Disseny orientat a objectes

Benet Campderrich Falgueras
Recerca Informàtica, SL

P00/05007/00304

UOC
www.uoc.edu

Índex

Introducció

Aquest mòdul tracta del disseny del programari amb tècniques orientades a objectes, aplicant les notacions i conceptes d'UML i seguint el cicle de vida del Rational Unified Process. Recordem que, en aquest cicle de vida, l'anàlisi i el disseny constitueixen un sol component de procés, però nosaltres els considerarem dues etapes diferents perquè tenen una finalitat diferent i perquè el resultat de l'un i de l'altre són també clarament diferents.

Es veurà amb detall el paper del disseny en relació amb l'etapa que el precedeix, l'anàlisi, i la que el segueix, la realització. Avancem, però, que així com l'anàlisi formalitza els requisits recollits anteriorment, el disseny és el primer pas de l'elaboració d'una resposta a aquests requisits.

Un dels avantatges esperats de la tecnologia orientada a objectes, i potser el més important, és la posibilitat de reutilitzar programari. Durant el disseny es decideix què es reutilitza i què es fa de bell nou i, per tant, en aquest mòdul hem de tractar les tècniques de reutilització.

Dins el disseny distingirem els passos següents:

- El disseny arquitèctònic.
- El disseny dels casos d'ús.
- L'obtenció del diagrama estàtic de disseny.
- L'especificació de les classes del disseny.
- El disseny de la persistència.
- El disseny de la interfície d'usuari.
- El disseny dels subsistemes.

Objectius

En aquest mòdul aprendreu a fer el disseny de programari orientat a objectes, fent servir els conceptes i notacions d'UML. Aquest objectiu es compon dels objectius secundaris següents:

1. Entendre el paper del disseny i les diferències amb l'etapa anterior (l'anàlisi) i la següent (la realització).

2. Entendre les opcions de reutilització que hi ha i com es fan servir.

3. Saber especificar la implementació dels casos d'ús descrits en el model d'anàlisi.

4. Aprendre a elaborar el diagrama estàtic de disseny.

5. Aprendre a dissenyar la interfície d'usuari.

1. El paper del disseny

L'etapa de disseny fa de pont entre l'anàlisi i la realització. El model de l'anàlisi descriu les funcions que ha de fer el programari i també especifica els eventuals requisits no funcionals, és a dir, planteja el problema que s'ha de resoldre amb el programari. Les etapes següents tindran a veure amb l'elaboració d'una solució d'aquest problema. Ara bé, el model de l'anàlisi no és una base adequada per a emprendre directament la realització, ja que no s'expressa en termes de la tecnologia que s'haurà de fer servir en el projecte (principalment el llenguatge de programació i l'eina de suport de la interfície gràfica d'usuari).

Justificació de l'etapa de disseny

És clar que en especificar els requisits s'hauran tingut en compte les possibilitats generals de la tecnologia disponible, a fi de no demanar requisits impossibles ni tampoc d'infrautilitzar aquesta tecnologia en cas que permetés d'atendre més necessitats de l'usuari o d'atendre-les millor (per exemple, avui en dia l'anàlisi considera implícitament que la presentació de la informació als usuaris es farà mitjançant pantalles amb suport de gràfics i impressores modernes). Però segurament no s'haurà tingut en compte, si no és d'una manera molt general, en quin llenguatge de programació s'implementaran les classes i quines classes programades en projectes anteriors es reutilitzaran, o quin sistema de gestió de bases de dades es farà servir.

Dins el Rational Unified Process, el disseny es fa principalment al final de la fase d'elaboració i al començament de la de construcció.

1.1. La relació entre el disseny i la realització

La realització dóna com a producte el programari acabat, i comprèn la programació pròpiament dita més la generació de fitxers binaris i executables, la prova un a un d'aquests i l'enllaç de tot plegat. Les classes definides en el disseny es programen una a una, i es generen els components i els subsistemes. Entre el model del disseny i el programari acabat hi ha la mateixa relació que entre el projecte d'un edifici i l'edifici construït.

1.2. La utilitat del disseny

Hem vist que en projectes molt senzills es pot arribar a prescindir de l'anàlisi. En canvi, el disseny cal sempre.

Preguntar per a què serveix el disseny és el mateix que preguntar per què no es pot implementar directament el model d'anàlisi. Jacobson, Booch i Rumbaugh esmenten una relació d'1 a 5 entre el cost de l'anàlisi i el del disseny; aquest fet ja indica que en el model del disseny hi ha moltes més classes i operacions que cal identificar i especificar abans de programar-les.

2. La reutilització

La reutilització en el disseny orientat a objectes té quatre modalitats: la reutilització de classes, la reutilització de components, els patrons i els bastiments.

Meyer esmenta els avantatges de la reutilització següents:

- S'abreuja el desenvolupament del programari.
- Disminueix la feina de manteniment del programari en el futur.
- Millora la fiabilitat.
- Sovint el codi reutilitzat és més eficient, probablement perquè s'ha anat millorant al llarg del temps. És cert que el codi fet a mida per a un cas concret podria ser més eficient, però a la pràctica no és possible optimitzar tot el codi d'un projecte.
- Es guanya en coherència: normalment no es reutilitza una sola classe, sinó diverses, procedents d'una mateixa llibreria i que, per tant, segurament estaran programades segons les mateixes normes i amb el mateix estil.
- Preservació de la inversió: si es reutilitza un bon fragment de codi, serà moltG més difícil que es perdi, ja que n'hi haurà moltes còpies.

2.1. La reutilització de classes

Principalment es reutilitzen classes que tenen funcions independents del domini, com ara elements d'interfície gràfica i estructures de dades generals. En general, la reutilització de classes d'un domini concret només és viable en el cas de sistemes de programari d'una mateixa organització. Les classes que es reutilitzen se solen agrupar per funció* en llibreries com ara els paquets de Java.

*** Per exemple, estructures de dades.**

La reutilització de classes es pot fer de diverses maneres. La més senzilla i directa és reutilitzar la classe mateixa, en forma font o compilada. Però si se li han d'afegir o modificar operacions o atributs, cal complementar-la amb subclasses o bé mitjançant agregació. Si la classe és abstracta, l'única manera de reutilitzar-la és, evidentment, amb subclasses.

Vegeu les relacions d'agregació entre classes al subapartat 4.3.5 del mòdul "Anàlisi orientada a objectes".

2.2. La reutilització de components

Un component és un conjunt de classes, els objectes de les quals col·laboren en temps d'execució amb vista a dur a terme una funció concreta.

Un component implementa una interfície determinada, que és el conjunt de totes aquelles operacions de les seves classes que es poden demanar des de

l'exterior del component. Per tant, es pot substituir un component per un altre que implementi la mateixa interfície.

Un component, pel fet que implementa una interfície, es comporta com una classe, i les classes que el componen no són visibles des de l'exterior. Perquè un component es pugui reutilitzar correctament, cal conèixer-ne la interfície i també el contracte. El **contracte** descriu els efectes de les operacions compreses en la interfície i quins invariants manté.

2.3. Els patrons

Els patrons* són una manera organitzada de recollir l'experiència dels dissenyadors de programari per a tornar-la a utilitzar en casos semblants.

* En anglès, *patterns*.

Quan un expert treballa en un problema, rarament inventa una solució enteramente nova, partint de zero, sinó que en general recorda algun cas que té semblances amb el seu i n'adapta la solució; això és el que acostumem a anomenar aplicar l'*experiència*, i se suposa que és el que fa que els experts ho siguin.

No cal dir que seria molt millor tenir aquesta experiència recollida i documentada de manera més o menys formal per tal de fer-la accessible a d'altres experts o futurs experts, d'una banda, i per a fer més sistemàtica i coherent l'aplicació de l'experiència pels experts mateixos, d'una altra.

Un patró no és un programa, encara que pot incloure un programa com a mostra, però no pas per a utilitzar-lo directament.

Un **patró** és una idea de disseny i implementació detallada i pràctica (una "recepta") que constitueix un esbós de solució d'un problema que es presenta amb una certa freqüència. Els detalls d'aquesta solució canvien per a cada cas a què s'aplica. Per tant, el nucli d'un patró és un parell problema-solució.

En general, un patró no està lligat a un mètode concret de disseny. De fet, molts patrons es poden fer servir tant en disseny orientat a objectes com en disseny estructurat, si bé els patrons tendeixen a estar documentats en forma orientada a objectes, simplement perquè s'han divulgat quan aquesta tecnologia ja estava en voga.

Els patrons representen un nou intent d'augmentar la reusabilitat del programari (com la tecnologia d'objectes, per exemple) partint de la idea que en

casos determinats en què no es pot reutilitzar codi, almenys se'n pot reutilitzar el disseny, si més no les idees bàsiques.

De la mateixa manera que els entorns de desenvolupament orientats a objectes acostumen a oferir una àmplia biblioteca de classes predefinides, en el disseny orientat a patrons es pot disposar de "receptaris" de patrons preexistents, que cal esperar que es vagin enriquint en el curs del temps; a més també cal esperar que en vagin apareixent (ja han començat a fer-ho) d'específics per dominis*.

*** Com ara programació en temps real, programació distribuïda i molts altres.**

El principal guany que s'espera obtenir de la utilització de patrons és que no cal pensar una solució per a molts dels problemes de disseny més freqüents i, per tant, hom pot concentrar l'esforç de disseny en els aspectes més innovadors de cada projecte.

2.3.1. Característiques dels patrons

Les característiques més importants que presenten els patrons són les que esmentem a continuació:

1) Recullen l'experiència, és a dir, són un extracte i un denominador comú de nombrosos dissenys anteriors: no s'inventen en un moment donat.

2) Són quelcom més ampli que, per exemple, una classe o un objecte.

3) Creen vocabulari: dins un disseny es fa referència als patrons per llur nom. En particular, dins un patró es pot fer referència a d'altres.

4) Són un instrument de documentació, atès que dins la documentació del disseny, una referència a un patró estalvia de descriure amb detall una part del disseny.

5) Si es fa ús del mateix patró, faciliten la coherència dels dissenys pertot on es presenta el mateix problema.

6) No donen una solució completa a un problema en un cas concret: només proporcionen una solució genèrica que cal completar.

7) Ajuden a fer front a la complexitat del disseny, bo i resolent-ne d'entrada algunes parts.

2.3.2. Components dels patrons

Patrons i drets de propietat

Sens dubte es plantejaran, si encara no s'han presentat, qüestions com ara patents i propietat intel·lectual sobre patrons, patrons que siguin secret d'empresa, etc.

Un patró té les quatre parts que esmentem a continuació:

1) El **nom**: l'assignació de nom a un patró no és una qüestió trivial, ja que, pel fet que creen vocabulari, és vital que en els noms dels patrons no hi hagi sinònims

o àlies, ni menys encara homònims; i això, no tan sols pel que fa al projecte o empresa, sinó quant a la literatura sobre patrons en general, atès que qualsevol expert que es miri el disseny ha de donar-hi la mateixa interpretació.

2) El **context**: és l'entorn o la situació dins la qual es presenta el problema que el patró resol; sovint es defineix en termes d'una llista de situacions.

3) El **problema**: es defineix en termes d'allò que es coneix com a forces: requisits que la solució ha de complir, restriccions i qualitats que es desitja que tingui la solució; de vegades, les forces són contraposades.

4) La **solució**: és un compromís entre les forces. La solució té els dos aspectes següents:

- Estàtic o estructural, que s'expressa en termes de components i relacions entre ells.
- Dinàmic o de comportament, que descriu les funcions de cada component i com col·laboren entre si al llarg del temps.

2.3.3. Classificacions dels patrons

S'admeten diversos tipus de classificacions per als patrons. En concret, se'ls pot classificar segons el nivell, el propòsit o l'àmbit:

1) En la **classificació per nivell**, alguns autors parlen simplement de patrons de disseny; d'altres en distingeixen tres classes:

- **Patrons arquitectònics**: es fan servir durant el disseny arquitectònic i tracten sobre l'estructuració d'un sistema de programari en subsistemes.
- **Patrons de disseny** pròpiament dits: patrons adreçats a resoldre problemes puntuals del disseny.
- **Frases fetes (*idioms*)**: descriuen la manera d'implementar quelcom (eventualment un component d'un patró de disseny) en un llenguatge de progamació concret.

Diversitat de definicions per a *patró*

La inclusió de les frases fetes permet veure que no tots els autors entenen el mateix per patró, ja que algunes definicions afirmen que els patrons no han d'estar lligats a un llenguatge de programació.

2) La **classificació per propòsit** estableix la jerarquia següent:

- **Patrons de creació**: es fan servir per a la instanciació, per tal de fer que el programari sigui independent de com les classes i objectes es creen, representen i agreguen. Els patrons al nivell de classe fan servir herència per a variar la classe, mentre que al nivell d'objectes es delega la instanciació en un altre objecte.

- **Patrons d'estructura**: fan referència a la manera com les classes i els objectes es combinen per a formar estructures més grans. Al nivell de classe es fa ús de l'herència per a agregar interfície o implementació, i al nivell d'objectes es fa agregació en temps d'execució.

- Patrons de comportament: descriuen la comunicació entre classes i objectes; al nivell de classes s'aplica l'herència i al nivell d'objectes, l'agregació.

3) La **classificació per àmbit** dels patrons inclou els elements següents:

- **Patrons sobre classes**: fan referència a les relacions d'herència, que s'estableixen en temps de compilació.

- **Sobre objectes**: fan referència a les relacions entre objectes, les quals poden variar dinàmicament en temps d'execució.

2.3.4. Un exemple de patró: *Composite*

El patró *Composite* té com a objectiu representar jerarquies d'estructures part-tot.

Lectura complementària

Trobareu el patró *Composite* explicat amb més detall a l'obra següent:

E. Gamma; R. Helm; R. Johnson; J. Vlissides (1995). *Design Patterns*. Addison-Wesley.

Aquest patró es pot fer servir en els casos que s'esmenten a continuació:

a) Quan es volen representar jerarquies de tipus d'agregació entre objectes.

b) Quan es vol que els clients* no necessitin distingir entre objectes compostos i no compostos. Els clients tractaran tots els objectes de la mateixa manera.

*** És a dir, els objectes que usen l'objecte eventualment compost.**

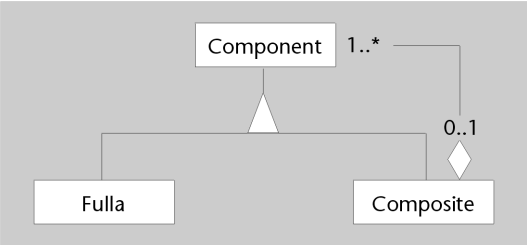

Figura 1

Les classes que figuren en aquesta estructura tenen les funcions següents:

a) La classe ***Component*** implementa el comportament dels objectes eventualment compostos i declara una interfície per a accedir als seus components i gestionar-los.

b) La classe ***Fulla*** representa els objectes que no tenen components.

c) La classe ***Composite*** defineix el comportament dels components que tenen components i emmagatzema aquests darrers.

2.3.5. Sistemes de patrons

Ja s'ha dit que els patrons sovint fan referència a d'altres patrons, ja sigui perquè es fan servir en llur implementació, ja sigui perquè els complementen. Per tant, més que de *llistes de patrons* parlarem de *sistemes de patrons relacionats*. La documentació d'un sistema de patrons serà quelcom més que la suma de la documentació de cadascun d'ells: haurà d'incloure també un esquema de les relacions entre aquests i la descripció de les dependències entre ells quan s'utilitzen plegats.

Algunes característiques desitjables en un sistema de patrons són les que enumerem a continuació:

1) El sistema de patrons ha de tenir un nombre suficient de patrons de diverses classes o nivells; altrament no es pot fer disseny basat en patrons, sinó només disseny ordinari utilitzant algun patró.

2) Tots els patrons del sistema han d'estar descrits d'una manera tan uniforme com sigui possible.

3) Les relacions entre patrons han d'estar indicades explícitament.

4) El catàleg de patrons del sistema ha d'estar organitzat de manera que sigui fàcil de trobar el patró més adequat a cada cas; en particular, hi ha d'haver un sistema de criteris de classificació com els descrits abans.

5) El sistema de patrons ha de ser pràctic, en el sentit que es vegi fàcilment com es poden utilitzar i implementar els patrons.

6) El sistema ha de ser obert: s'hi han de poder integrar amb facilitat nous patrons, que s'han de poder incorporar de manera natural a l'esquema de classificació esmentat, i els patrons que ja existeixen s'han de poder adaptar als canvis tecnològics.

Quan es parla de *sistemes de patrons* es pensa principalment en sistemes públics, per exemple, descrits en llibres. Però també hi pot haver sistemes per a ús intern d'una empresa o d'un projecte, que probablement incloguin també patrons públics.

Els sistemes de patrons més coneguts són els que esmentem a continuació:

a) El sistema de Gamma, Helm, Johnson i Vlissides.

b) El sistema de Buschmann, Meunier, Rohnert, Sommerlad i Stal.

c) Els patrons documentats per Larman, que corresponen a principis generals de disseny més que a problemes específics.

La Banda dels quatre

Els autors Gamma, Helm, Johnson i Vlissides es coneixen de manera informal amb el nom de *Gang of Four* (La Banda dels quatre).

Aquests sistemes es van publicar en el mateix ordre en què s'han esmentat, i cada sistema fa referències als anteriors. Per aquesta raó creiem que es poden considerar un sol sistema. N'esmentem tots els patrons en una única taula que veieu a continuació. La descripció detallada de cada patró es pot trobar a les obres citades a la bibliografia d'aquest mòdul.

Nom	Tipus	Sistema	Propòsit
Abstract Factory	Creació	GHJV	Proporciona una interfície per a crear objectes relacionats sense necessitar saber-ne la classe.
Adapter	Estructura	GHJV	Substitueix la interfície d'una classe, permetent així reutilitzar algunes classes.
Blackboard	Arquitectònic	BMRSS	Coordina la comunicació entre els components de sistemes de programari distribuïts.
Bridge	Estructura	GHJV	Desacobla una classe abstracta de la seva implementació per tal que puguin variar independentment.
Builder	Creació	GHJV	Permet que la construcció d'un objecte complex en pugui crear diferents representacions.
Chain of responsibility	Comportament	GHJV	Implementa crides d'operacions a través d'altres objectes.
Client-Dispatcher-Server	Estructura	BMRSS	Proporciona transparència respecte a la ubicació i els mecanismes de connexió entre client i servidor.
Command	Comportament	GHJV	Converteix una crida d'una operació en un objecte que es pot manipular.
Command Processor	Comportament	BMRSS	Separa la petició d'una operació de la seva execució amb vista a permetre desfer-la, per exemple.
Composite	Estructura	GHJV	Construeix una jerarquia de composició d'objectes.
Controller	Comportament	Larman	Assigna el tractament dels esdeveniments a aquelles classes que representen el sistema o l'organització en conjunt, o un paper o un cas d'ús.
Creator	Estructura	Larman	Assigna a la classe B la responsabilitat de crear els objectes d'una altra classe A si B ja agrega, conté, enregistra o usa de manera principal objectes d'A o té la informació per a inicialitzar-los.
Decorator	Estructura	GHJV	Afegeix responsabilitats a un objecte dinàmicament, sense afegir-les a la classe.
Don't call to Strangers	Comportament	Larman	Limita la varietat d'objectes als quals demana operacions un objecte dins la implementació de les seves operacions: només ell mateix o un atribut seu, els paràmetres de l'operació o els objectes creats dins el mètode.

Nom	Tipus	Sistema	Propòsit
Expert	Comportament	Larman	Assigna cada operació a aquella classe que té els atributs que hi intervenen.
Facade	Estructura	GHJV	Proporciona una interfície única a un subsistema amb molts objectes.
Factory Method	Creació	GHJV	Defineix una interfície per a instanciar un objecte, la classe del qual es decideix a nivell de subclasses.
Flyweight	Estructura	GHJV	Permet que molts petits objectes puguin ser compartits fent que llur estat sigui independent del context on se'ls usa.
Forwarder-Receiver	Estructura	BMRSS	Aïlla dels mecanismes de comunicació la comunicació entre processos.
High Cohesion	Comportament	Larman	Assigna les operacions a les classes de manera que la cohesió del disseny sigui alta (això és, que cada classe implementi un conjunt no gaire gran d'operacions molt relacionades).
Indirection	Comportament	Larman	Assigna una operació a una classe intermediària per tal de desacoblar la classe client de la servidora.
Interpreter	Comportament	GHJV	Representa les regles d'un llenguatge mitjançant classes.
Iterator	Comportament	GHJV	Accedeix als components d'un objecte agregat un darrere l'altre, sense veure'n el contingut.
Layers	Arquitectònic	BMRSS	Estructura una aplicació en grups –de diferents nivells– de substasques
Low Coupling	Comportament	Larman	Assigna les operacions a les classes de manera que l'acoblament entre les classes sigui baix, cosa que vol dir que cada classe faci poc ús d'operacions d'altres classes dins les seves.
Master-Slave	Estructura	BMRSS	El mestre reparteix feina a diversos esclaus idèntics i obté el resultat final a partir dels resultats obtinguts per aquests.
Mediator	Comportament	GHJV	Defineix un objecte que encapsula la interacció d'altres.
Memento	Comportament	GHJV	Captura i externalitza l'estat d'un objecte (sense violar-ne l'encapsulació), per tal que pugui ser restituït més endavant.
Microkernel	Arquitectònic	BMRSS	Dins un programari, separa un nucli funcional de la funcionalitat afegida.
Model-View-Controller	Arquitectònic	BMRSS	Descompon un programari interactiu en tres components: el Model que en conté la funcionalitat i les dades, la Vista que presenta la informació a la pantalla i el Controlador que tracta les entrades d'informació.
Observer	Comportament	GHJV	Defineix una dependència entre objectes de manera que quan canvia l'estat d'un objecte es comunica el canvi a uns altres.
Pipes and Filters	Arquitectònic	BMRSS	Proporciona un estructura de tubs i filtres per a tractar fluxos seqüencials de dades
Polymorphism	Comportament	Larman	Convé aplicar el polimorfisme (donar el mateix nom a operacions anàlogues de classes diferents) per tal que l'objecte client no hagi de preguntar per la classe abans de demanar-li l'operació.

Nom	Tipus	Sistema	Propòsit
Presentation-Abstraction-Control	Arquitectònic	BMRSS	Descompon un programari interactiu en una jerarquia d'agents que col·laboren. Cada agent fa alguna funció de presentació per pantalla, funcionalitat bàsica (abstracció) o comunicació entre agents (control).
Prototype	Creació	GHJV	Instancia objectes fent còpies d'un prototip.
Proxy (1)	Estructura	GHJV	Proporciona un representant per a un objecte, que no cal crear fins que es necessiti efectivament; també en pot controlar l'accés.
Proxy (2)	Estructura	BMRSS	Fa que els clients d'un component demanin les operacions a un representant d'aquest i no a ell directament, per raons d'eficiència, control d'accés o altres.
Publisher-Subscriber	Comportament	BMRSS	Un publicador notifica els seus canvis a diversos subscriptors. És una variant d'Observer.
Pure Fabrication	Estructura	Larman	De vegades convé definir una classe que no correspon a cap entitat del domini sinó que és una "pura invenció" per agrupar operacions de manera més coherent que si estiguessin assignades a classes d'entitats.
Reflection	Arquitectònic	BMRSS	Proporciona un mitjà per a poder canviar dinàmicament l'estructura i comportament d'un programari.
Singleton	Creació	GHJV	Garanteix que d'una classe només n'hi hagi un objecte.
State	Comportament	GHJV	Fa que quan un objecte canvia d'estat canviï el seu comportament com si hagués canviat de classe.
Strategy	Comportament	GHJV	Encapsula diversos algorismes fent-los intercanviables.
Template Method	Comportament	GHJV	Defineix l'esquelet d'un algorisme d'una operació, alguns passos del qual es concretaran en subclasses.
View Handler	Comportament	BMRSS	Permet als seus clients d'obrir, manipular i tancar les presentacions de les dades i en gestiona les dependències.
Visitor	Comportament	GHJV	Substitueix la interfície d'una classe, permetent així reutilitzar algunes classes.
Whole-part	Estructura	BMRSS	Fa que una operació sobre els components d'una estructura pugui canviar sense que els components canviïn de classe.

GHJV = Gamma, Helms, Johnson i Vlissides
BMRSS = Buschmann, Meunier, Rohnert, Sommerlad i Stal

2.3.6. Els patrons per a l'ajut en la resolució de problemes de disseny

Els patrons serveixen d'ajuda a l'hora de resoldre problemes de disseny en els aspectes següents:

a) Suggereixen classes i objectes.

b) Suggereixen interfícies entre objectes, independents de la implementació d'aquests*.

* Per exemple, mitjançant classes i operacions abstractes.

c) Ofereixen possibilitats de reutilització de codi mitjançant superclasses i objectes que es puguin emprar com a components d'altres.

d) Suggereixen possibilitats de delegar operacions d'una clase cap a una altra.

e) Els sistemes de programari són més fàcils de modificar pel fet que moltes vegades els patrons es poden implementar de diverses maneres, tot mantenint-ne estables les interfícies.

Més endavant veureu l'ús de diversos patrons combinats per a resoldre un sol problema.

2.3.7. Selecció del patró adient

Per a seleccionar el patró adient en cada cas, cal suposar que es disposa d'un únic sistema de patrons (si inicialment se'n tenien diversos, convé consolidar-los) amb un catàleg adequat.

El procés de selecció del patró adient consisteix a seguir els passos següents:

1) Primer cal especificar per escrit el problema que intentem resoldre amb patrons, i si es veu que consta de parts ben diferenciades, descompondre'l en subproblemes; de cada subproblema s'han de descriure les forces que l'afecten.

2) En segon lloc, haurem de fer una primera delimitació del conjunt de patrons aplicables mitjançant el catàleg.

3) A continuació, afinarem més la selecció tenint en compte els objectius dels patrons seleccionats abans; interessaran tant els patrons que contemplen tot el (sub)problema com també aquells que en poden resoldre alguna part.

4) Després, afegirem a la selecció tots els patrons relacionats directament amb els anteriors.

5) Aleshores considerarem els patrons que tenen com a objectiu facilitar les modificacions del programari futures, i inclourem a la selecció aquells que siguin aplicables.

6) Més tard considerarem els avantatges i inconvenients descrits en la documentació, i avaluarem quina importància tenen en el cas considerat.

7) Finalment, quan un patró seleccionat tingui variants, seleccionarem la variant més adient.

2.3.8. Utilització d'un patró

A continuació es descriu la manera de fer servir un patró:

1) Es fa una lectura general d'aquells punts de la documentació del patró que no s'haguessin llegit encara.

2) S'estudien amb detall els apartats sobre participants, estructura i paper dels participants.

3) S'estudia l'exemple resolt.

4) Se substitueix la terminologia abstracta del patró per la del projecte i s'estableix una llista d'equivalències.

5) Es defineixen les classes noves necessàries, i es modifiquen les classes ja existents que calgui entre les que ja s'havien definit durant el projecte.

2.4. Bastiments d'aplicacions

Un bastiment* és un conjunt de classes que constitueix una aplicació incompleta i genèrica. Si el bastiment es complementa de manera adequada (si s''especialitza'), s'obtenen aplicacions especialitzades d'un cert tipus.

*** En anglès, *framework*.**

Hi ha dos tipus de bastiments:

1) Un **bastiment de caixa blanca** consta d'un conjunt de classes (anomenades **abstraccions**) de les quals hi ha definida tant la interfície com la implementació. Per a especialitzar-lo, cal implementar subclasses d'aquestes classes.

2) Un **bastiment de caixa negra**, en comptes d'abstraccions, té definides unes interfícies anomenades **papers***. Per a especialitzar-lo, cal afegir-li classes que n'implementin els papers.

*** En anglès, *roles*.**

2.4.1. Avantatges i inconvenients dels bastiments

Alguns **avantatges dels bastiments** són els que esmentem a continuació:

a) Redueixen la feina de programació i manteniment d'aplicacions. Els bastiments redueixen la codificació i la posada a punt, ja que proporcionen subsistemes que sabem que funcionen. En definitiva, subministren codi que ja no s'haurà de tornar a escriure ni mantenir.

b) Proporcionen una arquitectura per al programari.

c) Porten a desenvolupar petites aplicacions que encaixen dins els bastiments en comptes d'aplicacions monolítiques.

d) Són una bona base per a la indústria de components de programari. Els bastiments ben dissenyats permeten que terceres companyies puguin subministrar components o parts de components que els desenvolupadors podran afegir.

Com a **inconvenients dels bastiments** podem enumerar els següents:

a) Limiten la flexibilitat. Els components que es construeixin per a un bastiment han d'emmotllar-se a les restriccions imposades per l'arquitectura d'aquest.

b) Dificulten l'aprenentatge. Abans de fer servir un bastiment per a construir aplicacions, cal estudiar-lo a fons. Aquest aprenentatge es pot estimar en un temps inicial de 3 setmanes (per a un bastiment de complexitat mitjana) més un temps addicional amb un termini mitjà determinat per l'arquitectura de l'aplicació que cal construir. De totes maneres, aquest aprenentatge només s'ha de fer una vegada i pot servir per a moltes aplicacions basades en el bastiment.

c) Redueixen del grau de creativitat del treball dels desenvolupadors.

2.4.2. Comparació entre bastiments i patrons

Finalment, establirem una comparació entre els bastiments i els patrons. Aquesta comparació es centra en els aspectes següents:

1) Els patrons són més petits: un bastiment pot contenir diversos patrons, mai el contrari.

2) Els patrons són més abstractes: cada vegada que s'aplica un mateix patró, produeix programes diferents.

3) Els patrons són menys especialitzats, ja que es poden utilitzar en qualsevol mena d'aplicació.

3. El disseny arquitectònic

El disseny arquitectònic té com a objectiu definir les grans línies del model del disseny.

El disseny arquitectònic comprèn les activitats següents: establir la configuració de la xarxa, decidir la utilització d'un bastiment ja disponible, si escau, i establir els subsistemes, llurs interfícies i les dependències entre aquests.

3.1. Establiment de la configuració de la xarxa

L'establiment de la configuració de la xarxa consisteix a determinar els nodes que hi haurà i les característiques que tindran, les connexions entre aquests i com seran, i els protocols de comunicacions quant a amplada de banda, fiabilitat i qualitat.

3.2. Establiment dels subsistemes

Els subsistemes poden ser propis del projecte, reutilitzats d'altres projectes o bé programari del mercat, com ara programari de sistemes i programari intermediari o *middleware*.

Vegeu el concepte de *middleware* al subapartat 3.1 del mòdul "Introducció al programari distribuït" d'aquesta assignatura.

Es pot partir de la descomposició del programari en paquets fets a l'etapa d'anàlisi, i les modificacions que s'hi faran normalment consistiran a segregar algun subsistema d'un paquet de servei amb vista a reutilitzar-lo en diversos llocs del programari, o a separar en subsistemes diferents allò que ja està fet d'allò que s'ha de dur a terme en el projecte, o a permetre de repartir un paquet entre diversos nodes. També convé tenir en compte la possibilitat d'aplicar patrons arquitectònics.

Per **interfície d'un subsistema** entenem les operacions que es poden demanar al subsistema des d'altres subsistemes. Ara bé, a part d'aquestes interfícies no necessàriament definides de manera explícita, per tal d'aconseguir un disseny més flexible amb vista a modificacions futures, pot ser convenient de definir interfícies explícites implementades per subsistemes que es poden anar substituint al llarg del temps, igual que en el cas d'interfícies de classes considerat de manera estàndard a UML.

És especialment important definir interfícies per a aquells subsistemes que corresponen a programari del sistema o a *middleware,* atès que cal esperar que vagin evolucionant al llarg del temps d'una manera independent del programari que desenvolupem.

Les dependències entre subsistemes es presenten quan hi ha elements d'un subsistema que tenen relacions amb elements d'un altre. Com a mínim hi haurà totes aquelles dependències que ja existeixen entre els paquets d'anàlisi i de serveis corresponents.

4. El disseny dels casos d'ús

Atès que els requisits es van recollir essencialment en forma de casos d'ús, una manera lògica d'enfocar el disseny és descriure'n la implementació de cadascun, partint de la versió revisada i documentada amb diagrames d'interacció obtinguda a l'etapa d'anàlisi.

El **disseny de la implementació dels casos d'ús** (que en endavant anomenarem simplement **disseny dels casos d'ús**) parteix del diagrama de col·laboració resumit que s'ha fet a l'anàlisi, i es consideren per separat les classes de frontera, d'entitats i de control.

Les classes de frontera són la base de partida del disseny de la interfície d'usuari. El disseny de les classes d'entitats inclou el disseny dels instruments per a gestionar-ne la persistència. Del disseny de la interfície amb l'usuari i de la persistència, no en parlarem aquí, sinó en altres apartats.

Vegeu el disseny de la persistència i el disseny de la interfície d'usuari, respectivament, als apartats 6 i 7 d'aquest mòdul didàctic.

La implementació de la funcionalitat dels casos d'ús es fa dins les classes de control i les classes d'entitats. Atès que aquesta funcionalitat pot ser extremadament variada, només es poden donar algunes regles generals sobre la seva distribució entre classes i les operacions d'aquestes:

a) Convé que una de les classes de control dirigeixi tot el procés del cas d'ús.

b) Cal aprofitar les oportunitats de reutilitzar components i aplicar patrons. Pel que fa a les altres formes de reutilització, l'ús de bastiments ja s'haurà considerat durant el disseny arquitectònic i la possibilitat de reutilització de classes s'estudia sistemàticament més endavant. Tanmateix, si es troben possibilitats òbvies de reutilitzar alguna classe, cal fer-ho.

El procés del disseny de les classes de control és aquest: s'estudia la implementació de les operacions ja identificades a l'anàlisi, una a una; sovint es troba que per a implementar-ne una, calen noves operacions de classes ja definides o també de classes noves; després caldrà estudiar la implementació d'aquestes operacions noves, etc.

Com veiem, mentre es dissenyen els casos d'ús es van incorporant classes i operacions noves al diagrama estàtic d'anàlisi, que així esdevé el **diagrama estàtic de disseny**. Un cop acabat el disseny dels casos d'ús, caldrà revisar-lo de la manera que descriurem més endavant, cosa que pot portar a fer retocs també al disseny dels casos d'ús.

5. Revisió del diagrama estàtic de disseny

Com ja hem vist, l'obtenció del diagrama estàtic de disseny no és un pas independent, sinó que es va fent essencialment durant el disseny dels casos d'ús. Un cop acabat aquest, resta per fer una revisió del diagrama obtingut.

El diagrama revisat, juntament amb l'especificació de les operacions de les seves classes, serà la base per a la implementació de les classes de control i d'entitats.

Diferències entre el diagrama estàtic de disseny i el d'anàlisi

Generalment, el diagrama estàtic de disseny conté moltes més classes que el d'anàlisi (ja hem esmentat abans un factor típic de 5 a 1). Però això no vol dir que el diagrama estàtic de disseny tingui totes les classes que tindrà el programari un cop implementat. Deixant de banda les classes de la interfície gràfica, si l'eina utilitzada és orientada a objectes, durant la programació es definiran moltes més classes que tindran un paper purament instrumental i que generalment no es reflecteixen en un diagrama, ja que aquest seria molt complex (tanmateix no es descarta fer diagrames estàtics parcials si es creu convenient).

La revisió del diagrama estàtic de disseny té en compte els aspectes següents: la normalització dels noms, la reutilització de classes, l'adaptació de l'herència al nivell suportat pel llenguatge de programació, la millora del rendiment, l'increment de la velocitat i la reducció del trànsit de missatges mitjançant l'agrupació de classes, l'addició de classes temporals per a emmagatzemar resultats intermedis, i la revisió des del punt de vista de cohesió i acoblament.

Totes aquestes raons poden justificar la modificació del model creat anteriorment i ho farem en un nou diagrama per respectar el que hem pactat amb el nostre client.

5.1. Normalització dels noms

En la recollida i documentació de requisits és molt possible (i fins i tot recomanable) que hàgim utilitzat la terminologia del client per a facilitar la comunicació amb ell. Per continuïtat, segurament serà bo emprar la mateixa terminologia al diagrama estàtic de l'anàlisi i a l'especificació formal dels casos d'ús feta dins l'anàlisi i, per tant, al disseny dels casos d'ús.

Però fins i tot si s'ha anat en compte de no fer servir noms no suportats pels llenguatges de programació, pot ser que calgui canviar alguns noms de classes, atributs i operacions, bé perquè vulnerin alguna norma aplicable al projecte o bé per respectar una terminologia ja establerta en projectes anteriors. Això és important amb vista a la reutilització de classes, ja que la llibreria de classes pot contenir-ne moltes i poden ser difícils de trobar, si no es fa servir una nomenclatura unificada.

5.2. Reutilització de classes

Durant el disseny dels casos d'ús, les classes es fan "a mida" d'acord amb les operacions que cal que continguin, i només es reutilitzen classes quan la possibilitat de fer-ho és òbvia. Ara es tracta de revisar sistemàticament les possibilitats de reutilitzar classes ja existents d'acord amb el que sabem.

Vegeu la informació sobre la reutilització de classes al subapartat 2.1 d'aquest mòdul didàctic.

5.3. Adaptació de l'herència en el nivell suportat pel llenguatge de programació

L'herència múltiple es dóna força sovint en diagrames estàtics d'anàlisi, però hi ha llenguatges de programació que no la suporten, i per aquesta raó, quan es pretén fer la implementació amb un d'aquests llenguatges, en el diagrama estàtic de disseny cal substituir els casos d'herència múltiple per alguna altra estructura equivalent.

Justificació de l'ús de l'herència múltiple

Es podria pensar que si sabem que l'aplicació s'ha de desenvolupar en un llenguatge que no suporta l'herència múltiple, el millor és evitar-la des del començament, de la mateixa manera que es recomana que els noms respectin les restriccions més habituals en els llenguatges de programació. Tanmateix, utilitzar l'herència múltiple quan s'escau pot fer el diagrama estàtic d'anàlisi molt més entenedor.

Per a desfer l'herència múltiple caldrà utilitzar alguna de les tècniques que descrivim a continuació. En tots els casos ens referirem al mateix exemple:

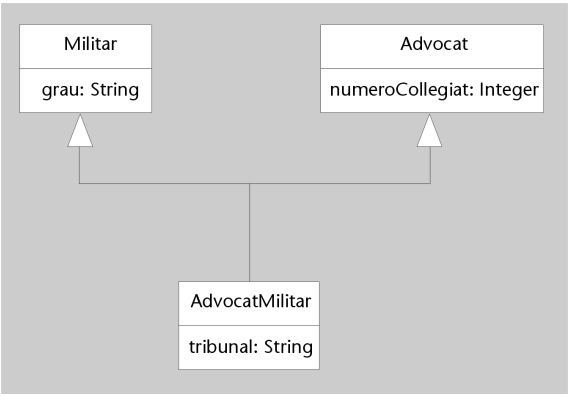

Figura 2

5.3.1. Supressió de l'herència múltiple per duplicació

Una manera de suprimir l'herència múltiple consisteix a duplicar a la subclasse els atributs que heretaria d'una de les superclasses:

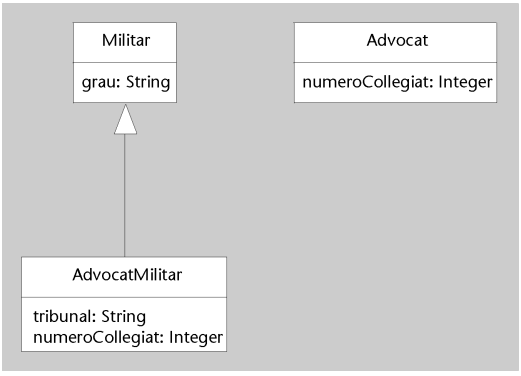

Figura 3

o bé suprimir totalment l'herència:

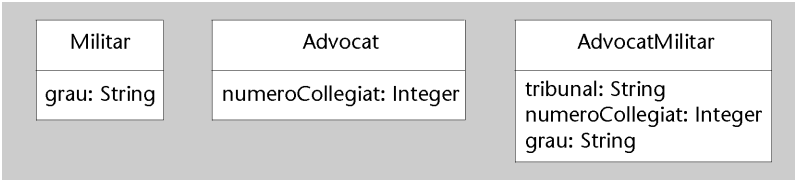

Figura 4

5.3.2. Supressió de l'herència múltiple per delegació

L'herència múltiple es pot suprimir per delegació si es manté una sola de les superclasses i s'inclou a dins de la subclasse una referència a un objecte de la que era l'altra superclasse, que té el valor dels atributs corresponents:

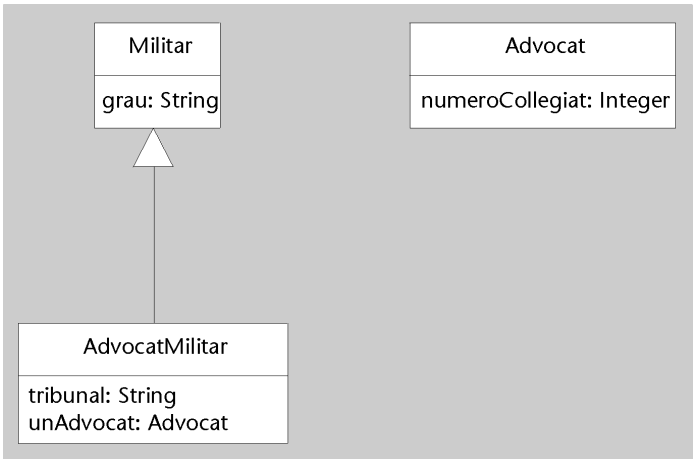

Figura 5

5.3.3. Supressió de l'herència múltiple amb interfícies

També es pot suprimir l'herència múltiple si se substitueix l'herència doble per herència simple més una interfície implementada per l'altra superclasse:

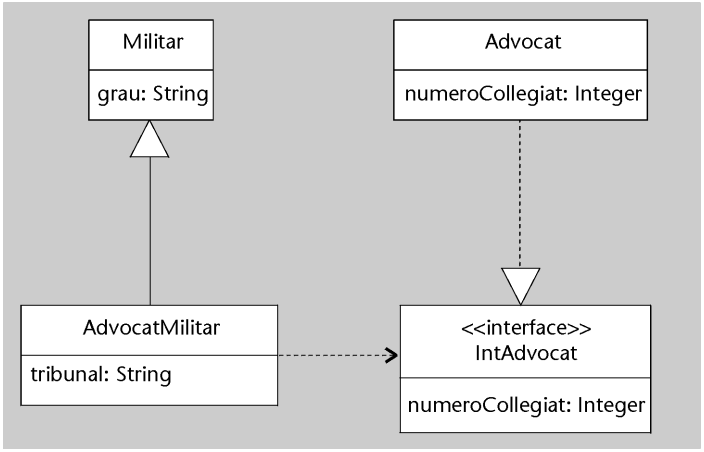

Figura 6

5.3.4. Supressió de l'herència múltiple per agregació

Les relacions d'herència es poden substituir per agregacions. Això es pot fer de dues maneres:

Vegeu la substitució de les relacions d'herència mitjançant agregacions al subapartat 4.3.5 del mòdul "Anàlisi orientada a objectes" d'aquesta assignatura.

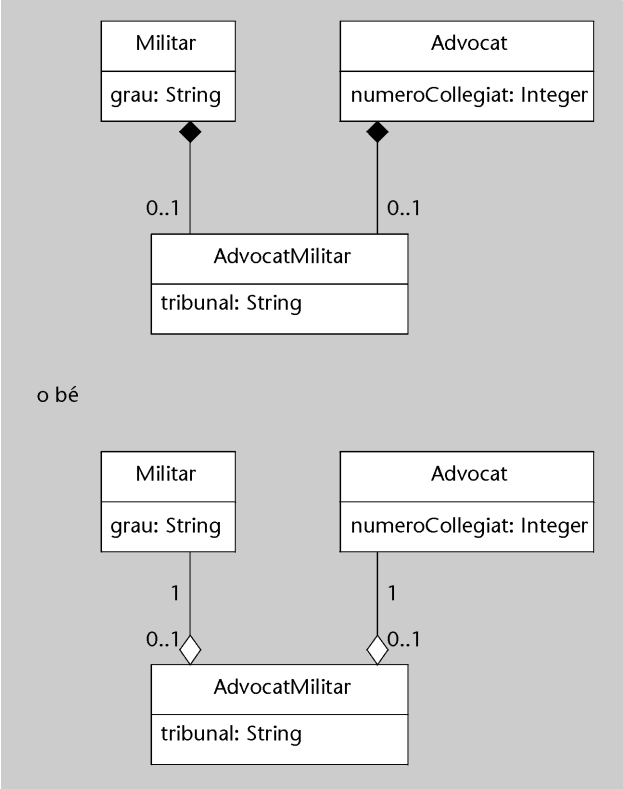

Figura 7

5.4. Substitució de les interfícies

Si el llenguatge de programació no suporta les interfícies, es poden substituir per classes abstractes, de les quals, com sabem, les interfícies en són un cas particular.

5.5. Canvis per a la millora del rendiment

A l'anàlisi normalment no ens preocupem de qüestions d'eficiència, però en el disseny és imprescindible. Es poden fer canvis al diagrama estàtic de disseny simplement per aquesta raó.

5.5.1. Agrupació de classes per a reduir el trànsit de missatges

Hem vist que, per flexibilitat, és millor substituir els atributs amb valors múltiples per una classe a part que comparteixi amb la primera una associació d'*n* a 1. Doncs bé, per raons d'eficiència potser és convenient fer el canvi invers, ja que així no cal enviar un missatge d'una classe a una altra.

5.6. Especificació de les operacions implícites

A tots els projectes o gairebé tots hi ha operacions que romanen implícites dins les especificacions de l'anàlisi. Atès que s'hauran d'implementar igual que les altres, també convé tenir-les en compte en la fase de disseny.

Heus ací algunes regles que ens poden servir d'ajuda a l'hora de trobar operacions implícites:

1) Per a tota classe hi ha d'haver una operació que la instanciï, una altra que en destrueixi els objectes i operacions que llegeixin i que posin els valors de tots els atributs (naturalment, pot ser que aquestes accions quedin compreses dins d'operacions que fan més coses). L'operació que crea un objecte agregat ha d'inicialitzar a nul la llista dels seus components.

2) Per a cada associació hi ha d'haver una operació que creï un enllaç entre objectes i una que el recorri.

3) Per a cada estat que puguin tenir els objectes d'una classe hi ha d'haver una operació que els faci arribar a aquest estat.

4) Per a cada atribut derivat hi ha d'haver una operació que en calculi el valor.

5.7. Referències a les classes de frontera

Sabem que les classes de control demanen operacions a les classes de frontera. Aquestes classes són provisionals i se substitueixen per elements gràfics (generalment també en forma de classes) durant el disseny de la interfície d'usuari. A menys que aquest s'hagués dut a terme abans del disseny dels casos d'ús, dins l'especificació de les operacions de les classes de control caldrà substituir les crides a operacions de les classes de frontera per operacions en elements de la interfície d'usuari.

5.8. La classe inicial

Generalment, es crea un objecte quan ho demana una operació que s'executa en relació amb un altre objecte ja existent, però és obvi que hi ha d'haver un primer objecte que no és creat per cap altre. Aquest mecanisme, que varia segons el llenguatge de programació, s'ha de preveure en el disseny.

5.9. Cohesió i acoblament

Arriba un moment que s'està raonablement segur que el diagrama estàtic té totes les classes que haurà de tenir per a la implementació i que totes aquestes

classes tenen definides totes les operacions. Aleshores convé revisar aquest diagrama des del punt de vista de la cohesió i l'acoblament per tal de donar la màxima flexibilitat al disseny amb vista a canvis futurs.

5.9.1. Cohesió

Una **classe molt coherent** ho és si els seus atributs i operacions tenen molta relació entre si, fins al punt que es pot considerar que formen una unitat*. En canvi, una **classe poc coherent** agrupa atributs o operacions que no tenen res o gairebé res a veure els uns amb els altres, sovint a conseqüència del fet que la classe implementa allò que en realitat són dues entitats més o menys autònomes. Una **operació poc coherent** és aquella que no té un procés amb un objectiu únic i clar.

*** Si una classe és molt coherent, la seva funció es pot definir en una sola frase, sense cap "i".**

Les classes i les operacions poc coherents són més difícils d'entendre, i a més tenen els inconvenients següents:

- Pressuposen que les diverses entitats que representen estan en relació d'1 a 1, la qual cosa no sempre és certa.
- Pot ser que s'hagin d'especialitzar segons diversos criteris.
- Es veuen afectades per més modificacions.

Ús inevitable de classes i operacions poc coherents

De vegades no hi ha més remei que agrupar diversos processos independents, que no escauen a cap classe, en una de sola. Per això hi ha les classes d'utilitat d'UML i el patró Pure Fabrication.

En general interessa evitar les classes i operacions poc coherents. La cohesió de les classes es pot aconseguir si es traslladen a aquestes les regles emprades en la normalització de bases de dades relacionals. D'altra banda, les operacions poc coherents, convé descompondre-les en diverses operacions, de la mateixa classe o de diverses, de manera que cadascuna tingui un propòsit concret.

5.9.2. Acoblament

L'acoblament de les classes i els objectes expressa el grau en què aquests depenen d'altres classes i objectes per a dur a terme la seva responsabilitat.

Com més acoblament tingui una classe, més es veurà afectada per canvis en altres i, per tant, s'haurà de modificar més sovint. Hi ha diverses formes d'acoblament:

1) Acoblament per referència: acoblament en què un objecte conté una referència a un altre. Es pot reduir si es suprimeixen associacions o si les referències es fan unidireccionals.

2) Acoblament per representació: en aquest tipus d'acoblament, un objecte fa ús d'un altre. L'acoblament per representació pot tenir diversos graus: l'accés directe

a atributs públics (el grau més baix), la petició mitjançant operacions dels valors dels atributs (privats) per a avaluar un resultat, o la crida a una operació que doni aquest resultat ja calculat (el grau més alt).

3) Acoblament per subclasses: acoblament en què una classe A té una associació amb cada una de les subclasses o els demana operacions. Per a estendre-ho a una subclasse nova caldria modificar la classe A. En canvi, l'acoblament és menor si l'associació o les crides d'operacions s'adrecen a la superclasse, ja que no cal modificar A pel fet que s'hi afegeixin subclasses. En aquest cas hi hauria problemes si l'associació o l'operació no s'haguessin d'estendre a la subclasse nova.

L'acoblament és encara més baix si les crides a operacions s'adrecen a una interfície, ja que llavors es podria fins i tot substituir la superclasse sense haver de modificar la classe A, sempre que la nova classe implementés la mateixa interfície.

4) Acoblament per herència: la subclasse no pot prescindir dels atributs i operacions que hereta de la superclasse. En canvi, això sí que és possible si se substitueix l'herència per l'agregació.

Vegeu la supressió de l'herència múltiple per agregació al subapartat 5.3.4 d'aquest mòdul didàctic.

6. Disseny de la persistència

Quan un procés acaba, allibera la memòria que utilitzava i tot el que hi havia, en principi, es perd. Si un objecte ha de tenir una vida més llarga que el procés que el crea o, dit d'una altra manera, l'objecte es crea en un procés i es fa servir en processos posteriors, cal gravar-lo en un sistema d'emmagatzemament permanent; llavors es diu que l'objecte en qüestió s'ha fet persistent i es parla d'un **objecte persistent**.

Anomenem **classes persistents** les classes que poden tenir objectes persistents. Anomenem **classes temporals** les classes no persistents.

En relació amb un objecte persistent s'han de poder dur a terme almenys dues operacions: gravar-lo i llegir-lo (l'operació *llegir* també es coneix per *materialitzar*). A la pràctica també caldrà poder-lo esborrar i identificar entre els altres objectes persistents de la mateixa classe.

D'un objecte, se'n graven els valors dels atributs (és a dir, el que s'anomena **estat de l'objecte**). Ara bé, pot ser que no tots els atributs d'un objecte persistent siguin persistents. L'**estat persistent de l'objecte** en qüestió, el constitueixen els valors dels atributs que es fan persistents. Els objectes persistents d'una classe es poden llegir de dues maneres: o bé tots alhora al començament del procés, o bé cada un quan és necessari (modalitat de **materialització segons demanda**).

Podem distingir tres tipus de sistemes d'emmagatzemament segons la manera com s'implementa la persistència: bases de dades orientades a objectes, bases de dades relacionals i fitxers clàssics, i bases de dades *object-relational*.

6.1. Persistència amb bases de dades orientades a objectes

La implementació de la persistència amb bases de dades orientades a objectes és el cas més senzill de tots, ja que no cal transformar els objectes per a fer-los persistents.

No és necessari fer un disseny de la persistència: per a especificar que els objectes d'una classe poden ser persistents, només cal enriquir la definició de la classe, indicant quins dels atributs són persistents i quina política de lectura d'objectes es vol seguir (es llegeixen tots de cop, o bé segons demanda). A partir de les definicions de les classes enriquides d'aquesta manera, es genera codi

Càlcul automàtic dels atributs no persistents

El sistema de gestió de bases de dades calcularà de manera automàtica els atributs en materialitzar els objectes.

que es passa a un preprocessador que li afegirà els mètodes necessaris per a gestionar la persistència dels objectes.

6.2. El model per a bases de dades relacionals i fitxers clàssics: alternatives

En el model per a bases de dades relacionals i fitxers clàssics, a un objecte li correspon, en principi, una fila d'una taula d'una base de dades relacional o un enregistrament d'un fitxer, respectivament* i als atributs dels objectes els corresponen columnes de la taula corresponent.

*** Ara bé, aquesta correspondència no és necessàriament d'1 a 1.**

Cal fer la transformació esmentada abans de gravar un objecte, i cal fer-ne la transformació inversa abans de poder-lo fer servir un cop llegit.

Amb aquest sistema d'emmagatzemament hi ha tres maneres de dur a terme la gestió de la persistència:

1) Fer que cada classe persistent tingui operacions perquè els objectes es gravin, esborrin, etc., per si mateixos. Aquesta opció té l'avantatge que és més eficient que les altres, atès que requereix menys crides entre objectes, però té l'inconvenient que la implementació de les classes persistents dependrà d'un sistema de gestió de bases de dades concret pel fet que són classes d'entitats que corresponen a entitats del domini del programari i això limita la portabilitat de l'aplicació.

2) El segon mètode defineix una classe anomenada *gestor de disc* per a cada classe persistent. El **gestor de disc** accedeix directament al fitxer o base de dades. Amb aquest mètode, la classe d'entitats es desacobla del sistema de gestió de base de dades, i té l'avantatge addicional que el gestor de disc pot fer de memòria cau* dels objectes de la classe d'entitats (si els objectes llegits es materialitzen dins d'instàncies del gestor abans de crear instàncies de la classe d'entitats). Ara bé, aquesta solució és menys eficient que l'anterior.

*** També anomenada, *memòria cache*.**

3) El tercer mètode és una barreja dels anteriors i consisteix a crear els gestors de disc i a més afegir operacions de gravació, lectura, etc., a les classes del domini. La diferència amb el primer mètode és que les operacions de la classe no implementen la persistència directament, sinó que criden operacions del gestor de disc.

Per tal d'aïllar les classes d'entitats del sistema de gestió de bases de dades és molt útil implementar la persistència per mitjà d'un bastiment.

Vegeu els bastiments d'aplicacions al subapartat 2.4 d'aquest mòdul didàctic.

6.2.1. Obtenció de la definició de l'estructura de base de dades relacional o de fitxers clàssics

La base de partida per a obtenir la definició de l'estructura de bases de dades relacional o de fitxers clàssics és la part del diagrama estàtic de disseny que conté aquelles classes d'entitats que són classes persistents i les relacions entre aquestes.

Els passos per a arribar a obtenir la definició d'aquesta estructura seran els següents:

Abreujarem model entitat-relació amb l'expressió model ER.

1) Transformar el model estàtic en un model entitat-relació (model ER).

2) Suprimir l'herència, atès que el model relacional no la suporta. Si el model ER utilitzat tampoc la suporta, aquest pas s'haurà hagut de fer abans de l'anterior.

3) Transformar el model ER en un model relacional segons les regles habituals del disseny de bases de dades relacionals. En el cas de fitxers clàssics, en comptes de taules relacionals tindrem fitxers plans.

4) Crear un gestor de disc per cada classe persistent que implementi els accessos a les taules o fitxers plans corresponents.

6.2.2. Transformació del model estàtic al model ER

Si ens hi fixem, un model OO és molt semblant a un model ER, sobretot en el cas d'aquelles variants del model ER que suporten l'herència de tipus d'entitats.

En aquest cas, la transformació del model estàtic al model ER consisteix a convertir cada classe amb un atribut amb valors múltiples en dues classes unides per una associació, i després fer correspondre un tipus d'entitat a cada classe no associativa. Més tard s'afegeix una relació per a cada associació o classe associativa o agregació, amb cardinalitats que són òbvies.

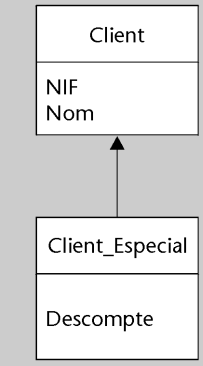

Figura 8

6.2.3. Supressió de l'herència

Suposarem el cas presentat a la figura 8:

Hi ha tres maneres de resoldre la situació plantejada:

1) Definir una taula per a cada classe, que contindria tant els atributs propis de la classe com els heretats.

2) Crear una taula per a la superclasse i una de complementària per cada subclasse d'aquesta que tingui atributs propis. Cada taula complementària contindria els atributs propis de la subclasse respectiva més els atributs que identifiquen cada fila (cada objecte) dins la taula de la superclasse per a poder ajuntar tots els atributs (heretats i no heretats) de cada objecte de la subclasse.

Nota terminològica

Quan la supressió de l'herència es duu a terme en passar del model estàtic al model ER, el terme *taules* fa referència a entitats i relacions; en passar del model ER al relacional, el concepte *classes* es refereix a entitats i relacions.

3) Crear una sola taula per a tota la jerarquia de classes, que tindria els atributs propis de cada subclasse més els de la superclasse. En cada fila, els atributs que no fossin aplicables a la subclasse a què pertany l'objecte haurien de tenir el valor nul.

Supressió de l'herència per definició d'una taula per cada subclasse

En definir una taula per cada classe, les files de *Client_Especial* tindran tant els atributs específics de *Client_Especial* com els heretats.

Taula Client:
NIF: String, clau principal
Nom: String

...

Taula Client_Especial:
NIF: String, clau principal
Nom: String
Descompte: Integer

...

Evidentment, aquesta solució és molt senzilla però té els inconvenients següents:

- Hi ha tantes taules com subclasses.
- Cal crear un gestor de disc per cada subclasse.
- Un canvi en la definició de *Client* obligaria a canviar els gestor de disc de totes les subclasses.
- Complicarà aquells processos que hagin de tractar tots els clients, ja que hi caldrà fer una fusió* de totes les taules.

*** En anglès, *merge*.**

Supressió de l'herència per creació d'una taula per a la superclasse i una complementària per subclasse

Per a suprimir l'herència també es pot crear una taula per a la superclasse en la qual es graven tots els clients. Alhora, crearem una taula complementària

per cada subclasse amb l'identificador de l'objecte i els atributs específics de la subclasse. Un gestor de disc únic s'encarregarà de llegir i farà un *join* per identificador entre les diferents taules a fi d'obtenir tota la informació de cada classe.

Segons la informació que llegeixi, el gestor crearà una instància d'una subclasse o altra i l'emplenarà amb les seves dades.

Taula Client:

NIF: String, clau principal

Nom: String

...

Taula Auxiliar Client_Especial:

NIF: String, clau principal

Descompte: Integer

...

Aquesta solució pot ser la millor opció si només una fracció petita dels objectes de la classe pertanyen a la subclasse i, a més, els valors dels atributs d'aquesta són de longitud fixa, ja que aleshores els valors nuls ocuparien molt espai, si hi fossin per a tots els objectes de la superclasse.

Supressió de l'herència per creació d'una taula única per a tota la jerarquia d'herència

Aquesta tercera solució és més eficient en temps, però ho és menys en ocupació de disc.

La supressió de l'herència per creació d'una taula única per a tota la jerarquia d'herència consisteix a crear una sola taula amb tots els camps, tant de la superclasse com de totes les seves subclasses, i afegir-hi un camp que ens digui quina subclasse de l'objecte correspon a cada fila.

Taula Client:

NIF: String, clau principal

Tipus: Integer

Nom: String

...

Descompte: Integer

...

on *Tipus* es defineix de la manera següent:

$$Tipus = \begin{cases} 0 \text{ si és } Client \\ 1 \text{ si és } Client\_Especial, \text{ etc.} \end{cases}$$

Aquesta solució pot ser bona quan l'espai ocupat pels valors nuls dels atributs no aplicables a totes les subclasses sigui reduït: és a dir, si hi ha poques subclasses, aquestes tenen pocs atributs propis o bé quasi tots els atributs són aplicables a la majoria d'objectes.

6.2.4. Els gestors de disc

Considerem ara el mètode alternatiu del gestor de disc per a la supressió de l'herència.

> El gestor de disc és una classe diferent de la classe persistent, i dins les operacions d'aquesta hi ha crides a operacions del gestor de disc.

Considerarem per separat el gestor de disc bàsic i els gestors de disc per a la materialització segons demanda.

Disseny bàsic de gestors de disc

Podem considerar que tots els gestors de disc tenen almenys unes operacions bàsiques (llegir, gravar, regravar i esborrar). Per tant, podem entendre que tots són subclasses d'una superclasse que anomenarem *GestorGeneric*, en la qual aquestes operacions serien abstractes. Fins i tot podríem considerar que hi ha unes subclasses intermèdies *GestorRelacional* i *GestorFitxers*, que serien superclasse de tots els gestors relacionals i de tots els gestors de fitxers clàssics, respectivament.

En el disseny bàsic de gestors de disc pot aplicar-se el patró *Template Method*, en el qual hi ha una superclasse abstracta que té una operació concreta que crida operacions abstractes de si mateixa que esdevenen concretes a les subclasses.

El diagrama de classes corresponent a aquest disseny és com el que veieu a continuació (el paràmetre *id* és de tipus *string* arreu).

La classe abstracta *GestorGeneric* té l'operació *llegirId* (entre altres operacions públiques per a esborrar, etc.) que llegeix un objecte a partir d'un identificador d'aquest. En primer lloc, aquesta operació comprova si l'objecte ja està materialitzat i llavors l'agafa de memòria (operació *enCache*). Altra-

ment, l'ha de llegir de la base de dades o fitxer amb l'operació abstracta materialitzar. Aquesta operació esdevé concreta a les subclasses (també abstractes) *GestorRelacional* i *GestorFitxers*, i crida l'operació *llegirFila* o la *llegirEnregistrament*, respectivament, i després, *filaAObjecte* o *enregistramentAObjecte*, també respectivament, que fan els moviments de dades entre la fila o enregistrament llegit i l'objecte creat (d'una classe concreta). Per tant, aquestes operacions depenen de quina sigui la classe persistent i, en conseqüència, són abstractes.

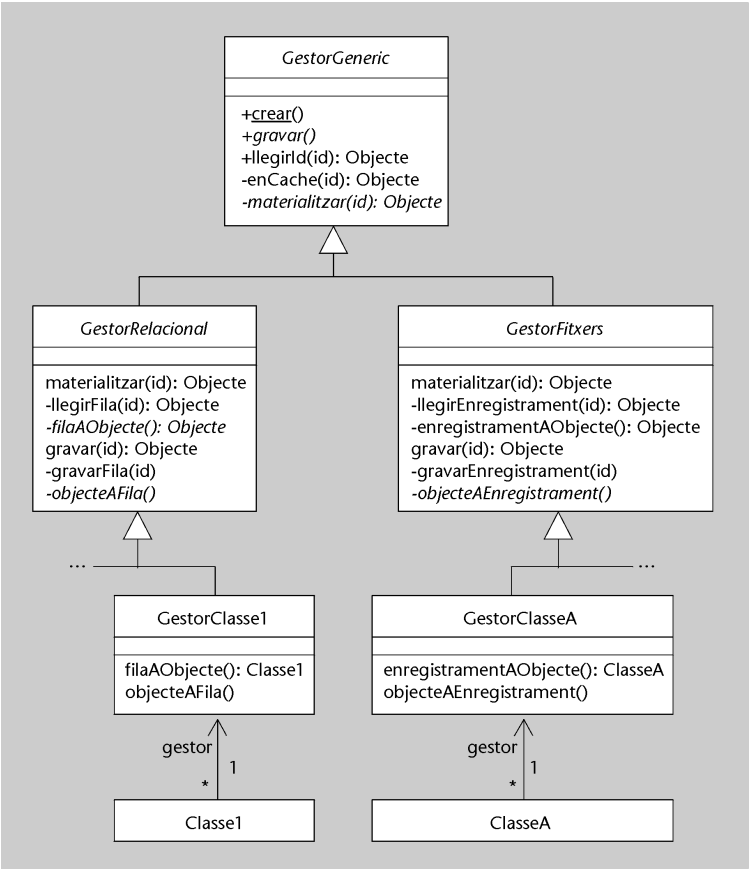

Figura 9

Les operacions *llegirFila* i *llegirEnregistrament* també depenen de la taula o fitxer que s'ha de llegir, que podria ser un paràmetre addicional de l'operació (per exemple, si es fa servir una tècnica d'accés com ara SQL dinàmic) o bé hauria de ser substituïda en el nivell de subclasse per mitjà de polimorfisme. Això darrer s'hauria de fer necessàriament si la materialització requerís d'accedir a més d'una taula o fitxer.

El procés per a afegir o modificar una fila és simètric del de materialitzar: a *GestorGeneric* i a *GestorRelacional* hi ha una operació *gravar* abstracta i *GestorFitxers* crida dues operacions: *objecteAFila* i *gravarFila*, i *objecteAEnregistrament* i *gravarEnregistrament*, respectivament. Si la fila o enregistrament existeix, se'n substitueixen els valors, i si no, s'afegeix a la taula. També caldria preveure una operació per a esborrar files o enregistraments un a un.

Per a implementar la materialització segons demanda es pot utilitzar una combinació del patró *Template Method* amb la modalitat anomenada *Virtual* del patró *Proxy* més els patrons *Factory Method* i *Singleton*.

Disseny de gestors de disc per a la materialització segons demanada

Aquest disseny és evidentment més complex i només està justificat o bé quan s'ha de poder substituir la classe d'entitat sense haver de modificar totes les que la fan servir, o bé quan la materialització és costosa i cal esperar a fer-la quan sigui absolutament necessari; però mentre no es faci, cal que les referències exteriors a l'objecte apuntin a un objecte existent.

Les referències exteriors no es fan a objectes de la classe d'entitats en qüestió, sinó a objectes d'una classe substituta. La classe original i la substituta implementen la mateixa interfície (o classe abstracta, si el llenguatge de programació no suporta explícitament les interfícies), i la classe substituta crida les operacions de l'original. La classe substituta pot ser subclasse d'una classe abstracta.

Abans de poder demanar operacions al gestor de disc, cal crear-ne una instància. Òbviament, la instància que s'ha de crear ha de ser del gestor de disc corresponent a la classe d'entitats en qüestió i a la seva substituta. El patró *Factory Method* defineix una classe creadora abstracta (que pot ser la classe substituta abstracta) i crea una instància de la classe adequada amb una operació abstracta de creació. Aquesta operació és substituïda per operacions concretes en el nivell de subclasses que creen instàncies del gestor corresponent.

Lectura recomanada

Trobareu ampliacions sobre el disseny de gestors de disc per a la materialització segons demanda que consideren altres aspectes de la persistència (per exemple, transaccions) amb més patrons a l'obra següent:

C. Larman (1998). *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*. Prentice Hall.

L'aplicació del patró *Singleton* garanteix que només es creï una instància de cada classe gestor, i ho aconsegueix fent que la instància creada sigui el valor d'un atribut de classe i l'operació de creació sigui una operació de classe, mentre que l'estat de l'objecte es compon d'atributs d'instància i és accessible per mitjà d'operacions d'instància.

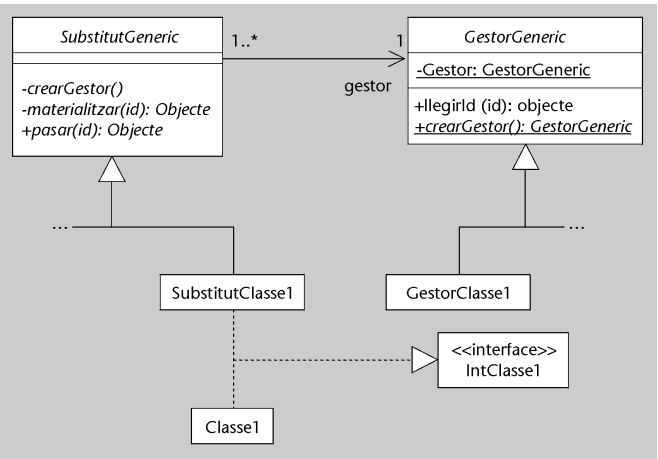

Figura 10

6.3. Persistència amb bases de dades *object-relational*

En principi, el mètode disseny de la persistència amb bases de dades *object-relational* és el mateix que el de les bases de dades relacionals, amb la diferència que aquest nou model pot tenir atributs de tipus compost i, per tant, no caldrà crear gestors de disc per a totes les classes persistents, sinó només per a aquelles a les quals s'accedirà directament.

Exemple de disseny de persistència amb bases de dades *object-relational*

En l'exemple d'una classe *Persona* amb fills, si la base de dades que utilitzem permet tenir com a camp un *array* de fills, modelarem aquesta relació com un atribut de persona i evitarem haver de definir un gestor de disc de fills. Diem "si permet" perquè de moment no existeix cap estàndard i cada fabricant de sistemes de gestió de bases de dades ofereix un model propi. Per tant, caldrà anar amb molta cura en triar una base de dades d'aquest tipus per tal que satisfaci les necessitats de la nostra empresa.

7. Disseny de la interfície gràfica d'usuari

A les etapes de recollida i documentació de requisits hi havia un cert paral·lelisme entre les activitats relatives a la funcionalitat i les relatives a la interfície d'usuari: a la recollida i documentació de requisits hi havia casos d'ús per una banda i tasques per una altra; a l'etapa d'anàlisi, hi havia col·laboració entre classes de frontera, de control i d'entitats, per una banda, i esquema de l'aspecte visual de les classes de frontera per una altra.

La raó d'aquesta dualitat és clara: quan es tracta de la funcionalitat es considera el funcionament del programari des del punt de vista intern, mentre que quan es tracta de la interfície, el que compta són els efectes del funcionament del programari visibles per l'usuari.

En l'etapa de disseny aquesta dualitat es manté per dos motius:

- D'una banda, per la raó ja esmentada: hi ha un disseny de la interfície d'usuari separat del disseny dels casos d'ús.

- D'altra banda, com veurem tot seguit, perquè en el disseny de la interfície d'usuari s'hi fa servir una metodologia diferent en molts aspectes.

També pot ser que al començament de la implementació es treballi per separat en la funcionalitat i en la interfície d'usuari, però en algun moment s'hauran d'ajuntar les classes de la interfície gràfica amb la resta, atès que hauran de col·laborar.

La coherència entre la funcionalitat i la interfície d'usuari hauria d'estar garantida pel fet que les bases de partida dels desenvolupaments respectius (la documentació dels casos d'ús i la de les tasques futures, totes dues obtingudes a l'etapa de recollida i documentació de requisits) descriuen el mateix des de dos punts de vista; però és clar que convé comprovar que aquesta concordança es mantingui entre la documentació respectiva corresponent a les etapes d'anàlisi i de disseny.

El **disseny de la interfície d'usuari** considera tres aspectes d'aquesta: 

a) El **contingut**, que ja està establert en els esquemes de les finestres elaborats a l'anàlisi.

b) El **format**, que s'especifica íntegrament en aquesta fase.

c) La **interacció**, és a dir, la dinàmica de la interfície d'usuari, anomenada també *diàleg entre l'usuari i el sistema*. La interacció està establerta dins l'especificació

dels casos d'ús en termes formals (classes de frontera) però independents de l'eina que es farà servir i, per tant, en el disseny caldrà descriure'n la implementació amb els elements que ofereix l'eina de suport de la interfície gràfica.

Abans d'entrar en el disseny de la interfície gràfica d'usuari pròpiament dit veurem breument els aspectes estàtics i dinàmics de la interfície gràfica d'usuari, que són els elements amb què se n'elabora el disseny.

La **interfície gràfica d'usuari** és només una part de la interfície d'usuari en general: és únicament la interfície implementada per mitjà de pantalles gràfiques amb teclat i ratolí.

La interfície gràfica d'usuari no inclou, per tant, les sortides impreses, en especial les alfanumèriques (els clàssics *llistats*). 

Interfícies alfanumèriques

Les interfícies alfanumèriques per pantalla són una relíquia de l'època en què la unitat adreçable dins les pantalles no era el *pixel*, sinó el caràcter, i es poden considerar un cas particular, molt simple, de les interfícies gràfiques.

7.1. Elements i funcionament de la interfície gràfica d'usuari

Començarem per descriure breument els principals tipus d'elements que componen les interfícies gràfiques i després parlarem de la interacció entre l'usuari i el sistema.

7.1.1. Elements de la interfície gràfica d'usuari

Els elements de les interfícies gràfiques s'acostumen a anomenar **objectes** o **objectes gràfics**.

Exemples d'objectes gràfics

Un objecte de la interfície pot ser tant una finestra, com un botó, com un text, com un mot o un caràcter dins un text.

Els objectes gràfics no són necessàriament instàncies de classes, encara que sí que ho són quan la interfície s'implementa amb una eina orientada a objectes. És normal que un objecte de la interfície estigui format per altres objectes.

Els objectes gràfics es poden basar en *pixels** o en uns objectes que s'anomenen **primitives gràfiques****. Tanmateix, per a visualitzar i imprimir les primitives gràfiques generalment cal convertir-les a mapes de bits (*rasterització*).

*** Com ara els mapes de bits.**
**** Per exemple, punts, línies, segments, cercles, triangles i quadrats.**

Una interfície gràfica té els següents tipus de components: escriptori, cursors i punters, finestres, menús, controls, elements complexos i caixes de diàleg.

L'escriptori

L'escriptori és el fons de pantalla que veu l'usuari quan s'engega el sistema i abans d'engegar cap aplicació; al damunt d'aquest apareixen els objectes de la interfície gràfica.

Els dispositius d'entrada: ratolí, teclat, ploma i pantalla tàctil

Una interfície gràfica fa servir dos dispositius d'entrada típics: el teclat i el ratolí.

El **punter** segueix el moviment del ratolí i canvia de forma segons els usos: fletxes d'una, dues o quatre puntes, creu, rellotge, etc. Tant el punter com els objectes que pot seleccionar tenen una zona al voltant dins la qual els clics tenen efecte.

El ratolí pot tenir d'un a tres botons i es pot fer servir de quatre maneres bàsiques:

- arrossegar el punter sense prémer cap botó.
- pitjar el botó sobre una icona i deixar-lo anar immediatament (clicar).
- pitjar el botó i arrossegar la icona abans de deixar-lo anar.
- pitjar el botó i deixar-lo anar dues vegades seguides o més sobre una icona (doble clic, etc.).

Del teclat ens interessen especialment les combinacions de tecles que equivalen a la selecció d'opcions dins un menú. El **cursor** assenyala el punt, dins un camp o un text, on té efecte allò que es faci amb el teclat. Es diu que l'objecte on hi ha el cursor "té el **focus**". La navegació del cursor es pot dur a terme o bé fent clic amb el ratolí en la posició desitjada, o bé amb les tecles de principi de línia (*Home*), final de línia (*End*), retrocés de pàgina (*Page Up*), avanç de pàgina (*Page Down*), les quatre fletxes i el tabulador (*Tab*).

La **ploma** pot fer les funcions del ratolí, però a més pot escriure. Aleshores, segons la forma i direcció del traç s'executen diferents comandaments: seleccionar, copiar/enganxar, anul·lar, tabular, esborrar un caràcter, tecla de retorn de carro (*Return*), espai, etc.

Funcions equivalents de la ploma i el ratolí

Picar en un punt amb la ploma equival a fer-hi clic amb el ratolí; picar-hi dues vegades equival a fer-hi doble clic; d'altra banda, amb la ploma es pot arrossegar com amb el ratolí.

Les **pantalles tàctils** són útils quan el volum de dades que s'ha d'introduir és molt petit i l'entrada consisteix essencialment en seleccions.

Les finestres

Una finestra ve a ser una pantalla virtual associada a una aplicació activa, i és un marc dins del qual apareixen els objectes propis de l'aplicació.

Si en un moment donat hi ha diverses aplicacions actives en un ordinador amb una sola pantalla física, i per tant amb un sol escriptori, l'han de compartir, i la manera de fer-ho és que cada aplicació tingui la seva finestra (de fet una aplicació pot tenir diverses finestres, com veurem).

La finestra de l'aplicació amb què treballa l'usuari en cada moment rep l'entrada del ratolí i el teclat (es diu que "té el **focus**") i sovint tapa totalment o parcialment les altres, mentre que les finestres de les altres aplicacions actives romanen en segon pla, però a punt per a tornar a aparèixer quan l'usuari torni a treballar amb l'aplicació respectiva.

Hi ha dues maneres de gestionar el focus de les aplicacions:

1) La **gestió implícita del focus**, en què la finestra que té el focus és sempre aquella damunt la qual hi ha el punter del ratolí.

2) La **gestió explícita del focus**, en què l'assignació del focus a una finestra es fa explícitament i independentment de la posició del cursor.

En el cas més general, les finestres tenen un marc i poden portar un títol i contenir elements gràfics actius (controls) per a les operacions següents:

- Variar-ne la grandària.
- Maximitzar-la, que vol dir fer que ocupi tota la pantalla.
- Minimitzar-la, convertint-la en una icona; fent-li doble clic recupera les dimensions que tenia abans de la minimització.
- Restaurar-la, és a dir, restituir-li les dimensiones que tenia abans de maximitzar-la.
- Dividir-la en dues, verticalment o horitzontalment, arrossegant la línia de divisió.
- Desplaçar-ne el contingut amunt i avall i a dreta i esquerra (*scroll*) mitjançant les barres que hi ha a la dreta i a sota respectivament.
- Arrossegar tota la finestra, normalment situant el cursor a la barra de títol.
- Amagar-la (és a dir, fer-la transparent, de manera que deixi veure la finestra de sota o bé l'escriptori).
- Tancar-la, acabant o no l'aplicació respectiva.

Podem distingir diferents tipus de finestres:

1) Finestres d'aplicació: són les finestres que apareixen quan un usuari engega una aplicació. Tenen un títol amb el nom de l'aplicació, controls per a redimensionar-les, el menú de barra de l'aplicació, botons per a maximitzar-les, minimitzar-les i tancar-les, una àrea de missatges i una àrea d'estat, i són movibles.

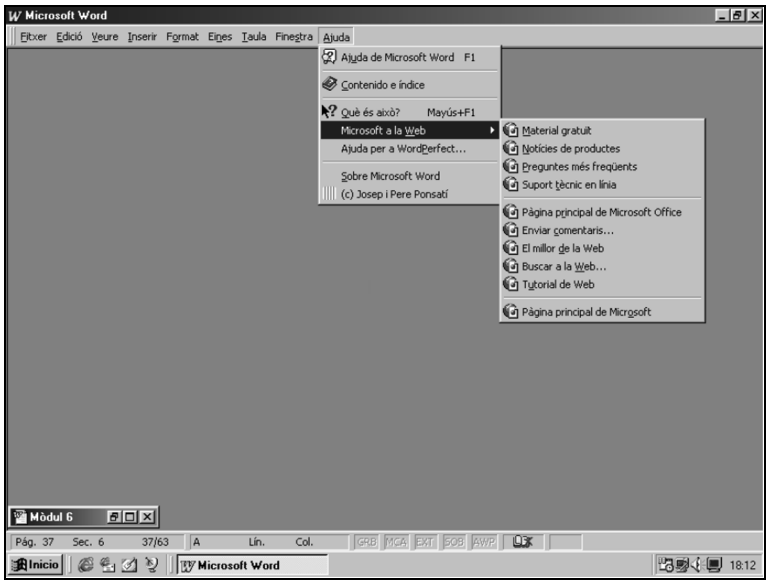

Figura 11

Finestra d'aplicació i menús

A la figura podeu veure una finestra d'aplicació i una cascada de menús: el menú de barra i dos nivells de menús desplegables.

2) Finestres de document: són finestres associades a una aplicació i generalment dintre de la finestra d'aquesta (vegeu la figura següent). Se'n crea una cada vegada que s'obre un document o es crea un document nou. Hi pot haver diverses finestres de document obertes alhora, però només una en té el focus, fins i tot si n'hi ha diverses de visibles al mateix temps, és a dir, l'usuari en cada moment només pot treballar amb un document. Hi ha d'haver alguna manera de passar d'un document a un altre: generalment se selecciona d'una llista o bé se'n fa clic en una porció visible.

Les finestres de document tenen un títol amb el nom del document, controls de finestra amb les mateixes opcions que la finestra de l'aplicació i també controls per al desplaçament* horitzontal i vertical i per a dividir la finestra.

*** En anglès, *scroll*.**

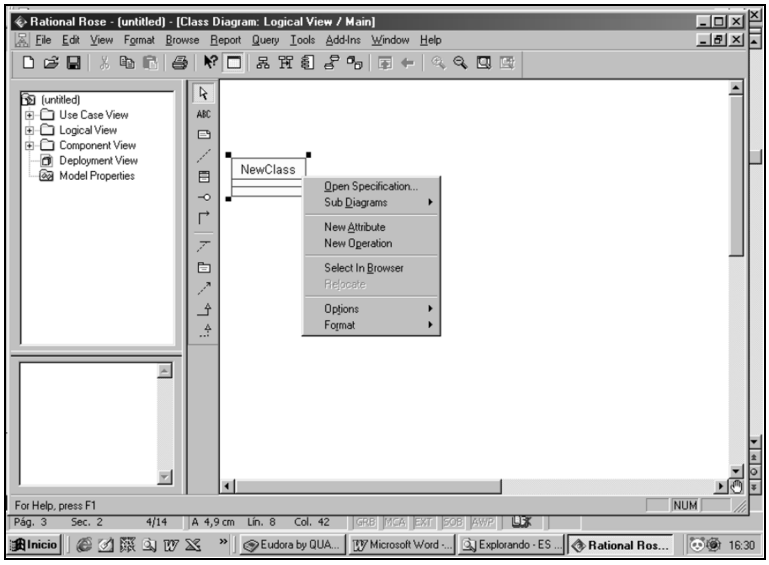

Figura 12

Finestra de document i menú emergent

Aquesta figura és un exemple de finestra de document (en aquest cas, un diagrama) en la qual s'hi veu també un menú emergent.

3) Finestres de caixes de diàleg o finestres secundàries: tenen una barra de títol amb el nom del comandament que les ha obert i contenen almenys una

caixa de diàleg. No tenen àrea per a documents, ni barres de menú i d'estats, ni controls de redimensionament, ni àrea de missatges; es poden moure i redimensionar estirant perquè hi hagi més opcions visibles. Generalment són **finestres modals**, és a dir, mentre són obertes, l'usuari no pot treballar amb cap altra finestra de la mateixa aplicació o fins i tot, segons en quin sistema operatiu, de tot el sistema.

Elements estàtics

Els elements estàtics són elements fixos i passius continguts dins una finestra o un control complex: títols i altres etiquetes, línies, logotips, dibuixos que no siguin icones, etc.

Controls simples

Els controls són els components actius de la interfície per mitjà dels quals l'usuari interactua amb el programari. N'hi ha de simples i de compostos.

Els principals tipus de controls simples són els que enumerem a continuació:

a) Botons de comandament: acostumen a ser quadrats o rectangulars. Sovint van agrupats per funcions relacionades. Per a identificar-ne la funció, porten una icona, un text o totes dues coses. Quan se seleccionen aparenten estar enfonsats o presenten algun altre canvi d'aspecte.

Vegeu exemples de botons de comandes a la figura 13, just a sota del menú de barra i, també a la part de baix de la finestra.

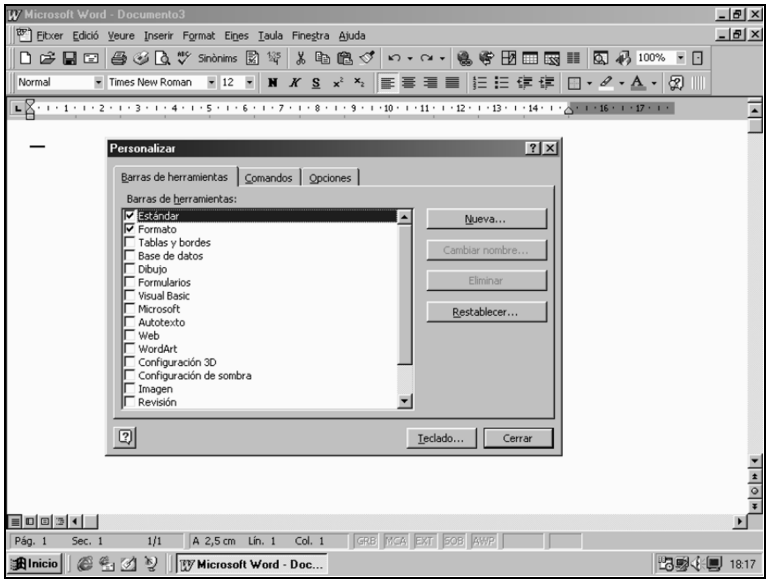

Figura 13

Exemples de botons de comandament

A la figura podeu veure un exemple de finestra modal amb una caixa de diàleg, carpeta amb separadors i *Check boxes*.

b) Icones: són petits dibuixos. Quan no van associats a botons s'hi pot fer clic, doble clic i arrossegar / deixar anar*.

* En anglès, *drag/drop*.

c) *Check boxes*: són caselles quadrades que corresponen a una variable booleana; quan aquesta pren el valor "vertader", la casella apareix marcada.

d) Botons de ràdio: representen variables booleanes de les quals només una pot ser certa en cada moment, o bé valors possibles i incompatibles d'una variable. Per tant, en un grup de botons de ràdio, n'hi ha sempre un de seleccionat, i només un. En canvi, les *check boxes* són independents.

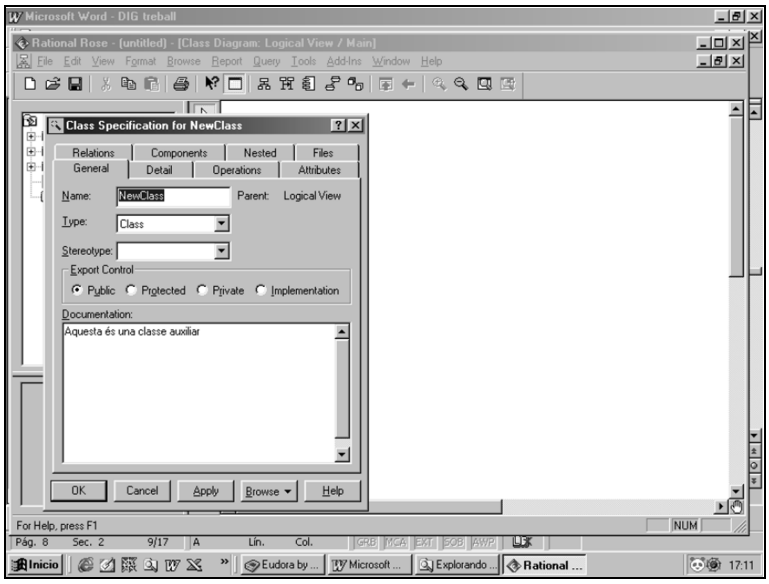

Figura 14

Caixa de diàleg i text amb *scroll*

A la figura podeu observar una caixa de diàleg amb carpeta amb separadors, un conjunt de botons de ràdio ("Export Control") i un text de possibles línies múltiples amb barra de desplaçament vertical.

e) ***Value sets***: són com botons de ràdio, però amb icones en lloc d'etiquetes*.

*** Per exemple, les paletes de colors.**

f) Camps de text: un text és un camp alfanúmeric de llargada virtualment il·limitada. Segons l'aplicació, un text pot ser un document o només el contingut d'un control; en el darrer cas, el text pot ser d'una línia o de línies múltiples amb eventual barra de desplaçament vertical i horitzontal.

g) ***Spin boxes***: es pot donar un valor o augmentar-lo o disminuir-lo per salts clicant uns botons amb puntes de fletxa.

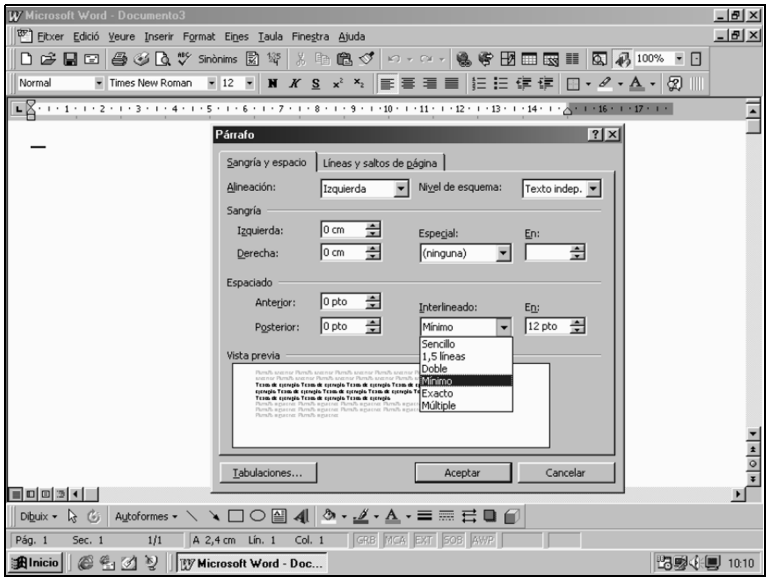

Figura 15

Llistes desplegables i *spin boxes*

A la figura es poden veure exemples de llistes desplegables (una està desplegada) i algunes *spin boxes* ("Sangría izquierda", "Sangría derecha" i d'altres).

Controls complexos

Els controls complexos són aquells que estan formats per altres controls.

Alguns tipus de controls d'aquesta mena són els que esmentem a continuació:

a) Llistes i llistes *combo*: els seus elements poden ser etiquetes o, més rarament, icones. A les llistes *combo* es pot seleccionar un element existent o bé teclejar-lo; en canvi, a les llistes ordinàries només es pot seleccionar un element existent. Les llistes i llistes *combo* poden ser permanents (amb barres de desplaçament o no) o desplegables (en aquest cas tenen un control per a desplegar-les, normalment un botó amb una punta de fletxa cap avall).

b) Taules o **matrius**: així com les llistes són unidimensionals, les taules o matrius són bidimensionals i s'organitzen en cel·les.

c) Caixes de diàleg: en aquest context, un diàleg és un intercanvi de símbols i accions entre el sistema i l'usuari. Les caixes de diàleg presenten opcions per a escollir en forma de controls simples de tota mena i també de llistes, i tenen almenys un botó per a confirmar i sovint botons d'ajuda i de cancel·lació. Generalment van totes soles dins una finestra modal. Usos típics de les caixes de diàleg són la tria d'opcions d'impressió, l'obertura de fitxers i la presentació de missatges que exigeixen resposta, o almenys confirmació, que s'han vist.

Vegeu alguns exemples de caixes de diàleg a les figures 13, 14 i 16.

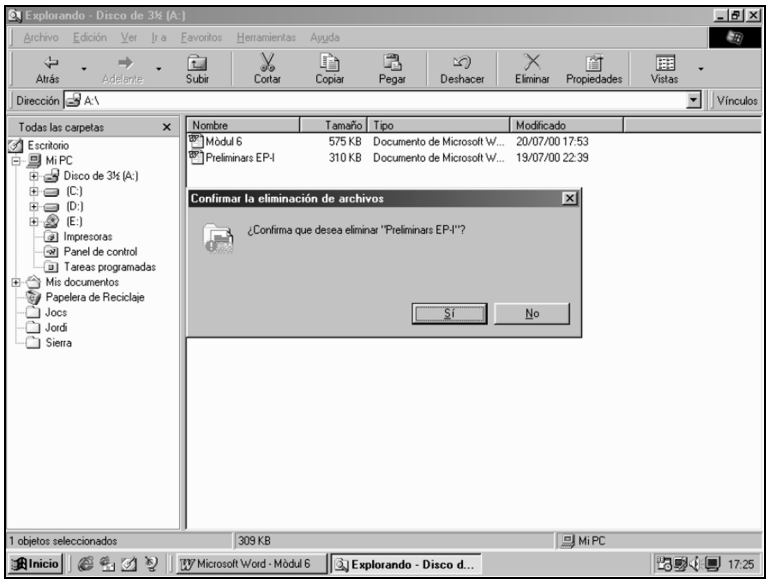

Figura 16

Caixes de diàleg

A la figura podeu veure una caixa de diàleg modal de missatge.

d) Formularis: són plantilles per a la introducció de dades. Es poden considerar caixes de diàleg en les quals predominen els camps de text damunt els controls per a la selecció d'opcions.

e) Carpetes: carpetes de fitxers, i també carpetes amb separadors a cadascun dels quals correspon una caixa de diàleg.

Vegeu un exemple de carpetes amb una fila de separadors a la figura 13 i amb diverses files a la figura 14.

f) Barres d'eines: són grups de botons de comandament, que de vegades l'usuari pot personalitzar, de manera que s'hi pot posar només els botons desitjats. Les barres d'eines es poden ocultar o fer canviar de forma, de més allargada a més quadrada, i viceversa. De vegades són flotants i tenen finestra pròpia, que roman en primer terme encara que no tingui el focus.

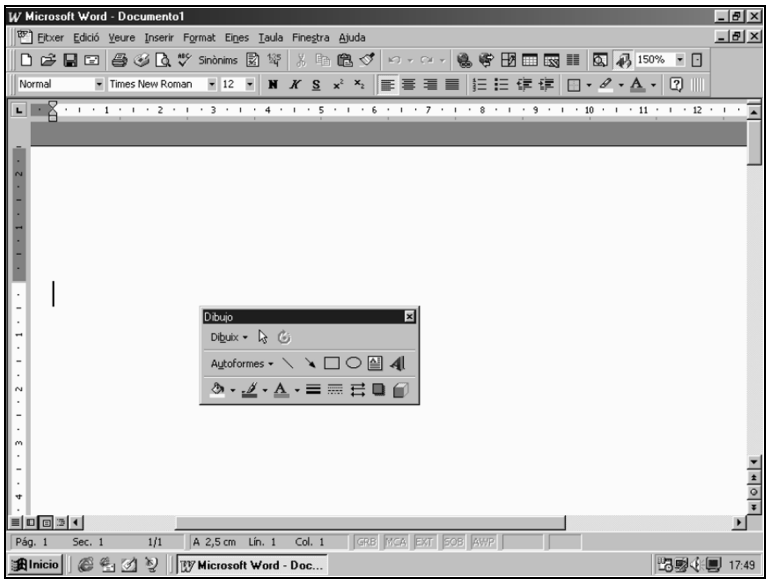

Figura 17

Els menús

Un **menú** és una llista d'etiquetes (que anomenem *opcions del menú*) dins la qual l'usuari en pot seleccionar una cada vegada clicant-la amb el ratolí.

Exemples d'opcions de menú

Les opcions del menú poden representar funcions d'una aplicació, fitxers d'una carpeta, ordinadors d'una xarxa, valors d'una variable, etc.

Les **opcions del menú** poden ser paraules o frases curtes o bé icones i poden tenir significats molt variats. Pot haver-hi opcions que en un moment donat no siguin seleccionables, i llavors apareixen en gris en comptes de negre.

Classificació dels menús

a) Menús permanents: són aquells menús que estan associats a una finestra d'aplicació i són visibles mentre la finestra roman activa. Els més habituals són els **menús de barra**.

Vegeu un exemple de menú de barra a la figura 11.

b) Menús temporals: aquests menús esdevenen visibles només quan l'usuari els obre, i desapareixen quan l'usuari en selecciona un element, o bé quan deixa anar el botó del ratolí (en alguns casos), o clica un altre objecte.

Un menú s'obre...

... per exemple, seleccionant un element del menú de barra o d'un altre menú temporal, o bé clicant una icona.

A més de les opcions, hi pot haver línies de separació de grups d'opcions. Cada opció pot anar acompanyada d'un **accelerador** (combinació de tecles equivalent a l'opció), un **mnemònic** (una lletra equivalent a l'opció) i una marca que indica que s'ha seleccionat l'opció.

Els anomenats **menús desplegables** o **menús de persiana*** es despleguen cap avall. En els s**istemes de menús en cascada** quan se selecciona un element d'un menú desplegable es desplega un altre menú al costat. Un element d'un menú al qual li correspon un nou menú acostuma a anar acompanyat d'un triangle negre amb la punxa cap al costat on hi ha aquest menú.

*** En anglès, *pull-down, drop-down menus*.**

Vegeu un exemple de menús en cascada a la figura 11.

c) Menús emergents*: aquests menús no formen part d'una cascada, sinó que sorgeixen (sovint cap amunt o bé cap a la direcció en què hi ha més espai disponible) quan l'usuari clica el ratolí dins una àrea determinada d'una finestra o de l'escriptori. Entre aquests tipus de menús hi ha els associats a un objecte gràfic, que sovint es fan desplegar amb el botó dret.

*** En anglès, *pop-up menus*.**

Vegeu un exemple de menú emergent a la figura 12.

7.1.2. La interacció

En aquest subapartat passem a esmentar els diferents tipus d'interacció que podem trobar i la manera com es resolen.

El model objecte-acció

Per a descriure la interacció entre l'usuari i l'ordinador mitjançant una interfície gràfica farem servir el model objecte-acció.

El model objecte-acció considera que una interacció elemental té dues fases:

- Primer, l'usuari selecciona un objecte.
- Després, l'usuari selecciona una acció que s'aplicarà sobre aquest objecte.

De vegades intervenen dos objectes: aquell amb el qual l'usuari treballa i aquell al qual transfereix informació*.

*** Per exemple, quan un fitxer es mou a una carpeta.**

La selecció d'objectes

La selecció d'objectes consisteix en la identificació per l'usuari d'un objecte o més amb vista a fer després una acció o més sobre aquests, però no sobre la resta d'objectes.

La selecció es fa principalment clicant els objectes amb el ratolí, però també es pot dur a terme amb el cursor fent servir les tecles que el mouen*. Els objectes seleccionats canvien d'aspecte.

*** Per exemple, les tecles de tabulació.**

La selecció pot ser de dos tipus:

1) Selecció simple: se selecciona un sol element d'una llista (una icona o un caràcter d'un text).

2) Selecció múltiple: els altres casos de selecció, inclosa la selecció total d'un document, llista, taula o diagrama. En la selecció múltiple dels objectes d'una llista o de caràcters d'un text, aquesta pot ser contigua o no.

L'activació

L'activació és la invocació d'objectes com finestres, botons de comandament i menús emergents.

Exemples d'activació d'objectes

Per a activar un botó, se l'invoca mitjançant la selecció i alhora s'engega una acció lligada a aquest.

Per a activar una finestra, cal fer-li clic a qualsevol lloc; llavors la finestra passa al primer terme, hi apareixen els controls de redimensionament i el seu marc pren color, mentre que la finestra que abans era activa perd els controls, el seu marc es torna gris i queda en segon terme.

L'activació de menús emergents es fa sovint amb el botó dret del ratolí.

Transferència de dades

La transferència de dades consisteix a moure objectes d'un objecte contenidor (**objecte font**) a un altre (**objecte de destinació**).

Les modalitats generals de transferència de dades que podem trobar són:

- Arrossegar i deixar*, per manipulació directa.

* En anglès, *drag and drop*.

- Retallar/copiar i enganxar mitjançant el portapapers**, per manipulació indirecta.

** En anglès, *cut/copy and paste*.

La realimentació

La finalitat de la realimentació* és que l'usuari vegi immediatament l'efecte de les seves accions, fins i tot quan encara no s'han completat.

* En anglès, *feedback*.

Exemples de modalitats de realimentació

Hi ha moltes modalitats de realimentació; a continuació n'esmentem algunes:

- Els canvis de la forma del cursor quan se situa en diferents parts d'una finestra.

- Els canvis de color dels objectes seleccionats.
- La silueta de l'objecte font que segueix el cursor durant una operació d'arrossegament, mentre que l'objecte de destinació canvia de color quan s'hi entra.
- Els indicadors de progrés d'operacions llargues, en forma de barres buides (graduades o no) que es van omplint, o de percentatges que es van actualitzant, o de missatges.

Estils d'interacció

La interacció amb la interfície d'usuari es pot dur a terme mitjançant menús, formularis, pregunta-resposta, que pot ser lineal (sempre la mateixa sèrie de preguntes) o jeràrquica (cada pregunta depèn de les respostes anteriors); llenguatge de comandaments, tecles de funció, que poden ser dedicades (sempre la mateixa funció) o bé programables; manipulació directa o llenguatge natural.

Manipulació directa i indirecta

La manipulació dels objectes per l'usuari pot tenir lloc d'una manera més concreta o més abstracta, segons una gamma de possibilitats força àmplia; els extrems d'aquesta gamma són els estils genèrics següents:

a) Manipulació directa, en la qual l'usuari indica al sistema l'acció que vol dur a terme sobre l'objecte seleccionat tot simulant-la amb el ratolí estirant l'objecte, arrossegant-lo, etc.

b) Manipulació indirecta, en la qual, un cop seleccionat l'objecte, l'usuari indica al sistema l'acció que vol dur a terme sobre l'objecte d'una manera abstracta.

Definició d'*abstracte* i *concret*

La definició de què és concret i què és abstracte depèn de l'usuari, atès que per a un matemàtic el fet d'operar amb equacions es podria entendre com una manipulació directa.

Els avantatges de la manipulació directa són els següents:

- La manipulació directa fa l'aprenentatge molt més fàcil.
- L'usuari recorda millor l'ús del producte.
- L'usuari té la sensació que controla més allò que passa.
- L'usuari veu de manera immediata el resultat de les seves accions.

D'altra banda, la manipulació directa presenta els inconvenients següents:

- Sovint ocupa molt espai a la pantalla.
- De vegades, el seu ús és més lent.
- No sempre és fàcil de trobar un model adient.

Ajut en línia a l'usuari

L'ajut en línia* a l'usuari és una alternativa als tradicionals manuals impresos que té importants avantatges en comparació amb aquests:

*** En anglès, *on line*.**

- No ocupa lloc en prestatges ni a la taula de treball.
- No té deteriorament físic.
- És més fàcil i barat de posar al dia.
- Es presta a l'aprenentatge interactiu (tutorials en línia).

- L'usuari pot recórrer a la informació que busca d'una manera més ràpida i flexible.
- No té una estructura lineal, sinó en xarxa (hipertext).

Ara bé, també presenta alguns inconvenients i limitacions. A continuació els esmentem:

- Generalment és una mica més difícil llegir d'una pantalla que d'un manual.
- El pas entre pàgines consecutives és més lent (en canvi, el salt a una pàgina referenciada és molt més ràpid).
- No es poden fer anotacions al marge (però hi ha sistemes que permeten que l'usuari afegeixi anotacions pròpies als textos d'ajuda).
- L'ajuda presentada tapa temporalment la finestra on treballava l'usuari, si més no, en part.

7.2. El disseny de les interfícies gràfiques

En primer lloc veurem algunes característiques que convé que tinguin les interfícies, després n'estudiarem el procés del disseny i, finalment, farem algunes recomanacions sobre l'ús dels diferents elements gràfics.

7.2.1. Característiques d'un bon disseny

El disseny ha de tendir a aconseguir el millor compromís, per a un programari concret, entre les característiques següents:

a) Sensació de control: els usuaris no volen tenir la sensació de ser controlats per l'ordinador, i fins i tot en el cas de tutorials guiats convé deixar que l'usuari prengui la iniciativa de tant en tant; convé que la interfície sigui essencialment passiva i l'ajuda en línia, discreta.

b) Adequació a l'usuari: els usuaris no són iguals que el dissenyador (no són professionals de les GUI) i tampoc no són iguals entre ells.

c) Familiaritat per a l'usuari: convé fer servir els conceptes, terminologia i organització a què l'usuari està acostumat.

d) Adequació a la tasca de l'usuari: la interfície gràfica d'usuari ha d'integrar-se en el flux de treball de l'usuari.

e) Adequació al flux d'operacions: hi ha casos en què l'usuari canvia de tasca de manera freqüent i imprevisible; en aquestes situacions cal que el canvi de tasca informàtica sigui àgil. En canvi, en ocasions l'usuari canvia d'una tasca determinada a una altra tasca, que és sempre la mateixa; aleshores és convenient que el sistema hi passi directament.

f) Continuïtat amb les versions anteriors del programari: tot i que un programari nou sempre es fa per a millorar, convé tanmateix aprofitar al màxim l'experiència de l'usuari amb les versions anteriors, i presentar-li la mateixa cosa de la mateixa manera si no hi ha una raó de pes per a fer-ho diferent.

g) Manipulació directa: generalment serà millor per a l'usuari. Ara bé, la manipulació directa no sempre és viable.

h) Flexibilitat: convé que la interfície sigui adaptable a les preferències de l'usuari, i també al seu nivell d'expertesa.

i) Temps de resposta adequat: l'usuari espera que l'ordinador respongui ràpid, però sobretot que cada vegada que fa la mateixa operació trigui més o menys el mateix temps. Quan es preveu que una operació pot ser llarga, convé treure un missatge per a tranquil·litzar l'usuari que digui que l'operació va endavant, o bé mostrar-li l'avanç del procés per mitjà d'una barra que va canviant de color per parts o un percentatge d'acompliment de la tasca.

j) Atenció a la possibilitat d'errors d'usuari.

k) *Feedback* immediat a l'usuari: d'una banda, l'usuari s'ha de poder adonar del fet que la seva acció ha estat acceptada o rebutjada o bé ja s'ha completat o encara no. D'altra banda, quan l'acció s'ha completat, l'usuari n'ha de poder veure el resultat clarament (punters i cursors situats immediatament, siluetes que segueixen l'arrossegament d'objectes), etc.

l) Sensació de seguretat de l'usuari: s'ha de permetre que l'usuari explori el producte sense risc de perdre informació o de perdre's, permetent-li desfer la darrera acció i demanant confirmació de les accions perilloses.

m) Robustesa: l'ideal de robustesa és que, faci el que faci l'usuari, el programari no es descontroli.

n) Coherència: cal que els elements que fan el mateix tipus de funció tinguin el mateix aspecte: títols, noms de camps. És desitjable que hi hagi consistència tant dins d'una aplicació com entre aplicacions, ja que en tots dos casos pot facilitar extraordinàriament l'aprenentatge. En el cas d'interfícies implementades amb una eina orientada a objectes, l'ús sistemàtic de l'herència facilita el manteniment de la coherència.

Exemple de coherència

Convé que els botons en pantalles diferents que serveixen per a la mateixa funció siguin iguals quant a forma, mides, color, text i, si escau, posició.

o) Facilitat d'utilització.

p) Facilitat d'aprenentatge.

Sovint és difícil de fer compatibles totes aquestes característiques. 

7.2.2. Mètode de disseny de les interfícies gràfiques

Considerarem un mètode amb les etapes següents: elaboració d'una guia d'estil, disseny i avaluació del disseny quant a les característiques de bon disseny esmentades i al compliment de la normativa i recomanacions d'estil.

Vegeu les característiques de bon disseny esmentades al subapartat 7.2.1 d'aquest mòdul didàctic.

1) Elaboració d'una guia d'estil

La guia d'estil ha d'incloure una normativa sobre tots aquells aspectes de la interfície que convé que estiguin normalitzats:

- Finestres d'aplicació, de document i de caixes de diàleg: estil, colors i controls.
- Tipus de lletra i icones estàndard.
- Controls estàndard, com ara botons de confirmació, cancel·lació i ajuda.
- *Feedback*: formes del punter i del cursor i l'ús de cada una, etc.
- Si l'eina d'interfície gràfica és orientada a objectes, definir les superclasses bàsiques, de les quals es definiran subclasses a l'etapa d'implementació.

2) Disseny

L'etapa de disseny parteix de l'esquema del contingut de les finestres i de l'especificació formal dels casos d'ús obtinguts a l'etapa d'anàlisi.

L'**esquema del contingut** només indica quines dades han de figurar a cada finestra i quines d'aquestes tenen una llista de valors possibles limitada, i aquells aspectes que hagin de ser comuns a totes les finestres d'un mateix tipus estaran fixats per la guia d'estil. A partir d'aquesta informació s'ha de determinar la millor manera de representar cada dada amb l'eina d'interfícies gràfiques disponible i després establir el format de cada finestra, ja sigui mitjançant l'eina amb què s'implementarà la interfície d'usuari definitivament, ja sigui fent alguna mena de prototip que després s'haurà d'implementar de manera fidel.

Tenint en compte el diàleg* descrit dins l'especificació formal dels casos d'ús, es dissenyarà l'instrument per a seleccionar la funció (menús, barres d'eines, etc.) i també els controls que serviran per a passar d'una finestra a una altra. Caldrà buscar unes icones adequades; per a assegurar-se que ho són, caldrà comprovar si els usuaris les reconeixen fàcilment.

*** El diàleg és l'intercanvi de missatges entre el programari i l'usuari.**

7.2.3. Recomanacions sobre el disseny d'objectes gràfics

Aquí veurem algunes recomanacions relatives a finestres, menús, controls i caixes de diàleg i també algunes orientacions sobre l'ajuda en línia.

Recomanacions sobre finestres

a) Quan s'obri una finestra, convé situar-la davant de tot, donar-li el focus i mantenir una llista seleccionable de finestres obertes; si cal, convé posar-hi controls de maximitzar i minimitzar.

b) Quan es reobri una finestra, és recomanable que ho faci a la mateixa posició i amb les mateixes mides.

c) És convenient preveure algun control per a tancar les finestres d'aplicació i de document, tancant o no l'aplicació o document respectius.

d) No s'ha de permetre que la finestra es pugui moure a una posició on sigui inaccessible, i convé fer saber a l'usuari que s'és a punt de moure la finestra o de redimensionar-la (punter amb una forma especial) i del moviment en curs.

e) Quan es maximitza una finestra, cal guardar-ne les dimensions anteriors i posar-hi el control de minimitzar o restaurar.

f) Quan es minimitza una finestra per primera vegada, s'ha de col·locar la icona a la part baixa de l'escriptori; si la finestra ja havia estat minimitzada, cal posar la icona al lloc on era abans.

Recomanacions sobre menús

a) Cal presentar de manera diferent de la resta l'opció que se selecciona i les que estan inhabilitades; en el cas d'opcions binàries, convé indicar quines estan seleccionades, o bé canviar-ne el text segons estiguin seleccionades o no.

b) Les opcions que es preveu que es faran servir més han d'anar al començament del menú.

c) Convé evitar un nombre de nivells excessiu en els menús en cascada.

d) Els menús *pop-up* han de ser d'un sol nivell i no repetir les opcions dels menús ordinaris.

Recomanacions sobre controls

a) La llista *combo* es farà servir quan un camp pugui prendre uns valors molt freqüents (que seran els de la llista), però tanmateix, en pugui prendre d'altres, que s'han de poder teclejar.

b) La llista ordinària és adequada quan la llista de valors possibles és tancada i reduïda. Tanmateix, si hi ha pocs valors possibles, cal considerar l'alternativa molt més ràpida dels botons de ràdio, si els valors són excloents entre ells, o les *check boxes* en el cas contrari.

c) Tant la llista ordinària com la llista *combo* es faran desplegables quan hi hagi problemes d'espai, i persistents amb barra de desplaçament quan es disposi d'espai però la llista tingui massa valors. Els valors de la llista han d'anar ordenats per freqüència d'ús decreixent o bé per ordre alfabètic, etc.

d) Una *spin box* és convenient principalment quan els valors suggerits són numèrics amb intervals iguals. Es pot admetre o no un valor teclejat directament.

e) Els camps de text d'una sola línia poden tenir visible només la part on hi ha el cursor. Els camps de línies múltiples poden tenir barra de desplaçament vertical i horitzontal, i pot ser necessari permetre els salts de línia amb una tecla* que ja no es podrà dedicar a un altre ús en la mateixa finestra.

*** Per exemple, el retorn de carro.**

Recomanacions sobre caixes de diàleg

a) Una caixa de diàleg modal és adequada quan cal interrompre el procés per a advertir l'usuari d'una circumstància anòmala o demanar-li una confirmació d'una acció conflictiva que hagi demanat. Ha d'aparèixer en primer terme i rebre el focus. No s'ha de recórrer gaire sovint a les caixes de diàleg, ja que fan perdre temps a l'usuari.

b) Si la caixa de diàleg pot tapar informació que l'usuari pot necessitar per a prendre la decisió que se li demana, cal que es pugui desplaçar per la pantalla.

c) Les caixes de diàleg no han d'omplir tota la pantalla. Poden tenir doble format: primer es presenta una caixa de diàleg amb els controls imprescindibles, i es dóna opció a l'usuari de desplegar la resta.

d) En una caixa de diàleg hi ha d'haver almenys un botó per a tancar-la; sovint n'hi ha dos, un per a confirmar i un altre per a cancel·lar, que es recomana que estiguin separats dels eventuals botons de comandament. Es recomana que els botons estiguin alineats horitzontalment (a la part de baix, cap a l'angle dret) o bé verticalment. Convé que hi hagi un botó per omissió (això és, el botó que se selecciona amb la tecla de retorn de carro) amb el contorn més gruixut que els altres botons i col·locat a dalt de tot d'una columna de botons o a l'extrem d'una fila. El botó d'ajuda també ha d'estar en una posició destacada.

e) Si hi ha un nombre significatiu de controls, s'han de distribuir en grups. Els controls d'un grup han d'estar relacionats lògicament, però no cal que siguin del mateix tipus*. Els grups han d'estar envoltats per alguna línia o bé han de tenir un text de capçalera del grup i els seus camps han d'estar alineats entre si i sagnats respecte al text de capçalera; si hi ha diversos grups, els textos de capçalera respectius han d'estar alineats (vegeu els grups de controls de la figura 18).

*** En un mateix grup de controls pot haver-hi camps de text i grups de botons de ràdio.**

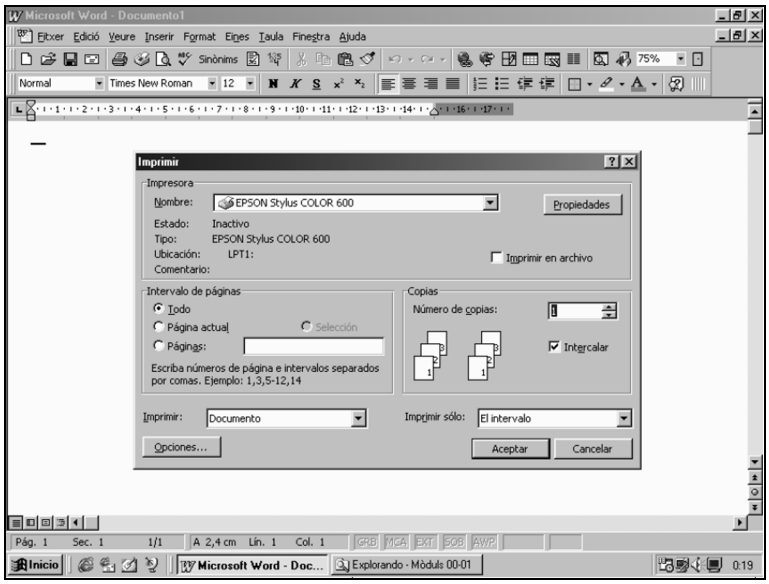

Figura 18

f) Les carpetes amb separadors són convenients quan la quantitat d'informació que es presenta és tan gran que pot omplir diverses vegades la finestra de la caixa de diàleg.

Principis del disseny de l'ajuda en línia

A continuació esmentem alguns principis que cal tenir en compte a l'hora de fer el disseny de l'ajuda en línia:

1) L'accés a l'ajuda ha de ser fàcil de trobar.

2) La redacció dels textos s'ha de fer posant-se en el lloc de l'usuari, igual que en el cas dels manuals.

3) La informació ha de ser tan completa com sigui possible, però ha d'estar estructurada de manera que sigui fàcil de localitzar exactament allò que l'usuari vol en cada moment.

4) S'ha de poder tornar als textos d'ajuda ja vistos a la mateixa sessió d'ajuda.

5) S'ha de poder sortir fàcilment de l'ajuda i tornar al punt en què l'usuari havia interromput el seu treball.

6) El temps de resposta ha de ser breu, ja que sovint l'usuari no troba allò que buscava al primer intent.

7) Ha de tenir un format i estil coherents tant amb la resta de la interfície com amb la resta de la documentació.

8. Disseny dels subsistemes

En començar el disseny, dins el disseny arquitèctonic, s'havia establert una descomposició del sistema de programari en subsistemes i s'havien especificat les interfícies respectives i les dependències entre aquests; tot això era provisional perquè no es coneixien encara totes les classes del diagrama estàtic del disseny.

En acabar el disseny convé omplir de contingut els subsistemes esmentats especificant quines classes comprèn cada un, fent explícites les dependències en el nivell de classe i d'operació entre si i comprovant si entre totes les classes i subsistemes de cada subsistema s'implementa la interfície explícita o implícita de la qual és responsable el subsistema. Si hi ha interfícies definides explícitament, convé que les dependències quedin especificades no pas entre subsistemes sinó entre un subsistema i una interfície, per tal de facilitar la substitució d'un subsistema per un altre que implementi la mateixa interfície.

9. Exemple

En aquest apartat continuem l'exemple dels mòduls anteriors. Vegem-ne ara el disseny.

En situacions reals les especificacions del disseny farien servir la terminologia pròpia del llenguatge de programació, l'eina d'interfície gràfica i el sistema de gestió de bases de dades concrets que s'hagin d'utilitzar a la implementació, però en tractar-se d'un exemple general s'ha utilitzat una terminologia neutra.

9.1. El disseny arquitectònic

Atès que no es faran servir subsistemes ja existents ni tampoc no es preveu que cap subsistema sigui reutilitzable, els subsistemes corresponen als paquets identificats a l'anàlisi.

9.2. El disseny dels casos d'ús

Cas d'ús 1: "Afegir local"

Quan l'usuari selecciona l'opció corresponent al menú, es crida l'operació crear de la classe *AddicioLocal*. Aquesta operació instancia la classe, determina el valor del número correlatiu del local, obre una finestra i hi instancia el format de pantalla *PAfegirLocal*.

Un altre procediment

En comptes d'especificacions textuals (sagnades per tal d'indicar quines condicions estan subordinades a d'altres) es podrien fer diagrames de seqüència, però en els casos reals sovint serien molt complexos.

1) Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *AddicioLocal* i es torna control al menú.

2) Si l'usuari confirma, es crida l'operació *buscar* de la classe *Propietari* amb el NIF del propietari com a paràmetre. Aleshores aquesta classe crida *materialitzar de GestorClient* amb el mateix paràmetre.

Vegeu els gestors de disc d'aquest exemple al subapartat 9.4.1 d'aquest mòdul didàctic.

a) Si no es troba el propietari, es crida l'operació *crear* de la classe *AddicioPropietari* descrita al cas d'ús 2.

b) Si hi ha el propietari o s'ha afegit, es crida l'operació *crear* de la classe *BotigaMagatzem*, *Oficina*, *Immoble* o *Polivalent* i se li passa com a paràmetres la zona, el tipus, el número, l'adreça del local, la superfície, el NIF del propietari, les característiques, les restriccions i l'agent i el volum en el cas de botiga-magatzem i polivalent, i les característiques polivalents en el cas de polivalent; aquesta operació crea un objecte de la subclasse corresponent, mou aquests valors un a un als seus atributs i crida *gravar* de *GestorLocal* amb l'objecte creat com a paràmetre.

Cas d'ús 2: "Afegir propietari"

Quan l'usuari selecciona l'opció corresponent al menú, es crida l'operació *crear* de la classe *AddicioPropietari* i aquesta obre una finestra modal per a introduir el NIF.

1) Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *AddicioPropietari* i es torna el control al menú.

2) Si l'usuari confirma, es crida l'operació *buscar* de la classe *Propietari* amb el NIF del propietari com a paràmetre i aleshores la classe crida *materialitzar* de *GestorClient* amb el mateix paràmetre.

a) Si es troba el propietari, s'obre una finestra modal i es mostra el missatge "El propietari amb NIF x ja existeix".

b) Si no es troba el propietari, s'obre una finestra i s'hi presenta el format de pantalla *PAfegirPropietari*.

- Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *AddicioPropietari* i es torna el control al menú.
- Si l'usuari confirma, es crida l'operació *crear* de la classe *Propietari* i se li passen com a paràmetres el NIF, els cognoms i nom/raó social, l'adreça i el telèfon. La crea l'objecte, li posa els valors dels atributs i crida *gravar* de *GestorClients* amb l'objecte com a paràmetre.

Cas d'ús 3: "Introduir informe"

Quan l'usuari selecciona l'opció corresponent al menú, es crida l'operació *crear* de la classe *IntroduccioInforme*. Aquesta operació instancia la classe i obre una finestra modal que demana el codi del local.

1) Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *IntroduccioInforme* i es torna control al menú.

2) Si l'usuari confirma, es crida l'operació *buscar* de la classe *Local* amb els components del codi del local (zona, tipus i número) com a paràmetres. Aquesta cridarà *materialitzar* de *GestorLocal*.

a) Si no es troba el local, es mostra el missatge "El local xxx no existeix" en una finestra modal, es crida l'operació *esborrar* de la classe *IntroduccioInforme* i es torna el control al menú.

b) Altrament, s'obre una finestra en la qual es presenta el format de pantalla *PintroduirInforme*.

- Si l'usuari introdueix les dades i confirma, es crida l'operació *modificar* de la classe *Local* i se li passen com a paràmetres els mateixos que per a l'operació *crear* més un valor nul per al NIF del llogater, la forma, l'accessibilitat, les instal·lacions, la visibilitat, la conservació, el preu i l'inspector. Aquesta ope-

ració substitueix els valors dels atributs pels valors teclejats, crida *gravar* de *GestorLocal* i li passa l'objecte de *Local* com a paràmetre.

- Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *AddicioPropietari* i es torna el control al menú.

Cas d´ús 4: "Introduir contracte"

Quan l'usuari selecciona l'opció corresponent al menú, es crida l'operació *crear* de la classe *IntroduccioContracte*. L'operació instancia la classe, obre una finestra i hi presenta el format de pantalla *PIntroduirContracte*.

1) Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *IntroduccioContracte* i es torna control al menú.

2) Si l'usuari confirma, es crida l'operació *buscar* de la classe *Local* amb la zona, el tipus i el número com a paràmetres.

a) Si no es troba el local, es mostra el missatge "El local x x x no existeix", es crida l'operació *esborrar* de la classe *IntroduccioContracte* i es torna el control al menú. Altrament, es crida l'operació buscar de la classe *Llogater* amb el NIF com a paràmetre.

b) Si no es troba el llogater, es crida l'operació *crear* de la classe *AddicioLlogater*, que obre una finestra i hi presenta el format de pantalla *PIntroduirLlogater*.

- Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *IntroduccioContracte* i es torna el control al menú.

- Si l'usuari confirma, es crida l'operació *crear* de la classe *Llogater* i se li passen com a paràmetres el NIF, els cognoms i nom/raó social, l'adreça i el telèfon. Aleshores la classe cridarà *gravar* de *GestorClient* amb els mateixos paràmetres.

  Després es crida l'operació *crear* de la classe *Contracte* i se li passa com a paràmetres el NIF del llogater i el NIF del propietari, el codi del local, la data d'inici i la data de finalització. Aquesta operació crea un objecte *Contractes*, li posa els valors dels atributs i crida aquestes altres: *buscar* de *Propietari* i de *Llogater* amb els NIF respectius com a paràmetres, *buscar* de *Local* amb la zona, el tipus i el número com a paràmetres, *modificar* de *Local* amb les dades llegides més el NIF del llogater i *gravar* de *GestorContracte*, i imprimeix les dades del contracte: NIF, cognoms i nom/raó social i adreça tant del propietari com del llogater, l'adreça del local, la data d'inici i la data de finalització.

Cas d'ús 5: "Afegir llogater"

Quan l'usuari selecciona l'opció corresponent al menú, es crida l'operació *crear* de la classe *AddicioLlogater*. Aquesta obre una finestra modal per a introduir el NIF.

1) Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *AddicioLlogater* i es torna el control al menú.

2) Si l'usuari confirma, es crida l'operació *buscar* de la classe *Llogater* amb el NIF del llogater com a paràmetre.

a) Si es troba el llogater, s'obre una finestra modal i es mostra el missatge "El llogater amb NIF x ja existeix".

b) Si no es troba el llogater, es crida l'operació *crear* de la classe *AddicioLlogater*, que obre una finestra i hi presenta el format de pantalla *PAfegirLlogater*.

- Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *AddicioLlogater* i es torna control al menú.
- Si l'usuari confirma, es crida l'operació *crear* de la classe *Llogater* i hi passa com a paràmetres el NIF, els cognoms i nom/raó social, l'adreça, el telèfon. Aquesta operació crea un objecte *Client*, li posa els valors dels atributs i crida gravar de *GestorClient* amb l'objecte creat com a paràmetre.

Cas d'ús 6: "Afegir eventual llogater"

Aquest cas és una especialització del cas d'ús número 5 en la qual les classes *Llogater* i *AddicioLlogater* se substitueixen per *EventualLlogater* i *AddicioEventualLlogater* respectivament. L'operació *crear* de la classe *EventualLlogater* té com a paràmetres la llista de zones, el tipus, la superfície i els requisits a més dels paràmetres de *crear* de *Llogater*.

Cas d'ús 7: "Buscar locals"

Quan l'usuari selecciona l'opció corresponent al menú, es crida l'operació *crear* de la classe *CercaLocals*. L'operació instancia la classe, obre una finestra i hi presenta el format de pantalla *PBuscarLocals*.

Casos d'ús addicionals

D'acord amb el que s'ha especificat a l'etapa d'anàlisi (al subapartat 7.1 del mòdul didàctic "Anàlisi orientada a objectes"), caldria preveure casos d'ús addicionals per tal que tota la informació es pogués consultar, modificar i esborrar.

1) Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *CercaLocals* i es torna el control al menú.

2) Si l'usuari confirma, es crida l'operació *consultar* de la classe *Local* amb el NIF del propietari i del llogater, la zona, el tipus, el número i la superfície mínima com a paràmetres. Aquesta operació crida *consultar* de *GestorLocal* amb els mateixos paràmetres. En el cas que el tipus demanat sigui botiga-magatzem o oficina es busquen també els de tipus polivalent.

a) Si no es troba cap local, s'obre una finestra modal i es mostra el missatge "No hi ha cap local que compleixi les condicions".

b) Si es troba almenys un local, s'obre una finestra i s'hi presenta el format de pantalla *PLlistaLocals*.

- Si l'usuari selecciona un local de la llista i confirma, es crida l'operació *buscar* de la classe *Local* i se li passen com a paràmetres la zona, el tipus i el número

del local seleccionat; després s'obre una finestra i s'hi presenta el format de pantalla *PDadesLocal*. Quan l'usuari cancel·la, es torna a treure el format de pantalla *PllistaLocals*.

- Si l'usuari cancel·la, es crida l'operació *esborrar* de la classe *CercaLocals* i es torna el control al menú.

9.3. El diagrama estàtic de disseny

A les classes del diagrama estàtic de disseny s'indiquen només aquells atributs que s'han afegit a la versió d'anàlisi per a implementar les associacions; les operacions s'indiquen totes.

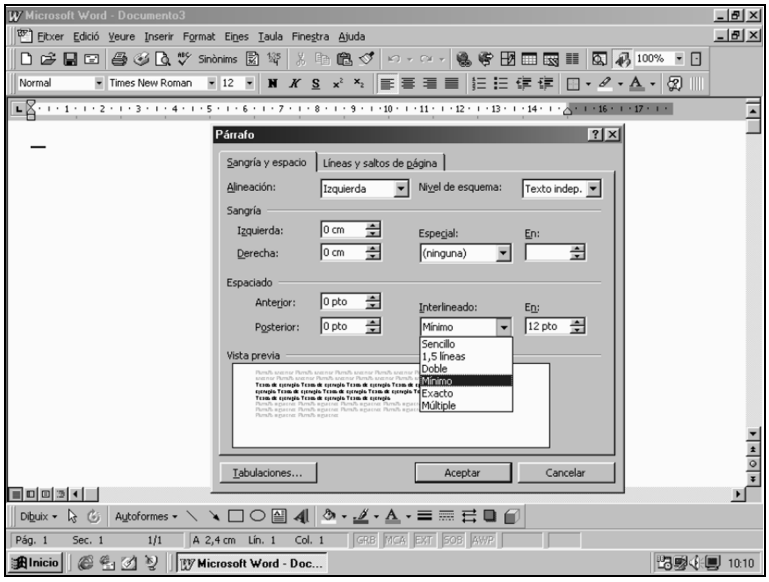

Figura 19

A *Local* s'han inclòs el NIF del propietari i el del llogater per a implementar les associacions d'aquesta classe amb *Propietari* i *Llogater* respectivament; només s'ha implementat un dels sentits d'aquestes associacions, ja que no cal la navegabilitat en sentit contrari. A *Contracte* s'han afegit el NIF del llogater i la zona, tipus i número del local per a implementar les associacions corresponents.

L'herència doble de *Polivalent* s'ha convertit en herència simple de *BotigaMagatzem* sense haver de fer res més, ja que no hereta ni atributs ni operacions d'*Oficina*.

Només s'especifica la llista de paràmetres de les operacions quan és curta, per raons d'espai i perquè ja estan descrites completament a l'especificació dels casos d'ús. Tanmateix, quan es documenten les classes amb eines CASE, que sovint permeten de visualitzar les llistes de paràmetres de diverses maneres, sí que s'especifiquen completament.

Només s'han inclòs les operacions que figuren al disseny dels casos d'ús; totes les classes han de tenir una operació per a esborrar-ne els objectes i totes les classes d'entitats és convenient que tinguin almenys una operació per a modificar-ne els atributs; naturalment, hi hauria d'haver també casos d'ús que les fessin servir.

9.4. El disseny de la persistència

Per a realitzar el disseny de la persistència en aquest exemple, considerarem que es fa servir una base de dades relacional.

9.4.1. Disseny de la base de dades

Per a fer el disseny de la base de dades, primer eliminarem l'herència; després, elaborarem el diagrama ER equivalent al diagrama estàtic de les classes persistents i finalment obtindrem l'especificació de la base de dades relacional.

El diagrama ER i les bases de dades relacionals es tracten a l'assignatura *Bases de dades I*.

Supressió de l'herència

Hi ha dues jerarquies d'herència encapçalades per les classes *Local* i *Client*, respectivament. Entre les opcions esmentades per a la supressió de l'herència (una taula diferent per a cada subclasse, una taula per a la superclasse i una altra per a cada subclasse, i una sola taula per a tot) fem l'elecció següent:

Vegeu les opcions possibles per a la supressió de l'herència al subapartat 6.2.3 d'aquest mòdul didàctic.

1) Per a la jerarquia de *Local*, la tercera, ja que se suposa que les botigues-magatzem seran la majoria dels locals i l'atribut *caracteristiques_polivalents* és de longitud variable i, per tant, el seu valor nul ocupa molt poc espai.

2) Per a la jerarquia de *Client*, la segona, ja que només cal preveure una taula complementària per a una de les subclasses, *EventualLlogater*, i aquesta és minòritària, atès que un eventual llogater, o bé passa a llogater (i llavors se n'esborren els atributs de la subclasse) o bé s'esborra completament al cap de poc temps.

Diagrama entitat-relació

El diagrama entitat relació que correspon a aquest cas és el que veiem a continuació:

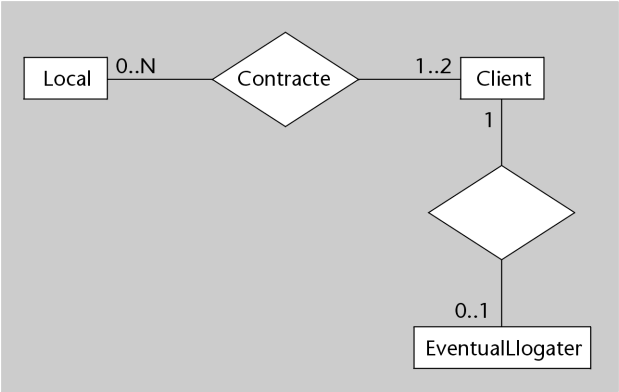

Figura 20

Base de dades relacional

Als casos reals, la descripció de la base de dades es fa en termes d'un sistema de gestió de bases de dades concret, òbviament aquell que es farà servir al projecte; però aquí, per tal que l'exemple sigui més general, s'empra la terminologia general del model relacional.

Les taules de la base de dades seran aquestes:

- *taula_clients*, amb les columnes NIF, *nom*, *adreca* i *telefon*. La clau primària és *NIF*.
- *taula_eventuals_llogaters*, amb les columnes NIF, *tipus_local*, *zona1*, ..., *zona5*, *superficie_minima* i *requisits_diversos*. *NIF* és la clau primària i alhora clau forana corresponent a *NIF* de *taula_clients*.
- *taula_locals*, amb les columnes *zona*, *tipus*, *numero*, *adreca*, *superficie*, *caracteristiques*, *NIF_propietari*, *accessibilitat*, *installacions*, *agent*, *inspector*, *volum* i *caracteristiques_polivalent*. La clau primària es compon dels atributs *zona*, *tipus* i *numero* i *NIF_propietari* és clau forana corresponent a *NIF* de *taula_clients*.
- *taula_contractes*, amb les columnes *zona*, *tipus*, *numero*, *NIF_llogater*, *data_inici*, *data_fi* i *venedor*. La clau primària es compon dels atributs *zona*, *tipus*, *numero*, *NIF_llogater* i *data_inici*. Hi ha dues claus foranes *NIF_llogater* corresponents a *NIF* de *taula_clients*, i una altra composta dels atributs *zona*, *tipus* i *numero* que correspon a la clau primària de *taula_locals*.

Pel que fa als índexs, n'hi hauria d'haver un per cada clau primària; en principi no cal índex per cap de les claus foranes.

9.4.2. Gestors de disc

Aplicarem a aquest cas l'esquema general que seguim en aquest mòdul. Els gestors són *GestorLocal*, *GestorClient* i *GestorContracte*, i són subclasses de *GestorRelacional*:

Vegeu l'esquema general presentat al disseny bàsic de gestors de disc del subapartat 6.2.4 d'aquest mòdul didàctic.

a) *GestorLocal* s'instancia en engegar el sistema, per mitjà de l'operació *crear*. L'operació *llegirFila* llegeix una fila de *taula_locals* identificada per la clau primària amb els valors de totes les columnes. L'operació *filaAObjecte* crea un objecte de *Local* i mou aquests valors un a un als seus atributs. L'operació *objecteAFila* mou un a un els atributs de l'objecte de *Local* als valors de les columnes d'una fila de *taula_locals*. L'operació *gravarFila* grava aquesta fila a *taula_locals*. L'operació *consultar* fa una consulta a *taula_locals* amb una condició que dependrà de quins dels paràmetres tinguin un valor no nul i crida *filaAObjecte* per a cada una de les files obtingudes.

b) *GestorClient* s'instancia en engegar el sistema per mitjà de l'operació *crear*. L'operació *llegirFila* llegeix una fila de *taula_clients* i una altra de *taula_eventuals_llogaters* identificades per la clau primària, i l'operació *filaAObjecte* crea un objecte d'*EventualLlogater*, si algun dels atributs específics d'aquesta subclasse té un valor no nul, o bé un objecte de *Client* i li mou els valors de les files esmentades. L'operació *objecteAFila* mou un a un els atributs de l'objecte de *Client* als valors de les columnes d'una fila de *taula_clients* i d'una altra de *taula_eventuals_llogaters*. L'operació *gravarFila* grava la primera d'aquestes files sempre i la segona només si algun dels atributs específics d'*EventualLlogater* té un valor no nul.

c) *GestorContracte* s'instancia en engegar el sistema per mitjà de l'operació *crear*. L'operació *llegirFila* llegeix una fila de *taula_contractes* identificada per la clau primària amb els valors de totes les columnes i l'operació *filaAObjecte* crea un objecte de *Local* i mou aquests valors un a un als seus atributs. L'operació *objecteAFila* mou un a un els atributs de l'objecte de *Contracte* als valors de les columnes d'una fila de *taula_contractes*. L'operació *gravarFila* grava aquesta fila a *taula_contractes*.

9.5. El disseny de la interfície d'usuari

En aquest subapartat veurem com es representen les diferents dades, com s'implementen els diàlegs i el disseny d'algunes finestres. No establirem cap guia d'estil concreta, ja que pot variar bastant d'una organització a una altra.

9.5.1. Presentació de les dades

Amb vista a aconseguir una interfície d'usuari coherent, cal que cada dada es presenti de la mateixa manera a tots els formularis on surt; començarem per decidir com es presentarà cada dada:

- Les dades de tipus textual es presentaran en forma d'àrees de text amb barra de desplaçament vertical, ja que permeten textos mitjanament llargs i són menys incòmodes que les que tenen també barra de desplaçament horitzontal.

- Els camps de longitud fixa es presentaran com a camp de text quan són només de sortida; quan són també d'entrada, si el nombre de valors possibles és reduït, el valor es triarà d'una llista (una llista *combo*, si s'hi poden afegir valors i una llista ordinària, en cas contrari); en comptes d'una llista ordinària de menys de sis valors fixos es farà servir un conjunt de botons de ràdio.

- Per a dades booleanes, farem servir *check boxes*.

Aplicant aquestes normes, el tipus de local s'introduirà mitjançant botons de ràdio; la zona, l'agent, l'inspector i la conservació se seleccionaran de llistes ordinàries; el número de local, els NIF, els telèfons, les superfícies, les dates i el preu seran camps de text, i també ho seran totes les dades anteriors quan no es puguin modificar; la resta de dades seran textos amb barra de desplaçament vertical. No hi ha cap dada booleana i, per tant, les *check boxes* només s'usaran per a opcions dins les finestres.

9.5.2. Implementació dels diàlegs

Veurem primer la implementació de la selecció de la funció i després els instruments per a passar d'una pantalla a una altra.

Els menús

Les opcions del sistema de menús són les que corresponen als casos d'ús més les opcions de sortir del sistema i de cridar l'ajuda en línia. Com que són poques i caben al menú de barra, no calen menús desplegables. Tampoc cal una barra d'eines, ja que seria una duplicació del menú de barra amb icones en lloc d'etiquetes.

El menú de barra surt al capdamunt de la pantalla inicial i té les entrades corresponents a aquestes etiquetes, d'esquerra a dreta: "Sortir", "Alta local", "Informe", "Contracte", "Eventual llogater", "Locals", "Propietari", "Llogater" i "Ajuda". "Sortir" s'ha posat com a primera opció i "Ajuda" com a última perquè és la mateixa col·locació que a la resta de programari que fan servir els usuaris. S'ha fet servir l'etiqueta "Alta local" en lloc de "Local" perquè la darrera s'assembla massa a una altra.

Pas d'una pantalla a una altra

L'opció de cancel·lar la funció es demana per mitjà d'un botó amb l'etiqueta "Cancel·lar"; la crida a l'ajuda en línia es fa amb un botó amb l'etiqueta "Ajuda"; per a confirmar les dades introduïdes es clica un botó amb l'etiqueta "Confirmar", i per a sortir d'una finestra que mostra el resultat d'una consulta hi ha un botó amb l'etiqueta "Sortir".

Tots els elements que s'esmenten aquí haurien d'estar especificats en la guia d'estil.

9.5.3. Format d'algunes finestres

En una situació real, les finestres i el seu contingut estarien codificades amb l'eina amb què s'implementaran finalment, o bé amb una eina de prototipatge (en aquest darrer cas, durant la implementació es codificaria una interfície amb la mateixa aparença per mitjà de l'eina definitiva). Aquí hi ha un dibuix esquemàtic de les finestres, tenint en compte que molts detalls dependrien tant de les prescripcions de la guia d'estil, com de l'eina.

Veurem dues de les finestres: la de l'alta de local (*PantallaAfegirLocal*) i una finestra modal per a un missatge.

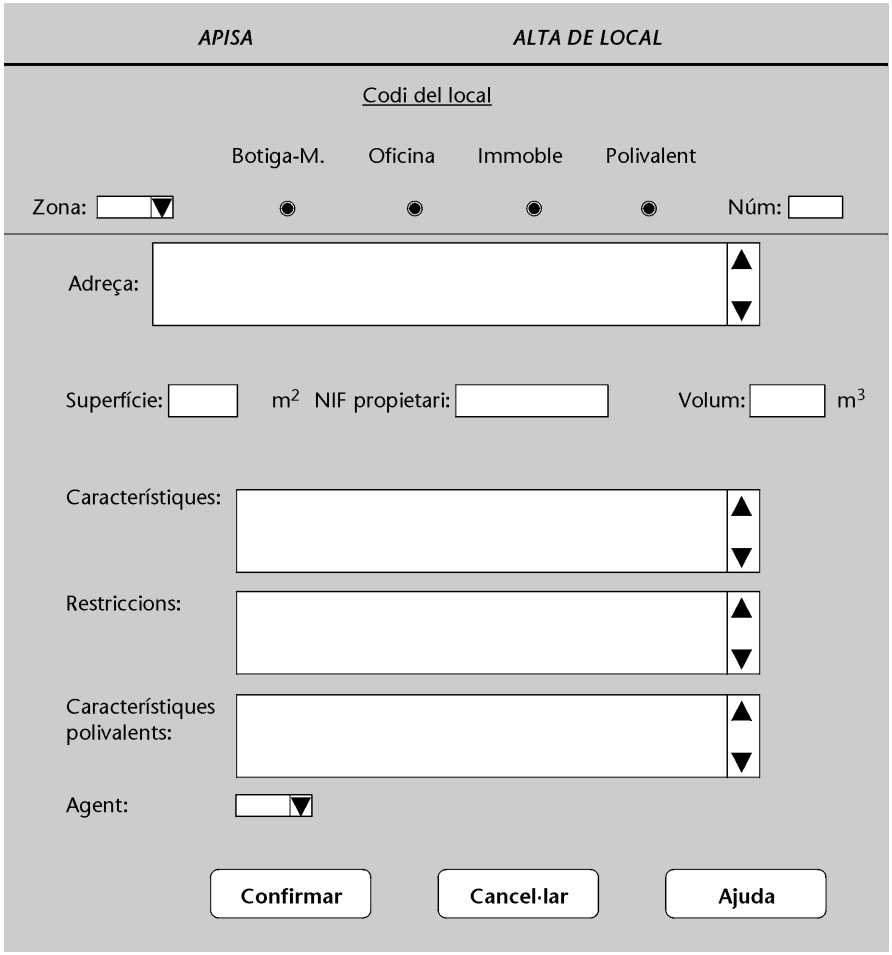

Figura 21

Tal com s'ha dit abans, el tipus de local es representa mitjançant un grup de botons de ràdio que corresponen als quatre valors possibles. Quan es presenta la finestra està seleccionat el valor "Botiga-Magatzem", que és el més freqüent; això té l'avantatge que moltes vegades l'usuari no haurà de seleccionar el valor i l'inconvenient que, si n'havia d'haver seleccionat un altre i no ho fa, dóna un valor erroni i no ha rebut cap avís.

Una punta de fletxa sola indica una llista desplegable, i un parell significa una barra de desplaçament vertical.

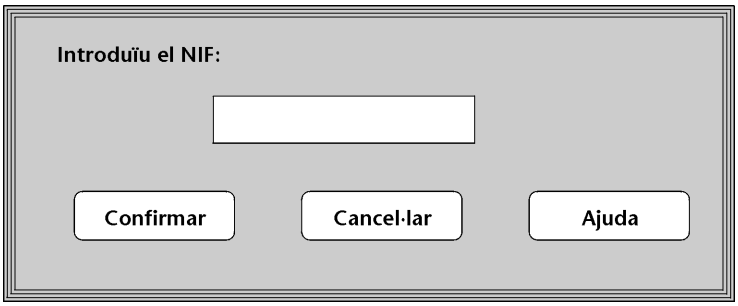

Figura 22

No totes les finestres modals han de tenir els tres botons indicats; per exemple, una finestra que simplement presenti un missatge molt fàcil d'interpretar només ha de tenir un botó "Sortir".

9.6. El disseny dels subsistemes

Com hem vist, els subsistemes coincideixen amb els paquets establerts a l'etapa d'anàlisi. Aquests paquets són els que veiem a continuació:

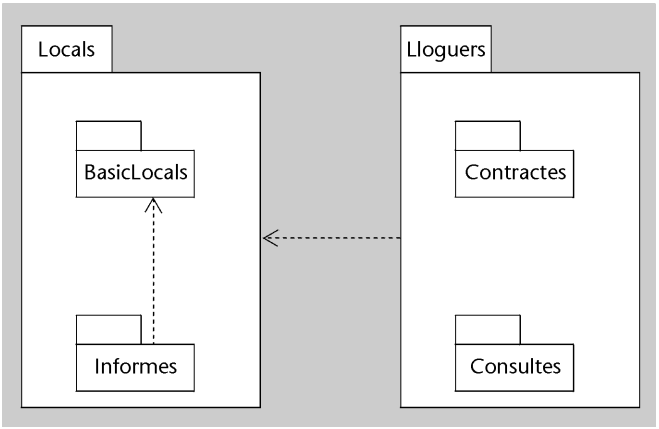

Figura 23

El contingut dels subsistemes és el següent:

a) *BasicLocals* comprèn les classes *Local* amb totes les seves subclasses i *Client* amb la seva subclasse *Propietari*, i els gestors de disc corresponents; pel que fa als casos d'ús, inclou l'1, "Afegir local", i el 2, "Afegir propietari".

b) *Informes* comprèn el cas d'ús 3, "Introduir informe".

c) *Contractes* comprèn les classes *Llogater* i *Contracte* i els casos d'ús 4 ("Introduir contracte"), i 5 ("Afegir llogater").

d) *Consultes* conté la classe *EventualLlogater* i els casos d'ús 6 ("Afegir eventual llogater"), i 7 ("Buscar locals").

De la utilització de les classes i les seves operacions en els casos d'ús obtenim que les interfícies (no formalitzades en termes d'UML) d'aquests subsistemes són les següents:

- *Locals*: les operacions *buscar, modificar* i *consultar* de la *classe Local*.

- *BasicLocals*: les operacions *buscar* i *modificar* de la *classe Local*.

Resum

A l'etapa de disseny s'especifica com s'ha d'implementar el programari per tal que satisfaci les necessitats d'informació recollides a l'etapa de recollida i documentació de requisits i formalitzades a l'etapa d'anàlisi.

La reutilització és un aspecte cada vegada més important de la construcció de programari; durant el disseny es prenen les decisions sobre la reutilització. Els bastiments, components i classes ja estan implementats i, per tant, es poden incloure directament dins el programari a l'hora d'implementar-lo, mentre que els patrons únicament són idees (provades i documentades) per a resoldre problemes de disseny concrets.

El primer pas del disseny és el disseny arquitectònic, dins el qual es decideixen dos aspectes que condicionaran de manera decisiva la resta del disseny: l'eventual utilització de bastiments i la descomposició del programari en subsistemes.

La part principal de l'etapa és el disseny de la implementació dels casos d'ús, que consisteix en l'especificació de la col·laboració entre les classes de control i també entre aquestes i les classes d'entitat; així, s'obté el diagrama estàtic d'anàlisi que, un cop revisat amb vista a aconseguir un acoblament baix, coherència alta i flexibilitat enfront dels probables canvis futurs, es complementa amb l'especificació detallada de les classes que conté, i en especial de les seves operacions.

També cal dissenyar la interfície d'usuari i els mecanismes relatius a la persistència de les classes d'entitats.

Activitats

1. Dueu a terme la documentació de disseny per al cas de l'assegurança d'automòbils partint de la documentació elaborada als mòduls anteriors.

Vegeu les activitat dels mòduls "Recollida i documentació de requisits" i "Anàlisi orientada a objectes" d'aquesta assignatura.

Exercicis d'autoavaluació

1. Té sentit dir "he inventat un patró"?

2. Per què es diu que els patrons creen vocabulari?

3. Per què cal normalitzar els noms de classes, operacions i atributs?

4. Quins avantatges i inconvenients té la supressió de l'herència múltiple per delegació comparada amb la supressió per duplicació?

5. Quins avantatges i inconvenients s'aconseguirien a l'exemple desenvolupat al llarg de l'assignatura si no s'haguessin definit subclasses a *Local* i, en canvi, s'acceptés que el atributs *volum* i *caracteristiques_polivalents* poguessin tenir el valor nul?

Vegeu l'exemple desenvolupat a l'apartat 9 d'aquest mòdul didàctic.

6. Com s'hauria de modificar la classe *Local* de l'exemple desenvolupat al llarg de l'assignatura perquè pugui representar que un immoble es compon de determinats locals d'altres tipus? Com seria el cas d'ús corresponent?

7. Valdria la pena definir la classe de control *AddicioEventualLlogater* com a subclasse d'*AddicioLlogater* de l'exemple?

8. Per a poder modificar qualsevol atribut d'una classe hi ha almenys dues opcions: definir una operació única per a modificar-los tots o bé definir una operació diferent per a cada un. Indiqueu els avantatges de cada una d'aquestes opcions.

Solucionari

1. L'expressió "he inventat un patró" no té sentit. Els patrons són en tot cas invents de disseny que ja han estat provats extensament.

Vegeu el subapartat 2.2 d'aquest mòdul didàctic.

2. Es diu que els patrons creen vocabulari perquè dins un disseny n'hi ha prou de dir que per a resoldre un problema determinat s'aplica un cert patró conegut, i llavors no cal donar més detalls.

Vegeu el subapartat 2.3.1 d'aquest mòdul didàctic.

3. És important normalitzar els noms de les classes, les operacions i els atributs perquè el diagrama estàtic de l'anàlisi haurà fet servir la terminologia de l'usuari i pot ser que aquesta vulneri les normes que els desenvolupadors fan servir, ja sigui per limitacions del llenguatge de programació o per facilitar la localització de les classes dins la biblioteca.

Vegeu el subapartat 5.1 d'aquest mòdul didàctic.

4. Amb supressió de l'herència múltiple per delegació, l'accés a la subclasse és més lent perquè cal anar a buscar atributs a les superclasses; en canvi, hi ha herència, si més no simple.

Vegeu el subapartat 5.3.2 d'aquest mòdul didàctic.

5. L'avantatge que representaria el fet de modificar l'exemple de la manera com s'explica a l'enunciat és que l'accés als atributs de la subclasse seria més ràpid, ja que no caldria ajuntar els propis i els heretats. D'altra banda, tindria l'inconvenient que qualsevol canvi en la definició dels atributs esmentats afectaria tots els locals i els processos que es fan amb aquests.

6. Amb els canvis esmentats, caldria definir aquestes composicions:

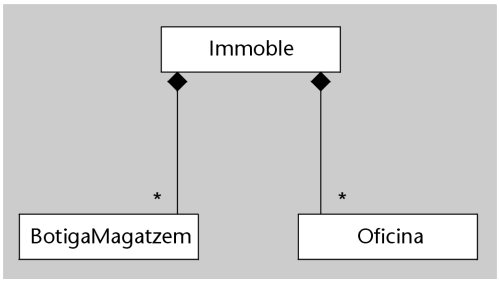

Figura 24

Seria necessària una operació per a afegir un component a un immoble, i probablement també una altra per a recórrer els components d'un immoble. Pel que fa a la persistència, caldria definir una taula addicional que tindria a cada fila el codi de local d'un immoble i el codi de local d'un local component d'aquest.

El cas d'ús que afegís components a un immoble es podria dissenyar de maneres bastant variades. Si se suposa que els components es creen com a locals independents i després s'enllacen com a components a un immoble ja existent, el cas d'ús demanaria el codi de local de l'immoble i el del component i cridaria l'operació d'addició d'un component ja esmentada. Aquesta operació afegiria una fila a la nova taula per mitjà del gestor de disc corresponent. Probablement, caldria comprovar que l'adreça, el NIF del propietari i la zona de l'immoble i del component fossin els mateixos a tots dos locals.

7. Perquè pogués ser avantatjós definir *AddicioEventualLlogater* com a subclasse d'*AddicioLlogater* caldria que el disseny dels casos d'ús 5 i 6 es fes de manera que les diferències entre ells dos fossin additives i no pas substitutives. Per exemple, que en comptes de presentar dues finestres diferents per a la introducció de dades, se'n presentés una amb les dades (atributs) comunes i una altra amb les dades específiques d'*EventualLlogater*; d'aquesta manera, l'operació *crear* seria la mateixa (és a dir, s'heretaria) a les dues classes de control, i a la subclasse hi hauria una operació addicional que presentaria la segona finestra i en faria les dades persistents.

8. Un disseny amb una sola operació és més eficient si s'han de modificar diversos atributs alhora, però és menys flexible, ja que cada vegada que s'afegeixi o suprimeixi un atribut caldrà canviar l'operació i, probablement, també els seus paràmetres, cosa que afectaria les classes que la fan servir.

Glossari

bastiment
Conjunt de classes que constitueix una aplicació genèrica i incompleta que cal complementar amb classes de l'usuari.

classe persistent
Classe que pot tenir objectes que s'hagin de fer persistents.

classe temporal
Classe que no és persistent, és a dir, els objectes de la qual no es poden fer persistents.

component
Conjunt de classes que col·laboren per a dur a terme una funció concreta. Exteriorment es veu com una classe que implementa una interfície determinada.

estat d'un objecte
Conjunt dels valors dels atributs d'un objecte en un moment donat.

estat persistent d'un objecte
Estat de l'objecte que està gravat en un sistema d'emmagatzemament permanent. Pot ser només una part de l'estat de l'objecte, i en un moment donat pot ser diferent de l'estat de l'objecte a memòria.

framework
Vegeu *bastiment*.

herència
Característica per la qual totes les subclasses tenen almenys tots els atributs i operacions de cadascuna de les seves superclasses.

objecte persistent
Objecte que no s'ha de destruir quan acaba el procés que l'ha creat perquè l'ha d'utilitzar algun procés posterior. Cal gravar-ne l'estat en un fitxer permanent o base de dades.

operació abstracta
Operació d'una superclasse que no està implementada en aquesta, sinó solament a les seves subclasses.

patró
Idea de disseny provada extensament i documentada.

Bibliografia

Booch, G.; Rumbaugh, J.; Jacobson, I. (1999). *UML. El lenguaje de modelado unificado. Guía del Usuario*. Addison-Wesley.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. (1996). *A System of Patterns. Pattern-Oriented Software Architecture*. Addison-Wesley.

Coplien, J.O.; Schmidt, D.C.; (1995). *Pattern Languages of Program Design*. Reading: Addison-Wesley.

Coplien, J.O.; Schmidt, D.C.; Vlissides, J.M.; Kerth, N. (1996). *Pattern Languages of Program Design 2*. Reading: Addison-Wesley.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995). *Design Patterns*. Reading: Addison-Wesley.

Harrison, N.; Foote, B.; Rohnert, H. (1999). *Pattern Languages of Program Design 4*. Addison-Wesley.

Jacobson, I. (1994). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.

Jacobson, I; Booch, G.; Rumbaugh, J. (2000). *El proceso unificado de desarrollo de software*. Madrid: Addison-Wesley.

Larman, C. (1998). *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*. Amsterdam: Prentice-Hall.

Martin, R.C.; Riehle, D.; Buschmann, F.; Vlissides, J.M. (1997). *Pattern Languages of Program Design 3 (Software Patterns Series)*. Addison-Wesley.

Meyer, B. (1997). *Object-Oriented Software Construction* (2a. ed.). Upper Saddle River: Prentice-Hall.

Richter, Ch. (1999). *Designing Flexible Object-Oriented Systems with UML*. Indianapolis: MacMillan.

Rosenberg, D.; Scott, K. (1999). *Use Case Driven Object Modeling with UML: A Practical Approach*. Reading: Addison-Wesley.

Rumbaugh, J.; Jacobson, I.; Booch, G. (2000). *UML. El lenguaje de modelado unificado. Manual de referencia*. Madrid: Addison-Wesley.

Weinschenk, S; Jamar, P.; Yeu, S.C. (1997). *GUI Design Essentials*. Nova York: Wiley & Sons.