, a més a més, serem a temps de superar els errors que s'hagin fet, o, si l'hem mesurat massa tard, potser no hi ha possibilitat de tornar enrere sense provocar un gran desgavell en la planificació.

Metodologies de gestió de projectes

Hi ha moltes metodologies de gestió de projectes informàtics i molts tipus de cicles de vida de desenvolupament del programari. Una proposta de metodologia de gestió de projectes en dividiria el desenvolupament en estudi de viabilitat, anàlisi, disseny, desenvolupament, finalització i transferència.

Es podrien dur a terme algunes evaluacions qualitatives, però cal establir un sistema quantitatiu de la usabilitat. Per a definir quantitativament la usabilitat, s'han definit algunes mètriques que intenten avaluar les aplicacions des d'un punt de vista de les característiques i atributs de la usabilitat.

Tipus de dades.

Les dades **qualitatives** són les que ens ofereixen informació de qualitat, expressada de manera verbal o escrita pels usuaris investigats. És a dir, no cercarem respostes

controlades com un *sí* o un *no*, o una selecció entre diferents opcions, sinó que deixarem respostes obertes perquè s'hi escoli l'opinió real i subjectiva de l'usuari.

Les dades **quantitatives** són dades que es poden quantificar. Aquestes dades es tractaran posteriorment amb mètodes matemàtics per a arribar a conclusions. Normalment les respostes seran numèriques o booleanes, valoracions amb una resposta gradual o percentatges subjectius, i les preguntes seran molt més concretes.

Per a obtenir dades qualitatives cal aconseguir informacions d'usuaris finals. En l'apartat següent es parlarà amb detall de les proves d'usuari, però, a tall d'exemple, podem entendre aquestes dades qualitatives com el resultat d'enquestes, de valoracions subjectives, de preguntes amb resposta oberta, opinions del producte, etc.

En canvi, si parlem de dades quantitatives, caldrà analitzar el comportament de l'usuari davant la nova aplicació, web o interfície, recollint informacions objectives com poden ser:

- Temps dedicat a cada interfície abans d'escollir una opció
- Temps o nombre de vegades d'utilització de l'ajuda o documentació
- Freqüència d'ús de les opcions
- Nombre d'errors en l'ús de les interfícies per usuari o cada cert temps
- Nombre d'accions completades en un temps determinat
- Comparació de temps utilitzat en segones o terceres visites a una mateixa interfície
- Nombre de suggeriments o queixes del producte

A partir d'aquestes mètriques quantitatives intentarem avaluar la usabilitat des del punt de vista de les seves dimensions (efectivitat, eficiència, atractiu, tolerància a l'error i facilitat d'aprenentatge).

Per a avaluar la usabilitat d'una interfície, d'una pàgina o portal web o d'una aplicació informàtica, hi ha moltes tècniques que fan servir les mètriques exposades més amunt.

Segons Rubin (1994), hi ha quatre tipus de tests. Aquests tests serveixen per a avaluar la usabilitat al llarg de les diferents fases del cicle de vida d'un projecte de desenvolupament de programari.

Aquests quatre tipus de test són:

- **Test exploratori.** Es durà a terme en les fases inicials del cicle de vida (requisits, anàlisi, disseny). Tindrà com a objectiu l'avaluació de l'eficiència dels conceptes de disseny inicial i localitzar errors inicials en la definició de les necessitats i assumpcions dels usuaris.
- **Test d'avaluació** d'operacions i aspectes del producte o servei. Durant les fases inicials i intermèdies del projecte (des de la fase de presa de requisits

fins a la de desenvolupament). Servirà per a avaluar les conclusions extrems dels tests exploratoris al principi del desenvolupament del programari per a validar que no s'han propagat els errors.

- **Test de validació.** Es du a terme durant les fases finals (provees, finalització i transferència). Servirà per a avaluar si el producte final compleix els requisits predeterminats d'usabilitat establerts en iniciar el projecte, permetent d'aquesta manera avançar-se a les possibles deficiències del producte. Aquest test certificarà la usabilitat del producte.
- **Test de comparació.** Aquest test implica a totes les fases del projecte. Anirà comprovant el producte amb els que ofereix la competència, i també comprovarà les diferents alternatives de disseny amb l'objectiu d'escollar la més senzilla de fer servir i d'aprendre. Aquest test es podrà aplicar en paral·lel amb altres tipus de test.

2.1.5 Proves d'experts i proves amb usuaris

Fins ara hem vist el concepte d'usabilitat, les seves dimensions, les seves característiques; hem vist mètriques per a mesurar-la, però ara caldrà analitzar les interfícies mitjançant proves fetes per experts en la matèria i proves fetes per usuaris finals de les aplicacions desenvolupades.

Aquestes proves estaran relacionades amb les característiques i dimensions, com també la mesura de la usabilitat i els tipus de mètriques existents.

Utilitzaran algunes de les mètriques descrites, tant les qualitatives en el cas de les proves d'experts com les quantitatives en les proves amb usuaris.

En primer lloc, caldrà tenir molt clar que el que provem serà el sistema desenvolupat (i, sobretot, les interfícies), no els usuaris o verificadors. Com més aviat fem les proves dintre del desenvolupament del projecte, millor serà per a corregir errors, però caldrà saber distingir entre les proves que poden fer els experts durant el desenvolupament i les interfícies que es podran mostrar als usuaris.

En aquests casos, caldrà tenir en compte que certes decisions preses arran dels resultats de les proves s'hauran de justificar molt bé davant els usuaris, acció que implica certa burocràcia. En altres situacions, les proves ajudaran a triar entre diverses alternatives plantejades, ja que l'usuari ens acabarà de donar un cop de mà en algunes decisions.

Per a dur a terme les proves caldrà desenvolupar prèviament un pla de proves. Aquest pla haurà de preveure:

- **L'abast** de les proves (què provarem i que no, fins on arribarem).
- Els **propòsits** d'aquestes (quins són els objectius, les raons o les justificacions per a provar en cada cas).

- Les **dates i els llocs** on es duran a terme (si anirem a casa de l'usuari, si farem que vingui ell, si necessitarem una ubicació especial, quanta estona haurà de durar, en quantes sessions diferents, etc.).
- Els **participants** en les proves (quants usuaris, si els usuaris seran *stakeholders* o no tindran cap vinculació amb el desenvolupament del projecte, quants experts, si estaran vinculats amb el projecte o seran externs).

2.2 Pautes de disseny de les interfícies d'usuari

A l'hora de desenvolupar un projecte de creació d'un programari, caldrà tenir en compte, en totes les fases del seu cicle de vida, la creació de les interfícies que facilitaran la interacció amb els usuaris. En cada una de les fases s'hauran de prendre decisions referents a les interfícies d'usuari, en més o menys mesura, que tindran un efecte decisiu en l'èxit del projecte.

Fases d'un projecte

Recordem que les fases del cicle de vida d'un projecte de desenvolupament de programari són:

1. Estudi de viabilitat
2. Anàlisi
3. Disseny
4. Desenvolupament
5. Finalització
6. Transferència

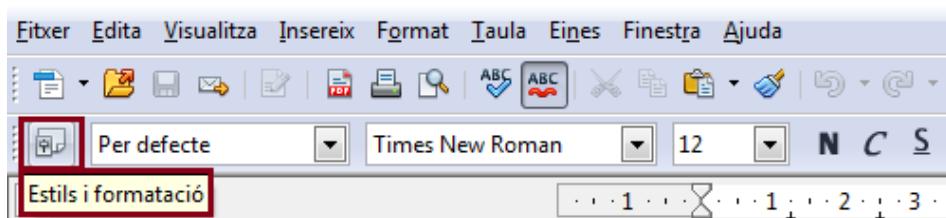
Per a desenvolupar les pautes de disseny de les interfícies d'usuari d'una aplicació informàtica, s'hauran de tenir en compte, al llarg de totes les fases del projecte, diferents elements com ara l'estructura, l'aspecte i els elements de les interfícies d'usuari.

Per a establir les pautes de disseny, també caldrà conèixer i entendre el tipus d'usuari per al qual es desenvolupa el programari i, en definitiva, les interfícies amb les quals interactuarà. Aquests podran ser usuaris experts, usuaris intermedis, o bé usuaris novells, sense gaire experiència.

- Un **usuari expert**. Dintre dels usuaris experts encara podem fer una segona classificació. Cal diferenciar entre l'usuari expert en l'ús de les tecnologies de la informació i l'usuari expert en les regles de negoci en què ubiquem el programari. Sigui com sigui, un usuari expert no necessita uns menús molt intuitius, ni unes icones amb unes explicacions molt clares, sinó que amb l'experiència i el sentit comú un usuari expert pot fer servir una interfície que es preocupa més d'ofrir molta funcionalitat que molta informació. Per exemple, per a aquest tipus d'usuari expert podrem oferir moltes funcionalitats amb molts menús i submenús o amb més necessitat de passos fins a arribar a l'opció desitjada. Aquest usuari tindrà la capacitat de trobar l'opció que necessita i podrà navegar per les opcions sense gaires problemes.
- Els **usuaris intermedis** són usuaris que es troben entre els experts i els novells. En aquest cas no parlarem de coneixement de negoci. Poden tenir un nivell mitjà de coneixement o ús de les TIC. Això farà que no necessitem oferir-los tantes facilitats ni informació com als usuaris novells, però sí que caldrà facilitar-los la feina tant com sigui possible.
- Un **usuari novell**. Un usuari sense gaire experiència (en l'ús de les TIC

o en coneixements sobre les regles del joc del negoci a què pertany el programari) necessita una interacció amb el programari molt més guiada, amb molta més ajuda, tant textual com gràfica. Un usuari novell necessita veure unes icones molt més explícites, que li facilitin la comprensió de les opcions. També necessita tenir una ajuda molt accessible i fàcil d'entendre, com, per exemple, mostrar petites frases o informacions quan el ratolí passa per sobre d'una icona, indicant-ne la utilitat (anomenades *tooltip* o indicador de funció, com es pot veure en la figura 2.4).

FIGURA 2.4. Tooltip o indicador de funció: informa sobre el significat d'una icona



Com es pot saber quan una aplicació s'ha de dissenyar pensant en usuaris experts, usuaris intermedis o usuaris novells? Aquesta pregunta no és senzilla de respondre.

Una aplicació informàtica desenvolupada des de zero, pot estar inicialment plantejada per a usuaris novells. Si els usuaris faran servir l'aplicació de manera esporàdica, o bé, si hi haurà una rotació d'usuaris molt gran en la utilització de l'aplicació, llavors aquesta s'haurà de dissenyar per a usuaris novells.

Si els usuaris sempre seran els mateixos, al cap d'un cert temps d'ús del programari, aquests mateixos usuaris es podran considerar usuaris experts i, potser, reclamaran un altre tipus d'interfície. Un exemple seria una interfície d'un TPV que faran servir diferents cambrers o encarregats al llarg del dia en un restaurant.

El mateix passarà amb les interfícies web. Haurem d'aconseguir un compromís entre disseny agradable, funcionalitats completes i facilitat d'ús.

Per tot això podem concloure que caldrà fer partícips els usuaris implicats en el projecte de la decisió del tipus d'interfície que s'haurà de dissenyar.

En el disseny d'una interfície caldrà tenir en compte alguns principis bàsics. Per exemple, l'any 1971 Hansen proposava quatre principis bàsics:

- **Conèixer l'usuari:** les seves capacitats, les seves necessitats funcionals i la seva evolució.
- **Minimitzar la memorització:** utilitzant selecció en lloc d'entrada de dades, emprant noms amb sentit en lloc de codis i noms críptics, fent que els resultats de les operacions siguin predictibles i facilitant l'accés a ajudes i a la documentació.
- **Optimitzar les operacions:** possibilitant l'execució ràpida d'operacions usuals, preservant la consistència visual, aprofitant les capacitats memorís-

tiques de l'usuari, i organitzant i reorganitzant les ordres d'acord amb la utilització que l'usuari en fa.

- **Permetre els errors:** donant a l'usuari missatges d'error entenedors, dissenyant per a evitar els errors comuns, permetent que les accions siguin reversibles i garantint la integritat del sistema en cas d'errors greus.

La major part de la interacció amb un ordinador es fa de manera visual. Això fa que les decisions sobre la tipografia, la distribució de la informació en la pantalla o de la selecció dels colors més apropiats no solament faran que la interfície sigui més o menys atractiva, sinó que la faran més o menys usable i en determinaran l'èxit. Alguns principis de disseny gràfic a tenir en compte són:

- **El principi d'agrupament.** Organitzar l'espai visible en blocs separats de controls similars i amb un títol per a cada bloc. Els sistemes de finestres actuals apliquen aquest principi de manera recursiva. Això ajudarà l'usuari a trobar la informació que necessita i a formar-se un model conceptual del funcionament del programa.
- **El principi de visibilitat i utilitat.** Els controls usats freqüentment han de ser visibles i fàcilment accessibles. Anàlogament s'hauran de dissenyar sistemes per a amagar o per a comprimir els menys utilitzats.
- **El principi de consistència intel·ligent.** Utilitzar una distribució de la informació similar per a funcions similars, per a habituar els usuaris a trobar la informació en els mateixos llocs en situacions semblants.
- **El principi d'economia del disseny.** Ometre qualsevol element que no aporti cap informació a les interfícies gràfiques d'usuari.
- **El principi del color com a suplement.** Utilitzar els colors amb mesura per a emfasitzar informació, sense ser un element exclusiu per a comunicar informació. El color és més fàcil d'usar malament que d'una manera correcta.
- **El principi de reducció del desordre.** És un principi que resumeix la resta de principis vistos anteriorment. Si només són visibles els controls utilitzats més freqüentment, agrupats en grups amb sentit, amb un ús minimalist del color i sense elements superflus, llavors es tindrà una interfície prou atractiva i prou funcional en la qual el desordre i l'arbitrarietat s'hauran reduït al mínim.

A continuació es tractaran diferents pautes de disseny que haurem de tenir en compte tant en el que afecta a l'estructura, a l'aspecte com als elements que componen una interfície d'usuari.

2.2.1 Disseny de l'estructura de les interfícies d'usuari

Quan parlem de l'estructura de les interfícies d'usuari ens referim a tots els elements que compondran aquesta interfície d'usuari i la seva ubicació dintre d'aquesta. Aquest conjunt d'elements i la seva ubicació formaran l'estructura de la interfície.

Els elements més habituals que podem trobar en l'estructura de les interfícies, i que ens permetran la comunicació entre l'home i la màquina, són els següents: menús, finestres, quadres de diàleg i dreceres de teclat.

Menús

El primer element que trobarem en una interfície serà el menú o els menús, que ens guiaran per totes les possibilitats que ens oferirà l'aplicació. En funció del tipus d'aplicació i de la interfície corresponent, aquests menús seran més o menys necessaris i més o menys profunds.

Per exemple, continuant amb el cas d'una interfície que es pugui fer servir a bars o restaurants, caldrà que ens situem i ens adaptem a les necessitats que puguin tenir, per a entendre que, molt probablement, tindrem interfícies a fer servir amb pantalles tàctils o amb terminals portàtils. Si es treballa amb pantalles tàctils no es productiu afegir menús de text, ja que la seva accessibilitat serà força complicada. En canvi, les opcions representades amb unes icones grans ens permetran utilitzar les interfícies de manera més adequada.

Els menús són una llista ordenada d'opcions que es mostraran en la interfície, amb possibilitat de contenir submenús en cada opció, que facilitaran la interacció amb l'usuari d'una aplicació, oferint-li la possibilitat de navegar per la interfície i d'escol·lir la funcionalitat que vol utilitzar en un moment determinat.

La importància dels menús en les interfícies rau en el fet que els usuaris finals de les aplicacions seran els responsables de la seva utilització i hauran de tenir l'autonomia en el seu ús, sense la necessitat d'una comunicació contínua amb el programador o dissenyador.

A continuació podem veure alguns exemples de diferents tipus de menús que podrem utilitzar en les interfícies d'una aplicació informàtica.

Menú de barra

És el menú més habitual, tant a aplicacions específiques com d'ús genèric, com poden ser de navegació per Internet o d'aplicacions ofimàtiques.

Aquest menú (vegeu la figura 2.5 i figura 2.6) s'acostuma a trobar en la part

superior de l'aplicació. Cada menú conté un altre menú desplegable. I dintre d'aquests, en algunes opcions, tenim altres menús desplegables. Els podríem anomenar *els menús en cascada*. Les opcions a què podrem accedir seran **totes** les opcions que ens oferirà l'aplicació informàtica, repetint algunes de les opcions que ens han ofert les icones que trobarem en les barres d'eines.

FIGURA 2.5. Menú de barra

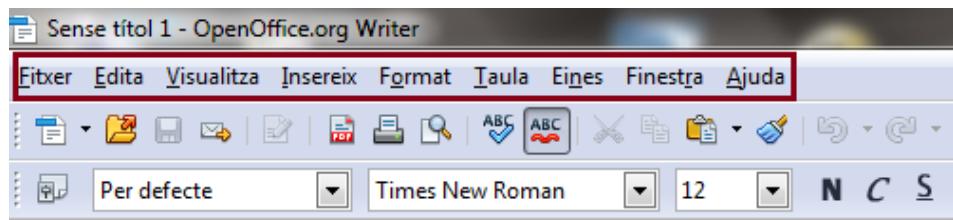
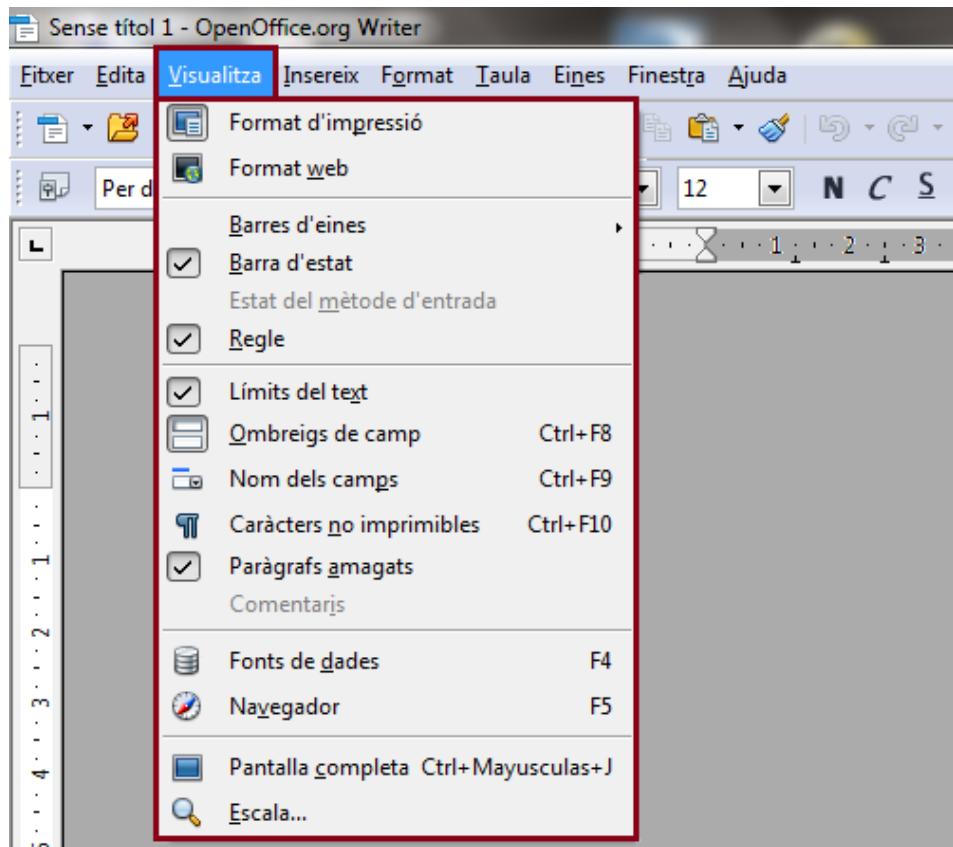


FIGURA 2.6. Menú de barra desplegat



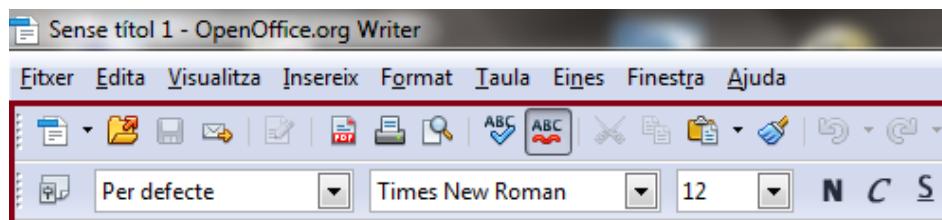
menú de barra desplegat en el fitxer

Normalment, les aplicacions d'ús genèric es poden personalitzar segons els gustos o les necessitats dels usuaris, al igual de les paletes o barres d'eines o altres funcionalitats de les aplicacions. Això donarà la possibilitat d'escol·lir quines opcions es volen veure o no en els menús, o que, de vegades, les mateixes aplicacions mostren només les opcions més utilitzades de manera prioritària. En altres casos, fins i tot es podrà seleccionar l'idioma d'aquests menús. Aquestes possibilitats són força més complicades de trobar en les aplicacions fetes a mida, llevat que l'usuari final les hagi establert com a requisit.

Barres d'eines o paletes d'eines

Aquest altre tipus de menú (vegeu la figura 2.7) és complementari de l'anterior. En les aplicacions trobarem els dos tipus de menús. El 90% de les vegades els usuaris faran servir els menús de les barres d'eines, deixant per a casos molt concrets la necessitat d'accendir als menús de barra.

FIGURA 2.7. Menú barra d'eines



Com hem comentat, en alguns casos concrets pot arribar fins i tot a substituir el menú de barra, com en els exemples de les aplicacions desenvolupades per a pantalles tàctils o per a dispositius mòbils.

Serà important seleccionar correctament les icones a fer servir en les barres d'eines perquè s'interpretin correctament. També cal indicar amb l'ajuda de petites explicacions el significat de les icones quan hi passem el ratolí per sobre.

Com a exemple d'icons en mostrarem algunes de les més habituals que podrem trobar en moltes aplicacions d'edició de textos (aplicacions ofimàtiques). Entre elles podem trobar les que fan referència a les propietats de les fonts o dels paràgraf. En la taula 2.2 es poden trobar alguns exemples:

TAULA 2.2. Exemples d'icons habituals

Icona	Descripció/funcionalitat
	Centrar el text. Pertany al grup que fa referència al paràgraf. Aquesta opció ens permetrà centrar el text seleccionat al mig del full d'edició sense cap alineació ni a l'esquerra ni a la dreta.
	Justificar. Pertany al grup que fa referència al paràgraf. Una vegada seleccionat un text, la justificació l'alinearà en els marges esquerre i dret, agregant espais addicionals entre les paraules si és necessari. Es crea una homogeneïtat en els laterals en l'aparença del text que s'edita.
	Negreta. Pertany al grup que fa referència a la font. Aplicar al text seleccionat la propietat de text en negreta, fent la font una mica més gruixuda i vistosa sense canviar-la de mida.
	Subratllat. Pertany al grup que fa referència a la font. Aplica al text seleccionat la propietat de text subratllat, afegint una línia molt fina a sota dels caràcters.
	Color de la font. Pertany al grup que fa referència a la font. Aplica al text seleccionat la propietat de canvi de color. En comptes del color per defecte, normalment el negre, podem escollir un altre color a aplicar en aquest text.

TAULA 2.2 (continuació)

Icona	Descripció/funcionalitat
	Reducir sagnat. Pertany al grup que fa referència al paràgraf. Una vegada seleccionat el text al qual volem aplicar aquesta acció, aquest text es desplaçarà cap a l'esquerra, la qual cosa reduirà els espais que hi havia entre el començament de la línia i el del text.

Altres aplicacions més específiques podran tenir les pròpies icons no tan fàcilment identificables. Per aquesta raó, en aquest cas o en altres aplicacions pensades per a usuaris novells, un recurs com l'indicador de funció ens podrà ajudar molt.

En moltes de les aplicacions, les barres d'eines es poden configurar. D'aquesta manera l'usuari podrà escollir quines funcionalitats pot tenir a mà perquè són més habituals i quines no li faran tanta falta.

Menús contextuais

:important: El menú contextual és la finestra que s'obrirà, en determinats sistemes operatius, quan fem clic amb el botó secundari del ratolí (botó dret o botó esquerre en funció de com estigui configurat) a sobre d'un espai concret d'una aplicació. Normalment són menús que restaran ocults a l'usuari i que l'usuari activarà a sobre d'un objecte determinat. Els menús contextuais s'obriran amb una finestra flotant al costat de l'objecte. :::

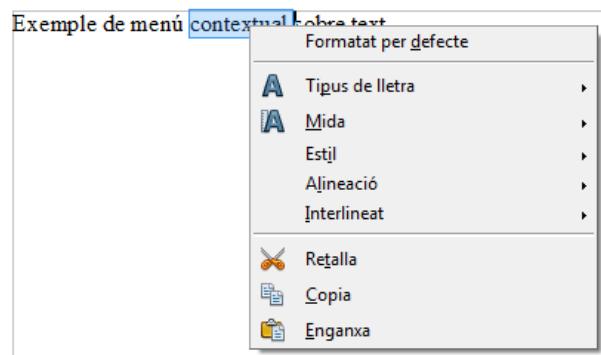
Per exemple, si hem començat a treballar amb un sistema operatiu i cliquem amb el botó dret del ratolí en l'escriptori, veurem una sèrie d'opcions que ens permetran treballar amb la seva configuració, oferint-nos opcions com la de crear nous elements (carpetes, arxius d'ofimàtica, etc.) o d'anar a la configuració d'altres elements del sistema.

En cas de fer servir el menú contextual en un entorn de sistema operatiu, veurem que les opcions que podem escollir creixen a mesura que instal·lem programes, de tal manera que serà un element viu.

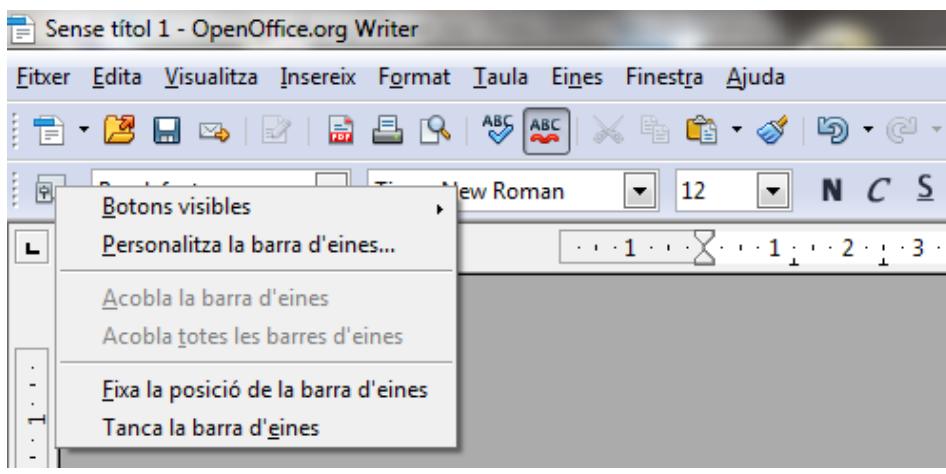
Moltes aplicacions d'ús genèric ofereixen menús contextuels. Per exemple, les aplicacions ofimàtiques com l'OpenOffice Writer.

Com es pot veure en la figura 2.8 i figura 2.9 el menú contextual ens oferirà unes opcions en funció del lloc concret de l'aplicació en què cliquem amb el botó dret. El menú contextual que podrem veure serà diferent si es genera en les barres d'eines que si es genera en el text que estem editant. A la vegada trobarem que en un mateix objecte un menú contextual tindrà unes opcions en un moment determinat i unes altres en un altre moment. Això serà així perquè un objecte o un element de la interfície pot passar per diferents estats i diferents contexts al llarg de l'execució del programari.

D'aquesta manera, un menú contextual oferirà les funcionalitats que podem aplicar a un objecte en un moment determinat.

FIGURA 2.8. Menú contextual sobre text

menú contextual en l'OpenOffice.

FIGURA 2.9. Menú contextual

Menú contextual barra d'eines

Per a què servirà el menú contextual? Ens facilitarà el treball més habitual amb l'aplicació que fem servir. Quan obrim un menú contextual obtindrem una finestra amb les funcionalitats relacionades amb la ubicació del punter (a sobre de quin objecte es trobi) en el moment de demanar el menú. En el cas que es mostra en la figura 2.9, podem veure com en generar el menú contextual obtenim un menú amb funcionalitats específiques.

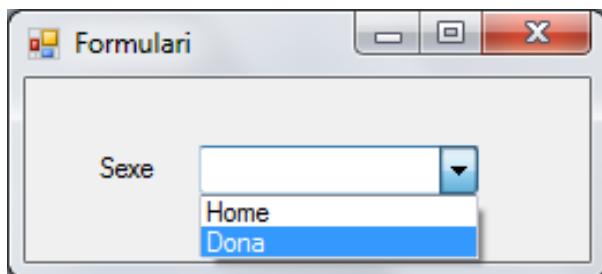
Menús desplegables

El *drop-down list* (llista desplegable) és un element GUI que es podrà fer servir en les interfícies, però també serà un tipus de menú. Si el relacionem amb altres elements GUI, podem trobar algunes similituds amb els quadres de llista o *list box* (llistes desplegables amb totes les opcions visibles) o els *combo box* (llistes d'opcions desplegables que permeten escriure).

En aquest apartat nosaltres ens fixarem en els menús desplegables com una opció més de menú per a escollir entre una llista d'alternatives.

Els menús desplegables ofereixen a l'usuari la possibilitat d'escol·lar entre un nombre determinat d'opcions. El menú no estarà desplegat, sinó que caldrà cliclar-hi a sobre per a veure les opcions que s'ofereixen. Quan l'usuari pot escollir una opció, la llista desplegable es tanca, torna a l'estat inicial amb l'opció seleccionada i aplica la funcionalitat vinculada.

FIGURA 2.10. Menú desplegable obert



Caldrà diferenciar entre els menús desplegables estàtics i els dinàmics. Els estàtics ofereixen un nombre determinat d'opcions que s'hauran programat manualment, que no es modificaran llevat que es modifiqui el programari desenvolupat.

Els menús desplegables dinàmics agafaran les opcions d'una base de dades o d'un fitxer. Si s'ha mordicat la informació externa, els menús oferiran unes opcions diferents en diferents moments d'execució.

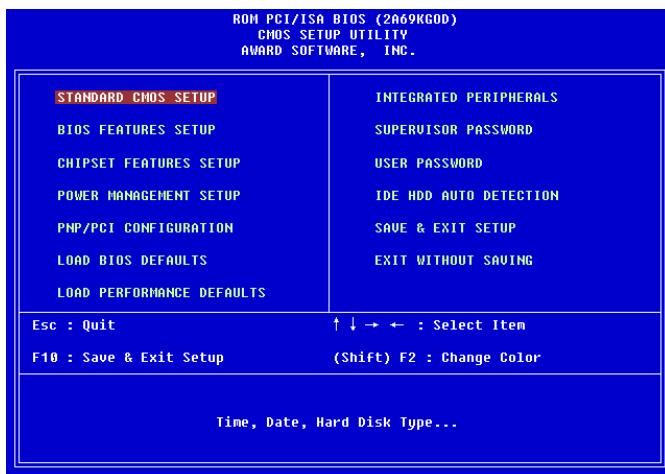
Menús de pantalla completa

Una opció cada vegada més en desús és la dels menús de pantalla completa. Aquests menús eren més habituals amb versions més antigues dels sistemes operatius, però encara es poden trobar en algunes aplicacions o en determinats entorns de treball, com les aplicacions per a dispositius mòbils. Per a entendre un exemple de menú de pantalla completa, podem fer referència als menús que trobem en les BIOS de les plaques mare. En podeu veure un exemple en la figura 2.11.

Aquest tipus de menú monopolitza tota la pantalla i només ens permet interactuar amb ella, fins que no seleccionem l'opció de sortir. Normalment, els menús de pantalla completa no permeten la convivència amb altres menús com els contextials o els de barres d'eines.

Una vegada vistos els diferents tipus de menús amb què podrem treballar, cal veure algunes **indicacions de disseny** d'aquests a tenir en compte a l'hora de desenvolupar les nostres interfícies.

És important seguir els estàndards de disseny de l'entorn en què es farà servir l'aplicació (és a dir, en funció del sistema operatiu amb què es treballi, caldrà adaptar-se a les pautes de disseny estableties pel sistema operatiu i per les aplicacions d'ús genèric habituals per a aquest entorn).

FIGURA 2.11. Menú de pantalla completa

En termes d'usabilitat del menú s'hauran de tenir en compte alguns aspectes que poden comportar problemes si no es dissenyen correctament: normalment els usuaris només veuran el primer nivell de jerarquia del menú i no, la resta de les opcions. Si han d'accedir a un segon o tercer nivell, hauran de memoritzar on es troba, cosa que pot provocar problemes amb usuaris d'una determinada edat o amb problemes psicomotoros.

Altres recomanacions per a seguir un estàndard en el disseny de menús podrien ser:

- Utilitzar una opció d'ajuda en els menús de barra o un quadre de diàleg amb les dades de contacte dels desenvolupadors del programari.
- S'ha d'arribar a un compromís entre els menús i el seu nombre d'opcions i de submenús. No es pot fer una jerarquia tan gran que després ens costi recordar on és una opció.
- Els menús no poden ocupar gaire espai en la interfície.
- Els termes s'han d'escollar de manera clara i concisa en les opcions dels menús i en les icones de les barres d'eines.
- Els menús s'han de dissenyar de manera visible atenent a la millor usabilitat per part dels usuaris.
- S'ha de permetre que els usuaris personalitzin, en la mesura del possible, els menús.
- Els menús han de ser els adequats tant per a usuaris experts com per a usuaris novells.

Finestres

El sistema més utilitzat actualment en aplicacions informàtiques és el sistema de finestres. Aquest sistema segueix les característiques dels sistemes operatius que permeten un treball amb més d'una finestra (o amb més d'una aplicació) alhora

WYSIWYG

El significat de **WYSIWYG** és *what you see is what you get*, és a dir, 'el que veus és el que rep's'.

seguint la filosofia del WYSIWYG. Aquest sistema permet l'accés de qualsevol usuari a l'ordinador (excepte els usuaris amb deficiències visuals).

WIMP

El significat de **WIMP** és *window, icon, menu, pointer*, és a dir, ‘finestres, icones, menús i punters’.

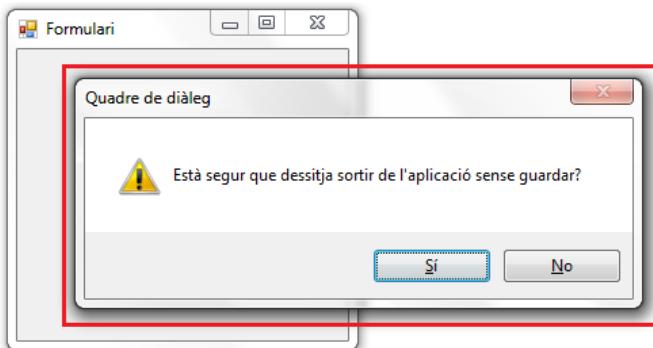
En termes informàtics, una finestra és un espai o àrea que conté una interfície d'usuari que pot permetre l'entrada de dades i mostra la sortida del sistema. Les finestres implementen en concepte WIMP.

Les **finestres** es poden classificar segons diversos tipus: les finestres d'utilitat, les finestres d'aplicació i les finestres emergents o *pop-up*.

Les **finestres d'utilitat** són finestres petites que aporten barres d'eines o altres informacions a les finestres d'aplicació, oferint-los un complement.

Les **finestres d'aplicació** contenen els menús de barra i les àrees amb els documents amb què es treballarà. Són les finestres principals de les aplicacions.

FIGURA 2.12. Finestres emergents



Les **finestres emergents** també són finestres complementàries a les finestres d'aplicació. Duen a terme una funció d'interactuació amb l'usuari oferint alguna informació o demanant alguna dada. En podeu veure un exemple en la figura 2.12.

Algunes de les propietats més importants de les finestres són:

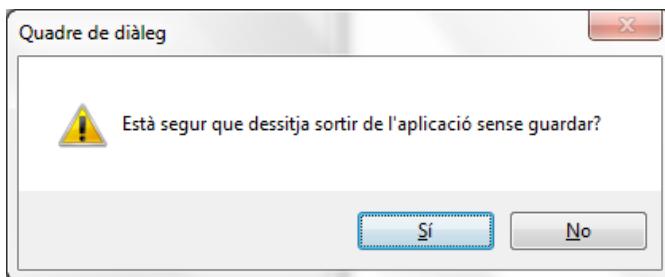
- Les finestres poden canviar de mida, fent-se més grans o més petites dintre de l'àrea de treball.
- Les finestres es poden desplaçar per la pantalla tant de manera horitzontal com vertical, només arrossegant-les per la seva barra de títol.
- Les finestres poden estar visibles o no visibles (ocultes), encara que continuin obertes o, fins i tot, treballant. Si una finestra es minimitza, es pot recuperar des de la barra de tasques del sistema operatiu.

Quadres de diàleg

Els quadres de diàleg o *dialog box* són un tipus de finestra especial que permet oferir a l'usuari una informació determinada o recollir d'aquest una resposta a alguna acció concreta de l'aplicació.

Podem classificar els quadres de diàleg en diferents tipus: els essencials, els de tipus alerta o els que permeten continuar treballant amb un programa. Els quadres de diàleg essencials són interaccions obligatòries per als usuaris. Els quadres s'obren de manera automàtica o amb alguna acció de l'usuari, per a indicar un error o per a prohibir l'accés a una determinada opció o per a indicar la introducció incorrecta d'un tipus de dada. Si no es contesta al quadre, no es pot continuar treballant.

FIGURA 2.13. Quadre de diàleg



Tipus de diàlegs

- **Diàlegs modals:** mitjançant finestres essencials, de resposta obligada.
- **Diàlegs no modals:** mitjançant finestres no essencials, de resposta no obligada per als usuaris.

Els quadres de diàleg no essencials no interrompen el funcionament de l'aplicació, com els anteriors. Es poden trobar dintre de la mateixa finestra de la interfície o obrir-se en una nova finestra, però sense l'obligatorietat de respondre al missatge per a continuar treballant. Podeu veure un exemple d'un quadre de diàleg es pot observar en la figura 2.13.

Els quadres de diàleg de tipus alerta són els que donen una informació o un missatge després de la interacció de l'usuari, i aquest haurà de seleccionar l'opció de continuar per a poder treballar. No són essencials, però es tipifiquen com un tipus independent per la seva característica informativa d'alertes.

Dreceres de teclat

Uns altres elements importants en el disseny de les interfícies d'usuari són les dreceres de teclat que es poden establir per a accedir a determinades funcionalitat.

Una drecera de teclat és una combinació de tecles o una combinació de tecles i clic de ratolí que ens permet accedir a una funcionalitat de manera directa sense haver de passar pel menú de barra o les barres d'eines.

És important conèixer l'entorn d'execució de les interfícies dissenyades. Un entorn amb un sistema operatiu Windows oferirà una sèrie de dreceres de teclat

Drecera de teclat

Un exemple universal de drecera de teclat és el que ens permet copiar un text seleccionat (**Control+C**) i enganxar-lo en un altre lloc de la interfície (**Control+V**)

molt conegeudes per usuaris experts. Per exemple, treballem amb la finestra que treballem, amb Control+Esc, accedirem al menú Inici del Windows de manera prioritària.

També hi ha dreceres de ratolí. Per exemple, si fem clic dues vegades en la barra de títol de qualsevol finestra, aquesta es maximitzarà.

Les dreceres de teclat (i de ratolí) ens permeten treballar de manera més ràpida i més eficient amb el sistema informàtic i, més concretament, amb l'aplicació amb la qual estem treballant. És important dotar una aplicació de dreceres per a les funcionalitats que s'utilitzin amb més freqüència, establint aquestes amb els usuaris en les fases de compliment de requisits.

2.2.2 Disseny de l'aspecte de les interfícies d'usuari

L'aspecte de les interfícies és tant o més important que l'estructura i els altres elements interactius. A l'hora de dissenyar les interfícies d'usuari és important, com hem vist, l'estructura, amb components com els menús, les finestres, els quadres de diàleg i les dreceres de teclat. Escollir els elements i on han d'anar és una tasca important.

En el disseny de l'aspecte de les interfícies d'usuari ens haurem de fixar en temes com els colors, les fonts que es faran servir per als diferents apartats de la interfície, les icones i la distribució dels elements.

Cal tenir en compte que per a tractar tots aquests aspectes no quedará clara la línia de separació entre el desenvolupador de programari o d'aplicacions informàtiques i el dissenyador. Moltes empreses deleguen aquestes funcions a la mateixa persona, però, de mica en mica, aquesta tasca es comença a diferenciar i es contracta una o més persones especialitzades en disseny, específicament per a aquest apartat.

El disseny de la interfície d'usuari és un element molt important (en alguns casos es podrà considerar crític) en el producte final a lliurar. Hem de tenir en compte que algunes decisions seran molt personals (ja coneixem la dita “contra gustos no hi ha res escrit”), però s'hauran de seguir unes indicacions普遍的.

Colors

Model RGB

El model codifica cada color en funció de la intensitat dels colors primaris que es fan servir per a formar-lo. Els colors primaris són el blau, el verd i el vermell. Per exemple, el **color negre** serà $(0,0,0)$, és a dir, l'absència de color. El **color blanc** serà tot el contrari, $(255,255,255)$, els tres color al nivell màxim.

Els colors és el primer que veurem quan la nostra vista es fixi en un objecte o, en el nostre cas, en una aplicació informàtica o una pàgina web. Aquesta primera visió o interrelació dels usuaris amb les interfícies és molt important, pràcticament crítica. Un usuari no deixarà de fer servir una interfície per raons de discrepàncies amb els colors, però, potser, l'obligarà a fer canvis, si els pot demanar.

Les paletes de colors ens donaran l'opció d'escol·lir entre una varietat molt àmplia de colors per a tots els elements de la interfície, tant per al fons com per a les

ícones i les lletres. Fins i tot es podria escollir fer servir una fotografia de fons.

FIGURA 2.14. Paleta de colors



A tall d'exemple, en la figura 2.14 es mostren les opcions a l'hora d'escol·lir colors amb el programa Paint. Aquesta paleta està basada en el model RGB (de l'anglès *red, green, blue*), model basat en la intensitat que donarem als tres colors primaris. Això ens ofereix la possibilitat d'escol·lir entre més de setze milions de colors.

Però podem veure que, a més de poder escollir la intensitat de cada color primari, ens donen l'opció d'escol·lir la intensitat de tres aspectes del color, como són la tonalitat o matís, la lluminositat o brillantor i la saturació.

Hi ha altres models de colors, com el model CMYK (de l'anglès *cyan, magenta, yellow* i *key/black*). Aquest model, que es farà servir més en la impressió en colors, està basat en l'absorció de la llum. Un altre model, semblant al CMYK, és l'RYB (*red, yellow, blue*) utilitzat de manera més comuna en les belles arts.

Si parlem de colors en el disseny, haurem de tenir en compte diversos aspectes.

El més habitual es trobar contrast entre els colors per a poder entendre millor la informació. Per exemple, farem servir text amb lletres fosques sobre un fons més clar, o al revés. També caldrà seguir una coherència en l'elecció dels colors, intentant relacionar un apartat de l'aplicació o un conjunt de funcionalitats amb un color i vincular aquest color amb tot el que tingui a veure amb les opcions relacionades amb l'apartat (per exemple, la gestió dels usuaris o la gestió dels proveïdors, etc.).

Un altre tema important a l'hora d'escol·lir els colors és el sistema operatiu per al qual es desenvoluparà l'aplicació. Hem d'intentar adaptar-nos a les característiques del sistema operatiu, seguint el seu tipus de finestres, ícones, colors, etc.

També cal tenir en compte la resolució de la pantalla en què es desenvoluparà la interfície. Fins fa poc, la mida habitual era el de 640x480 píxels amb 256 colors. Avui en dia, pràcticament tots els desenvolupadors treballen amb resolucions de 800x600 o superiors amb color real o *true color*.

Aspectes del color

- **Tonalitat:** tipus de longitud d'ona de la radiació.
- **Brillantor:** intensitat subjectiva del color.
- **Saturació:** pureza del color.

Per exemple, la llum blanca no té tonalitat, però sí brillantor.

Hi ha una sèrie de normes per a la utilització efectiva del color en les interfícies gràfiques d'usuari. En cas de seguir-les, el disseny cromàtic de la interfície serà correcte des d'un punt de vista cromàtic. És molt difícil donar un conjunt de normes tancades, a causa de la gran quantitat de variables que influeixen en la representació del color i en la seva percepció posterior. Algunes d'aquestes normes són:

- Evitar la visualització simultània de colors complementaris molt saturats.
- Evitar el color blau pur per al text o per a línies molt fines: el sistema visual dels éssers humans no està preparat per a rebre estímuls amb una longitud d'ona molt curta. El blau va bé per al fons, però no per al text.
- Els usuaris de més edat necessiten colors amb una lluminositat més alta per a poder-los distingir.
- Els colors es percepren diferents si les condicions d'il·luminació ambientals canvien: amb la llum del sol, de bombeta incandescent o de fluorescent, els colors es percepren diferents.
- La magnitud d'un canvi que es detecti en un color canvia al llarg de l'espectre: els petits canvis en vermells i en verds són més difícils de detectar que en els grocs o en els blaus verds.
- Cal evitar l'ús d'una gamma de colors basats en una sola tonalitat: els usuaris amb problemes de visió els perceben com a iguals.
- La lluminositat no és igual a la brillantor: dos colors amb la mateixa lluminositat poden ser perceperts amb diferent brillantor dependent de la tonalitat.
- Diferents tonalitats tenen nivells diferents de saturació: per exemple, el groc sempre sembla que està menys saturat que la resta de colors.
- No tots els colors són igualment llegibles: estem acostumats a llegir en negre sobre blanc; obtindrem la màxima llegibilitat amb fons clars i lletres fosques.
- Els colors s'interfereixen mútuament: la percepció d'un color canvia dependent dels colors que l'envolten.
- Cal tenir en compte la diferència entre el color de pantalla i el color imprès: els dispositius visualitzadors i els d'impressió no visualitzen els colors de la mateixa manera (l'un utilitza RGB i l'altre CMYK). Penseu que un color que utilitzeu en pantalla pot no tenir res a veure amb el mateix color imprès.
- S'han d'agrupar els elements relacionats sobre un mateix color de fons.
- Els colors similars expressen significats similars: es poden mostrar relacions entre elements assignant-los tonalitats similars o saturacions diferents de la mateixa tonalitat.
- Els colors lluminosos i saturats atreuen l'atenció de l'usuari.

- Cal ordenar els colors per la seva posició en l'espectre: si és necessària una utilització extensiva del color en una interfície, s'ha de fer de manera ordenada. La manera més natural és la de l'espectre, representada pel mnemotècnic anglès ROYGBIV (vermell, taronja, groc, verd, blau, indi i violeta).

Finalment, també s'ha de tenir en compte si algun usuari pot tenir problemes visuals amb certes combinacions de colors. Això és més fàcil de saber si el projecte es desenvolupa per a un determinat grup d'usuaris; si, en canvi, es desenvolupa per a usuaris no coneixuts o qualsevol persona pot ser usuari, llavors ho haurem de tenir en compte d'una manera més genèrica.

Fonts

Les fonts que es faran servir en una interfície també és un altre element important, tant pel tipus com per la mida de la font.

Les fonts són el format que donarem a les lletres que compondran els missatges de text per a interaccionar amb l'usuari.

Hi ha molts tipus de fonts i moltes possibilitats de fer-les servir, però cal anar amb compte amb el fet que aquestes fonts estiguin incorporades al sistema operatiu o que no siguin compatibles, cosa que pot fer que no es puguin fer servir de manera adequada.

Per a poder escollir les fonts haurem de tenir molt en compte els sistemes operatius o els entorns d'execució de les aplicacions desenvolupades. Com s'ha comentat en el apartat que fa referència al color, caldrà adequar-se a les característiques i/o a les fonts del sistema operatiu i procurar estar-hi en sintonia.

A més a més, com s'ha fet dit més amunt, caldrà utilitzar fonts anomenades *bàsiques* perquè siguin acceptades per qualsevol entorn de treball o sistema operatiu. En cas de fer servir una font no bàsica, caldrà empaquetar-la amb l'aplicació desenvolupada i instal·lar-la amb el programari d'instal·lació.

La **tipografia** és l'aparença que té el text i és l'element de disseny gràfic més bàsic d'una interfície. Algunes de les seves característiques són les següents: la font, el cos, el serif, el pes i la inclinació.

- **Font.** És el tipus de lletra. Per exemple: Helvètica, Times, Brush Script, etc.
- **Cos.** El cos és la grandària de la lletra i es mesura en punts o piques (1 pica = 12 punts).
- **Serif.** El serif és una herència romana de l'escriptura amb cisell damunt pedra i consisteix en una petita projecció al final dels pals de les lletres. Les lletres poden ser amb serif o sense.

- **Pes.** El pes fa referència al gruix de la lletra. Aquesta pot ser en negreta o normal.
- **Inclinació.** Segons la inclinació, la lletra pot ser vertical o cursiva (anomenada també *itàlica*).

La **llegibilitat del text** es veu afectada per diferents factors:

- **Proporcionalitat.** És una característica de la majoria de fonts que consisteix a assignar un espai horitzontal a cada caràcter d'acord amb l'amplada d'aquest. Les fonts que assignen a tots els caràcters el mateix espai s'anomenen *fons d'amplada fixa* (en què un caràcter com una *i* ocupa el mateix espai que un caràcter com una *m*).
- **Grandària de la font.** Cal recordar que el text en una pantalla es llegeix un 25% més a poc a poc que en paper. Un text normal té com a grandària apropiada 10 punts. Una de 12 punts és llegeix encara millor. Fent servir diferents grandàries es poden aconseguir jerarquies de text (títols, subtítols, capçaleres, etc.).
- **Majúscules/minúscules.** La barreja de majúscules i minúscules fa els textos més llegibles. Això és degut al fet que quan llegim reconeixem les paraules familiars per la forma i no les llegim caràcter a caràcter.
- **Espaiat i interlineat.** L'espaiat, tant de caràcter com de paraula, depèn de la font que utilitzem i és recomanable no tocar-lo. La modificació de l'espaiat es pot justificar en situacions determinades com en títols o etiquetes de botons, però poques vegades en el text normal. Pel que fa a l'interliniat, cal tenir present que com més petit sigui més difícil serà la lectura perquè costarà trobar el començament de les línies.
- **Longitud de la línia.** Si la longitud de la línia de text és massa llarga, es fa difícil de trobar el començament de la línia següent, i si és més llarga del que podem abastar amb el moviment de l'ull ens obligarà a moure el cap, cosa que ens provocarà fatiga. Els textos impresos en un llibre han de tenir una longitud d'uns seixanta caràcters, i els d'una columna de diari, d'uns trenta caràcters.
- **Justificació.** La justificació és la inserció d'espais extra entre les paraules per a alinear les línies tant a la dreta com a l'esquerra. Aquesta tècnica, encara que visualment és més estètica, alenteix la lectura.
- **Maquetació.** La maquetació és la distribució dels diversos paràgrafs de text (i d'altres elements gràfics) en un espai determinat. La primera impressió que tenim en obrir un llibre, executar un programa o visitar un web és deguda a la maquetació. La maquetació pot fer que ens quedem en el web que estem visitant o que anem a un altre de continguts similars però amb una presentació visual més atractiva.
- **Marges.** Cal evitar la “síndrome del processador de textos”, que consisteix a escriure de banda a banda de la pàgina sense deixar marges ni a la dreta ni

a l'esquerra. Cal deixar marges generosos d'acord amb la vostra maquetació i amb la longitud de la línia.

- **Distinció tipogràfica.** Cal utilitzar els canvis tipogràfics de cursiva, el canvi de font i negreta només quan aportin realment informació i sense abusar-ne. L'ús excessiu de fonts i de recursos tipogràfics s'anomena correntment *fontitis*.

Icônes

Les icônes que es fan servir en les interfícies també són importants en la interacció amb els usuaris. Ja hem parlat d'algunes icônes en l'apartat dels menús, quan fèiem referència al menú de barres d'eines. Parlarem del mateix tipus d'icônes amb les mateixes funcionalitats, però en aquest apartat ens fixarem més en les icônes d'acció de les interfícies d'usuari.

Les icônes ofereixen la representació d'una unitat de significat. Si aquesta unitat de significat es volgués expressar amb text, ocuparia molt més espai del que ocuparem fent servir una icôna.

Aquestes unitats de significat poden ser idees, conceptes, accions o altres. Per exemple, podem tenir la icôna que representa amb un dibuix d'un disquet l'acció d'enregistrar una feina feta o un document.

D'aquesta manera podrem representar més accions amb un espai més limitat, a més d'ofrir una usabilitat millor, més directa, més ràpida i intuïtiva, sempre que les icônes que es facin servir siguin conegeudes.

Quan un usuari fa servir de manera repetitiva una aplicació, les icônes i el seu significat es memoritzen fàcilment. En canvi, l'ús d'una aplicació per part de molts usuaris diferents o d'un mateix usuari de manera no gaire continuada implicarà la necessitat d'utilitzar icônes molt fàcilment recognoscibles per als usuaris, que siguin semblants a les que es fan servir de manera habitual en altres sistemes operatius o altres aplicacions molt utilitzades.

Hi ha interfícies en què l'aspecte i l'impacte visual és molt important, pràcticament són uns dels objectius de la interfície. Cal que la interfície cridi l'atenció de l'usuari de manera molt impactant. En aquests casos, també és molt adequat fer servir icônes. L'usuari coneixerà el funcionament de la interfície a mesura que descobreixi les icônes i el seu significat.

A l'hora de dissenyar interfícies amb icônes, també hem de tenir en compte les limitacions que poden presentar.

De vegades, la missió de les icônes és difícil en intentar representar una acció o idea difícilment representable. Si hi ha accions similars és complicat trobar icônes que diferenciïn amb exactitud els matisos que faran que les accions siguin diferents. A més, la interpretació per part dels usuaris serà molt personal i pot donar peu a cometre errors. És per aquesta raó que es recomana no identificar

amb icones accions crítiques d'una aplicació.

A l'hora de dissenyar o escollir icones, hem de tenir en compte que n'hi ha de diferents tipus. Les que es basen en l'analogia ofereixen imatges que intenten semblar el que volen representar. Les que es basen en les mostres simbolitzen elements que intervenen en el que es vol referenciar. Les que es basen en els símbols són representacions més abstractes. Les icones arbitràries reclamen que l'usuari n'aprengui el significat.

Cal fer un ús raonable de les icones, sense utilitzar el recurs més del recomanable. La velocitat de reconeixement de les icones acaba essent semblant a la del text, però mai no és superior. De fet, es produueixen més errors amb l'ús de les icones que amb l'ús dels menús de text. Aquests errors poden influir en l'aprenentatge de l'aplicació per part de l'usuari o, fins i tot, en el seu rebuig o la seva falta de satisfacció.

Distribució dels elements

Arribats a aquest punt, s'ha parlat de diferents conceptes importants en l'aspecte d'una interfície d'usuari, com poden ser els colors, les fonts i les icones. En l'apartat anterior s'han vist diferents elements de l'estructura de les interfícies com els menús, les finestres, els quadres de diàleg i les dreceres de teclat. Ara toca parlar de com es distribueixen aquests elements per la interfície.

La distribució dels elements ha de permetre un ús de les interfícies i un aprenentatge òptim d'aquestes als usuaris. Cal decidir la manera de presentar els menús de text, les barres d'eines, les icones, els quadres de diàleg, etc. i la seva ubicació en la interfície. Els elements han de presentar una distribució uniforme, tenint en compte l'agrupació de funcionalitats relacionades i les zones de la pantalla que els usuaris visualitzaran més.

Els objectius de la distribució dels elements per la interfície són els següents: aconseguir que la interfície sigui fàcil d'usar, fàcil d'aprendre, segura, fiable i efectiva a l'hora de dur a terme les accions necessàries per a una aplicació. A més ha de ser consistent.

Al igual que s'ha anat comentant al llarg d'aquest apartat, és interessant adaptar la interfície desenvolupada a l'entorn de treball d'aquesta, distribuint els elements de manera anàloga a altres aplicacions o interfícies del sistema operatiu.

Tot allò que el dissenyador posa en el disseny d'una interfície ha d'estar justificat, ha de tenir una consistència i ha de seguir una organització espacial. Les interfícies d'un navegador d'Internet, per exemple, contenen uns exemples que estan disposats d'una manera determinada en l'espai que serà familiar per als usuaris. Aquesta distribució obedeix a unes normes de disseny de GUI, i ofereix una coherència externa amb altres programes similars que són coneguts.

Alguns autors, com l'artista i dissenyador de la Bauhaus, Josef Albers, han parlat així del disseny:

Dissenyar és planificar i organitzar, ordenar, relacionar i controlar. Resumint, comprèn tot el que sigui anar en contra del desordre i la casualitat. Per tant, mostra una necessitat humana i descriu el pensament i l'actuació de l'ésser humà.

Un altre concepte que cal tenir en compte és el nombre **màgic de Miller**:

Tots els components gràfics d'una interfície d'una aplicació informàtica (menús, àrees de dades, icones, diàlegs, taulets o imatges) estan disposats damunt d'una retícula imaginària que és respectada al llarg de l'aplicació (els menús no canvien de lloc ni tampoc no ho fa l'àrea de visualització de dades, per exemple).

La grandària, distribució i nombre de divisions de la retícula és crucial per a la usabilitat de la interfície. Aquests paràmetres estan determinats pel tipus d'informació que cal visualitzar, per la resolució de la pantalla i per les condicions de visualització (no és el mateix un caixer automàtic que un ordinador de butxaca o *palmtop*).

En general, el nombre de divisions de la retícula segueix el recurrent nombre màgic de Miller de $7 +/- 2$, que és el nombre màxim d'elements que una persona pot retenir en la memòria de curt termini.

El mateix esquema es pot aplicar recursivament per a qualsevol element, tant si és un tauler de control com una barra de menús, una barra d'icones o un diàleg.

Un altre aspecte que cal tenir en compte en l'organització espacial de la nostra informació són les relacions entre els diferents elements d'informació. Els elements d'informació relacionats pel contingut també han d'estar visualment relacionats.

2.2.3 Disseny dels elements interactius de les interfícies d'usuari

Hi ha molts altres elements que es fan servir en les interfícies d'usuari, a més dels que hem comentat fins ara. Tant els menús com les finestres, els quadres de diàleg o les icones són elements que faciliten la interacció entre l'usuari i la interfície.

Aquests elements permeten la navegació per l'aplicació, mostren les alternatives que té l'usuari i li ofereixen la possibilitat d'executar algunes accions. Són sempre, però, informacions estàtiques. Altres elements que oferiran un altre tipus d'informació a l'usuari, una informació més dinàmica, per exemple, amb el resultat de les dades obtingudes a partir d'una consulta o oferint la possibilitat d'afegir informació a la ja emmagatzemada. Entre aquests altres elements trobem els botons d'ordres, els botons de relleu, els *radio buttons*, les caselles de selecció o les llistes desplegables.

Elements d'una GUI

Entre els elements d'una interfície d'usuari podem trobar menús, finestres, text (tipografia digital), icones, dispositius de control i altres elements d'entrada i/o de sortida de dades com botons d'ordres, com els *radio buttons*, les caselles de selecció (*check box*), els *scrolls*, o les llistes de distribució.

Botons d'ordres

Els botons d'ordres es consideren dispositius de control. D'aquesta manera un botó és un objecte de control de la interfície d'usuari. Aquests botons permeten introduir dades de confirmació a l'aplicació, però també permeten seleccionar dades o introduir-ne de noves en el sistema.

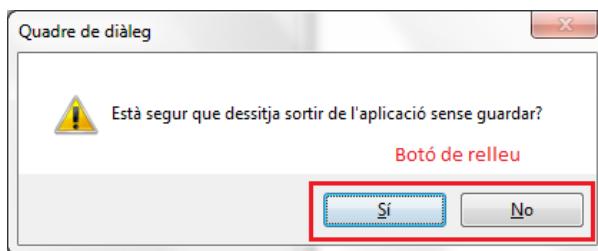
Els botons ens serveixen per a construir les interfícies i actuen com a interpretacions visuals de les funcionalitats que representen. Els botons són part, com les icones, de la gramàtica visual de les interfícies d'usuari.

Podem trobar diversos tipus de botons, com els botons d'opcio (*radio buttons*), els botons de confirmació (*check box*), els botons de relleu, les pestanyes, les barres d'arrossegament (*sliders*), barres de desplaçament o *scroll*, etc.

Botons de relleu

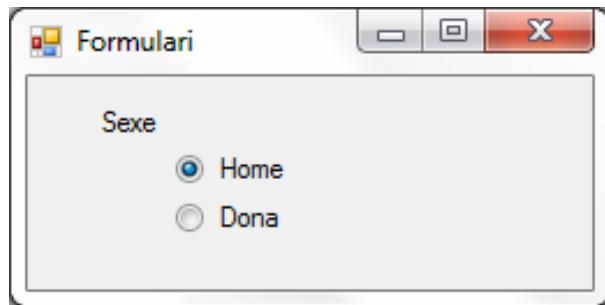
Són els botons més habituals, els que més es fan servir en tota mena de sistemes operatius. El seu nom el deuen a la seva forma, que cerca la simulació de volum. Un botó de relleu conté un text o una imatge que identifica una funcionalitat de l'aplicació. Quan un usuari tria un botó de relleu i hi clica a sobre, el botó simula un enfonsament i du a terme l'acció relacionada amb ell. Podeu veure un exemple d'aquest tipus de botons a la figura 2.15.

FIGURA 2.15. Botó de relleu



Radio buttons

Els botons en forma de radi o botons d'opcio ofereixen a l'usuari una llista d'opcions que tenen a l'esquerra o la dreta (menys habitual) una petita rodona per a poder clicar a dintre. Si hem seleccionat una de les opcions, aquesta rodona petita s'omplirà amb un punt i ens indicarà que aquesta és l'opcio escollida. Cal tenir en compte que, amb l'opcio dels botons d'opcio, només podem seleccionar una de les opcions ofertes.

FIGURA 2.16. Botó d'opcio

Com a exemple d'utilització tindríem les enquestes, en què darrera d'una pregunta s'ofereixen quatre o cinc respostes, de les quals l'usuari només en pot seleccionar una. Al final hi ha un botó de relleu sobre el qual s'ha de clicar per a indicar que aquesta és la nostra selecció correcta. En la figura 2.16 hi podeu veure un exemple on es fa escollir el gènere a un formulari.

Aquest recurs s'ha fet servir sovint en el sistema operatiu Mac OS.

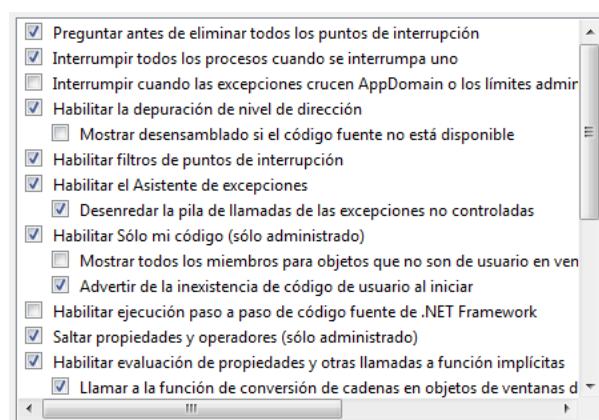
Check box

Aquests botons són molt semblants als anteriors, però amb la particularitat que, a l'esquerra o la dreta del text indicatiu de les opcions, estan representats amb petits quadradets. En cas de seleccionar l'opció corresponent a un quadre, aquest s'omplirà amb un símbol de *check*.

La gran diferència entre els botons de relleu i els botons de *check box* és que en els *radio buttons* només podem seleccionar una opció. En canvi, amb els *check box* podem seleccionar tantes opcions com vulguem.

Un exemple podria ser un formulari de registre en què un usuari està introduint les seves dades i es pregunta per les seves aficions. N'haurà de seleccionar tantes com vulgui dins d'una llista de quinze o vint opcions.

Podeu veure un altre exemple d'aplicació dels Check Box en la figura 2.17.

FIGURA 2.17. Check box

En la taula 2.3 revisarem algunes de les propietats que tenen tant els *radio button*

com els *check box* per entendre'n millor les funcionalitats.

TAULA 2.3. Algunes propietats dels radio button i els check box

Propietat	Descripció
<i>Name</i>	Com per a tots els controls, podrem escollir un nom que ens identificarà o bé en un <i>check box</i> o en un grup de <i>radio buttons</i> que caldrà que tinguin tots el mateix nom. Serà important poder-los distingir a l'hora de treballar amb codi i haver-hi de fer referència.
<i>Value</i>	És la propietat que ens indica el valor del control. Aquest valor estarà associat a cada opció del <i>check box</i> o del <i>radio button</i>
<i>Checked</i>	Ens indicarà si l'opció està marcada o no. Amb un valor <i>verdader</i> l'opció estarà escollida, amb un <i>fals</i> no ho estarà.
<i>Disabled</i>	Bloquejarà el control amb què treballem i actuarà com si no existís.

Alternatives

Drop-down list: és un menú desplegable en què podrem escollir una opció dins una llista de possibilitats, sense poder escriure. **List box:** ens ofereix una llista d'opcions amb totes elles a la vista, sense cap desplegable.

Llistes desplegables

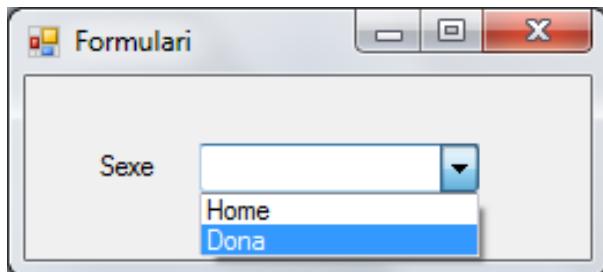
Uns altres elements interactius de les interfícies d'usuari són les llistes desplegables, conegudes en anglès com a *combo box*.

Per a entendre els *combo box* cal fer una revisió d'altres elements semblants, però amb matisos que els fan diferents. Hi ha els menús desplegables (*drop-down list*), que ens servien tant per a mostrar un conjunt d'informacions com per a fer servir la seva funcionalitat de selector d'opcions.

En canvi, el *combo box* ens permet escriure en un quadre de text, a la vegada que podem veure una llista d'opcions. Es tracta del mateix sistema que fa servir el cercador Google: quan comencem a escriure la paraula o frase que volem cercar, ens va mostrant propostes de paraules de cerca, basades en altres cerques d'altres usuaris o en l'historial de cerques que nosaltres haurem fet.

Una llista desplegable és una combinació d'un quadre de text (*text box*) i un menú desplegable (*list box*).

Si volguéssim oferir una llista d'opcions sense permetre l'opció d'agregar un nou valor, llavors estaríem obligats a fer servir el *list box*, ja que el *combo box* no tindrà cap opció que permeti aquesta configuració. En canvi, sí que podrem afegir a la llista d'opcions desplegables nous ítems. Amb les propietats de l'element podem agregar nous ítems sense haver d'actualitzar la interfície. En podeu veure un exemple en la figura 2.18.

FIGURA 2.18. Drop-down list

En la taula 2.4 revisarem algunes de les propietats que tenen els combo box:

TAULA 2.4. Propietats dels //combo box//

Propietat	Descripció
<i>Name</i>	Identifica el <i>combo box</i> . Aquesta propietat és present en tots els controls.
<i>Text</i>	Conté una cadena com a contingut de text del control. Serà el text associat al control.
<i>SelectedValue</i>	Serà el valor seleccionat en un moment determinat en el <i>combo box</i> .
<i>Items</i>	Contindrà un objecte que representa la col·lecció d'elements que contindrà el <i>combo box</i> .

2.3 Altres pautes de disseny

El disseny de les interfícies d'usuari ha de tenir en compte molts elements i moltes regles i detalls per a arribar a oferir una usabilitat acceptable.

Ara cal parlar dels elements i regles vinculats amb la gestió de les dades, tant del que té a veure amb la seva presentació, com del relacionat amb la seva possible manipulació o destrucció. És a dir, la vinculació de les dades a una base de dades amb la interfície amb què treballarà l'usuari.

Altres elements a tenir en compte són la seqüència de control de l'aplicació, l'asegurament de la informació i altres pautes específiques per al disseny d'interfícies amb elements multimèdia.

Abans cal revisar algunes de les regles no escrites que hem de tenir en compte en el disseny d'interfícies d'usuari:

- L'usuari de la interfície ha de tenir el control d'aquesta (i, a més a més, sentir que té aquest control).
- S'han de fer servir missatges i textos descriptius, fàcils d'entendre per als usuaris.
- Els missatges que es donaran a l'usuari han de ser positius i actius sense semblar insultants o graciosos.

- Les accions i tasques vinculades s'han de poder interrompre i reiniciar en un altre moment.
- Els usuaris han de poder utilitzar el teclat i el ratolí per a comunicar-se amb la interfície.
- Conèixer la cultura del país pot ajudar a evitar malentesos en els missatges a l'usuari.
- La interfície ha de poder ser personalitzada i manipulada.
- Ha de facilitar la navegació per la interfície, trobar les funcionalitats i la sortida o tancament d'aquesta.
- Ha de permetre diferents nivells d'ús de la interfície per a diferents nivells d'usuaris (amb més o menys experiència).
- Ha d'intentar facilitar al màxim la feina als usuaris, per exemple, mostrant llistes en les quals es pugui escollir una opció abans d'haver de teclejar el nom d'aquesta opció.
- Ha d'associar accions als diferents objectes presents en les interfícies.
- Ha de fer la interfície transparent per als usuaris

2.3.1 Presentació de les dades

Les dades que es mostraran fent servir les interfícies d'una aplicació han de seguir unes pautes concretes per a no confondre els usuaris. Segons quina informació es vol presentar s'ha de tenir accés a un suport extern que contingui les dades, o bé es tindran les dades incorporades en la nostra interfície.

En fixarem en cas que les dades estiguin ubicades en un suport extern a l'aplicació, normalment en una base de dades o en un arxiu de text o binari. Potser volem fer servir dades que es troben en altres aplicacions o són resultats d'aquestes, però per fer això tractarem la informació amb fitxers intermedis fins a fer-la arribar al nostre programari.

L'aplicació desenvolupada (i les seves interfícies) servirà per a comunicar l'usuari amb aquestes dades, però això s'ha de fer de manera transparent per a l'usuari.

A l'hora de dissenyar la presentació de les dades, hem de tenir en compte algunes qüestions:

- **Quin tipus d'informació vol veure l'usuari?** Vol tenir accés a la informació de gestió o vol tenir accés a informació estadística o de resum per a poder prendre decisions? Per posar un exemple, serà diferent la manera de presentar les dades a un usuari que ha de donar d'alta nous proveïdors, ha

de poder modificar les seves dades i ha de poder crear comandes a aquests proveïdors, que si l'usuari està interessat a tenir un resum anual de les vendes per referència, per mes i per tenda dels productes que compra a un proveïdor.

- **Quin ús en voldrà fer?** Voldrà només consultar, o haurà de poder modificar, esborrar i crear nova informació? També es tractarà diferent la informació a què es podrà donar accés per a gestionar i per a manipular que les dades que seran només de consulta.
- **Si les dades es modifiquen de manera contínua, l'usuari ha de veure reflectits aquests canvis en el mateix moment?**
- **La informació a mostrar serà numèrica o en mode text?**

Totes aquestes qüestions ens ajudaran a decidir-nos per una manera de presentar les dades o per una altra.

Un bon coneixement dels usuaris també ens ajudarà a oferir una presentació adequada de les dades amb les quals haurà de treballar.

2.3.2 Seqüència de control de l'aplicació

Tota aplicació amb una interfície d'usuari ofereix molts elements i moltes funcionalitats als usuaris. Aquests elements ens portaran a poder dur a terme moltes operacions, la majoria de les quals treballaran amb dades. Algunes d'aquestes operacions seran directes i s'executaràn de manera immediata. A altres operacions, hi podrem arribar des de diferents ubicacions de la interfície. Altres operacions seran compostes i comportaran més d'un accés o una modificació a la base de dades.

Hem de tenir en compte que certes operacions es podran interrompre per un mal ús de la interfície o de l'aplicació o per raons alienes al programari desenvolupat. Altres operacions hauran de ser confirmades per l'usuari per a validar-ne l'execució correcta.

La seqüència de control de l'aplicació es refereix a les accions que l'usuari durà a terme amb el programari i la lògica d'execució d'aquest. És a dir, si l'usuari demana executar una operació que comporta diverses accions amb la base de dades, haurem d'establir un control de la seqüència d'operacions que es duran a terme per a garantir que la informació s'ha mantingut consistent en la base de dades.

Un bon disseny d'una interfície (i d'una aplicació en general) ha d'incloure uns objectius que incloguin accions de control de consistència, una necessitat mínima d'accions de control i flexibilitat en els controls de seqüència per a adaptar-se a les diferents necessitats dels usuaris.

Un dels punts més determinants en la satisfacció i en l'acceptació per part de l'usuari d'una aplicació informàtica a l'hora d'utilitzar una interfície és la sensació de control que tindrà en una sessió d'ús de l'aplicació amb diverses accions interactives. Si no pot controlar la seqüència d'interacció, se sentirà sense el control de l'aplicació, insatisfet, cosa que pot provocar el rebuig a l'aplicació sincera.

2.3.3 Assegurament de la informació

La informació amb què treballarem estarà en una base de dades. La introducció, modificació o esborrat de les dades es farà per mitjà d'una interfície. Una vegada aquesta s'ha fet servir i les dades estan estables en la base de dades, aquesta ha d'estar disponible per a quan els usuaris la necessitin i ha d'estar assegurada.

Per a aconseguir això cal establir un procés de manteniment i una utilització que no en comprometi la integritat. A més, es poden fer servir altres sistemes per a assegurar la disponibilitat de la informació, la qual cosa minimitzaria el risc de fallida dels sistemes informàtics. Aquests poden ser sistemes de reserva o d'emergència que facin còpies de seguretat de manera automàtica i regular.

Un altre mètode per a assegurar la integritat de la informació consisteix a establir sistemes per a evitar l'accés directe a les bases de dades per part d'usuaris no controlats, en cas de treballar amb una informació crítica per a les organitzacions. Aquests sistemes poden incloure un sistema de permisos per a perfils d'usuaris o unes contrasenyes. Amb aquest sistema, només un nombre determinat d'usuaris tindrà permís per a modificar les dades.

A més, també cal minimitzar els errors humans que es poden produir en gestionar aquesta informació els usuaris, manipulant l'aplicació o errors en introduir les dades.

Per a evitar això, es pot demanar la confirmació de les dades en introduir informació i comprovar que aquesta compleix les característiques establetes pels requisits, com ara el tipus de dades que es poden introduir en un *text box* (només nombres o només lletres o amb un format determinat).

Una altra manera d'assegurar la informació és fer revisar a l'usuari les dades que ha introduït o amb les quals ha tractat, fent servir un sistema de pregunta-resposta mitjançant quadres de diàleg essencials o, també, algun quadre de diàleg de tipus alerta. Si un usuari ha d'introduir dades mitjançant una interfície, abans de gravar la informació en la base de dades es podrà mostrar un quadre de diàleg amb les dades a l'usuari per a demanar la seva confirmació.

2.3.4 Específiques per a aplicacions multimèdia

En funció del tipus d'interfícies que es desenvolupin o del tipus d'aplicació o l'entorn per al qual es plantegi l'ús de l'aplicació, ens podem trobar amb la necessitat de fer servir alguns elements multimèdia específics com ara l'animació o el so.

Si parlem d'interfícies desenvolupades per a dispositius mòbils o per a entorns web, podrem fer servir alguns elements com el so en determinades accions o determinats elements creats amb Flash que aportin algunes animacions, bé per necessitat o bé per a dotar les interfícies d'un disseny més agradable.

Altres exemples d'utilització són la utilització d'alertes sonores per a determinats errors en la interacció amb la interfície, o la utilització d'animacions en l'ajuda a l'usuari, animacions molt més explícites que certes indicacions en format text.

Un altre cas és el que ens podem trobar en aplicacions per al control de la gestió d'una empresa, en què es cerqui dotar d'estadístiques, gràfiques i molts nombres els directius perquè puguin prendre decisions. Aquestes informacions, en comptes de mostrar-les amb uns gràfics normals en dues dimensions, es poden desenvolupar fent servir eines multimèdia o animacions que mostrin de manera més entenedora les dades que es volen consultar.

Per a resumir, podríem remarcar un dels objectius de la usabilitat i el disseny d'interfícies: “Una interfície d'usuari estarà ben dissenyada quan el programa es comporti exactament com l'usuari pensa que ho hauria de fer.”

Confecció d'informes. Documentació d'aplicacions

Eduard Latorre

Desenvolupament d'interfícies

Índex

Introducció	5
Resultats d'aprenentatge	7
1 Confecció d'informes	9
1.1 Informes incrustats i no incrustats a l'aplicació	9
1.1.1 Quina diferència hi ha entre els informes incrustats i els informes no incrustats?	10
1.1.2 Com funciona un informe incrustat?	10
1.1.3 Com funciona un informe no incrustat?	10
1.1.4 Informes incrustats o no incrustats?	11
1.2 Eines gràfiques integrades i externes a l'IDE	11
1.3 Estructura general. Seccions	12
1.3.1 Grups	13
1.4 Filtratge de dades	14
1.5 Numeració de línies, recomptes i totals	15
1.5.1 Grup i sentit de l'ordenació	15
1.6 Llibreries de generació d'informes. Classes, mètodes i atributs	16
1.6.1 PHPReports	17
1.6.2 Crystal Reports	24
1.7 Connexió amb les fonts de dades. Execució de consultes	25
1.8 Primer informe pas a pas	26
2 Documentació d'aplicacions	35
2.1 Importància de la documentació d'aplicacions	35
2.2 Fitxers d'ajuda. Formats	36
2.2.1 Microsoft Windows	36
2.2.2 Mac OS X	38
2.2.3 UNIX - GNU/Linux	38
2.3 Eines de generació d'ajudes	38
2.3.1 Microsoft HTML Help Workshop (HHW)	38
2.3.2 El DocBook	59
2.4 Taules de continguts, índexs i sistemes de cerca, entre d'altres	75
2.5 Tipus de manuals: manual d'usuari, guia de referència, guia ràpida i manual d'instal·lació, configuració i administració	76
2.5.1 El manual d'usuari	76
2.5.2 La guia de referència	76
2.5.3 La guia ràpida	76
2.5.4 El manual d'instal·lació	77
2.5.5 El manual de configuració	77
2.5.6 El manual d'administració	77

Introducció

Una de les funcions habituals d'un sistema d'informació és recollir dades. Aquestes dades, una vegada processades, s'han de poder comunicar d'una manera amigable a l'usuari. Depenent del tipus d'usuari, aquesta comunicació pot ser força diferent. Un usuari tècnic pot necessitar dades molt concretes, mentre que un executiu agrairà un resum segurament accompanyat d'un gràfic. La necessitat d'aquesta informació obereix a preguntes que l'usuari es fa o a la necessitat de fer un estudi o buscar respostes a problemes ocults darrere d'un munt de dades. Arribats a aquest punt, els informes són l'eina que ens permet donar forma al resultat de les nostres consultes.

En l'apartat “Confecció d'informes” veureu els diferents tipus d'informes que hi ha segons l'aplicació que els genera i els components principals que es poden trobar en cadascun. També treballareu amb eines tant privatives com lliures que permeten generar informes.

En l'apartat “Documentació d'aplicacions” tractarem un altre tema bàsic dins del desenvolupament de sistemes: la documentació d'aplicacions. A vegades és un tema que es deixa de banda, pensant que la nostra aplicació ja és prou intuïtiva, però segurament no serà suficient, sobretot depenent del tipus d'aplicació. En aquest apartat veureu algunes de les eines que us poden ajudar a generar documentació per a les vostres aplicacions, així com llenguatges de marques especialment pensats per generar documentació tècnica.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Crea informes avaluant i utilitzant eines gràfiques.

- Estableix l'estructura de l'informe
- Genera informes bàsics a partir d'una font de dades mitjançant assistents.
- Estableix filtres sobre els valors a presentar als informes.
- Inclou valors calculats, recomptes i totals.
- Inclou gràfics generats a partir de les dades.
- Utilitza eines per generar codi corresponent als informes d'un aplicació.
- Modifica el codi corresponent als informes.
- Desenvolupa una aplicació que inclou informes incrustats.

2. Documenta aplicacions seleccionant i utilitzant eines específiques.

- Identifica sistemes de generació d'ajudes.
- Genera ajudes als formats habituals.
- Genera ajudes sensibles al context.
- Documenta l'estructura de la informació persistent.
- Confecciona el manual d'usuari i la guia de referència.
- Confecciona els manuals d'instal·lació, configuració i administració.
- Confecciona tutorials.
- Prepara aplicacions per la seva distribució avaluant i analitzant eines específiques.

1. Confecció d'informes

Els informes són la cirereta del pastís real quan es tracta d'obtenir informació útil d'una base de dades.

Els informes proporcionen informació en un format que qualsevol persona pot entendre i, com que la majoria dels informes estan dissenyats per a impressió, donen a les seves dades un element de portabilitat.

La construcció d'informes és, sens dubte, un projecte més complex que la creació de consultes o formularis. La creació d'un informe és una tasca que requereix una mica de planificació.

Els informes solen incloure números de pàgina, dates i temps en què es van imprimir, agrupacions de registres i sumaris de dades.

Un dels propòsits de gran utilitat dels informes és mostrar resultats que no es poden representar directament en un format de taula de base de dades. Un informe ha de permetre veure els detalls importants a primera vista.

Un exemple d'informe que resumeix els resultats d'una base de dades de subhastes no solament mostraria els registres individuals, sinó també grups d'acord amb el millor postor. També mostraria el subtotal, la prima del comprador i el total de la suma per a cada guanyador. Aquest informe és un bon exemple de com un informe pot proporcionar informació que simplement no apareix per si sola en una taula d'una base de dades.

Les taules no presenten una forma resumida de les dades d'una col·lecció de registres, de manera que si no us voleu asseure amb una calculadora per esbrinar aquests detalls, heu de crear informes per obtenir aquest tipus d'informació de la base de dades.

1.1 Informes incrustats i no incrustats a l'aplicació

En planejar la creació d'una aplicació amb un entorn de desenvolupament integrat (IDE), una de les consideracions més importants és si s'utilitzen informes incrustats o no incrustats. Saber quins són els aspectes fonamentals que afecten la incrustació d'informes us ajudarà a triar la millor estructura per al projecte.

Entorn de desenvolupament integrat

Un IDE, entorn de desenvolupament integrat, és una aplicació de programari que proporciona ajudes als programadors informàtics per al desenvolupament de programari.

1.1.1 Quina diferència hi ha entre els informes incrustats i els informes no incrustats?

Un *informe incrustat* és un informe que s'ha importat a un projecte de l'IDE o que s'hi ha creat.

Quan un informe s'incrusta en el projecte, automàticament es genera una classe contenidora per a l'informe.

Un *informe no incrustat* és un informe extern al projecte de l'IDE.

Hi ha moltes maneres d'accendir a l'informe per carregar-lo en un model d'objectes a fi d'habilitar la interacció mitjançant programació, però l'informe sempre serà extern al projecte de l'IDE.

1.1.2 Com funciona un informe incrustat?

Quan l'informe s'importa al projecte o s'hi crea, es genera una classe contenidora que normalment té el mateix nom que l'informe. Aquesta classe conté o representa l'informe en el projecte. Quan passa això, tot el codi del projecte interactua amb la classe de l'informe que s'ha creat per representar-lo, en comptes de fer-ho amb l'arxiu mateix de l'informe original.

En compilar el projecte, tant l'informe com la seva classe contenidora s'incrusten en l'assemblatge, tal com passaria amb qualsevol altre recurs del projecte.

1.1.3 Com funciona un informe no incrustat?

L'accés a un informe no incrustat sempre s'obté externament. El programari hi pot accendir de diverses maneres. Per exemple:

- L'informe pot ser en una unitat de disc dur en una ruta de directori de fitxers.
- L'informe pot estar exposat com a servei web d'informes.

Mai s'importen informes no incrustats en el projecte i, per tant, mai es crea cap classe contenidora d'informe, a diferència dels informes incrustats. En comptes d'això, l'informe no incrustat es carrega en un dels models d'objectes en temps d'execució.

1.1.4 Informes incrustats o no incrustats?

Per simplificar la implementació del projecte, és preferible que utilitzeu informes incrustats. Haureu de treballar amb menys arxius i no us haureu de preocupar de si els informes estan col·locats correctament en la ruta de directori de fitxers definida. A més, aquesta solució és més segura, ja que els informes no s'exposen a modificacions.

Si esteu aprenent a desenvolupar i a implementar, us serà més fàcil incrustar els informes. Per exemple, si treballieu amb l'entorn de desenvolupament Microsoft Visual Studio, els informes, un cop incrustats, sempre apareixen com una classe del projecte, estan disponibles des de l'IntelliSense i es veuen en l'explorador d'objectes. No us heu de preocupar de si es mouen o s'eliminen en el directori de fitxers ni d'escriure la ruta correctament.

Els informes incrustats són més senzills i segurs, però comporten més feina.
No es poden modificar sense tornar a compilar tot el projecte.

Hi ha límits pel que fa a la mida que pot tenir un informe incrustat. Un informe molt gran es compila com un recurs incrustat.

Si els informes s'han de modificar regularment, és preferible que utilitzeu informes no incrustats. D'aquesta manera, hi podreu accedir i els podreu modificar més fàcilment, sense haver-vos de preocupar per la necessitat de tornar a compilar els afegitons cada vegada.

Els informes no incrustats tenen avantatges d'escalabilitat.

1.2 Eines gràfiques integrades i externes a l'IDE

Les eines gràfiques integrades a l'entorn de desenvolupament integrat permeten dissenyar gràficament les connexions de dades i disseny de l'informe. Quant a la base de dades, podeu seleccionar taules de diverses fonts de dades (incloent-hi fulls de càlcul i bases de dades) i vincular-les a la informació local del sistema de fitxers. Els camps d'aquestes taules es poden col·locar a la superfície de disseny de l'informe. També es poden utilitzar en les fòrmules personalitzades, que es col·loquen sobre la superfície de disseny. Les fòrmules es poden avaluar en diverses fases durant la generació d'informes segons les especificacions del desenvolupador.

Les eines externes a l'IDE afegeixen més funcionalitats a les eines integrades. A més d'ajudar a crear les definicions dels reports perquè després es puguin utilitzar, els poden generar directament en el format final.

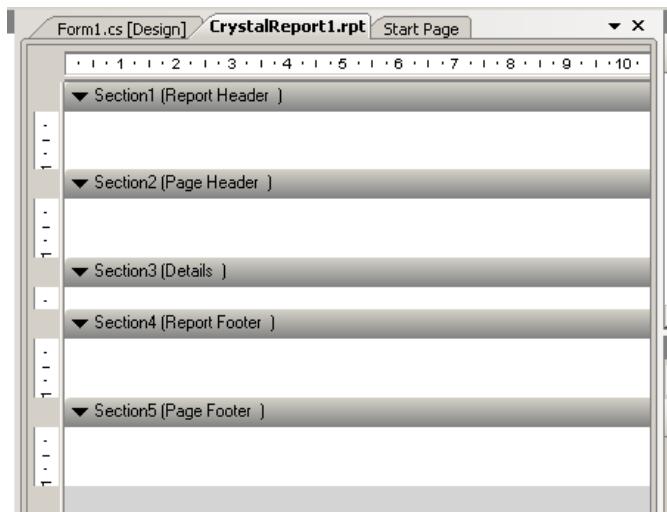
Microsoft Visual Studio IntelliSense

L'IntelliSense és la implementació de l'autoemplenament de noms, variables i paràmetres de funcions, entre d'altres, que el programador escriu en l'entorn integrat de desenvolupament Microsoft Visual Studio. Gràcies a l'IntelliSense, el programador no necessita memoritzar tantes dades i pot anar més de pressa a l'hora d'entrar informació mitjançant el teclat. Es tracta, doncs, d'una eina que ajuda a desenvolupar el programari, ja que accelera el procés.

1.3 Estructura general. Seccions

Els reports fan servir diverses seccions per controlar el disseny i les agrupacions dels informes. Aquestes seccions sempre s'organitzen en un ordre específic, com si es tractés d'un esquema. A vegades, una secció en particular pot estar buida. Per exemple, és possible que no es vulgui utilitzar el peu de l'informe si no s'hi ha de posar res.

FIGURA 1.1. Seccions per defecte del Crystal Reports per al Visual Studio



Tot seguit es mostren les seccions que podrien aparèixer en un informe:

- Capçalera de l'informe: apareix una vegada al començament d'un informe. Sovint inclou el títol de l'informe i la data en què es produueix. També es pot fer servir per inserir gràfics i taules creuades que facin referència a tot l'informe.
- Capçalera de pàgina: s'utilitza per afegir informació que apareixerà a la part superior de cada pàgina de l'informe. Es pot fer servir per escriure-hi l'encapçalament de columna, el número de pàgina o la data.
- Capçalera de grup: es fa servir, en el grup de registres d'un informe, per indicar on comença cada grup. Normalment inclou el valor del camp que s'utilitza per a l'agrupació. Si teniu múltiples nivells d'agrupació, cadascun d'aquests nivells tindrà el seu propi grup encapçalat.
- Detall: és la secció més interna. Constitueix el cos de l'informe i s'imprimeix una vegada per registre (fila). Mostra els detalls de cada registre. La majoria de les dades de l'informe apareixen en aquesta secció.
- Peu de pàgina: es fa servir per especificar la informació que es vol que aparegui a la part inferior de cada pàgina de l'informe, com el número de pàgina o la data.

- Peu de l'informe: conté la informació que es vol que aparegui només un cop a la part final de l'informe (per exemple, totals generals). També hi pot haver els gràfics i les taules creuades que inclouen dades relatives a tot l'informe. Sovint es deixa en blanc.

1.3.1 Grups

Els grups són una mena de filtres que permeten ordenar les dades segons el criteri escollit, és a dir, segons el grup específic seleccionat. Si s'afegeix un grup, un resum o un subtotal a l'informe, es pot treballar amb dues seccions més:

- Capçalera de grup: aquesta secció conté el camp de nom de grup. Se sol utilitzar per mostrar gràfics o taules creuades de dades específiques del grup. Només s'imprimeix una vegada al principi d'un grup.
- Peu de grup: aquesta secció inclou el valor de resum, si n'hi ha, i es pot usar per inserir gràfics, taules creuades, separadors o resums (per exemple, els totals del grup). S'imprimeix només una vegada al final d'un grup.

L'aplicació PHPReports, per exemple, té les seccions següents:

- Secció de document
- Secció de pàgina
- Secció de grup

S'ha de definir una secció de document, una secció de pàgina (hi pot haver un munt de pàgines, però només una secció de pàgina a configurar) i les seccions de grup que es vulguin.

Totes aquestes seccions recopilen informació sobre les dades de l'informe, com el nombre de línies i estadístiques sobre els camps, entre altres.

La secció de document recopila les estadístiques i les guarda fins al final de l'informe.

La secció de pàgina recopila informació fins al final de la pàgina i es restableix al començament de la pàgina nova.

La secció de grup manté la informació fins al final del grup. El grup constitueix un conjunt de dades definides per una expressió de ruptura, que podria ser qualsevol tipus de camp que figuri en el seu conjunt de dades.

Cada secció té el seu propi encapçalament i peu de pàgina. La secció de grup té una divisió més en què es mostra la informació de dades. Si hi ha més d'un grup (fins i tot quan només hi ha un informe simple cal afegir un altre grup per tractar la informació), com a mínim el grup més intern ha de mostrar les dades.

FIGURA 1.2. Seccions PHPReports

1.4 Filtratge de dades

En seleccionar un subconjunt dels registres totals, l'informe només es generarà sobre els registres que compleixin les condicions que s'hagin establert. Aquestes condicions es crearan sobre el tipus d'informació que vulgueu que aparegui en l'informe final, de manera que no s'hi inclouran tots els valors, sinó només un subconjunt. Per exemple, es poden crear filtres per seleccionar els elements següents:

- Només un grup específic de dades.
- Els registres d'un rang de dades seleccionat de la quantitat total de registres en la base de dades.
- Només els valors dels registres que entren en un rang de dates concret.

Suposem, per exemple, que voleu un informe que només mostri les dades de Barcelona. El repte és trobar la millor manera d'identificar els registres que provenen de Barcelona.

- Si la taula que s'utilitzarà en l'informe té un camp de ciutat, es poden seleccionar només els registres en què el valor del camp de ciutat és igual a Barcelona.
- Si la taula no té un camp de ciutat i voleu un informe que només mostri dades sobre Barcelona, hi pot haver una altra manera d'identificar aquestes dades.

- Si la taula té un camp de codi postal, el registre de selecció es pot basar en el rang de codis que s'apliquen a Barcelona (080XX).
- Si la taula té un camp de número de telèfon, el registre de selecció es pot basar en els codis de Barcelona (comencen per 93).

Com a regla general, és recomanable basar la selecció de registres en una sèrie de camps indexats per aconseguir un millor rendiment en la generació de l'informe.

1.5 Numeració de línies, recomptes i totals

Un dels propòsits principals de dividir les dades en grups és poder fer càlculs en cada grup de registres, en lloc de fer-los en tots els registres de l'informe.

Les dades, en agrupar-les, es classifiquen i es divideixen en grups significatius. Per exemple:

- En una llista de clients, un grup podria incloure tots els clients que tenen el mateix codi postal o que viuen en la mateixa regió.
- En un informe de vendes, un grup podria incloure totes les comandes fetes pel mateix client o per un representant comercial concret.

1.5.1 Grup i sentit de l'ordenació

Quan les dades s'agrupen, es poden classificar i la direcció dels valors es pot definir. La direcció fa referència a l'ordre en què es mostren els valors.

- Ascendent: de més petit a més gran (de l'1 al 9, de la A a la Z, de *False* a *True*). Els registres s'ordenen de manera ascendent i cada vegada que canvia el valor comença un grup nou.
- Descendent: de més gran a més petit (del 9 a l'1, de la Z a la A, de *True* a *False*). Els registres s'ordenen de manera descendent i cada vegada que canvia el valor comença un grup nou.
- Especificat: el defineix l'usuari. Cada registre es col·loca en el grup personalitzat que s'especifiqi i els registres de cada grup es deixen en l'ordre original (ascendent o descendent, segons l'especificació).
- Original: ordre en què es mostraven les dades quan es van guardar originalment a la base de dades. Es crea un grup nou quan el valor canvia en el camp de grup seleccionat.

Un dels grans avantatges d'agrupar els registres és que es poden generar subtotals de cada grup, a més dels totals generals corresponents a tot l'informe.

El programa Crystal Reports per al Visual Studio, per exemple, pot resumir les dades, ordenar-les, dividir-les en grups i, a continuació, resumir els valors de cada grup automàticament.

El programa inclou una sèrie d'opcions de resum. En funció del tipus de dades del camp que es vulgui resumir, pot fer el següent:

- Sumar els valors de cada grup.
- Comptar tots els valors o només els valors que són diferents l'un de l'altre.
- Determinar el valor màxim, el valor mínim, el valor mitjà o el valor enèsim més gran.
- Calcular fins a dos tipus de desviacions estàndard i variàncies.

Exemples:

- Un informe sobre la llista de clients: determinar el nombre de clients en cada regió. En el resum es compten els clients de manera diferent en cada grup d'estat.
- Un informe mensual: determinar la mitjana de comandes venudes cada mes. En el resum es calcula la mida de la mitjana de comandes de cada mes per grup.
- Un informe de vendes: determinar el total de vendes per representant comercial. En el resum se suma l'ordre dels imports de les vendes per a cadascun dels grups representatius.

1.6 Llibreries de generació d'informes. Classes, mètodes i atributs

En les llibreries de generació d'informes trobareu dues possibilitats, l'una per al programari de propietat i l'altra per al programari lliure.

Per tal d'exemplificar detalladament totes dues possibilitats, s'han escollit dues aplicacions: el Crystal Reports per al programari de propietat Microsoft Windows i el PHPReports, desenvolupada per Eustáquio Rangel, “TaQ”, per al programari lliure GNU/Linux.

A continuació es mostraran els detalls de cadascuna de les aplicacions.

1.6.1 PHPReports

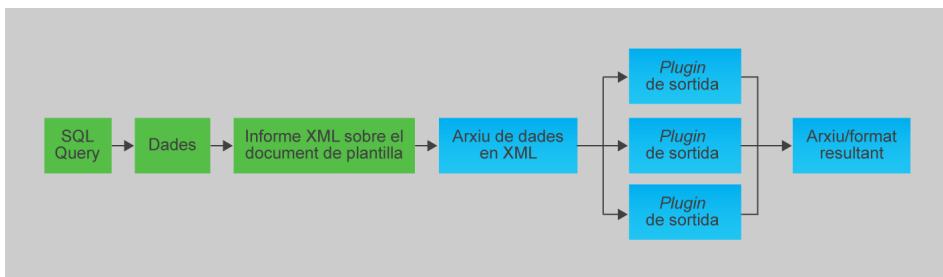
Entre les diverses possibilitats que ofereix el programari lliure hi ha l'aplicació PHPReports per a l'entorn GNU/Linux. El PHPReports és un programa dissenyat per facilitar la creació d'informes mitjançant plantilles d'arxius XML que s'executen en un entorn d'un servidor web.

Per poder fer proves, per exemple, necessitareu un servidor web Apache (<http://www.apache.org>) i la versió 5 del llenguatge PHP, el qual s'haurà d'executar en aquest servidor (<http://www.php.net>) compilat amb les extensions XSL (<http://www.php.net/manual/en/ref.xsl.php>).

La figura 1.3 mostra el funcionament del PHPReports. En els quadres de color verd es produeix l'entrada de dades i en els de color groc, l'informe de sortida.

Els connectors (*plug-in*) de sortida constitueixen maneres de convertir l'arxiu transformat de l'informe XML en l'HTML (o un altre format). En el cas del connector *BOOKMARK*, s'especifica una manera de processar l'arxiu per mitjà de marcs *FRAMES*.

FIGURA 1.3. Funcionament del PHPReports



En el primer quadre groc, l'informe ja està processat amb totes les dades a l'interior (en un format XML especial amb elements reduïts a fi que la mida sigui al més petita possible). Seguidament, la informació arriba a un connector de sortida. En aquest connector, la informació es tracta perquè tingui l'aspecte que es vol.

Fins i tot l'informe més senzill funciona d'aquesta manera, és a dir, el connector de sortida per defecte el transforma en una simple pàgina HTML.

El codi PHP

Ara veureu el codi PHP necessari per generar l'informe, en un arxiu anomenat vendes.php:

```

1 <?php
2 // la línia de sota només és necessària si la ruta d'inclusió include path
3 no s'estableix en php.ini
4 ini set ("include path",ini get("include path").":/usr/lib/phpreports/");
5
6 include once "PHPReportMaker.php";
7
8 $oRpt = new PHPReportMaker();
  
```

```

9
10    $oRpt->setUser("usuari");
11    $oRpt->setPassword("*****");
12    $oRpt->setXML("vendes.xml");
13
14    $oRpt->run();
15 ?>

```

Estructura de les plantilles XML

El codi següent és un exemple de plantilla d'un informe en format XML.

```

1  <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2  <!DOCTYPE REPORT SYSTEM "PHPReport.dtd">
3
4  <REPORT MARGINWIDTH="5" MARGINHEIGHT="5">
5      <TITLE>Informe de Vendas</TITLE>
6      <BACKGROUND_COLOR>#FFFFFF</BACKGROUND_COLOR>
7      <CSS>phpreports/css/phpreports.css</CSS>
8      <SQL>select * from vendes</SQL>
9      <INTERFACE>mysql</INTERFACE>
10     <CONNECTION>localhost</CONNECTION>
11     <DATABASE>phpreports</DATABASE>
12     <NO DATA MSG>No hi ha dades. Reviseu la Consulta.</NO DATA MSG>
13
14     <PAGE BORDER="1" SIZE="25" CELLSPACING="0" CELLPADDING="5" WIDTH="500">
15         <HEADER>
16             <ROW>
17                 <COL COLSPAN="4" CELLCLASS="PAGE_LAYER" TEXTCLASS="BOLD">Tenda </
18                     COL>
19             </ROW>
20             <ROW>
21                 <COL COLSPAN="4" CELLCLASS="PAGE_LAYER" TEXTCLASS="BOLD">Informe de
22 Vendes</COL>
23             </ROW>
24         </HEADER>
25         <FOOTER>
26             <ROW>
27                 <COL ALIGN="RIGHT" COLSPAN="3" CELLCLASS="PAGE_LAYER">total plana</
28                     COL>
29                 <COL ALIGN="LEFT" NUMBERFORMATEX="2" CELLCLASS="PAGE_LAYER"
30 TEXTCLASS="BOLD" TYPE="EXPRESSION">$this->getSum("valor")</COL>
31             </ROW>
32         </FOOTER>
33     <GROUPS>
34         <GROUP NAME="citygroup" EXPRESSION="ciutat">
35             <HEADER>
36                 <ROW>
37                     <COL CELLCLASS="GROUP_LAYER">ciutat:</COL>
38                     <COL CELLCLASS="GROUP_LAYER" TEXTCLASS="BOLD" TYPE="EXPRESSION"
39 COLSPAN="3">$this->getValue("ciutat")</COL>
40                 </ROW>
41                 <ROW>
42                     <COL CELLCLASS="GROUP_LAYER">nom</COL>
43                     <COL CELLCLASS="GROUP_LAYER">tipus</COL>
44                     <COL CELLCLASS="GROUP_LAYER">producte</COL>
45                     <COL CELLCLASS="GROUP_LAYER">Eur</COL>
46                 </ROW>
47             </HEADER>
48             <FOOTER>
49                 <ROW>
50                     <COL ALIGN="RIGHT" COLSPAN="3" CELLCLASS="GROUP_LAYER">total</
51                         COL>
52                     <COL ALIGN="LEFT" CELLCLASS="GROUP_LAYER" TEXTCLASS="BOLD"
53 NUMBERFORMATEX="2" TYPE="EXPRESSION">$this->getSum("valor")</COL>
54                 </ROW>

```

```

53     </FOOTER>
54     <FIELDS>
55         <ROW>
56             <COL TYPE="FIELD" CELLCLASSEVEN="GROUP_LAYER"
57             CELLCLASSODD="YELLOW_ROW">nom</COL>
58             <COL TYPE="FIELD" CELLCLASSEVEN="GROUP_LAYER"
59             CELLCLASSODD="YELLOW_ROW">tipus</COL>
60             <COL TYPE="FIELD" CELLCLASSEVEN="GROUP_LAYER"
61             CELLCLASSODD="YELLOW_ROW">producte</COL>
62             <COL TYPE="FIELD" CELLCLASSEVEN="GROUP_LAYER"
63             CELLCLASSODD="YELLOW_ROW">valor</COL>
64         </ROW>
65     </FIELDS>
66     </GROUP>
67 </GROUPS>
68 </REPORT>

```

Element REPORT

Fa referència a l'objecte principal de l'informe. És l'element que conté els altres.

Elements

- **TITLE** El títol de l'informe.
- **PATH** La ruta en què hi ha les classes de PHPReports.
- **BACKGROUND COLOR** El color de fons en format hexadecimal.
- **BACKGROUND IMAGE** Una adreça URL vàlida per una imatge de fons.
- **CSS** Una adreça URL vàlida per l'arxiu CSS.
- **SQL** La consulta SQL per cridar la base de dades.
- **USER** L'identificador d'usuari per obrir la connexió de base de dades.
- **PASSWORD** La contrasenya per obrir la connexió de base de dades.
- **CONNECTION** El nom de la connexió de base de dades a utilitzar (per exemple, el nom de l'entrada TNS de l'Oracle).
- **INTERFACE** La interfície de base de dades.
- **NO DATA MSG** Un missatge personalitzat quan no es troben dades.
- **DEBUG** S'utilitza per mostrar missatges de depuració per mitjà de l'informe.
- **FORM** Formulari utilitzat per crear un formulari HTML amb l'informe.
- **DOCUMENT** L'element *DOCUMENT*.
- **PAGE** L'element *PAGE*.
- **GROUPS** L'element *GROUPS*.

Atributs

- **MARGINWIDTH** El marge d'ample, en píxels.

- MARGINHEIGHT El marge d'alçària, en píxels.
- MAX_ROW_BUFFER Nombre màxim de files per emmagatzemar en la memòria abans d'escriure en un arxiu.

El valor per defecte és de 2.500 files.

Element CSS

Els arxius CSS que podeu utilitzar a l'informe.

Atributs

- MEDIA El tipus de medi que fa servir aquest arxiu CSS. Podeu especificar, per exemple, *screen* (pantalla) o *print* (impressió).

Element FORM

Es refereix a un element utilitzat per crear formularis HTML en l'informe.

Elements

- FORM_NAME El nom del formulari.
- FORM_METHOD El mètode de formulari.
- FORM_ACTION L'acció del formulari.

Element DOCUMENT

L'element en què els valors de tot l'informe s'emmagatzemen. És el controlador de valors globals.

Elements

- HEADER L'element capçalera.
- FOOTER L'element peu de pàgina.

Element HEADER

L'element amb les files utilitzat a la part superior de les pàgines.

Elements

- ROW Un o més elements *ROW*.

Element FOOTER

L'element amb les files utilitzat a la part inferior de les pàgines.

Elements

- ROW Un o més elements *ROW*.

Element ROW

L'element amb el *COL*.

Elements

- COL L'element *COL*.

Element COL

L'element amb els camps i els valors d'expressió.

Elements

- LINK L'element *LINK*.
- BOOKMARK L'element *BOOKMARK*.
- XHTML L'element *XHTML*.

Atributs

- TYPE Pot ser *REGULAR*, *EXPRESSION*, *RAW EXPRESSION* o *FIELD*.
- NUMBERFORMAT Utilitza l'*sprintf* (C, PHP, etc.) per donar format a valors numèrics.
- NUMBERFORMATEX Format per als valors decimals.
- CELLCLASS Classe CSS per donar format a les *COL*.
- CELLCLASSEVEN Classe CSS per donar format a les *COL* parells.
- CELLCLASSODD Classe CSS per donar format a les *COL* senars.
- CELLCLASSEXPRESSION Expressió per calcular la classe CSS per donar format a les *COL*.
- TEXTCLASS Classe CSS per donar format al text dins de la *COL*.
- ROWSPAN El nombre de *ROW* (files) per abastar.
- COLSPAN El nombre de *COL* (columnes) per abastar.
- WIDTH L'ample de *COL*, en píxels.
- HEIGHT L'alçària de *COL*, en píxels.
- ALIGN L'alignació horitzontal de *COL* (valors d'ús d'HTML).
- VALIGN L'alignació vertical de *COL* (valors d'ús d'HTML).
- SUPPRESS Si el valor *COL* s'imprimirà o no si és igual a l'anterior (*boolean*).
- ONCLICK Javascript perquè es dispari quan el ratolí faci clic a la *COL*.
- ONMOUSEOVER Javascript perquè es dispari quan el ratolí sigui sobre la *COL*.
- ONMOUSEOUT Javascript perquè es dispari quan el ratolí marxi de la *COL*.

Element PAGE

L'element de pàgina. Hi ha les característiques de format de la pàgina i es mostren els valors.

Elements

- HEADER L'element *HEADER* (capçalera).
- FOOTER L'element *FOOTER* (peu de pàgina).

Atributs

- SIZE La mida de la pàgina, en files.
- WIDTH L'ample de la pàgina, en píxels.
- HEIGHT L'alçària de la pàgina, en píxels.
- CELLPADDING El farciment (*padding*) entre les *COL*.
- CELLSPACING Els espais entre les *COL*.
- BORDER La mida de la frontera entre les *COL*, per defecte, 0.
- ALIGN L'alignació de la pàgina.

Element GROUPS

L'element que conté els elements *GROUPS*.

Elements

- GROUP L'element *GROUP*.

Element GROUP

Aquí és on s'agrupen les dades, els trencaments, etc.

Elements

- HEADER L'element *HEADER* (capçalera).
- FIELDS L'element *FIELDS*.
- FOOTER L'element *FOOTER* (peu de pàgina).
- GROUP Un altre element *GROUP* que hi ha.

Atributs

- NAME El nom del grup.
- EXPRESSION L'expressió en què el grup es divideix.

- PAGEBREAK *TRUE* per inserir un salt de pàgina després que el grup es divideixi.
- REPRINT HEADER ON PAGEBREAK *TRUE* si necessita tornar a imprimir la capçalera en el salt de pàgina.
- RESET SUPPRESS ON PAGEBREAK *TRUE* per establir tots els valors suprimits quan es fan els salts de pàgina.

Element FIELDS

Només és un contenidor per a les files que contenen protocols del tipus *TYPE=“FIELDS”*.

Elements

- ROW Alguns elements *ROW*.

Element LINK

Element per enllaçar amb un altre URL.

Atributs

- TYPE *STATIC, DYNAMIC o EXPRESSION*.
- TARGET El marc de destinació en què s'obrirà l'URL.
- TITLE El títol del text que apareix en la informació de l'enllaç.

Element BOOKMARK

Element per fer un marcador a la *COL* actual.

Atributs

- TYPE *STATIC, DYNAMIC o EXPRESSION*.
- CELLCLASS Classe CSS per donar format a la *COL*.
- TEXTCLASS Classe CSS per donar format al text dins la *COL*.

Element IMG

Element per inserir una imatge dins d'un arxiu *COL*. Poseu l'URL de la imatge entre i </> *IMG*

Atributs

- WIDTH Ample de la imatge, en píxels.
- HEIGHT Alçària de la imatge, en píxels.
- BORDER Frontera vora de la imatge, en píxels.
- ALT Text alternatiu *ALT*.

Funcions per utilitzar dins de l'element COL

Les funcions d'aquesta secció es poden utilitzar dins dels elements *COL* en el fitxer XML de disseny de l'informe. També es poden fer servir des del codi PHP, però només seran útils quan l'informe funcioni.

- `getValue(<field name>)` Retorna el valor actual del camp en el grup actual. El valor de la columna de la fila actual s'està processant.
- `getSum(<field name>)` Torna la suma dels camps en el grup actual.
- `getMax(<field name>)` Torna el valor màxim del camp en el grup actual.
- `getMin(<field name>)` Torna el valor mínim dels camps en el grup actual.
- `getAvg(<field name>)` Torna la mitjana dels camps en el grup actual. És el màxim / el recompte de files.
- `getRowCount()` Torna el nombre de registres actuals del grup.
- `getRowNum()` Torna el número de la fila actual.
- `getPageNum()` Torna el número de la pàgina actual.
- `getParameter(<name>)` Torna el valor del paràmetre amb el nom especificat.

L'SDK (*software development kit*) és un conjunt d'eines de desenvolupament que permeten crear aplicacions.

1.6.2 Crystal Reports

Pel que fa al Windows, el Crystal Reports és l'estàndard d'elaboració d'informes del Visual Studio. S'inclou en totes les còpies del Visual Studio Professional i s'integra directament en l'entorn de desenvolupament.

El Crystal Reports permet crear informes complexos i professionals en un programa basat en GUI. Els informes es poden connectar a bases de dades, a dades de servidors intermediaris i a conjunts de resultats personalitzats (per exemple, un DataSet). Els auxiliars del dissenyador de la GUI permeten establir els criteris de format, agrupament i gràfics, entre altres.

El model de classes del Crystal Reports per al Visual Studio inclou un SDK extens. Per interactuar amb l'informe mitjançant programació en temps d'execució es pot fer servir un dels quatre models d'objectes possibles:

- `CrystalReportViewer`, el model d'objectes més senzill.
- `ReportDocument`, el model d'objectes més complet.

Els models que es mostren a continuació necessiten una llicència actualitzada:

- ReportClientDocument, el model d'objectes més complet.
- InfoObject, un model d'objectes molt eficaç per a la programació i la configuració d'informes en el marc del Crystal Reports Server o el BusinessObjects Enterprise.

En un informe incrustat, la classe contenidora d'un informe s'estén des del ReportDocument de la classe base comuna. Hereta totes les propietats i els mètodes del ReportDocument. El ReportDocument és la classe arrel del model d'objectes ReportDocument.

En un informe no incrustat com, per exemple, un informe ubicat en una ruta de directori de fitxers, el Crystal Reports dins del model d'objectes ReportDocument utilitzaria el mètode ReportDocument.Load () per carregar l'informe en el model d'objectes ReportDocument.

Els llocs web en Crystal Reports per a Visual Studio 2005 i Crystal Reports Basic per a Visual Studio 2008 només treballen amb informes no incrustats.

1.7 Connexió amb les fonts de dades. Execució de consultes

Un dels primers passos per crear un informe sempre és identificar un origen de dades. Algunes fonts de dades són el que es consideraria bases de dades tradicionals, com el Microsoft SQL Server, l'Oracle, l'Informix, el Sybase, el MySQL, l'IBM DB2 i el Microsoft Access. Altres fonts de dades són formes més abstractes de dades com fitxers de registre, correu electrònic, XML, fulls de càlcul i bases de dades multidimensionals (OLAP).

Per tal de donar suport a totes les fonts de dades i, alhora, preservar una experiència d'usuari, el suport per a cada font de dades es proporciona per mitjà d'un controlador de base de dades. Un controlador de base de dades actua com una passarel·la entre l'aplicació i un tipus específic de base de dades o tecnologia d'accés a dades.

Pel que fa als informes basats en bases de dades, es necessitaria una manera d'especificar quins camps de les taules han d'aparèixer, quins filtres s'han d'aplicar i com s'han d'ordenar les dades. El llenguatge SQL proporciona aquesta possibilitat. La sintaxi d'aquest llenguatge es basa en un sistema d'enviament de les instruccions SQL al servidor de base de dades SQL. Cada directriu és una petició per fer una operació de base de dades: crear un arxiu de base de dades, incorporar taules i camps a una base de dades, afegir registres a taules o recuperar dades de bases de dades. El servidor SQL analitza la instrucció SQL i efectua l'operació sol·licitada. Per exemple, si la declaració és una petició de dades, el servidor recull les dades i les torna al programa client perquè les tracti.

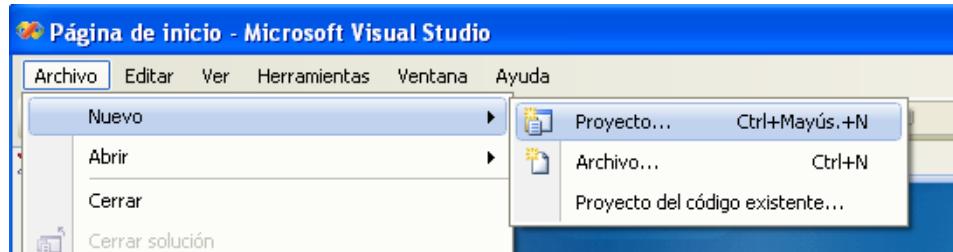
1.8 Primer informe pas a pas

En l'entorn Windows treballarem amb l'SQL Server i el Visual Studio.

Ara creareu un informe nou mitjançant el Visual Studio 2008, el Visual C# i el Crystal Reports des de la taula de productes en la base de dades. La taula de productes té tres camps (*producte_id*, *producte_nom*, *producte_preu*), els quals mostrareu amb tota la informació de la taula a l'informe.

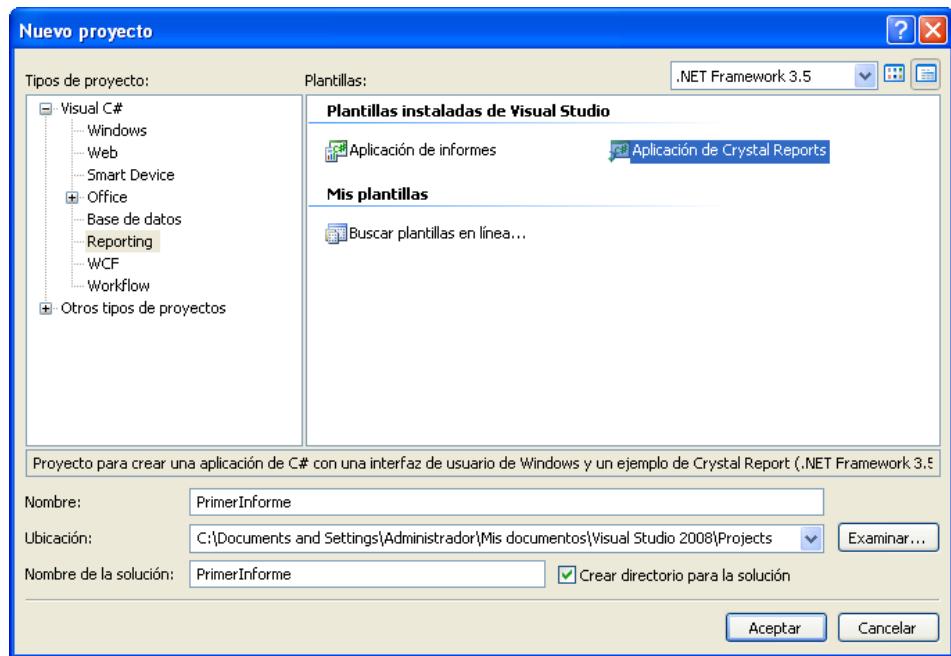
Obriu el Visual Studio .NET i seleccioneu *Nuevo > Proyecto* (vegeu la figura 1.4).

FIGURA 1.4. Creació d'un projecte nou

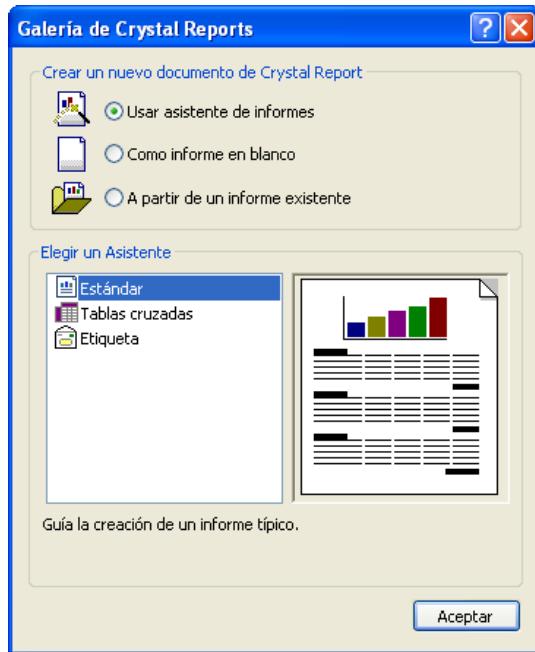


Seleccioneu un projecte nou de *Reporting* dins el Visual CSharp i la plantilla *Aplicación de Crystal Reports* (vegeu la figura 1.5).

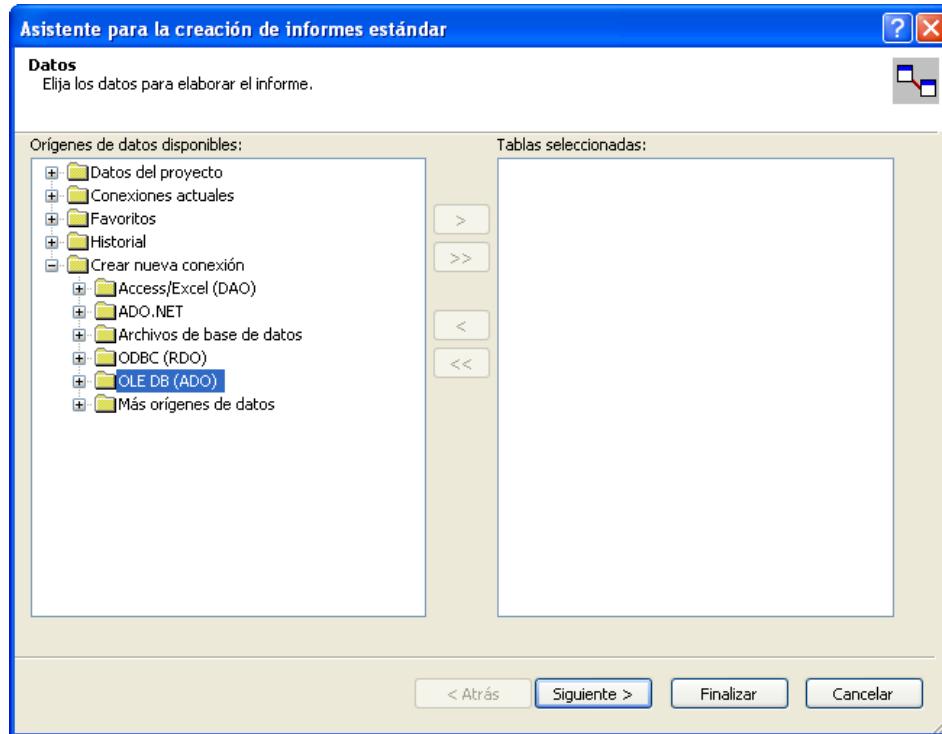
FIGURA 1.5. Selecció d'un projecte del Crystal Report



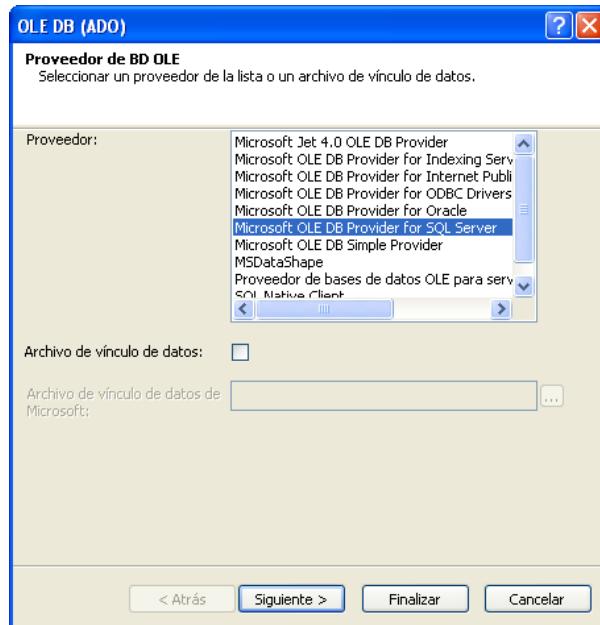
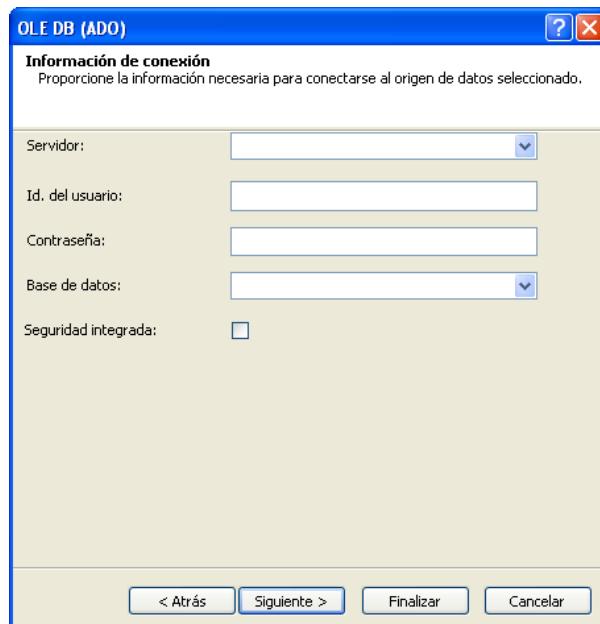
En la finestra *Galería de Crystal Reports*, seleccioneu *Usar asistente de informes* i l'auxiliar *Estándar*, que correspon a la configuració predeterminada, i feu *Aceptar* (vegeu la figura 1.6).

FIGURA 1.6. Galeria d'informes del Crystal Reports

Seleccioneu la connexió adequada per a la base de dades. En aquest cas, *OLE DB* per a l'SQL Server (vegeu la figura 1.7 i figura 1.8).

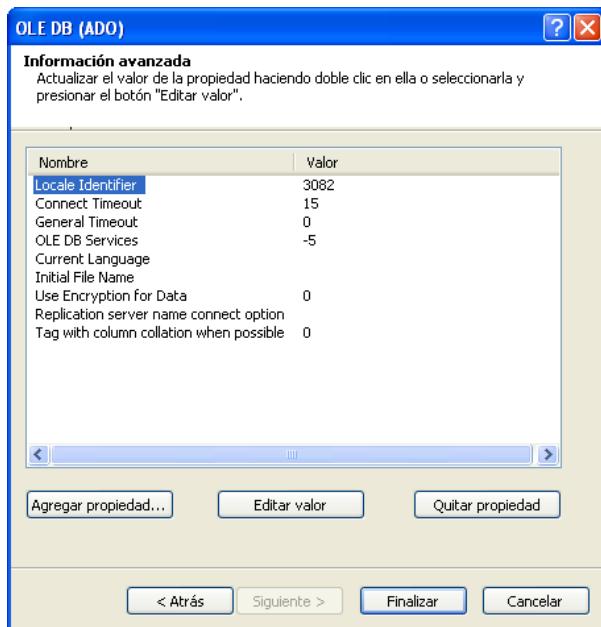
FIGURA 1.7. Selecció de la connexió

Escolliu *OLE DB (ADO)* per crear una connexió nova.

FIGURA 1.8. Proveïdor OLE DB (ADO)**FIGURA 1.9.** Configuració de la connexió OLE DB

Seleccioneu *Microsoft OLE DB Provider for SQL Server*. La pantalla següent és la d'autenticació de l'SQL Server per connectar-se a la base de dades. Trieu el nom del servidor de l'SQL Server, escriviu l'identificador d'usuari, entreu la contrasenya i seleccioneu el nom de la base de dades (vegeu la figura 1.9).

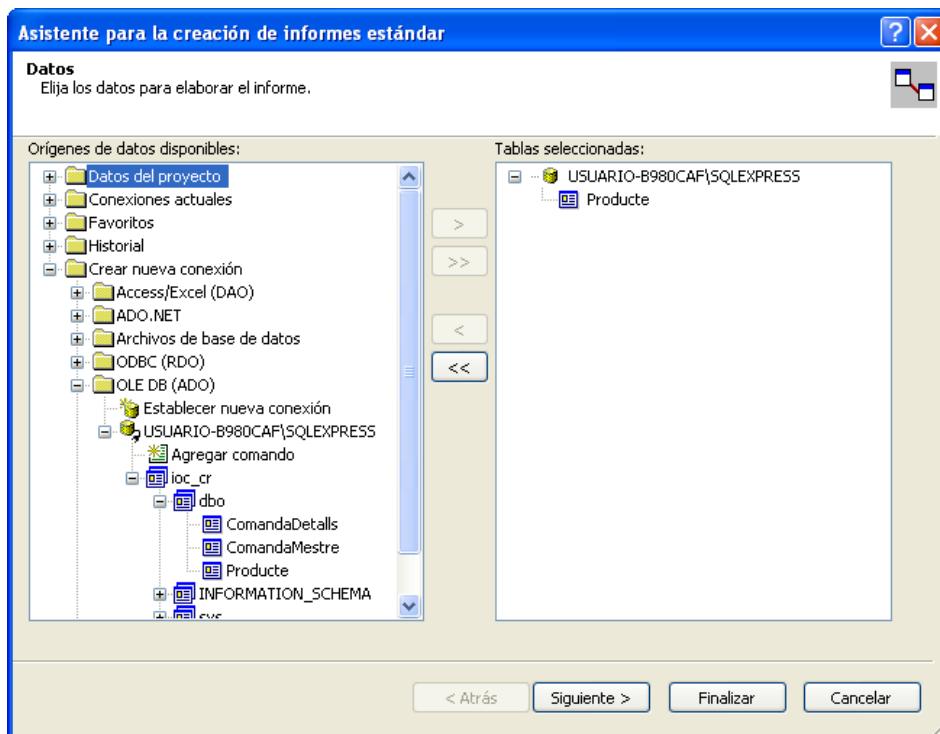
Si cliqueu a *Siguiente*, la pantalla us mostrarà valors de les propietats d'OLE DB. Deixeu-los tal com estan i feu *Finalizar* (vegeu la figura 1.10).

FIGURA 1.10. Propietats d'OLE DB

Després de clicar a *Finalizar*, en la finestra següent apareixerà el nom del servidor dins l'arbre d'OLE DB. Seleccioneu el nom de la base de dades (*ioc_cr*) i marqueu els requadres. A continuació, podreu veure totes les taules de la base de dades.

En les taules de la llista, cal que feu doble clic a la taula *Producte*. A continuació, aquesta taula apareixerà a la llista de la dreta (vegeu la figura 1.11).

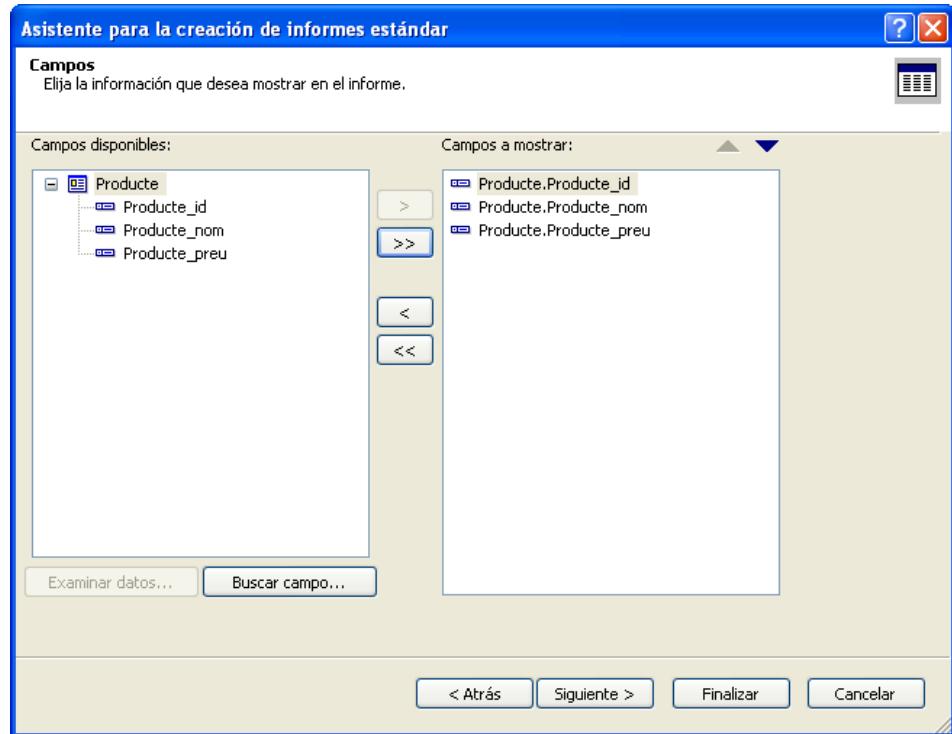
Feu clic a *Siguiente*.

FIGURA 1.11. Taules seleccionades

Seleccioneu tots els camps de la taula *Producte* a la llista de la dreta (vegeu la

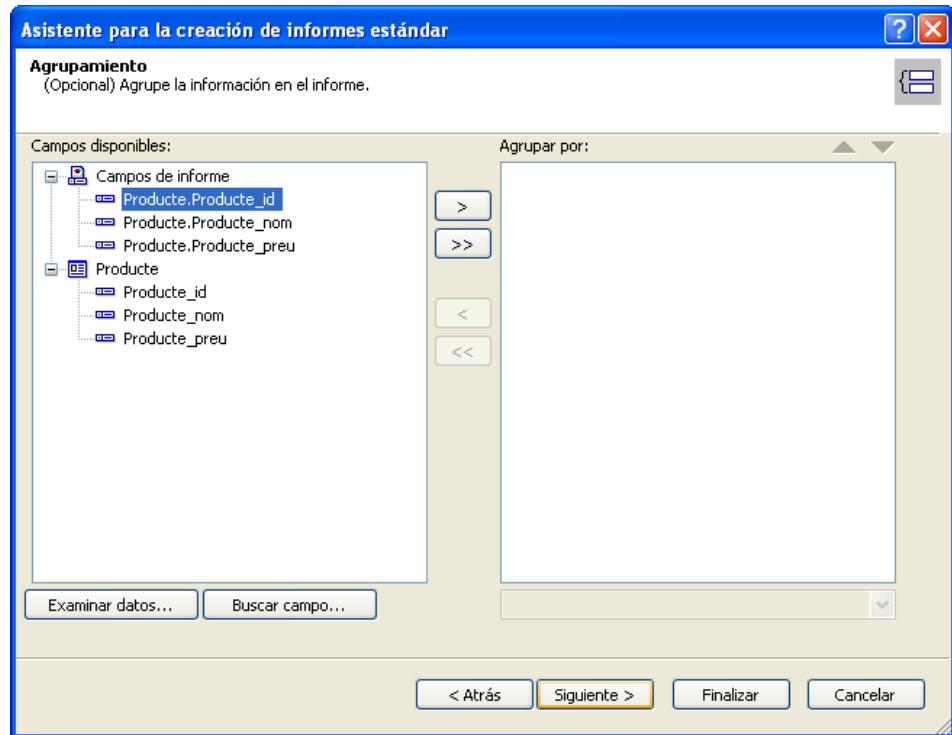
figura 1.12).

FIGURA 1.12. Camps a mostrar



En aquest exemple no farem cap agrupació (vegeu la figura 1.13). Feu clic a *Siguiente*.

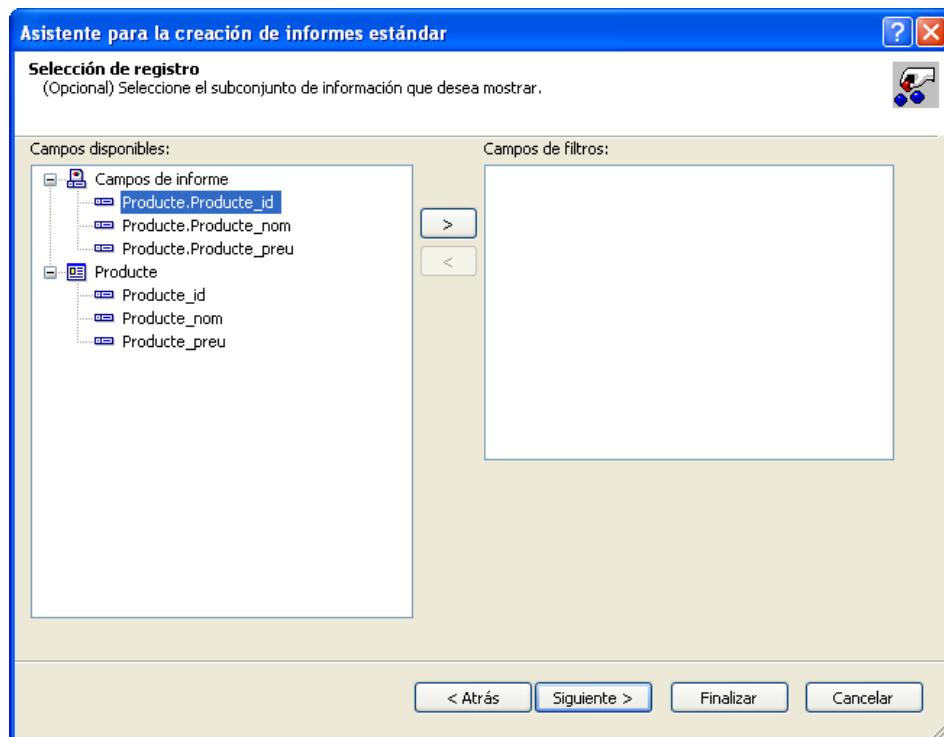
FIGURA 1.13. Agrupament de dades



En aquest exemple no aplicarem cap filtre (vegeu la figura 1.14). Feu clic a

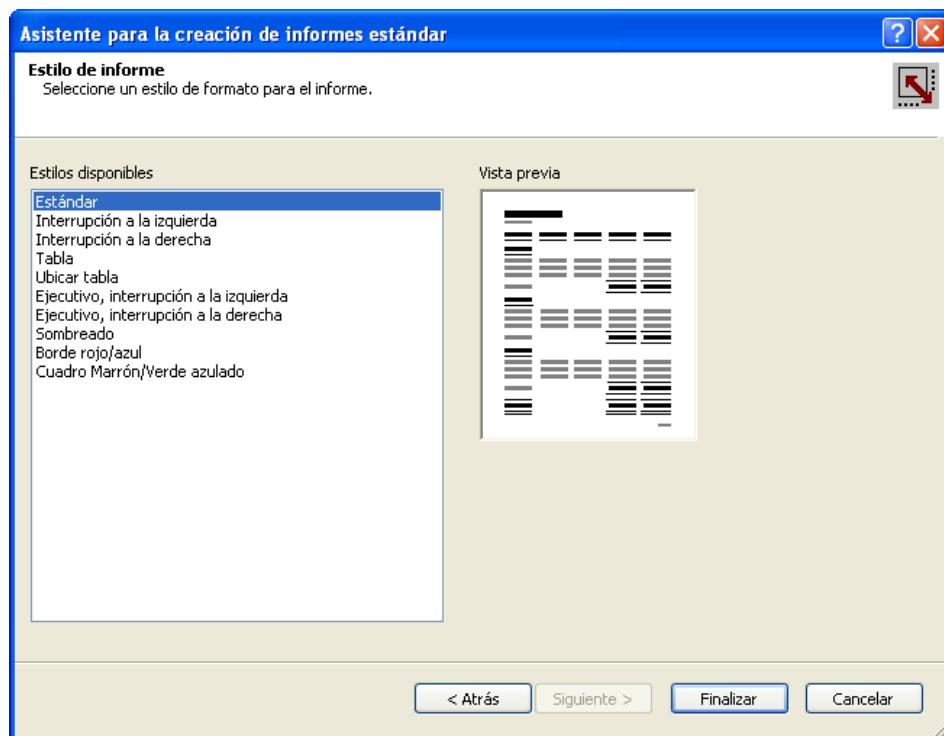
Siguiente.

FIGURA 1.14. Filtre de dades



Escolliu l'estil *Estándar*, que es mostra per defecte (vegeu la figura 1.15). Feu clic a *Siguiente*.

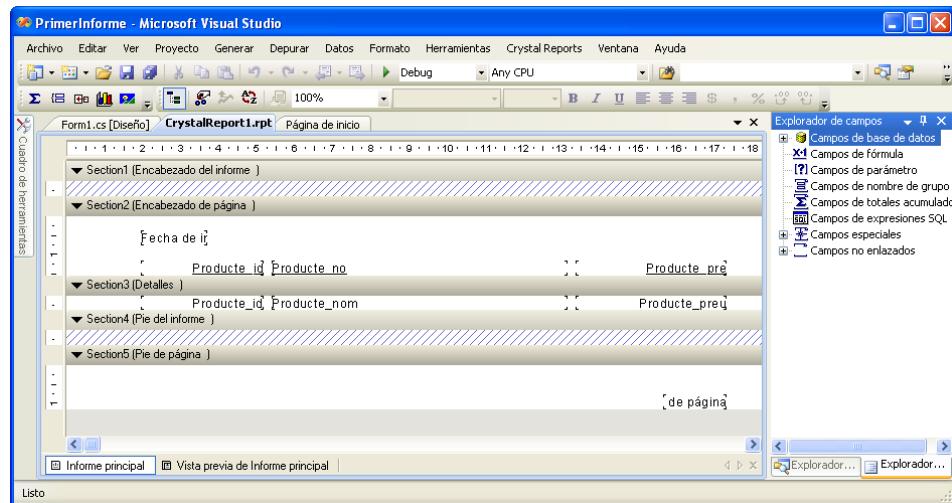
FIGURA 1.15. Selecció de l'estil de l'informe



L'auxiliar ja ha finalitzat. Podeu veure la presentació de disseny de l'informe

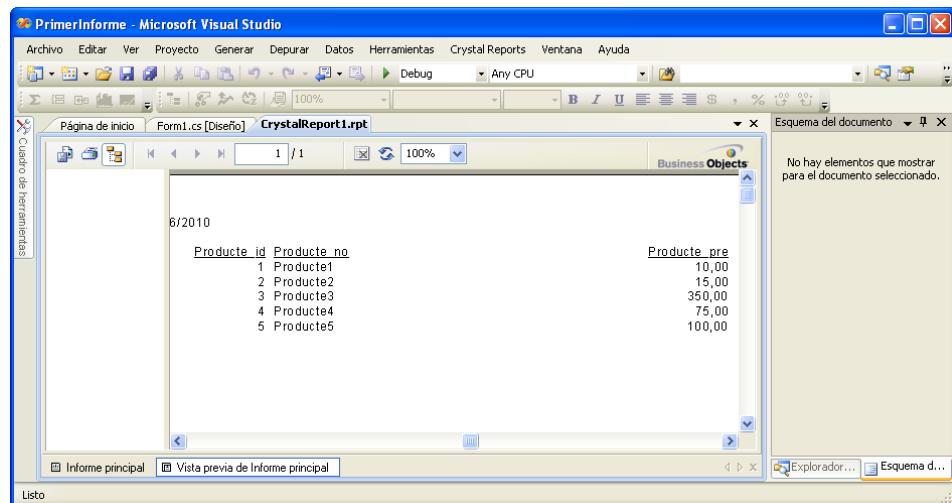
(vegeu la figura 1.16).

FIGURA 1.16. Presentació de disseny de l'informe

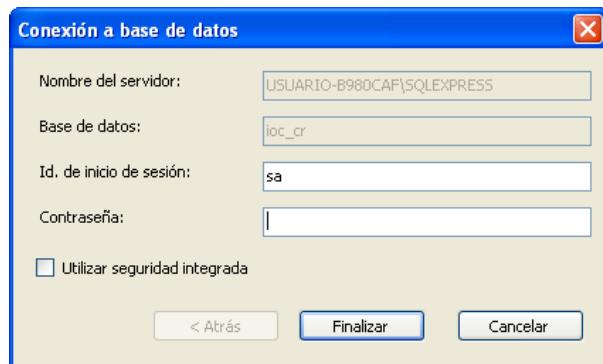


Seleccioneu *Vista previa de Informe principal*, a la part inferior de la presentació de disseny, per veure una presentació prèvia amb les dades de la base de dades (vegeu la figura 1.17).

FIGURA 1.17. Presentació prèvia de l'informe



Si executeu el projecte apareixerà una finestra que us demanarà la contrasenya per connectar-se a la base de dades (vegeu la figura 1.18). Es pot configurar perquè no ho demani.

FIGURA 1.18. Connexió a una base de dades

Un cop entrada la contrasenya, apareixerà la finestra amb l'informe final (vegeu la figura 1.19).

FIGURA 1.19. Informe final

2. Documentació d'aplicacions

La documentació d'aplicacions constitueix el conjunt d'informació que ens diu què fan les aplicacions, com ho fan i per a qui ho fan.

Una *aplicació* és un programa informàtic dissenyat perquè un usuari pugui desenvolupar una tasca.

La documentació consisteix en un material que explica les característiques tècniques i les funcionalitats que presenta l'aplicació. Almenys ha de proporcionar informació sobre la manera com s'instal·la, la manera correcta de fer-la funcionar i la manera com es manté.

2.1 Importància de la documentació d'aplicacions

La documentació de les aplicacions és un aspecte summament important, tant pel que fa al desenvolupament com pel que fa al manteniment de l'aplicació. Hi ha molts programadors que no li donen gaire importància i no s'adonen que perdren la possibilitat de reutilitzar part del programa en altres aplicacions, sense necessitat de conèixer el codi fil per randa.

La importància de la documentació es podria comparar amb la importància de tenir una pòlissa d'assegurança; quan tot va bé no es té la precaució de confirmar si la pòlissa d'assegurança és vigent o no.

La documentació d'una aplicació comença en el moment que es comença a construir l'aplicació i finalitza just abans de lliurar-la al client. Així mateix, la documentació que es lliura al client ha de coincidir amb la versió final dels programes que componen l'aplicació.

L'estil de redacció dels manuals de documentació ha de complir aquests requisits:

- Ser concret.
- Ser precís.
- Definir els termes utilitzats.
- Utilitzar paràgrafs curts.
- Utilitzar títols i subtítols.
- Utilitzar formes actives en lloc de passives.
- Evitar frases llargues que presentin fets diferents.
- Evitar referir-se a una informació només amb el número de referència.

2.2 Fitxers d'ajuda. Formats

Els fitxers d'ajuda contenen la informació per ajudar l'usuari a fer servir l'aplicació.

Ara veureu els formats principals del sistemes operatius següents:

- Microsoft Windows
- Mac OS X
- UNIX - Gnu/Linux

2.2.1 Microsoft Windows

Dins del sistema operatiu Microsoft Windows, veureu el primer format, el Microsoft Compiled HTML Help, que es va crear l'any 1997. També veureu l'evolució d'aquest format, el Microsoft Assistance Markup Language, que incorpora més garanties de seguretat.

Microsoft Compiled HTML Help

El Microsoft Compiled HTML Help és un format desenvolupat per Microsoft que es va llançar el 1997 per als fitxers d'ajuda del sistema. Es va introduir per primera vegada amb el llançament del Windows 98 i encara s'utilitza en les plataformes Windows XP, Windows Vista i Windows 7. El 2002 es va anunciar que hi havia riscos de seguretat associats a aquest format, per la qual cosa es va decidir no continuar utilitzant-lo i crear-ne un de nou, el Microsoft Assistance Markup Language.

Per crear els fitxers d'ajuda HTML (extensió *chm*) s'utilitzen eines d'autoria d'ajuda. Microsoft proporciona una eina, el Help Workshop, que es pot descarregar gratuïtament. A partir dels arxius amb el text de l'ajuda, un compilador de línia d'ordres (*hhc.exe*) els compila i crea un arxiu CHM. També hi ha una sèrie d'eines d'autor de tercers disponibles.

Un fitxer amb extensió *chm* constitueix un conjunt de documents d'HTML i altres dades, com imatges i JavaScript, comprimit en un sol arxiu. Els arxius CHM contenen un nombre de característiques molt útils que ajuden l'usuari a trobar el que busca:

- Taula de contingut
- Índex de paraules clau
- Funcionalitat de cerca amb text complet

Microsoft Assistance Markup Language

El Microsoft Assistance Markup Language (conegit normalment com a MAML) és un llenguatge de marques basat en XML. Aquest llenguatge el va desenvolupar l'equip d'assistència a l'usuari de Microsoft a fi d'ofrir ajuda per al sistema operatiu Microsoft Windows Vista. Algunes de les característiques del MAML han estat disponibles en el .NET Framework 2, però s'hi han afegit més opcions amb el llançament del .NET Framework 3.

L'aspecte més significatiu del MAML és que desplaça la producció de l'assistència a l'usuari amb el concepte *autoria estructurada* (semblant al DocBook). Els documents i els elements que els constitueixen es defineixen pel context en què es troben. L'èmfasi es posa en el contingut i en les tasques que un usuari fa amb l'ordinador, no en les característiques del programari.

L'estructura del MAML es divideix en segments relacionats amb un tipus de contingut:

- Resolució de problemes conceptuals
- Preguntes freqüents
- Glossari
- Procediment de referència
- Contingut reutilitzable
- Tasca
- Tutorial

Quan apareix un tema, es produeixen tres nivells de transformació:

- Estructura
- Presentació
- Representació

La transformació estructural conté contingut reutilitzable. La lògica condicional s'aplica per determinar l'estructura que ha de tenir el contingut quan es mostra i el contingut del text mateix.

La transformació de la presentació permet que el contingut creat en MAML es pugui transformar en molts formats diferents, incloent-hi el DHTML, l'XAML, el format de text enriquit i altre material imprès.

La transformació de la representació aplica fulls d'estil i mostra el contingut final als usuaris.

.NET Framework

El .NET Framework és un component de programari per al sistema operatiu Microsoft Windows. Gestiona l'execució de les aplicacions creades amb aquest component.

El DocBook és una aplicació que s'utilitza per crear, principalment, documentació tècnica.

El DHTML (*dynamic hypertext markup language*, llenguatge d'etiquetatge d'hypertext dinàmic) descriu l'art de crear pàgines web dinàmiques i interactives. El DHTML combina HTML, JavaScript, HTML DOM i CSS.

El format de text enriquit (*rich text format*, RTF) és un format de fitxers de propietat de Microsoft.

2.2.2 Mac OS X

L'ajuda d'Apple proporciona assistència a l'usuari basada en el format de fitxers HTML.

L'ajuda d'Apple gestiona i mostra els llibres ajuda. Un llibre d'ajuda és la col·lecció de fitxers HTML que constitueixen l'ajuda a l'usuari d'un producte de programari, a més d'un índex de l'ajuda dels arxius indexats generats. En crear un llibre d'ajuda, els usuaris poden accedir a l'ajuda de la seva interfície d'usuari i veure-ho directament en el visualitzador d'ajuda d'Apple.

2.2.3 UNIX - GNU/Linux

L'ajuda en els entorns Linux està lligada a l'ajuda que ofereixen les ordres de sistema de l'UNIX. Per mostrar l'ajuda executarem:

¹ `man <nom ordre>`

Cada pàgina és un document independent.

La majoria d'aplicacions gràfiques sobre l'UNIX (sobretot les construïdes amb els entorns de desenvolupament GNOME i KDE) proporcionen a l'usuari final la documentació en HTML i inclouen visors d'HTML incrustats, com el *Yelp*, el qual s'utilitza per a la lectura de l'ajuda dins de l'aplicació.

2.3 Eines de generació d'ajudes

A continuació, veureu dues eines gratuïtes per generar fitxers d'ajuda. L'una és el Microsoft HTML Help Workshop, per al Microsoft Windows, que únicament genera fitxers amb extensió *chm*, i l'altra és el DocBook, que pot generar un rang més ampli de fitxers d'ajuda.

L'ActiveX constitueix un conjunt de tecnologies desenvolupades per Microsoft per compartir informació entre diferents aplicacions.

2.3.1 Microsoft HTML Help Workshop (HHW)

L'ajuda HTML (.chm) és un estàndard de l'ajuda del Windows. Combina la funcionalitat del programari del Windows WinHelp amb la flexibilitat de l'HTML (*hypertext markup language*) i el poder dels controls ActiveX.

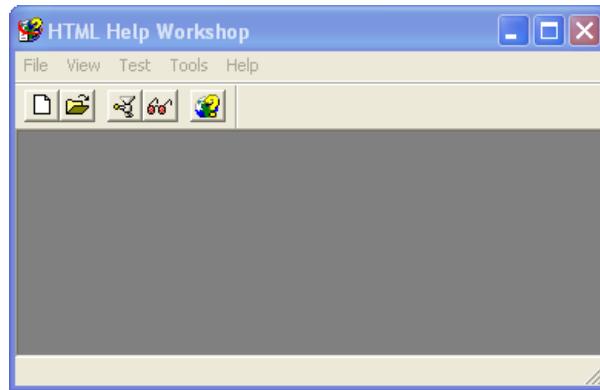
Creació d'un projecte d'ajuda amb l'HHW

Creareu un arxiu de projecte, el qual permetrà compilar la llista de fitxers (HTML i gràfics) que formaran el sistema d'ajuda. Modificareu l'aspecte de l'arxiu i en definireu unes quantes característiques especials.

Per crear un projecte nou, cal que seguiu els passos següents:

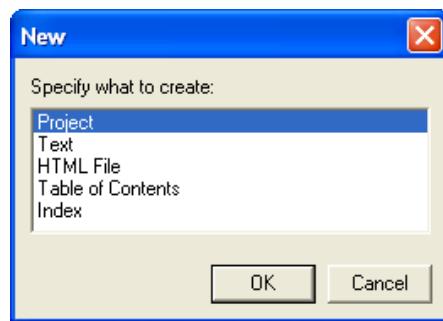
1. Executeu l'aplicació Microsoft HTML Help Workshop. Apareixerà la finestra que es pot veure en la figura 2.1.

FIGURA 2.1. Finestra principal del Microsoft HTML Help Workshop



2. Seleccioneu l'opció *File – New*. Apareixerà el quadre de diàleg que es mostra en la figura 2.2.

FIGURA 2.2. Quadre de diàleg New



3. L'opció *Project* està seleccionada per defecte. Feu clic al botó *OK* i s'obrirà el quadre de diàleg *New Project* (vegeu la figura 2.3).
4. Com que creeu un projecte nou, no marqueu l'opció i feu *Siguiente*. A continuació, apareixerà el quadre de diàleg *New Project – Destination* (vegeu la figura 2.4).
5. Escriviu la ruta i el nom del fitxer en el quadre de text d'entrada. Cliqueu a *Siguiente*. Apareixerà el quadre de diàleg *New Project – Existing Files*.

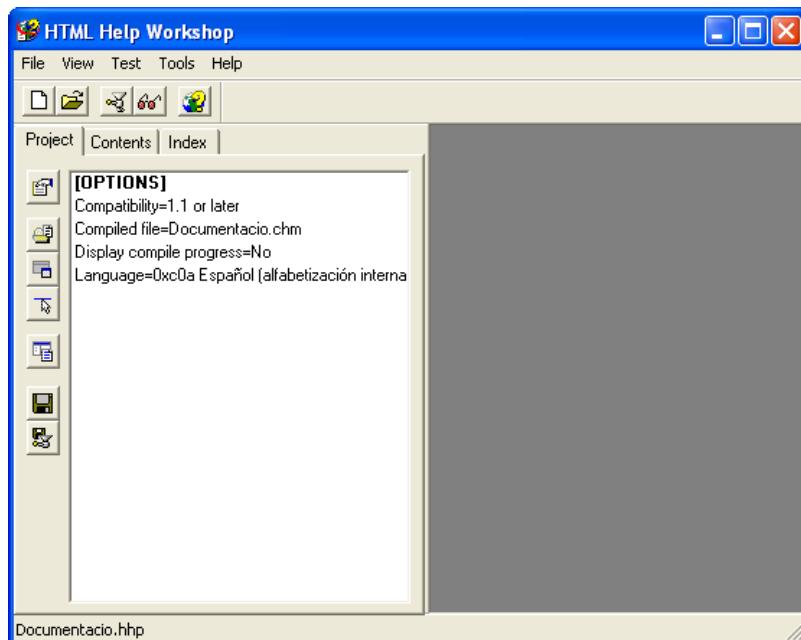
FIGURA 2.3. Quadre de diàleg New Project**FIGURA 2.4.** New Project – Destination

6. Com que creeu un projecte des de zero i no teniu cap fitxer creat, feu *Siguiente*. Apareixerà l'últim quadre de diàleg, *New Project - Finish* (vegeu la figura 2.5 i figura 2.6).

FIGURA 2.5. Quadre de diàleg New Project - Existing Files

FIGURA 2.6. Quadre de diàleg New Project - Finish

7. En prémer el botó *Finalizar*, apareixerà el quadre de diàleg de l'aplicació, *HTML Help Workshop* (vegeu la figura 2.7).

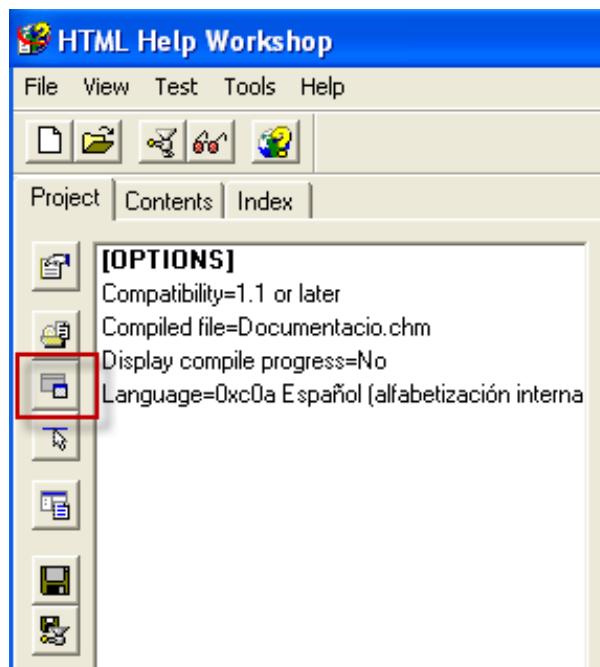
FIGURA 2.7. HTML Help Workshop

Personalitzar la finestra d'ajuda

El pas següent consisteix a establir les característiques que tindrà la finestra d'ajuda.

1. Feu clic a *Add/Modify Window Definitions* (vegeu la figura 2.8).

Llavors apareixerà el quadre de diàleg *Add a New Window Type* (figura 2.9).

FIGURA 2.8. Quadre de diàleg Add

Modify Window Definitions

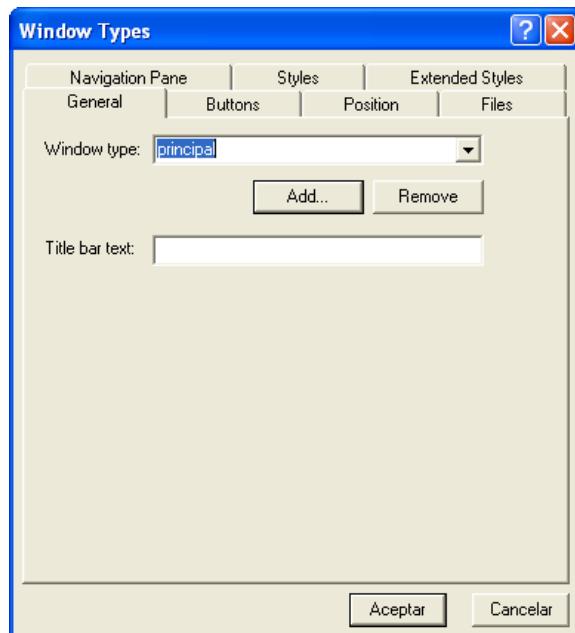
FIGURA 2.9. Quadre de diàleg Add a New Window Type

2. Entreu el nom (per exemple, *principal*) i feu *OK*. Apareixerà el quadre de diàleg *Window Types* (figura 2.10).

En la finestra es mostren les pestanyes següents:

- *Buttons*: especifica els botons que apareixeran en la finestra d'ajuda.
- *Position*: especifica la posició i la mida de la finestra d'ajuda.
- *Files*: especifica els fitxers que s'utilitzaran en les diverses parts de l'ajuda.
- *Navigation Pane*: especifica el panell de navegació de la finestra d'ajuda.
- *Styles* i *Extended Styles*: especificuen els estils que s'aplicaran a la finestra d'ajuda.

3. Especifiqueu les característiques que vulgueu a la finestra i feu *Aceptar* per acabar.

FIGURA 2.10. Quadre de diàleg Window Types

Establir un grup de paraules d'aturada (stop words)

Si se selecciona la cerca de text complet, la llista de paraules d'aturada impedeix que les paraules comunes, com i/o, es mostrin en els resultats de la cerca. Incloure una llista d'aturada redueix la mida de l'índex de cerca de text complet, cosa que significa que l'arxiu que en resultarà també serà més petit.

Les paraules d'aturada es guarden en un fitxer de text amb extensió *stp*.

Treballar amb temes (topics)

Un *tema* (*topic*) és un “tros” d'informació que es guarda en un fitxer en un format HTML i amb el qual es treballa per crear l'ajuda.

Per crear un tema nou, cal seguir els passos següents:

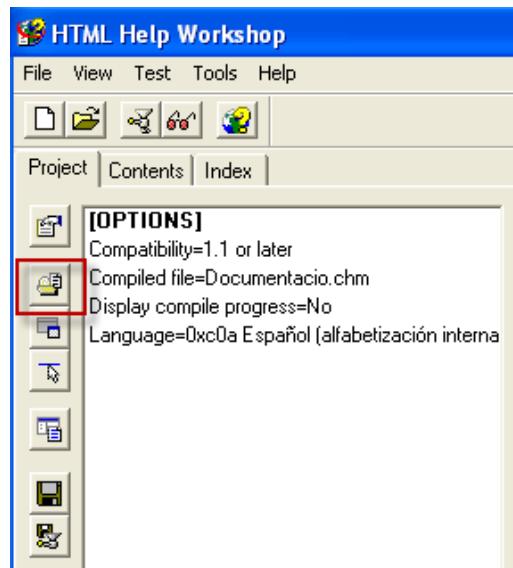
1. Seleccioneu l'opció *File > New > HTML file*.
2. Entreu el títol del tema i feu *OK*. Apareixerà una finestra nova a la dreta.
3. Modifiqueu el *DOCTYPE*, perquè el que es genera no és correcte. Poseu-hi l'HTML 4.01 Transitional DOCTYPE:

```

1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2   "http://www.w3.org/TR/html4/loose.dtd">

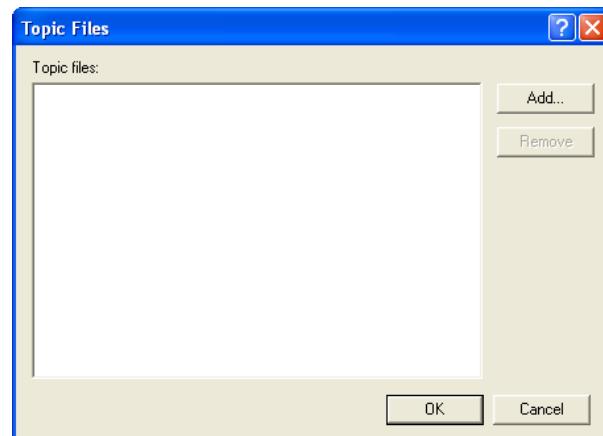
```

4. Entreu el contingut del tema en format HTML.
5. Deseu el fitxer i poseu-li un nom.
6. Incorporeu el fitxer creat com a tema dins l'ajuda. Feu clic a la icona *Add/Remove Topic Files* (figura 2.11).

FIGURA 2.11. Ícones Add

Remove Topic Files

Apareixerà el quadre de diàleg *Topic Files* (figura 2.12).

FIGURA 2.12. Quadre de diàleg Topic Files

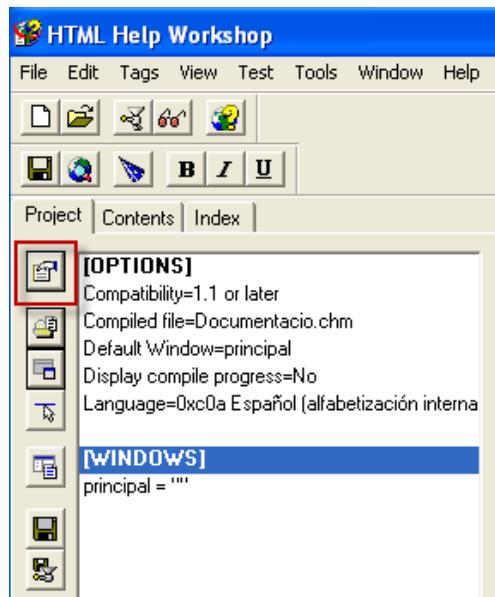
7. Premeu el botó *Add* i seleccioneu el fitxer acabat de crear com a tema nou per a l'ajuda.

Establir el tema d'inici

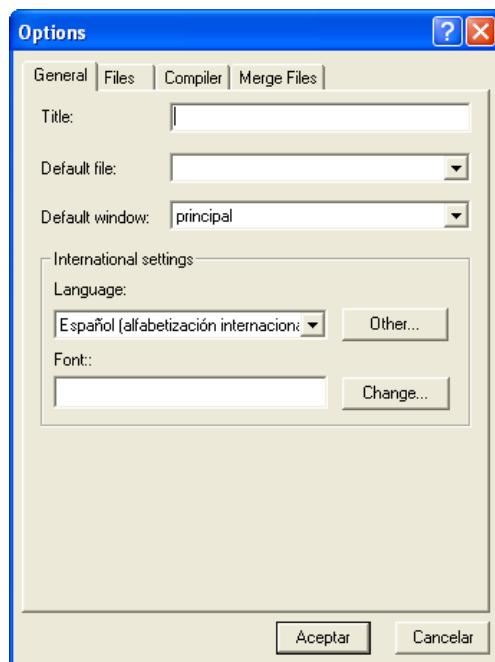
El tema d'inici apareix quan els usuaris obren l'arxiu d'ajuda. És la primera informació que els usuaris veuen dins la finestra d'ajuda.

Per definir el tema d'inici, cal seguir els passos següents:

1. Feu clic a *Change Project Options* (vegeu la figura 2.13).

FIGURA 2.13. Change Project Options

2. S'obrirà el quadre de diàleg d'opcions del projecte, *Options*, i apareixerà la pestanya *General* (figura 2.14).

FIGURA 2.14. :figure:Figure33:

3. Seleccioneu, en el menú desplegable *Default file*, el fitxer amb el tema d'inici que vulgueu.
4. Feu *Aceptar*.

Navegació entre temes dins l'ajuda

Com que treballem amb fitxers en format HTML, la navegació entre els diferents temes relacionats es fa per mitjà d'enllaços (*hyperlinks*).

Un enllaç a HTML està format de la manera següent:

1 ` Text explicatiu de l'enllaç`

El protocol de transferència de fitxers (*file transfer protocol*, FTP) és un estàndard per enviar o rebre fitxers entre ordinadors qualsevol.

Segons el contingut de l'adreça URL de destinació, es pot indicar una de les possibilitats següents:

- Pàgina a Internet: URL destinació = <http://ioc.xtec.cat/>
- Enllaç a un tema: URL destinació = Tema.htm"
- Enllaç a un tema dins un projecte d'ajuda diferent: URL destinació = projecte2.chm::/Tema2.htm
- FTP: URL destinació = <ftp://ftp.xtec.cat>
- Marca dins del text del tema: URL destinació = Tema.htm#Marca
- Adreça correu = mailto:nom@ioc.cat

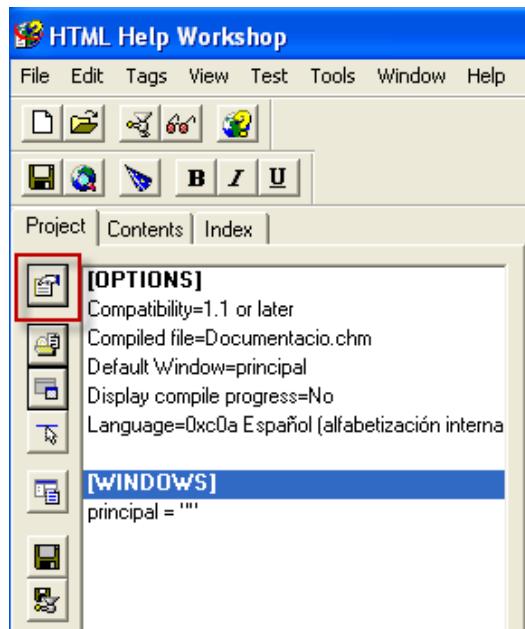
Compilar el projecte

El procés de compilació agafa tots els fitxers que heu creat en el projecte, els comprimeix i genera un fitxer de sortida en format .chm que conté el sistema d'ajuda muntat.

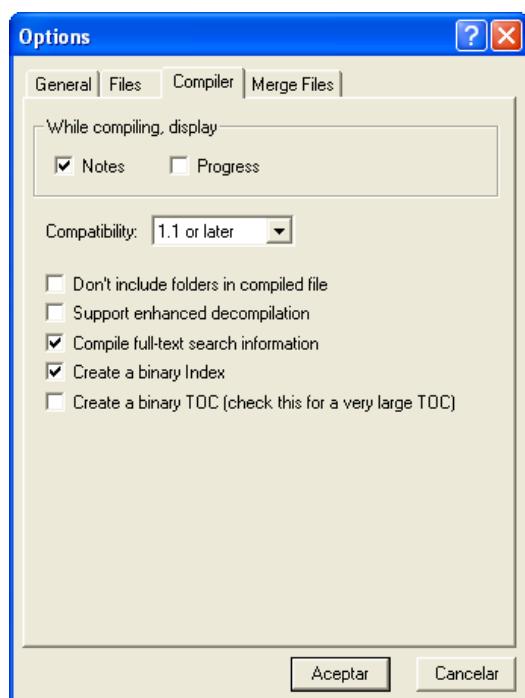
Abans de compilar el projecte, cal que comproveu les opcions de compilació i que tots els fitxers estan desats en el disc.

Per comprovar les opcions de compilació, cal que seguiu els passos següents:

1. Feu clic a *Change Project Options* (figura 2.15).

FIGURA 2.15. Change Project Options

2. S'obrirà el quadre de diàleg d'opcions del projecte, *Options*. Seleccioneu la pestanya *Compiler* (figura 2.16).

FIGURA 2.16

En aquesta pestanya es poden especificar les opcions de compilació. Les més significatives són les següents:

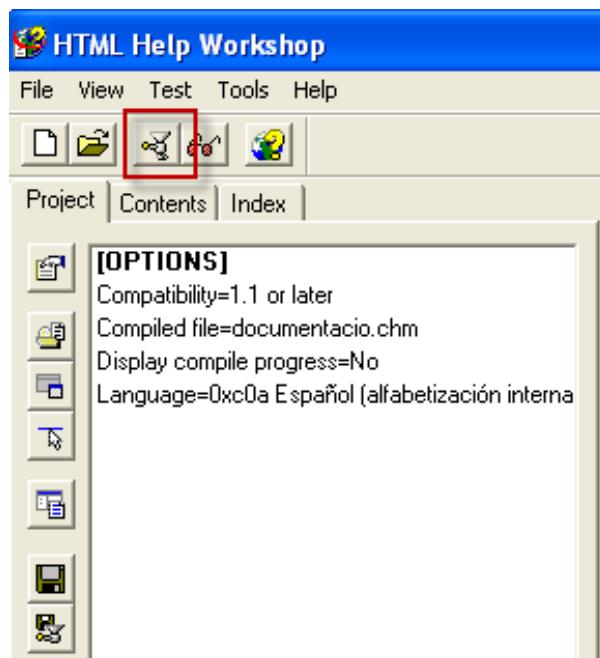
- *While compiling, display*: permet especificar la sortida de l'informe de compilació. Se li pot demanar informació sobre el fitxer que compila o bé que faci un recull dels possibles errors, enllaços trencats, etc.

- *Compatibility*: permet definir la compatibilitat amb la versió 1.0.
- *Don't include folders in compiled file*: permet definir que tots els fitxers s'ubiquin en l'arrel de l'ajuda. Si s'utilitza aquesta opció, s'ha de vigilar que els noms dels fitxers que hi hagi en les diverses carpetes no siguin iguals.
- *Compile full-text search information*: permet especificar que la cerca dins l'ajuda es faci a tot el text.

Per compilar el projecte, cal seguir els passos següents:

1. Cliqueu, en la barra principal, a la icona *Compile HTML File* (figura 2.17).

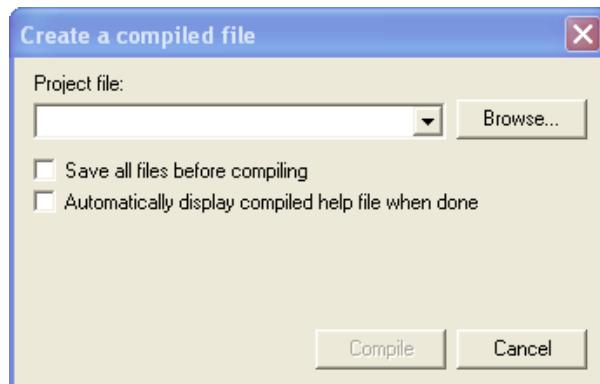
FIGURA 2.17. Ícone Compile HTML File



2. Apareixerà la finestra prèvia a la compilació.

Es pot marcar que tots els fitxers es desin al disc abans de la compilació, *Save all files before compiling*, i també que després de la compilació es mostri l'ajuda resultant, *Automatically display compiled help file when done* (vegeu la figura 2.18).

FIGURA 2.18. Create a compiled file



3. Feu *Compile*.

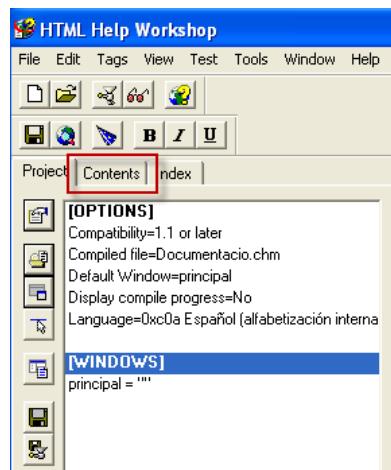
Taula de continguts

La taula de continguts constitueix una manera de navegar per la finestra d'ajuda. Presenta la informació d'una manera jeràrquica per mitjà d'encapçalaments i pàgines.

Creació d'una taula de continguts

1. Per accedir a la taula de continguts, cal que premeu la pestanya *Contents* (figura 2.19).

FIGURA 2.19. Pestanya de la taula de continguts

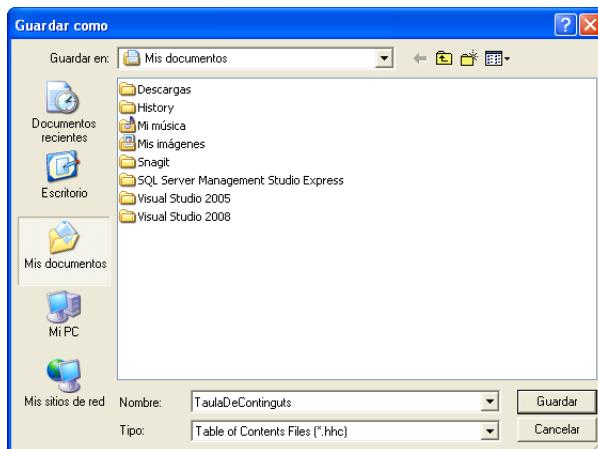


2. Com que la taula de continguts encara no està definida, apareixerà un quadre de diàleg que us preguntarà si voleu crear una taula de continguts o bé si en voleu obrir una que ja estigui creada. Per crear-ne una, haureu de mantenir el valor seleccionat per defecte (vegeu la figura 2.20).

FIGURA 2.20. Taula de continguts



3. En fer *OK*, apareixerà el quadre de diàleg *Guardar como*, en què haureu de posar un nom al fitxer (figura 2.21).

FIGURA 2.21. Desar el fitxer TaulaDeContinguts

A continuació, haureu de definir la taula de continguts, la qual consta d'encapçalaments i pàgines de contingut.

Creació d'un encapçalament

1. Feu clic a la icona *Insert a heading* per inserir un encapçalament (figura 2.22).

Apareixerà el quadre de diàleg d'entrada nova en la taula de continguts, *Table of Contents Entry* (figura 2.23).

2. Entreu el títol i feu clic al botó *Aceptar*. L'encapçalament es mostra a la part superior de la pestanya *Contents*.

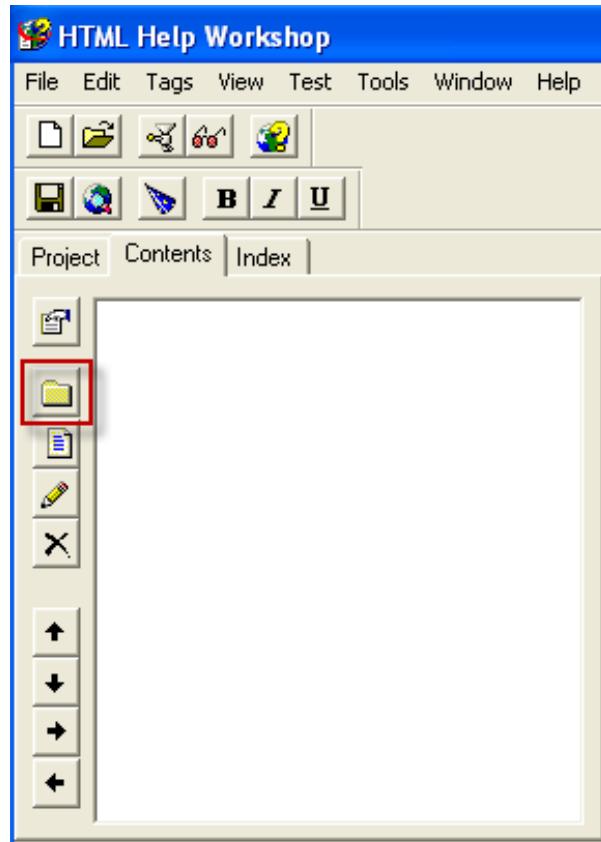
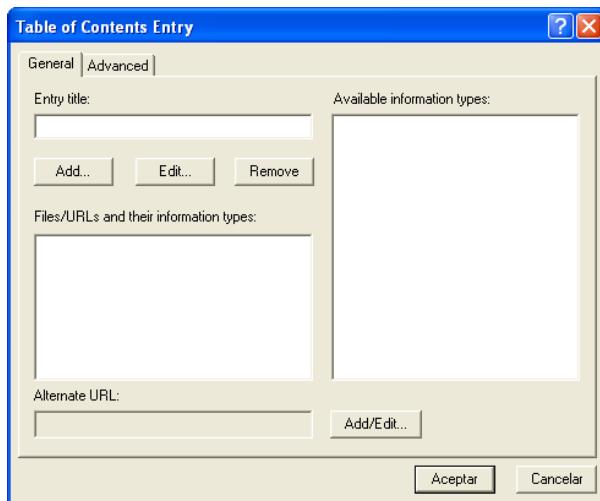
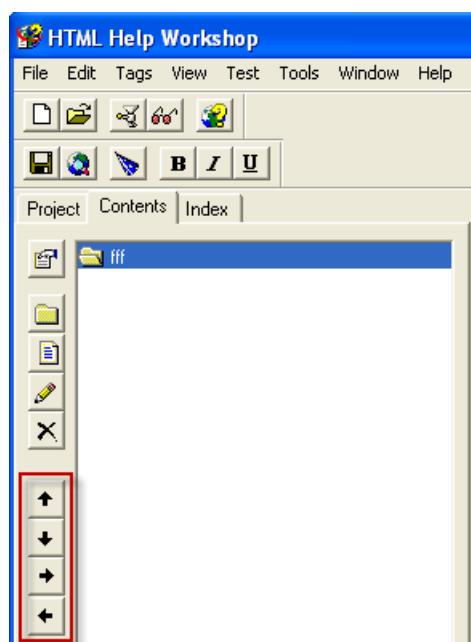
FIGURA 2.22. Icôna Insert a heading

FIGURA 2.23. Entrada nova en la taula de continguts

Per crear un encapçalament nou, feu clic a inserir un encapçalament nou (vegeu la figura 2.24). Apareixerà un missatge que us demanarà si voleu inserir aquest encapçalament a la part superior de la jerarquia. Premeu *Sí* o *No*.

FIGURA 2.24. Inserir a la part superior de la jerarquia

La posició de les pàgines i els encapçalaments sempre es pot canviar més tard amb les fletxes de posició (figura 2.25).

FIGURA 2.25. Fletxes de posició

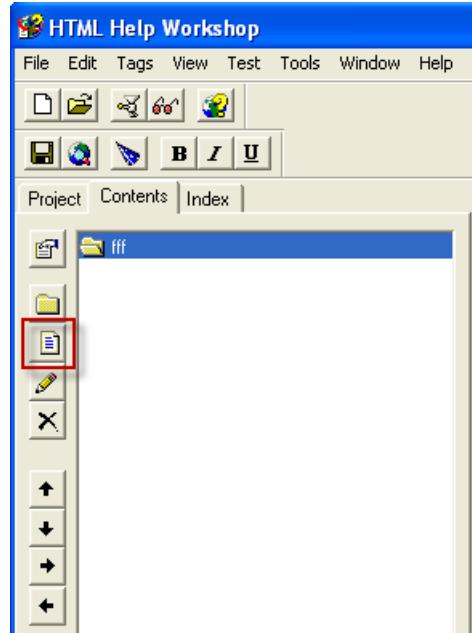
Un encapçalament també pot enllaçar amb un tema. Poseu-hi el nom i premeu el

botó *Add* per introduir el lligam amb el tema que vulgueu.

Crear una pàgina en un tema

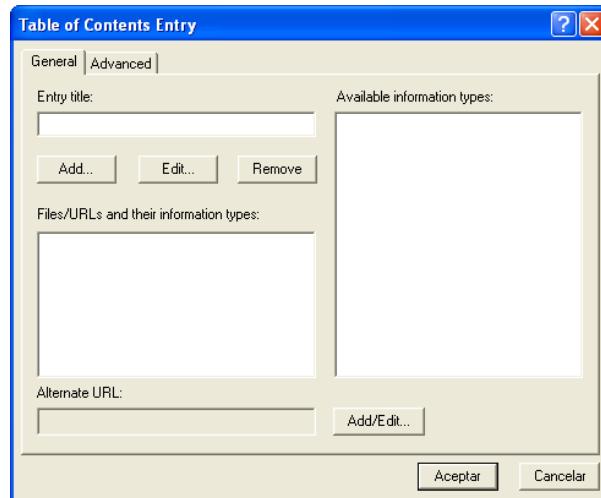
1. Feu clic a *Insert a Page* (figura 2.26).

FIGURA 2.26. Insert a Page

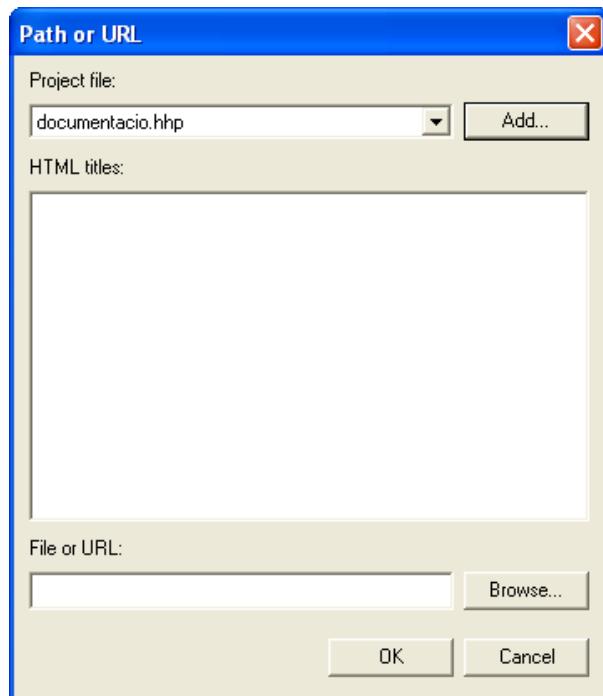


La primera vegada que s'insereix una pàgina, apareix un missatge que demana si es vol inserir la pàgina a la part superior de la jerarquia.

FIGURA 2.27. Entrada nova en la taula de continguts



2. Premeu *Sí* o *No*. Apareixerà el quadre de diàleg d'entrada nova en la taula de continguts, *Table of Contents Entry* (figura 2.27).
3. Escriviu el títol de la pàgina.
4. Feu clic al botó *Add*. Apareixerà el quadre de diàleg *Path or URL* per poder escollir el fitxer (figura 2.28).

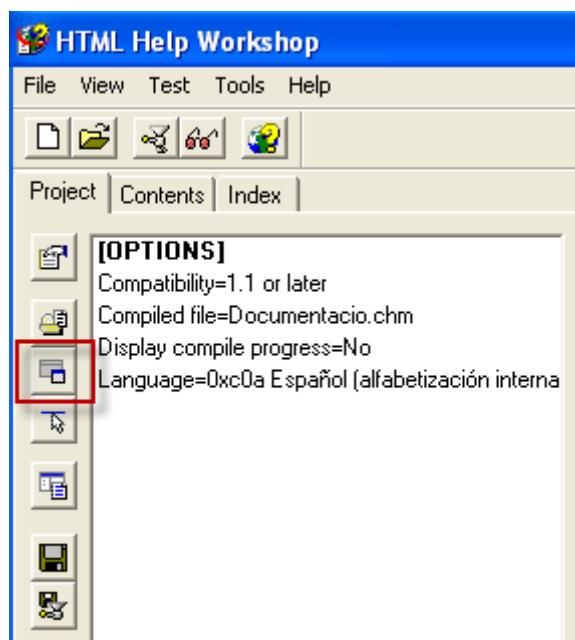
FIGURA 2.28. Entrada nova en la taula de continguts

5. Podeu seleccionar un tema de la llista o bé una adreça URL per escollir-ne un que sigui en un fitxer.

6. Feu *OK*.

Modificar el projecte per utilitzar la taula de continguts

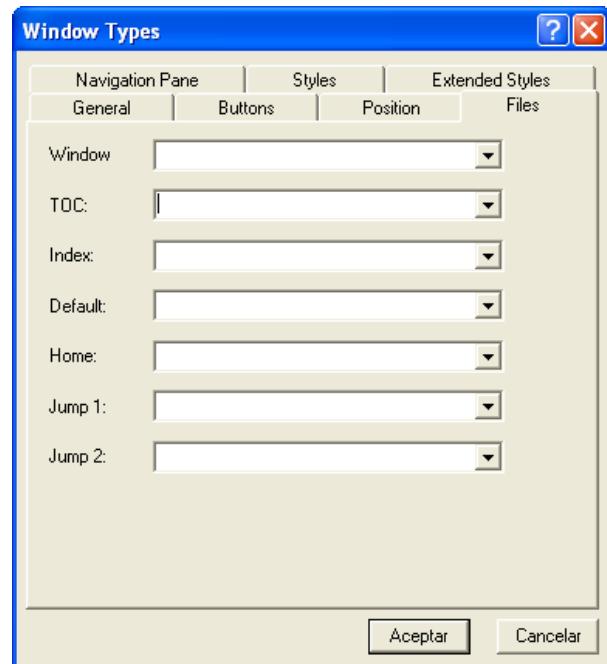
1. Cliqueu, en la pestanya *Project*, a *Add/Modify Window Definitions* (figura 2.29).

FIGURA 2.29. Add

Modify Window Definitions

2. Seleccioneu la pestanya *Files* (figura 2.30).

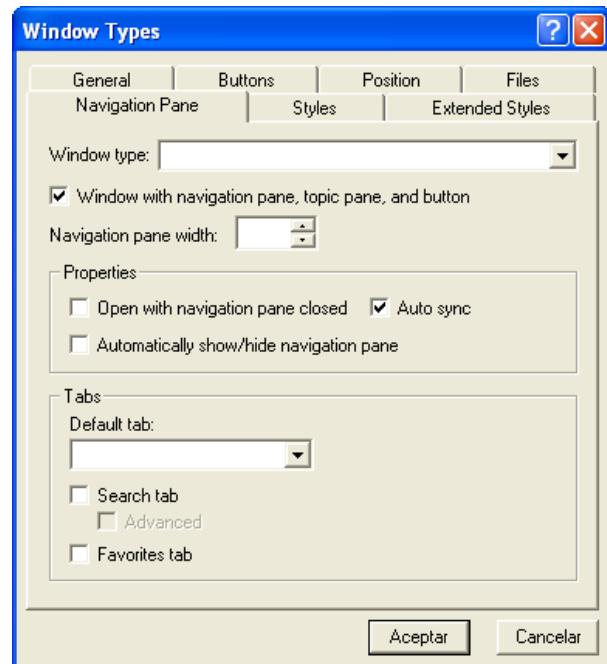
FIGURA 2.30. Window Types



3. Feu clic a la fletxa avall, al costat de TOC, i seleccioneu el contingut del fitxer amb extensió *.hhc*.

4. Seleccioneu la pestanya *Navigation Pane* (figura 2.31).

FIGURA 2.31. Window Types



5. Seleccioneu la sincronització automàtica, *Auto Sync*, i modifiqueu la pestanya per defecte, *Default tab*.

6. Premeu el botó *Aceptar*.

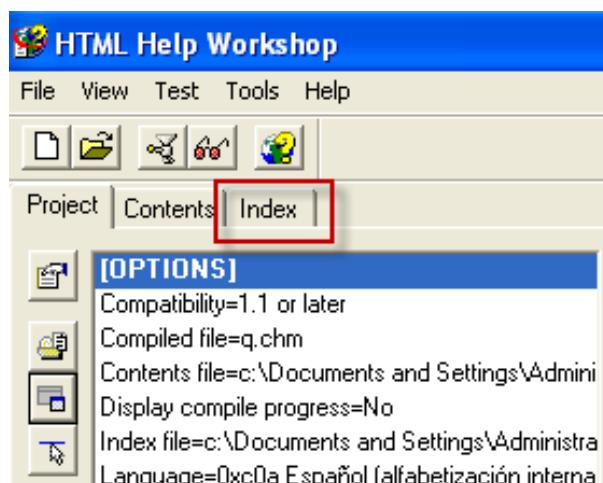
Índex

L'índex està format per un conjunt de paraules clau. Cada paraula clau està associada amb un tema. Fa la mateixa funció que un índex d'un llibre, és a dir, ajuda a trobar el que es busca. L'índex es desa en un fitxer amb extensió *.hhk* .

Creació d'un índex

1. Per accedir a l'índex, cal que feu clic a la pestanya *Index* (figura 2.32).

FIGURA 2.32. Pestanya Index

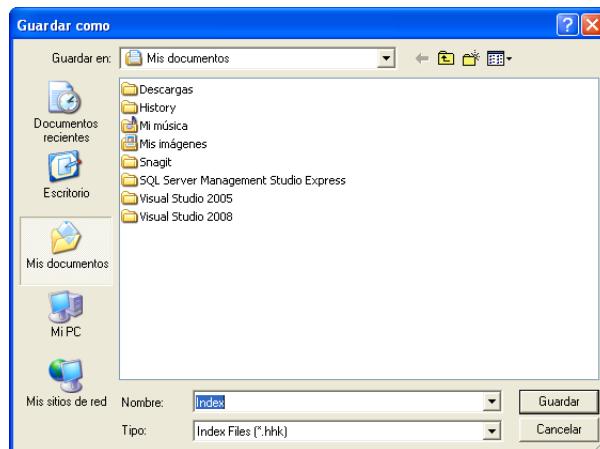


2. Com que l'índex encara no està definit, apareixerà un quadre de diàleg que us preguntarà si en voleu crear un o bé si en voleu obrir un que ja estigui creat (vegeu la figura 2.33). Per crear-ne un, cal mantenir el valor seleccionat per defecte.

FIGURA 2.33. Index Not Specified



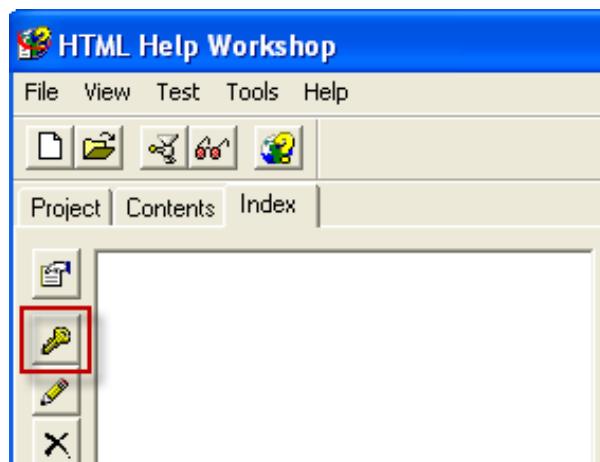
3. En premer *OK*, apareixerà el quadre de diàleg *Guardar como*, en què caldrà introduir un nom per al fitxer (figura 2.34).

FIGURA 2.34. Quadre de diàleg Guardar como

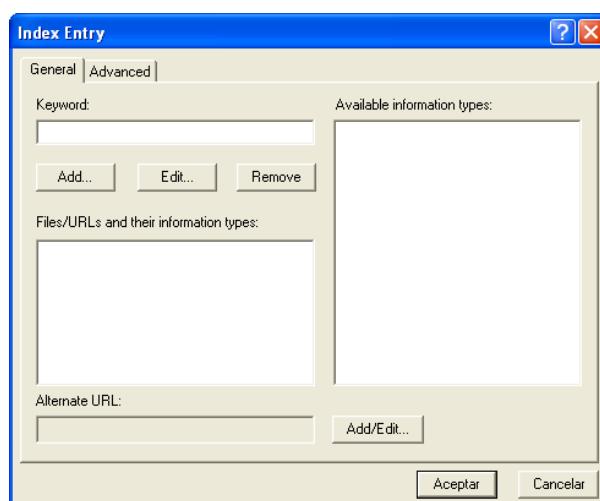
Afegir entrades d'índex a l'arxiu de l'índex

Per crear una paraula clau, cal seguir els passos següents:

1. Feu clic a inserir una paraula clau, *Insert a Keyword* (figura 2.35).

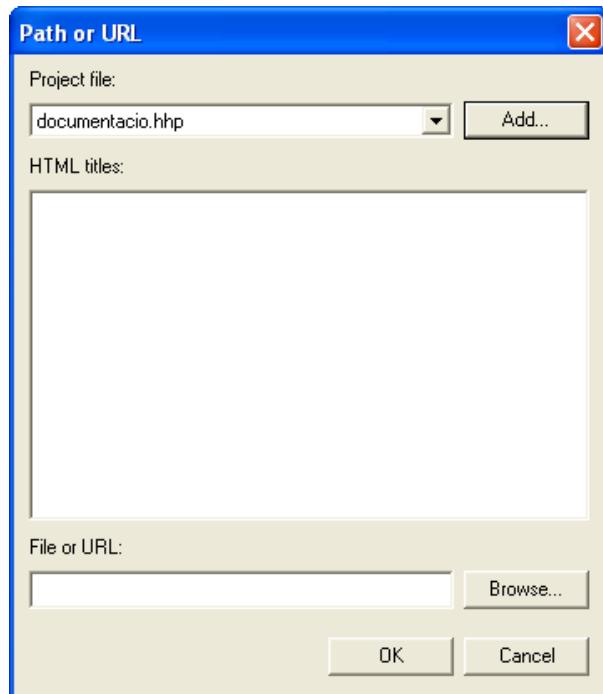
FIGURA 2.35. Pestanya Index

2. Apareixerà el quadre de diàleg *Index Entry* (figura 2.36).

FIGURA 2.36. Quadre de diàleg Index Entry

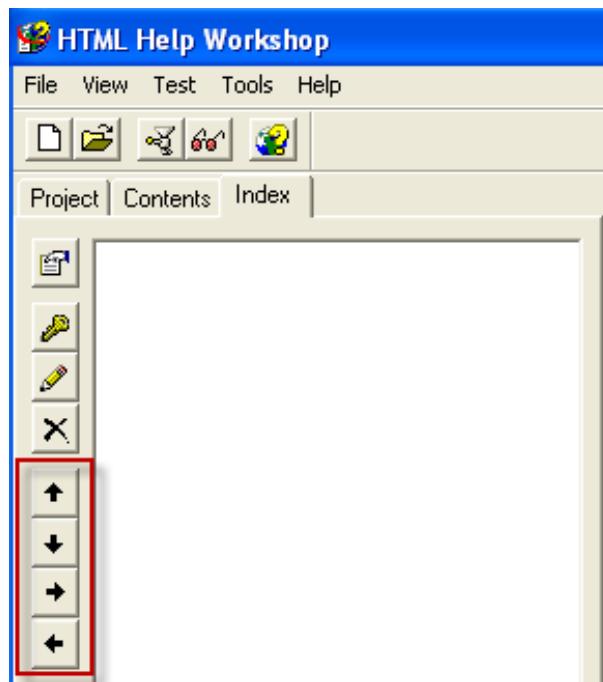
3. Escriviu la paraula clau en el quadre d'entrada de paraules clau, *Keyword*. Feu clic a *Add*. Apareixerà el quadre de diàleg *Path or URL* (figura 2.37).

FIGURA 2.37. Quadre de diàleg Path or URL



4. Seleccioneu el tema de la llista de fitxers i feu clic *OK*.

FIGURA 2.38. Fletxes per posicionar



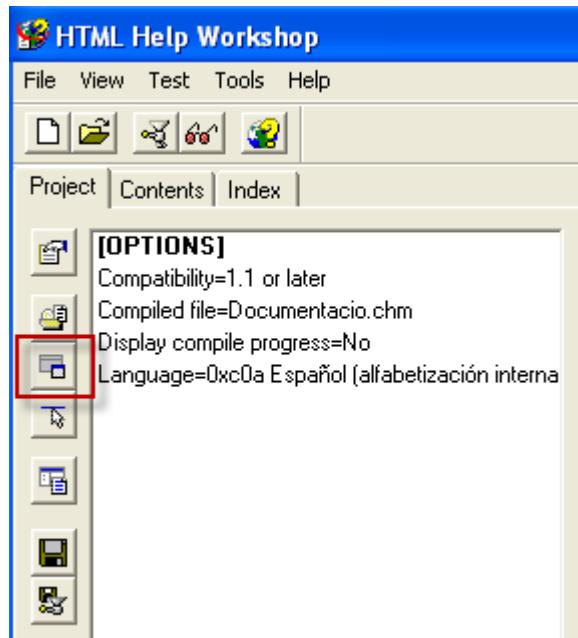
5. Premeu *Aceptar*.

Per crear entrades a dos nivells, seguiu els passos anteriors i utilitzeu les fletxes per col·locar l'entrada en el nivell i la posició corresponent (vegeu la figura 2.38).

Modificar el projecte d'ajuda per utilitzar el fitxer amb l'índex

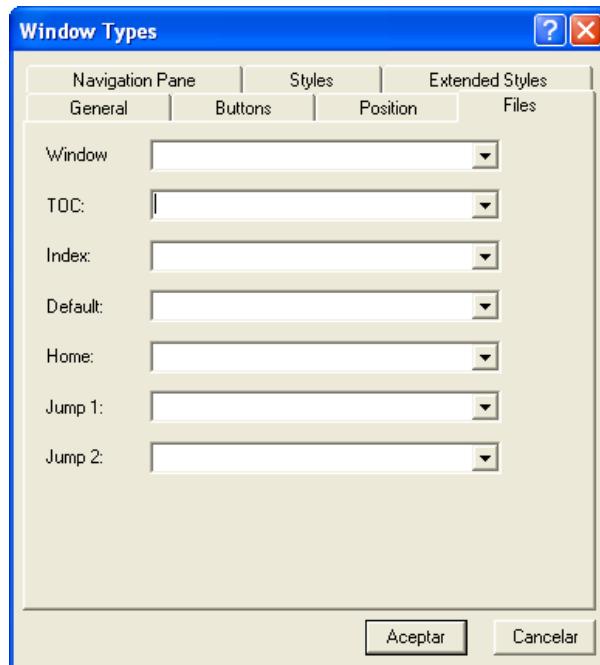
1. Feu clic, en la pestanya *Project*, a *Add/Modify Window Definitions* (figura 2.39).

FIGURA 2.39. Add



Modify Window Definitions

FIGURA 2.40. Window Types



2. Seleccioneu la pestanya *Files* (figura 2.40).

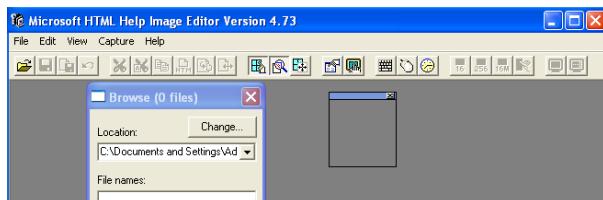
3. Feu clic a la fletxa avall, al costat d'*Index*, i seleccioneu el contingut del fitxer amb extensió *.hhk*.

4. Premeu el botó *Aceptar*.

Afegir gràfics

El programa Microsoft HTML Help Workshop inclou un programa editor d'imatges, l'HTML Help Image Editor, que es pot fer servir per crear captures de pantalla i altres gràfics (figura 2.41).

FIGURA 2.41. HTML Help Image Editor



Per iniciar l'editor d'imatges, cal que seleccioneu *Tools > HTML Help Image Editor* (figura 2.42).

FIGURA 2.42. HTML Help Image Editor



En un projecte d'ajuda es poden incloure els tipus de gràfics següents:

- GIF
- JPG
- BMP
- ICO

2.3.2 El DocBook

Què és el DocBook?

El DocBook és un llenguatge de marques pensat per escriure documentació tècnica, tot i que es pot fer servir per escriure qualsevol tipus de document.

Entorns d'escriptori

El GNOME i el KDE són entorns d'escriptori per a sistemes operatius del tipus UNIX. El GNOME va néixer el 1997 com a projecte encapçalat per Miguel de Icaza, mentre que el KDE el va iniciar Matthias Ettrich l'any 1996.

HAL Computer Systems i *O'Reilly & Associates* el van crear l'any 1991. Avui dia, però, hi participen empreses com Novell, Sun Microsystems i Hewlett Packard, entre altres. També s'utilitza en la documentació de projectes de programari lliure, com el GNOME, el KDE i el PHP.

Un *llenguatge de marques* és un sistema de paraules inserides dins del text del document. Aquest llenguatge aporta significat a les parts del text en què s'aplica. Les paraules inserides, les marques, no apareixen en la versió final del text.

Un exemple de text que conté marques té la forma següent:

```
1 <titol> Text de Títol </titol>
2
3   <paragraf>
4     Això seria un paràgraf, amb un text en <negreta>cursiva</negreta>
5   </paragraf>
```

Les marques d'inici serien:

```
1 <titol>
2 <paragraf>
3 <negreta>
```

i les de final:

```
1 </titol>
2 </paragraf>
3 </negreta>
```

El text que hi ha entre la marca d'inici i la de final està subjecte al significat que es dóna a la marca. El text que hi hagués entre les dues marques (per exemple, en el cas de la marca “títol”) es tractaria com un títol i s’hi podria definir un estil més destacat que el que es donaria a la marca “paragraf”.

Avantatges d'utilitzar el DocBook

L'autor no s'ha de preocupar de l'estil final del document. Mentre crea el text, hi va afegint les marques que donen significat a determinades parts. Un cop creat, s'apliquen els estils, segons el format que es vol crear, a les marques que hi ha i el resultat es guarda en el format seleccionat.

El DocBook permet exportar el text als formats següents:

- HTML, XHTML, Ajuda Java, Ajuda HTML
- PDF, PostScript, RTF, Text
- Format d'ajuda d'UNIX <>

Com que la informació no té format visual, és més fàcil reutilitzar-la en altres documents.

El DocBook és un estàndard, és de codi obert (*open source*) i té una gran base de desenvolupadors i una comunitat de suport important.

Desavantatges d'utilitzar el DocBook

- L'aprenentatge del llenguatge de marques és una mica lent.
- Hi ha moltes marques, més de quatre-centes etiquetes diferents.
- No totes les eines “bones” per crear algun tipus de format són lliures.

Estructura dels documents DocBook

Els documents necessiten un diccionari. En aquest diccionari es descriu la sintaxi i l'estructura que han de tenir els documents escrits amb llenguatge de marques perquè siguin correctes. Aquesta definició del tipus de document s'anomena DTD (*document type definition*, definició de tipus de document).

El DocBook pot treballar amb dos llenguatges de marques diferents, l'XML (*extensible markup language*, llenguatge d'etiquetatge extensible) o l'SGML (*standard generalised markup language*, llenguatge d'etiquetatge generalitzat estàndard). Així, els documents es poden crear utilitzant la sintaxi de l'XML o l'SGML. Quan un document segueix les normes i l'estructura definides en el diccionari, es pot considerar **vàlid**.

L'XML el va desenvolupar el World Wide Web Consortium (W3C). És una simplificació i una adaptació de l'experimental SGML, el qual va crear l'empresa IBM.

Tipus de documents DocBook

Els tipus de documents principals del DocBook són el llibre (*book*) i l'article (*article*).

L'article correspon a documents més petits que un llibre, com els articles de revista, els llibres blancs o les notes tècniques. Sovint és el punt de partida més lògic.

Un llibre típic (en anglès, si més no) conté entre les etiquetes

¹ <bookinfo> i </bookinfo>

informació del llibre mateix sobre el títol, l'autor i el *copyright*. També inclou un o diversos prefacis entre les etiquetes

¹ <preface> i </preface>

així com capítols variats entre les etiquetes

¹ <chapter> i </chapter>

i, a vegades, un apèndix entre les etiquetes

¹ <appendix> i </appendix>

A més, pot contenir bibliografia, glossaris, índexs i un colofó.

Estructura d'un llibre:

```

1  <!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
2
3  <book>
4
5  <bookinfo>
6      <title>El meu primer llibre</title>
7      <author><firstname>Nom</firstname><surname>Cognom</surname></author>
8      <copyright><year>Any</year><holder>Nom Cognom</holder></copyright>
9  </bookinfo>
10
11 <preface><title>Títol prefaci</title> ... </preface>
12
13 <chapter> ... </chapter>
14 <chapter> ... </chapter>
15 <chapter> ... </chapter>
16
17 <appendix> ... </appendix>
18 <appendix> ... </appendix>
19
20 <index> ... </index>
21
22 </book>
```

Els capítols

```
1 <chapter> i </chapter>
```

així com els pròlegs

```
1 <preface> i </preface>
```

i els apèndixs

```
1 <appendix> i </appendix>
```

tenen una estructura semblant. Es componen d'un títol, d'informació addicional, si escau, i d'un nombre qualsevol d'elements a escala de bloc, seguit per un nombre qualsevol de seccions de nivell superior.

Una secció es una divisió lògica del contingut del document. Les seccions es defineixen per mitjà de marques:

```
1 de <sect1> a <sect5>
```

La marca “section” només es pot utilitzar en documents de tipus article. Cada secció, al seu torn, pot contenir un nombre qualsevol d'elements a escala de bloc seguit per un nombre qualsevol de seccions, com es mostra en l'exemple següent.

```

1 <!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
2
3 <chapter>
4     <title>My Chapter</title>
5
6     <para> ... </para>
7
8     <sect1 id="seccio_1">
```

```

9   <title>Títol secció 1</title>
10  <para>Paràgraf secció 1</para>
11  </sect1>
12
13  <sect1 id="seccio_2">
14    <title>Títol secció 2</surname>
15    <para>Paràgraf secció 2</para>
16    <sect2 id="seccio_2.1">
17      <title>Títol secció 2.1</email>
18      <para>...</para>
19    </sect2>
20  </sect1>
21
22  </chapter>

```

Essencialment, el cos d'un article és el mateix que el d'un capítol, o que el de qualsevol altre element a escala de component. Es mostra en l'exemple següent.

```

1  <!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
2  <article>
3
4    <artheader>
5      <title>El meu Article</title>
6      <author><honorable>Dr</honorable><firstname>Eduard</firstname>
7        <surname>Jarder</surname></author>
8    </artheader>
9
10   <para> ... </para>
11
12   <sect1>
13     <title>Títol secció 1</title>
14     <para> ... </para>
15   </sect1>
16
17   <bibliography> ... </bibliography>
18
19 </article>

```

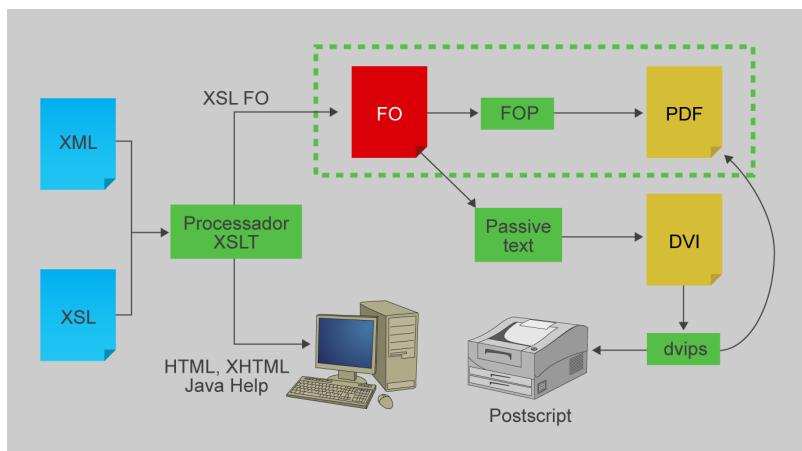
XSLT

L'XSLT (*extensible stylesheet language transformations*) és un estàndard de l'organització W3C que defineix una manera de transformar documents XML en un altre tipus de document.

Procés de transformació dels documents DocBook

Quan l'autor del contingut acaba de crear un fitxer XML amb el format del DocBook, comença el procés de transformació. Cal aplicar un estil identificat segons uns patrons a cada marca. Aquests estils es desen en uns arxius d'extensió *xsl*, anomenats *fulls d'estil*, que defineixen les transformacions que cal aplicar a cada element DocBook.

Per aplicar els estils es fa servir un processador XSL, programari que s'encarrega d'aplicar les transformacions, definides per les fulles d'estil, a cada element DocBook de l'arxiu que processa. El resultat de la transformació conté la mateixa informació que ha creat l'autor del contingut, però presenta un altre format que permet veure-la com a PDF o HTML (vegeu la figura 2.43).

FIGURA 2.43. Procés de transformació DocBook

El llenguatge de marques DocBook

Característiques generals:

- Totes les marques s'han d'escriure en minúscules
- Tota marca que s'obri s'ha de tancar
- Els identificadors de capítols, seccions i taules, entre altres, han de ser únics.

Marca "application"

1 <application>

S'usa per referir-se a una aplicació gràfica o text.

Per exemple:

1 Per treballar amb arxius Zip comprimits, pot utilitzar el programa
 2 <application> File Roller </ application>
 3 en mode gràfic o el programa <application> Zip </application>
 4 des de la línia d'ordres.

Marca "citetitle"

1 <citetitle>

Es fa servir per citar una referència externa.

Per exemple:

1 Si voleu obtenir una descripció detallada del procés d'installació,
 2 vegeu la <citetitle> Guia oficial d'installació
 3 per a Intel x86 </citetitle>.

Marca "command"

```
1 <command>
```

Es fa servir per fer referència al codi executable d'una aplicació, una ordre o una directriu de configuració.

Per exemple:

```
1 Per configurar la xarxa amb el Linux Red Hat, pot utilitzar
2   l'ordre <command> netconfig </command>. <command> DocumentRoot </command>
3   Aquesta directriu de configuració indica la ubicació al servidor Apache.
```

Marca "computeroutput"

```
1 <computeroutput>
```

Es fa servir per mostrar missatges que apareixen a la pantalla.

Per exemple:

```
1 <computeroutput> rm: esborrar el fitxer normal «»Makefile? (S/N) </
  computeroutput>
```

El contingut s'ha de posar immediatament després de la marca. Com en la marca "screen", es respecten els espais i les línies extres.

Marca "emphasis"

```
1 <emphasis>
```

Es fa servir per posar èmfasi en una frase.

Per exemple:

```
1 Si s'utilitza <command> /sbin/mke2fs </command> sobre un sistema d'arxius,
2   es perdrà <emphasis> tota </emphasis> la informació que hi hagi.
```

Marca "example"

```
1 <example>
```

Es fa servir per mostrar exemples acompañats d'un títol.

Típicament són seccions de codi o de fitxers de configuració.

Per exemple:

```

1 <example id="ej-resolv">
2   <title> Domini i servidor DNS a utilitzar </title>
3   <screen>
4     <computeroutput> domain unsitio.com
5     nameserver 192.168.1.4; </computeroutput>
6   </screen>
7 </example>

```

Marca "filename"

```

1 <filename>

```

Es fa servir per fer referència a noms de fitxers o directoris.

Si es tracta d'un directori, és preferible incloure / al final per aclarir el text.

Per exemple:

```

1 L'arxiu <filename> /etc/passwd </filename> conté diverses
2   peces d'informació d'un compte d'usuari.

```

Marca "figure"

```

1 <figure>

```

S'utilitza si es vol incloure una figura que tingui un títol i una descripció breu.

Per exemple:

```

1 <figure id="f-pref-mozilla">
2   <title> Preferències del Mozilla </title>
3   <mediaobject>
4     <imageobject>
5       <imagedata fileref=".//figs/mozpref.eps" format="EPS"/>
6     </imageobject>
7     <imageobject>
8       <imagedata fileref=".//figs/mozpref.png" format="PNG"/>
9     </imageobject>
10    <textobject>
11      <phrase> La figura mostra les preferències...</phrase>
12    </textobject>
13  </mediaobject>
14 </figure>

```

Marca "footnote"

```

1 <footnote>

```

S'utilitza per inserir una nota a peu de pàgina.

Per exemple:

```
1 Hi ha, per defecte, la partició assignada a <filename> /boot </filename>.  
2 <footnote>  
3   <para> En alguns sistemes, a causa de problemes amb discos grans, s'acostuma  
4     a crear aquesta partició  
5     dins els primers 1.024 cilindres. </para>  
6   </footnote> Requerirà un espai de 60 a 100 MiB.
```

Marca "foreignphrase"

```
1 <foreignphrase>
```

S'usa per mostrar una paraula o una frase en una llengua diferent a la del document.

Per exemple:

```
1 Les ordres següents s'han d'escriure des de l'indicador (<foreignphrase>  
2   indicador  
3   </foreignphrase>).
```

Llista d'elements "itemizedlist"

```
1 <itemizedlist>
```

S'usa per mostrar informació breu que no requereix un ordre específic.

Per exemple:

```
1 <itemizedlist>  
2  
3   <listitem>  
4     <para> Revisar ús d'espai lliure en disc. </para>  
5   </listitem>  
6  
7   <listitem>  
8     <para> Monitorar i gestionar processos. </para>  
9   </listitem>  
10  
11   <listitem>  
12     <para> Mantenir programes al dia. </para>  
13   </listitem>  
14  
15 </itemizedlist>
```

Llista ordenada "orderedlist"

```
1 <orderedlist>
```

S'usa per mostrar una llista d'elements en què l'ordre és important.

Per exemple:

```

1 <orderedlist>
2
3   <listitem>
4     <para> ·Installar el paquet. </para>
5   </listitem>
6
7   <listitem>
8     <para> Modificar el fitxer de configuració. </para>
9   </listitem>
10
11  <listitem>
12    <para> Iniciar el servei. </para>
13  </listitem>
14
15 </orderedlist>

```

Llista de termes i definicions "variablelist"

```

1 <variablelist>

```

S'usa per mostrar una llista de termes i definicions amb les descripcions corresponents.

Per exemple:

```

1 <variablelist>
2
3   <varlistentry>
4     <term> Servidor X </term>
5
6   <listitem>
7     <para> Ofereix les operacions bàsiques de...</para>
8   </listitem>
9
10  </varlistentry>
11
12  <varlistentry>
13
14    <term> Servidor de lletres X </term>
15
16    <listitem>
17      <para> Proveeix els tipus de lletres... </para>
18    </listitem>
19
20  </varlistentry>
21  ...
22 </variablelist>

```

Llista simple "simplelist"

```

1 <simplelist>

```

S'usa principalment per crear una llista dins una taula.

Per exemple:

```

1 <simplelist>
2   <member> Processador Intel </member>

```

```

3 <member> Pentium IV 02/04 Ghz </member>
4 <member> 512 MIB RAM </member>
5 <member> 80 GIB Disc </member>
6 <member> CDWR / DVD </member>
7 </simplelist>
```

Marca "option"

```
1 <option>
```

S'usa per indicar una opció d'una determinada ordre.

Per exemple:

```

1 L'ordre <command> uname </command> seguida de l'opció
2   <option> r </option> mostra la versió de nucli que està
3   executant el seu sistema.
```

Marca "para"

```
1 <para>
```

S'utilitza al voltant de qualsevol paràgraf simple.

Per exemple:

```

1 <para>
2   L'ordre <command> uname </command> seguida de l'opció
3     <option> r </option> mostra la versió de nucli que
4     està executant el seu sistema.
5 </para>
```

Només heu d'utilitzar marques “para” al voltant de paràgrafs simples.

Concretament, no heu de fer servir “para” al voltant de les marques següents:

```

1 <itemizedlist>
2 <orderedlist>
3 <variablelist>
4 <screen>
5 <table>
```

Marca "prompt"

```
1 <prompt>
```

S'usa per mostrar un indicador.

Per exemple:

```
1 Per iniciar el sistema, cal que escriuvi //linux//  
2 en l'indicador <prompt> LILO: </prompt>.  
3 L'indicador <prompt> # </prompt> es reserva a l'usuari  
4 primari.
```

Marca "replaceable"

```
1 <replaceable>
```

Es fa servir per indicar que el lector ha de substituir el text que hi ha entre les marques per la informació més adequada, segons el seu cas concret.

Per exemple:

```
1 Els mòduls del nucli es troben al directori  
2   <filename> /lib/modules/ <replaceable> versió de  
3   nucli </replace> / </filename>.
```

Marca "screen"

```
1 <screen>
```

S'usa per mostrar una llista de programes, arxius o qualsevol resultat mostrat a la pantalla.

Per exemple:

```
1 <para> Per veure la versió del nucli, heu d'escriure el següent: </para>  
2  
3   <screen>  
4     <userinput> uname -r </userinput>  
5   </screen>  
6  
7   <para> Mostrarà un resultat semblant a aquesta línia: </para>  
8  
9   <screen>  
10    <computeroutput> 2.4.29-686 </computeroutput>  
11  </screen>
```

La marca i el contingut han d'estar justificats a l'esquerra.

Qualsevol espai dins la marca "screen" es conserva.

Aquesta marca en pot contenir d'altres:

```
1 <computeroutput>  
2 <userinput>  
3 <replaceable>
```

Per definició, no és necessari incloure altres marques dins de "screen".

Marca "table"

```
1 <table>
```

S'usa per mostrar una taula.

Per exemple:

```
1 <table id="tbpopimap">
2
3     <title>Característiques del POP i l'IMAP</title>
4
5     <tgroup cols="3">
6
7         <colspec colnum="1" colname="carac" colwidth="120pt"/>
8         <colspec colnum="2" colname="pop" colwidth="30pt"/>
9         <colspec colnum="3" colname="imap" colwidth="30pt"/>
10
11     <thead>
12         <row>
13             <entry>Característica</entry>
14             <entry>POP</entry>
15             <entry>IMAP</entry>
16         </row>
17     </thead>
18
19     <tbody>
20         <row>
21             <entry>T Treballa en línia (//online//)</entry>
22             <entry>Sí</entry>
23             <entry>Sí</entry>
24         </row>
25     </tbody>
26
27 </tgroup>
28
29 </table>
```

Marca "userinput"

```
1 <userinput>
```

S'utilitza per indicar el que l'usuari ha d'escriure.

Per exemple:

```
1 En l'indicador del sistema, cal que escriuvi el següent:
2     <userinput> fdformat / dev/fd0 </ userinput>
```

Marca "xref"

```
1 <xref>
```

S'usa per fer referència a una altra secció o capítol del document.

Per exemple:

```
1 Per obtenir més informació sobre les particions de disc
2 en Linux, vegeu <xref linkend="secparticions" />.
```

Instal·lar el DocBook en sistemes GNU/Linux (Debian)

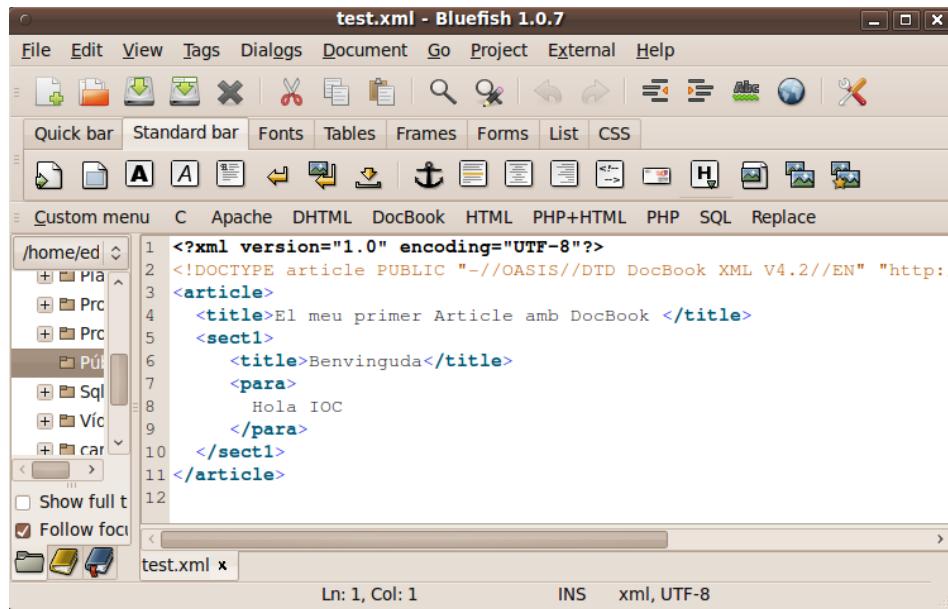
1. Instal·leu els fulls d'estil XSL per generar els documents HTML, XHTML, HTML Help, etc.
 - apt-get install docbook-xsl
2. Instal·leu els fulls d'estil XSL per generar documents PDF
 - apt-get install fop docbook-xsl-doc-pdf
3. Instal·leu el processador per fer les transformacions XSL
 - apt-get install xsltproc
4. Instal·leu el Bluefish, processador de textos que ajuda a treballar amb fitxers XML i DocBook.
 - apt-get install bluefish

Un cop instal·lat el programari necessari, començareu a escriure el vostre primer document amb el DocBook.

Primer document amb el DocBook

Obriu l'editor Bluefish i creeu el fitxer *test.xml* amb el contingut següent:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2   <!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
3 "http://docbook.org/xml/4.2/docbookx.dtd">
4
5   <article>
6
7     <title>El meu primer article amb el DocBook </title>
8
9     <sect1>
10       <title>Benvinguda</title>
11       <para>
12         Hola, IOC
13       </para>
14     </sect1>
15
16   </article>
```

FIGURA 2.44. Bluefish

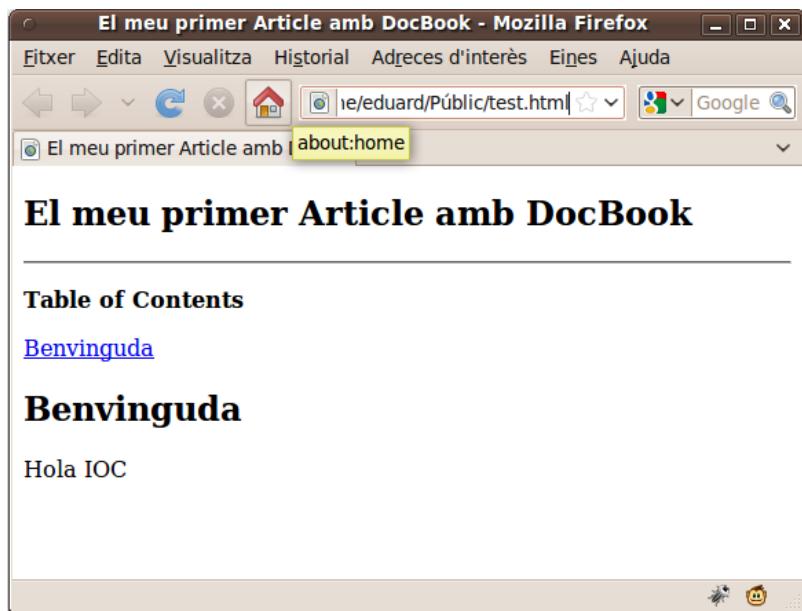
Executeu la instrucció següent en un terminal. Transformeu el fitxer XML que acabeu de crear en un fitxer amb format HTML.

```
1 xsltproc -o test.html /usr/share/xml/docbook/stylesheet/nwalsh/xhtml/docbook.xsl
2 test.xml
```

Explicació de la sentència:

- *xsltproc* és l'aplicació que fa la conversió.
- *-o test.html* estableix el fitxer de sortida.
- *.../docbook.xsl* és el full d'estil que s'ha utilitzat per fer la conversió (converteix XML a XHTML).
- *test.xml* diu a l'aplicació xsltproc on és el fitxer d'entrada.

El resultat és un fitxer en format HTML. El podeu veure, en la figura següent, obert amb el navegador Firefox (vegeu la figura 2.45).

FIGURA 2.45. Article DocBook

Convertir el fitxer XML a PDF és un procés de dos passos:

En primer lloc, cal transformar el fitxer XML en un fitxer amb format intermig FO amb l'aplicació xsltproc.

```
1 xsltproc -o test.fo /usr/share/xml/docbook/stylesheet/nwalsh/fo/docbook.xsl
2 test.xml
```

Explicació de la sentència:

- *xsltproc* és l'aplicació que fa la conversió.
- *-o test.fo* estableix el fitxer de sortida en el format intermig.
- *.../docbook.xsl* és el full d'estil que s'utilitza per fer la conversió (converteix XML a FO).
- *test.xml* diu a l'aplicació xsltproc on és el fitxer d'entrada.

En segon lloc, cal convertir el fitxer FO a PDF amb l'aplicació FOP.

```
1 fop -pdf test.pdf -fo test.fo
```

Explicació de la sentència:

- *FOP* és l'aplicació que fa la conversió a PDF.
- *-pdf test.pdf* estableix el fitxer de sortida en el format PDF.
- *test.fo* diu a l'aplicació xsltproc on és el fitxer d'entrada.

El resultat és un fitxer en format PDF. El podeu veure, en la figura 2.46, obert amb un visualitzador de PDF.

FIGURA 2.46. :figure:Figure64:

2.4 Taules de continguts, índexs i sistemes de cerca, entre d'altres

Les taules de contingut proporcionen una classificació de la informació per temes a fi de facilitar la cerca a l'usuari.

L'organització de la informació es fa en temes i subtemes. Per tal de diferenciar aquests dos elements de manera visual, es poden fer servir icones. Així, s'ajuda l'usuari a trobar més ràpidament el contingut que busca.

Un índex dins d'un sistema d'ajuda ha de servir perquè els usuaris trobin ràpidament la informació que necessiten.

Les paraules que els usuaris consideren clau han de constar en el contingut de l'índex. Aquestes paraules es poden adreçar a diferents tipus d'usuaris. Poden ser específiques, generals o sinònims de termes que es fan servir en l'aplicació.

L'objectiu de l'índex és ajudar l'usuari a trobar el que busca. Per tant, l'índex ha d'intentar que l'usuari pugui arribar a la informació de moltes maneres, perquè així la probabilitat que trobi alguna paraula que li resulti útil serà més gran.

Normalment un índex es dissenya perquè tingui dos nivells. Les entrades del primer nivell descriuen un tema general, mentre que les del segon corresponen a temes específics dins del tema més general del primer nivell.

2.5 Tipus de manuals: manual d'usuari, guia de referència, guia ràpida i manual d'instal·lació, configuració i administració

Hi ha diversos tipus de documentació. Aquests documents abracen des del desenvolupament fins a la implementació, l'operació i el manteniment de l'aplicació.

2.5.1 El manual d'usuari

El manual d'usuari conté la informació necessària perquè els usuaris puguin utilitzar correctament l'aplicació. Ha de servir com a manual d'aprenentatge. Permet que els usuaris sàpiguen de manera detallada quines activitats poden desenvolupar en l'aplicació i quines dades o documents en poden obtenir.

Reuneix la informació, les normes i la documentació necessària perquè l'usuari conegui i utilitzi adequadament l'aplicació desenvolupada. Ha de definir les funcions que l'usuari hi pot dur a terme i l'ha d'informar sobre la resposta que ha de donar a cada missatge d'error.

En elaborar el manual d'usuari, cal tenir en compte a qui es dirigeix, ja que el manual el pot utilitzar des del director d'una empresa fins a la persona que introduceix dades. Per tant, s'ha de redactar de manera clara i senzilla perquè qualsevol tipus d'usuari en pugui comprendre el contingut.

2.5.2 La guia de referència

La guia de referència és el document definitiu per a l'usuari i ha de ser completa i exhaustiva.

Descriu amb detall les qualitats i les possibilitats del sistema i l'ús, els informes d'error generats i les situacions en què sorgeixen aquests errors.

2.5.3 La guia ràpida

La guia ràpida constitueix un subconjunt de la guia d'usuari.

En un espai molt més reduït, inclou l'explicació de les accions principals que l'aplicació permet dur a terme per tal de poder-la començar a utilitzar sense haver de llegir tot el manual d'usuari.

2.5.4 El manual d'instal·lació

El manual d'instal·lació és la guia que conté la informació necessària per implementar l'aplicació en un ambient particular.

Dins d'aquest document hi ha les instruccions per posar en marxa el sistema. També inclou les normes d'utilització, entre les quals hi ha les de seguretat (físiques i d'accés a la informació).

En aquest manual s'explica com iniciar-se en el sistema i com utilitzar les qualitats comunes que té. Aquesta documentació ha de dir a l'usuari com sortir d'un problema quan les coses funcionen malament.

2.5.5 El manual de configuració

El manual de configuració conté una explicació detallada de tots els paràmetres de configuració disponibles en l'aplicació.

2.5.6 El manual d'administració

L'objectiu d'aquest manual és explicar tot el que un usuari administrador ha de saber per dur a terme l'administració de l'aplicació.

Distribució d'aplicacions. Proves

Marcel García

Desenvolupament d'interfícies

Índex

Introducció	5
Resultats d'aprenentatge	7
1 Distribució d'aplicacions	9
1.1 Cicle de vida d'una aplicació. Fase de finalització i transferència	10
1.2 Components d'una aplicació. Empaquetament	12
1.2.1 Biblioteques	13
1.2.2 Entorn de treball	13
1.2.3 Empaquetament	14
1.2.4 Assemblament o assembly	17
1.3 Tipus d'empaquetaments	18
1.4 Paquets autoinstal·lables. Desinstal·lables	21
1.5 Eines per a la creació de paquets d'instal·lació	24
1.6 Exemple de la creació d'un paquet de tipus .deb en LINUX	25
1.7 Personalització de la instal·lació. Interacció amb l'usuari	30
1.8 Exemple de la distribució d'una aplicació amb Visual Studio 2010	30
2 Realització de proves	41
2.1 Validació de la correctesa en el desenvolupament d'una aplicació	41
2.2 Objectius, importància i limitacions del procés de prova. Estratègies	43
2.3 Classificacions dels tipus de proves	46
2.4 Proves unitàries	47
2.4.1 Proves de caixa negra	48
2.4.2 Proves de caixa blanca	50
2.4.3 Revisions	51
2.5 Proves d'integració	52
2.5.1 Prova d'integració ascendent	53
2.5.2 Prova d'integració descendent	54
2.5.3 Prova d'integració combinada	54
2.5.4 Prova d'integració de big bang o gran explosió	55
2.6 Proves de càrrega i acceptació	55
2.7 Proves de sistema i de seguretat	56
2.8 Proves de regressió i proves de fum	57
2.9 Validació formal de la correctesa	58
2.10 Derivació formal de programes	60
2.11 Proves de programari: metodologies, IEEE, TMM	61
2.11.1 Metodologia de sistemes d'informació: Mètrica 3	61
2.11.2 IEEE Estàndard 829-1983	62
2.11.3 Test per als models de maduresa	63
2.12 Eines per desenvolupar proves	64
2.12.1 JUnit	64
2.12.2 NUnit	65

2.12.3 XUnit	66
2.12.4 Clover	66
2.12.5 OpenSTA	67
2.12.6 Bugzilla	67

Introducció

Quan s'està desenvolupant un programari, una de les tasques bàsiques del programador i del seu equip és fer les proves necessàries per tal de garantir que el producte arriba al client sense errors i complint els requisits demanats. Aquesta no és una tasca senzilla i s'ha de ser tenir en compte des de l'inici del desenvolupament. De fet, un producte en contínua evolució exigeix proves constants i, per tant, es converteix a més en una de les etapes més costoses dins del manteniment de l'aplicació.

En l'apartat “Realització de proves” veurem els principals objectius que perseguiem i els diferents tipus de proves que podem fer. Hem de tenir en compte que la programació, com a activitat humana considerada moltes vegades com a “artesanal”, no està exempta d'errades i, per tant, les proves seran una gran ajuda per garantir la qualitat del producte final.

Una vegada tenim l'aplicació desenvolupada i provada, haurem de distribuir-la. Els clients poden ser usuaris finals que executen una aplicació web o d'escriptori en els diferents sistemes operatius. A vegades la distribució serà per a un petit grup i en altres ocasions pot ser per a milers d'usuaris. En qualsevol cas, hem de fer arribar als nostres clients el desenvolupament que hem fet.

En l'apartat “Distribució d'applicacions” veurem com podem empaquetar la nostra aplicació perquè es distribueixi fàcilment i estudiarem tant diferents formats com les eines que ens ajudaran a obtenir-los.

Resultats d'aprenentatge

En finalitzar aquest mòdul l'alumne:

1. Prepara aplicacions per la seva distribució avaluant i analitzant eines específiques.
 - Empaqueta els components que requereix l'aplicació.
 - Personalitza l'assistent d'instal·lació.
 - Empaqueta l'aplicació per ser instal·lada de manera típica, completa o personalitzada.
 - Genera paquets d'instal·lació fent servir l'entorn de desenvolupament.
 - Genera paquets d'instal·lació fent servir eines externes.
 - Genera paquets instal·lables en mode desatès.
 - Prepara el paquet d'instal·lació perquè l'aplicació pugui ser correctament desinstal·lada.
 - Prepara l'aplicació per ser baixada des d'un servidor web i executada.
2. Avalua el funcionament d'applicacions dissenyant i executant proves.
 - Estableix una estratègia de proves.
 - Realitza proves d'integració dels diferents elements.
 - Realitza proves de regressió.
 - Realitza proves de volum i estrès.
 - Realitza proves de seguretat.
 - Realitza proves d'ús de recursos per part de l'aplicació.
 - Documenta l'estratègia de proves i els resultats obtinguts.

1. Distribució d'applicacions

El desenvolupament de programari és una tasca que es pot considerar en contínua evolució al llarg del temps. Ja sigui una petita aplicació per a un particular o per a una petita empresa on serà utilitzada o un programari a gran escala per a molts usuaris simultanis, el producte desenvolupat haurà de seguir una evolució i un manteniment.

Aquest manteniment es podrà donar per diferents raons:

- Actualitzacions que els desenvolupadors han fet del programari, per voluntat pròpia.
- Actualitzacions que ha reclamat l'usuari.
- Actualitzacions necessàries per un canvi en l'entorn.

Quins poden ser els motius d'aquestes actualitzacions?

- Adaptació del programari a un nou sistema operatiu (o a les seves evolucions).
- Adaptació del programari a modificacions en les lleis o les normes d'un país que afecten al seu funcionament (per exemple, en temes d'economia o comptabilitat)

Altres motius provocats per l'usuari o client final:

- Adaptació del programari a noves necessitats dels usuaris.
- Adaptació del programari a canvis en les regles del joc de les empreses.

Un punt i a part en el desenvolupament de programari es troba en la distribució al client. En aquell moment l'usuari final disposarà d'una versió concreta del producte desenvolupat que podrà instal·lar en les seves oficines (en el cas de ser distribuït en un format autoinstal·lable) o que li hauran vingut a instal·lar i a parametrizar adequant el programari desenvolupat al seu entorn i a les seves necessitats específiques.

Aquesta distribució inicial oferirà un estat concret de l'aplicació. A partir d'aquesta versió, en el cas de necessitar evolucions, s'haurà d'observar i decidir la millor manera de dur-les a terme. Les distribucions de les evolucions del programari també es podran dur a terme a partir de nous paquets autoinstal·lables que facin una instal·lació nova de tota l'aplicació, o bé a partir d'ampliacions o substitucions de les biblioteques o paquets concrets que s'hagin modificat.

1.1 Cicle de vida d'una aplicació. Fase de finalització i transferència

Per arribar a parlar de la fase de finalització i transferència, que és aquella fase en la qual es lliurarà (o instal·larà) el producte finalitzat a l'usuari final o client, cal abans revisar les fases per les quals passarà un projecte de desenvolupament de programari.

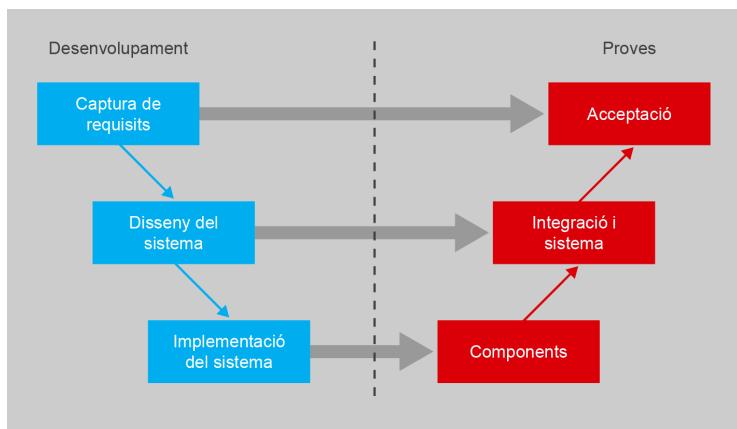
Hi ha molts tipus de cicle de vida del programari, com per exemple:

- Model iteratiu incremental.
- Model en cascada pur o seqüencial.
- Model en cascada iteratiu.
- Model evolutiu incremental
- Model en V.
- Model espiral

Per entendre la importància de la distribució de les aplicacions i la seva vinculació amb totes les fases del cicle de vida d'una aplicació, escollirem dos models: el model en cascada i el model en V.

El **model en cascada** és un tipus de model en el qual es fa servir la finalització de cada fase com a entrada per a la fase següent. Cal revisar cada fase abans que estigui finalitzada. Una vegada validada arrancarà la fase següent. Si no hi ha una correcta revisió de les possibles errades aquestes es poden arrossegar al llarg de tot el projecte i fer molt costosa la solució.

FIGURA 1.1. Model en V



Un altre exemple de cicle de vida és el **model en V** (figura 1.1). Aquest tipus de model es basa en unes fases semblants a les fases del model en cascada, però aquestes fases s'agrupen en dos nivells, en l'anomenat *diagrama bidimensional de nivell-desenvolupament*. En aquest tipus de model no hi ha una diferenciació tan

gran entre fases ni una independència entre aquestes. Per exemple, durant la fase d'implementació del sistema també es treballa amb parts de la fase del disseny del sistema i de la integració. Es treballa amb un sistema de programari que s'organitza (durant el disseny) i es construeix (durant la integració) amb diversos mòduls.

Quines són algunes de les fases que es poden trobar en un o altre model de cicle de vida?

1. Anàlisi: és la primera fase que caldrà dur a terme just després de la proposta de projecte (que molts autors deixen fora del projecte en si). A la fase d'anàlisi es durà a terme una presa de requeriment exhaustiva i es definirà què haurà de fer el sistema o aplicació per desenvolupar. No es demanen detalls tecnològics ni de desenvolupament.
2. Disseny: consisteix a elaborar i prendre les decisions adients per al desenvolupament correcte de l'aplicació definida a la fase d'anàlisi. Cal descompondre el sistema i organitzar-lo per desenvolupar el programari entre un equip de treball, definint l'estructura global i especificant què ha de fer cada part de manera detallada.
3. Desenvolupament o codificació: implementació del codi que compondrà l'aplicació o el sistema per desenvolupar. Aquesta fase implica la implementació del que s'ha dissenyat i analitzat a les fases anterior. Cada mòdul desenvolupat haurà de passar les proves unitàries de manera individual. En aquesta fase encara no es pot crear un paquet o un instal·latable dels mòduls o les biblioteques, i caldrà esperar a haver integrat tots els continguts i a haver passat les proves de sistema.
4. Integració i proves: la integració correspon a agafar tots els mòduls ja finalitzats i anar-los ajuntant per formar l'aplicació completa. Caldrà validar i verificar el funcionament correcte de tots els mòduls per separat i de l'aplicació una vegada integrada. També caldrà revisar tota la documentació generada, tant la tècnica com la comercial o la d'usuari.
5. Finalització i transferència: en aquesta fase s'haurà de tenir en compte tot el referent a l'empaquetament i la instal·lació de les aplicacions. Però caldrà tenir en compte tot el que s'ha anat decidint pel que fa a totes les fases anterior. La fase de finalització i transferència haurà de tenir en compte aspectes com el lliurament del producte o aplicació desenvolupada al client, la manera d'instal·lar-lo als usuaris, la seva formació i els diferents assessoraments que caldrà fer.

Les proves unitàries són proves que s'encarreguen de l'avaluació i verificació mòdul a mòdul per separat d'una aplicació.

Aquesta fase quedarà, però, vinculada amb la resta de fases, ja que en totes hi haurà decisions per prendre relacionades amb aquesta darrera fase. La presa de requisits indica com serà el producte que vol i necessita el client. El disseny explicita com haurà de ser aquest producte. El desenvolupament prepara el producte. A l'hora d'empaquetar el producte i crear els instal·tables i desinstal·tables es tindran en compte les accions i decisions preses a totes les fases anterior.

Hi ha altres accions per dur a terme després de la finalització i transferència. Hi ha autors que inclouen aquestes noves tasques (com el manteniment de l'aplicació) dintre de les tasques o fases del projecte. Altres consideren aquestes accions com a nous projectes per si sols.

També cal tenir en compte els diferents tipus d'aplicacions informàtiques que es poden crear. Serà diferent la manera de finalitzar i transferir un producte si aquest s'executarà en local en una única màquina, si es trobarà instal·lat en un servidor dins una empresa amb accessos locals de tipus Winforms o si s'haurà desenvolupat en un entorn client-servidor amb accessos remots des d'un navegador web. Tots aquests tipus d'aplicacions necessiten paquets i biblioteques diferents, i la forma d'instal·lació és completament diferent. Es tracta d'un altre exemple per indicar com les decisions de les fases inicials influiran, i molt, en la fase de finalització i transferència.

ERP és l'acrònim en anglès *d'enterprise resource planning*, és a dir, sistemes de planificació de recursos empresarials.

Cal també diferenciar la manera de finalitzar una aplicació i generar els arxius per a la instal·lació si l'aplicació és una aplicació genèrica dús massiu (com pot ser un paquet d'ofimàtica o un programari genèric de gestió de correu que es penjarà en un portal web perquè els usuaris anònims el baixin i instal·lin en massa) o si l'aplicació desenvolupada està implementada a mesura de l'usuari final per a un entorn concret. També és pot donar un cas híbrid, com per exemple en el cas d'alguns ERP que disposen d'una versió bàsica per ser instal·lada, però necessitaran parametritzacions i adaptacions a l'organització que l'hagi adquirit.

1.2 Components d'una aplicació. Empaquetament

Per comprendre correctament la manera de distribuir una aplicació informàtica cal conèixer abans quins són els elements que la componen i les alternatives d'empaquetament i distribució que hi ha.

IEEE és l'acrònim de l'anglès *Institute of Electrical and Electronics Engineers*, és a dir Institut d'Enginyers Elèctrics i Electrònics.

Segons l'IEEE, el programari o aplicacions informàtiques o **software** és aquell conjunt de programes de càlcul, procediments, regles, documentació i dades associats que formen part de les operacions d'un sistema de computació.

Les aplicacions informàtiques, programari o *software* estan compostes per molts components del programari que és necessari que siguin executats conjuntament perquè funcionin correctament.

Un objecte precompilat i amb interfícies gràfiques d'usuari ben definides que estigui preparat per ser utilitzat en diferents entorns es pot denominar, en molt casos, com a *component del programari*.

Els components podran estar preparats per ser utilitzats (executats) directament pels usuaris finals, o es poden crear per intercomunicar-se amb altres components del programari.

Es pot considerar un **component del programari** tot recurs, independent d'altres, que hagi estat desenvolupat per a una determinada tasca i que pot formar un entorn per ser executat per resoldre un procediment definit de manera única o en conjunt amb altres components.

Cada component serà independent dels altres tant en el seu desenvolupament, en la seva estructura, en la seva tecnologia i en la seva arquitectura, com en la seva execució i resolució de processos.

A partir dels components del programari cal revisar altres components relacionats, que ajudaran a entendre el procediment de la distribució, com:

- Biblioteques (“llibreries”).
- Entorn de treball (*framework*, conjunt de components).
- Paquets de programari.
- Assemblatge o *assembly*.

Hi ha tres tipus d'arquitectura de programari: monolítica, client-servidor i arquitectura de tres nivells.

1.2.1 Biblioteques

Les **biblioteques** (habitualment anomenades “llibreries” per la similitud amb el terme anglès *library*) són un conjunt de programes o arxius que s'utilitzaran per al desenvolupament d'aplicacions informàtiques.

Les biblioteques disposen de funcionalitats concretes ja implementades que faciliten molt la feina als desenvolupadors. Moltes de les funcions que ofereixen les biblioteques ofereixen solucions per a la comunicació de les aplicacions amb l'entorn en el qual es desenvoluparan. Moltes de les biblioteques són específiques per a un determinat sistema operatiu, i implementen la majoria dels serveis del sistema.

Les biblioteques que s'hagin utilitzat en el desenvolupament del programari s'hauran d'incorporar a la distribució d'aquest, i passaran a ser un component més.

1.2.2 Entorn de treball

Un entorn de treball (*framework*) no és ben bé un component, sinó que més aviat és un conjunt de components, i quelcom més. Conceptualment, la paraula *framework* fa referència a un conjunt de termes, criteris i processos que serveixen

com a referència per resoldre nous problemes semblants a un tipus de problemàtica particular plantejada.

La definició d'**entorn de treball**, en un àmbit de desenvolupament del programari, ofereix una arquitectura ben definida amb diversos components de programari, a partir de la qual un altre projecte de programari pot ser desenvolupat.

Un entorn de treball pot estar compost de biblioteques, components o paquets de programari, altre programari i un llenguatge de programació. Tots aquests components oferiran suport per al desenvolupament o integració d'altres components o programaris. Ofereix també una estructura i una metodologia de treball, la qual amplia o utilitza les aplicacions del domini.

A partir del disseny de les classes abstractes i segons les regles del joc definides, l'entorn de treball proporcionarà l'arquitectura necessària per al desenvolupament del programari. El desenvolupador farà una aplicació a partir de subclasses i component d'instàncies a partir de les classes definides per l'entorn de treball.

Quines diferències es poden trobar entre un entorn de treball i una biblioteca?

La biblioteca contindrà mètodes que seran cridats des del codi del programari desenvolupat. En canvi, en un entorn de treball, és el codi d'aquest el que crida el codi desenvolupat al programari.

Un entorn de treball és una aplicació semicompleta, de control invertit.

Un entorn de treball acostuma a incorporar:

- Suport a programari.
- Facilitats per estalviar al programador detalls tecnològics de baix nivell, perquè dediqui més esforços a les regles de negoci i disseny del programari.
- Programari per desenvolupar i unir diferents components d'un projecte de desenvolupament de programes.
- Biblioteques.
- Facilitats per desenvolupar programari.
- Llenguatge interpretat.

1.2.3 Empaquetament

El tercer tipus de concepte per vincular amb els components de les aplicacions són els paquets de programari.

Un **paquet de programari** és un conjunt d'aplicacions que es distribueixen conjuntament. Aquestes aplicacions es necessiten unes a altres per desenvolupar de manera correcta les funcionalitats d'un programari.

Però per entendre correctament aquest concepte també cal fer referència a la distribució del programari. I per a això encara manca parlar d'un altre concepte, que és el de l'empaquetament d'aplicacions.

Les aplicacions informàtiques es poden distribuir en forma de paquets. Aquesta divisió és el que es coneix com a **empaquetament d'aplicacions**.

Un **paquet de programari** és un concepte que s'utilitza en el desenvolupament d'aplicacions informàtiques, tant en la programació orientada a objectes per anomenar un grup de classes o objectes relacionats entre si, com en la programació modular per referir-se a un mòdul o component que es pot integrar en el programa principal.

El concepte **software bundle** o *application bundle* fa referència a la distribució d'aplicacions informàtiques en forma de paquets. Els paquets estan formats per les biblioteques de les quals depenen, per programes executables i altres tipus de fitxers necessaris per al desenvolupament correcte de l'aplicació implementada (àudio, imatges, fitxers annexos, traduccions...). Tot aquest conjunt de fitxers que formen un paquet és lliure com un únic arxiu. D'aquesta manera l'usuari o client té l'oportunitat d'instal·lar l'aplicació a partir d'un únic arxiu que els va cridant i ubicant tots, i els porta a les diferents carpetes necessàries per al desenvolupament correcte, sense necessitat de cap altra acció per part de l'usuari (o petites accions per prendre decisions no decisives referents a la parametrització de la instal·lació).

Cal tenir en compte també el que succeeix amb les desinstal·lacions d'aplicacions. Si aquesta fa servir alguna biblioteca compartida per altres aplicacions o bé pel sistema operatiu, l'usuari es pot trobar que després de desinstal·lar-la el sistema operatiu no funcioni correctament o que alguna altra aplicació hagi deixat de funcionar.

Les versions de les aplicacions informàtiques i les seves actualitzacions són altres punts en què hi pot haver problemes. Potser dues aplicacions diferents estan fent servir versions diferents d'una mateixa biblioteca. Aquestes versions poden interferir entre si i provocar errades de funcionament d'un o més programes.

Per solucionar aquests dos problemes (de desinstal·lacions i de versions) és molt convenient fer que les aplicacions que es distribueixin estiguin empaquetades, amb el programa executable, amb totes les biblioteques vinculades (a excepció de les que es troben al sistema operatiu) i els altres fitxers. En el moment d'actualitzar les versions, es farà de tots els arxius relacionats amb aquell paquet. En el moment de desinstal·lar, es farà de tot el vinculat amb aquell paquet.

Una altra solució possible és fer servir un **gestor de paquets**. Aquests gestors s'encarreguen de les dependències dels arxius relacionats amb una mateixa aplicació informàtica. Són els responsables de mantenir les versions o gestionar

les baixades, però sempre amb la garantia de deixar estables tots i cadascun dels paquets que estiguin gestionant. Moltes vegades un gestor de paquets no és necessari, en gestionar el mateix paquet totes aquests processos.

Els empaquetaments d'aplicacions informàtiques presenten una sèrie d'avantages:

- Evitar els problemes de dependències dels programaris a l'hora d'instal·lar (o desinstal·lar) i d'explotar les aplicacions o parts d'aquestes. Estalviarà a l'usuari haver de cercar una o altra biblioteca o paquet o aplicació per poder tenir un funcionament i explotació correctes de l'aplicació que està instal·lant, cosa que algunes aplicacions actuals obliguen a fer als usuaris en instal·lar-les.
- Millorar i facilitar el procés d'instal·lació de les aplicacions desenvolupades.
- Permet una vinculació senzilla de diferents parts dels programes, com són les biblioteques.
- Oferir la possibilitat de traslladar aplicacions d'una màquina a una altra sense necessitat de reinstal·lació, en portar els paquets tots els fitxers i dades necessaris per a la seva execució.
- Oferir als usuaris una manera molt senzilla d'instal·lar les aplicacions, en portar els paquets tot el necessari per a una instal·lació automàtica.

Per contra, es pot trobar, com a possible inconvenient, el fet que els paquets tinguin una ocupació molt més gran de disc de la que tindria un arxiu instal·lable només amb el codi desenvolupat, sense la resta d'arxius. De totes maneres, aquest inconvenient és com més va més insignificant per l'evolució a la baixa dels preus de l'espai dels disc i per la capacitat més gran de distribuir arxius, amb una mida considerable, per mitjà d'Internet.

Un paquet d'una aplicació específica serà la millor forma de distribució del programari, en trobar-se ja compilat i configurat per a la instal·lació. Aquestes distribucions les poden donar directament els fabricants del programari (a partir d'enllaços des dels seus portals a Internet o per mitjà d'enviaments físics de discs o CD amb el programari) o es poden donar a partir de terceres empreses distribuïdores de programari.

És força habitual trobar-se amb un programari que no es troba empaquetat per a un sistema operatiu concret o per a una distribució d'aquest. En el cas de disposar del codi font, qualsevol usuari pot crear el seu empaquetament d'una aplicació, i cal que compili el codi i faci servir un generador de paquets. Caldrà, això sí, que tingui accés a totes les biblioteques i altres arxius necessaris per a la creació correcta del paquet. No hi ha un únic procediment per a la creació de paquets. Aquest procés serà diferent en funció del sistema operatiu, del llenguatge de programació que s'utilitzi, del generador de paquets...

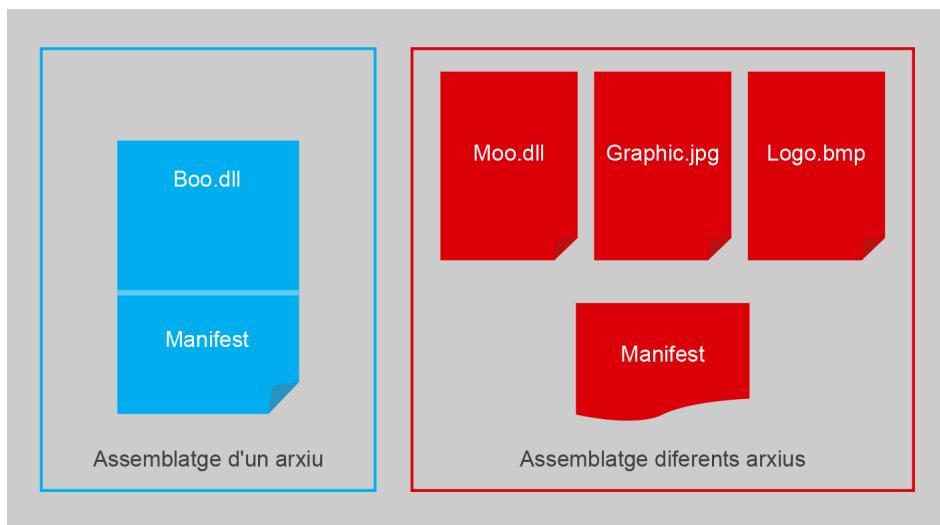
1.2.4 Assemblament o assembly

Un altre tipus de component de les aplicacions, aquesta vegada específic de les tecnologies .NET, són els *assembly* o assemblaments.

El codi, una vegada desenvolupat i compilat, s'ha d'empaquetar en una unitat funcional denominada *assemblament* abans de poder ser executat pel Common Language Runtime (figura 1.2).

El **Common Language Runtime (CLR)** és un component de la màquina virtual de la plataforma .NET. En aquest tipus de plataformes s'ha de diferenciar entre el temps de compilació i el temps d'execució. En el temps d'execució, el CLR converteix el codi per ser comprensible per al sistema operatiu, facilitar les compilacions i execucions posteriors i accelerar aquest procés.

FIGURA 1.2. Assemblament d'arxius

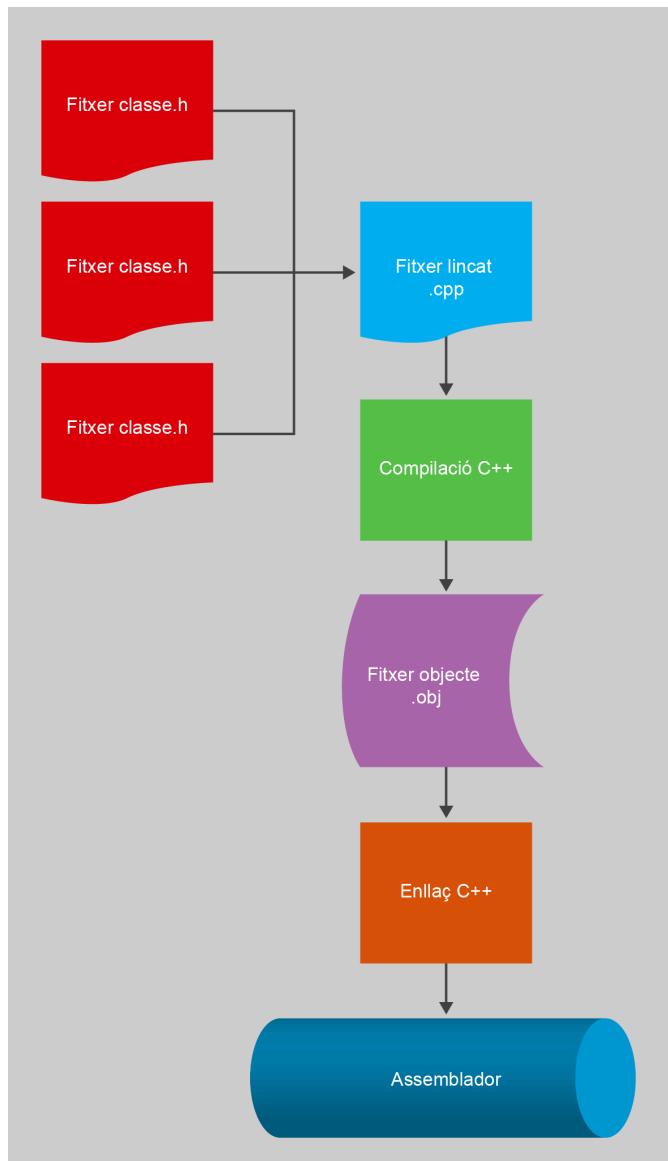


Un *assembly* (concepte utilitzat en la plataforma .NET) contindrà, en un únic arxiu o en diversos, el codi i els arxius necessaris per a una aplicació informàtica. L'*assembly* formarà una biblioteca de codi parcialment compilat.

El paquet generat (anomenat *assembly*) serà reutilitzable, versionable i autodescriptiu d'una aplicació de Common Language Runtime, en un entorn en temps d'execució que proporciona la plataforma .NET i que executa el codi i proporciona serveis que faciliten el procés del desenvolupament de l'aplicació.

L'*assembly* podrà ser utilitzat per diferents aplicacions.

A la figura 1.3 es pot observar l'arquitectura de l'assemblament per a un projecte desenvolupat en C++.

FIGURA 1.3. Assemblament d'arxius

1.3 Tipus d'empaquetaments

Hi ha molts tipus de paquets i moltes maneres d'empaquetar aplicacions. Aquestes es poden classificar en funció dels sistemes per als quals s'han desenvolupat o les característiques dels paquets. En aquest apartat es tractaran alguns del tipus d'empaquetaments que hi ha.

Hi ha paquets per a sistemes operatius concrets, com poden ser Windows (**MSI**), Linux (**DEB**, **RPM**), MAC OS (**PKG**) o Solaris (**SysV**). Altres paquets es podran fer servir amb diferents sistemes operatius (**PKG**).

També és poden distingir en funció de si s'han desenvolupat per a un programari lliure o per a un programari de propietat. En el cas dels paquets que es poden utilitzar amb diferents sistemes operatius, aquests compleixen les característiques

del programari lliure. S'hauran desenvolupat els paquets sota llicències compatibles i similars. Aquest tipus de paquets es podran combinar i distribuir fent servir aplicacions específiques per configurar les diferents parametritzacions en funció de les dependències i característiques específiques de cada sistema. Aquestes aplicacions es coneixen com a *sistemes d'empaquetament*.

Altres tipus d'empaquetaments són els referents als sistemes de propietat. Alguns exemples per a aquests casos poden ser el gestor de paquets d'HP-UX, Software Distributor, el format **SysV**, de Solaris, o el **framework .NET** de Microsoft.

Hi ha alguns sistemes basats en paquets binaris. Entre aquests podem trobar **deb** o **dpkg** (Debian), **rpm** (Red Hat), **tgz** (Slackware Linux, la distribució més antiga que hi ha de Linux) o **fink** (MAC OS X).

Altres sistemes generen paquets des del codi i altres són híbrids que poden utilitzar sistemes de generació i instal·lació de programari des de codi font o des de fitxer binari (com **FreeBSD**, **MacPorts** o **pkgsrc**).

En resum, es pot veure com hi ha molts tipus d'empaquetaments en funció del sistema operatiu amb el qual es treballi o de l'arquitectura. A continuació es detallaran alguns paquets de programari i de distribució, com:

- DEB
- RPM
- MSI
- EXE
- JAR
- PKG

1) **DEB (Linux):** és l'extensió dels paquets de programari de la distribució Debian GNU/Linux i les derivades seves, com Ubuntu. Un paquet de tipus DEB és un arxiu o un conjunt d'arxius estàndards de UNIX que porten dos tipus d'arxius, un amb la informació de control i un altre amb la informació de les dades. Aquests dos arxius són de tipus *tar* i es troben en format *qzip*, *bzip2* o *lzma*.

2) **RPM (Linux):** és un format d'arxiu usat en GNU/Linux. És un tipus de paquet desenvolupat originalment per a les distribucions Red Hat. RPM Package Manager és una eina d'administració de paquets. Actualment és un format de paquets utilitzat per moltes distribucions GNU/Linux, com SuSE, Mandriva o Fedora. Igual que els paquets DEB, un RPM també pot ser transformat i portat a altres sistemes operatius.

Un paquet de tipus DEB pot ser transformat en altres formats de paquets.



Logo de Slackware Linux



Debian és una comunitat de desenvolupadors i usuaris que han desenvolupat i mantenen una distribució de GNULinux.

Alien és una aplicació que permet la conversió entre diferents tipus de paquets.



Logo d'RPM

3) MSI: hi ha diverses eines per a la creació de paquets del tipus MSI o EXE, pensats per a sistemes operatius Windows. Algunes d'aquestes eines són programari de propietat i altres són programari lliure.

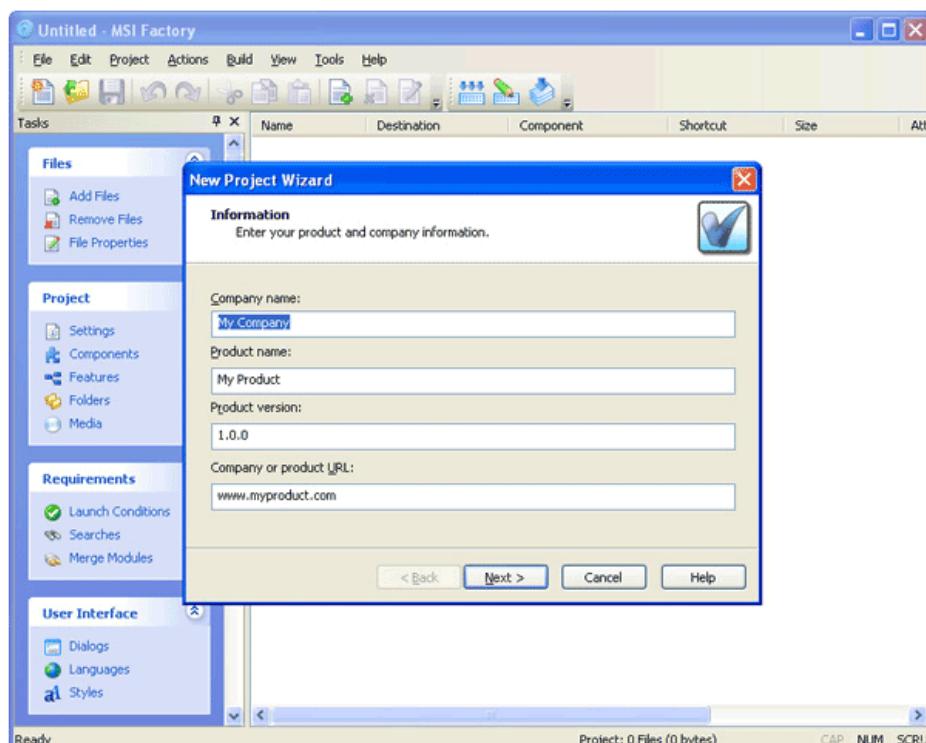
S'enumeren a continuació algunes d'aquestes eines. Com a exemples de programari de propietat:



Logo d'MSDN

1) Microsoft Visual Studio: és una eina integrada de desenvolupament de programari per a llenguatges .NET que permet crear paquets instal·lables. Està desenvolupada per Microsoft.

2) Windows Installer Development Tools: eina inclosa en l'MS Windows SDK per a desenvolupadors. Permet la creació de fitxers MSI i EXE. És una eina sense cost.

FIGURA 1.4. MSI

Logo d'MSI Package Builder

3) MSI Factory: eina basada en Windows Installer XML per a la creació de paquets MSI. Eina de programari de propietat (figura 1.4).

4) MSI Package Builder: eina de creació de paquets de tipus MSI a partir d'altres tipus de paquets. És una eina de propietat.

5) 7-zip: aplicació semblant al Winrar o al Winzip per a la gestió de fitxers i arxius. 7-zip és una aplicació de codi obert amb llicència GNU. Dóna l'opció d'obrir i accedir als arxius continguts per un fitxer de tipus MSI.

6) EXE: els arxius amb una extensió .exe són arxius executables. Històricament en sistemes operatius de Microsoft hi ha hagut una sèrie d'arxius executables, com els arxius .bat, els .com o els .exe. Una vegada que els sistemes operatius de Microsoft han arribat a un entorn gràfic els més utilitzats són els .exe i els .msi.



Logo de 7ZIP

Les diferències bàsiques són: els arxius .msi no són arxius executables, sinó que els obre l'Installer de Windows. Contenen totes les dades i arxius referents a la instal·lació d'una aplicació, es poden executar sense haver d'atendre a interaccions amb l'usuari, mostren només les informacions configurades i ofereixen l'opció de la recuperació. Un arxiu .exe no cridarà el Windows Installer a l'hora de ser executat, sinó que durà a terme comprovacions del sistema i executarà directament la instal·lació mostrant tots els detalls als usuaris.

7) JAR: aquest tipus d'arxiu és el que correspon a un paquet del llenguatge de programació Java. Un paquet en Java és un conjunt de classes relacionades entre si. Els paquets defineixen una jerarquia de directoris que permeten l'agrupació de les classes.

8) PKG (Mac OS X i paquets SVR4): Open Solaris i Sun Microsystems han creat un sistema de gestió de paquets per ser utilitzat en el sistema operatiu OpenSolaris. Aquest sistema de gestió de paquets és conegut també com a IPS o PKG. Aquest sistema també donarà suport al sistema de paquets SVR4, un tipus de paquets habituals en Solaris.

1.4 Paquets autoinstal·lables. Desinstal·lables

Hi ha molts tipus de paquets que es podran utilitzar en molts entorns diferents i amb moltes funcionalitats. Molts tenen com a objectiu oferir una manera d'instal·lar aplicacions o bé de desinstal·lar-les. Es coneixen com a paquets *per autoinstal·lar* o paquets *per desinstal·lar*.

Aquest paquets, a l'hora de crear-los, es poden parametrizar amb les opcions que el desenvolupador vulgui. Normalment un paquet preparat per executar una autoinstal·lació d'una aplicació no oferirà gaires oportunitats als usuaris. A tot estirar mostrarà la barra d'estat de la instal·lació o oferirà la possibilitat de seleccionar algun paràmetre molt concret. La resta ja haurà estat escollit en crear el paquet per a la instal·lació. Aquest paquet s'encarregarà d'ubicar tots els arxius que s'han de copiar al disc dur, i després l'aplicació els trobarà quan els necessiti en temps d'execució.

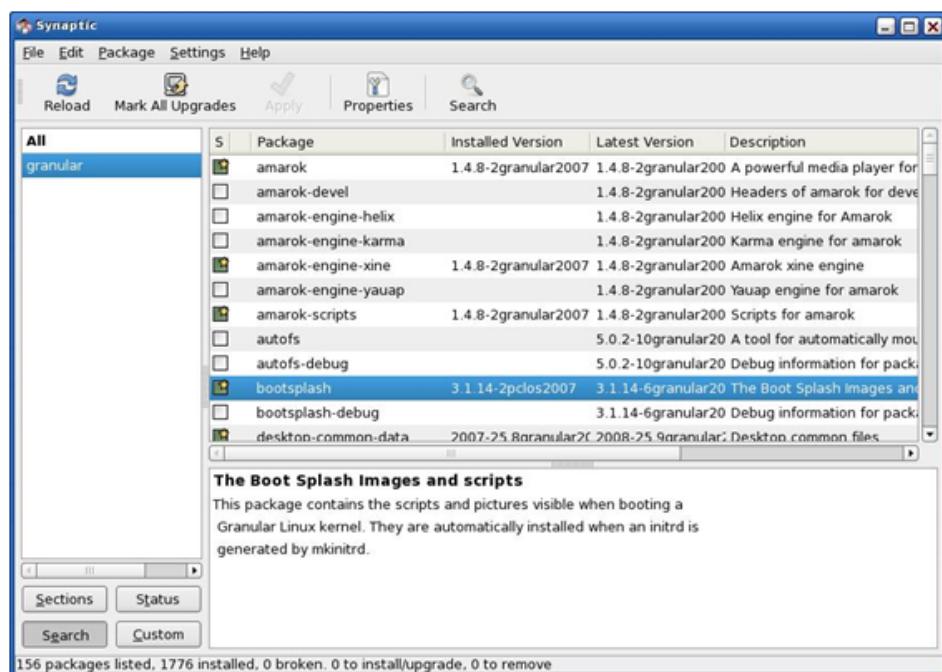
Passarà quelcom semblant amb els paquets que serveixen per desinstal·lar una aplicació. Serà millor desinstal·lar una aplicació fent servir les eines pròpies que ofereixen les aplicacions que fer-ho de manera directa. El nombre de components instal·lats i la seva ubicació (i la seva versió) és una informació molt difícil de controlar per a un usuari.

Per a la gestió dels paquets caldrà fer servir els **sistemes de gestió de paquets**. Aquests sistemes aporten un conjunt d'eines que serveixen per automatitzar el procés de gestió dels paquets de programari que es poden tenir al sistema. Això permetrà instal·lar-los, desinstal·lar-los, actualitzar-los, configurar-los, controlar-ne les versions...

Normalment una aplicació, si es vol distribuir i instal·lar en altres sistemes, quedarà agrupada en un únic arxiu: es distribueix en forma de paquets. En aquest arxiu, a més de les dades, biblioteques i arxius necessaris, hi haurà incorporada molta altra informació (es coneix com a *metainformació*). Aquesta pot agrupar informació important com dades referents a l'aplicació desenvolupada, descripció i documentació referent a la funcionalitat, a les versions dels paquets, al distribuïdor del programari...

Un exemple de sistema gestor de paquets és el programari Synaptic Package Manager, per a Linux. Es pot veure un exemple del seu funcionament a la figura 1.5.

FIGURA 1.5. Synaptic Package



Els sistemes gestors de paquets són més habituals en sistemes operatius Linux. Això ofereix més control del programari instal·lat. En canvi, en sistemes Windows tota aquesta gestió és menys transparent als usuaris, cosa que fa el sistema també menys controlable.

De fet, alguns sistemes de gestió de paquets formen part bàsica dels sistemes operatius basats en Linux o, fins i tot, en els Mac OS X. Però hi ha altres sistemes de gestió de paquets que són independents.

Un cas pot ser el de FINK, en els sistemes Mac, que permet portar codi desenvolupat per a UNIX a sistemes Mac, o l'entorn semblant a UNIX per a sistemes

Windows.

Fins i tot alguns llenguatges de programació interpretats tenen el seu propi sistema de gestió de paquets. En el cas del llenguatge PHP tenim PEAR un entorn de treball i sistema de distribució per a components PHP. En el cas de Perl s'ofereix CPAN. Per una banda, CPAN fa referència a la Comprehensive Perl Archive Network, una xarxa molt important de documentació i programari de Perl. Però també és un mòdul de Perl que serveix per baixar i instal·lar paquets i mòduls desenvolupats en Perl.



PEAR: PHP extension and application repository

Algunes funcionalitats dels sistemes de gestió de paquets són:

- Organitzar tots els paquets que es trobin al sistema.
- Mantenir-ne la usabilitat.
- Resoldre dependències entre mòduls per garantir que el programari funcioni correctament.
- Instal·lar, actualitzar i eliminar paquets.
- Comprovació de les firmes digitals.
- Agrupar paquets segons les seves funcionalitats.

Però poden tenir també alguns inconvenients, com poden ser:

- Necesiten un gestor de dependències extern.
- La BD que porten incorporada amb les informacions dels paquets pot no estar sincronitzada amb el sistema d'arxius.
- No permeten gestionar diverses versions de biblioteques o components.
- Molts paquets no es poden conèixer ni gestionar fins que no s'obren per instal·lar-los.

Si comparem un instal·lador amb un sistema de gestió de paquets podem trobar algunes diferències. Un instal·lador pot presentar les característiques següents:

- S'encarrega de la seva instal·lació pròpia.
- Cada producte tindrà el seu instal·lador propi.
- Només treballaran amb una aplicació.
- Hi pot haver molts distribuïdors i venedors per a una mateixa aplicació informàtica.

Un sistema de gestió de paquets ampliarà molt més les funcionalitats d'un instal·lador, sobretot amb la característica d'ofrir la possibilitat de treballar amb molts més paquets a la vegada, gestionant també les diferències entre aquests.

1.5 Eines per a la creació de paquets d'instal·lació

Una vegada que una aplicació ha estat desenvolupada, cal decidir la manera d'empaquetar-la o de crear l'arxiu autoinstal·lable que es distribuirà. En funció de les característiques que s'hagin decidit la distribució estarà destinada a un determinat tipus de sistema. Aquesta decisió haurà influït també en altres paràmetres com la selecció del llenguatge de programació o de l'entorn de desenvolupament.

Precisament la majoria d'entorns integrats de desenvolupament incorporen eines per facilitar la tasca de crear paquets que siguin autoinstal·lables i dels seus desinstal·ladors.

Exemples d'IDE que ofereixen aquesta característica poden ser:

- Visual Studio.
- Mono Develop.
- NetBeans.
- PHP Pear.

Però altres desenvolupaments de programari no oferiran aquestes alternatives. En aquests casos serà necessari fer servir un programari específic per a la creació dels paquets.

Algunes eines que es poden utilitzar en diversos sistemes operatius poden ser:



1) InstallBuilder: és una eina desenvolupada per BitRock i, com a tal, és una eina de propietat. La seva funcionalitat és la de crear instal·ladors vàlids per a diferents plataformes i sistemes operatius (com Linux, Windows, Mac OS X, Solaris i altres). Install Builder pot crear paquets de tipus RPM i DEB.

2) Install Anywhere: és una eina de propietat desenvolupada per Flexera Software. És una solució multiplataforma per crear instal·lables en diferents entorns, com poden ser Windows, Linux, Mac OS X, Solaris, AIX, HP-UX o IBM iSeries.

3) IzPack: és una eina lliure que permet als usuaris la creació d'instal·ladors molt senzills en la distribució de programari desenvolupat en el llenguatge de programació Java. Necessiten qualsevol sistema operatiu que tingui instal·lat la màquina virtual de Java. Crea un instal·lador únic (un únic arxiu) per a diferent plataformes, a més d'ofrir solucions de gestió de paquets i alternatives als instal·ladors per a plataformes específiques. L'IzPack també és pot integrar amb altres entorns integrats de desenvolupament com, per exemple, l'Eclipse.

A continuació es mostren algunes eines d'empaquetament específiques per a Windows:

1) Inno Setup: és una eina que es distribueix de manera gratuïta per a entorns Windows. És una eina molt estable i molt utilitzada per a la creació de molts

instal·lables a escala comercial. Suporta tots els sistemes operatius desenvolupats fins avui dia per Microsoft i dóna suport a les aplicacions desenvolupades per a sistemes en 64 bits. Ofereix la possibilitat de crear paquets MSI o arxius únics EXE i ofereix la possibilitat de crear instal·ladors multilingües. Aquesta eina ha estat desenvolupada en Delphi.

2) WiX: és una eina desenvolupada per Microsoft per a la creació de paquets en entorns Windows. El seu acrònim significa *Windows Installer XML*. Crea arxius MSI (Windows Installer) a partir de documents XML. Com a curiositat cal dir que WiX és la primera eina desenvolupada per Microsoft sota una llicència oberta, anomenada *Common Public License*.

3) InstallShield: és una eina de propietat desenvolupada, igual que Install Anywhere, per Flexera Software. És una eina que ajuda a la creació d'instal·lables a partir de programari desenvolupat amb la plataforma Visual Studio 2010.

Autopackage és un sistema de gestió de paquets destinat a fer simple la creació de paquets que es puguin instal·lar en distribucions Linux. Permet la creació de paquets RPM i DEB i suporta moltes de les distribucions més habituals de Linux.

Cal també tenir en compte la destinació de les aplicacions desenvolupades per seleccionar una eina adequada a les necessitats. Per exemple, si es vol crear un paquet per a una PDA, per a un iPhone o per a qualsevol dispositiu mòbil amb un sistema operatiu determinat, caldrà seleccionar una eina específica per a aquest tipus de processos.

InstallShield®



Logo d'Install Shield



Logo d'Autopackage

Les PDA (de l'anglès *personal digital assistant*) són dispositius mòbils que poden processar aplicacions.

1.6 Exemple de la creació d'un paquet de tipus .deb en LINUX

A continuació es mostra un exemple de creació d'un paquet de tipus .deb. Per això s'ha desenvolupat un programari que, donat un nombre, n'indiqui el factorial.

Per a això s'ha desenvolupat el codi en el llenguatge de programació C, que es pot veure a la figura 1.6.

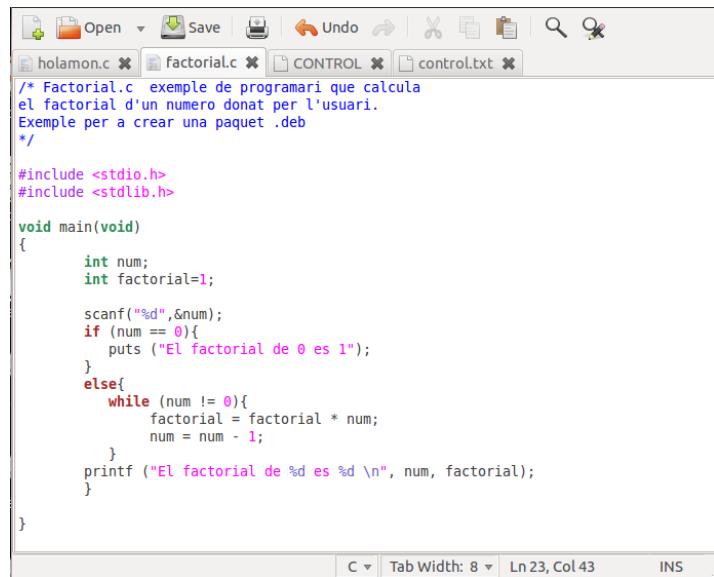
El codi en llenguatge C és:

```

1  /* Factorial.c exemple de programari que calcula
2   el factorial d'un nombre donat per l'usuari.
3   Exemple per crear una paquet .deb
4 */
5
6 #include <stdio.h>
7 #include <stdlib.h>
8
9 void main(void)
{
    int num;
```

```

12     int factorial=1;
13
14     scanf("%d",&num);
15     if (num == 0){
16         puts ("El factorial de 0 és 1");
17     }
18     else{
19         while (num != 0){
20             factorial = factorial * num;
21             num = num - 1;
22         }
23         printf ("El factorial de %d és %d \n", num, factorial);
24     }
25
26 }
```

FIGURA 1.6. Codi Factorial


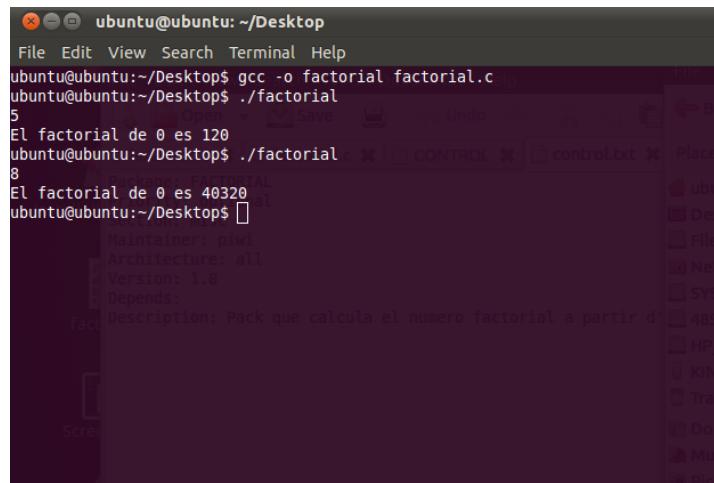
```

/* Factorial.c exemple de programari que calcula
el factorial d'un numero donat per l'usuari.
Exemple per a crear una paquet .deb
*/
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    int num;
    int factorial=1;

    scanf("%d",&num);
    if (num == 0){
        puts ("El factorial de 0 es 1");
    }
    else{
        while (num != 0){
            factorial = factorial * num;
            num = num - 1;
        }
        printf ("El factorial de %d es %d \n", num, factorial);
    }
}
```

Una vegada el codi es compila es genera un arxiu executable. Es pot veure a la figura 1.7 la compilació del codi i un parell d'exemples d'execució.

FIGURA 1.7. Creació DEB


```

ubuntu@ubuntu:~/Desktop$ gcc -o factorial factorial.c
ubuntu@ubuntu:~/Desktop$ ./factorial
5
El factorial de 0 es 120
ubuntu@ubuntu:~/Desktop$ ./factorial
8
El factorial de 0 es 40320
ubuntu@ubuntu:~/Desktop$ dpkg-deb -b .
ubuntu@ubuntu:~/Desktop$ ls
Maintainer: piwi
Architecture: all
Version: 1.0
Depends:
Description: Pack que calcula el numero factorial a partir d'un numero

```

A partir d'aquest exemple és vol crear un paquet de tipus .deb que permeti la instal·lació del programa en un sistema Linux.

A partir de l'arxiu executable factorial es vol crear un paquet .deb que instal·li l'arxiu en /usr/bin.

El primer que caldrà fer és crear un directori on es tindran els fitxers que es faran servir per a la creació del paquet. La instrucció que es farà servir, en el cas de crear, per exemple, un directori amb nom *deb*, és:

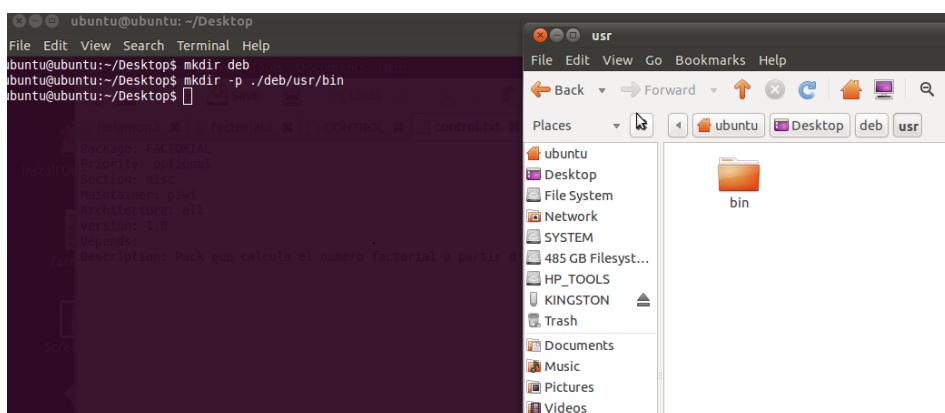
```
1 mkdir deb
```

Just a continuació cal crear un altre directori amb l'estrucció de destinació on es vol deixar instal·lat l'arxiu. Per això cal crear els directoris *usr* i *bin* dintre de *deb* i *usr*, respectivament; es pot fer amb l'ordre:

```
1 mkdir -p ./deb/usr/bin
```

A la figura 1.8 es pot veure l'execució de les dues ordres anteriors i la creació de les carpetes.

FIGURA 1.8. Creació de les carpetes

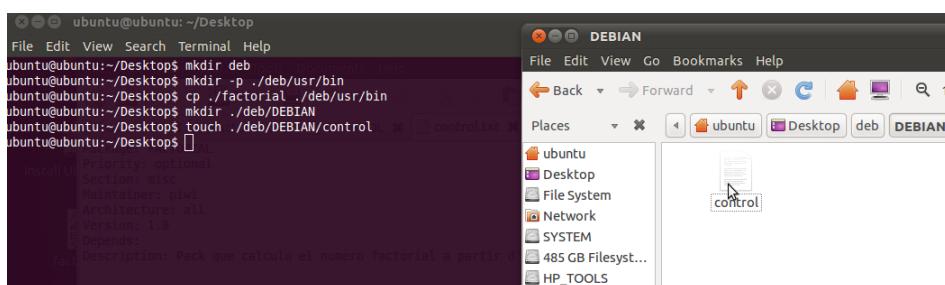


Caldrà crear un altre subdirectorí, amb el nom de *DEBIAN*, on s'hauria de crear un arxiu anomenat *control* que contindrà les indicacions per a la creació del paquet. Es fa amb les ordres:

```
1 mkdir ./deb/DEBIAN
2
3 touch ./deb/DEBIAN/control
```

A la figura 1.9 es pot veure la creació de la carpeta *DEBIAN* i la creació de l'arxiu *control*.

FIGURA 1.9. Arxiu Control

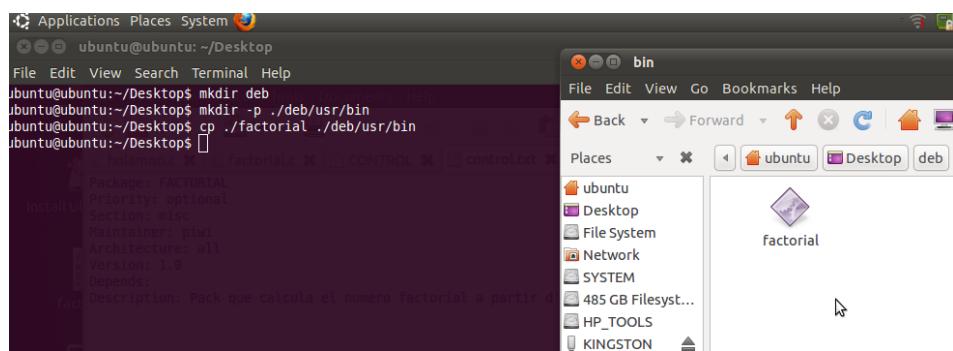


El pas següent és copiar els arxius necessaris per a la instal·lació al directori creat. En el cas tractat es tracta de copiar només l'arxiu executable (*factorial*) a la carpeta */deb/usr/bin*, amb l'ordre:

```
1 cp ./factorial ./deb/usr/bin
```

A la figura 1.10 es pot observar la còpia de l'executable *factorial* a la carpeta *bin*.

FIGURA 1.10. Còpia de l'executable



Ara caldrà configurar l'arxiu control amb els paràmetres necessaris per crear el paquet de tipus DEB.

```
1 Package: FACTORIAL
2 Priority: optional
3 Section: misc
4 Maintainer: piwi
5 Architecture: all
6 Version: 1.0
7 Depends:
8 Description: programa que calcula el nombre factorial a partir d'un nombre enter donat.
```

Aquest arxiu cal editar-lo amb un editor de text, però cal desar-lo amb el nom de *control* i sense extensió.

Una vegada tot això està preparat, només manquen dues passes més per tenir el paquet creat. En primer lloc cal modificar el propietari del directori creat, amb l'ordre:

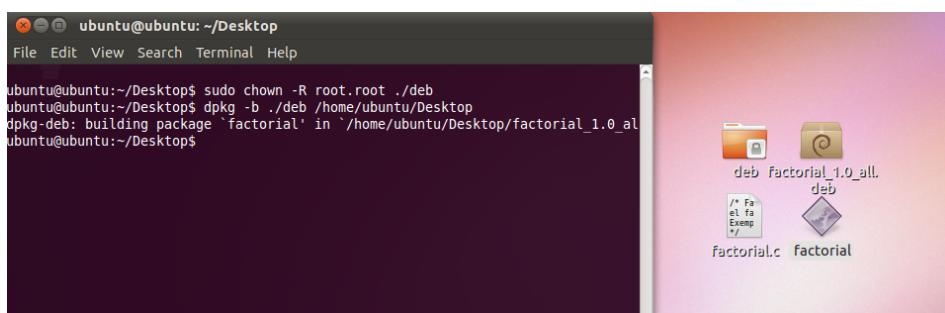
```
1 sudo chown -R root.root ./deb
```

Finalment es crearà el paquet amb format .deb.

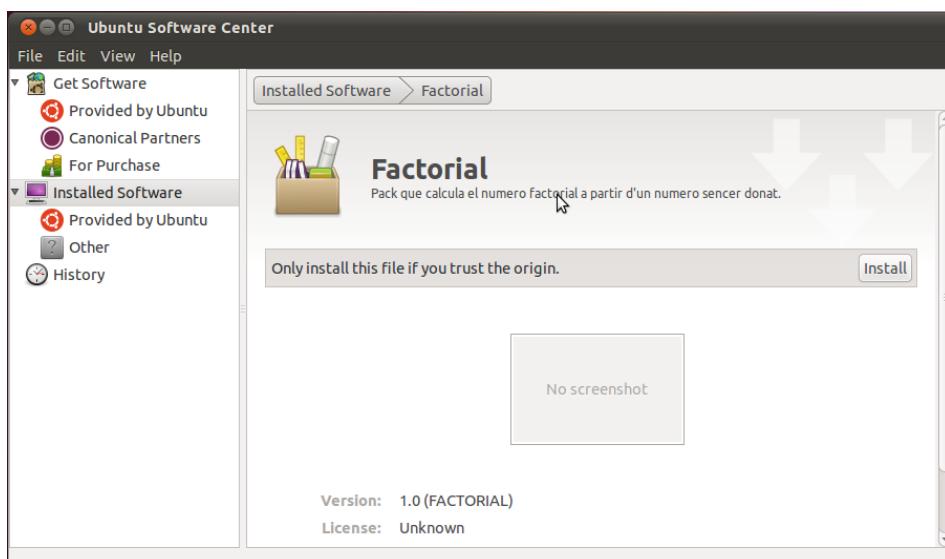
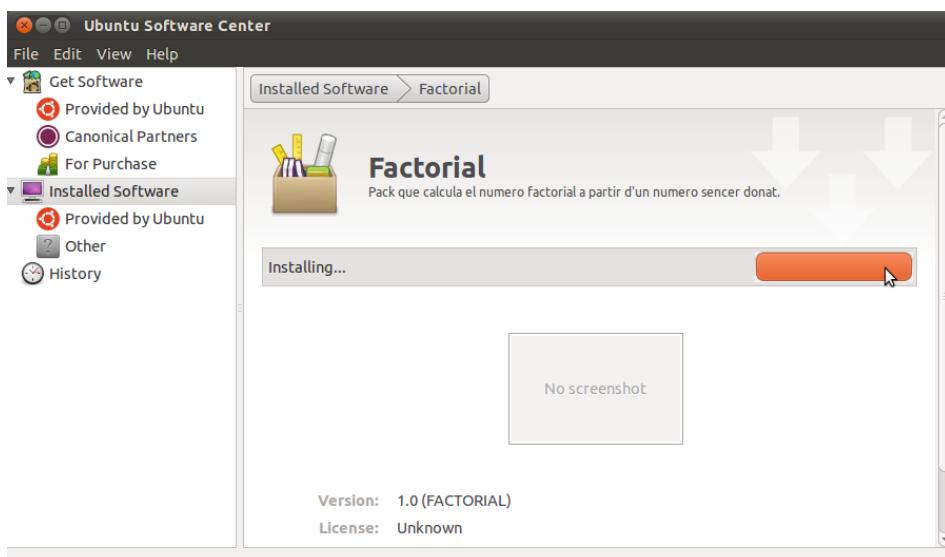
```
1 dpkg -b ./deb /home/ubuntu/Desktop
```

En el cas que ens ocupa el paquet .deb es crearà a l'escriptori.

A la figura 1.11 es veu com s'ha creat el paquet amb el nom *factorial_1.0_all.deb*.

FIGURA 1.11. Creació del paquet.

Una vegada creat el paquet, a les figures 12 i 13 es pot veure un exemple de l'execució amb la instal·lació que durà a terme.

FIGURA 1.12. Exemple d'execució.**FIGURA 1.13.** Continuació de l'exemple d'execució.

1.7 Personalització de la instal·lació. Interacció amb l'usuari

Als annexos es pot trobar un exemple de creació d'un paquet a partir de l'eina Inno Setup.

Ja sigui fent servir les eines que ofereixen els entorns integrats de desenvolupament o fent servir eines específiques de creació d'instal·lables o paquets hi haurà tota una sèrie de paràmetres que es podran personalitzar en funció de les necessitats o les decisions dels desenvolupadors.

Algunes d'aquestes decisions seran ofertes als usuaris en temps d'execució de l'arxiu instal·lable. Altres paràmetres podran ser escollits per part del desenvolupador, com, per exemple:

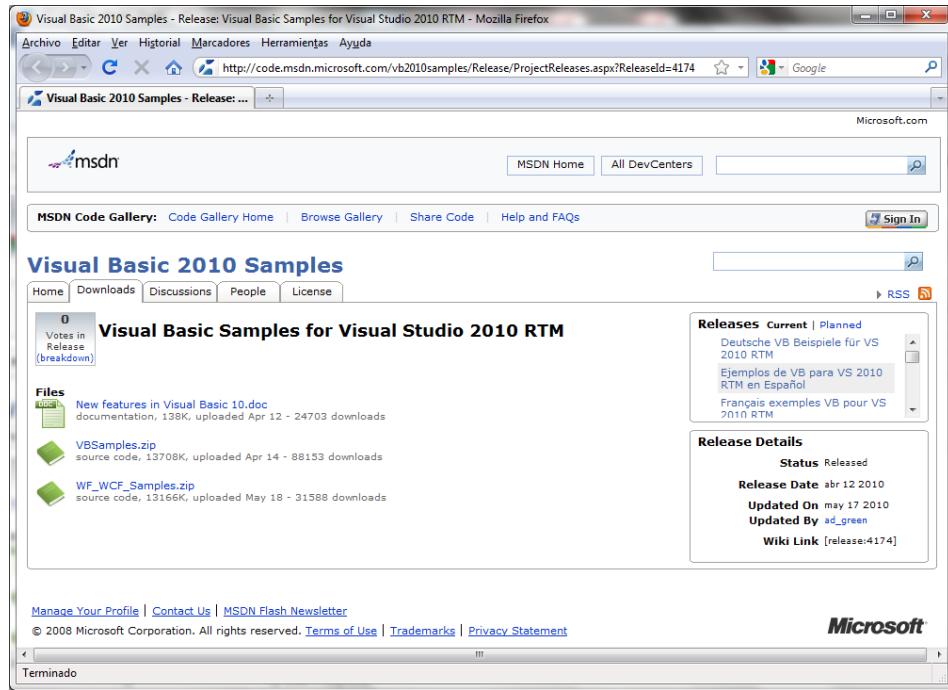
- Imatge de la icona del paquet.
- Diàlegs per establir amb l'usuari en temps d'instal·lació.
- Logos de l'aplicació (que es veuran durant la instal·lació).
- Barra d'estat de la instal·lació.
- Imatges de fons.

1.8 Exemple de la distribució d'una aplicació amb Visual Studio 2010

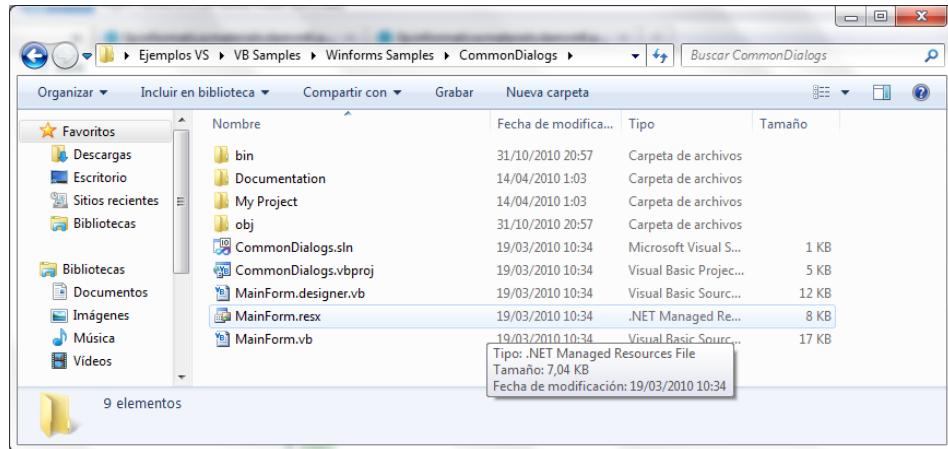
A continuació és representa un exemple de com cal empaquetar un projecte desenvolupat amb la plataforma .NET, concretament amb el Visual Studio 2010. A les figures següents es pot anar veient, pas per pas, com s'ha dut a terme el procés d'empaquetament i la decisió d'alguns dels paràmetres seleccionats.

En primer lloc hem d'implementar el codi que desenvolupi l'aplicació que es vol empaquetar. A la mateixa web que ofereix Microsoft es poden seleccionar exemples desenvolupats en diferents llenguatges de programació. S'ha seleccionat un exemple implementat en Visual Basic per a Visual Studio 2010.

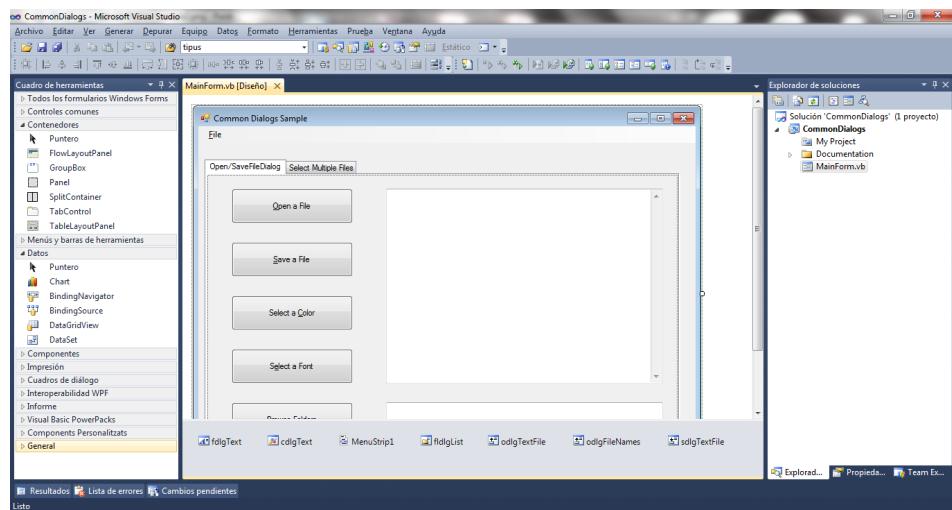
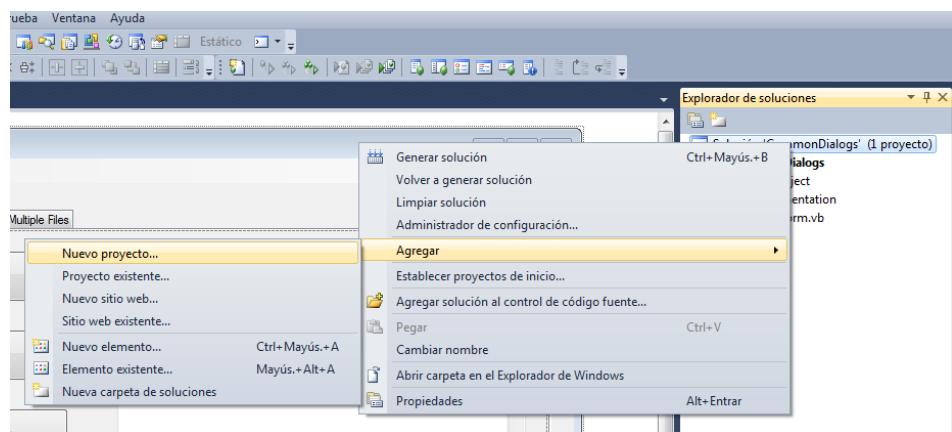
A les figures 14 i 15 es pot veure com se selecciona un exemple per al Visual Studio 2010 implementat en Visual Basic. Realment qualsevol exemple serviria per entendre aquest exemple d'empaquetament.

FIGURA 1.14. Selecció exemple en Visual Basic.

Aquest exemple es pot agafar de l'adreça <http://code.msdn.microsoft.com/vb2010samples>

FIGURA 1.15. Ubicació del codi descarregat.

A la figura 1.16 s'obre l'exemple de codi baixat i es poden observar les interfícies que conté i la seva execució.

FIGURA 1.16. Obrim l'exemple de codi descarregat.**FIGURA 1.17.** Agregació d'un nou projecte.

A la figura 1.17 i en la figura 1.18 es veu com s'ha d'agregar un nou projecte al projecte que ja tenim, concretament un projecte d'instal·lació. Per a això farem servir l'assistent per a aquest tipus de projectes, que s'obre a la figura 1.19.

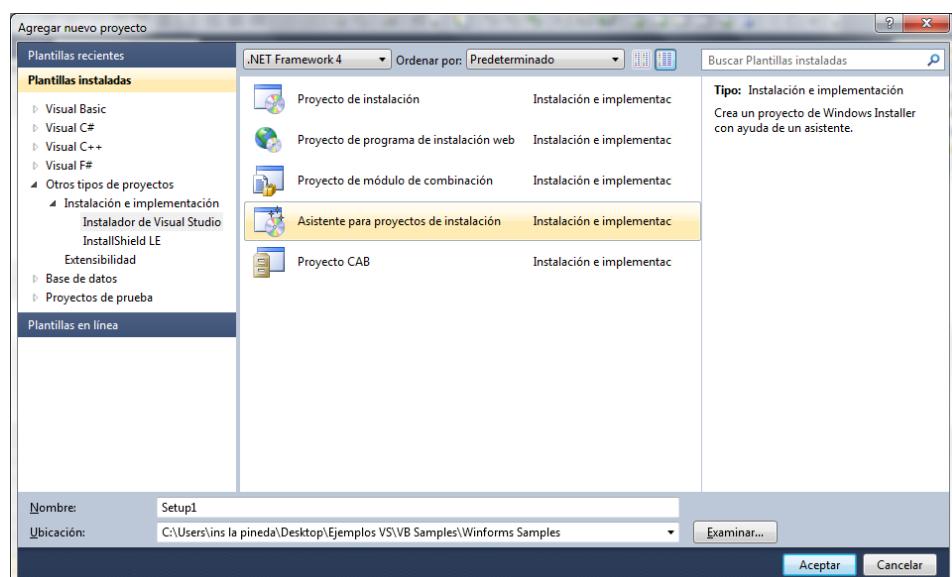
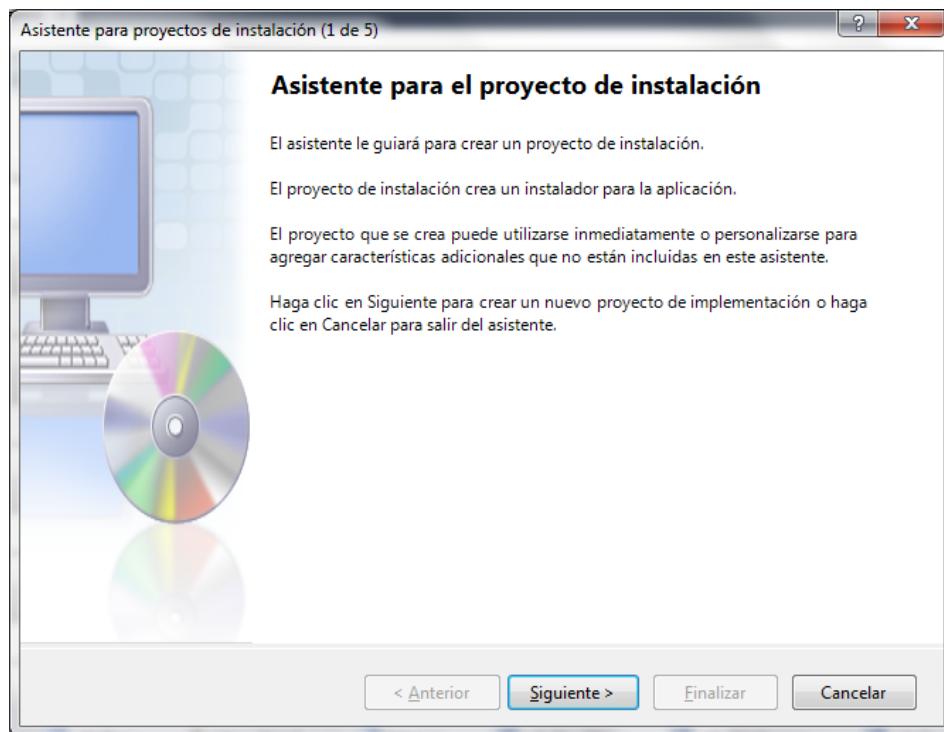
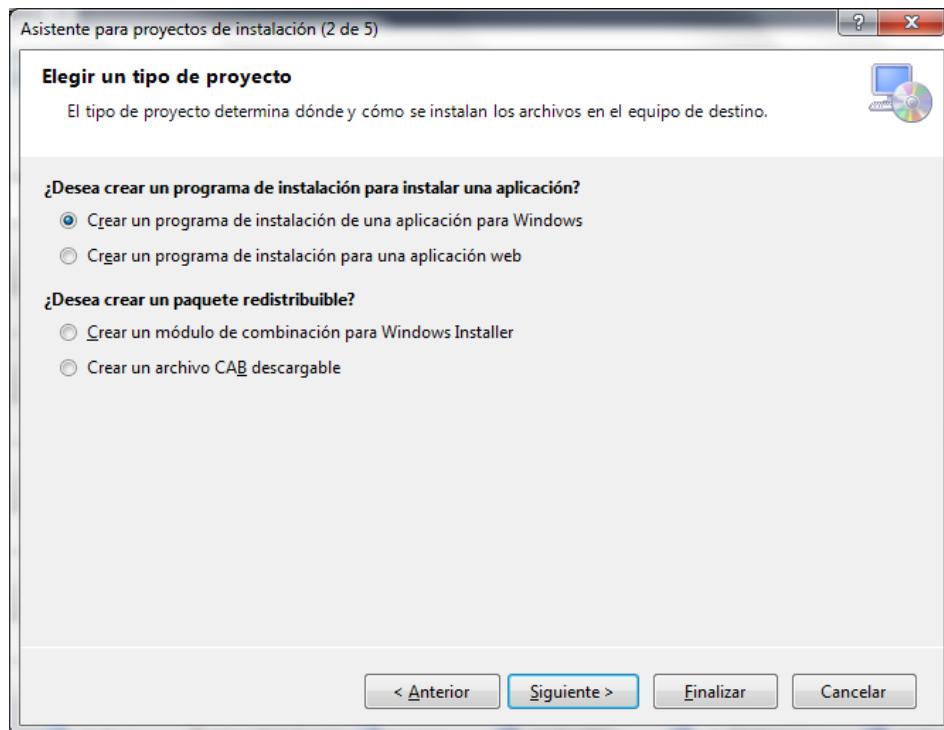
FIGURA 1.18. Projecte d'instal·lació.

FIGURA 1.19. Assistent.

Aquest assistent ofereix una sèrie de possibilitats. Com es veu a la figura 1.20, es voldrà crear un programa d'instal·lació per instal·lar una aplicació, concretament amb l'opció “Crear un programa d'instal·lació per a una aplicació per a Windows”.

En la figura 1.21 i en la figura 1.22 es veu com en els passos següents es podran escollir els resultats de projecte que es volen incloure. Escollirem les opcions:

FIGURA 1.20. Crear un programa d'instal·lació per a una aplicació per a Windows.

- *Recursos adaptats de Common Dialogs.*
- *Arxius de codi font de Common Dialogs.*
- *Arxius de documentació de Common Dialogs.*

FIGURA 1.21. Escolir els resultats de projecte.

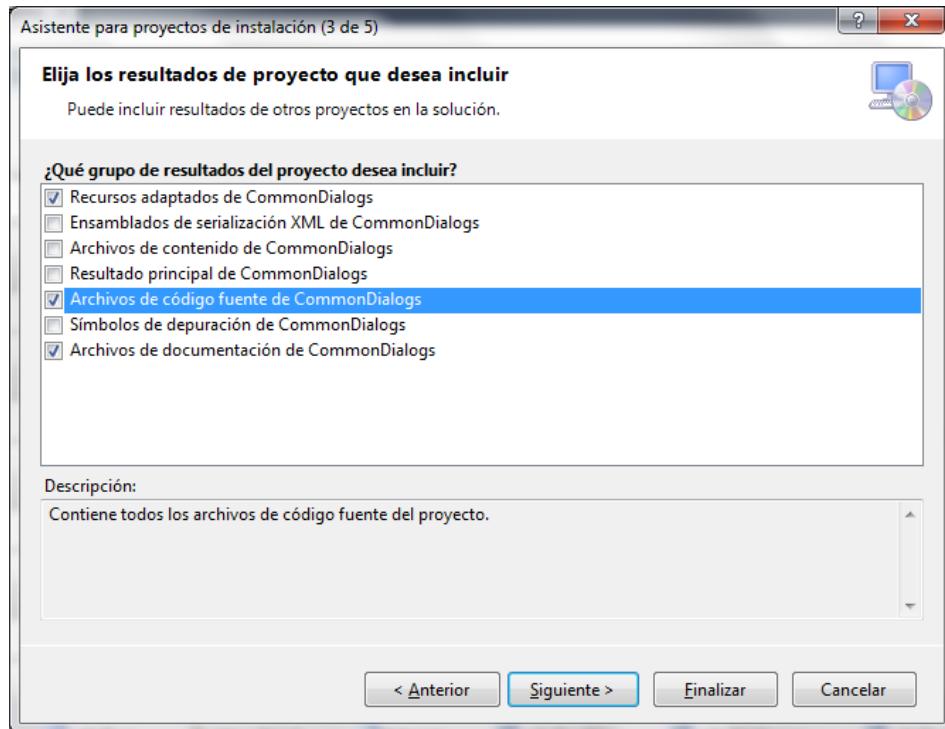
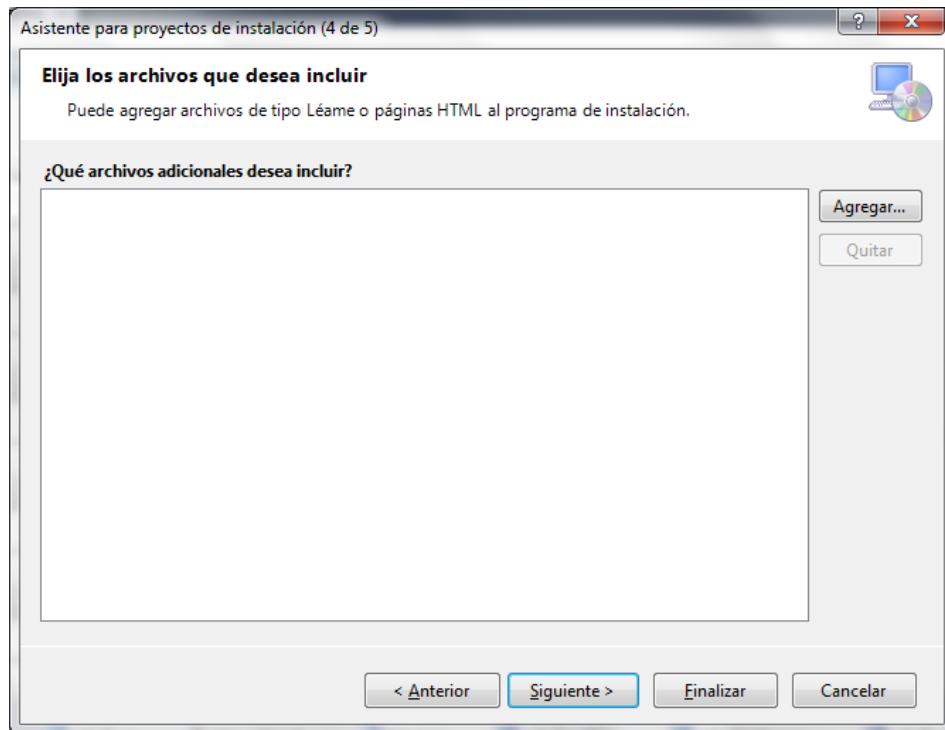
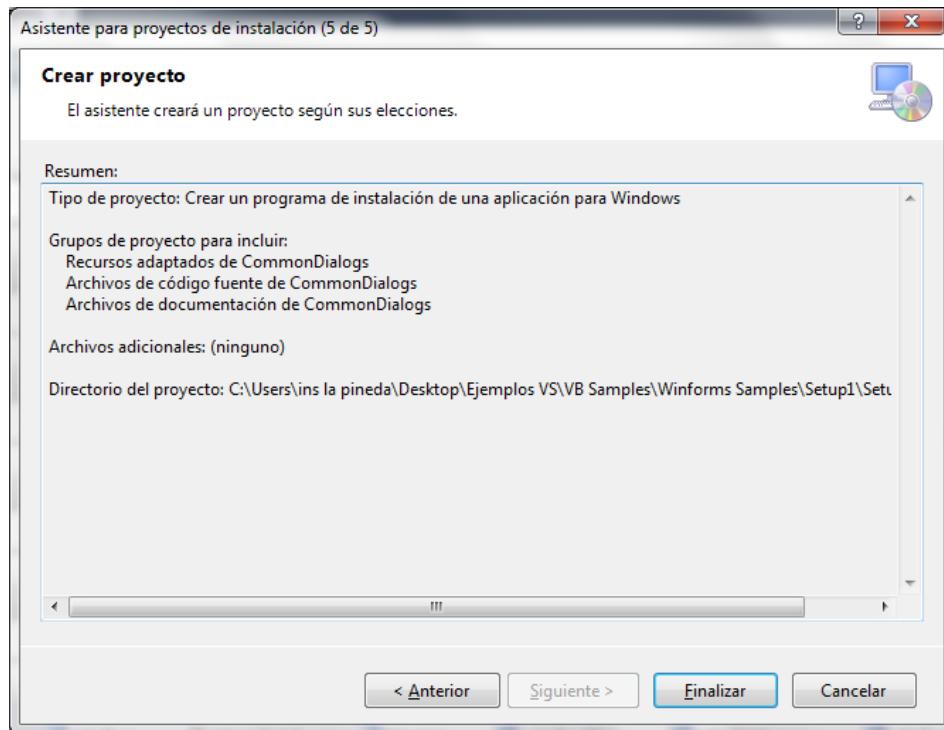


FIGURA 1.22. Escolir arxius addicionals.



En aquest cas no caldrà seleccionar cap altre arxiu addicional.

FIGURA 1.23. Confirmació de configuració.

A la figura 1.23 es pot veure com es crearà el projecte a partir de les decisions preses.

Tornant al projecte, a la figura 1.24 es veu com s'ha afegit aquest nou component com a *Setup1*, que és un instal·lador del codi que hem agafat com a exemple. Ara aquest instal·lador es podrà executar des d'un arxiu o des del Visual Studio mateix, com es pot veure a les figures 25 i 26.

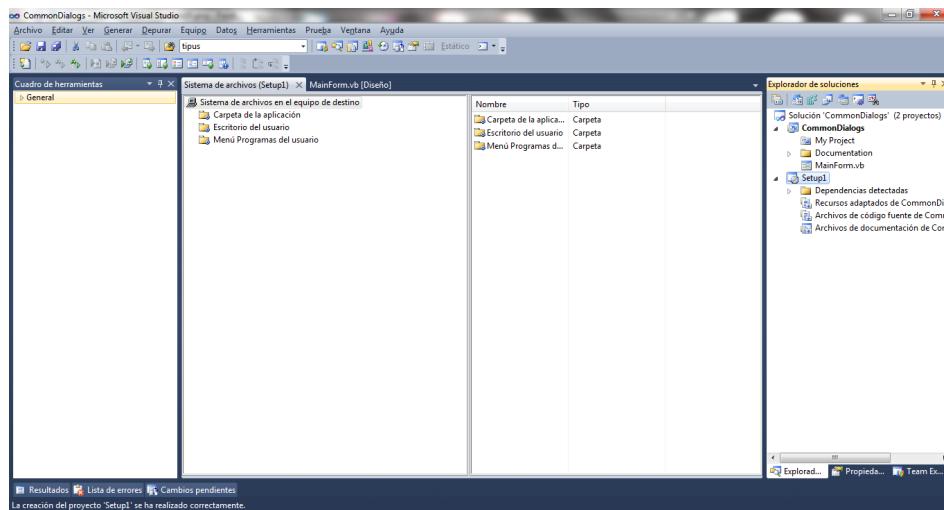
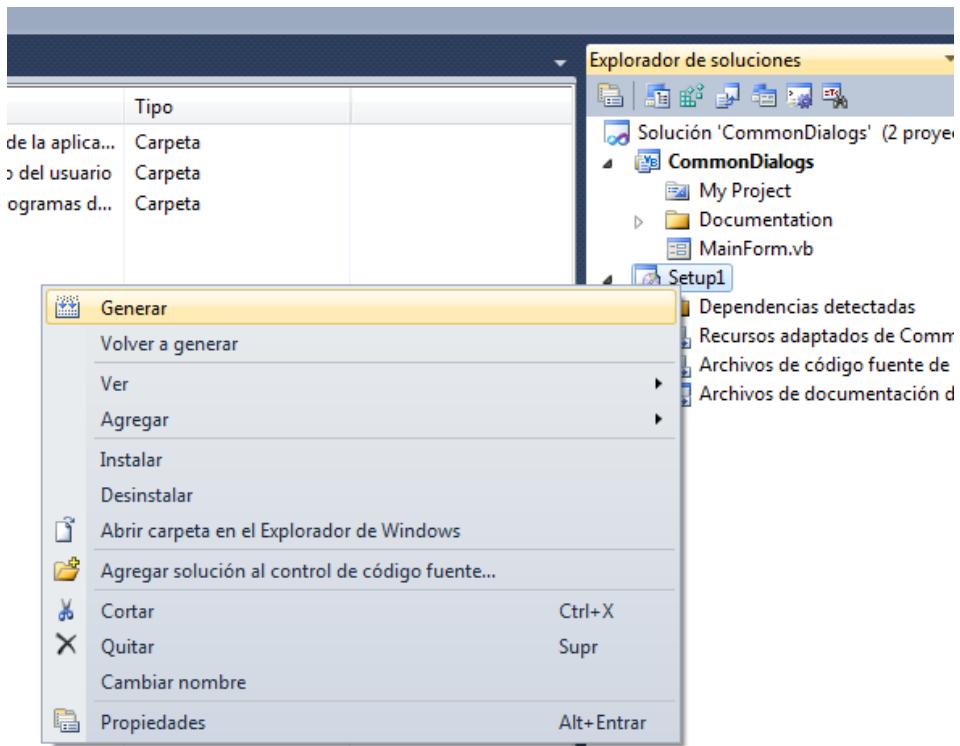
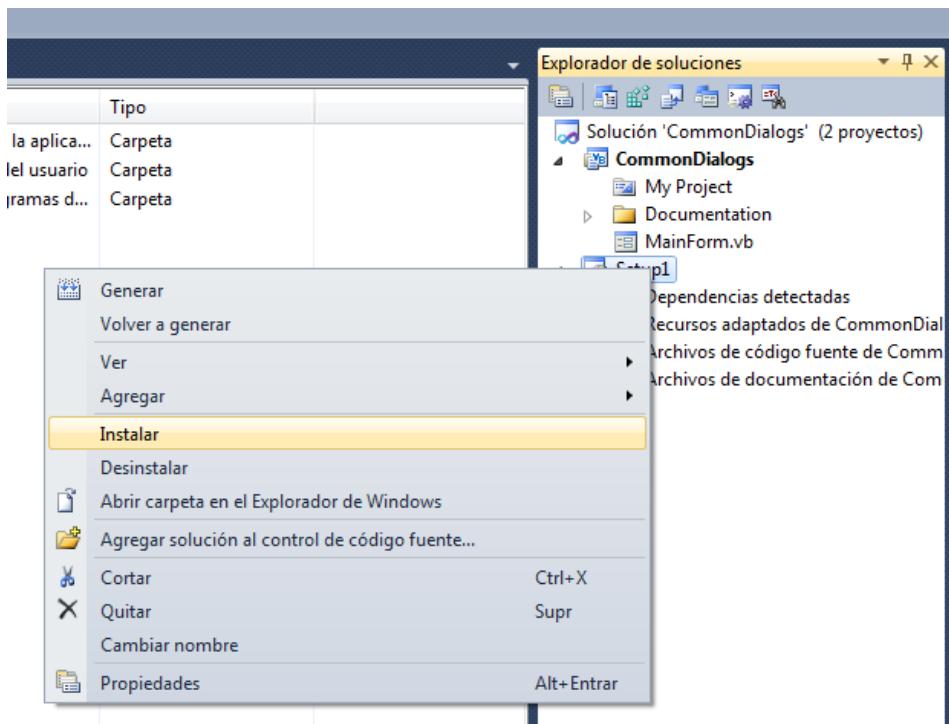
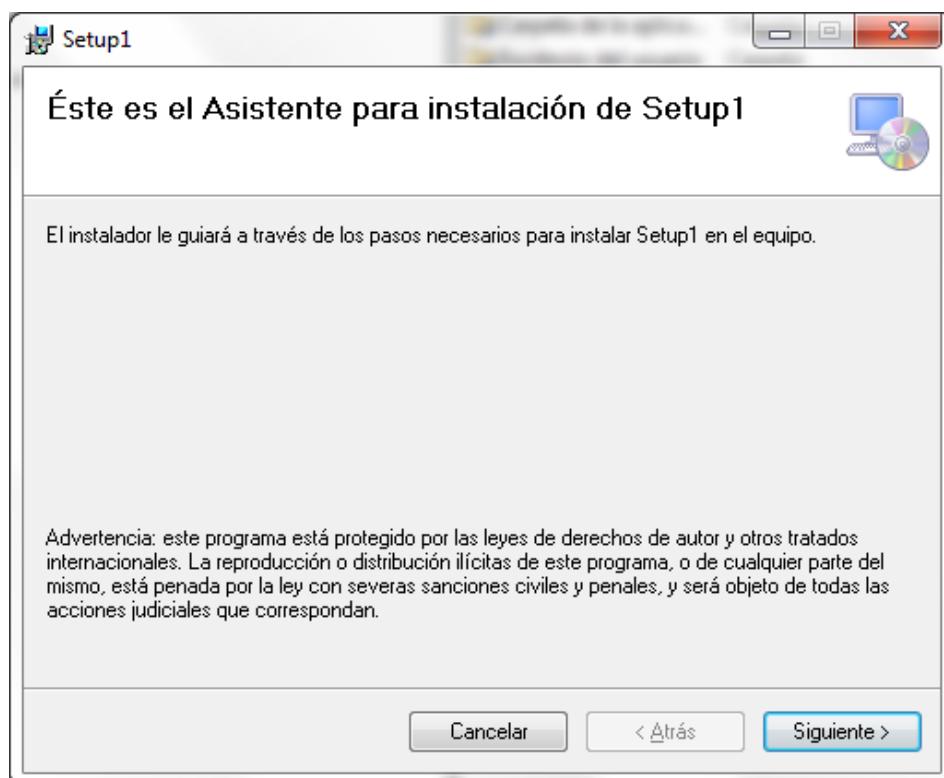
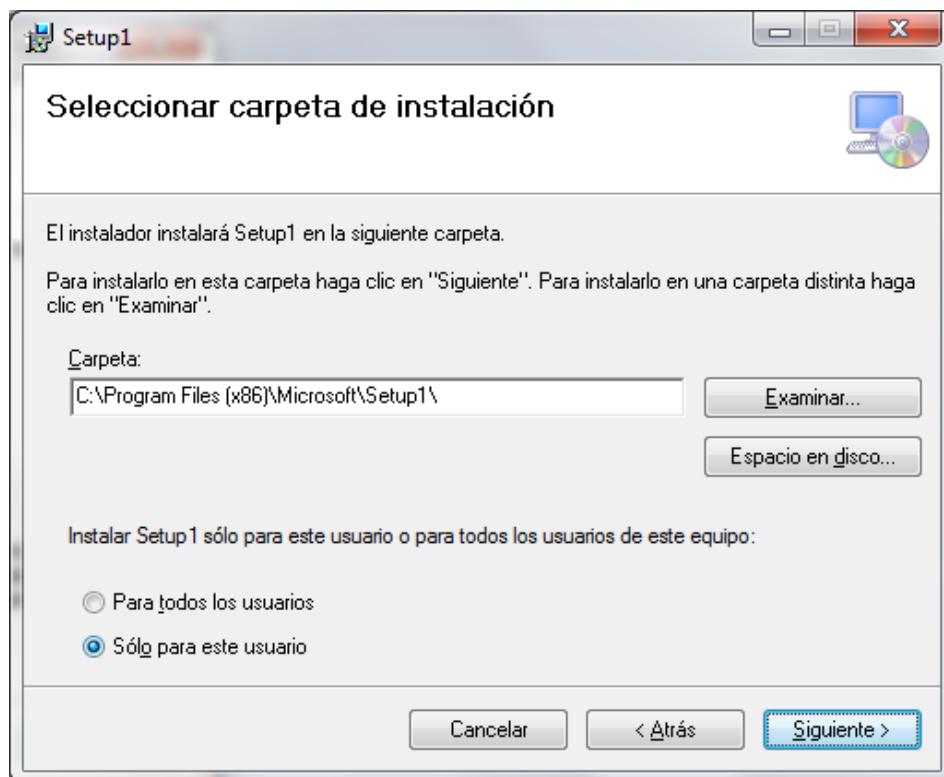
FIGURA 1.24. Afegit nou component.

FIGURA 1.25. Generació.**FIGURA 1.26.** Instal·lació.

A partir de la figura 1.27 es pot veure com s'executarà l'instal·latable des del Visual Studio.

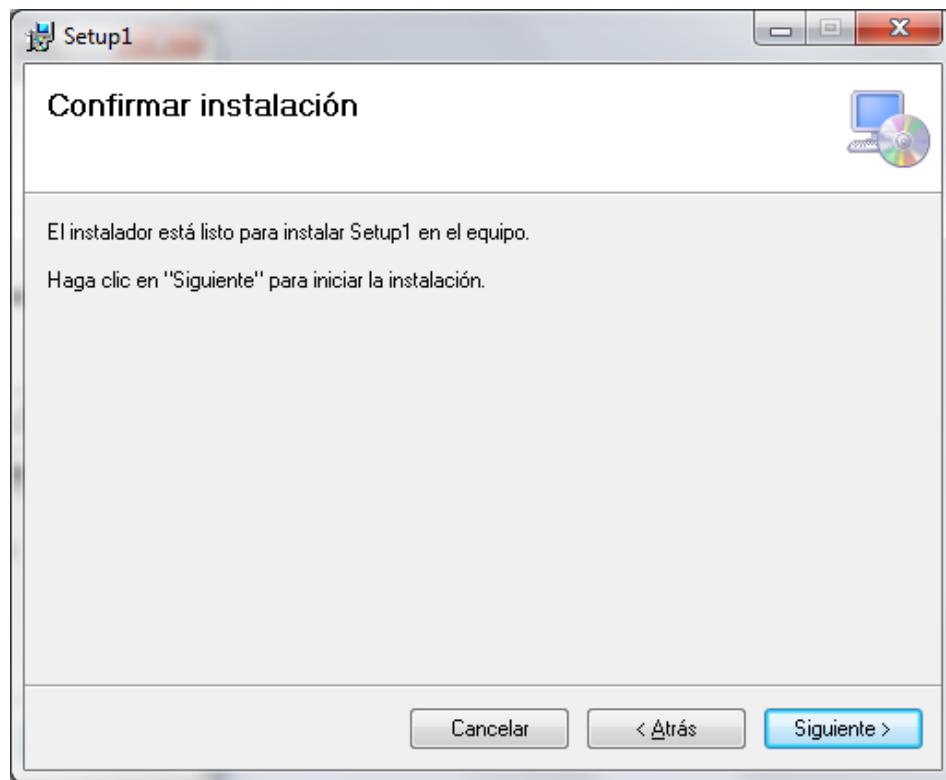
FIGURA 1.27. Assistent per la instal·lació.

Aquesta instal·lació comença a la figura 1.27 amb un assistent que permetrà seleccionar la carpeta on es voldrà tenir físicament l'aplicació desenvolupada (figura 1.28).

FIGURA 1.28. Selecció de la carpeta.

A la figura 1.29 es veu com demana la confirmació per part de l'usuari.

FIGURA 1.29. Confirmació de la instal·lació.



A les següents figures es pot observar l'evolució de l'arxiu instal·ladur i les diferents informacions que ofereix a l'usuari, fins a donar per finalitzada la instal·lació a la figura 1.32.

FIGURA 1.30. Instal·lació.

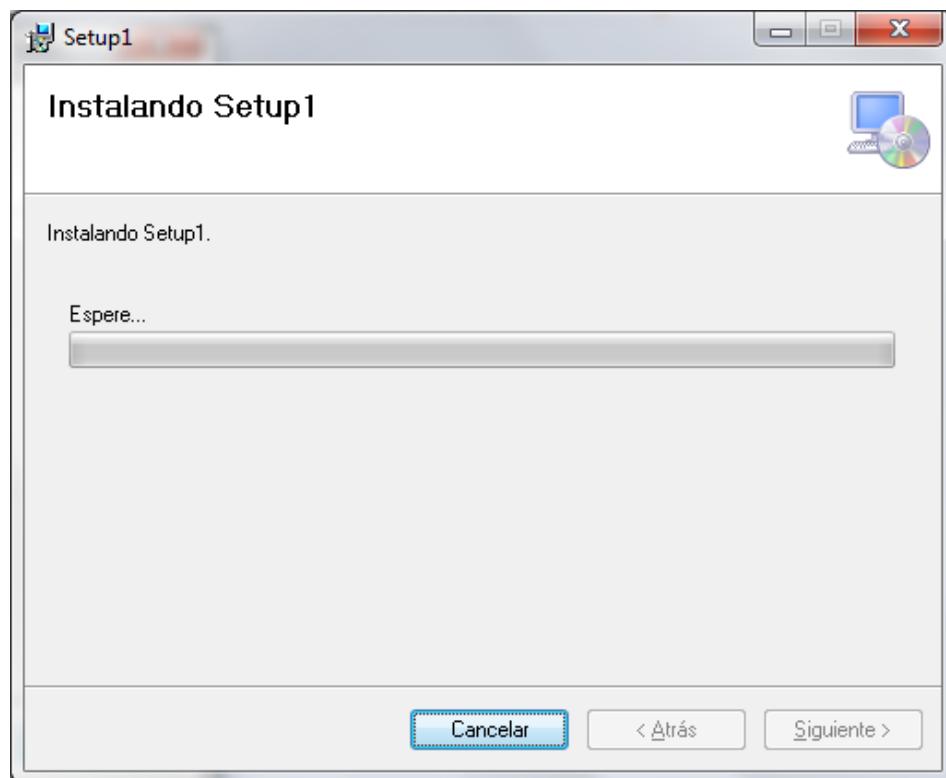
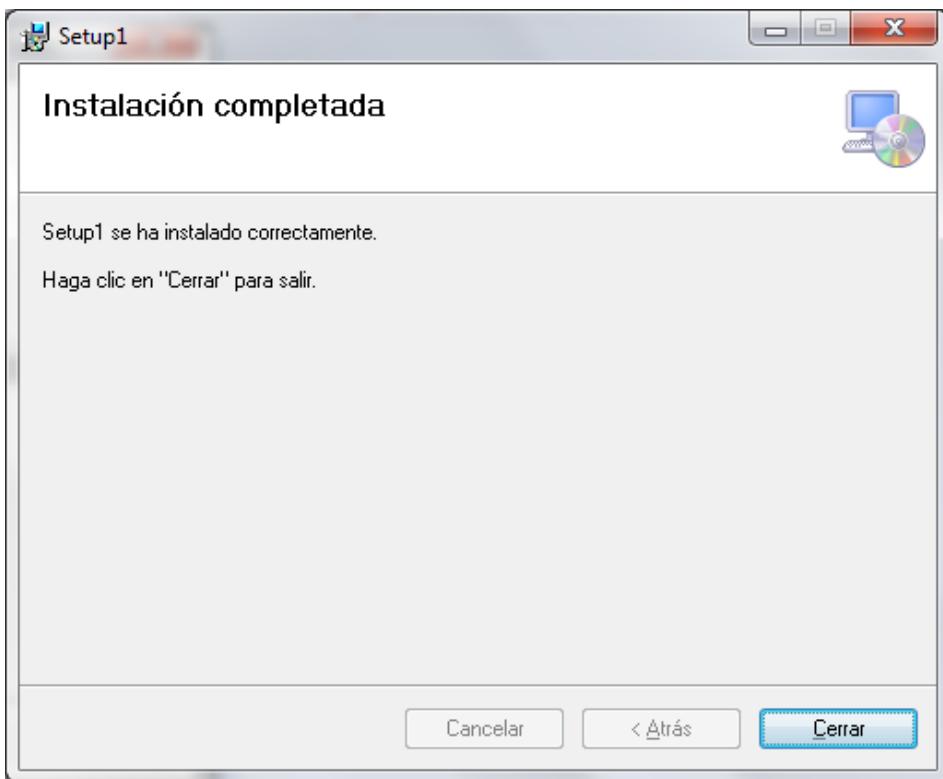
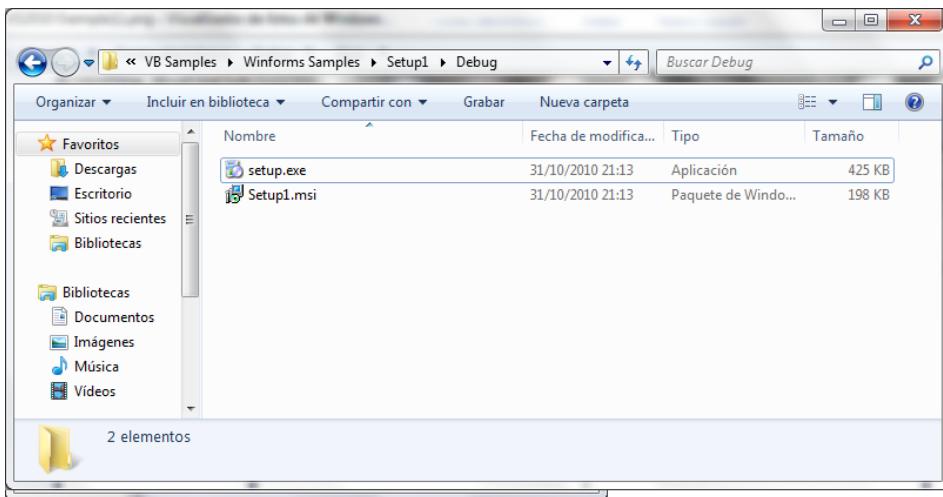


FIGURA 1.31. Fi de la instal·lació.**FIGURA 1.32.** Arxiu MSI creat.

A la figura 1.32 es pot veure l'arxiu MSI creat com a instal·lador de l'aplicació agafada com a exemple.

2. Realització de proves

Totes les fases estableties en el desenvolupament del programari són importants. La manca o mala execució d'alguna pot provocar que la resta del projecte arrossegui un o diversos errors que seran determinants per a l'èxit del projecte.

Una aplicació informàtica no pot arribar a les mans d'un usuari final errades, i menys si aquesta és prou visible i clara per haver estat detectada pels desenvolupadors. Es donaria una situació de manca de professionalitat i de confiança per part dels usuaris en els desenvolupadors. Aquesta pèrdua de confiança desenvoluparia en una manca de futures oportunitats.

És per això que aquesta fase del desenvolupament d'un projecte de programari es considera bàsica abans de fer la transferència del projecte a l'usuari final. Qui donaria un cotxe per construït i finalitzat si en intentar arrencar-lo no funciona?

2.1 Validació de la correctesa en el desenvolupament d'una aplicació

Totes les fases de desenvolupament d'un projecte de programari ofereixen l'oportunitat de cometre errades per part de tots els membres d'un equip de treball. Algunes d'aquestes errades seran més determinants que altres i tindran més o menys implicacions en el futur del desenvolupament del projecte.

Per exemple, un cap de projecte, a l'hora de planificar tota la feina que caldrà fer, estipula el disseny de les interfícies en 4 hores de feina per a un únic dissenyador. Si una vegada executada aquesta tasca del projecte la durada ha estat de 8 hores i s'han necessitat dos dissenyadors, la repercussió en el desenvolupament del projecte serà una desviació en temps i en cost, que potser es podrà compensar utilitzant menys recursos o menys temps en alguna tasca posterior.

En canvi, si l'errada ha estat del creador de la base de dades, que ha obviat un camp clau d'una taula principal i la seva vinculació amb una segona taula, aquesta pot ser molt més determinant en les tasques posteriors. Si es creen les interfícies a partir d'aquesta base de dades errònia i es comença a desenvolupar el programari sense identificar l'errada, podrà succeir que al cap d'unes quantes tasques es detecti l'errada.

En funció de la importància de l'errada aquesta es podrà solucionar modificant algunes línies de codi o, potser, s'haurà treballat per al diable i caldrà tornar a fer tota la feina feta.

Cal remarcar dos conceptes abans de continuar: verificació i validació. Aquests dos conceptes estaran directament vinculats amb les proves que verificaran for-

Fases habituals

Cal recordar les fases habituals en el desenvolupament d'un projecte de programari: presa de requisits, anàlisi, disseny, desenvolupament, proves, finalització i transferència.

Perfils habituals

Els perfils habituals d'un equip de treball en un projecte de desenvolupament de programari són: cap de projecte, analista, dissenyador, programadors i testejadors.

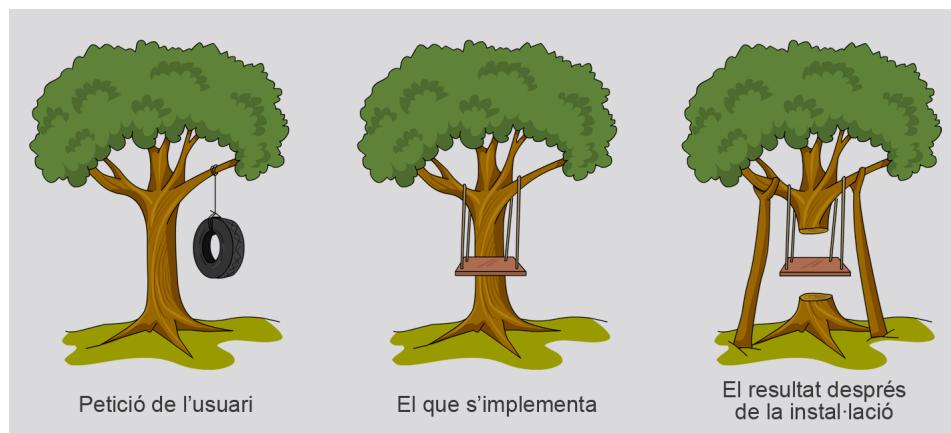
Un error no detectat a l'inici del desenvolupament d'un projecte pot arribar a necessitar cinquanta vegades més esforços per ser solucionat que si és detectat a temps.

malment la correctesa d'una aplicació informàtica.

Aquests dos conceptes són el nom que es dóna de manera genèrica als processos que serveixen per revisar que s'assegura que el programari desenvolupat satisfarà les especificacions recollides i que cobrirà les necessitats expressades pel client. Cal diferenciar-los de manera adequada, encara que s'acostumen a confondre.

La validació és l'activitat encarregada d'assegurar-se que l'aplicació informàtica obtinguda és la que s'havia determinat inicialment i és la que es volia construir. La validació confirmarà que l'aplicació funciona tal com el client ha demanat. El sistema fa el que el client vol?

FIGURA 2.1. Verificació i validació



A la figura 2.1 es pot veure un exemple visual sobre les diferències que ens podem trobar entre el que demana un usuari, el que es programa i el que es lliura en la fase de finalització i transferència, que serà el que rebrà i utilitzarà l'usuari.

La verificació és l'activitat que comprova el funcionament correcte del codi programat. Es comprovarà que tecnològicament l'aplicació informàtica s'hagi desenvolupat de manera correcta. El sistema fa el que ha de fer?

Al llarg del projecte de desenvolupament del programari el sistema cal que sigui verificat i validat a cada pas del procés, partint de la verificació i validació del pas anterior i establint la documentació adient per poder tornar a validar i verificar en el pas següent.

Aquests dos passos són els que determinen la validació formal de la correctesa del projecte, que començarà amb la revisió dels requisits en la fase d'anàlisi i continuuarà amb la revisió del disseny i la codificació fins a la prova de l'aplicació final per lliurar.

2.2 Objectius, importància i limitacions del procés de prova. Estratègies

El procés de prova es considera una fase independent més del projecte de desenvolupament del programari. Es durà a terme en finalitzar la codificació i abans de la transferència a l'usuari final.

El procés de prova és l'acció d'executar reiteradament una aplicació informàtica utilitzant totes les possibles variants en les seves dades, amb l'objectiu de preveure tots els casos possibles en la recerca d'errors.

És per això que entre les afirmacions relacionades amb el terme *prova* es poden trobar:

- L'equip de treball i els usuaris finals no han de participar en el procés de prova, sinó que han de seleccionar un equip extern al projecte per al disseny i desenvolupament.
- Una prova té èxit si descobreix un error no detectat fins llavors. Un bon pla de prova serà aquell que tingui altes probabilitats de trobar un error que no s'hagi descobert fins aquell moment.
- Cal provar tots els mòduls o parts d'un projecte de desenvolupament del programari (interfícies, documentació, ajuda...)
- Si en un mòdul es troba un error, hi haurà una alta probabilitat de trobar-ne més en el mateix mòdul o en altres mòduls amb una vinculació directa.

La fase de proves verificarà i validarà que el producte final sigui eficaç i efectiu. Per a això cal dur a terme el camí invers al que s'ha fet servir per al desenvolupament del programari. Si se segueix el mateix raonament que s'ha utilitzat per construir, serà més complicat trobar els errors, ja que el raonament serà semblant i els errors cometuts es reproduiran. És per això que cal seguir un camí oposat:

Cal anar del més petit (provees unitàries) al més gran (provees d'integració), en comptes de l'anàlisi *top-down*, que ha començat amb l'anàlisi dels problemes més grans per anar-los dividint en problemes més petits.

En les provees també cal anar del més concret (provees de subsistemes) al més general (provees de sistema).

La fase de proves es durà a terme just abans de lliurar un programa per a l'explotació (fase de finalització i transferència) i cal diferenciar-la de la fase de manteniment, en la qual es duen a terme algunes modificacions del programa, que caldrà provar també.

Una de les sorpreses que se solen trobar els caps de projecte i, també, els programadors, és l'enorme quantitat de temps i esforç que requereix aquesta fase.

El mòdul

Un mòdul és una part independent d'un programa que es dissenya, codifica i prova per separat. Un mòdul és una feina, activitat o funció ben definida, precisa, clara i única.

El **procés de prova** també inclou les proves independents per a cadascun dels mòduls o components que formen l'aplicació informàtica.

Si es parla de programes que involucren vides humanes (medicina, equips nuclears, controls aeris, etc) el cost de la fase de proves pot fàcilment superar el 80% del cost total del projecte.

La fase de proves és una de les parts del projecte que molts programadors encara consideren classificable com a art, és a dir, difícilment mesurable, i no solen valorar prou aquesta fase, malgrat l'enorme impacte que pot arribar a tenir en el cost de desenvolupament.

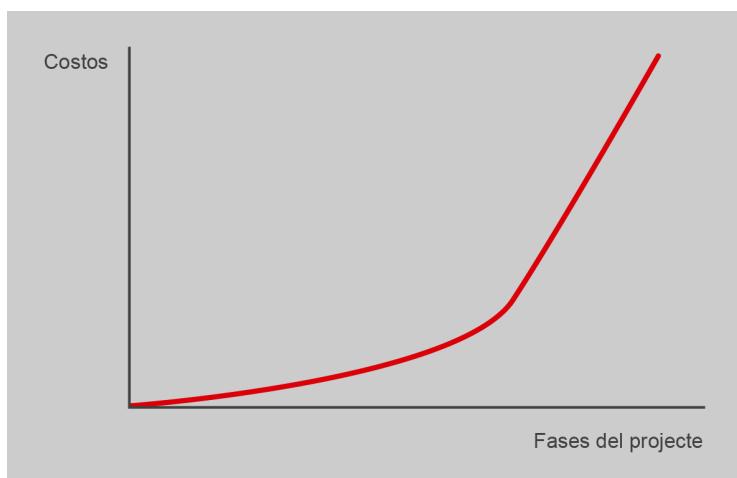
El nombre de proves no es pot valorar en funció del nombre de línies de codi, de variables o de valors que es facin servir en el desenvolupament del programari. Les proves per executar no han de ser necessàriament finites. En fer servir bucles en els quals és fàcil introduir condicions per a un funcionament sense un final, el nombre d'alternatives i combinacions possibles per desenvolupar les proves passa a ser exponencial, i es fa molt difícil (per no dir impossible) la identificació i execució completa a tots els efectes pràctics.

Cal identificar formes que siguin econòmicament acceptables i humanament assolibles per a la recerca dels errors en els projectes de desenvolupament de programari, assumint que les proves exhaustives no són possibles. Això porta els responsables a la creació dels **jocs de proves**, en el qual es provarà una part del programari (assumint que hi haurà parts que quedaràn sense explorar).

Les proves que integrin el joc de proves d'una aplicació informàtica hauran de ser representatives dels casos més importants o extrems que es puguin donar durant l'execució del programari.

Però no solament cal dur a terme les proves en la fase de proves. A cada fase del projecte de desenvolupament cal dur a terme les proves, verificacions i validacions pertinents. De la mateixa manera, a mesura que es va evolucionant en la codificació del programari, caldrà anar provant cada mòdul.

FIGURA 2.2. Temps enfront de cost



Com es pot veure a la figura 2.2, com més temps es tarda a detectar i solucionar un error en el codi, més costos serà solucionar-ho, i la gràfica pot arribar a ser exponencial. D'aquesta manera es pot observar com pot passar d'una proporció de 100 a 1.000 el preu de trobar i solucionar problemes del programari una vegada finalitzat el desenvolupament del codi font. A més a més, la major part dels errors es concentra en les primeres fases del projecte de creació d'aplicacions.

A manera de resum, els experts proposen una sèrie de recomanacions sobre com s'han de fer les proves:

- No s'ha de veure el procés de prova com a rutinari, sinó com un procés fonamental; per això, s'hi han de destinar recursos, temps, personal experimentat i un procés creatiu.
- Els casos de prova han d'incloure tant entrades correctes com incorrectes per avaluar el comportament del sistema en qualsevol situació.
- Les proves s'han de dissenyar de manera que tinguin la màxima probabilitat de trobar el nombre més gran d'errors amb la mínima quantitat d'esforç i temps.
- El programador no ha de provar els seu propis programes. A les grans empreses hi ha un equip de prova diferent del de desenvolupament.
- Les proves s'han de centrar i insistir més en les parts o mòduls que més s'utilitzen o siguin més crítics per al sistema.
- No s'ha d'associar l'error a la negligència d'un programador; la finalitat de les proves ha de ser trobar errades i no desprestigiar ningú.

Les proves tenen també les seves limitacions. Les proves no podran descobrir tots els possibles errors del codi desenvolupat. A mesura que les proves vagin involucrant més mòduls i parts del sistema, hi ha més possibilitats de trobar errors, però sempre amb una limitació.

Cal tenir també en compte els components de l'equip de treball que s'encarregaran de les proves. Es poden anomenar *enginyers de proves*.

Algunes de les habilitats professionals i humanes que han de tenir són:

- Disposar d'habilitats comunicatives per interactuar amb la resta de l'equip de treball.
- Concentrar la seva atenció en els detalls.
- Assumir el repte de trobar errades, més enllà de fer una mera comprovació del funcionament.
- Comunicar-se de manera clara i precisa amb la resta de l'equip de treball per comunicar les errades trobades (és a dir, facilitar la feina posterior dels programadors que hauran de solucionar els problemes).

- Disposar d'habilitats destacades en curiositat i intuïció per poder explorar situacions en què es puguin trobar errors.
- Saber treballar sota pressió, tenint en compte que aquesta fase acostuma a tenir poc temps per les desviacions dels projectes.

2.3 Classificacions dels tipus de proves

El procés de les proves acostuma a ser un procediment repetitiu i laboriós. Per això s'intenten automatitzar alguns processos de manera que els procediments de prova es puguin fer amb l'ajuda de programes informàtics.

Hi ha diferents tipificacions de les proves que es fan als projectes de desenvolupament de programari.

En primer lloc cal tenir en compte el moment per dur a terme les proves. Les proves es poden fer en el moment de donar per finalitzat un mòdul de codi de programació, o bé quan tots els mòduls estan finalitzats i es volen integrar, o bé quan l'aplicació ja està finalitzada i integrada i es vol dur a terme una càrrega de dades reals, o bé si és el client qui fa les proves.

Per a aquests casos es tenen el tipus de proves següents:

- Proves unitàries.
- Proves d'integració.
- Proves de càrrega.
- Proves d'acceptació.

Un altre tipus de classificació es pot fer en funció de la manera de fer les proves. En aquest cas les proves poden ser sense entrar en el detall de la implementació, és a dir, només interactuant amb les interfícies, o bé tenint en compte els detalls interns de la implementació. En aquest cas les proves es troben integrades dintre de les proves unitàries i es poden classificar en les següents:

- Proves unitàries.
- Proves de caixa negra.
- Proves de caixa blanca o transparent.
- Revisions.

Una vegada l'aplicació desenvolupada ja es troba validada caldrà dur a terme altres proves, que es coneixen com a *provees de sistema*. Aquestes proves validaràn la integració del programari desenvolupat en el sistema ja existent de l'usuari final o client. Algunes proves per desenvolupar són:

- Rendiment.
- Resistència.
- Robustesa.
- Seguretat.
- Usabilitat.
- Instal·lació.

2.4 Proves unitàries

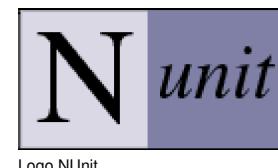
La prova unitària es planteja a petita escala, i consisteix a anar provant d'un en un (unitat per unitat) els diferents mòduls que constitueixen una aplicació.

Les **proves unitàries** es defineixen com el procediment per provar el funcionament correcte d'un mòdul de codi. Això serveix per assegurar que cadascun dels mòduls funcioni correctament per separat.

Normalment es pot distingir una fase informal abans d'entrar en la fase de proves pròpiament dita. La fase informal la du a terme el codificador en les seves oficines, i consisteix a anar executant el codi per validar que “bàsicament, funciona”. Aquesta fase sol consistir en petits exemples que s'intenten executar. Si el mòdul falla, es pot utilitzar un depurador per observar l'evolució dinàmica del sistema, localitzar l'error i reparar-lo.

D'altra banda, actualment, hi ha eines sofisticades capaces d'emetre “opinions” sobre un programa i alertar de construccions arriscades, d'expressions molt complicades, etc. De vegades poden prevenir sobre variables que es poden utilitzar abans de prendre algun valor (no inicialitzades) o bé de variables que es carreguen però després no s'usen.

Una eina de programari lliure que permet automatitzar les proves és **NUnit**. Es tracta d'un entorn de treball de programari lliure que permet la realització de proves unitàries per a Microsoft .NET i Mono. Serveix per al mateix propòsit que el que fa **JUnit** en el món Java.



Logo NUnit

JUnit és un conjunt de classes que permet fer l'execució de classes Java de manera controlada, per avaluar si cada un dels mètodes de la classe es comporta com s'espera. S'assignen diferents valors d'entrada i s'avaluen els valors de retorn esperats per indicar si els mètodes han complert de manera correcta les proves o han fallat en la seva execució.

Conceptualment es tracta d'anar creant proves unitàries a mesura que es vagin programant cada un dels mètodes/funcions o procediments de cada classe. Cal validar el funcionament d'un mòdul abans de passar al següent. Sembla una tasca

senzilla d'implementar, però de vegades és difícil de dur a terme. A la pràctica és difícil, ja que el programari és viu i s'ha de ser molt sistemàtic perquè qualsevol variació en el codi quedí reflectida en la prova unitària corresponent.

Les proves unitàries ha de complir característiques com ser automàtiques, completes, reutilitzables o independents.

2.4.1 Proves de caixa negra

Les **proves de caixa negra** duran a terme la comprovació del funcionament d'un component de programari per mitjà de la seva interfície, sense entrar a analitzar el seu funcionament intern.

Conceptualment es tractarà de comprovar que el component funciona correctament quan:

- S'introdueixen dades correctes.
- S'introdueixen dades errònies.
- S'introdueixen dades equivalents.
- S'introdueixen valors extrems superiors i inferiors.

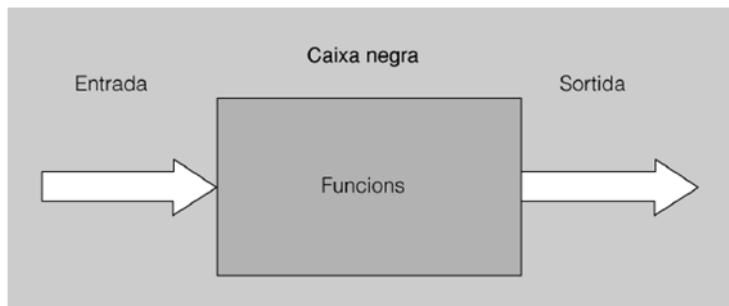
Per exemple, es té una funció que, donats dos nombres enters, retorna quin és el major. S'hauria de comprovar quina és la resposta quan introduïm:

- Dos nombres enters.
- Dos nombres enters iguals.
- Dos nombres reals.
- Un enter i un altre real.
- Un nombre positiu i un de negatiu.
- Un caràcter.

En tots els casos la funció ha de donar una resposta, encara que sigui un missatge d'error. El que no ha de passar és que la funció quedí bloquejada (vegeu la figura 2.3).

En cap moment no s'està accedint al codi programat ni s'està avaluant. Només es vol veure la resposta de la funció per a múltiples opcions donades.

Les **proves de caixa negra** proven la funcionalitat del programa per al qual es dissenyen casos de prova que comprovin les especificacions del programa.

FIGURA 2.3. Funcionament de la caixa negra

El joc de proves que es farà servir per validar amb les proves de la caixa negra haurà de ser tan petit com sigui possible, que es pugui executar en un temps raonable, i al mateix temps, que cobreixi la varietat d'entrades i sortides més àmplia possible.

Per aconseguir-ho s'han dissenyat diferents tècniques per establir aquests jocs de proves:

- **Classes d'equivalència:** es tracta de determinar els diferents tipus d'entrada i sortida, agrupar-los i escollir casos de prova per a cada tipus o conjunt de dades d'entrada i sortida.
- **Anàlisi dels valors límit:** estudien els valors inicials i finals, ja que estadísticament s'ha demostrat que tenen més tendència a detectar errors.
- Estudi d'errors típics: l'experiència ens diu que hi ha una sèrie d'errors que s'acostumen a repetir en molts programes; es tractaria de dissenyar casos de prova que provoquessin les situacions típiques d'aquest tipus d'errors.
- **Maneig d'interfície gràfica:** per provar el funcionament de les interfícies gràfiques s'han de dissenyar casos de prova que permetin descobrir errors en el maneig de finestres, botons, icones, etc.
- **Dades aleatòries:** es tracta d'utilitzar una eina que automatitzi les proves i que generi d'una manera aleatòria els casos de prova. Aquesta tècnica no optimitza l'elecció dels casos de prova, però si es fa durant prou temps amb moltes dades, podrà arribar a fer una prova bastant completa. Aquesta tècnica es podria usar com a complementària a les anteriors o en casos en què no sigui possible aplicar-ne cap altra.

Un avantatge de **les proves de caixa negra** és que són independents del llenguatge o paradigma de programació utilitzat, de manera que són vàlides tant per a programació estructurada com per a programació orientada a objectes.

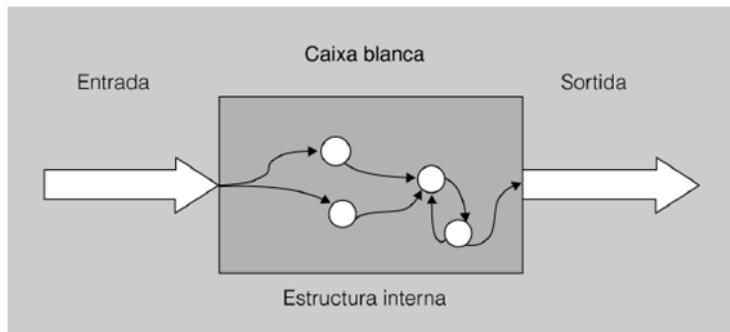
2.4.2 Proves de caixa blanca

Les **proves de caixa blanca** són les encarregades d'analitzar la manera en què un component de programari resol un determinat problema atenent als detalls interns d'implementació.

Conceptualment es tractarà d'examinar l'interior d'un mòdul de programari desenvolupat, com es pot observar a la figura 2.4. Es verificarà la correcció del següent:

- Sentències, condicions, bucles...
- Estructuració del codi.
- Optimització del codi.

FIGURA 2.4. Representació del funcionament de les proves de la caixa blanca



Les **proves de caixa blanca** es fixen en com s'ha implementat una determinada codificació d'un mòdul o una aplicació per seleccionar els jocs de proves que es duran a terme.

Els jocs de proves haurien de preveure tots els cassos possibles, tots els possibles camins del flux de control del programari implementat. Es fa un examen minuciós dels detalls procedimentals per comprovar els diferents camins lògics del programari, mitjançant els casos de prova que els recorren.

Un mètode per poder establir de manera adequada els jocs de proves és el mètode de cobertura de flux de control. Aquest mètode consisteix a utilitzar l'estructura de control del programa per obtenir els casos de prova, que són dissenyats de manera que garanteixin que almenys es passa una vegada per cada camí del programa.

Fent el camí invers del que s'acostuma a fer a l'hora de programar es pot cercar l'obtenció d'un diagrama de flux de control que representi el codi i provar tots els camins simples, totes les condicions i tots els bucles del programa.

Pot ser impossible cobrir el 100% si el programa és molt complex, però es pot aconseguir un mínim de garanties d'eficàcia si se segueixen els suggeriments per dissenyar els casos de prova per a cada element del codi programat, com són:

Es pot trobar més informació referent a aquest apartat en la secció Annexos del web del mòdul.

- Camí simple
- Condicions
- Bucles simples
- Bucles niats

2.4.3 Revisions

Un altre tipus de prova unitària són les revisions. Aquest tipus de prova no consisteix a posar a prova al codi generat en temps d'execució, sinó que es basa en la inspecció directa del codi de manera completa o per mòduls.

Fer una revisió del codi de manera sistematitzada i seguint unes normes és el que ho converteix en un tipus de prova unitària anomenada **revisions**.

Les revisions les acostuma a fer, inicialment, el programador mateix, però també es fan servir altres tècniques de grup que s'utilitzen per a la comprovació del codi, denominades *revisions tècniques*.

Entre aquestes revisions tècniques es poden trobar **reunions**, **walkthroughs** o **inspeccions**.

Les **reunions** són situacions flexibles que acostumen a requerir poca preparació prèvia, en les quals es troben els diversos desenvolupadors per revisar el codi i mirar la qualitat i provar de detectar errors abans de la fase de proves.

Els **walkthroughs** són un procediment més formal que s'aplica a la revisió de codi. El desenvolupador es reuneix amb revisors experts, que indicaran qualsevol error o dubte que trobin en el codi programat. S'establirà una discussió per explicar els dubtes i trobar possibles solucions.

Les **inspeccions** són el procediment més formal dels tres. Requereixen una preparació prèvia i la preparació per part d'un moderador, que repartirà papers i feines entre els participants. Aquests seran els desenvolupadors, els revisors o enginyers de proves i el moderador que, seguint una llista, inspeccionaran el programari abans de la reunió.

Les revisions seran importants no solament com a prova unitària, sinó que també caldrà utilitzar-les en tasques més vinculades amb el control de la qualitat i altres validacions i verificacions del sistema.

Alguns avantatges que les proves unitàries poden aportar al desenvolupament de codi són:

- Documentació del codi, en deixar constància de les proves fetes i els seus resultats.

- Simplificació de la integració entre mòduls en garantir la manca d'errors a cada mòdul de manera independent.
- Separació de les interfícies gràfiques d'usuari i la implementació del codi.

2.5 Proves d'integració

Les proves unitàries no poden identificar tots els errors que tindrà el codi per si soles. Hi ha errors que poden aparèixer en integrar dos o més mòduls o en carregar les dades o lliurar el sistema creat a l'usuari final. És per aquesta raó que cal tenir en compte altres tipus de proves que prevegin aquest casos.

Les proves d'integració tenen com a base les proves unitàries i consisteixen en una progressió ordenada de tests per als quals els diferents mòduls van essent assemblats i provats fins haver integrat el sistema complet. Consisteixen a verificar que un gran nombre de parts del programari (mòduls de l'aplicació desenvolupada) continuen funcionant quan aquests s'han ajuntat.

A vegades es produeix la situació que uns components que funcionaven perfectament per separat, en integrar-se, fallen. Normalment es deu a un error en la comunicació entre uns i altres. És convenient que aquesta tasca la facin persones alienes a la programació dels mòduls, tant dels mòduls independents com de les persones que els han integrat. Si un programador que ha desenvolupat un mòdul detecta una errada en integrar-lo amb un altre i la soluciona sense deixar-ne constància, pot estar ocultant un error que afecti altres parts del codi o altres mòduls. Com més tard es detecta un error, més costosa serà la resolució.

Si en algun moment de la prova es detectessin un o més errors, cal deixar constància del fet i reenviar els mòduls afectats al responsable de la programació perquè identifiqui la causa del problema i el solucioni. Després es tornaran a efectuar les proves i així successivament fins que el sistema sencer estigui integrat i sense errors.

Aquest tipus de proves se sol fer de manera gradual. Es proven dos components junts. Si funcionen, s'incorpora un tercer, i així de manera successiva.

Si seguim amb l'exemple de la funció que ha de retornar el nombre més gran donats dos nombres enters, es pot donar el cas que aquesta funció s'hagi d'integrar en una altra que, donades dues dates, retorna la data més gran.

Aquesta segona funció cridarà a la primera per comprovar l'any, després ho tornarà a fer per comprovar el mes i, al final, ho farà per comprovar el dia. S'ha d'esbrinar si totes dues responen correctament juntes, quan una crida l'altra i en rep els resultats.

Les interfícies en programació són el mètode de comunicar-se les distintes unitats d'una aplicació per enviar-se dades, missatges, ordres, paràmetres, etc.

Un objectiu important de les **proves d'integració** és localitzar errors en les interfícies entre les distintes unitats.

Els elements no s'integren tots al mateix temps, sinó que s'utilitzen diferents estratègies d'integració incremental, que, bàsicament, consisteixen en el següent:

- integrar uns quants components,
- provar-los, i
 - si no hi ha errors, afegir-hi components nous;
 - si hi ha errors, se solucionen i es tornen a provar.

En el desenvolupament del programari acostuma a haver-hi una organització jeràrquica dels mòduls, organitzats per nivells.

En l'exemple de la funció que retorna el nombre més gran i la data més gran, el nivell més baix compararà enters i el nivell superior compararà dates. La funció que compara les dates serà més propera a l'usuari, es considerarà un mòdul superior o un pare. En canvi, la funció que compara enters, que serà utilitzada pel pare, es considera un mòdul inferir o fill i es trobarà allunyada de l'usuari que, probablement, en desconeixerà l'existència.

Aquest tipus d'organització permet unes facilitats en l'execució i localització d'errors, en poder tenir una estructura dels mòduls ordenada que indica en tot moment quins han estat els darrers a acoblar-se i provar-se i, en funció del moment de la detecció de l'errada, saber on es troba localitzada o el seu motiu.

A partir d'aquestes informacions es poden desenvolupar diferents tipus d'estratègies de proves d'integració:

- Proves d'integració ascendents.
- Proves d'integració descendents.
- Proves d'integració combinades.
- Proves d'integració de *big bang*.

2.5.1 Prova d'integració ascendent

Les proves d'integració ascendents comencen combinant en grups les unitats o mòduls de baix nivell per provar-los (provees dobles, triples, quadruples, etc., segons el nombre de mòduls combinats).

Per fer les proves és necessari dissenyar mòduls programari com ara components de proves, denominats *programes controladors de proves* o *impulsors*, que permeten simular el comportament dels mòduls superiors.

Els programes controladors tenen la mateixa interfície del mòdul que substitueixen, però no fan la seva funció; per tant, són fàcils de construir, ja que no simulen l'activitat dels mòduls pare, sinó que serveixen simplement per poder executar el mòdul que es provarà.

Amb els programes controladors duem a terme les proves de les interfícies i els mòduls. Quan s'ha provat un grup de mòduls es continua el procés amb la resta d'agrupacions del mateix nivell.

Un cop provat un nivell, se substitueixen els programes controladors pels mòduls superiors i es repeteix el procés; per aquesta raó, serà necessari anar desenvolupant nous programes controladors superiors. El procés s'anirà repetint successivament fins que s'integren tots els nivells.

2.5.2 Prova d'integració descendent

En primer lloc, es proven els mòduls de nivell superior i després es van integrant els components de la capa següent. Una vegada provats els components d'aquesta capa es proven els de nivell inferior i així successivament fins a involucrar totes les capes.

Per poder fer les proves descendents sense incloure els mòduls inferiors, necessitem també components; concretament, és necessari crear mòduls programari simuladors de proves anomenats *ficticis* o *stub*, que n'emulin el comportament i tinguin la mateixa interfície.

Els mòduls ficticis simulen les funcions que haurien de fer els mòduls reals que substitueixen; per això, són més difícils de construir que els programes controladors. En aquest cas no es necessiten programes controladors de proves, com succeïa en els ascendents.

2.5.3 Prova d'integració combinada

Combinen la integració ascendent i descendente. Els mòduls individuals es proven amb controladors i ficticis, i després es van provant les capes superiors i inferiors substituint els controladors i ficticis pels mòduls provats.

En els nivells o capes superiors es fa servir una estratègia descendente, i en les inferiors, ascendent, fins a trobar-se en alguna capa intermèdia. Les proves es poden dur a terme en paral·lel per estalviar temps.

2.5.4 Prova d'integració de big bang o gran explosió

De primer, es proven tots els mètodes individualment i, després, tots els components del sistema s'integren a la vegada i s'hi fan proves.

Aquesta estratègia sembla simple d'aplicar, però té l'inconvenient important que quan es descobreix una fallada és molt difícil localitzar-la per poder-la corregir.

En totes les estratègies s'haurà de decidir l'ordre en què es van integrant els mòduls. Un bon criteri és començar amb els mòduls crítics del sistema i els que siguin més susceptibles de contenir errors pel fet de ser més complexos o estar relacionats amb més requisits.

2.6 Proves de càrrega i acceptació

El pas següent una vegada fets les proves unitàries i les proves d'integració serà dur a terme primer les proves de càrrega i, posteriorment, les proves d'acceptació.

Les proves de càrrega són proves que tenen com a objectiu comprovar el rendiment i la integritat de l'aplicació ja acabada amb dades reals. Es tracta de simular l'entorn d'explotació de l'aplicació.

Amb les anteriors proves (unitàries i d'integració) quedaria provada l'aplicació a escala de "laboratori". Però també es necessita comprovar la resposta de l'aplicació en situacions reals, i fins i tot, en situacions de sobrecàrrega, tant a escala de rendiment com de descontrol de dades.

Per exemple, una aplicació lenta pot ser poc operativa i no útil per a l'usuari.

Després de les proves de càrrega es troben les proves d'acceptació. Aquestes proves les fa el client o usuari final de l'aplicació desenvolupada. Són bàsicament proves funcionals, sobre el sistema complet, i busquen una cobertura de l'especificació de requisits i del manual de l'usuari. Aquestes proves no es fan durant el desenvolupament, ja que seria impresentable amb vista al client, sinó una vegada passades totes les proves anteriors per part del desenvolupador o l'equip de tests.

L'objectiu de la **prova d'acceptació** és obtenir l'aprovació del client sobre la qualitat de funcionament del sistema desenvolupat i provat.

L'experiència demostra que, encara després del més acurat procés de proves per part del desenvolupador i l'equip de treball, queden una sèrie d'errors que només apareixen quan el client posa en funcionament l'aplicació o el sistema desenvolupat.

Els programadors acostumen a obtenir sorpreses de les proves d'acceptació: "Però, a qui se li a ocorregut usar així el programa?".

Sigui com sigui, el client sempre té la raó. Per aquest motiu, molts desenvolupadors exerciten unes tècniques denominades **proves alfa i proves beta**.

Les **proves alfa** consisteixen a convidar el client que vingui a l'entorn de desenvolupament a provar el sistema. Es treballa en un entorn controlat i el client sempre té un expert a mà per ajudar-lo a usar el sistema i per analitzar els resultats.

Les **proves beta** vénen després de les proves alfa, i es desenvolupen en l'entorn del client, un entorn que és fora de control per al desenvolupador i l'equip de treball. Aquí el client es queda tot sol amb el producte i tracta de trobar els errors, dels quals informarà el desenvolupador.

Les proves alfa i beta són habituals en productes que es vendran a molts clients o que faran servir molts usuaris. Alguns dels compradors potencials es presten a aquestes proves bé per anar entrenant el seu personal amb temps, bé en compensació d'algun avantatge econòmic (millor preu sobre el producte acabat, dret a manteniment gratuït, a noves versions, etc.). L'experiència mostra que aquestes pràctiques són molt eficaces. En un entorn de desenvolupament de programari tenen sentit quan l'aplicació o sistema per desenvolupar el farà servir un gran nombre d'usuaris finals (empreses grans amb diferents departaments que hauran d'utilitzar aquesta nova eina).

2.7 Proves de sistema i de seguretat

Les proves de sistemes són aquelles proves que es duran a terme una vegada finalitzades les proves unitàries (cada mòdul per separat), les proves d'integració dels mòduls, les proves de càrrega i les proves d'acceptació per part de l'usuari.

Temporalment es troben en una situació en la qual l'usuari ha pogut testejar l'aplicació desenvolupada duent a terme les proves d'acceptació. Posteriorment l'aplicació s'ha integrat al seu nou entorn de treball. Les proves de sistema serviran per validar l'aplicació ja validada una vegada ha estat integrada amb la resta del sistema de l'usuari.

Alguns tipus de proves per desenvolupar durant les proves del sistema són:

- 1) Proves de rendiment: valoraran els temps de resposta de la nova aplicació, l'espai que ocuparà en disc, el flux de dades que generarà a través d'un canal de comunicació...
- 2) Proves de resistència: valorarà la resistència de l'aplicació per a determinades situacions del sistema.
- 3) Proves de robustesa: valorarà la capacitat de l'aplicació per suportar diverses entrades no correctes.

4) Proves de seguretat: aquestes proves ajudaran a determinar els nivells de permisos dels usuaris, les operacions que podran dur a terme i les d'accés al sistema i a les dades.

5) Proves d'usabilitat: aquestes proves determinaran la qualitat de l'experiència d'un usuari en la manera en la qual aquest interactua amb el sistema.

6) Proves d'instal·lació: aquestes proves indicaran les operacions d'arrencada i actualització dels programaris.

Les proves de sistema aglutinen tantes altres proves que tindran diversos objectius:

- Observar si l'aplicació fa les funcions que ha de fer i si el nou sistema es comporta com ho hauria de fer.
- Observar els temps de resposta per a les diferents proves de rendiment, volum i sobrecàrrega.
- Observar la disponibilitat de les dades en el moment de recuperació d'una errada (a la vegada que la correctesa).
- Observar la usabilitat.
- Observar la instal·lació (assistents, operadors d'arrencada de l'aplicació, actualitzacions del programari...).
- Observar l'entorn una vegada l'aplicació està funcionant a ple rendiment (comunicacions, interaccions amb altres sistemes...).
- Observar el funcionament de tot el sistema a partir de les proves globals fites.
- Observar la seguretat (el control d'accés i intrusions...).

Les proves a escala global del sistema s'han anat produint a mesura que es tenien funcionalitats perfectament acabades. És el cas, per exemple, de provar el funcionament correcte del desenvolupament d'una partida en un dels tres jocs, o la navegació correcta per les diferents pantalles dels menús.

Proves de validació: aquestes proves permeten comprovar si efectivament es compleixen els requisits proposats pel nostre sistema.

2.8 Proves de regressió i proves de fum

Les proves de regressió són un tipus més de proves de programari.

Aquestes proves de regressió cerquen detectar possibles nous errors o problemes que puguin sortir en haver introduït canvis o millors en el programari.

Aquests canvis poden haver estat introduïts per solucionar algun problema detectat en la revisió del programari arran d'una prova unitària o d'integració o de sistema. Aquests canvis poden solucionar un problema però provocar-ne d'altres, sense haver-ho previst, en altres llocs dels programari. És per aquesta raó que cal dur a terme les proves de regressió en finalitzar la resta de proves.

Un control no convenient dels canvis de versions, o una falta de consideració envers altres mòduls o parts del programari, poden ser raons dels problemes per detectar en les proves de regressió.

Es pot automatitzar la detecció d'aquest tipus d'errors amb l'ajuda d'eines específiques. L'automatització és complementària a la resta de proves, però en facilitarà la repetibilitat. Els problemes que es poden trobar de l'automatització de les proves de regressió és que demanaran un manteniment complex.

"Proves de fum"

El terme *proves de fum* sorgeix a partir de la fabricació de maquinari. Si després de reparar un component de maquinari, aquest "no treu fum", és que funcionarà correctament.

Les proves de fum es fan servir per descriure la validació dels canvis de codi en el programari abans que els canvis en el codi es registrin en la documentació del projecte.

S'acostuma a dur a terme una revisió del codi abans d'executar les proves de fum. Aquesta revisió del codi es farà sobretot pel que fa als canvis que s'hagin produït en el codi.

2.9 Validació formal de la correctesa

Per poder validar la correctesa d'un programa i dur a terme unes proves fidedignes cal establir-ne amb precisió la semàntica. El fet que un programa tingui un error pot resultar summament costós, no sols en termes econòmics, sinó que en certs casos fins i tot es pot posar en perill la vida de molts éssers humans (per exemple, programes utilitzats en medicina o en controls aeris).

Generalment les especificacions solen estar donades en llenguatge natural. Per exemple: "vull un gronxador en l'arbre del jardí". Naturalment, atesa l'ambigüitat i falta de precisió del llenguatge natural, es fa necessari tenir mètodes formals que permetin validar les especificacions, com per exemple la tècnica de predicats (precondició i postcondició).

Els fonaments de la tècnica de **precondició i postcondició** són establir les condicions que han de complir inicialment (abans de l'execució) perquè el programa funcioni correctament (en el cas de les precondicions), i les postcondicions descriuen les condicions que es compleixen una vegada ha finalitzat el programa sempre que es compleixin les condicions de la precondició.

Per tal de descriure la tècnica de precondicions i postcondicions, platejarem un exemple: Sigui A un càlcul, i P i Q, assercions.

1 {P} A {Q}

La semàntica d'aquesta fórmula és la següent: qualsevol execució d'A que comença en un estat en el qual es compleix P donarà com a resultat un estat en el qual es compleix Q. Per exemple:

```

1 { x = a ^ y = b }
2   aux := x;
3   x := y;
4   y := aux;
5 { x = b ^ y = a}

6
7 { x = a ^ y = b }
8   x := x + y;
9   y := x - y;
10  x := x - y;
11 { x = b ^ y = a}

```

D'aquesta manera, observem que hi ha diferents codificacions per resoldre una mateixa situació. L'asserció P s'anomena *precondició* i Q s'anomena *postcondició*.

A continuació es planteja un altre exemple que permetrà fer la validació formal de la correctesa d'una classe. Imaginem que es vol reaprofitar el codi següent, que incorpora una divisió.

```

1 Classe A
2 ....
3 Si valor > 0 aleshores
4 Resultat := get_total / valor
5 Si no
6 Mostrar per pantalla missatge d'error
7   Fisi
8 ...
9 Fi classe

```

Aparentment sembla que és correcte. La rutina verifica que el paràmetre sigui més gran que zero per poder fer la divisió sense problemes. Ara bé, si la classe pertany a una biblioteca de classes, ens podríem preguntar: és correcte que interactuï amb els usuaris mitjançant missatges? Com es tractaran els missatges si l'aplicació corre en mode ocult? Una solució és que la rutina retorni un codi d'error, per exemple -1; ara bé, on queda documentat aquest comportament?

Una possible manera de tractar aquest tipus de situacions és fent ús de les precondicions i postcondicions, no permetent que la rutina es trobi en la situació en què es passa un paràmetre menor o igual que zero.

```
1 Precondició { valor >0 }
2 Postcondició { Resultat:= get_total / valor}
```

Després d'haver descrit en uns petits exemples la tècnica de precondició i postcondició, es podria observar que es podria descriure amb aquesta tècnica formalment el programa i validar-lo abans d'iniciar la implementació.

2.10 Derivació formal de programes

La **derivació** formal intenta demostrar matemàticament que un programa compleix amb les especificacions.

El disseny formal d'algorismes utilitza mètodes formals per abordar les seves tres fases fonamentals:

- Especificació
- Derivació
- Verificació

L'especificació formal d'algorismes és una tècnica que permet especificar la semàntica internacional d'un algorisme de manera breu i precisa. A partir d'una especificació formal es poden deduir formalment (**derivar**) les instruccions que ha de contenir l'algorisme i, un cop escrit, verificar-ne formalment la correcció.

La **verificació** formal d'algorismes té una gran importància, ja que permet verificar la correcció d'un algorisme abans de ser escrit en un llenguatge concret. És el que es coneix com a *verificació a priori*. Tradicionalment els algoritmes es proven, un cop escrits, amb l'ajuda d'eines de depuració: és el que es coneix com a *verificació a posteriori*.

Això vol dir, que fins que no es prova el programa, no apareixen els errors. Com en moltes altres activitats, com més aviat aflori correctament menys esforç costarà reparar-lo, cosa que dóna un gran valor a les tècniques formals d'especificació, derivació i verificació. La verificació formal també es pot aplicar a algoritmes que no hagin estat derivats formalment.

El principal problema de la derivació formal de programes és que es tracta d'una operació molt costosa.

2.11 Proves de programari: metodologies, IEEE, TMM

Hi ha altres temes vinculats amb les proves de programari que cal tractar per entendre-les correctament.

En primer lloc veurem com algunes metodologies de gestió de projectes de desenvolupament de programari tracten aquests apartats i quines tècniques ofereixen. L'existència d'estàndards que regulen aquests procediments, com els fets públics per l'IEEE o els models de maduresa de les proves, són altres aspectes per tenir en compte.

2.11.1 Metodologia de sistemes d'informació: Mètrica 3

Hi ha algunes metodologies de gestió de projectes que fan referència directament a la fase de proves, i indiquen, fins i tot, algunes activitats i tasques que cal dur a terme i les tècniques i eines necessàries.

Per exemple, la metodologia oferta per l'Administració pública espanyola (Mètrica 3, desenvolupada pel CASE, al Ministeri d'Administracions Pùbliques) ofereix una sèrie de fases, activitats i tasques que poden ajudar al desenvolupament d'un projecte de programari. Aquesta metodologia està més enfocada al desenvolupament de tot un sistema d'informació en una empresa, però moltes de les seves activitats i tasques serveixen per al desenvolupament del programari.

El CSAE és el Consejo Superior de Administración Electrónica. En permet l'ús lliure sempre que es faci referència a la font (MAP).

Fent la similitud amb un model en cascada o un model en V, es poden trobar fases com:

- Estudi de viabilitat.
- Anàlisis del sistema.
- Disseny del sistema.
- Construcció del sistema.
- Implementació i acceptació del sistema.
- Manteniment del sistema.

Concretament, pel que fa a les proves, es poden trobar activitats i tasques en totes i cada una de les fases que fan referència a les verificacions i validacions dels processos abans de continuar a la fase següent. Però, concretament, a la fase d'implantació i acceptació del sistema, es poden trobar activitats com:

- IAS 5. Proves d'implantació del sistema (preparació, realització i avaluació).

- IAS 6. Proves d'acceptació del sistema (preparació, realització i avaluació).

A més, ofereix explicacions de tècniques per al desenvolupament de les proves com:

- Proves unitàries
- Proves d'integració
- Proves de sistema
- Proves d'implantació
- Proves d'acceptació
- Proves de regressió
- Revisions formals
- Revisions tècniques

L'IEEE, Institute of Electrical and Electronics Engineers, és una associació internacional dedicada a l'estandardització.

2.11.2 IEEE Estàndard 829-1983

L'IEEE també ha desenvolupat normes i estàndards per als processos relacionats amb les proves en el desenvolupament del programari. Concretament l'IEEE Standard for Software Test Documentation. És l'IEEE Std 829-1983.

Aquest estàndard presenta un conjunt de normes i especificacions que indicaran, pel que fa a les activitats de proves previstes, el seu:

- Àmbit
- Enfocament
- Recursos
- Calendari

A més, queden identificades en aquest estàndard les especificacions referents al següent:

- Disseny de les proves.
- Elements dels casos de proves.
- Característiques de les proves.
- Activitats i tasques de les proves.
- Què farà cada prova.

- Informes del desenvolupament de les proves.
- Registres de proves.
- Riscos que requereixen plans de contingència.

En funció de la mida del projecte potser seguir totes les recomanacions i estàndards serà un treball molt més costós que el desenvolupament de la resta del projecte. En funció de la mida del projecte es poden dur a terme alguns controls i revisions sense tant de cost, com per exemple, fer servir un full de càlcul amb totes les informacions completes de les proves i els seus plans.

2.11.3 Test per als models de maduresa

Al llarg dels anys els responsables de projectes de desenvolupament de programari han anat millorant les seves metodologies i han agrupat esforços.

S'han creat tot tipus d'eines i de tècniques i, també, d'associacions que han creat models, com el CMM, model de maduresa de capacitats.

El CMM presenta un model d'avaluació de procediments d'una organització. Es tracta d'un model desenvolupat pensant en els processos vinculats amb projectes de desenvolupament de programari. Aquest model es considera l'estàndard de referència en la indústria de desenvolupament del programari. A partir del CMM es va crear el TMM, el test per als models de maduresa. Aquest és un model que indicarà la maduresa de les proves.

El *test maturity model* és un complement del CMM (*capability maturity model*). Hi ha la TMMi Foundation, dedicada a la seva millora i evolucions.

Actualment s'han desenvolupat els successors de tots dos models, el CMMi (*capability maturity model integration*, integració de models de maduresa de capacitats) i el TMMi (*test maturity model integration*, integració de test de models de maduresa).

El TMMi és un model detallat per a la millora del procés de proves. Aquest model consisteix en cinc nivells:

- Inicial: consisteix a verificar que el sistema funciona sense problemes ni errades greus.
- Definició: consisteix a verificar que els requisits inicials del projecte s'acompleixen.
- Integració: consisteix a verificar que no hi ha errades ni problemes en el funcionament del sistema una vegada integrat.
- Gestió i mesurament: consisteix a verificar i avaluar durant tot el cicle de vida el desenvolupament del projecte.
- Optimització: consisteix a desenvolupar nous processos i millorar el projecte per a la prevenció de possibles defectes en el futur.

2.12 Eines per desenvolupar proves

Hi ha moltes eines que ajudaran al desenvolupament de les proves. Algunes eines estan integrades als entorns de desenvolupament integrats, però altres eines són independents i caldrà seleccionar-les en funció de l'entorn i llenguatge de programació utilitzat.

Alguns exemples d'aquestes eines independents poden ser:

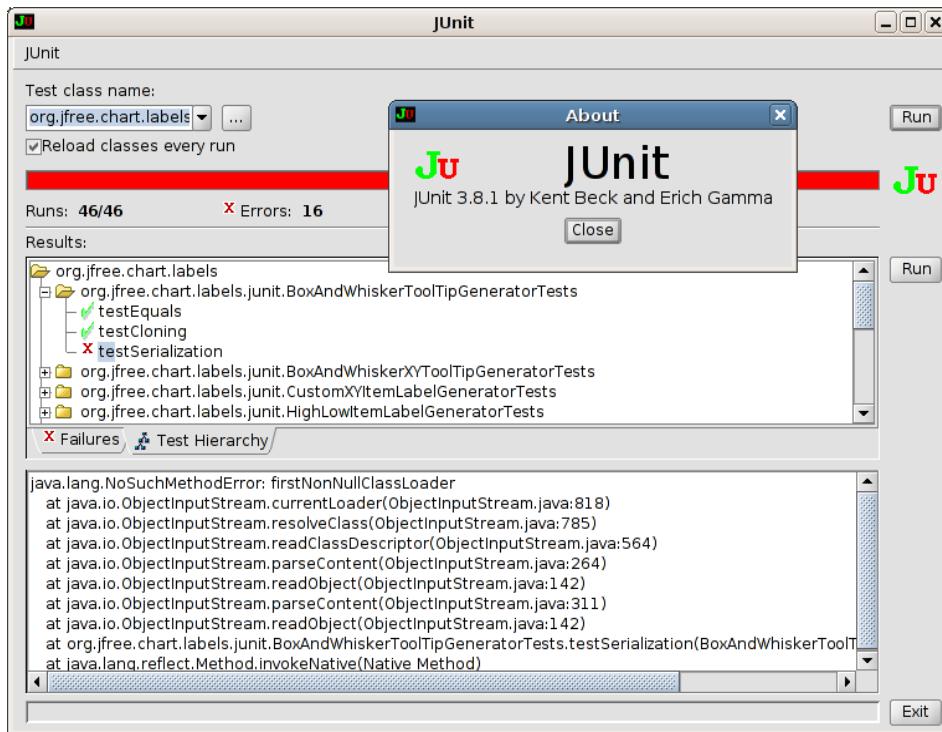
- JUnit
- NUnit
- XUnit
- Clover
- OpenSTA
- Bugzilla

2.12.1 JUnit

És una eina de proves per a projectes desenvolupats en el llenguatge de programació Java. Aquesta eina serveix per dur a terme proves unitàries executant classes Java de manera controlada per avaluar el comportament de cada mètode. Es tracta d'una eina de codi obert.

JUnit és una eina que també es fa servir per automatitzar les proves de regressió. És una eina que es pot fer servir de manera independent o integrada en altres eines de desenvolupament de Java, com l'Eclipse.

A la figura 2.5 es pot veure un exemple d'execució del programari JUnit.

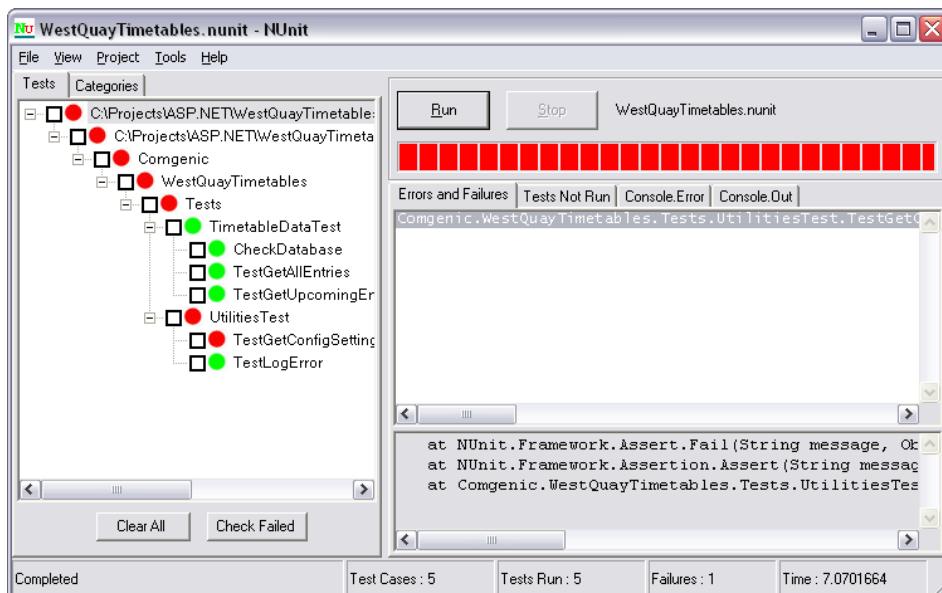
FIGURA 2.5. Exemple d'execució de JUnit.

2.12.2 NUnit

És una evolució de l'eina JUnit, però que accepta molts altres llenguatges. De fet, és un entorn de treball per a Microsoft .Net (utilitzable en Visual Studio i en Mono), de manera que permetrà treballar en tots els llenguatges .NET (Visual Basic, C#, ASP.NET, ...)

NUnit ofereix un nivell més baix d'integració amb altres eines. A la figura 2.6 es pot veure un exemple d'execució de NUnit.

A les activitats es pot trobar un exemple resolt de realització de proves amb NUnit.

FIGURA 2.6. Exemple d'execució de NUnit

2.12.3 XUnit

Igual que JUnit i NUnit, XUnit és un conjunt d'entorn de treball de proves. Són extensions de l'eina JUnit (per a aplicacions Java) o de NUnit (per a aplicacions .NET).

Aquest programari també és de codi obert i es troba en contínua evolució.

Hi ha, entre moltes altres, extensions per al següent:

- Dades, conegut com a DBUnit, que permet la connexió amb les bases de dades, i accepta diversos formats per a la càrrega de dades i per a la comparació de taules i camps.
- Càrrega massiva de dades, conegut com a DBMonster.
- Interfícies Web, conegut com a HttpUnit o JWebUnit, que permet interaccionar amb objectes de pàgines HTTP.
- Interfícies d'usuari Swing, conegut com a JFCUnit, que permet interactuar amb objectes gràfics.

2.12.4 Clover

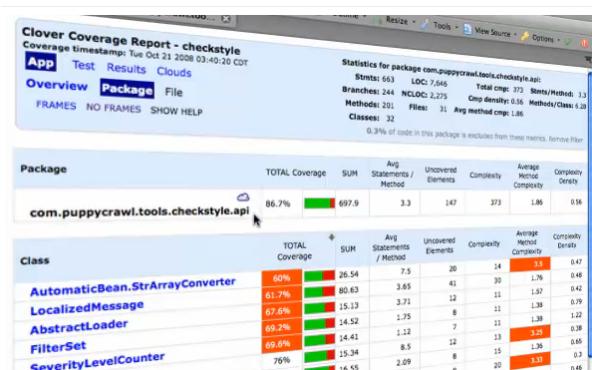


Clover és una eina que permet analitzar la cobertura de codi Java. Ofereix mètriques per a l'avaluació de l'impacte de les seves proves.

Durà a terme les proves unitàries. Clover es pot integrar amb altres eines integrades de desenvolupament del programari com l'Eclipse.

A la figura 2.7 es pot veure un exemple d'execució de Clover.

FIGURA 2.7. Execució de Clover



2.12.5 OpenSTA

OpenSTA és l'acrònim de *sistema de proves d'arquitectures obert (open systems testing architecture)*. Es va dissenyar entorn del llenguatge de programació CORBA. Actualment és un conjunt d'eines que tenen capacitat per executar seqüències de sentències HTTP i HTTPS per dur a terme proves amb càrrega massiva de dades, però també per a proves d'estrés i de rendiment.

Permet dur a terme estadístiques a partir de proves de rendiment a partir de registres virtuals d'usuaris i amb una càrrega massiva controlada de dades per part d'aquests.



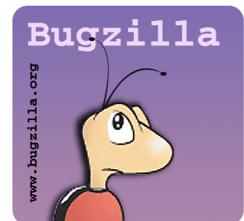
OpenSTA

2.12.6 Bugzilla

Més que un programa per dur a terme proves de programari, Bugzilla és un exemple de programari que serveix per assegurar la qualitat i per detectar errors de programari. Està inclosa en el projecte Mozilla, sota la seva llicència pública, ofert com a codi obert. Amb Bugzilla es podrà fer un seguiment de qualsevol codi web, i pot fer des d'execucions pas per pas fins a detectar la quantitat de dades que es transfereixen en cada comunicació amb el servidor o el temps de càrrega.

Alguns dels aspectes que es poden millorar amb l'ús de Bugzilla són:

- Seguiment de defectes.
- Millora de la comunicació.
- Augment de la qualitat del producte.
- Millorar la satisfacció del client.
- Augment de la productivitat.
- Adaptació a múltiples situacions.



Logo de Bugzilla