

Enginyeria del software

Disseny I

**Disseny de sistemes orientats a objectes
amb notació UML**

Cristina Gómez - Enric Mayol
Antoni Olivé - Ernest Teniente

Enginyeria del software

Disseny I

Disseny de sistemes orientats a objectes
amb notació UML

Primera edició: setembre de 1999
Reimpressió: setembre de 2000
Segona edició: febrer de 2001

Aquest llibre s'ha publicat amb la col·laboració
del Comissionat per a Universitats i Recerca
i del Departament de Cultura de la Generalitat de Catalunya.

En col·laboració amb el Servei de Llengües i Terminologia de la UPC.

Disseny de la coberta: Manuel Andreu

© Els autors, 1999

© Edicions UPC, 1999
Edicions de la Universitat Politècnica de Catalunya, SL
Jordi Girona Salgado 31, 08034 Barcelona
Tel. 93 401 68 83 Fax 93 401 58 85
Edicions Virtuals: www.edicionsupc.es
e-mail: edupc@sg.upc.es

Producció: CPET (Centre de Publicacions del Campus Nord)
La Cup. Gran Capità s/n, 08034 Barcelona

Dipòsit legal: B-2.615-2001
ISBN obra completa: 84-8301-437-8
ISBN: 84-8301-460-2

Són rigorosament prohibides, sense l'autorització escrita dels titulars del copyright, sota les sancions establertes a la llei, la reproducció total o parcial d'aquesta obra per qualsevol procediment, inclosos la reprografia i el tractament informàtic, i la distribució d'exemplars mitjançant lloguer o préstec públics.

Índex

1 Introducció

1.1 Introducció al disseny de software	9
1.2 Introducció a l'arquitectura del software.....	11
1.3 Introducció als patrons de disseny	17
1.4 Patró arquitectònic en capes	25
1.5 Arquitectura lògica i física dels Sistemes d'Informació	33

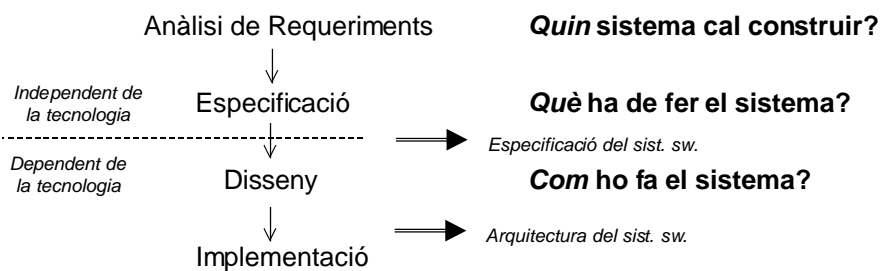
2 Disseny orientat a objectes en UML

2.1 Introducció al disseny orientat a objectes en UML.....	43
2.2 Conceptes d'UML per al disseny	49
2.3 Disseny de la Capa de Domini	61
2.3.1 – Especificació d'un exemple: gestió d'un club de tennis	63
2.3.2 – Normalització de l'esquema conceptual d'especificació	69
2.3.3 – Aplicació de patrons de disseny	75
2.3.3.1 – Patró Iterador	75
2.3.3.2 – Patró Controlador	81
2.3.3.3 – Assignació de Responsabilitats a Objectes	87
2.3.3.4 – Patró Estat	99
2.3.3.5 – Patró Mètode Plantilla	111
2.3.4 – Exemple de disseny de la capa de domini	119
2.4 Disseny de la Capa de Presentació.....	129
2.4.1 - Introducció.....	129
2.4.2 – Patró Model – Vista – Controlador	137
2.5 Disseny de la Capa de Gestió de Dades: persistència en BDOO	147

Introducció al disseny de software

- Etapes del desenvolupament de software
- Disseny de software
- Bibliografia

Etapes del desenvolupament de software



Disseny de software

- **Disseny de software** és l'activitat d'aplicar diferents tècniques i principis amb el propòsit de definir un sistema amb el suficient detall per permetre la seva construcció física (implementació).
- Punt de partida:
 - Resultat de l'especificació: (Què ha de fer el sistema?)
 - . Especificació de dades (Model de Dades)
 - . Especificació de processos (Models Funcional i de Comportament)
 - . Especificació de la interacció amb l'usuari (Model de la Interfície)
 - Tecnologia (Amb quins recursos)
 - . Recursos hardware i software disponibles
 - . Requeriments no funcionals del sistema
- Resultat del disseny: (Com ho fa el sistema?)
 - Estructura interna del sistema software (Arquitectura del Sistema Software)
 - Disseny de les Dades
 - Disseny de la Interfície
 - Disseny dels Programes
- Procés del disseny:
 - Metodologies de disseny
 - Adaptació de solucions genèriques a problemes de disseny (Patrons de disseny)

Bibliografia

- *Ingeniería del software*
Un enfoque práctico
R. G. Pressman
McGraw Hill, 1998. (Cuarta Edición)
Caps. 11 i 13.

Introducció a l'arquitectura del software

- Arquitectura del software
- Components
- Vistes
- Propietats de l'arquitectura
- Principis de disseny
- Relació entre propietats i principis de disseny
- Determinació de l'arquitectura
- Bibliografia

Arquitectura del software

L'**arquitectura del software** és una descripció dels subsistemes i components (computacionals) d'un sistema software, i les relacions entre ells.

- La determinació de l'arquitectura del software consisteix en la presa de decisions respecte a:
 - L'organització del sistema software
 - La selecció dels elements estructurals i les seves interfícies
 - Comportament d'aquests elements estructurals
 - La possible composició dels elements estructurals en subsistemes més grans
 - L'estil que guia aquesta organització

tenint en compte les propietats (requeriments no funcionals) del sistema software que es volen assolir:

- Eficiència, canviabilitat, usabilitat, fiabilitat, ...

- L'arquitectura del software és el resultat de l'activitat de disseny arquitectònic del software.

Components

- Un **component** és una part física i reemplaçable d'un sistema que :
 - és conforme a, i
 - proporciona una realitzaciód'un conjunt d'interfícies especificades contractualment.



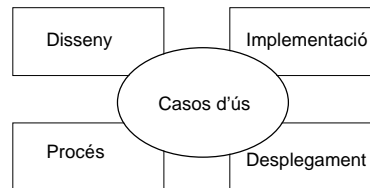
- Un component és una part encapsulada d'un sistema software
- Els components són els blocs constitutius de l'arquitectura del sistema
- Els components només tenen dependències contextuais explícites
- A nivell de llenguatge de programació, els components es poden representar com a:
 - mòduls
 - classes d'objectes
 - un conjunt de funcions relacionades
 - etc.

Relacions entre components

- Una **relació** denota una connexió entre components.
- Les relacions poden ser:
 - Estàtiques
 - Tenen a veure amb la col·locació dels components en l'arquitectura
 - Es veuen directament en el codi
 - Exemples: Crides a procediments, accés a variables compartides
 - Dinàmiques
 - Tracten de les connexions temporals i les interaccions dinàmiques entre els components
 - No són visibles fàcilment en el codi
 - Exemples: Protocols client-servidor, protocols d'accés a bases de dades

Vistes

- Els subsistemes i components s'especifiquen normalment en diferents vistes, per mostrar diferents propietats rellevants (req. func. i no func) del sistema software.
- Una **vista**:
 - representa un aspecte parcial de l'arquitectura,
 - mostra propietats específiques



Casos d'ús: Com l'usuari o analista veu el funcionament del sistema (cas d'ús, interacció)

Disseny: Descriu la solució software (classes, interacció entre objectes)

Implementació: Fitxers i components del sistema implementat (llibries, executables)

Procés: Mecanismes de concurrència i sincronització dels processos del sistema

Desplegament: Nodes que componen la topologia hardware del sistema instal·lat

Propietats de l'arquitectura

- **Canviable:**
 - **Extensible:** Noves funcionalitats o millores dels components
 - **Portable:** Canvis plataforma hardware, sistemes operatius, llenguatges
 - **Mantenible:** Detecció i reparació d'errors
 - **Reestructurable:** Reorganització de components
- **Interoperable**
 - Capacitat de dues o més entitats software d'intercanviar funcionalitat i dades
- **Eficient**
 - Temps de resposta, rendiment, ús de recursos
- **Fiable:**
 - **Tolerància a fallades:** Pèrdua de connexió i recuperació posterior
 - **Robustesa:** Protecció contra l'ús incorrecte i el tractament d'errors inesperats
- **Provable**
 - Facilitar les proves del sistema
- **Reusable:** Assolir el que es vol amb l'ajuda del que es té
 - Reusar software ja existent
 - Fer software per a ser reusat
- **Integritat conceptual:** Harmonia, simetria i predictibilitat

Principis de disseny

- Abstracció
- Encapsulament
- Ocultació d'informació
- Modularització
- Separació de preocupacions
- Acoblament i cohesió
- Sufficiència, completesa i primitivitat
- Separació de política i implementació
- Separació d'interfície i implementació
- Únic punt de referència
- Divideix-i-venceràs

Principis de disseny: Ocultació d'informació

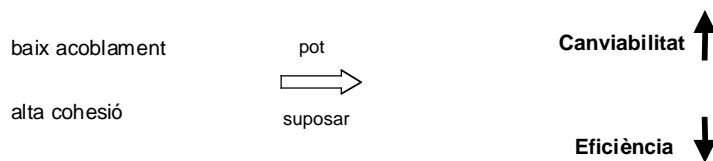
- Confinar (ocultar) en un únic mòdul (classe, etc.) un aspecte qualsevol del software que és possible (probable) que canviï, i establir interfícies intermòduls que siguin insensibles als canvis previstos.



- Si hi ha un canvi en l'aspecte *ocultat* (*secret*), quedarà limitat al mòdul corresponent, sense afectar la resta del sistema. Les interfícies no quedaran afectades.
- També es diu que el *secret* s'*oculta* en el mòdul.
- Aquest principi fou proposat per en D. Parnas, el 1972.

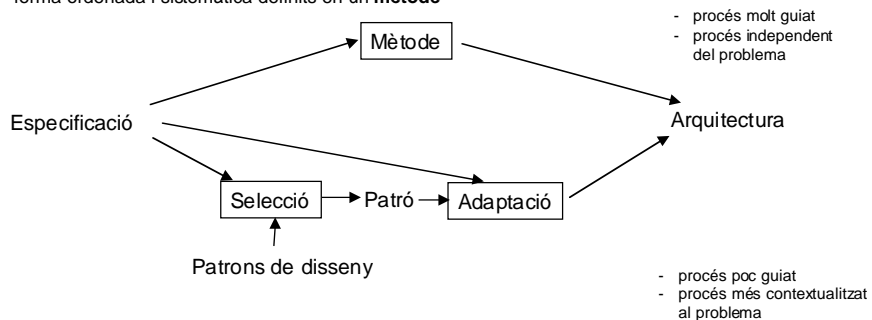
Relació entre propietats i principis de disseny

- L'aplicació d'un principi de disseny pot afectar l'assoliment d'una propietat de l'arquitectura del sistema
- Per exemple:



Determinació de l'arquitectura: Dues vies complementàries

Donats els requeriments del sistema degudament especificats, aplicar un conjunt passos genèrics de forma ordenada i sistemàtica definits en un **mètode**



Donats els requeriments del sistema degudament especificats,

- identificar els problemes específics a resoldre
- seleccionar solucions genèriques a problemes concrets (**patrons**)
- i adaptar-ne la seva aplicació tenint en compte el problema concret

Bibliografia

- *Pattern-oriented Software Architecture. A System of patterns*
F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal
John Wiley & Sons, 1996, Cap. 6.
- *The Unified Modeling Language User Guide*
G. Booch, J. Rumbaugh, I. Jacobson
Addison-Wesley, 1999, cap. 2, 25.
- *Component Software: Beyond Object-oriented programming*
C. Szyperski
Addison-Wesley, 1998, cap. 4.

Introducció als patrons de disseny

- Patrons de disseny: Definició.
- Patrons de disseny: Categories.
- Catàleg parcial de patrons arquitectònics.
- Catàleg parcial de patrons de disseny.
- Heterogeneïtat d'estils.
- Bibliografia

Patrons de disseny: definició

- **Context:**
 - Situació on es presenta el problema de disseny
- **Problema:**
 - Problema a resoldre
 - Forces: Aspectes que s'han de considerar en la solució
- **Solució:**
 - Esquema de solució del problema, que equilibra les forces.
 - Dos aspectes:
 - . Estàtic
 - . Dinàmic

Patrons de disseny: Categories

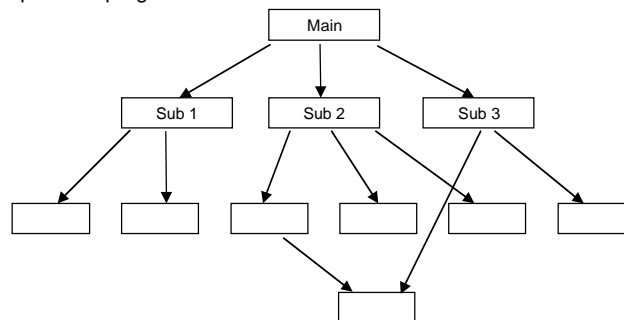
- **Patró arquitectònic:**
 - Un patró arquitectònic expressa un esquema d'organització estructural fonamental per a sistemes software.
 - Proporciona un conjunt de subsistemes predefinitos, especifica les seves responsabilitats, i inclou regles i guies per organitzar les relacions entre ells.
- **Patró de disseny:**
 - Un cop determinat el patró arquitectònic, un patró de disseny dóna un esquema per refinar els seus subsistemes o components, o les relacions entre ells.
 - Descriu l'estructura :
 - + d'una solució a un problema que apareix repetidament
 - + de components que es comuniquen entre ells
 - Resol un problema de disseny general en un context particular
- **Modisme:**
 - Usats en la fase d'implementació per a transformar una arquitectura en un programa escrit en un llenguatge específic.
 - Descriu l'estructura d'una solució depenent del llenguatge de programació

Catàleg parcial de patrons (o estils) arquitectònics

- **Crida i retorn:**
 - estils arquitectònics més usats, tradicionalment en grans sistemes software
 - . Programa principal i subrutina
 - . Orientació a objectes
 - . En capes
- **Centrat en dades:**
 - per sistemes que requereixen accés a dades compartides
 - . Repositori
 - . Pissarra
- **Flux de dades:**
 - per sistemes que donades unes dades d'entrada realitzen una sèrie de transformacions per obtenir unes dades de sortida
 - . Batch seqüencial
 - . Tubs i filtres
- **Model - Vista - Controlador:** per sistemes interactius

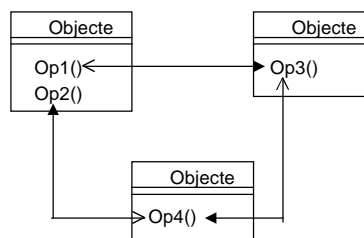
Crida i retorn: programa principal i subrutina

- Descomposició jeràrquica d'un programa principal en subrutines
- Una subrutina (component) al rebre el control (i dades) d'un dels seus pares, el passa cap als seus fills, el retorn del control es fa en sentit contrari (de fills a pares).
- Facilita la canviabilitat
- Estil típic de la programació tradicional



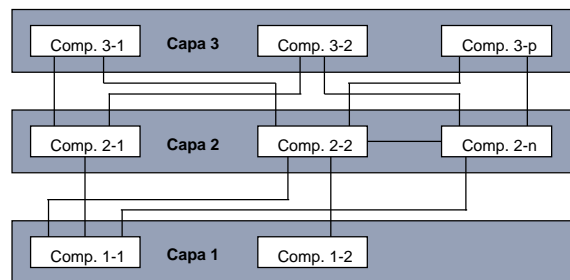
Crida i retorn: Orientació a objectes

- Cada component agrupa (encapsula) les dades i els mecanismes per a manipular-les.
- La comunicació entre components es realitza mitjançant la invocació de serveis oferts pels component (operacions).
- Facilita la canviabilitat i reusabilitat



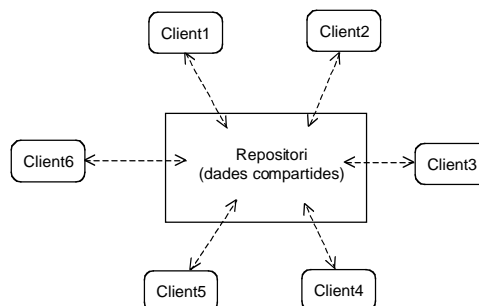
Crida i retorn: En capes

- Els components s'agrupen en capes.
- La comunicació solament pot ser entre components de la mateixa capa, o entre components de capes contigües.
- Facilita la canviabilitat i portabilitat.



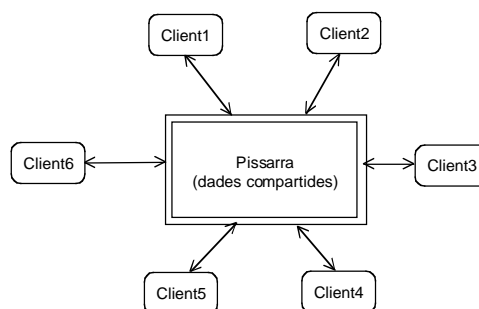
Centrat en dades: Repositori

- Les dades compartides per diferents clients estan en un mateix lloc
- Els clients es comuniquen amb el repositori per accedir i actualitzar les dades
- El Repositori és un medi d'emmagatzemament passiu



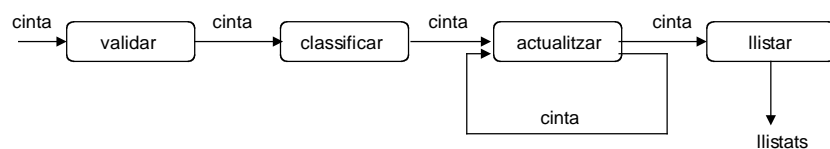
Centrat en dades: Pissarra

- Les dades compartides per diferents clients estan en un mateix lloc
- Els clients es comuniquen amb la pissarra per accedir i actualitzar les dades i la pissarra notifica als clients quan hi ha hagut canvis en les dades que els hi interessin
- La pissarra és un medi d'emmagatzemament actiu



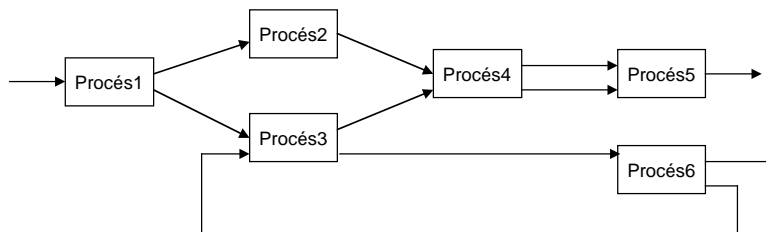
Flux de dades: Batch seqüencial

- Descomposició del sistema en una seqüència de programes independents
- Un programa no pot començar la seva execució fins que els precedents han finalitzat completament
- Les dades es transmeten entre components en la seva totalitat
- Facilita canviabilitat i reusabilitat
- Orientat a sistemes que s'executaran de forma diferida (Batch)



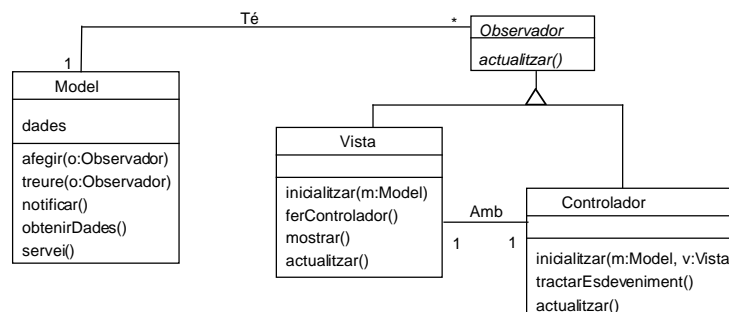
Tubs i filtres

- Descomposició del sistema en una seqüència de processos que realitzen modificacions incrementals de les dades
- Un procés pot començar la seva execució en el moment que rebí alguna dada d'entrada
- Els tubs (processos) no tenen informació d'estat
- Facilita canviabilitat i reusabilitat
- Estil arquitectònic típic dels sistemes operatius UNIX



Model-Vista-Controlador

- Descomposició del sistema software en:
 - Model: encarregat de la implementació de les funcionalitats i dades del sistema
 - La interfície amb l'usuari amb:
 - . Vista: encarregada de gestionar com es mostra la informació a l'usuari
 - . Controlador: encarregat de gestionar la interacció amb l'usuari
- Facilita canviabilitat i portabilitat



Heterogenïtat d'estils arquitectònics

- Sovint, els sistemes combinen dos o més estils:
 - s'anomenen sistemes heterogenis.
- Hi ha tres menes d'heterogeneïtat:
 - Locacional
 - . Diferents parts del sistema poden estar fetes en estils diferents
 - . Ex.: Algunes de les branques d'un sistema que segueix el patró programa principal i subrutina comparteixen una pissarra i altres branques no.
 - Jeràrquica
 - . La descomposició d'un component d'un estil es pot fer en un altre estil
 - . Ex.: descomposició en capes i orientació a objectes dins cada capa
 - Simultània
 - . Un mateix sistema es pot interpretar alternativament per més d'un estil
 - . Ex.: Un mateix sistema es pot veure estructurat seguint el patró tub-filtres (on cada procés espera una dada a l'entrada per processar) o vist com n subsistemes independents amb un ordre d'execució predeterminat.

Catàleg parcial de patrons de disseny

- **Creació:**
Patrons de disseny relatius a com es creen classes i objectes
- **Estructurals:**
Patrons de disseny relatius a com diferents classes i objectes es poden combinar per definir estructures més grans
- **Comportament:**
Patrons de disseny relatius a com les classes i objectes interaccionen i com s'els hi han d'assignar responsabilitats
 - . Assignació de responsabilitats (Controlador, Expert, Creador)
 - . Iterador
 - . Estat
 - . Mètode Plantilla
 - . Observador

Patrons que considerarem

- Patrons Arquitectònics:
 - Patró Arquitectònic en 3 Capes (Presentació, Domini i Gestió de Dades)
 - Patró Orientació a Objectes (dins de cada capa)
 - Patró Model-Vista-Controlador (en la Capa de Presentació)
- Patrons de Disseny:
 - Iterador
 - Controlador
 - Expert
 - Creador
 - Estat
 - Mètode Plantilla

Bibliografia

- *Software Architecture in Practice*
L. Bass; P. Clements; R. Kazman
Addison-Wesley, 1998, Cap. 5.
- *Software Architecture*
M. Shaw; D. Garlan
Prentice Hall, 1996, Cap. 2.
- *Pattern-oriented Software Architecture. A System of patterns*
F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal
John Wiley & Sons, 1996, Cap. 5 i 2
- *Design Patterns. Elements of Reusable Object-Oriented Software*
E. Gamma, R. Helm, R. Johnson, J. Vlissides
Addison-Wesley, 1995, Cap.1

Patró arquitectònic: Arquitectura en capes

- Context
- Problema
- Solució:
 - Estructura
 - Comportament
- Consideracions en la definició de l'arquitectura
- Beneficis i inconvenients
- Variants
- Bibliografia

Context

Un sistema gran que requereix ésser descomposat en grups de subtasques (components), tals que cada grup de subtasques està a un nivell determinat d'abstracció.

Problema

- Cal dissenyar un sistema amb la característica dominant d'incloure aspectes d'alt i baix nivell, on les tasques d'alt nivell es basen en les de baix nivell.
- El sistema requereix també una estructuració horitzontal, ortogonal a la vertical: tasques del mateix nivell d'abstracció independents entre elles
- L'especificació descriu les tasques a alt nivell, així com la plataforma.
- Es desitja portabilitat a d'altres plataformes.
- Les tasques d'alt nivell no es poden implementar utilitzant directament els serveis de la plataforma, a causa de la seva complexitat. Calen serveis intermediaris.

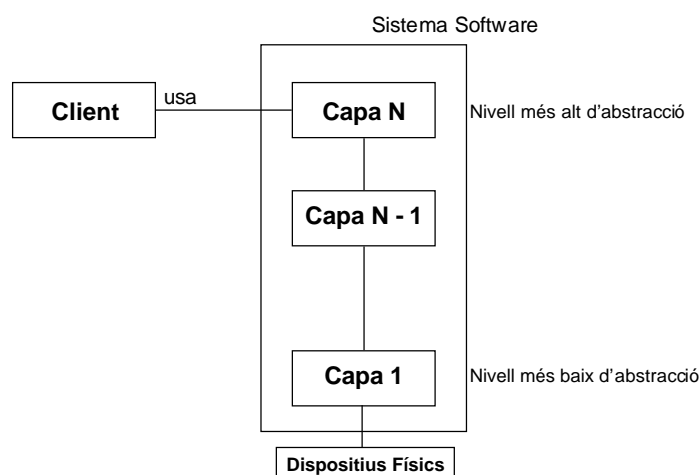
Problema: forces a equilibrar

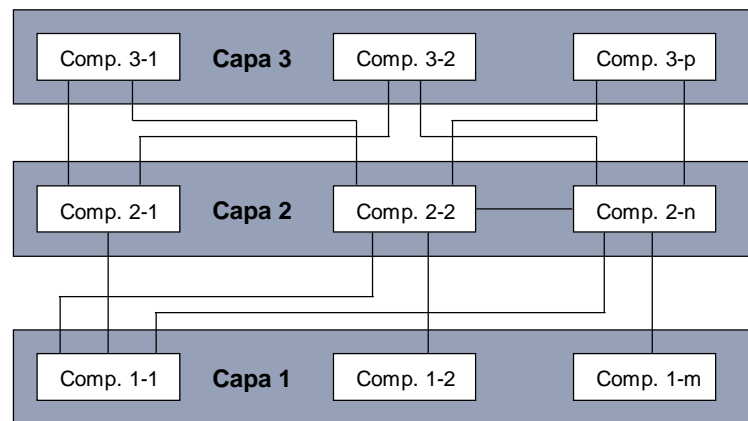
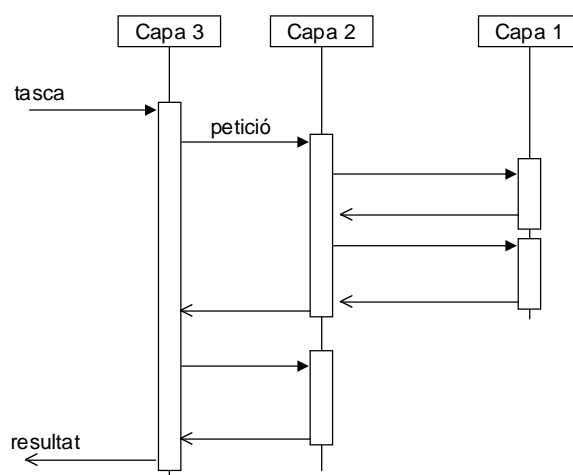
- Canvis en el codi no haurien de propagar-se en tot el sistema (Mantenible)
- Les interfícies dels components haurien de ser estables i clarament definides
- Els components s'haurien de poder reemplaçar per implementacions alternatives (Separació d'interfície i implementació)
- En el futur, pot ser necessari construir altres sistemes, amb els mateixos aspectes de baix nivell que aquest (Reusabilitat)
- Responsabilitats semblants s'haurien d'agrupar per ajudar a la comprensibilitat i mantenibilitat (Cohesió)
- No hi ha una granularitat "estàndard" de components
- Els components complexos necessiten una descomposició addicional

Solució

- Estructurar el sistema en un nombre apropiat de capes.
- Col·locar les capes verticalment.
- Tots els components d'una mateixa capa han de treballar al mateix nivell d'abstracció.
- Els serveis que proporciona la capa j utilitzen serveis proporcionats per la capa $j - 1$. Alhora, els serveis de la capa j poden dependre d'altres serveis en la mateixa capa.
- L'esquema més típic de comunicació consisteix en peticions de dalt a baix, i respostes a les peticions en la direcció oposada.

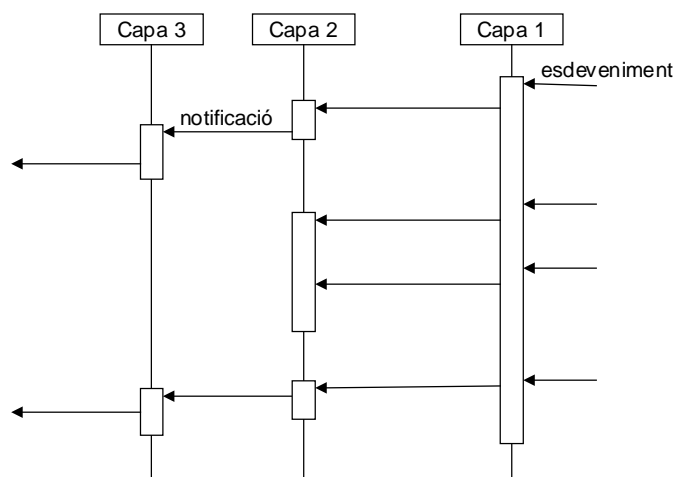
Solució: Estructura



Exemple: Arquitectura 3 capes**Solució: Comportament
(Comunicació de dalt a baix)**

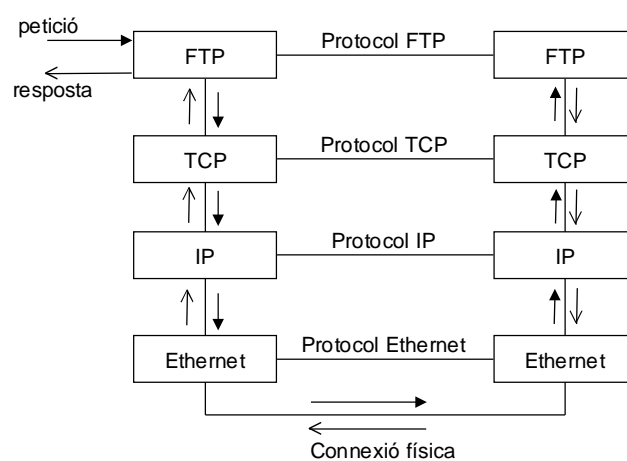
Un usuari realitza una petició d'una tasca a la capa superior

Solució: Comportament (Comunicació de baix a dalt)



Un dispositiu físic detecta l'ocurrència d'un esdeveniment a la cap inferior

Solució: Comportament (Comunicació bi-direccional)



Protocol de comunicació TCP/IP

Consideracions a la definició de l'arquitectura (I)

1. Definir el criteri d'abstracció per agrupar tasques en capes.
2. Determinar el nombre de nivells d'abstracció (capes):
 - Una capa per nivell d'abstracció.
 - Dos o més nivells en una capa.
 - Un nivell en dues o més capes.
3. Anomenar les capes i assignar tasques a cada una d'elles:
 - La capa superior correspon a la tasca del sistema.
 - Les altres capes donen suport a les capes superiors.
4. Especificar els serveis:
 - Cap component pot estar repartit en dues o més capes.
 - Pocs serveis en les capes inferiors.
5. Refinar l'arquitectura en capes.

Consideracions a la definició de l'arquitectura (II)

6. Especificar una interfície per cada capa. La capa j pot ésser:
 - Una caixa negra per a la $j+1$
 - Una caixa blanca (el seu contingut es visible per la capa $j+1$)
7. Estructurar les capes individualment
8. Especificar la comunicació entre capes adjacents:
 - Model "empenta": la informació es comunica en la petició del servei
 - Model "estirada": el servei demanat estira la informació de la capa superior
9. Desacoblar capes adjacents:
 - Comunicació descendent: Canvis en capa j poden ignorar presència capa $j+1$.
 - Comunicació ascendent: Capa j coneix quin servei específic demanar de la capa $j+1$ per notificar un esdeveniment rebut (Retrocrides)
10. Dissenyar una estratègia de tractament d'errors.
 - Tractar errors en la capa on es detecten
 - Tractar errors en les capes superiors

Beneficis i inconvenients

- Beneficis:
 - Reutilització de les capes en altres contexts (interfície i abstracció ben definides)
 - Suport a l'estandarització (nivells d'abstracció comunment acceptats)
 - Totes les dependències són locals (canvis locals)
 - Portabilitat
 - Facilitat de prova
 - Canviabilitat
- Inconvenients:
 - Menor eficiència
 - Feina innecessària o redundant
 - Dificultat en establir la granularitat i nombre de capes

Variants

- Arquitectura en capes relaxat
 - Una capa pot usar els serveis de totes les capes que estan per sota d'ella
 - Una capa pot ésser parcialment opaca
- Conseqüències
 - Possible guany en flexibilitat i eficiència
 - Possible pèrdua en la canviabilitat

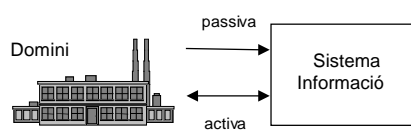
Bibliografia

- *Pattern-oriented Software Architecture. A System of patterns*
F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal
John Wiley & Sons, 1996. Pàgines 31-51

Arquitectura lògica i física de sistemes d'informació

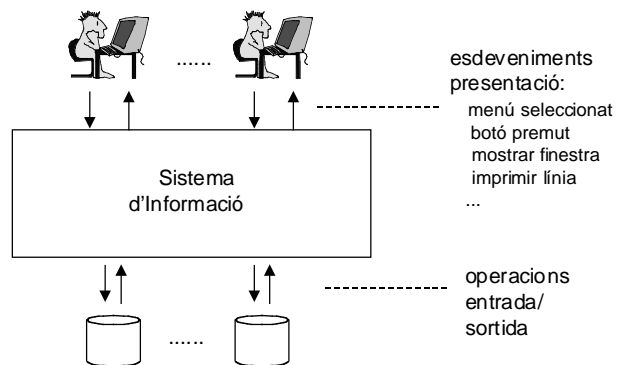
- Funcions d'un sistema d'informació.
- Arquitectura lògica d'un sistema d'informació.
- Arquitectura física no distribuïda:
 - Una capa.
 - Dues capes.
 - Tres capes.
- Bibliografia

Funcions d'un sistema d'informació



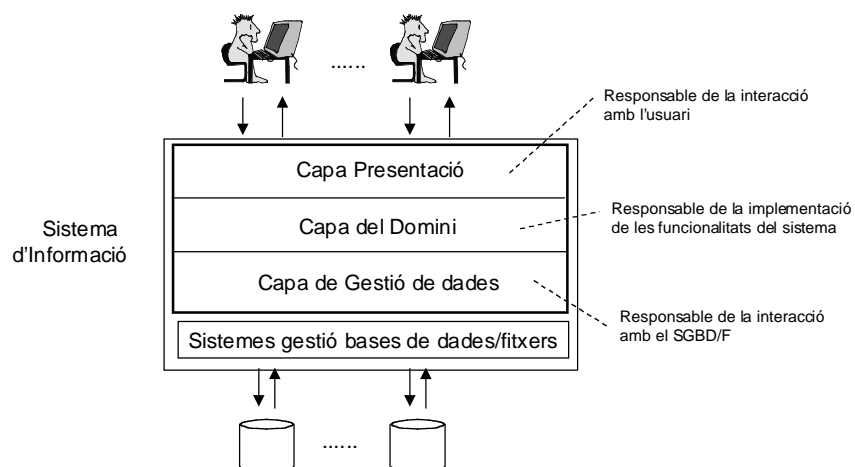
- Passiva:
 - Mantenir una representació consistent de l'estat del domini:
 - . Capturar els esdeveniments que ocorren al domini
 - . Actualitzar l'estat del sistema d'informació com a conseqüència d'aquests esdeveniments
 - . Assegurar la consistència de la representació
- Activa:
 - Respondre a consultes sobre l'estat del domini.
 - Produir reaccions quan es donen certes condicions predefinides.

Arquitectura lògica d'un sistema d'informació: Context

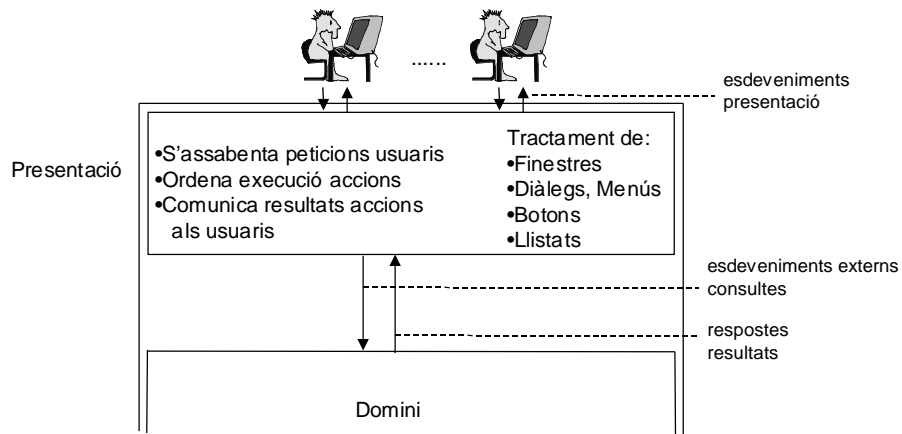


Arquitectura lògica d'un sistema d'informació: Visió general

Aplicació del patró "Arquitectura en 3 Capes" a un Sistema d'Informació

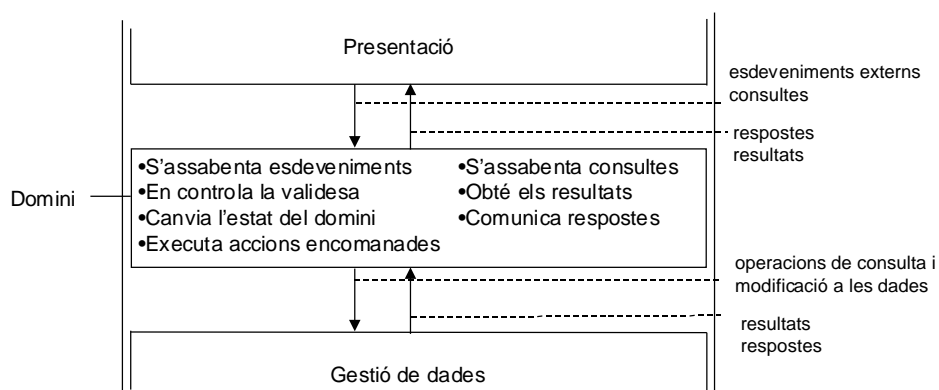


Arquitectura lògica d'un sistema d'informació: Capa de Presentació



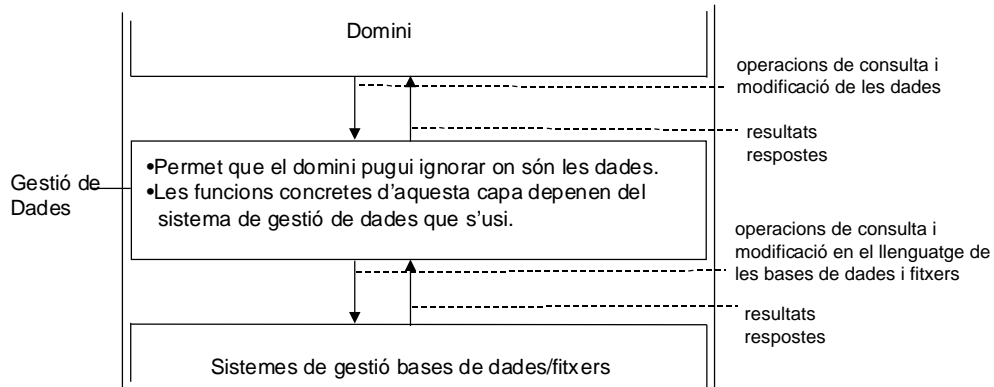
La Capa del Presentació coneix com presentar les dades a l'usuari, però ignora quines transformacions cal fer per donar resposta a les peticions de l'usuari

Arquitectura lògica d'un sistema d'informació: Capa del Domini



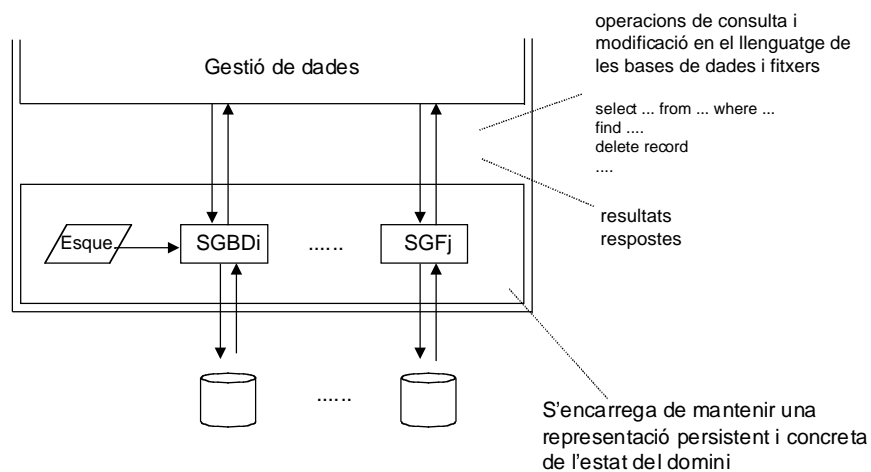
La Capa del Domini coneix com satisfer les peticions de l'usuari, però ignora on es guarden les dades i com es presenten a l'usuari

Arquitectura lògica d'un sistema d'informació: Capa Gestió de Dades

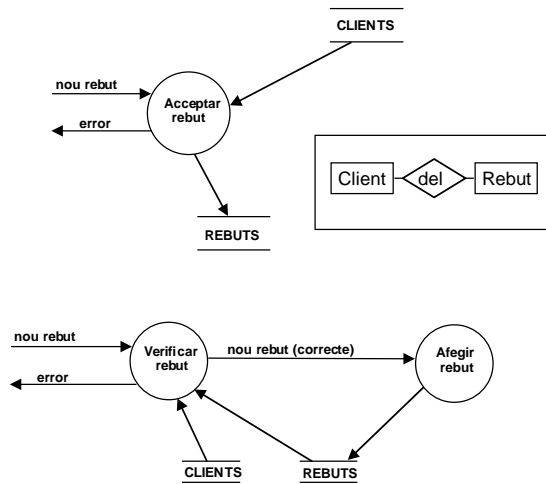


La capa del Gestió de Dades coneix on i com estan emmagatzemades les dades, però desconeix com tractar-les

Arquitectura lògica d'un sistema d'informació: Els sistemes de gestió de bases de dades/fitxers



Capa Gestió de Dades: Exemple en arquitectura basada en transaccions/bases de dades



Procés de dades discret Verificar rebut

Estímul: Flux de dades nou rebut

* El flux nou rebut ja està accessible

```

if client does not exist
    error := 'No conec el client'
    issue (error) * operació estàndard YSM
else if rebut exists
    error := 'Aquest rebut ja existeix'
    issue (error)
else
    issue (nou rebut (correcte))
end
  
```

Procés de dades discret Afegir rebut

Estímul: nou rebut (correcte)

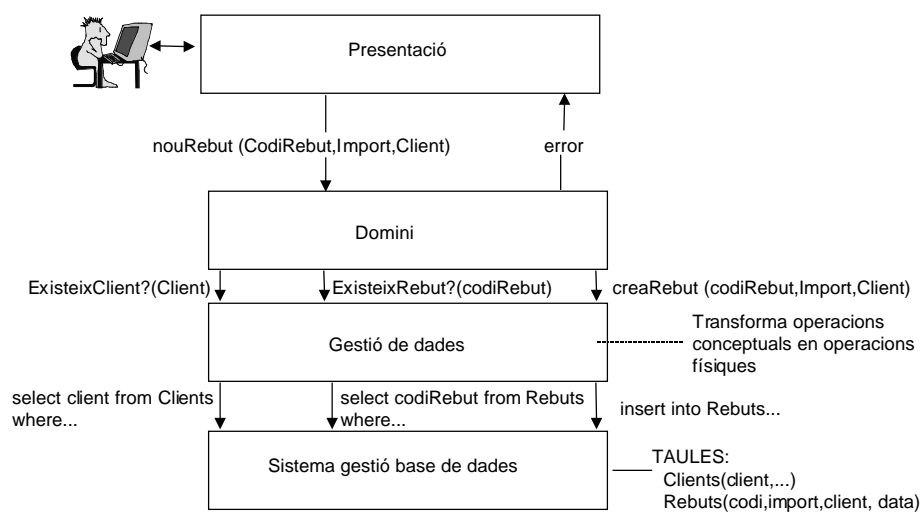
* Tenim nou rebut disponible

```

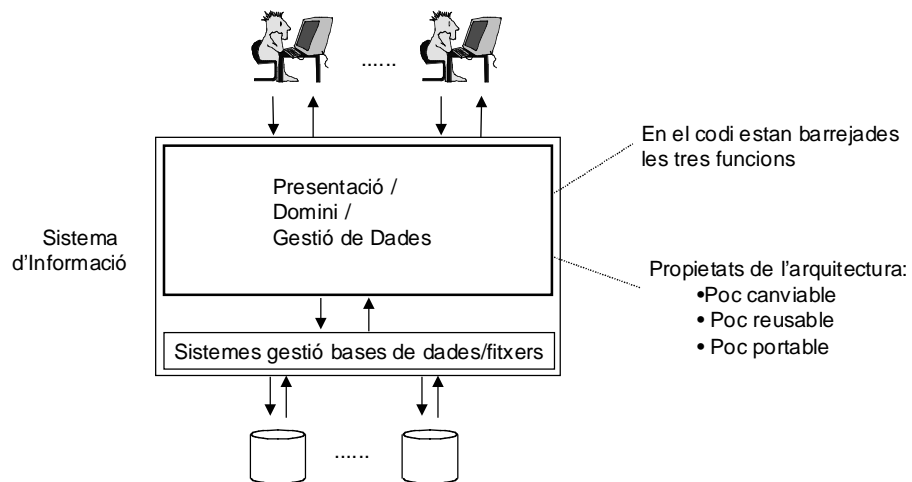
create REBUT with:
    codi rebut := nou rebut.codi rebut
    import := nou rebut.import
    data := today's date ()
    * today's date() és una operació estàndard del YSM
end;

* tenim REBUT lligat per la creació anterior i
* CLIENT lligat pel flux d'entrada
create REBUT del CLIENT;
end.
  
```

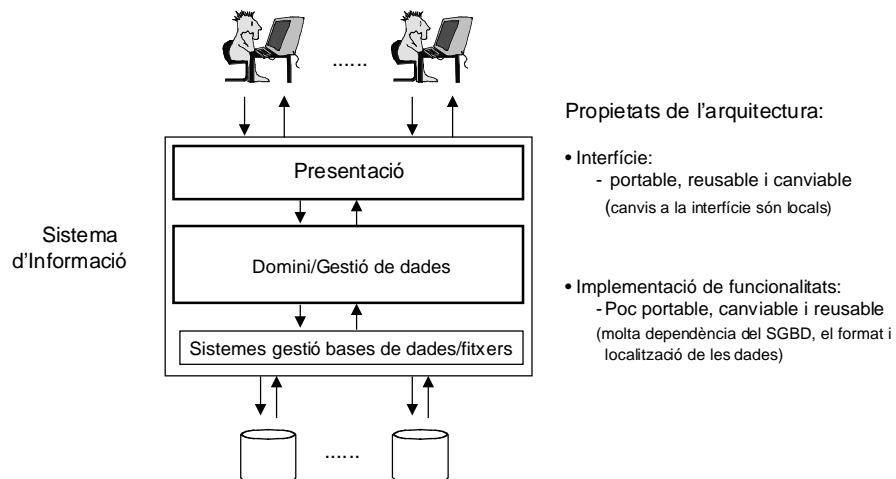
Capa Gestió de Dades: Exemple en arquitectura basada en transaccions/bases de dades



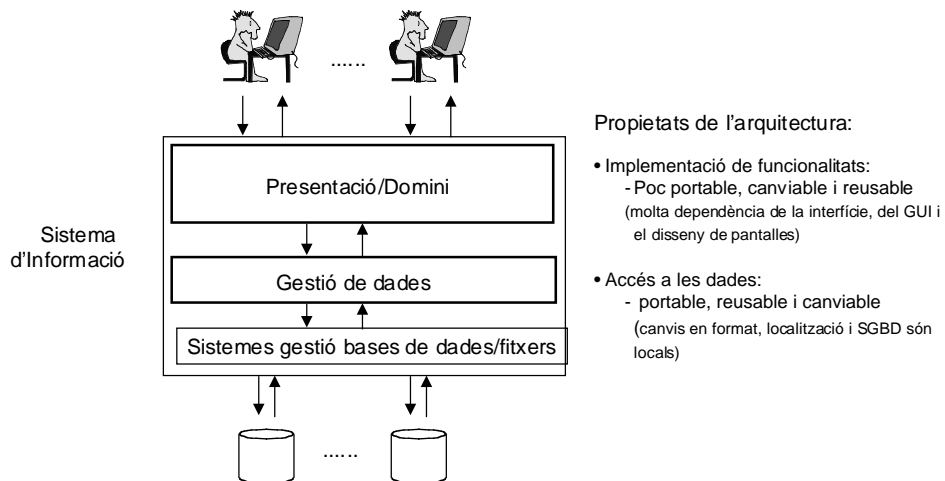
Arquitectura física no distribuïda: Una capa



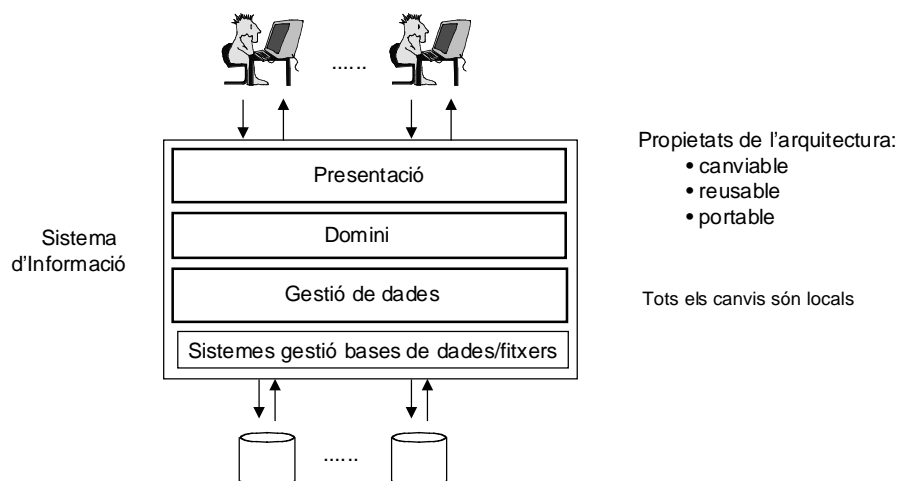
Arquitectura física no distribuïda: Dues capes (I)



Arquitectura física no distribuïda: Dues capes (II)



Arquitectura física no distribuïda: Tres capes



Bibliografia

- *Analysis Patterns*
Martin Fowler
Addison-Wesley, 1997, cap. 12.

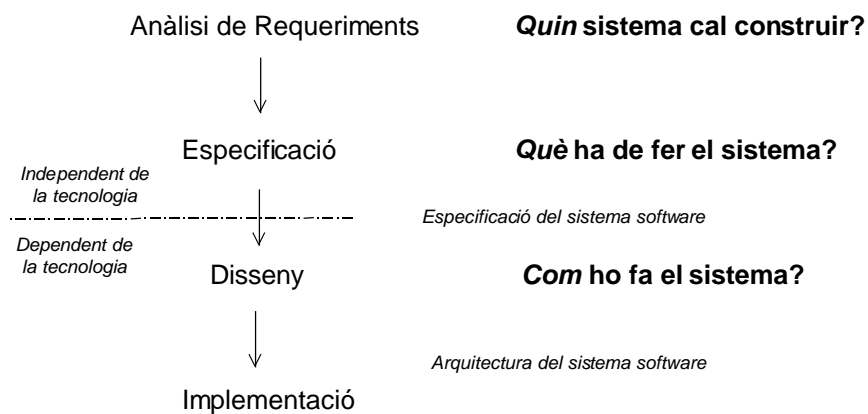
Disseny orientat a objectes en UML

- Introducció al disseny orientat a objectes en UML
- Conceptes d'UML per al disseny
- Disseny de la Capa de Domini
 - Especificació d'un exemple: gestió d'un club de tennis
 - Normalització de l'esquema conceptual d'especificació
 - Aplicació de patrons de disseny
 - Exemple de disseny de la capa de domini
- Disseny de la Capa de Presentació
- Disseny de la Capa de Gestió de Dades

Introducció al disseny orientat a objectes en UML

- Etapes del desenvolupament de software
- Determinació de l'arquitectura del software
- Especificació versus disseny
- Visió d'un sistema software en UML
- Punt de partida: especificació en UML
- Resultat a obtenir: disseny en UML
- Bibliografia

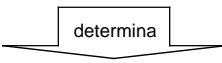
Etapes del desenvolupament de software



Determinació de l'arquitectura del software

Dependència tecnològica:

- Propietats que es volen assolir (requeriments no funcionals)
- Recursos tecnològics disponibles
 - família de llenguatges de programació
 - família de sistema gestor de bases de dades
 - etc.




determina

*L'arquitectura del sistema software i els patrons
(arquitectònics) que s'usaran per fer el disseny del sistema*

Determinació de l'arquitectura del software - ES:D1

El disseny que explicarem es basa en els supòsits següents:

- Propietats que es volen assolir: **canviabilitat** i **portabilitat**
- Recursos tecnològics disponibles
 - **llenguatge de programació orientat a objectes**
 - **base de dades orientada a objectes**



- Arquitectura en tres capes
- Orientació a objectes dins de cada capa

Especificació versus disseny

• En UML:

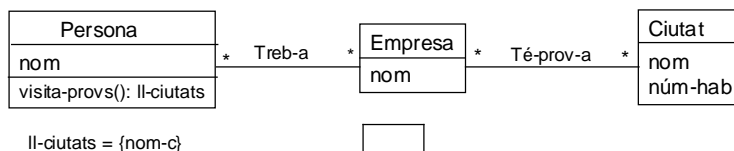
- Es fan servir els **mateixos models** al disseny que a l'especificació
- La **visió (enfocament)** dels models és molt **diferent** a ambdues etapes:
 - **Especificació**: els models defineixen els conceptes del món real.
(domini del problema)
 - **Disseny**: els models defineixen els conceptes que es desenvoluparan per proporcionar una solució a les necessitats del món real.
(domini de la solució)

Del domini del problema al de la solució (I)

- L'especificació no té en compte les propietats a assolir ni la tecnologia que s'usarà per implementar el sistema software.
- Durant el disseny, es poden haver de canviar els models d'especificació per tal d'assolir determinades propietats.

Exemple: propietat a assolir -> eficiència

Esquema conceptual d'especificació:



- **Problema**: operació *visita-provs* costosa perquè hi ha molts recorreguts
- **Solució**: afegir una associació derivada */visita* entre persona i ciutat
- *Malauradament, aquesta associació podria no ser rellevant al domini del problema*

Del domini del problema al de la solució (II)

Possibles coses a fer, des del punt de vista del domini de la solució:

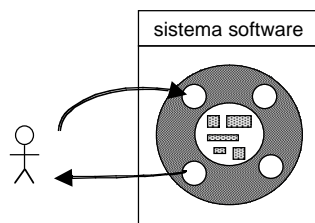
- afegir informació derivada (atributs i/o associacions)
- simplificar jerarquies de generalització / especialització (eliminar subclasses)
- afegir classes d'objectes redundants
- considerar classes abstractes
- etc.



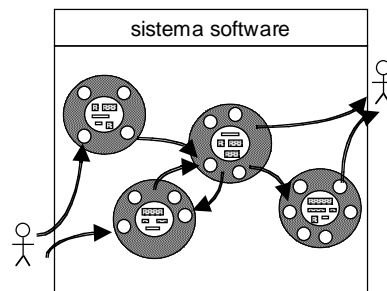
- *Aquest pas està poc formalitzat i poc explicat a la literatura actual.*
- *Les transformacions a aplicar depenen de molts factors externs: freqüència d'invocació de les operacions, cost d'execució, població de les classes, etc.*
- *Per tant, l'estudi d'aquestes transformacions queda fora de l'abast d'aquesta documentació.*

Visió d'un sistema software en UML

Especificació



Disseny



Especificació: el sistema software es veu com una sola classe d'objectes que engloba tota la informació i totes les operacions.

Disseny: cada classe d'objectes té les seves pròpies operacions de manipulació d'informació. Els objectes interactuen per satisfer les operacions del sistema.

Punt de partida: especificació en UML

- **Especificació:** *QUÈ* fa el sistema software?
- **Resultat de l'especificació en UML:**
 - Model de Casos d'Ús:
 - Quina interacció hi ha entre els actors i el sistema software?
 - Esquema Conceptual:
 - Quins són els conceptes rellevants del món real de referència ?
 - Diagrames de seqüència del sistema:
 - Quina resposta dona el sistema als esdeveniments externs? (quines operacions ha de tenir el sistema?)
 - Contractes de les operacions:
 - Què fan les operacions del sistema?

Resultat a assolir: disseny en UML

- **Disseny:** *COM* estructurarem el sistema perquè faci el que ha de fer?
⇒ *el disseny és una activitat iterativa i és difícil seqüencialitzar tot el que s'hi fa.*
- **Resultat del disseny en UML:**
 - Model de Casos d'Ús:
 - Defineix la interacció real, amb una interfície concreta.
 - Diagrama de classes de disseny:
 - Descriu les classes del software i les seves interfícies (operacions).
 - Diagrames de seqüència:
 - Defineixen la interacció entre les classes d'objectes per respondre un esdeveniment extern.
 - Contractes de les operacions:
 - Defineixen què fan les operacions de les classes d'objectes.

Bibliografia

- *Applying UML and Patterns*
An Introduction to Object-Oriented Analysis and Design
C. Larman
Prentice-Hall 1998
- *Object-Oriented Modeling and Design*
J.Rumbaugh, M.Balaha, W.Premarlani, F.Eddy, W.Lorensen
Prentice-Hall, 1991
- *Enginyeria del software: Especificació*
D.Costal, M.R.Sancho, E.Teniente
Edicions UPC, 1999

Conceptes d'UML per al disseny

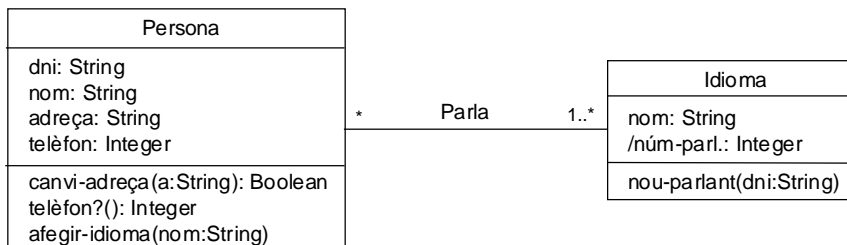
- Objecte i Classe d'objectes
- Atribut
- Associació
- Operació, Mètode i Contracte
- Agregació i Composició
- Generalització i Herència
- Polimorfisme
- Diagrames d'interacció
 - Diagrames de seqüència
- Invocació d'operacions en jerarquies
- Bibliografia

Objecte i classe d'objectes

- **Objecte:**
 - és una *manifestació concreta d'una abstracció*
 - és una *instància d'una classe*, que encapsula estat i comportament
 - té una *identitat pròpia*
 - el fa distingible d'altres objectes
 - la seva existència és independent dels valors de les seves propietats
- **Classe d'objectes:**
 - *descriu un conjunt d'objectes* que comparteixen atributs, associacions, operacions i mètodes.
 - *representa un concepte* dins el sistema que s'està modelitzant:
 - un concepte del món real (a l'especificació)
 - un concepte software equipat amb una implementació (a disseny)

Atribut, associació, operació i mètode

- **Atribut:** propietat compartida pels objectes d'una classe
- **Associació:** representa relacions entre dos o més objectes
- **Operació:**
 - transformació o consulta que poden executar els objectes d'una classe
 - té una signatura (nom i paràmetres de l'operació) i és invocada per altres objectes
- **Mètode:** implementació concreta d'una operació dins d'una classe



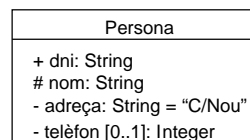
Atributs: sintaxi completa

[visibilitat] nom [multiplicitat] [:tipus] [=valor-inicial] [{propietats}]

Univaluat (per defecte)
Multivaluat: [1..*]
Admet valors nuls: [0..1]
 etc.

changeable
addOnly
frozen

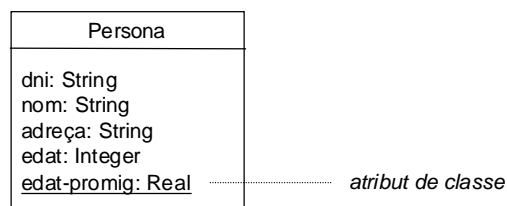
Public (+): qualsevol classe que pot veure la classe, veu l'atribut
Protected (#): només la pròpia classe i els seus descendents veuen l'atribut
Private (-): només la pròpia classe veu l'atribut



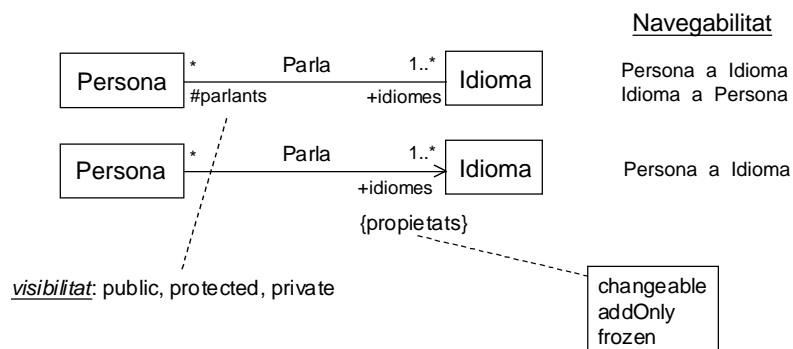
*Suposarem que els atributs són privats,
 a no ser que especifiquem una altra
 visibilitat*

Atributs: àmbit

- **Atribut d'instància:**
 - representa una propietat aplicable a tots els objectes d'una classe
 - cada objecte pot tenir un valor diferent d'aquest atribut
- **Atribut de classe:**
 - propietat aplicable a la classe d'objectes com a tal
 - tots els objectes d'una classe comparteixen el valor d'aquest atribut



Associacions: sintaxi



Navegabilitat:

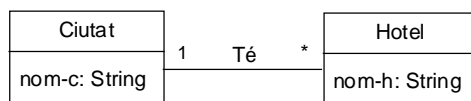
- indica si és possible o no travessar una associació binària d'una classe a una altra
- si hi ha navegabilitat, l'associació defineix un pseudoatribut (que correspon al rol d'aquell sentit de navegació) de la classe a partir de la qual es pot navegar
- suposarem que les associacions són privades, a no ser que s'indiqui el contrari

Associacions: qualificador

Qualificador:

- permet seleccionar un conjunt d'objectes relacionats amb un objecte qualificat mitjançant una associació.
- un objecte, conjuntament amb el valor del qualificador, determina un únic objecte (de vegades un conjunt d'objectes) de l'altre extrem de l'associació.
- és un índex en el recorregut de l'associació

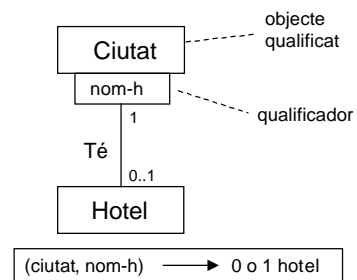
Associació no qualificada:



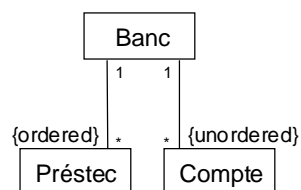
R.I. textuals:

- no hi pot haver dues ciutats amb idèntic nom-c
- una ciutat no pot tenir dos hotels amb nom-h

Associació qualificada:

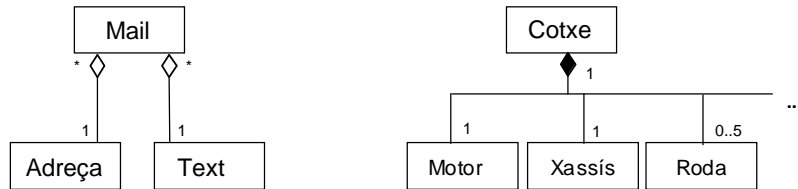


Associacions: ordenació



- És una propietat d'un conjunt d'objectes relacionats amb un altre objecte mitjançant una associació, que indica si el conjunt està o no ordenat (per posició).

Agregació i Composició



- L'agregació és una associació que relaciona el tot i les parts.
- Les parts poden pertanyer a més d'un agregat.
- A nivell d'objectes, no hi pot haver cicles en els camins d'agregació.
- La composició és una agregació amb una pertinença més forta entre el tot i les parts: si s'esborra el tot, també cal esborrar les parts.
- En un moment determinat, les parts pertanyen com a màxim a una composició.

Operacions: sintaxi completa

Signatura d'una operació:

[visibilitat] nom [(llista-paràmetres)] [:tipus-retorn] [{propietats}]

Public (+): l'operació pot ser invocada des de qualsevol objecte
Protected (#): pot ser invocada pels objectes de la classe i els seus descendents
Private (-): només els objectes de la pròpia classe poden invocar l'operació

isQuery
sequential
guarded
concurrent

Paràmetres d'una operació:

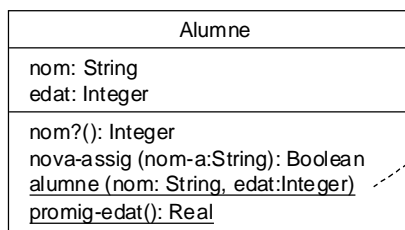
[direcció] nom : tipus [=valor-per-defecte]

in, out, inout

Suposarem que les operacions són públiques, si no s'indica el contrari

Operacions: àmbit

- **Operació d'instància:**
 - l'operació és invocada sobre objectes individuals
- **Operació de classe:**
 - l'operació s'aplica a la classe pròpiament dita
 - Exemple: operació constructora, que serveix per donar d'alta noves instàncies d'una classe.



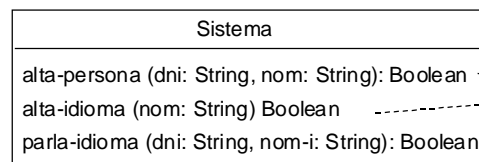
Operació constructora:

Suposarem que, si no s'indica el contrari, cada classe d'objectes té una operació constructora amb tants paràmetres com atributs té la classe.

Si es consideren altres constructores, caldrà definir-ne la seva signatura.

Operacions: contractes

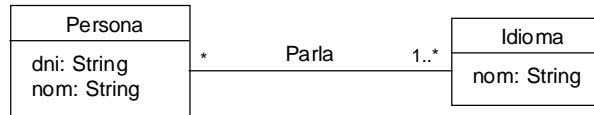
- **Contracte:** descriu l'efecte d'una operació en base a
 - canvis d'estat de la base d'informació
 - sortides que el sistema proporciona
- Els contractes garanteixen la **fiabilitat del software** mitjançant:
 - *precondicions*: condicions que estan garantides quan es crida l'operació
 - *postcondicions*: condicions que l'operació ha de garantir
- Tota operació té un contracte



Per cada operació es defineix un contracte

Contractes: exemple

- Operació:** parla-idioma (dni: String, nom-id: String)
- Classe retorn:** Booleà
- Semàntica:** Enregistrar que la persona dni parla l'idioma nom-id
- Precondicions:** Els paràmetres tenen valor
- Postcondicions:**
1. Si la persona amb dni no existeix, o l'idioma nom-id no existeix o la persona ja parla aquest idioma, aleshores operació invàlida i es retorna fals.
 2. En cas contrari, operació vàlida, es retorna cert i:
 - 2.1 - Es crea una nova instància de l'associació *Parla* entre la Persona i l'Idioma.

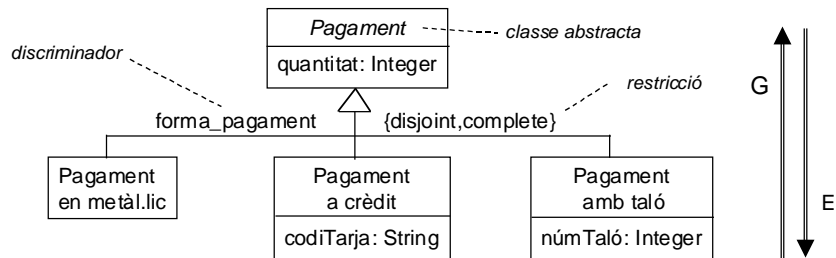


R.I. textuais:

- Claus de les classes no associatives: (Persona, dni); (Idioma,nom)

Generalització / Especialització

- **Generalització:**
 - relació taxonòmica entre una classe més general (*superclasse*) i una altra més específica (*subclasse*)
 - la *superclasse* descriu les característiques comunes a totes les subclasses
 - cada *subclasse* descriu un conjunt específic de característiques

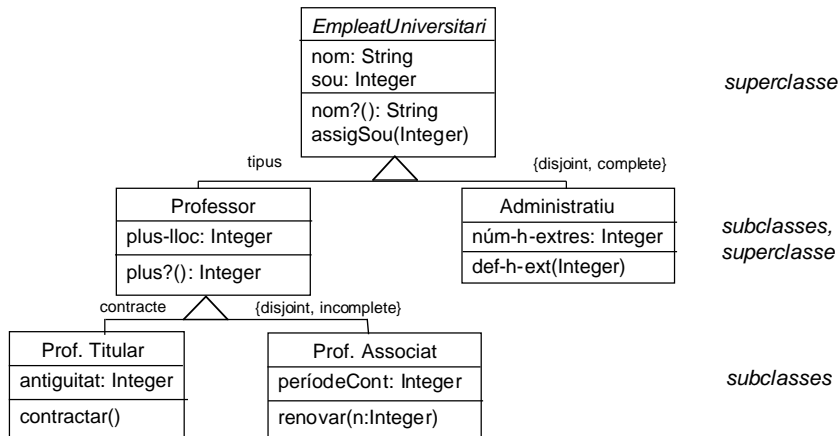


- **Classe abstracta:**
 - classe que no pot ser instanciada directament (no té objectes propis).

Herència

- Herència:**

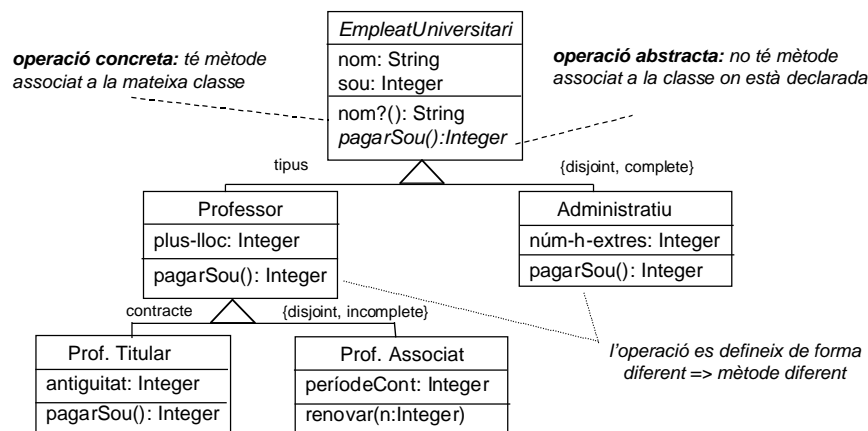
- en una generalització, les subclasses incorporen (hereten) l'estructura i el comportament definits a la superclasse.



Polimorfisme

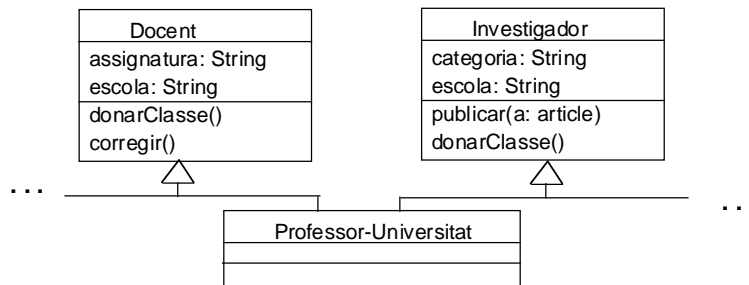
- Operació polimòrfica:**

- operació que s'aplica a diverses classes d'una jerarquia tal que la seva semàntica depèn de la subclasse concreta on s'aplica



Herència múltiple

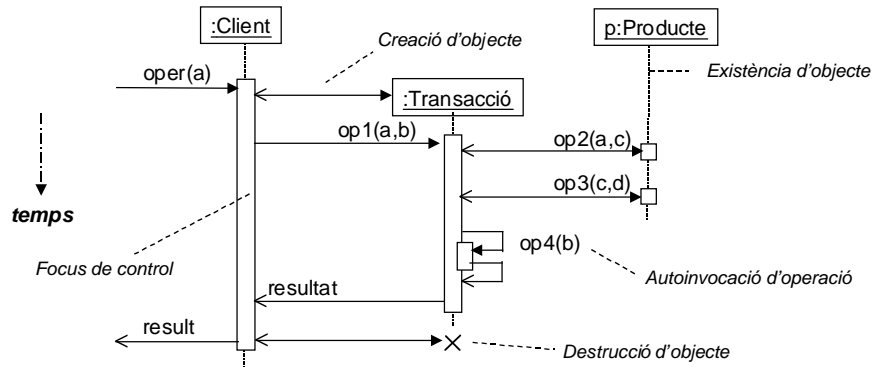
- Una classe pot ser subclasse directa d'un nombre arbitrari de superclasses
- Una classe pot **heretar** atributs i operacions **de diverses classes**.
- Es produeix un conflicte si una mateixa característica apareix a diverses superclasses.



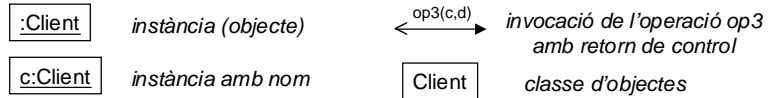
Diagrames d'interacció

- **Objectiu:** Mostrar les interaccions entre objectes:
 - Quan es produeix un esdeveniment extern.
 - Quan s'invoa una operació d'un objecte.
 - En un cas d'ús.
 - Etc.
- **Dos tipus de diagrama d'interacció** (semànticament equivalents):
 - Diagrames de seqüència.
 - Diagrames de col.laboració.

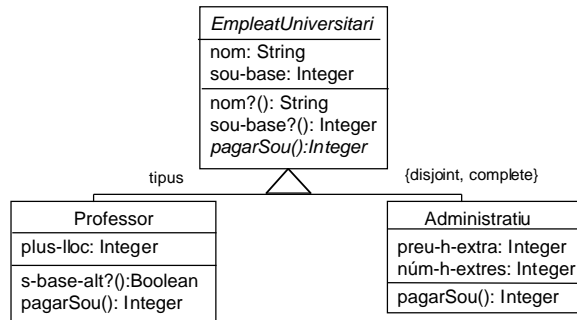
Diagrames de seqüència



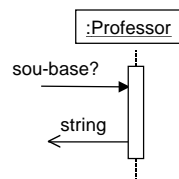
Notació:



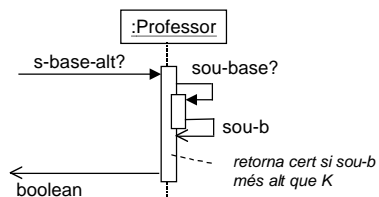
Invocació d'operacions en jerarquies (I)



sou-base? (invocada a Professor)

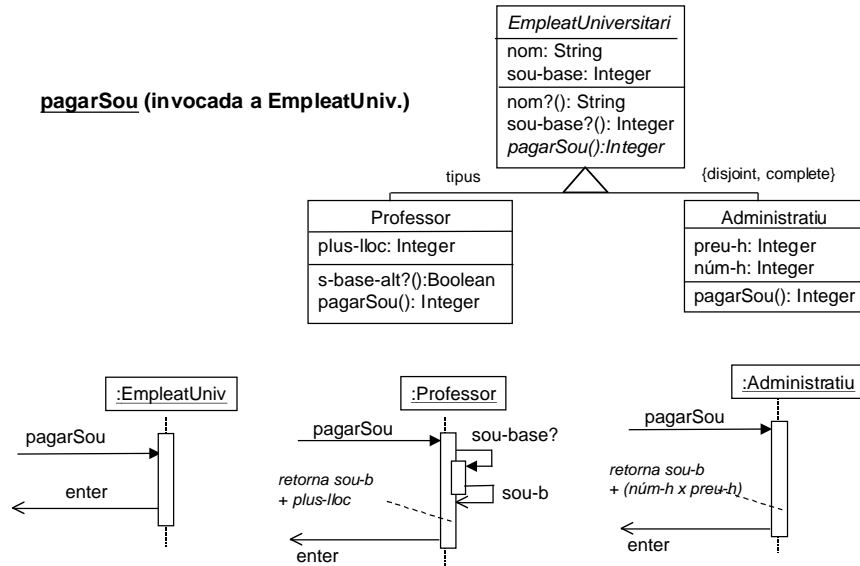


s-base-alt? (a Professor)



Invocació d'operacions en jerarquies (II)

pagarSou (invocada a EmpleatUniv.)



Bibliografia

- *The Unified Modeling Language Reference Manual*
J.Rumbaugh, I.Jacobson, G.Booch
Addison-Wesley 1999
- *Applying UML and Patterns*
An Introduction to Object-Oriented Analysis and Design
C. Larman
Prentice-Hall 1998
- *Construcción de Software Orientado a Objetos*
B.Meyer
Prentice-Hall, 2a Ed., 1998
- *The Unified Modeling Language User Guide*
G.Booch, J.Rumbaugh, I.Jacobson
Addison-Wesley 1999

Disseny de la Capa de Domini

- Especificació d'un **exemple**: *gestió d'un club de tennis*
- **Normalització** de l'esquema conceptual d'especificació
- Aplicació de **patrons de disseny**
 - Iterador
 - Controlador
 - Expert
 - Creador
 - Estat
 - Plantilla
- **Exemple de disseny** de la Capa de Domini

De moment, prescindirem del que fan la capa de Presentació i la capa de Gestió de Dades



Suposem que tot ho fa la Capa de Domini

Especificació d'un exemple: *gestió d'un club de tennis*

- Especificació d'un exemple: gestió d'un club de tennis
 - Model de Casos d'Ús
 - Esquema Conceptual
 - Diagrames de seqüència del sistema
 - Contractes de les operacions del sistema

Diagrama Casos d'ús: context del sistema

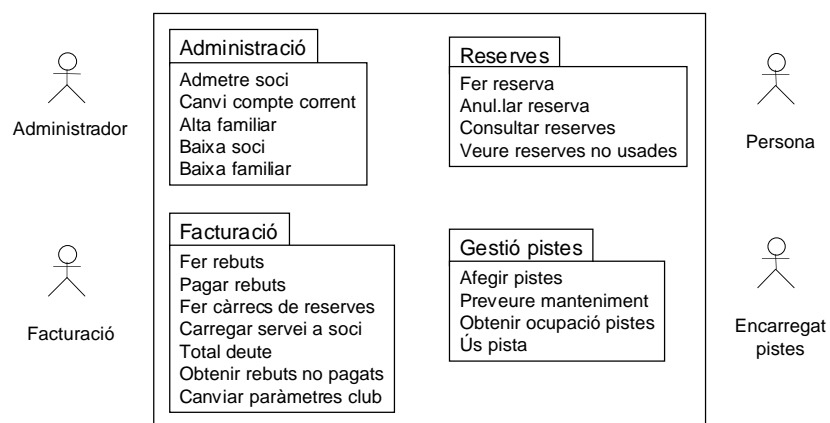
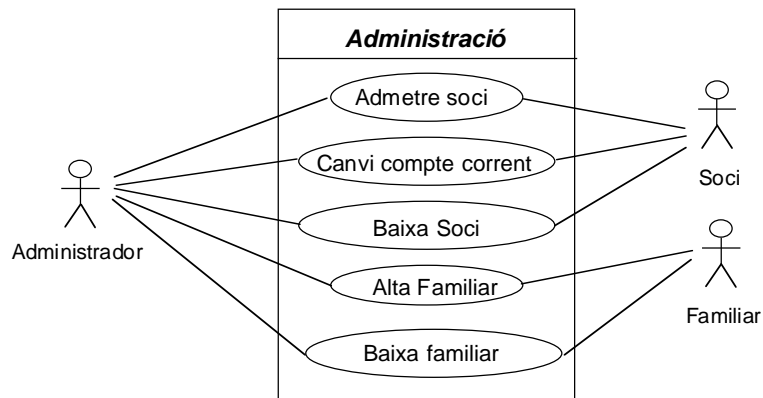


Diagrama de Casos d'Ús: package Administració



Descripció casos d'ús: Admetre soci

Cas d'Ús: Admetre soci

Actors: Soci (iniciador), Administrador

Propòsit: Donar d'alta un nou soci i alguns familiars seus

Curs típic d'esdeveniments

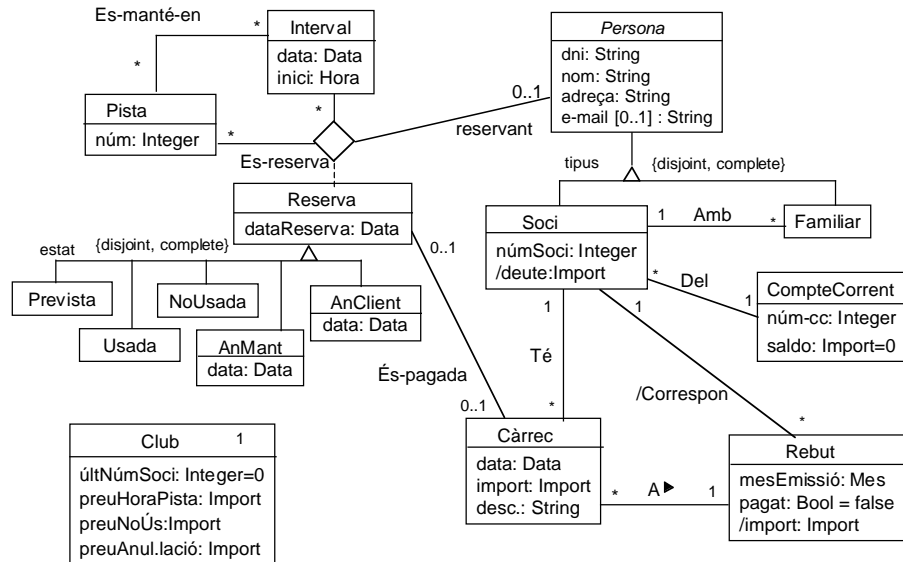
Accions dels Actors

1. El cas d'ús comença quan una persona es vol donar d'alta com a soci.
2. L'administrador indica el dni, el nom, l'adreça, l'e-mail i el número de compte corrent de la persona
4. Per cada familiar, l'administrador introdueix les seves dades (dni, nom, adreça i e-mail)

Resposta del sistema

3. Comprova que les dades són correctes, dóna d'alta el soci i enregistra que paga per aquell número de compte corrent
4. Comprova que les dades són correctes, dóna d'alta el familiar i enregistra que és familiar d'aquell soci

Esquema conceptual d'especificació (I)



Esquema conceptual d'especificació (II)

Supòsit:

- Tots els intervals tenen una durada d'una hora

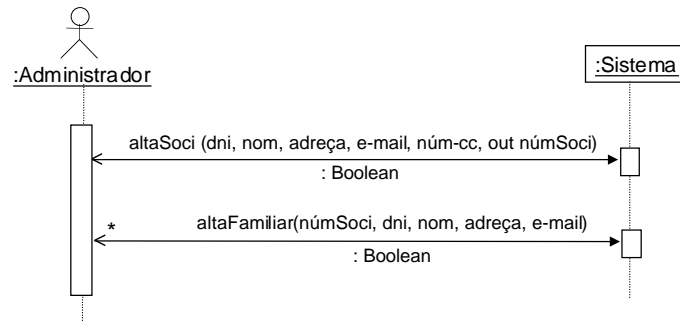
Restriccions d'integritat textuais:

- Claus classes no associatives: (Pista, núm); (Interval, data + inici); (Persona, dni); (CompteCorrent, núm-cc).
- No hi poden haver dos socis amb el mateix númSoci
- Tots els càrrecs d'un rebut tenen una data dins de mesEmissió
- Tots els càrrecs d'un mateix rebut són del mateix soci
- No hi pot haver més d'un rebut amb els mateixos númSoci i mesEmissió

Informació derivada:

- El deute d'un soci és la suma dels imports dels seus rebuts no pagats
- L'import d'un rebut és la suma dels imports dels càrrecs de que es compona
- L'associació *correspon* entre *Soci* i *Rebut* relaciona un soci amb els seus rebuts.

Diagrama de seqüència del sistema: Admetre soci



Contractes de les operacions del sistema

Operació:	altaSoci (dni: String, nom: String, adreça: String, e-mail: String, núm-cc: Integer, out númSoci: Integer)
Classe retorn:	Boolean
Semàntica:	Donar d'alta un nou soci i assignar-li el compte corrent.
Precondicions:	Els paràmetres tenen valor, excepte e-mail que pot ser nul.
Postcondicions:	<ol style="list-style-type: none"> 1. Si no existeix un compte corrent amb núm-cc, aleshores operació invàlida i es retorna fals. 2. En cas contrari, operació vàlida, es retorna cert i: <ol style="list-style-type: none"> 2.1 - Es crea un objecte de la classe Soci (i de la classe persona) 2.2 - L'atribut numSoci és igual a l'atribut últNúmSoci+1 de club 2.3 - L'atribut últNumSoci de club s'incrementa en una unitat 2.4 - Es crea una nova ocurrència de l'associació <i>Del</i> entre Soci i CompteCorrent

Operació:	altaFamiliar (númSoci: Integer; dni: String, nom: String, adreça: String, e-mail: String)
Classe retorn:	Boolean
Semàntica:	Donar d'alta un familiar d'un soci.
Precondicions:	Els paràmetres tenen valor, excepte e-mail que pot ser nul.
Postcondicions:	<ol style="list-style-type: none"> 1. Si no existeix un Soci amb númSoci, aleshores operació invàlida i es retorna fals. 2. En cas contrari, operació vàlida, es retorna cert i: <ol style="list-style-type: none"> 2.1 - Es crea un objecte de Familiar (i de Persona) 2.2 - Es crea una ocurrència de l'associació entre el familiar i el soci amb númSoci

Bibliografia

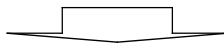
- *Enginyeria del software: Especificació*
D.Costal, M.R.Sancho, E.Teniente
Edicions UPC, 1999

Normalització de l'esquema conceptual

- Motivació
- Normalització de l'esquema conceptual
 - Tractament d'associacions n-àries i classes associatives
 - Tractament de la informació derivada
 - Tractament de les restriccions d'integritat
- Bibliografia

Motivació

- Al **disseny** hi tenim **components software** i no conceptes del domini
- **Limitació tecnologia orientada a objectes**: no permet implementar directament tots els conceptes que hem usat a l'especificació:
 - associacions n-àries, amb $n > 2$.
 - classes associatives
 - informació derivada
 - control de les restriccions d'integritat



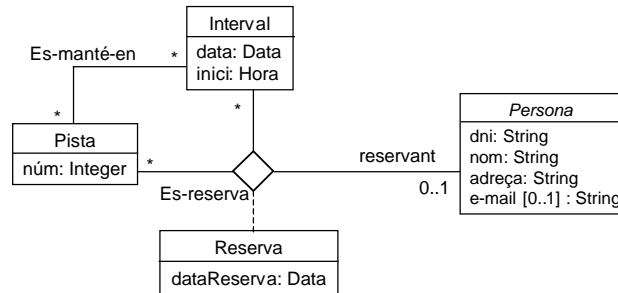
cal una transformació prèvia: **procés de normalització** (binarització)

- *eliminar n-àries i associatives*
- *tractar la informació derivada*
- *controlar les restriccions d'integritat*

que impacta tant al diagrama de classes com als contractes.

Eliminació d'associacions n-àries i classes associatives

Esquema conceptual:



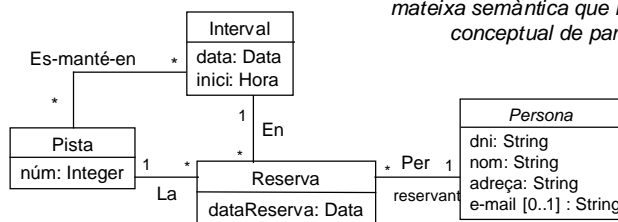
Restriccions d'integritat textuals:

- claus classes no associatives: (Pista, núm); (Interval, data + inici); (Persona, dni)

Normalització: pas a associacions binàries

Diagrama de classes normalitzat:

el resultat obtingut ha de tenir la mateixa semàntica que l'esquema conceptual de partida



Restriccions d'integritat textuals:

- claus classes no associatives: (Pista, núm); (Interval, data + inici); (Persona, dni)

~~----- (Afegida) No hi pot haver dues reserves amb els mateixos Pista, Interval i Persona -----~~

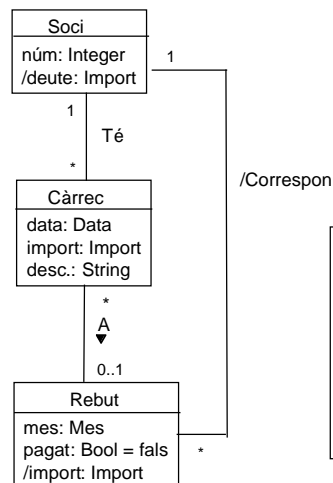
- (Afegida) Donats una pista i un interval, com a màxim els pot tenir reservats una Persona

s'elimina perquè està inclosa a la tercera restricció textual

Tractament de la informació derivada

- Els atributs i les associacions derivats es poden:
 - *Calcular* quan es necessiten.
 - *Materialitzar*
- Si es **calcula**:
 - **Desapareix** (explícitament) la **informació derivada** que es decideix calcular
 - **Apareixen** noves **operacions** per obtenir la informació derivada que es calcula
- Si es **materialitza**:
 - Cal **modificar** els **contractes** de les operacions que provoquen canvis al valor de la informació que es materialitza
 - S'elimina del diagrama de classes la indicació que la informació és derivada
- En la decisió hi influeix el temps de càlcul, la freqüència d'accés i l'espai ocupat.

Tractament de la informació derivada: exemple

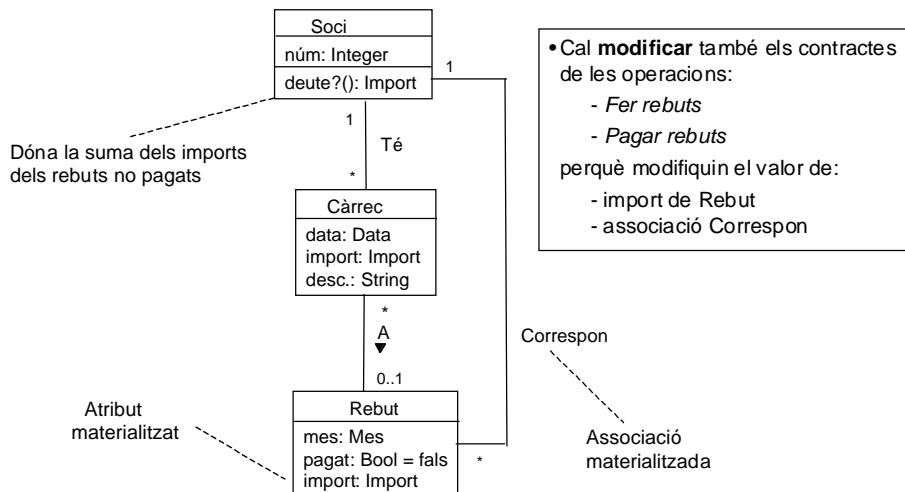


Esquema conceptual:

- Cal decidir si **calculem** o **materialitzem** la informació derivada.
- Decidim:
 - **calcular** l'atribut *deute* de *Soci*
 - **materialitzar** *import* de *Rebut*
 - **materialitzar** l'associació *Correspon*

Tractament de la informació derivada: resultat

Diagrama de classes resultant:



Control de les restriccions d'integritat

- Els models d'**especificació** són **no redundants** entre ells
- Quan dissenyem, cal garantir que el sistema **software** resultant **satisfaci les restriccions d'integritat** (gràfiques i textuais) de l'esquema conceptual.



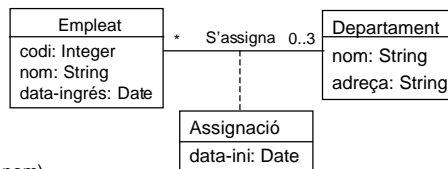
cal **modificar els contractes de les operacions** perquè controlin totes les restriccions d'integritat

com a conseqüència, les **restriccions d'integritat desapareixen** de l'esquema conceptual

- En general, la capa de presentació i la de gestió de dades també controlaran algunes restriccions d'integritat
- Per tant, aquest control no es farà únicament des de la capa de domini

Control de les restriccions d'integritat: exemple

Esquema conceptual:



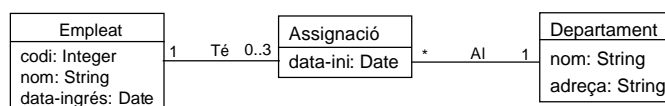
R.I. Textuals:

- Claus: (Empleat, codi); (Departament, nom)
- Data-ini d'assignació \geq data-ingrés
- Un empleat no pot estar assignat alhora als departaments de Vendes i de Control-Vendes

Operació:	novaAssig (codi: Enter, nom-d: String, data: Data)
Classe retorn:	Booleà
Semàntica:	Assignar un empleat a un departament
Precondicions:	Els paràmetres tenen valor
Postcondicions:	1. Si no existeix l'empleat, o no existeix el departament o l'empleat ja està assignat al departament aleshores operació invàlida i es retorna fals. 2. En cas contrari, operació vàlida, es retorna cert i: 2.1 - Es dona d'alta l'associació entre l'empleat i el departament.

Control de les rest. d'integritat: normalització de l'exemple

Diagrama de classes normalitzat:



R.I. Textuals:

- Claus: (Empleat, codi); (Departament, nom)
- Data-ini d'assignació \geq data-ingrés
- Un empleat no pot estar assignat alhora als departaments de Vendes i de Control-Vendes
- **(Afegida)** No hi pot haver dues assignacions amb els mateixos Empleat i Departament

Control de les restriccions d'integritat: resultat

Operació:	novaAssig (codi: Enter, nom-d: String, data: Data)
Classe retorn:	Booleà
Semàntica:	Assignar un empleat a un departament
Precondicions:	Els paràmetres tenen valor
Postcondicions:	<ol style="list-style-type: none"> 1. Si no existeix l'empleat, o no existeix el departament o l'empleat ja està assignat al departament aleshores operació invàlida i es retorna fals. 2. En cas contrari, operació vàlida, es retorna cert i: <ol style="list-style-type: none"> 2.1 - Es dona d'alta l'associació entre l'empleat i el departament.

Operació normalitzada:

Operació:	novaAssig (codi: Enter, nom-d: String, data: Data)
Classe retorn:	Booleà
Semàntica:	Assignar un empleat a un departament
Precondicions:	Els paràmetres tenen valor
Postcondicions:	<ol style="list-style-type: none"> 1. Si no existeix l'empleat, o no existeix el departament o l'empleat ja està assignat al departament, <i>o data < data-ingrés, o l'empleat ja està assignat a tres departaments o l'empleat estaria assignat a Vendes i a Control-Vendes</i>, aleshores operació invàlida i es retorna fals. (Modificada) 2. En cas contrari, operació vàlida, es retorna cert i: <ol style="list-style-type: none"> 2.1 - <i>Es crea un nou objecte Assignació amb les corresponents associacions amb Empleat i Departament.</i> (Modificada)

Bibliografia

- *The Unified Modeling Language User Guide*
G.Booch; J.Rumbaugh; I.Jacobson
Addison-Wesley, 1999.
- *Applying UML and Patterns.*
An Introduction to Object-Oriented Analysis and Design
C.Larman
Prentice Hall, 1998.

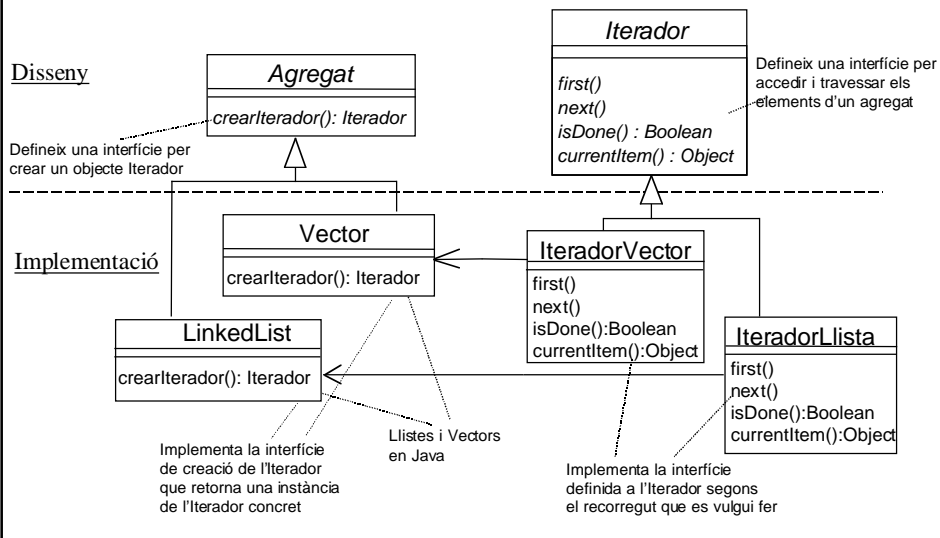
Patró disseny Iterador

- Descripció general.
- Solució: Estructura.
- Exemple d'aplicació en UML.
- Iteradors en Java: Enumeration.
- Exemple d'aplicació en Java.
- Diccionaris.
- Exemple d'iteració de diccionaris.
- Bibliografia.

Descripció general

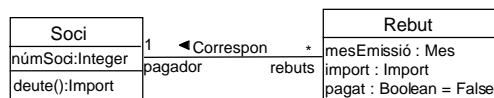
- **Context:**
 - Necessitat de fer recorreguts seqüencials dels elements d'un objecte agregat (col·lecció d'objectes).
- **Problema:**
 - Es vol accedir als elements d'un objecte agregat, sense exposar l'estructura interna.
 - Es vol poder tenir diverses travessies pendents del mateix objecte agregat.
 - Es vol poder travessar els elements d'un objecte agregat de formes diferents.
 - No es vol *inflar* la classe de l'objecte agregat amb operacions per les diverses travessies possibles.
- **Solució:**
 - Treure la responsabilitat de l'accés i recorregut de l'objecte agregat i assignar aquesta responsabilitat a una nova classe d'objectes que anomenarem Iterador.
 - L'Iterador proporcionarà una interfície per accedir i recórrer els elements de l'objecte agregat.

Solució: Estructura Cas d'un únic recorregut per agregat



Exemple d'aplicació en UML: Problema

- Diagrama de classes normalitzat:



R.I. Textuals

- No poden existir dos socis amb el mateix numSoc.
- Un soci no pot tenir diversos rebuts amb el mateix mesEmissió.

- Es vol dissenyar l'operació deute de Soci (suma dels imports dels seus rebuts no pagats).



Aplicació del patró Iterador

Exemple d'aplicació en UML: Solució

Diagrama de seqüència

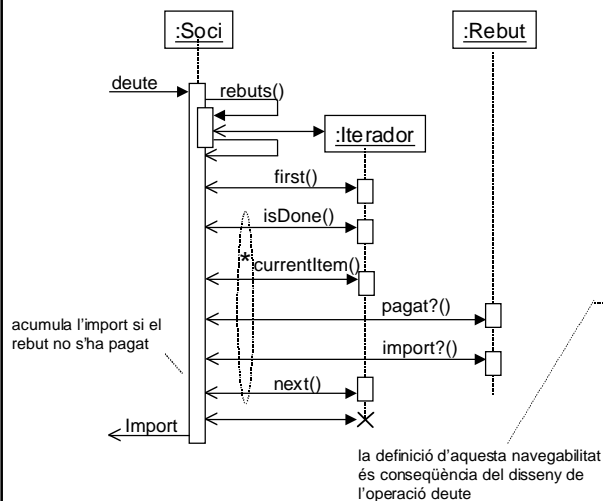
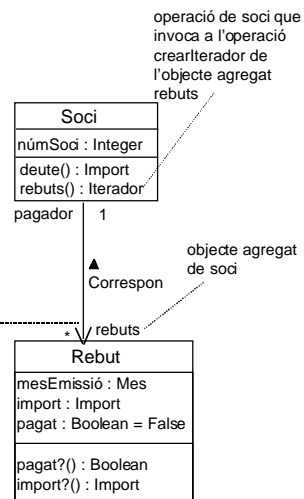
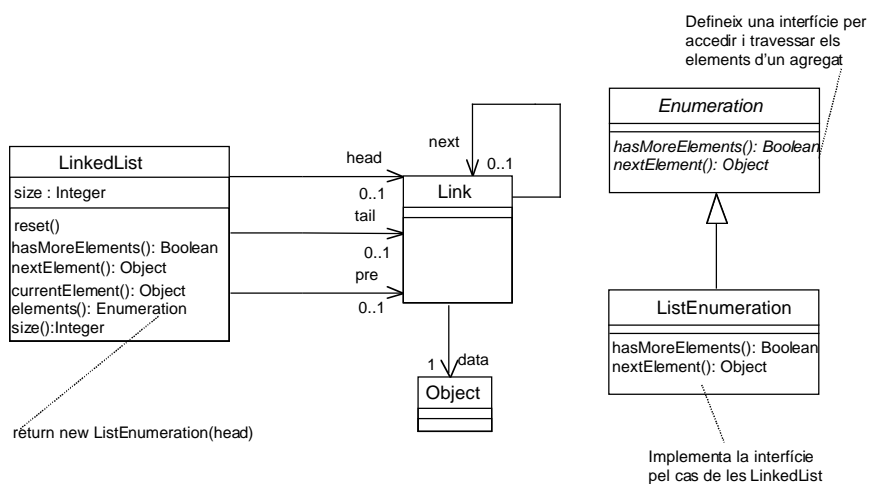


Diagrama de classes



Iteradors en Java: Enumeration



Exemple d'aplicació en Java

Definició de classes en Java

```

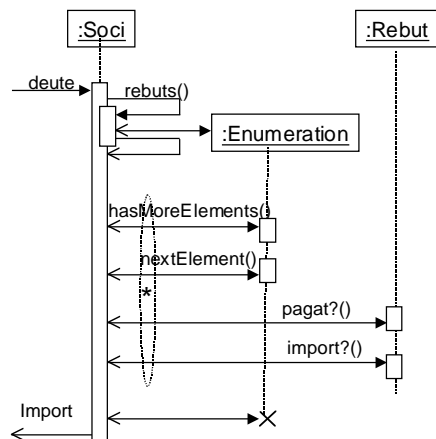
Class Soci
{
...
LinkedList rebuts
...
}

Class Rebut
{
...
}

```

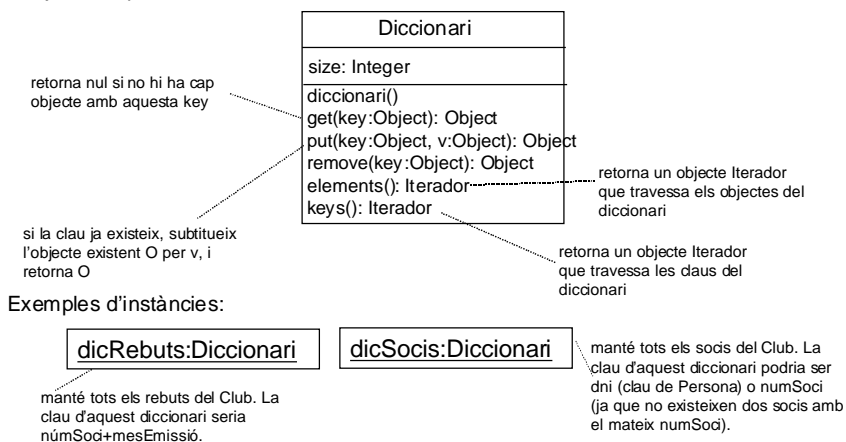
Implementació de l'associació Correspon i la seva navegabilitat, mitjançant atributs

Diagrama de seqüència usant Java

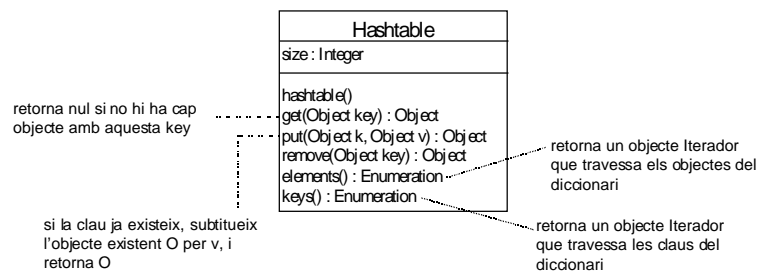


Diccionaris

- La classe d'objectes Diccionari manté una col·lecció d'objectes que poden ser accedits mitjançant una clau 'key'.
- La clau 'key' per accedir als objectes d'un diccionari ha de ser clau externa de l'objecte al que es vol accedir.



Diccionaris en Java: Hashtable



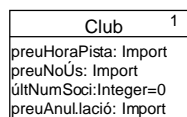
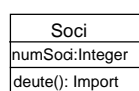
Exemple d'instància:

`dicRebut:Diccionari`

`dicSocis:Hashtable`

Exemple d'iteració de diccionaris

- Diagrama de classes normalitzat:



R.I. Textuals

- No poden existir dos socis amb el mateix numSoc.

- Es vol calcular el deute que té el Club (suma dels deutes de tots els socis).

Patró de disseny Controlador

- Descripció general
- Controlador Façana
 - façana-sistema
 - façana-empresa
- Controlador Paper
- Controlador Cas d'Ús
- Controlador Transacció
- Relació entre la Capa Presentació i:
 - Controlador Transacció
 - Combinació de controladors (Façana i Transacció)
- Bibliografia

Descripció general

- **Context:**
 - Els sistemes software reben esdeveniments externs
 - Un cop interceptat un esdeveniment a la Capa de Presentació, algun objecte de la Capa del Domini ha de rebre aquest esdeveniment i executar les accions corresponents
- **Problema:**
 - Quin objecte és el responsable de tractar un esdeveniment extern?
- **Solució:**
 - Assignar aquesta responsabilitat a un controlador
 - Un controlador és un objecte d'una certa classe
 - Possibles controladors:
 - . Façana-Sistema: Un objecte que representa tot el sistema
 - . Façana-Empresa: Un objecte que representa tot el domini (empresa, etc.)
 - . Paper: Un objecte que representa un actor
 - . Cas d'ús: Un objecte que representa una instància d'un cas d'ús
 - . Transacció: Un objecte que representa una instància d'esdeveniment extern

Controlador Façana: façana-sistema

- Tots els esdeveniments externs són processats per un sol objecte controlador façana
- Controladors *inflats* si hi ha molts esdeveniments externs
- N'hi ha de dos tipus depenent el significat que li donem:
 - Controlador façana-sistema
 - Controlador façana-empresa

Façana-sistema: Un objecte que representa tot el sistema software (SistemaGestióClub)

SistemaGestióClub	1
altaSoci(dni:String, nom:String, adreça:String, e-mail:String, núm-cc:Integer, out númSoci:Integer): Boolean altaFamiliar(númSoci:Integer, dni:String, nom:String, adreça:String, e-mail:String): Boolean baixaSoci(númSoci:Integer) : Boolean baixaFamiliar(númSoci:Integer, dni:String) : Boolean canviCompteCorrent(númSoci:Integer, núm-cc:Integer): Boolean	

Controlador Façana: façana-empresa

Façana-empresa: Un objecte que representa tot el domini del problema (Club)

Club	1
últNumsoci: Integer=0 preuNoÚs: Import preuHoraPista: Import preuAnul·lació: Import	
últNúmSoci?(): Integer incrementar-últNúmSoci() altaSoci(dni:String, nom:String, adreça:String, e-mail:String, núm-cc:Integer, out númSoci:Integer): Boolean baixaSoci(númSoci:Integer) : Boolean altaFamiliar(númSoci:Integer, dni:String, nom:String, adreça:String, e-mail:String): Boolean baixaFamiliar(númSoci:Integer, dni:String) : Boolean canviCompteCorrent(númSoci:Integer, núm-cc:Integer): Boolean	

En aquest exemple hem aprofitat l'objecte Club del model conceptual i li hem afegit la responsabilitat de ser el controlador.

Controlador Paper

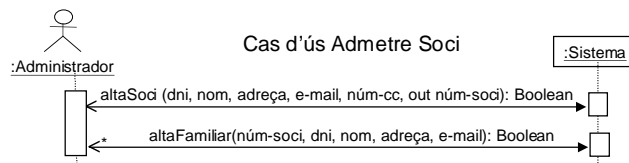
Controlador Paper: Un objecte que representa un actor dels diagrames de casos d'ús

Administrador
<code>altaSoci(dni:String, nom:String, adreça:String, e-mail:String, núm-cc:Integer, out númSoci:Integer): Boolean</code> <code>baixaSoci(númSoci:Integer) : Boolean</code> <code>canviCompteCorrent(númSoci:Integer, núm-cc:Integer): Boolean</code> <code>altaFamiliar(númSoci:Integer, dni:String, nom:String, adreça:String, e-mail:String): Boolean</code> <code>baixaFamiliar(númSoci:Integer, dni:String) : Boolean</code>

- Controlador idoni si es volen controlar les accions realitzades per l'actor
- Problema si un mateix esdeveniment extern pot ser generat per diversos actors
- Hi haurà un controlador per cada actor

Controlador Cas d'Ús

Controlador Cas d'Ús: Un objecte que representa un cas d'ús

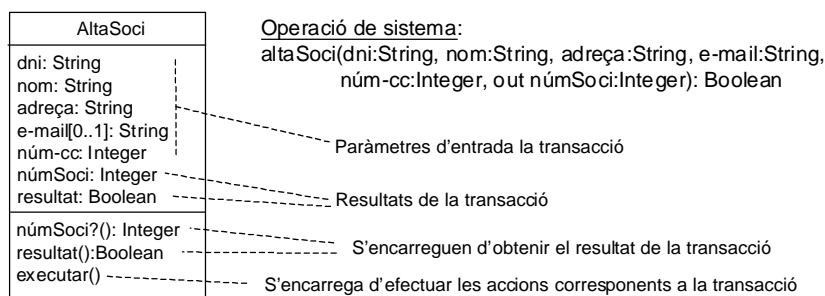


Admetre Soci
<code>altaSoci(dni:String, nom:String, adreça:String, e-mail:String, núm-cc:Integer, out númSoci:Integer): Boolean</code> <code>altaFamiliar(númSoci:Integer, dni:String, nom:String, adreça:String, e-mail:String): Boolean</code>

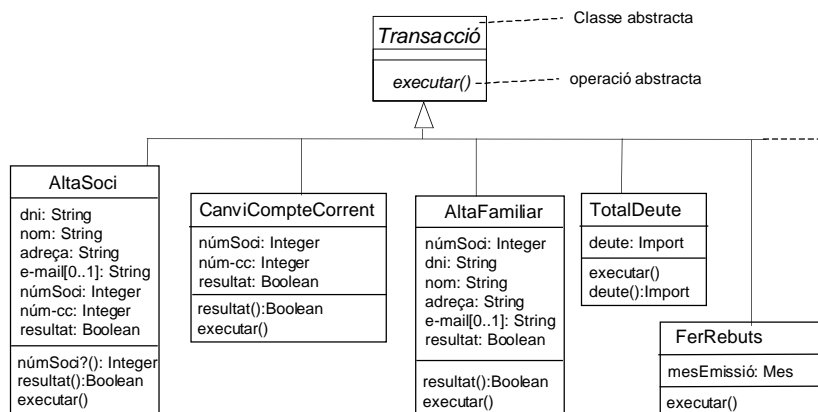
- Pot tenir atributs definits per controlar l'estat del cas d'ús
- Problema si un mateix esdeveniment extern apareix en diversos casos d'ús
- Hi haurà tants controladors com casos d'ús tingui el sistema

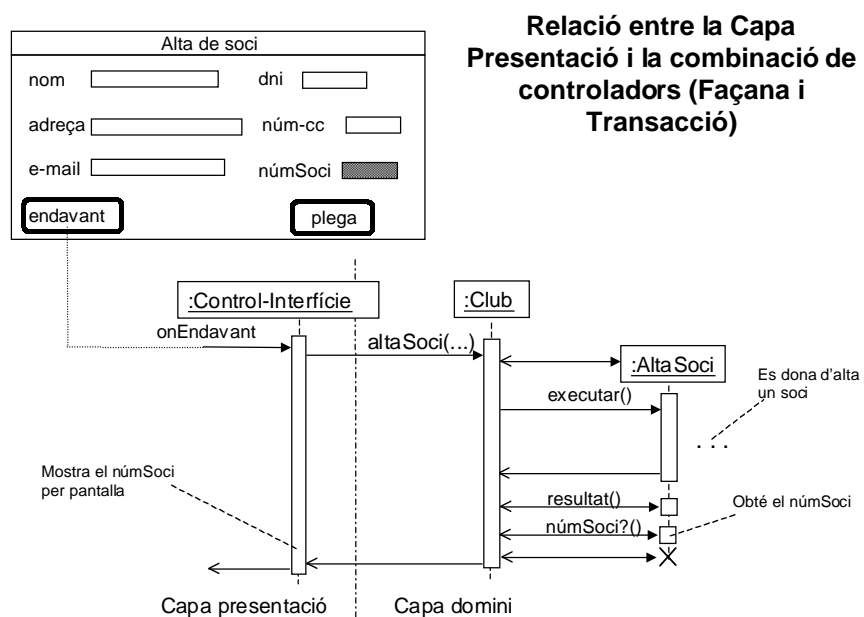
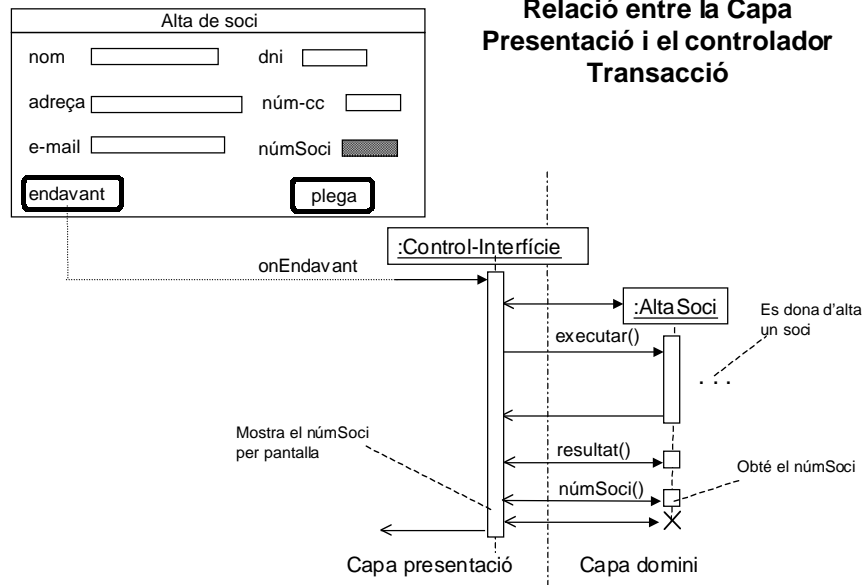
Controlador Transacció: Definició

- Basat en el patró de disseny "Command"
- Es crea un objecte transacció a cada ocurrència d'un esdeveniment extern
- Els paràmetres i el retorn de l'operació de sistema associada a l'esdeveniment extern es corresponen als atributs de la transacció
- L'execució de la transacció es realitza amb la invocació de l'operació **executar()**, i es defineixen operacions específiques per a consultar els resultats de l'execució de la transacció
- S'acostuma a destruir l'objecte transacció un cop recuperat el resultat



Estructura de les classes Transacció





Bibliografia

- *Applying UML and Patterns*
An Introduction to Object-Oriented Analysis and Design
C. Larman
Prentice Hall, 1998. Cap. 18.13.
- *Design Patterns. Elements of Reusable Object-Oriented Software*
E. Gamma, R. Helm, R. Johnson, J. Vlissides
Addison-Wesley, 1995, pp. 233-242.

Assignació de responsabilitats a objectes

- Responsabilitats dels objectes
- Criteris per a l'assignació de responsabilitats:
 - Acoblament baix
 - Cohesió alta
- Patrons d'assignació de responsabilitats:
 - (Controlador)
 - Expert
 - Creador
- Bibliografia

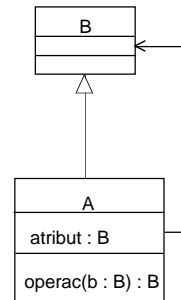
Responsabilitats dels objectes

- **L'assignació de responsabilitats a objectes** consisteix a determinar (assignar) quines són les obligacions (responsabilitats) concretes dels objectes del diagrama de classes per donar resposta als esdeveniments externs.
- Les responsabilitats d'un objecte consisteixen en:
 - Saber:
 - . Sobre els atributs de l'objecte
 - . Sobre els objectes associats
 - . Sobre dades que es poden derivar
 - Fer:
 - . Fer quelcom en el propi objecte
 - . Iniciar una acció en altres objectes
 - . Controlar i coordinar activitats en altres objectes
- L'assignació de responsabilitats a objectes es fa durant el disseny, al definir i localitzar les operacions de cada classe d'objectes.

Acoblament

- **Acoblament** d'una classe és una mesura del grau de connexió, coneixement i dependència d'aquesta classe respecte d'altres classes.

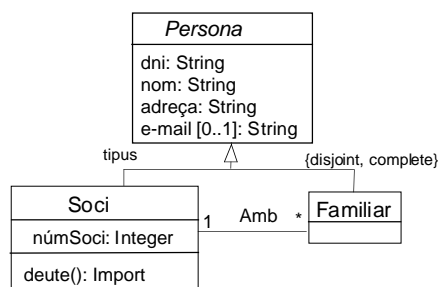
- Hi ha un acoblament de la classe A a la classe B si:
 - A té un atribut de tipus B
 - A té una associació navegable amb B
 - B és un paràmetre o el retorn d'una operació de A
 - Una operació de A referència un objecte de B
 - A és una subclasse directa o indirecta de B



- L'acoblament entre classes ha de ser baix per tal de:
 - Evitar que canvis en una classe tinguin efecte sobre altres classes
 - Facilitar la comprensió de les classes (sense recórrer a altres)
 - Facilitar la reutilització

Anàlisi d'acoblament: Exemple

Diagrama de classes normalitzat



Restriccions d'integritat:

Claus de classes no associatives (Persona, dni)

No hi poden haver dos socis amb el mateix númSoci

Suposicions:

L'associació Amb té navegabilitat doble

Anàlisi d'acoblament: Exemple

Contracte altaFamiliar normalitzat i controlador

Operació: altaFamiliar (númSoci: Integer, dni:String, nom:String, adreça:String, e-mail:String)

Classe Retorn: Boolean

Semàntica: Donar d'alta un familiar d'un soci.

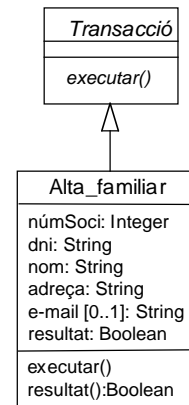
Precondicions: Els paràmetres tenen valor, excepte e-mail que pot ser nul.

Postcondicions: 1. En el casos següents, operació invàlida i es retorna fals:

- 1.1 No existeix un soci amb númSoci
- 1.2 Ja existeix una Persona amb aquest dni (*Afegida*)

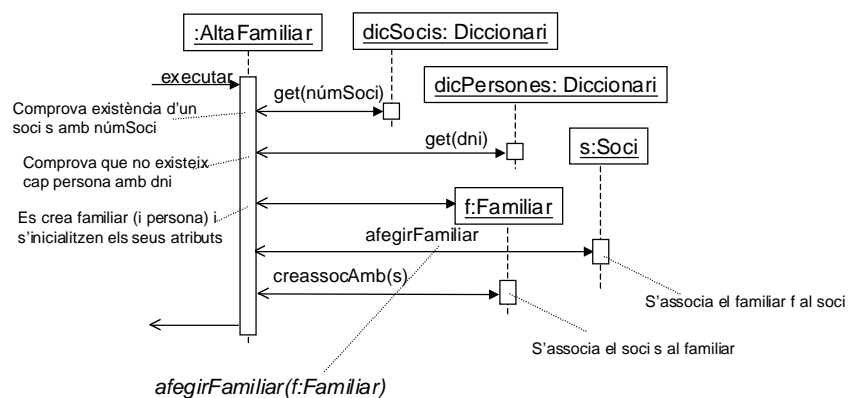
2. En cas contrari, operació vàlida, es retorna cert i:

- 2.1 Es crea un objecte de Familiar (i de Persona)
- 2.2 Es crea una ocurrència de l'associació *Amb* entre Familiar i el soci amb númSoci



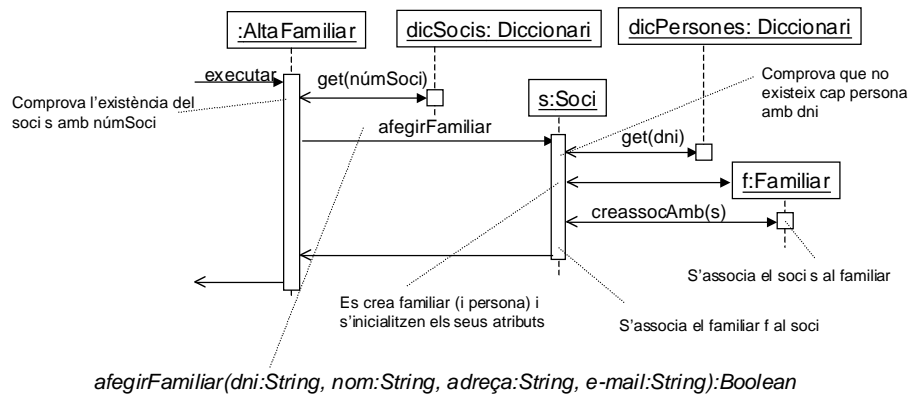
Anàlisi d'acoblament: Exemple

Diagrama de seqüència (I)



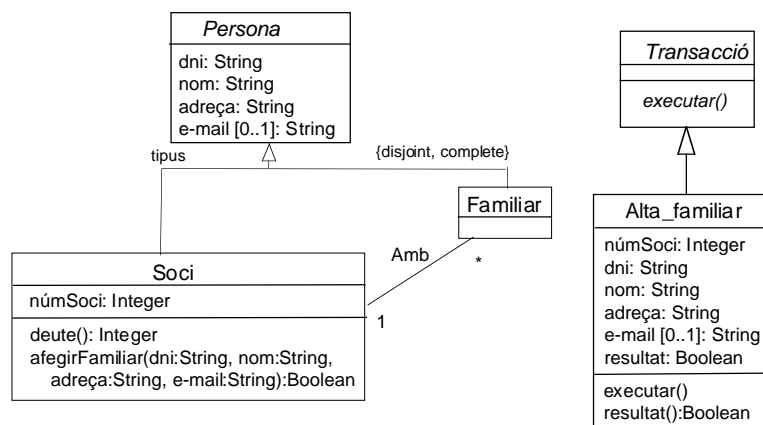
La classe AltaFamiliar està acoblada a dicSocis, dicPersones, Soci i Familiar
 La classe Soci està acoblada a Familiar i, aquesta a Soci

Anàlisi d'acoblament: Exemple Diagrama de seqüència (II)



La classe AltaFamiliar està acoblada a dicSocis i Soci
 La classe Soci està acoblada a dicPersones i Familiar.
 La classe Familiar està acoblada a Soci.

Anàlisi d'acoblament: Exemple Diagrama de classes de disseny



Cohesió

- Cohesió d'una classe és una mesura del grau de relació i de concentració de les diverses responsabilitats d'una classe.
- Una classe amb cohesió alta:
 - Té poques responsabilitats en una àrea funcional
 - Col.labora (delega) amb d'altres classes per a fer les tasques
 - Acostuma a tenir poques operacions
 - Aquestes operacions estan molt relacionades funcionalment
 - Avantatges:
 - Fàcil comprensió
 - Fàcil reutilització i manteniment
- Una classe amb cohesió baixa:
 - És l'única responsable de moltes coses i de diverses àrees funcionals
 - Té moltes operacions i/o operacions poc relacionades funcionalment
 - Inconvenients:
 - És difícil de comprendre
 - És difícil de reutilitzar i de mantenir
 - És molt sensible als canvis

Cohesió: exemple

- Els controladors façana acostumen a ser poc cohesius

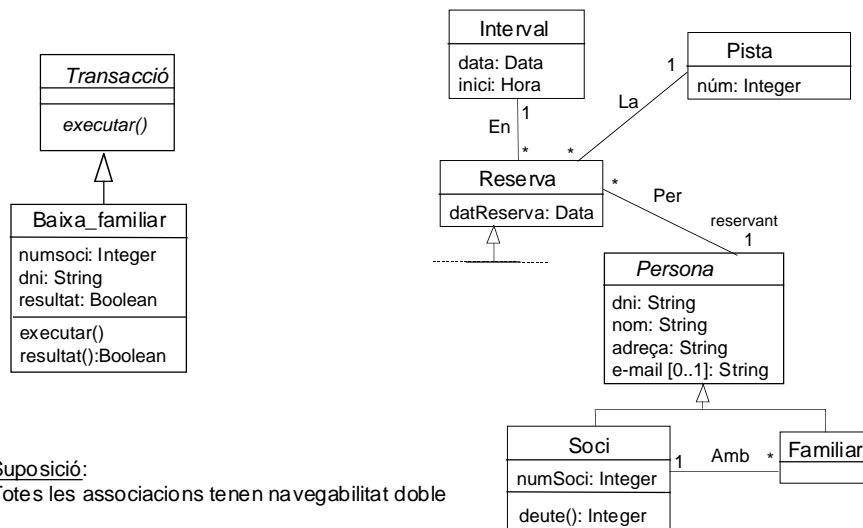
Club	1
últNumsoci: Integer=0 preuNoÚs: Import preuHoraPista: Import preuAnul·lació: Import	
últNúmSoci?(): Integer incrementar-últNúmSoci() altaSoci(dni:String, nom:String, adreça:String, e-mail:String, núm-cc:Integer, out númSoci:Integer): Boolean baixaSoci(númSoci:Integer) : Boolean altaFamiliar(númSoci:Integer, dni:String, nom:String, adreça:String, e-mail:String): Boolean baixaFamiliar(númSoci:Integer, dni:String) : Boolean canviCompteCorrent(númSoci:Integer, núm-cc:Integer): Boolean	

Patró Expert

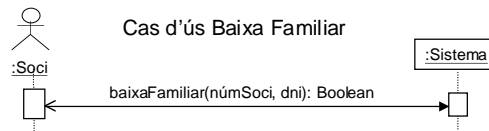
- Context:
 - Assignació de responsabilitats a objectes
- Problema:
 - Decidir a quina classe hem d'assignar una responsabilitat concreta
- Solució:
 - Assignar una responsabilitat a la classe que té la informació necessària per realitzar-la
 - L'aplicació del patró requereix tenir clarament definides les responsabilitats que es volen assignar (postcondicions de les operacions)
 - No sempre existeix un únic expert, sinó que poden existir diversos experts parcials que hauran de col·laborar.
- Beneficis:
 - Es manté l'encapsulament → baix acoblament
 - Conducta distribuïda entre les classes que tenen la informació → alta cohesió

Patró Expert: Exemple

Diagrama de classes inicial (normalitzat)



Patró Expert: Exemple Cas d'Ús Baixa Familiar



Operació: baixaFamiliar (númSoci:Integer, dni:String)

Classe Retorn: Boolean

Semàntica: Un soci vol donar de baixa un dels seus familiars.

Precondicions: Els paràmetres tenen valor.

Postcondicions:

1. En el casos següents, operació invàlida i es retorna fals:
 - 1.1 No existeix un Familiar amb el dni especificat
 - 1.2 No existeix un soci amb el númSoci especificat
 - 1.3 La persona amb dni no és familiar del soci amb númSoci
2. En cas contrari, operació vàlida, es retorna cert i:
 - 2.1 Es dóna de baixa l'objecte de Familiar (i de Persona)
 - 2.2 S'eliminen les Reserves realitzades pel Familiar

Patró Expert: Exemple Contracte operació baixaFamiliar normalitzat

Operació: baixaFamiliar (númSoci:Integer, dni:String)

Classe Retorn: Boolean

Semàntica: Donar de baixa un familiar d'un soci.

Precondicions: Els paràmetres tenen valor.

Postcondicions:

1. En el casos següents, operació invàlida i es retorna fals:
 - 1.1 No existeix un Familiar amb el dni especificat
 - 1.2 No existeix un soci amb el númSoci especificat
 - 1.3 La persona amb dni no és familiar del soci amb númSoci
2. En cas contrari, operació vàlida, es retorna cert i:
 - 2.1 Es dóna de baixa l'objecte de Familiar (i de Persona)
 - 2.2 S'eliminen les Reserves realitzades pel Familiar

Experts



(dicPersones i Persona)

(dicSocis)

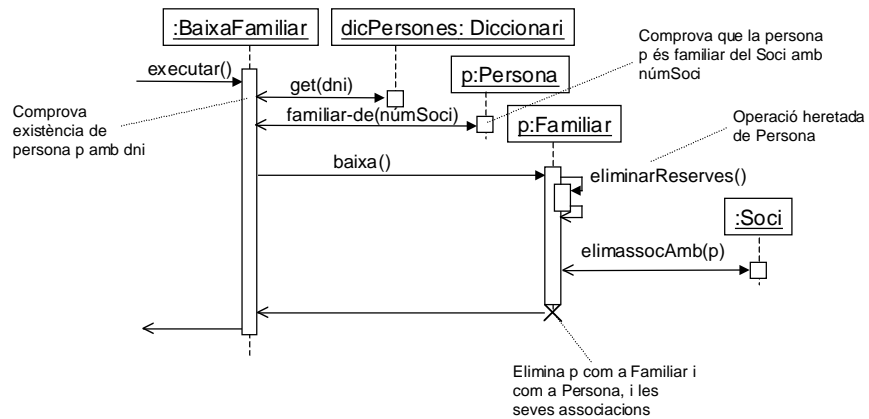
(Soci, Familiar)

(Familiar)

(Persona)

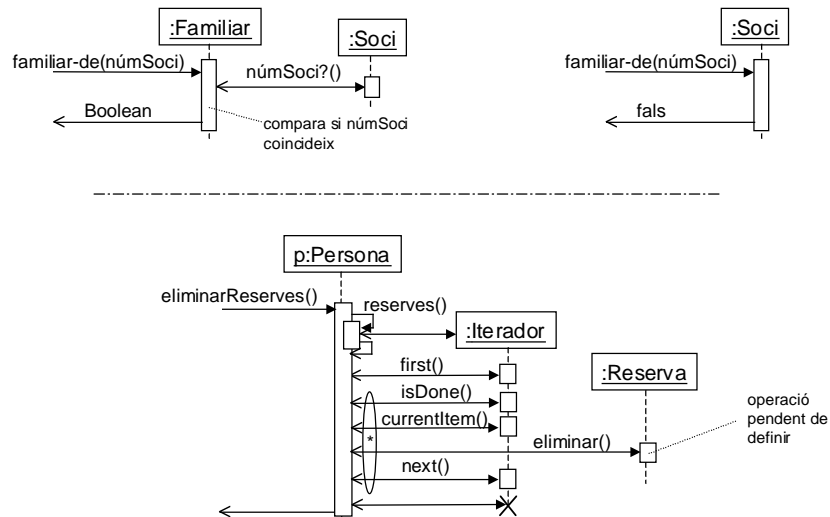
Patró Expert: Exemple

Diagrama de seqüència baixaFamiliar (I)



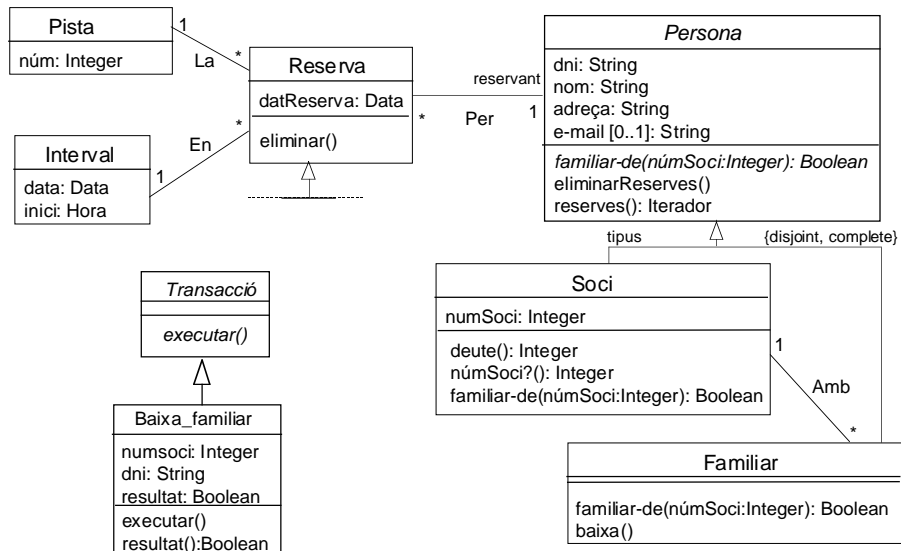
Patró Expert: Exemple

Diagrama de seqüència baixaFamiliar (II)



Patró Expert: Exemple

Diagrama de classes de disseny

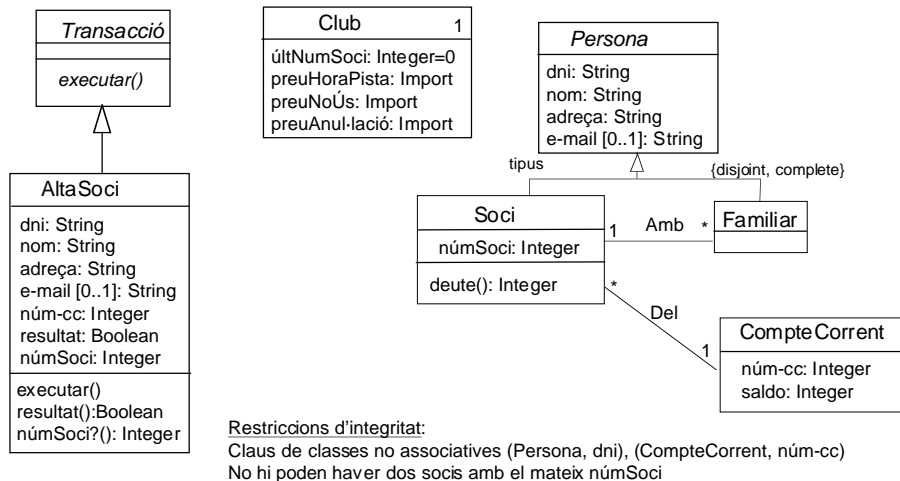


Patró Creador

- Context:
 - Assignació de responsabilitats a objectes
- Problema:
 - Qui ha de tenir la responsabilitat de crear una nova instància d'una classe.
- Solució:
 - Assignar a una classe B la responsabilitat de crear una instància d'una classe A si se satisfà una de les condicions següents:
 - B és un agregat d'objectes d'A
 - B conté objectes d'A
 - B enregistra instàncies d'objectes d'A
 - B usa molt objectes d'A
 - B té les dades necessàries per inicialitzar un objecte d'A (B té els valors dels paràmetres del constructor d'A)
- Beneficis:
 - Acoblament baix

Patró Creador: Exemple

Diagrama de classes normalitzat i Controlador



Patró Creador: Exemple

Contracte operació altaSoci normalitzat

Operació: altaSoci (dni:String, nom:String, adreça:String, e-mail:String, núm-cc: Integer, out númSoci: Integer)

Classe Retorn: Boolean

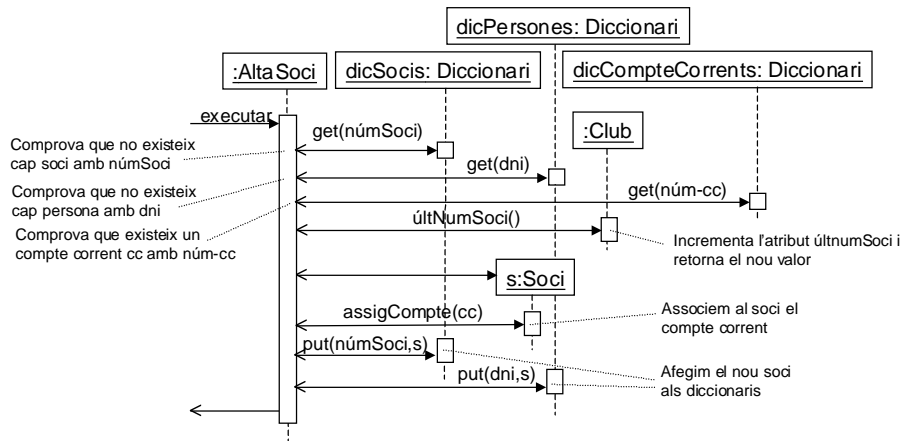
Semàntica: Donar d'alta un soci i assignar-li el compte corrent.

Precondicions: Els paràmetres tenen valor, excepte e-mail que pot ser nul.

Postcondicions:

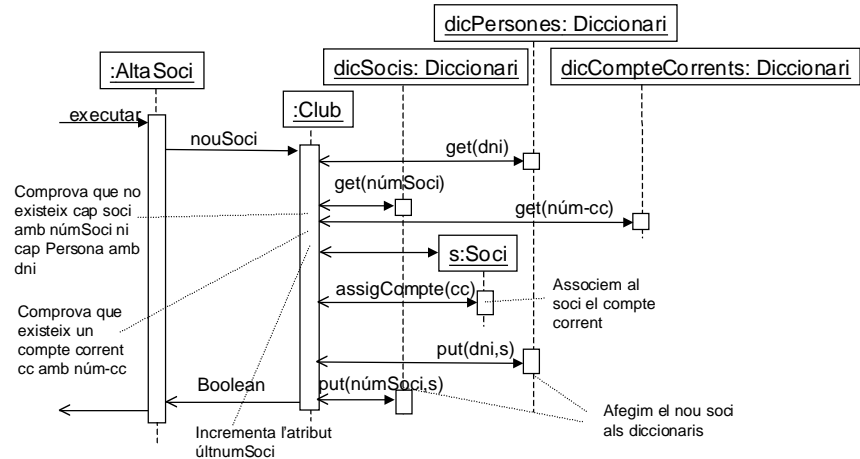
1. En el casos següents, operació invàlida i es retorna fals:
 - 1.1 No existeix un compte corrent amb núm-cc
 - 1.2 Ja existeix un Soci amb aquest númSoci (**Afegida**)
 - 1.3 Ja existeix una Persona amb aquest dni (**Afegida**)
2. En cas contrari, operació vàlida, es retorna cert i:
 - 2.1 Es crea un objecte de la classe Soci (i de la classe Persona)
 - 2.2 L'atribut númSoci és igual a l'atribut últNumSoci+1 de Club
 - 2.3 L'atribut últNumSoci de Club s'incrementa en una unitat
 - 2.4 Es crea una ocurrència de l'associació Del entre Soci i CompteCorrent

Patró Creador: Exemple Diagrama seqüència AltaSoci --- Cas 1



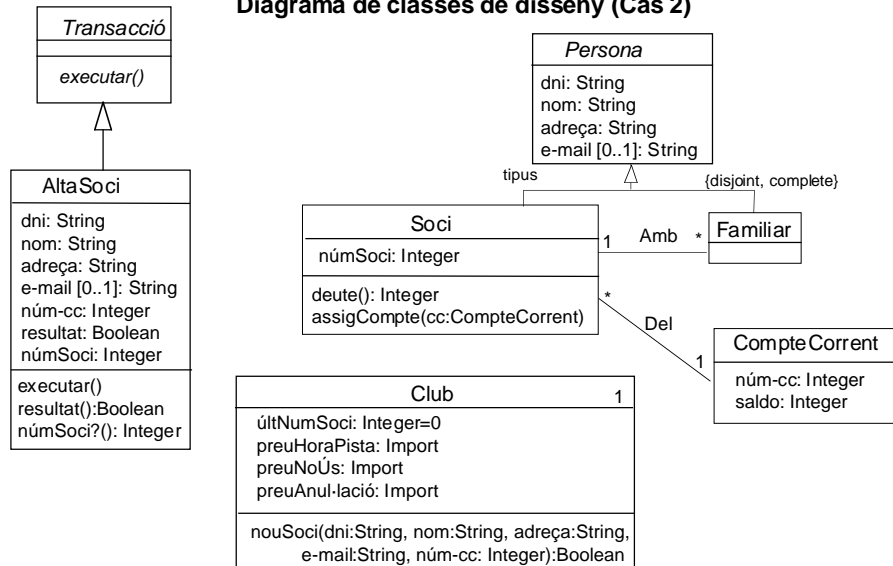
AltaSoci té totes les dades necessàries per inicialitzar Soci, excepte númSoci
Alta Soci acoblat amb els tres diccionaris, Club i Soci. Soci acoblat amb CompteCorrent

Patró Creador: Exemple Diagrama seqüència AltaSoci --- Cas 2



Club rep les dades necessàries per inicialitzar Soci, excepte númSoci.
Alta Soci acoblat amb Club. Club acoblat amb els tres diccionaris i Soci. Soci acoblat amb CompteCorrent
Disminueix la cohesió de Club (si no té altres responsabilitats relatives a Soci) i augmenta el seu acoblament.

Patró Creador: Exemple Diagrama de classes de disseny (Cas 2)



Bibliografia

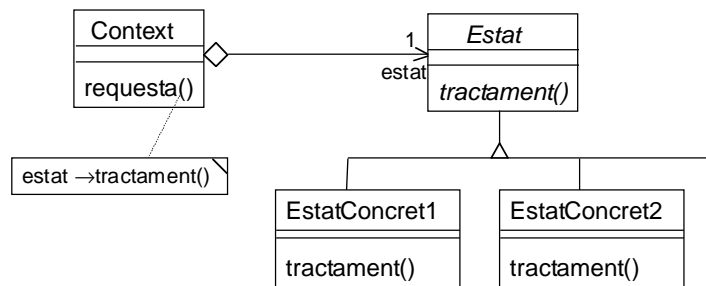
- *Applying UML and Patterns*
An Introduction to Object-Oriented Analysis and Design
Larman, C.
Prentice Hall, 1998. Caps. 18 i 19.

Patró de disseny Estat

- Descripció general.
- Solució: Estructura.
- Aspectes a considerar.
- Exemple 1.
- Conseqüències.
- Exemple 2.
- Bibliografia.

Descripció general

- **Context:**
 - En una jerarquia d'especialització, la semàntica d'una operació és diferent a les diverses subclasses per les que pot passar un objecte.
 - El comportament d'un objecte depèn del seu estat en temps d'execució, que és variable en el decurs del temps.
- **Problema:**
 - La tecnologia actual no permet que un objecte canviï dinàmicament de subclasse.
 - Les estructures condicionals per tractar el comportament en funció de l'estat no són desitjables ja que afegeixen complexitat i/o duplicació de codi.
- **Solució:**
 - Crear una classe per cada estat que pugui tenir l'objecte *context*.
 - El canvi de subclasse se simula pel canvi de l'associació amb la classe estat.
 - Basant-se en el polimorfisme, assignar mètodes a cada classe estat per tractar la conducta de l'objecte *context*.
 - Quan l'objecte *context* rep un missatge que depèn de l'estat, el reenvia a l'objecte *estat*.

Solució: Estructura

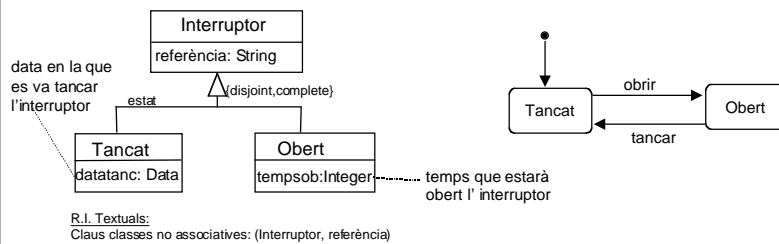
- Altres aspectes a considerar en l'aplicació del patró estat:
 - Compartició o no dels objectes estat concrets (multiplicitat associada al rol del context).
 - On estan definides les transicions d'estat.
 - Moment de creació i destrucció dels objectes estat.

Aspectes a considerar: Compartició d'objectes estat

- Estats no compartits:
 - Cada objecte context té el seu propi objecte estat, que inclou tota la informació rellevant d'aquell objecte en aquell estat.
- Estats compartits:
 - Diversos objectes context poden usar els mateixos objectes estat, si els objectes estat no tenen atributs/associacions.
 - Un objecte estat pot no tenir atributs/associacions:
 - Si no els necessita.
 - Si en té, però es defineixen en l'objecte context.
 - Dues variants:
 - » L'objecte context els passa a l'estat quan els necessita.
 - » Els objectes estat els obtenen del context quan els necessiten.

Exemple 1: Problema

- Esquema conceptual de l'especificació i diagrama d'estats d'Interruptor:



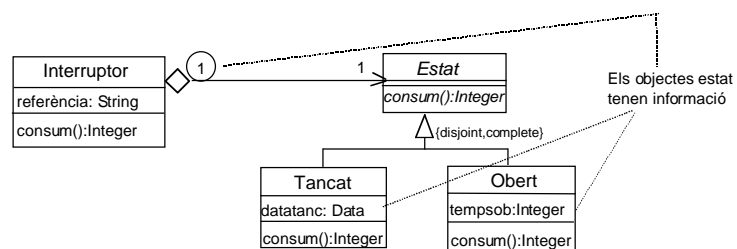
- Es vol dissenyar l'operació consum d'un interruptor:
 - és 0 si l'interruptor està tancat.
 - és el temps que estarà obert multiplicat per un factor alfa si l'interruptor està obert.



Aplicació del patró Estat

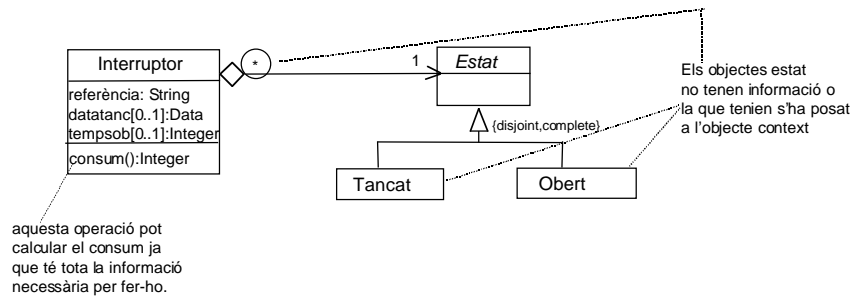
Exemple 1: Compartició d'objectes estat

- No Compartició dels objectes estat:



Exemple 1: Compartició d'objectes estat

- **Compartició dels objectes estat:**

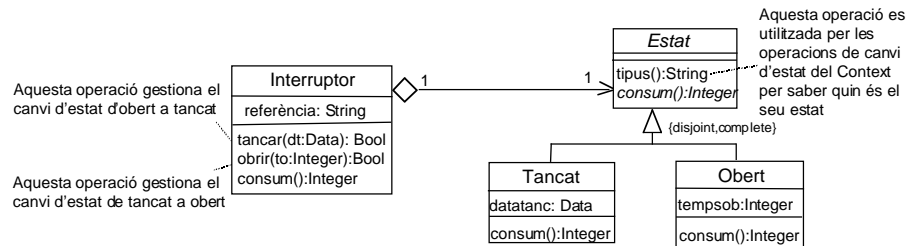


Aspectes a considerar: On es defineixen les transicions d'estat

- Les transicions d'estat vénen determinades pel diagrama d'estats d'especificació.
- El patró no especifica quin participant té la responsabilitat de portar a terme les transicions d'estat.
- La responsabilitat de portar a terme les transicions d'estat pot estar definida a:
 - L'objecte context:
 - Si els estats són compartits per diversos contextos.
 - Si les regles de canvi d'estat són fixes.
 - Els objectes estat:
 - És més flexible que siguin els propis objectes estat els qui decideixin quan fer la transició i especificar el canvi d'estat.
 - Hi ha d'haver una operació en l'objecte context que permeti canviar l'estat.
 - Té l'inconvenient que una subclasse estat ha de conèixer almenys una altra subclasse.

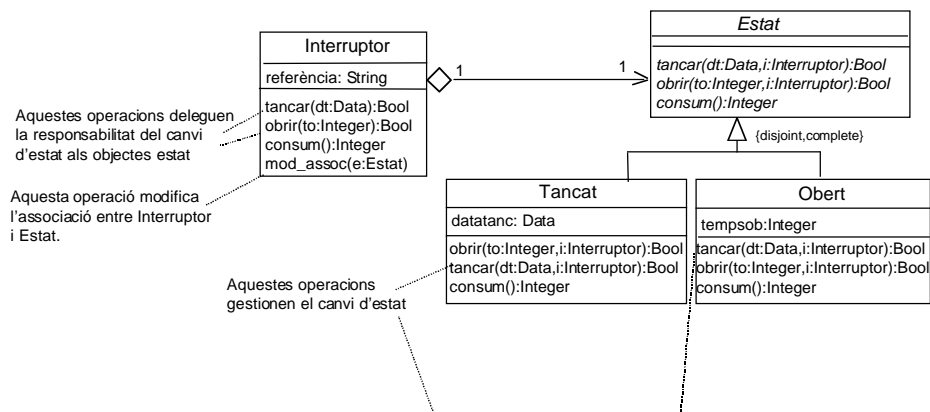
Exemple 1: On es defineixen les transicions d'estat

- Transicions d'estat definides a l'objecte context (estats no compartits):



Exemple 1: On es defineixen les transicions d'estat

- Transicions d'estat definides als objectes estat (estats no comparits):

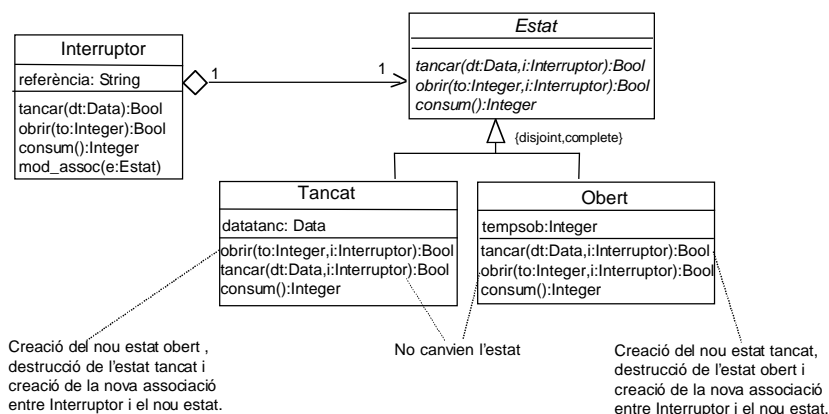


Aspectes a considerar: Creació i destrucció d'objectes estat

- Dues opcions:
 - A: Crear-los quan es necessiten i destruir-los quan ja no són necessaris.
 - B: Crear-los d'entrada i no destruir-los mai.
- Opció A:
 - Preferible quan:
 - L'estat inicial no es coneix en temps d'execució.
 - Els contextos rarament canvien d'estat.
 - Evita la creació d'objectes estat que no s'usaran.
- Opció B:
 - Preferible quan :
 - L'objecte context canvia d'estat molt sovint.
 - Evita la destrucció d'objectes que poden ser necessaris més endavant.
 - Els costos de creació (instanciació) es paguen un sol cop.
 - No hi ha costos de destrucció.

Exemple 1: Creació i destrucció dels objectes estat

- Creació i destrucció dels objectes estat **quan es necessiten**:

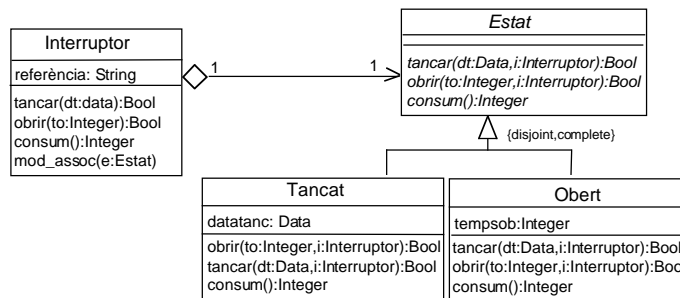


Exemple 1: Solució completa

- Diagrama de classes de disseny:

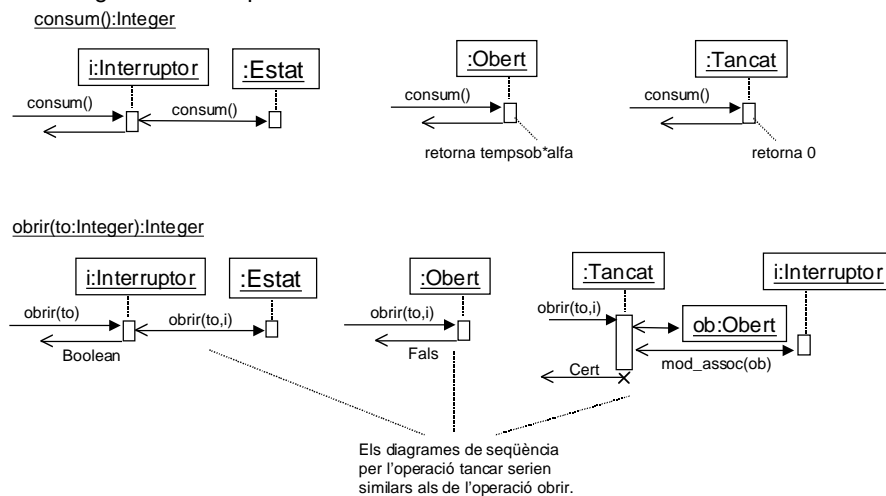
Hipòtesis d'aplicació del patró Estat:

- Objectes estat no compartits.
- Transicions d'estat definides en els objectes estat.
- Creació dels objectes quan es necessiten.



Exemple 1: Solució completa

- Diagrames de seqüència:

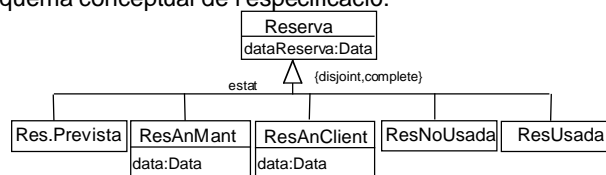


Conseqüències

- Descomposa la conducta dels diferents estats:
 - La conducta de cada estat està en una operació diferent.
- Localitza (en un sol lloc) la conducta específica d'un estat.
- Es poden definir fàcilment nous estats, sense alterar els existents.
- Fa explícites les transicions d'estat:
 - Cada estat té associat un objecte 'estat' diferent.
- Si els objectes estat no tenen atributs/associacions propis, poden ser compartits per diversos contextos.

Exemple 2: Problema

- Esquema conceptual de l'especificació:

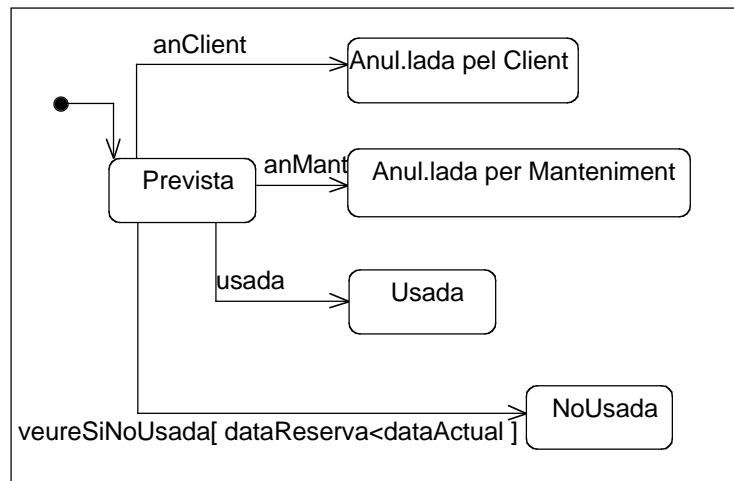


- Es vol dissenyar una operació de la classe d'objectes Reserva que calcula el cost d'una reserva.
 - cost= 0 si la reserva està prevista o anul.lada per manteniment.
 - cost= preuHoraPista (de Club) si la reserva està usada.
 - cost= preuNoÚs (de Club) si la reserva està no usada.
 - cost= preuAnul.lació (de Club) si la reserva està anul.lada pel client.



Aplicació del patró Estat

Exemple 2: Diagrama d'estats de Reserva

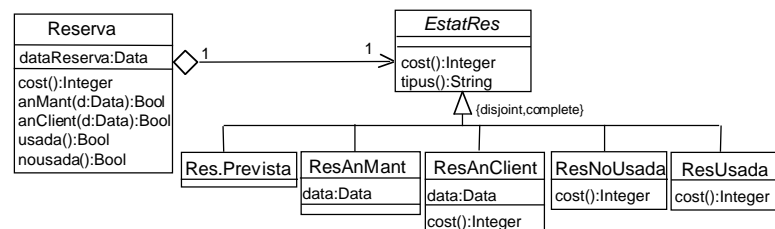


Exemple 2: Solució

- Diagrama de classes de disseny:

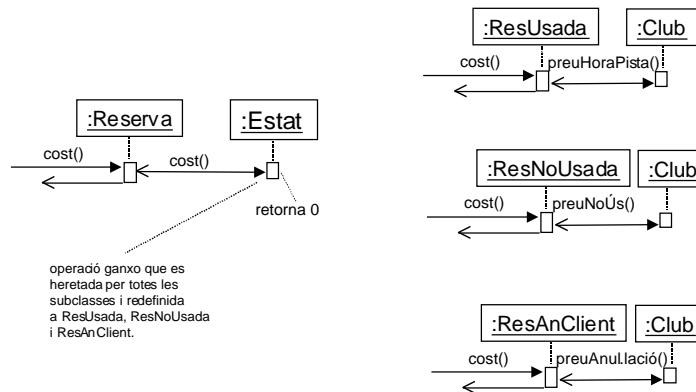
Hipòtesis d'aplicació del patró Estat:

- Objectes estat no compartits.
- Transicions d'estat definides en l'objecte context.
- Creació/destrucció dels objectes estat quan es necessiten.



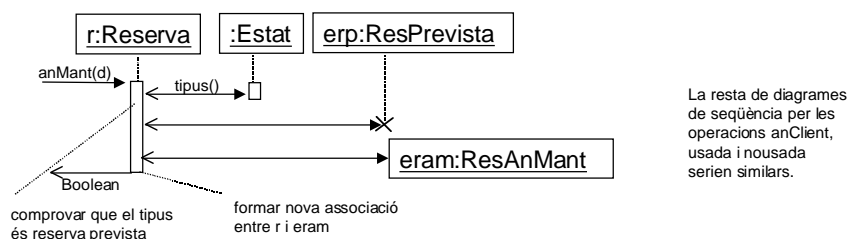
Exemple 2: Solució

- Operació **cost():Integer** : Diagrames de seqüència:



Exemple 2: Solució

- Operació **anMant(d:Data)** : Diagrames de seqüència:



Bibliografia

- *Elements of Reusable Object-Oriented Software.*
E. Gamma; R. Helm; R. Johnson; J. Vlissides
Addison-Wesley, 1995, pp. 305-313.
- *Applying UML and Patterns.*
An Introduction to Object-Oriented Analysis and Design.
C. Larman
Prentice Hall, 1998, pp. 406-410.
- <http://www.eli.sdsu.edu/courses/spring98/cs635/notes/state>

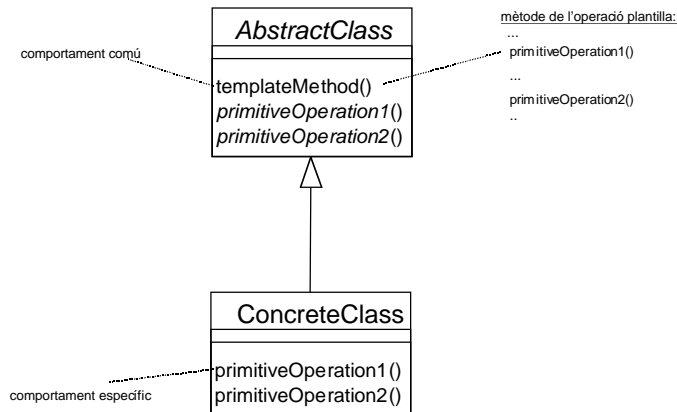
Patró de disseny Mètode Plantilla

- Descripció general.
- Solució: Estructura.
- Exemple 1.
- Conseqüències.
- Extensió Exemple 1.
- Exemple 2.
- Bibliografia.

Descripció general

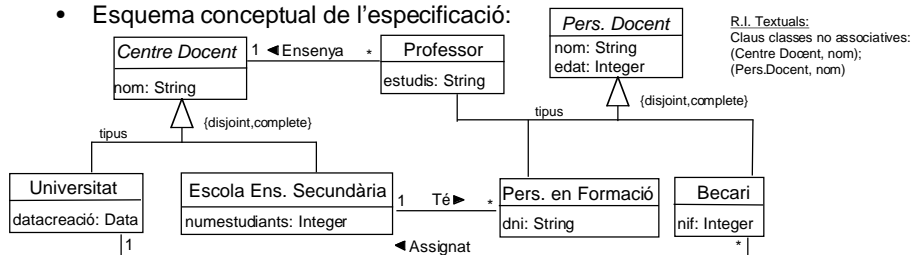
- **Context:**
 - La definició d'una operació en una jerarquia té un comportament comú a totes les subclasses i un comportament específic per cadascuna d'elles.
- **Problema:**
 - Repetir el comportament comú a totes les subclasses implica duplicació de codi i un manteniment més costós.
- **Solució:**
 - Definir l'algorisme (l'operació) en la superclasse, invocant operacions abstractes (amb signatura definida en la superclasse i mètode a les subclasses) que són redefinides a les subclasses.
 - L'operació de la superclasse defineix la part del comportament comú i les operacions abstractes la part del comportament específic.

Solució: Estructura



Exemple 1: Problema

- Esquema conceptual de l'especificació:



Nota: Suposem que els Centres Docents i el Personal Docent no poden canviar de subclasse

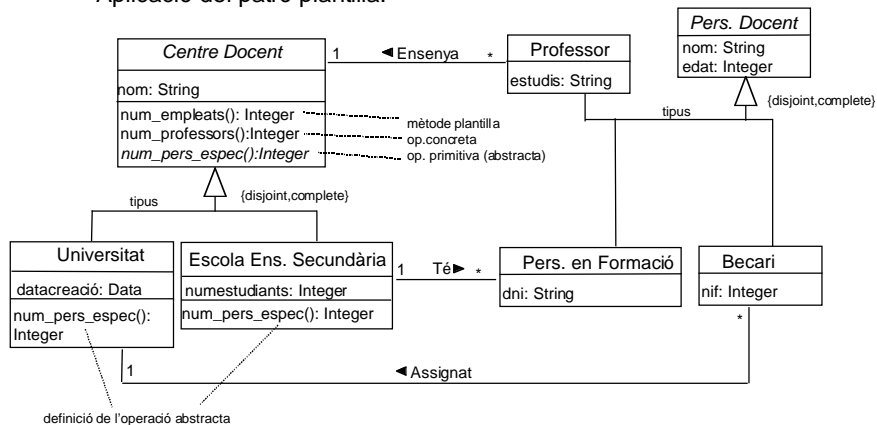
- Es vol dissenyar una operació de la classe d'objectes Centre Docent per calcular el nombre d'empleats que té un centre docent que tenen una edat < 25 anys.
 - num_empleats=num_professors+num_becaris amb edat < 25 anys si el centre és una Universitat
 - num_empleats=num_professors+num_pers_en_formació amb edat < 25 anys si el centre és una Escola d'Ensenyança Secundària.



Aplicació del patró Plantilla

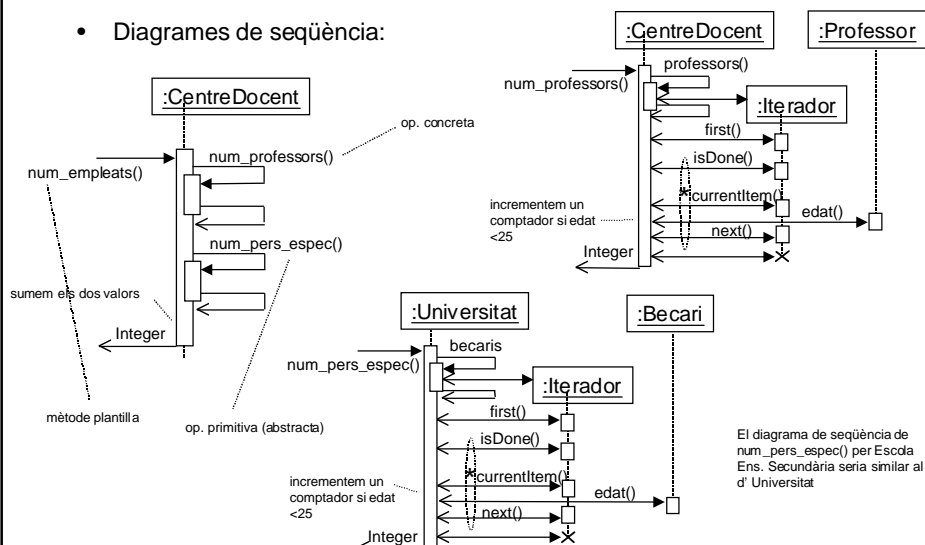
Exemple 1: Solució (I)

- Aplicació del patró plantilla:



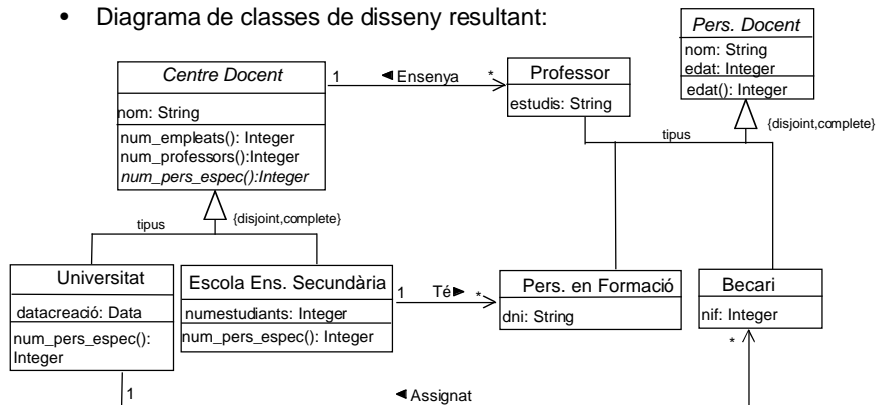
Exemple 1: Solució (II)

- Diagrames de seqüència:



Exemple 1: Solució (III)

- Diagrama de classes de disseny resultant:

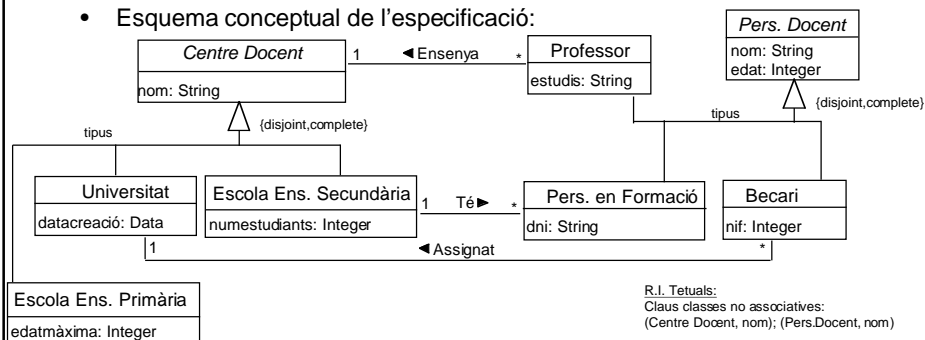


Conseqüències

- Tècnica fonamental per a la reutilització de codi.
- Els mètodes plantilla porten a una estructura de control invertida:
 - Principi de Hollywood.
- Els mètodes plantilla acostumen a cridar operacions dels següents tipus:
 - Operacions concretes de la AbstractClass.
 - Operacions primitives (és a dir, operacions abstractes).
 - Operacions *ganxo*, que proporcionen conducta per defecte (normalment, res), que les subclasses poden redefinir, si cal.
 - Operacions concretes (en ConcreteClass o en classes client).
- És important que els mètodes plantilla especifiquin:
 - quines operacions són *ganxos* (poden ser redefinides).
 - Quines operacions són abstractes (han de ser redefinides):
 - convé que n'hi hagin com menys millor.

Extensió Exemple 1: Problema

- Esquema conceptual de l'especificació:



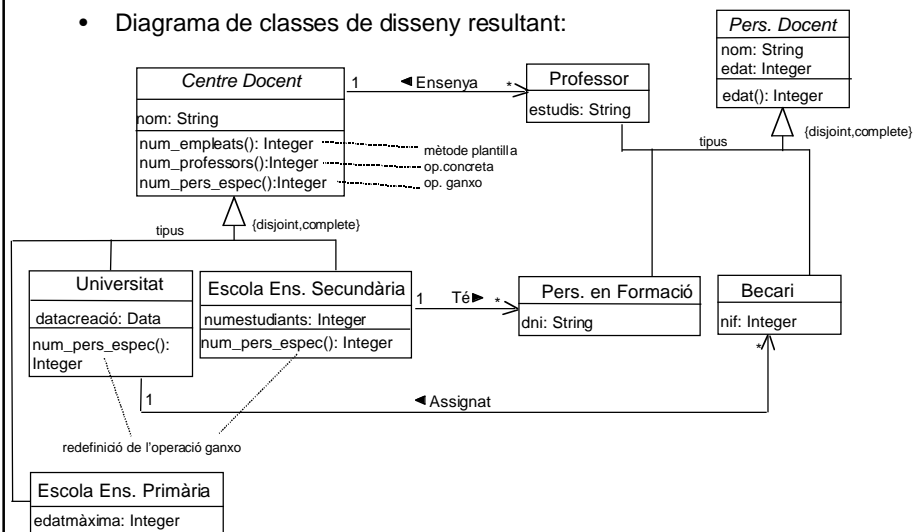
- Es vol dissenyar la mateixa operació nombre d'empleats que té un centre docent.
 - num_empleats d'Universitat i d'Escola Ensenyança Secundària igual que l'exemple anterior.
 - num_empleats=num_professors amb edat < 25 anys si el centre és una Escola d'Ensenyança Primària.



Aplicació del patró Plantilla

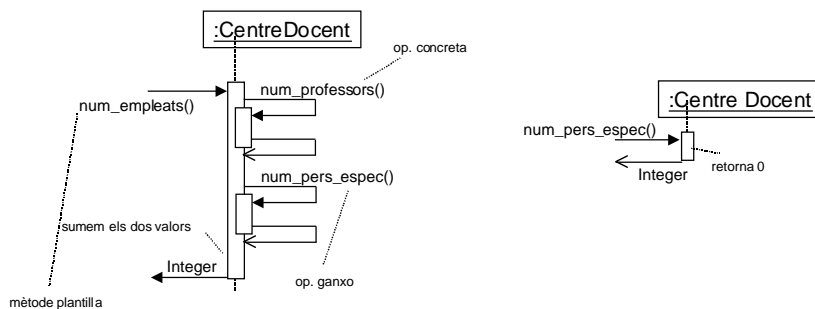
Extensió Exemple 1: Solució (I)

- Diagrama de classes de disseny resultant:



Extensió Exemple 1: Solució (II)

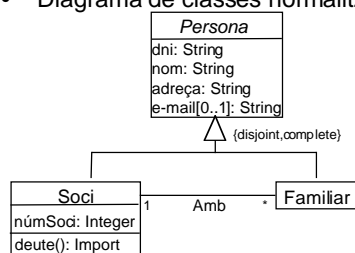
- Diagrames de seqüència:



El diagrames de seqüència de `num_pers_espec()` per Escola Ens. Secundària i Universitat i `num_professors()` per Centre Docent serien iguals als de l'exemple anterior

Exemple 2: Problema

- Diagrama de classes normalitzat:



R.I. Tetuals:

- Claus classes no associatives: (Persona,dni)
- No poden existir dos socis amb el mateix numSod.

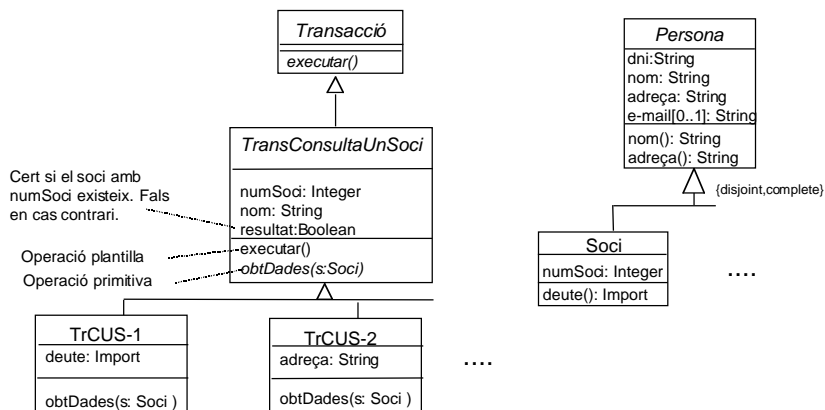
- Es vol dissenyar dos casos d'ús amb un únic esdeveniment per consultar les dades d'un soci:
 - El primer cas d'ús obté el nom i el deute d'un soci a partir del `numSod` en cas d'existir, i retorna fals en cas contrari.
 - El segon obté el nom i l'adreça també a partir del `numSod` en cas d'existir, i retorna fals en cas contrari.



Aplicació del patró Plantilla

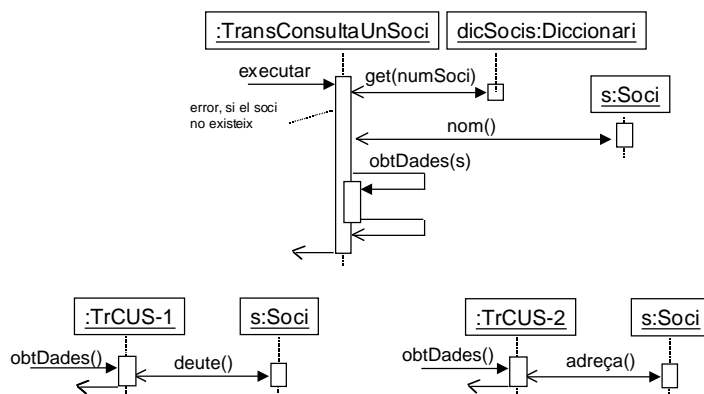
Exemple 2: Solució

- Diagrama de classes de disseny:



Exemple 2: Solució

- Diagrames de seqüència:



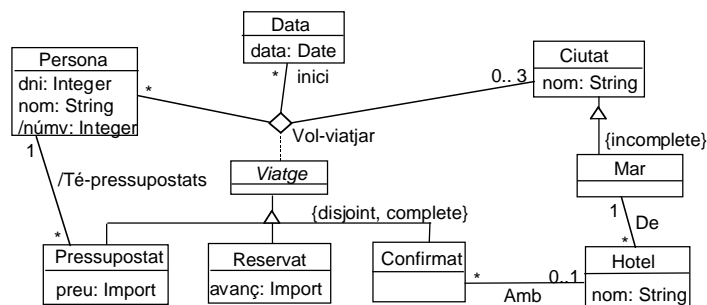
Bibliografia

- *Elements of Reusable Object-Oriented Software.*
E. Gamma; R. Helm; R. Johnson; J. Vlissides
Addison-Wesley, 1995, pp. 325-330.
- *Applying UML and Patterns.*
An Introduction to Object-Oriented Analysis and Design.
C. Larman
Prentice Hall, 1998, cap. 38.11.
- <http://www.ei.sdsu.edu/courses/spring98/cs635/notes/template>

Exemple Capa de Domini

- Especificació:
 - Esquema Conceptual
 - Cassos d'Ús
 - Diagrama d'Estats
- Normalització:
 - Diagrama de classes normalitzat
 - Contractes operacions normalitzats
- Aplicació del Patró Estat
- Aplicació del Patró Controlador
- Aplicació del Patró Expert
- Diagrames de Seqüència
- Diagrama de Classes de Disseny

Especificació: Esquema Conceptual



R.I. Textuals:

- Claus classes no associatives: (Persona, dni); (Data, data); (Ciutat, nom)
- Una Ciutat no pot tenir més d'un Hotel amb el mateix nom
- Un viatge confirmat es fa a un Hotel de la mateixa ciutat on es fa el viatge
- Una persona no pot tenir més d'un viatge confirmat amb una mateixa data d'inici

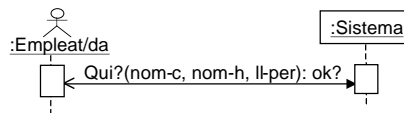
Informació derivada:

- númv: és el nombre de viatges confirmats d'aquella Persona
- Té-pressupostats: és igual al conjunt de viatges pressupostats d'una Persona

Suposicions:

- L'atribut derivat númv s'ha de calcular i l'associació derivada Té-pressupostats s'ha de materialitzar.
- Les associacions Amb i Té-pressupostats tenen navegabilitat doble

Especificació: Cas d'Ús "Quines-Persones"



Operació: qui? (nom-c: String, nom-h: String, out ll-per: Llista-noms) Llista-noms= { nom }

Classe retorn: Boolean

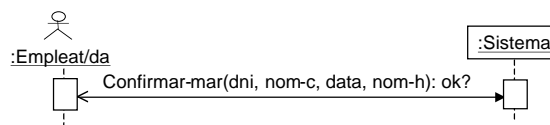
Precondicions: Els arguments han de tenir valor

Semàntica: Retorna la llista de noms de persones que s'han allotjat alguna vegada a l'hotel nom-h de la ciutat nom-c.

Postcondicions:

1. Si l'hotel nom-h no és de la ciutat nom-c o bé no hi cap persona que s'hi hagi allotjat, llavors operació invàlida i es retorna Fals
2. En cas contrari, operació vàlida, es retorna Cert i la llista que conté els noms de totes les persones que s'han allotjat alguna vegada a l'hotel.

Especificació: Cas d'Ús "Confirmar-Viatge-al-Mar"



Operació: confirmar-mar (dni: Integer, nom-c: String, data: Date, nom-h: String)

Classe retorn: Boolean

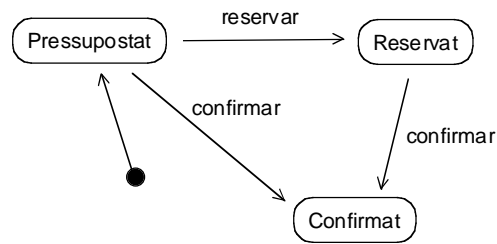
Precondicions: Els arguments han de tenir valor

Semàntica: Confirmar un viatge a una ciutat de mar i assignar-li l'hotel

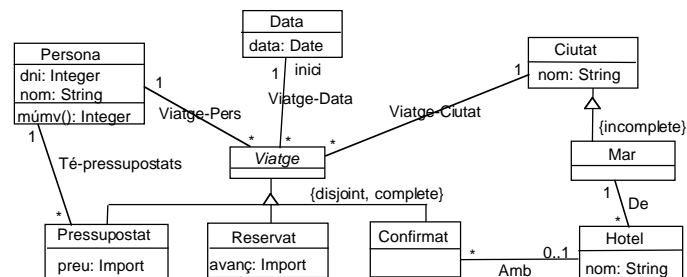
Postcondicions:

1. En el següents casos, operació invàlida i retorna Fals:
 - 1.1 Si la ciutat nom-c no és de mar
 - 1.2 Si no existeix un viatge de la persona a la ciutat i per a la data especificats
 - 1.3 L'hotel nom-h no és de la ciutat nom-c
2. En cas contrari, operació vàlida, es retorna Cert i:
 - 2.1 El viatge passa a estar Confirmat
 - 2.2 Es crea una nova ocurrència de l'associació *Amb* entre el viatge confirmat i l'Hotel

Especificació: Diagrama d'Estats



Normalització: Diagrama de classes normalitzat



R.I. Textuals:

- Claus d'associació no associatives: (Persona, dni); (Data, data); (Ciutat, nom)
- Una Ciutat de Mar no pot tenir més d'un Hotel amb el mateix nom
- Un viatge confirmat es fa a un Hotel de la mateixa ciutat on es fa el viatge
- Una persona no pot tenir més d'un viatge confirmat amb una mateixa data d'inici
- **Per una persona i una data no poden existir més de tres viatges a ciutats diferents**
- **No poden existir dos viatges de la mateixa persona a la mateixa ciutat amb la mateixa data d'inici**

Inf. derivada:

- númv: és el nombre de viatges confirmats d'aquella Persona
- Té-pessupostats: és igual al conjunt de viatges pressupostats d'una Persona

Suposicions:

- Les associacions *Amb* i *Té-pessupostats* tenen navegabilitat doble

Normalització: Contracte normalitzat “qui?”

Operació: qui? (nom-c: String, nom-h: String, out ll-per: Llista-noms) **(No es modifica)**

Classe retorn: Booleà

Precondicions: Els arguments han de tenir valor

Semàntica: Retorna llista de noms de persones que s'han allotjat alguna vegada a l'hotel nom-h de la ciutat nom-c.

Postcondicions:

1. Si l'hotel nom-h no és de la ciutat nom-c o bé no hi cap persona que s'hi hagi allotjat, llavors operació invàlida i es retorna Fals
2. En cas contrari, operació vàlida, es retorna Cert i la llista que conté els noms de totes les persones que s'han allotjat alguna vegada a l'hotel.

Normalització: Contracte normalitzat “confirmar-mar”

Operació: confirmar-mar (dni: Integer, nom-c: String, data: Date, nom-h: String)

Classe retorn: Boolean

Precondicions: Els arguments han de tenir valor

Semàntica: Confirmar un viatge a una ciutat de mar i assignar-li l'hotel

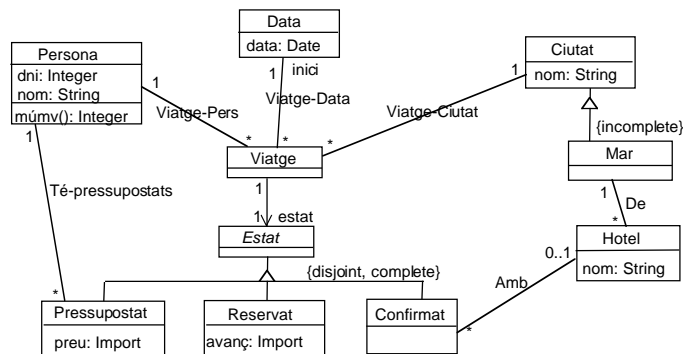
Postcondicions:

1. En el següents casos, operació invàlida i retorna Fals:
 - 1.1 La ciutat nom-c no és de mar
 - 1.2 No existeix un viatge de la persona a la ciutat per a la data especificats
 - 1.3 L'hotel nom-h no és de la ciutat nom-c
 - 1.4 **Ja existeix un viatge confirmat de la persona i data especificats**
2. En cas contrari, operació vàlida, es retorna Cert i:
 - 2.1 El viatge passa a estar Confirmat
 - 2.2 Es crea una nova ocurrència de l'associació *Amb* entre el viatge confirmat i l'Hotel
 - 2.3 **Si el viatge era Pressupostat, s'elimina la instància de l'associació Té-pressupostat entre Pressupostat i Persona**

Aplicació del Patró Estat

Supòsits en l'aplicació del Patró Estat

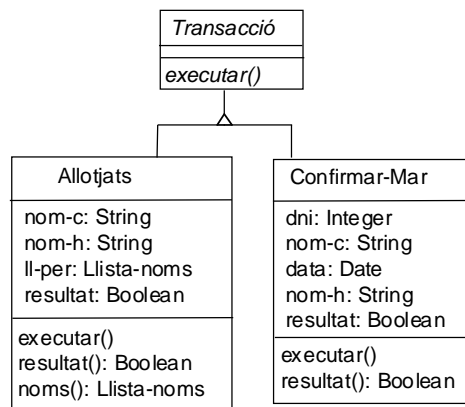
- Objectes estat no compartits
- Transicions d'estat definides als objectes estat
- Creació i destrucció d'objectes estat quan calgui



Suposicions:

- Les associacions *Amb* i *Té-pressupostats* tenen navegabilitat doble

Aplicació del Patró Controlador



qui? (nom-c: String, nom-h: String,
out ll-per: Llista-noms): Boolean

confirmar-mar (dni: Integer, nom-c: String, data: Date,
nom-h: String): Boolean

Aplicació Patró Expert qui? (nom-c, nom-h, out ll-per)

Experts

Operació: qui? (nom-c: String, nom-h: String, out ll-per: llista-noms)

Classe retorn: Booleà

Precondicions: Els arguments han de tenir valor

Semàntica: Retorna llista de noms de persones que s'han allotjat alguna vegada a l'hotel nom-h de la ciutat nom-c.

Postcondicions:

- En els següents casos, operació invàlida i es retorna fals:
 - Si l'hotel nom-h no és de la ciutat nom-c
 - O bé si no hi cap persona que s'hi hagi allotjat
- En cas contrari, operació vàlida, es retorna Cert i la llista que conté els noms de totes les persones que s'han allotjat alguna vegada a l'hotel

DicHotels

Allotjats, Hotel

Allotjats, Hotel

Diagrama Seqüència qui? (nom-c, nom-h, out ll-per)

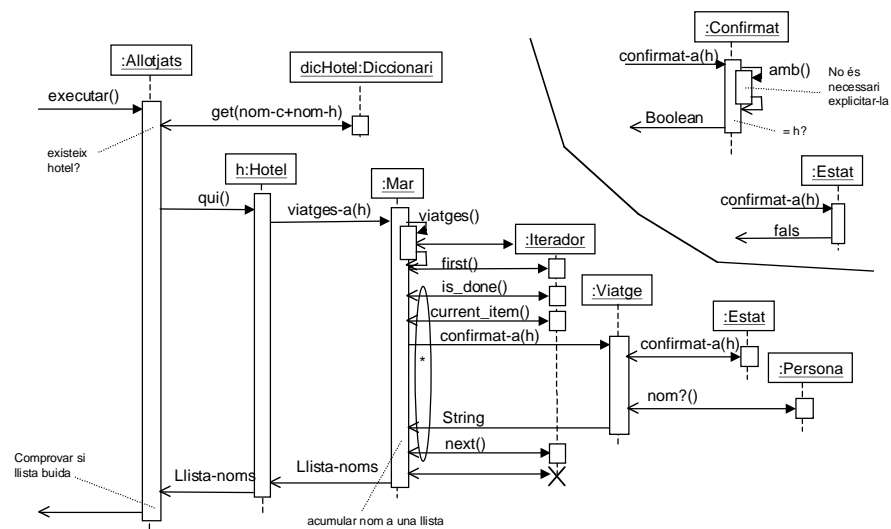
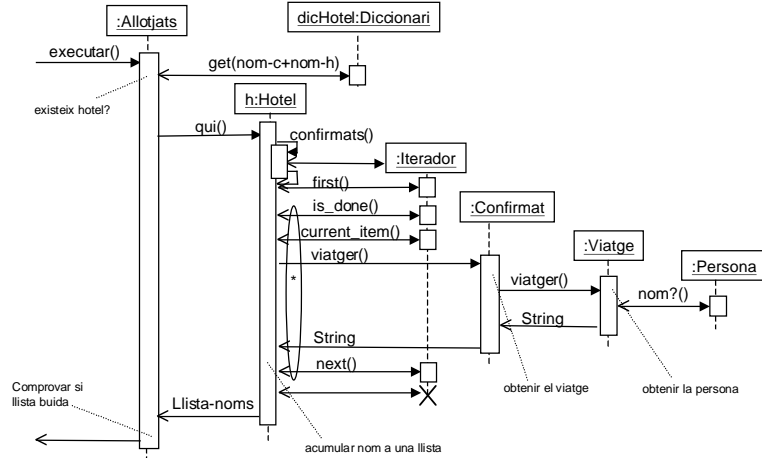


Diagrama Seqüència (alternativa) qui? (nom-c, nom-h, out ll-per)



Modificant el patró Estat, definint la navegabilitat de l'associació entre Viatge i Estat doble, es pot definir l'operació qui? d'una forma més eficient.

Aplicació Patró Expert confirmar-mar (dni, nom-c, data, nom-h)

Experts

Operació: confirmar-mar (dni: Enter, nom-c: String, data: Dia, nom-h: String)

Classe retorn: Booleà

Precondicions: Els arguments han de tenir valor

Semàntica: Confirmar un viatge a una ciutat de mar i assignar-li l'hotel

Postcondicions:

1. En el següents casos, operació invàlida i retorna Fals:

- 1.1 Si la ciutat nom-c no és de mar
- 1.2 Si no existeix un viatge de la persona a la ciutat per a la data especificats
- 1.3 L'hotel nom-h no és de la ciutat nom-c
- 1.4 Ja existeix un viatge confirmat de la persona i data especificats

Ciutat, dicMar

dicViatges

dicHotels

Persona, dicConfirmats

2. En cas contrari, operació vàlida, es retorna Cert i:

- 2.1 El viatge passa a estar Confirmat
- 2.2 Es crea una nova ocurrència de l'associació *Amb* entre el viatge confirmat i l'Hotel
- 2.3 Si el viatge era Pressupostat, s'elimina l'associació Té-pressupostat amb Persona

Estat + Viatge

Confirmat + Hotel

Pressupostat + Persona

Diagrama Seqüència (I) confirmar-mar(dni, nom-c, data, nom-h)

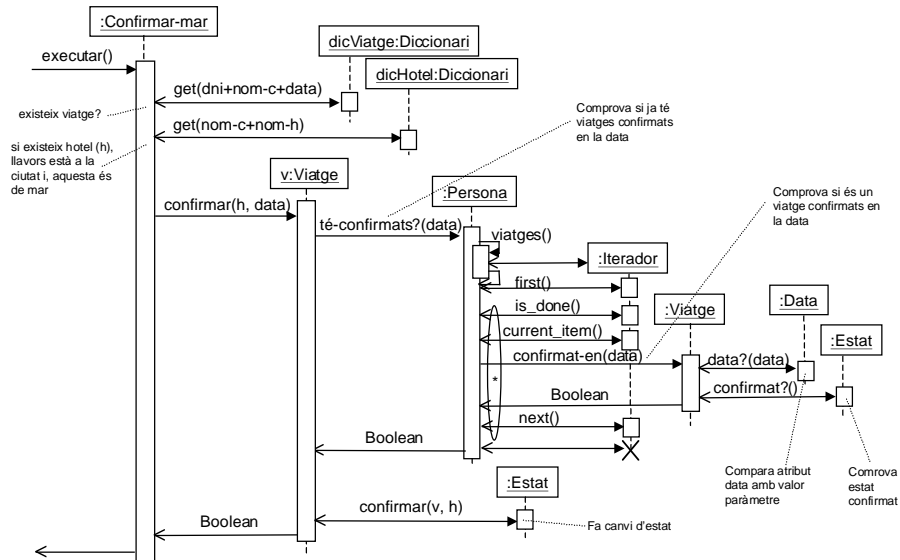


Diagrama Seqüència (II) confirmar-mar(dni, nom-c, data, nom-h)

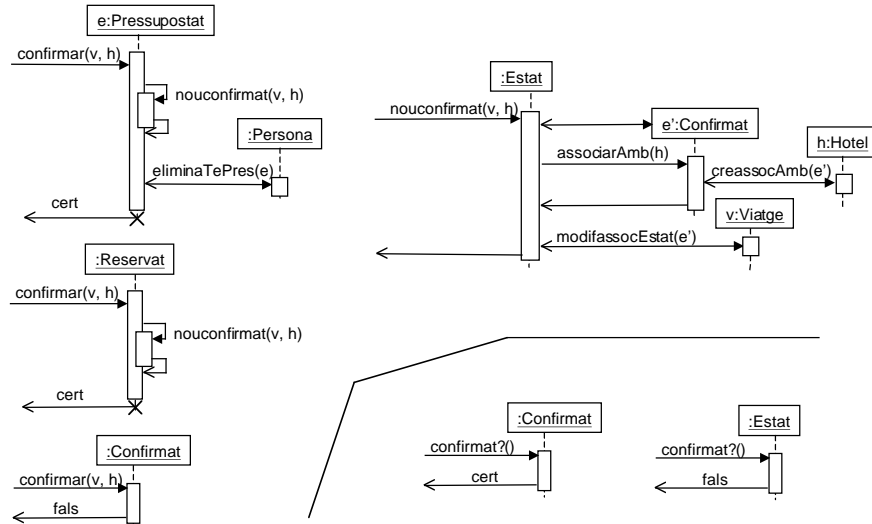
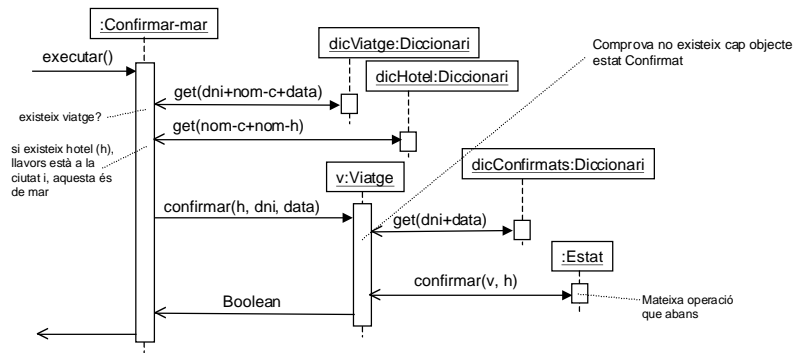


Diagrama Seqüència (I) (alternativa) confirmar-mar(dni, nom-c, data, nom-h)



La restricció "Una persona no pot tenir més d'un viatge confirmat amb la mateixa data d'inici" permet identificar els viatges confirmats amb (dni+data).

Diagrama Seqüència (II) (alternativa) confirmar-mar(dni, nom-c, data, nom-h)

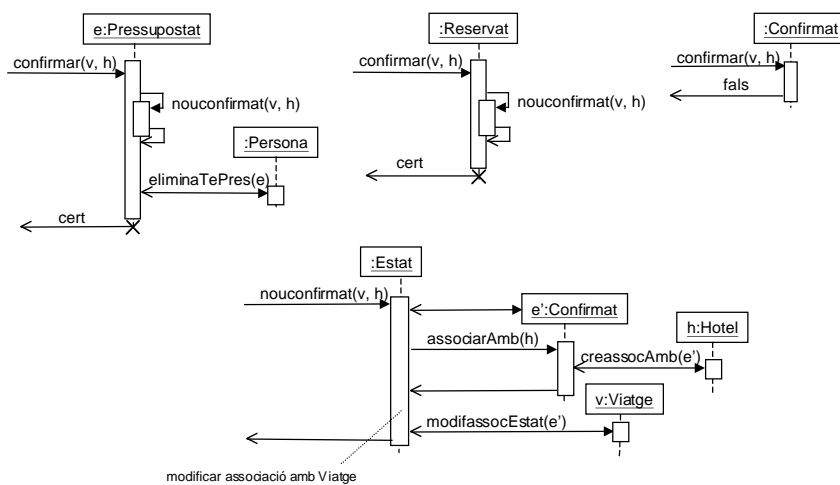


Diagrama de Classes de Disseny

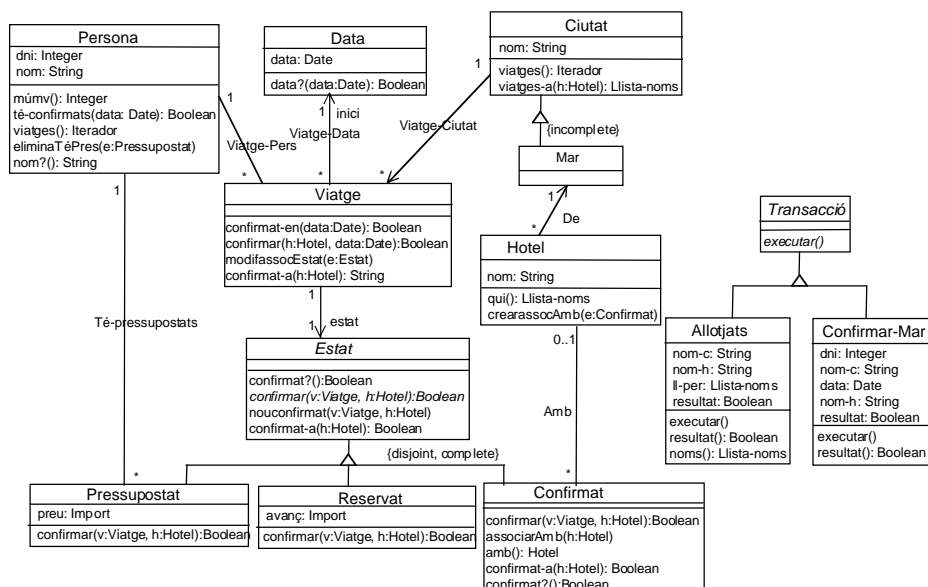
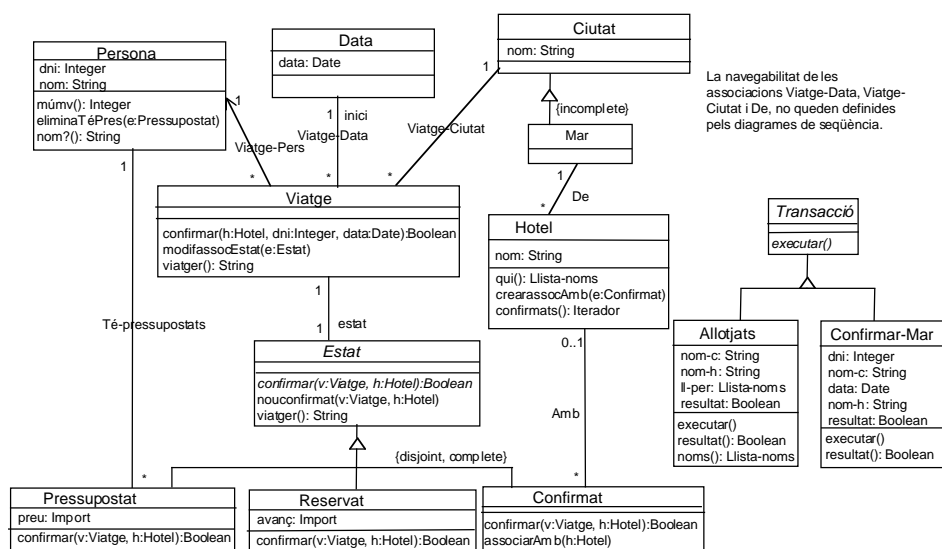


Diagrama de Classes de Disseny (alternativa)



Capa Presentació

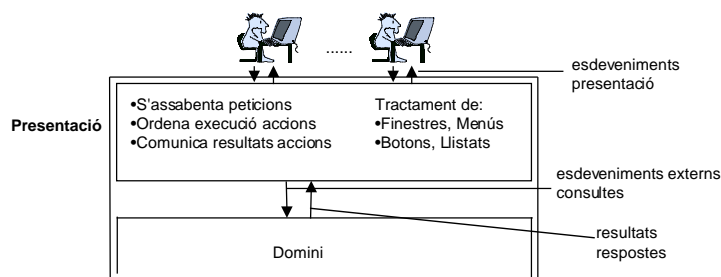
- Introducció
- Patró Model-Vista-Controlador

Introducció

- Què és i què inclou la Capa Presentació
- Disseny de la Capa Presentació:
 - Disseny Extern
 - Disseny Intern
- Punt de Partida
- Bibliografia

Què és i què inclou la Capa Presentació

- Capa Presentació: és el component del sistema software encarregat de gestionar la interacció amb l'usuari.



Què és i què inclou la Capa Presentació

- La interacció home-màquina consisteix en:
 - L'usuari té un objectiu i un pla d'acció.
 - L'usuari tradueix aquest pla amb accions concretes sobre la interfície del sistema software.
 - El sistema processa els inputs rebuts. Aquest processament consisteix en accedir, calcular i donar format a les dades.
 - A cada input rebut, el sistema respon presentant el resultat de les accions i encaminant l'usuari a la següent acció.
- La Capa Presentació inclou els elements necessaris per a:
 - rebre les ordres o peticions de l'usuari
 - mostrar a l'usuari el resultat d'aquestes peticions
 - resoldre o coordinar la resolució de les peticions amb col·laboració de la Capa Domini
- Interfícies:
 - Línies de comandes: Sistema porta el control
 - GUI (Basades en Esdeveniments): Usuari porta el control

Disseny de la Capa Presentació

- Disseny Capa Presentació és un procés, inclòs en l'etapa de disseny d'un sistema software, encarregat de definir:
 - com l'usuari interacciona amb el sistema software (**Disseny Extern**)
 - com la Capa Presentació interacciona amb la Capa Domini (**Disseny Intern**)
- Punt de partida pel disseny de la Capa Presentació:
 - Característiques tecnològiques perifèrics entrada (teclat, ratolí, ...) i perifèrics sortida (pantalla, impressora, ...)
 - Anàlisi d'usuaris i tasques → **Casos d'Ús**
- Procés de disseny basat en el prototipatge
- Equip de dissenyadors:
 - Coneixement del domini del sistema → participació usuari final
 - Coneixements en orientació a objectes → programadors
 - Coneixements en sociologia, psicologia i fisiologia → psicòlegs, ...
 - Coneixements en medis presentació informació → dissenyadors gràfics, ...

Disseny Extern

- Disseny Extern (de la Capa Presentació) consisteix en el disseny dels elements (tangibles) que l'usuari veu, sent i toca al interaccionar amb el sistema.
- El disseny extern consisteix en la definició de:
 - Mecanismes com l'usuari pot demanar peticions al sistema → **Mecanismes d'interacció**
 - Formes com es poden mostrar els resultats d'aquestes peticions a l'usuari → **Presentació de la informació**
- Exemples:
 - Mecanismes interacció: escriure comandes, tecles funció, apuntar objectes/menús amb ratolí o pantalla tàctil, comandes orals ...
 - Presentació de informació: formats gràfics, imatges, textual, vídeo; presentació a pantalla o en llistat imprès; ...

Disseny Extern: Mecanismes d'interacció

- Principis bàsics pel disseny de la interacció amb el sistema:
 - Integritat conceptual en la interacció (seqüència d'accions, abreviatures, ...)
 - Mínim nombre d'accions a realitzar (selecció en lloc d'introducció de dades, evitar introducció dades redundants, ...)
 - Memòria a curt termini usuari mínima (no cal recordar llistes de codis o comandes complexes, ...)
 - Compatibilitat entre presentació informació i dades introduïdes (format entrada dades similar al de presentació, ...)
 - Flexibilitat en la introducció de dades per part de l'usuari (permetre entrada dades en una seqüència diferent per a usuaris experts, ...)
 - Donar resposta a cada requesta de l'usuari.

Disseny Extern: Presentació de la Informació

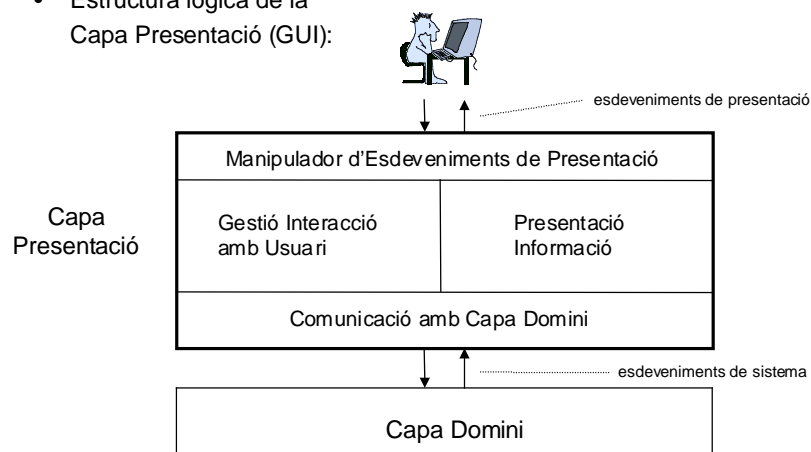
- Principis bàsics pel disseny de la presentació d'informació:
 - Integritat conceptual en la presentació (formats, colors, terminologia, ...)
 - Assimilació eficient de la informació per part de l'usuari (format de presentació familiar a l'usuari)
 - Memòria a curt termini usuari mínima (no cal recordar informació de pantalles anteriors)
 - Compatibilitat entre presentació informació i dades introduïdes (mateixos camps d'entrada i sortida)
 - Flexibilitat per controlar la presentació d'informació per part de l'usuari (permetre canvis en formats, ordre, ...)
 - Missatges d'error clars, informatius i orientatius.

Disseny Intern

- Disseny Intern (de la Capa Presentació) consisteix en dissenyar els mecanismes que recullen, processen i donen resposta a les requestes de l'usuari.
- El disseny intern i disseny extern es realitzen en paral·lel o iterativament.
- En una arquitectura lògica en tres capes cal definir, a més a més, la comunicació entre Capa Presentació i Capa Domini.
- Disseny Intern =>
 - Disseny mecanismes gestionen la Interacció
 - Disseny mecanismes permeten Presentació Informació
 - Disseny mecanismes de comunicació Capa Presentació i Capa Domini

Disseny Intern

- Estructura lògica de la Capa Presentació (GUI):



Disseny Intern

- Manipulador d'Esdeveniments de Presentació
 - Interfícies d'Usuari Gràfiques (GUI) basades en esdeveniments.
 - Com es comuniquen aquests esdeveniments a la Capa Presentació?.
 - Comunicació del sistema software amb el sistema operatiu.
- Gestió Interacció amb Usuari
 - Controlar la recepció d'esdeveniments de presentació del manipulador.
 - Processar aquests esdeveniments i generar esdeveniments de sistema als objectes que els han de processar.
- Presentació Informació
 - Recepció de les dades a presentar a l'usuari (pantalla, impressora).
 - Presentació de les dades en els formats específics de l'usuari.
- Comunicació amb Capa Domini
 - Enviar esdeveniments de sistema a processar.
 - Rebre respostes a aquests esdeveniments.

Disseny Intern

- Manipulador d'Esdeveniments de Presentació
 - Gestió d'Esdeveniments
- Gestió Interacció amb Usuari
 - Patró Model-Vista-**Controlador**
- Presentació Informació
 - Patró Model-**Vista**-Controlador
- Comunicació amb Capa Domini
 - Patró **Observador**
 - **Event Delegation Model** de JAVA 1.1

Punt de Partida

- Descripció dels Casos d'Ús.
- Cal adaptar-los al disseny (de pantalles i finestres) definits durant el Disseny Extern.

Cas d'Ús : Admetre Soci

Actor: Administrador

Objectiu: Admetre un soci i els seus familiars com a membres del club.

Descripció General: . . .

Seqüència d'esdeveniments:

Acció

1. Introduir nom, adreça i e-mail del soci

4. Introduir numero de compte on carregar pagament de rebuts.

6. Introducció nom, adreça i e-mail dels familiars

Resposta

2. Se li assigna número de soci

3. Es dona d'alta com a nou soci

5. Es guarda el compte on carregar rebuts

7. Es donen d'alta els seus familiars

Punt de Partida: Disseny Extern

Alta de soci	
Nom	<input type="text"/>
adreça	<input type="text"/>
e-mail	<input type="text"/>
<input type="button" value="endavan"/> <input type="button" value="plega"/>	

Número de Soci	
Numero Soci	<input type="text"/>
<input type="button" value="Ok"/>	

Assignació Compte	
Numero Compte Corrent	<input type="text"/>
<input type="button" value="endavan"/> <input type="button" value="plega"/>	

Punt de Partida: Cas d'ús dissenyat

Cas d'Ús : Admetre Soci

...

Acció

Resposta

1.1. L'administrador introdueix el nom del soci a A i introdueix l'adreça a B. En cas de tenir una adreça E-mail, aquesta s'introdueix a C.

1.2. Si es vol donar d'alta aquest nou soci, cal pitjar el botó Endavant. En cas de voler cancel·lar l'alta, cal prémer el botó Plegar. El botó Plegar, finalitza la admissió d'un soci.

1.3. Cal prémer el botó Ok per continuar amb l'admissió del soci.

4.1. En aquesta finestra l'administrador introduirà el compte corrent a (D), on cobrar els rebuts pendents. Si es prem, el botó Plegar, es retorna a la finestra anterior per a donar d'alta un nou soci.

2.1. El sistema genera un nou número de soci que guarda amb les dades anteriors.
2.2. El sistema mostra el número de soci en una finestra auxiliar.

3.1 Es dona d'alta com a nou soci
3.2. Apareix la finestra Assignar Compte Corrent.

....

Bibliografia

- Collins, D.
Designing Object-Oriented User Interfaces.
Benjamin/Cummings Publishing Company, 1995.
(Cap. 1, 5, 6, 11)
- Shneiderman, B.
Designing the User Interface.
Strategies for Effective Human-Computer Interaction (3ª edició).
Addison-Wesley, 1998. (Cap. 2)
- Larman, C.
Applying UML and Patterns.
An Introduction to Object-Oriented Analysis and Design.
Prentice Hall, 1998. (Cap. 16)

Patró Model-Vista-Controlador

- Descripció General
- Solució: Estructura
- Comportament
- Conseqüències
- Implementació
- Exemple
- Bibliografia

Descripció General (I)

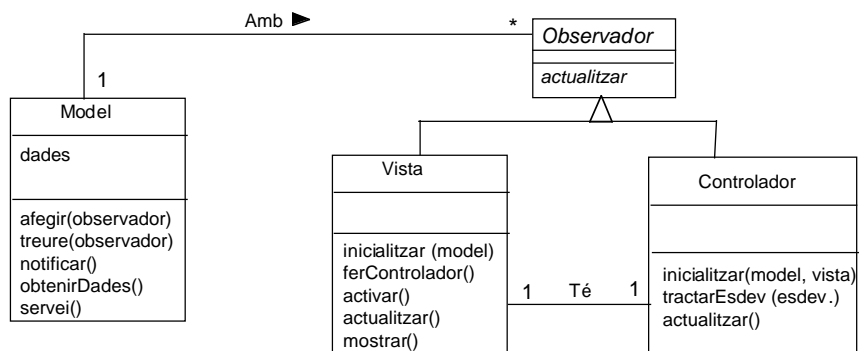
El Patró Model-Vista-Controlador és un patró arquitectònic

- **Context:**
 - Sistemes software interactius amb interfícies d'usuari flexibles.
- **Problema:**
 - La informació proporcionada pel sistema software i el seu comportament han de reflectir immediatament les manipulacions de la informació.
 - Pot interessar mostrar la mateixa informació a diverses finestres en diferents formats.
 - S'hauria de poder canviar la interfície d'usuari quan el sistema software ja està implantat sense que això afectés el codi del nucli del sistema.

Descripció General (II)

- **Solució:**
 - Descomposar el sistema en tres components:
 - Model (procés): inclou la implementació de les funcionalitats i les dades del sistema.
 - Vista (sortida): mostra la informació a l'usuari final
 - Controlador (entrada): responsable de gestionar la interacció amb l'usuari
- La *Capa de Presentació* inclou els components:
 - Vista: presentació d'informació
 - Controlador: mecanisme d'interacció
- El Model representa la *Capa de Domini* (i *Gestió de Dades*) del sistema

Solució: estructura estàtica

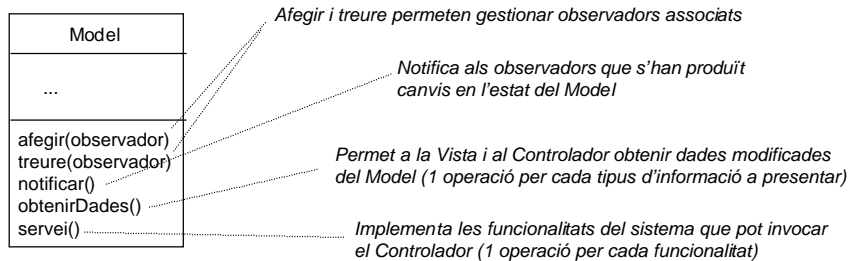


- En general, hi ha *tantes parelles Vista-Controlador com informacions i formats diferents es vulguin mostrar* d'un model determinat.
- Les operacions *actualitzar* (dels controladors), *mostrar* i *tractarEsdev* s'acostumen a *redefinir* a les subclasses de vista i controlador que es defineixin per a un model determinat

Descripció General: Model

- **Model:**

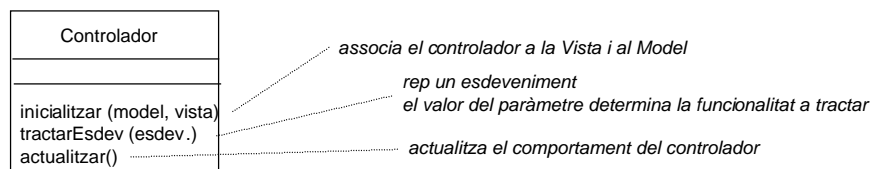
- Encapsula les funcionalitats i les dades del sistema.
- És independent dels mecanismes de presentació d'informació i d'interacció amb l'usuari.
- Proporciona al Controlador els serveis per satisfer les peticions de l'usuari.
- Manté un mecanisme de coordinació amb les Vistes i Controladors associats (patró Observador), per notificar-los qualsevol canvi en el seu estat.



Descripció General: Controlador

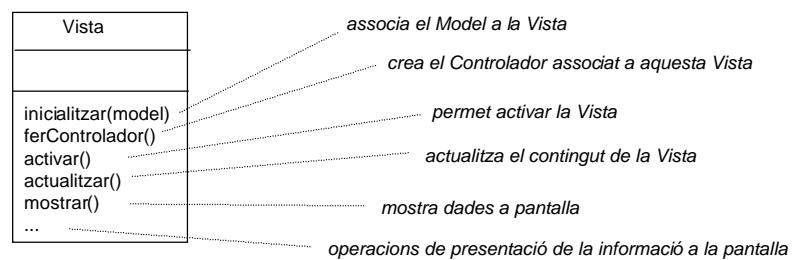
- **Controlador:**

- L'usuari interactua amb el sistema únicament mitjançant controladors
- Gestiona els esdeveniments de presentació i de modificació del model generats per l'usuari.
- La forma com rep aquests esdeveniments depèn de la plataforma utilitzada per interactuar amb l'usuari (Manipulador d'esdeveniments).
- Tradueix els esdeveniments de presentació en:
 - invocacions a serveis proporcionats pel Model.
 - peticions de funcionalitats pròpies de la Vista.
- El comportament del Controlador pot dependre de l'estat del Model.

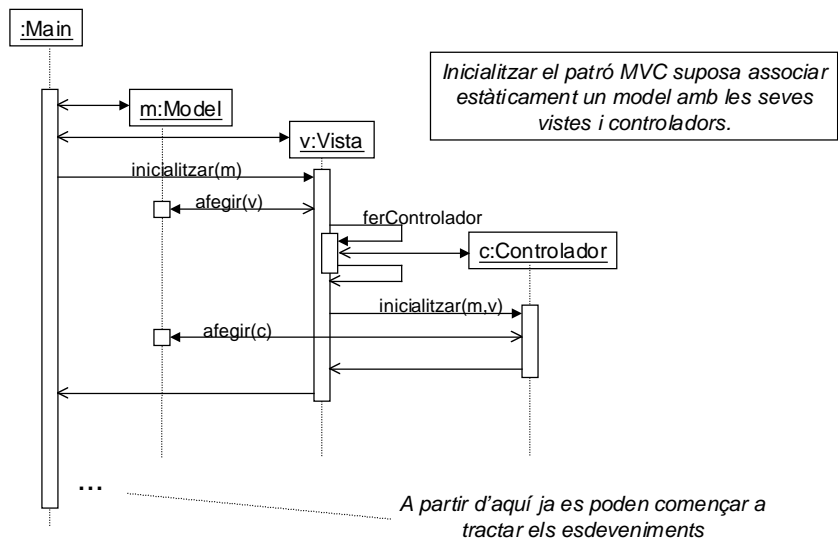


Descripció General: Vista

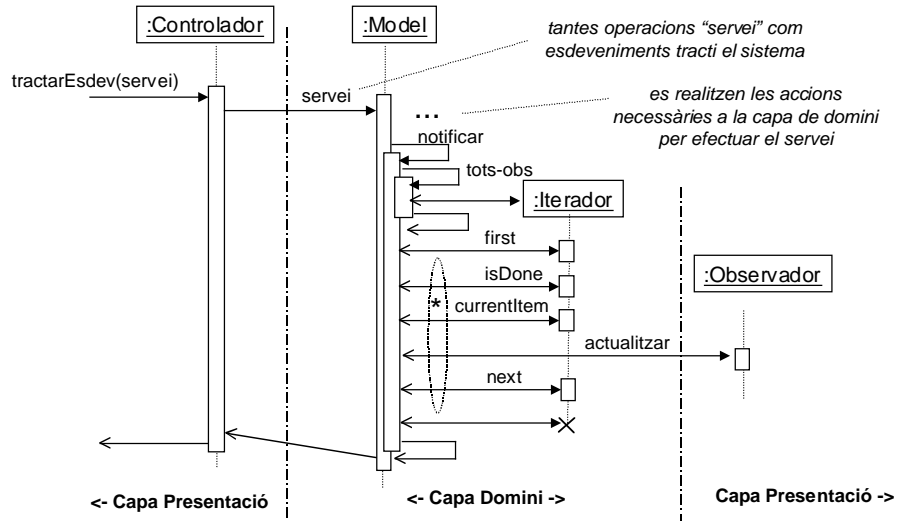
- **Vista:**
 - Permet presentar informació del model a l'usuari. Hi pot haver diverses vistes d'un mateix model.
 - La informació que mostra es pot veure afectada per canvis en l'estat del Model.
 - Té associat un Controlador que gestiona, si s'escau, els esdeveniments de modificació del Model.
 - Pot proporcionar operacions que permeten que els controladors gestionin la modificació del display (paginació, moure finestra, iconificar, ...).



Comportament: inicialització del patró MVC



Comportament: gestió d'un esdeveniment (I)

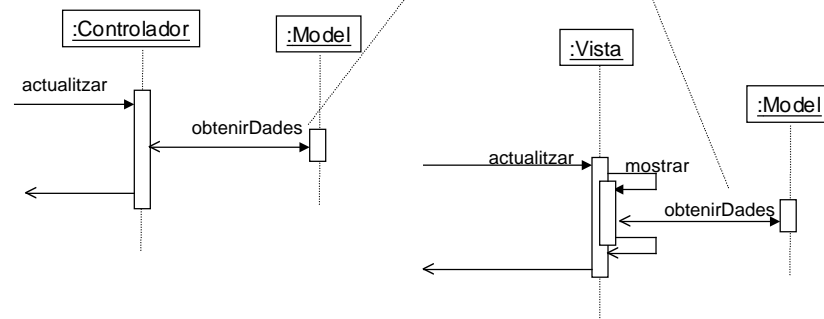


Comportament: gestió d'un esdeveniment (II)

Actualització dels observadors:

- Obté la informació del model, rellevant a la vista i al controlador
- No té perquè ser la mateixa informació i, aleshores, no serà la mateixa operació

Si el controlador modifica el comportament



Conseqüències

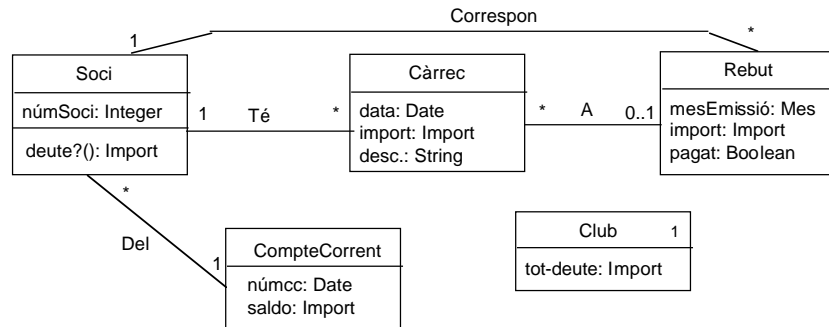
- **Beneficis:**
 - Es poden tenir diverses Vistes associades a un mateix Model.
 - En temps d'execució puc tenir diverses vistes obertes.
 - Les Vistes i Controladors estan sincronitzats davant canvis en el Model.
 - Fàcil reutilització de Vistes i Controladors.
 - Alta portabilitat de la Capa Presentació.
 - Acoblament baix entre Capa Domini i Capa Presentació.
- **Inconvenients:**
 - Afegeix complexitat a sistemes amb interfícies simples.
 - Alt acoblament entre Vista i Controlador.
 - No hi ha encapsulament de dades depenents de la plataforma gràfica, per facilitar portabilitat.
- **Variants al Patró:**
 - Patró Presentació-Abstracció-Control.
 - Patró Document-Vista.

Aspectes a considerar

- Canvis freqüents en l'estat del Model poden comportar actualitzacions innecessàries a vistes (inactives o iconificades) i reduir-ne l'eficiència.
- En sistemes amb més d'una vista oberta, hi haurà diversos controladors actius. Els esdeveniments rebuts han de transmetre's als controladors en un ordre específic (Patró Cadena de Responsabilitats).
- Definició de Vistes i Controladors com a especialitzacions d'altres permet la reutilització, herència i redefinició d'operacions.

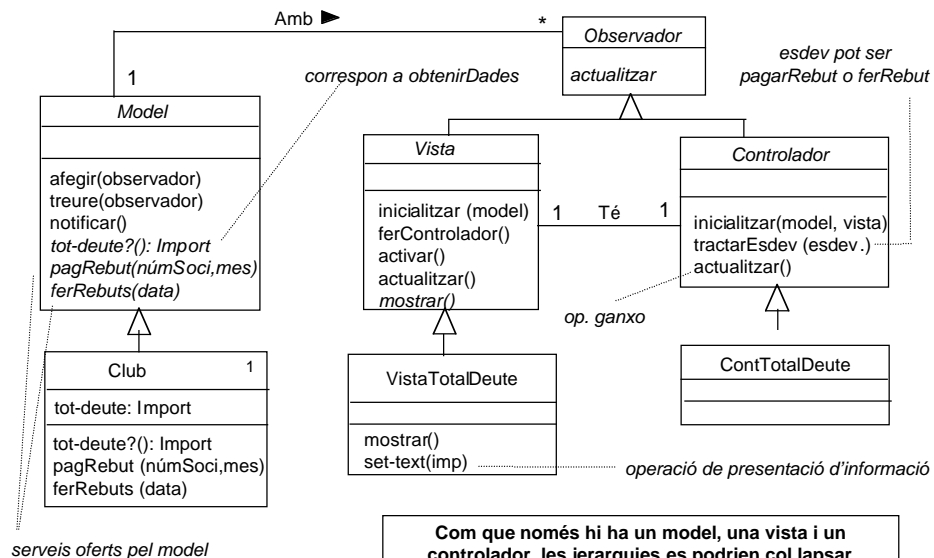
Exemple: deute total del club de tennis

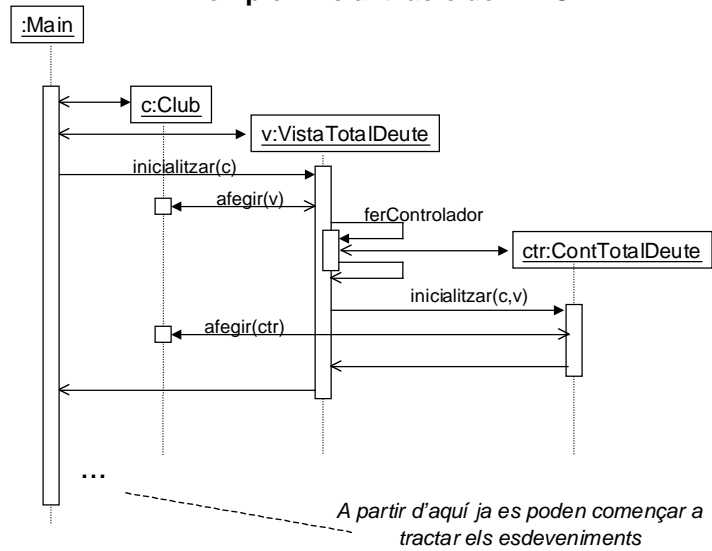
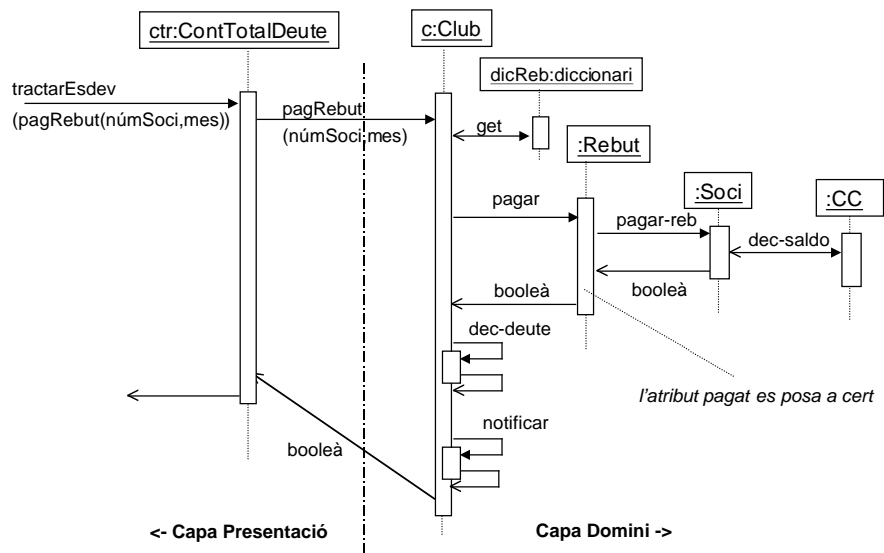
Diagrama de classes de disseny (inicial)



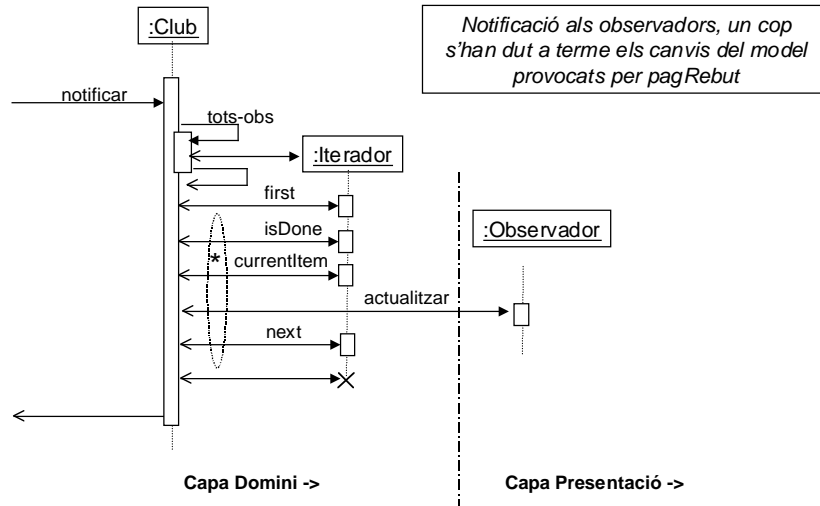
- Es vol visualitzar contínuament per pantalla el deute total del club de tennis
- El deute total del club de tennis es modifica mitjançant *ferRebuts* i *pagRebut*
- El comportament del controlador no depèn de l'estat del model

Exemple: estructura MVC

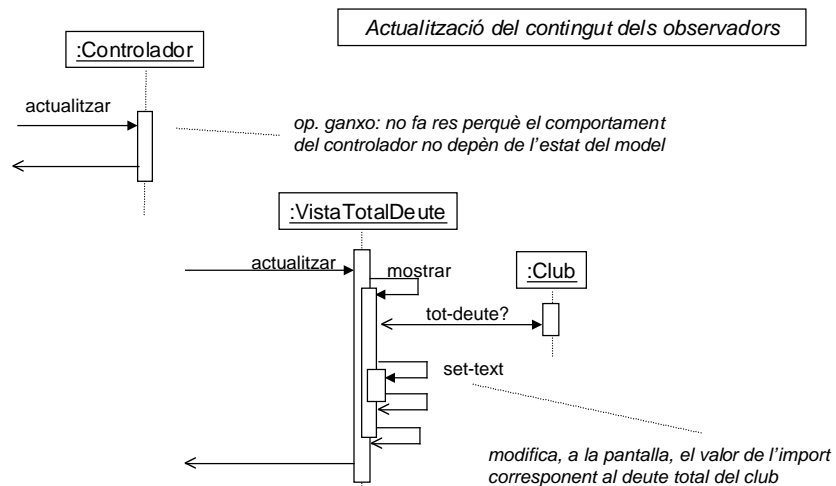


Exemple: inicialització del MVC**Exemple: gestió de pagRebut (I)**

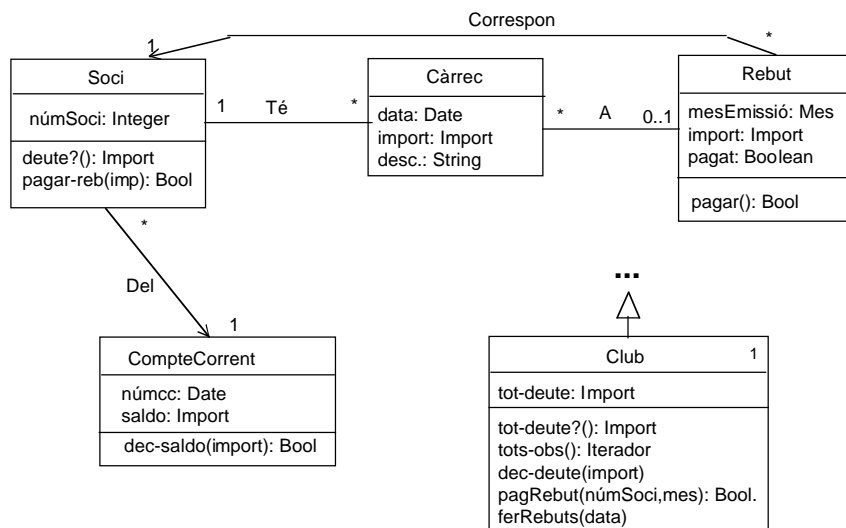
Exemple: gestió de pagRebut (II)



Exemple: gestió de pagRebut (III)



Exemple: diagrama de classes resultant



Bibliografia

- *Pattern-oriented Software Architecture. A System of patterns.* F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. John Wiley & Sons, 1996.

Persistència en BDOO

- Què és i com s'aconsegueix la persistència?.
- Antecedents i arquitectura d'un SGBDOO.
- Model d'Objectes i llenguatges d'especificació d'objectes (ODL).
- Llenguatge de consulta d'objectes (OQL).
- Java binding.
- Bibliografia

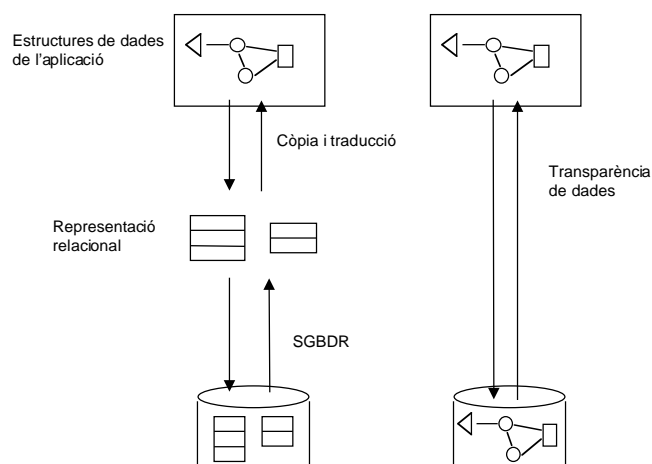
Què és i com s'aconsegueix la persistència?

- **Persistència:** és la capacitat que molts sistemes software requereixen per emmagatzemar i obtenir dades usant un sistema d'emmagatzemament permanent.
- Els objectes es poden fer persistents en:
 - BDOO.
 - BDR.
 - Altres.

Antecedents i arquitectura d'un SGBDOO Proposta d'un estàndar per BDOO

- ODMG (Object Database Management Group): és un grup de persones, membres d'empreses (JavaSoft, Windward Solution, UniSQL, ...), que proposen un estàndar per treballar amb sistemes de gestió de Base de Dades Orientades a Objectes.
- Objectiu de l'estàndar:
 - Portabilitat
- Sistemes de gestió de base de dades orientades a objectes:
 - ObjectStore
 - O2
 - ...

Antecedents i arquitectura d'un SGBDOO Comparació amb un SGBDR



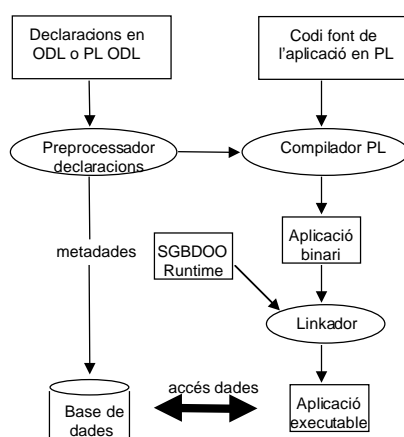
Antecedents i arquitectura d'un SGBDOO

Arquitectura d'un SGBDOO

- Model d'objectes: és el model de dades comú suportat per tots els SGBDOO.
- Llenguatge d'especificació d'objectes (ODL): llenguatge de definició dels objectes.
- Llenguatge de consulta d'objectes (OQL): llenguatge declaratiu per fer consultes i modificacions dels objectes de la BD.
- Java binding: defineix el lligam entre el model d'objectes i el llenguatge de programació Java.

Antecedents i arquitectura d'un SGBDOO

Arquitectura d'un SGBDOO



Model d'Objecte: Què és el Model d'Objectes?

- El model d'objectes especifica el tipus de semàntica que pot definir explícitament un SGBDOO.
- Les construccions suportades pel model d'objecte són:
 - Objecte i Literal
 - Tipus
 - Estat d'un objecte
 - Comportament de l'objecte
 - BD
 - Transaccions

Llenguatge d'especificació d'objectes (ODL)

- ODL és un llenguatge d'especificació utilitzat per definir les especificacions del tipus d'objectes que constitueixen el Model d'Objectes de ODMG.
- Principis que han guiat al desenvolupament del ODL:
 - ha de suportar totes les construccions semàntiques del model d'objectes.
 - no ha de ser un llenguatge de programació sencer complet, però sí un llenguatge de definició.
 - ha de ser un llenguatge de programació independent.
 - ha de ser extensible.
 - ha de ser pràctic.

Model d'Objectes: Objecte

- Creació d'objectes:

- En ODL:

```
interface ObjectFactory {
    Object new();
};
```

tots els objectes són creats per la invocació d'operacions de creació que hi ha en els objectes fàbrica proporcionats pels llenguatges

```
interface Object {
    enum Lock_Type{read,write,upgrade};
    exception LockNotGranted();
    void lock(in Lock_Type mode) raises (LockNotGranted)
    boolean try_lock(in Lock_Type mode);
    boolean same_as(in Object anObject);
    Object Copy();
    void delete();
};
```

tots els objectes tenen una interfície en ODL, que es heretada implícitament a les definicions d'objectes dels usuaris

- Identificador dels objectes: són generats pel SGBDOO. El domini és la BD.
- Nom dels objectes: Els SGBDOO proporcionen funcions que retornen l'objecte a partir del nom. L'àmbit del nom és la BD.
- Vida dels objectes:
 - Transitòria
 - Permanent

Model d'Objectes: Objecte i Literal

El model d'objectes suporta:

- Col·leccions d'objectes i literals:
 - En ODL: Set<t>, Bag<t>, List<t>, Array<t>, Dictionary<t,v>
- Objectes i literals estructurats:
 - En ODL: Date, Interval, Time, Timestamp
- Literals atòmics:
 - En ODL: long, short, unsigned long, unsigned short, float, double, boolean, octet, char, string, enum
- Literals nulls:
 - En ODL: nullable_float, nullable_set,...

Model d'Objectes: Tipus

- Un tipus pot tenir una definició externa i una o més implementacions.
 - La definició d'interfície és una especificació que defineix el comportament abstracte d'un tipus d'objecte.
 - En ODL: interface Empleat{...}
 - La definició de classe és una especificació que defineix el comportament abstracte i l'estat abstracte d'un tipus d'objecte.
 - En ODL: class Persona{...}
 - La definició de literal fa només referència a l'estat abstracte d'un tipus d'objecte.
 - En ODL: struct Complex{float re; float im}
- Herència del comportament:
 - En ODL: interface Empleat{...} interface Empleat_fix: Empleat {...}
- Herència de l'estat i comportament:
 - En ODL: class Persona{...}

 class Persona_empleada extends Persona{...}: Empleat
- Extents: conjunt de totes les instàncies d'un tipus en una BD concreta.
 - En ODL: class Persona (extent persones) {...}
- Key: instàncies individuals poden ser identificades pels valors de les propietats. L'àmbit de la clau és l'extent.

Model d'Objectes: Estat d'un objecte

- Atributs: defineixen un estat abstracte d'un tipus.
 - En ODL:


```
class Persona{
    attribute short age;
};
```
 - Relacions: es defineixen entre tipus i només poden ser binàries.
 - En ODL:
 - 1:1

<pre>interface Professor{ relationship curs ensenya inverse curs::es_ensenyat_per; };</pre>	<pre>interface Curs{ relationship professor es_ensenyat_per inverse professor::ensenya; };</pre>
---	--
 - 1:n

<pre>interface Professor{ relationship set<curs> ensenya inverse curs::es_ensenyat_per; };</pre>	<pre>interface Curs{ relationship professor es_ensenyat_per inverse professor::ensenya; };</pre>
--	--
 - m:n

<pre>interface Professor{ relationship set<curs> ensenya inverse curs::es_ensenyat_per; };</pre>	<pre>interface Curs{ relationship set<professor> es_ensenyat_per inverse professor::ensenya; };</pre>
--	---
- El SGBDOO manté la integritat referencial de les relacions.

Model d'Objectes: Comportament d'un objecte

- Operacions (signatura): defineixen el nom de l'operació, el nom i tipus de cada argument, el tipus de o dels valors que retornen i el nom de les excepcions.
- Model Excepcions: representen operacions que poden donar excepcions i comunicar resultats.
 - En ODL:


```
void acomiadar() raises (no_és_un_empleat)
```

Model d'Objectes: BD

- Cada BD és una instància del tipus Database, proporcionat pel SGBDOO.

```
interface DatabaseFactory {
    Database new();
};

interface Database {
    void open(in string database_name);
    void close();
    void bind(in any an_object, in string name);
    Object unbind(in string name);
    Object lookup(in string object_name);
    Module schema();
};
```

Model d'Objectes: Transaccions

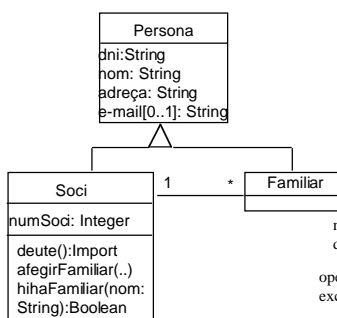
- Hi ha dos tipus definits per suportar l'activitat transaccional en un SGBDOO.

```
interface TransactionFactory {
    Transaction new();
    Transaction current();
};
```

```
interface Transaction {
    exception TransactionInProgress();
    exception TransactionNotInProgress();
    void begin() raises (TransactionInProgress);
    void commit() raises(TransactionNotInProgress);
    void abort() raises (TransactionNotInProgress);
    void checkpoint() raises (TransactionNotInProgress);
    void join();
    void leave();
    boolean isOpen();
};
```

Model d'Objectes: Exemple

Diagrama de classes de disseny



```
class Persona
(extent Persones ----- població de Persones
key dni) ----- clau de Persona
{
    attribute string dni;
    attribute string nom;
    attribute string adreça;
    attribute string e-mail;
};
class Soci extends Persona ----- herència de l'estat i comportament
{
    attribute short numSoci;
    relationship set<Familiar> familiars
    inverse Familiar::soci;
    short deute();
    void afegirFamiliar (in string nom, in string adreça, in
    string e-mail) raises (familiar_ja_existeix);
    boolean hihaFamiliar(in string nom);
};
class Familiar extends Persona
{
    relationship soci
    inverse Soci::familiars;
};
```

navegabilitat
doble
operació amb
excepcions

Llenguatge de consulta d'objectes (OQL)

- El disseny del llenguatge (OQL) està basat en els següents principis:
 - Està basat en el Model Objecte de ODMG.
 - Molt semblant a SQL92.
 - Proporciona primitives d'alt nivell per treballar amb col·leccions.
 - Es un llenguatge funcional on els operadors es poden compondre.
 - Proporciona un accés fàcil a un SGBDOO.
 - Pot ser invocat des d'operacions escrites en altres llenguatges i al contrari.
 - No té operadors d'actualització, sino que utilitza els mètodes dels objectes.
 - Proporciona un accés declaratiu als objectes.
 - La semàntica formal pot ser fàcilment definida.

Llenguatge de consulta d'objectes (OQL): Exemple

- Exemples de consultes en OQL

```
typedef set<string> vectad;  
vectad (select distinct adreça  
       from Persones  
       where nom='Pep')
```

```
typedef bag<soci> socs;  
socs( select distinct soci(a:nom,b:adreça,c:e-mail)  
      from Persones  
      where nom='Pep')
```

Java binding

- Principis de disseny del llenguatge:
- Principi bàsic:
 - El programador ha de veure el lligam com un llenguatge únic per expressar operacions de la BD i de l'aplicació.
- Corol.laris:
 - Hi ha només un únic sistema compartit pel llenguatge Java i la base de dades d'objectes.
 - El lligam respecta la sintaxi de Java.
 - Els objectes es convertiran en persistents quan siguin referenciats per altres objectes persistents de la BD (persistència per referència).
- El Java binding proporciona :
 - Un Model Objecte comparable amb el Model Objecte proposat per ODMG.
 - Java ODL
 - Java OML
 - Java OQL

Java binding: Model Objecte

- El Model Objecte de Java suporta totes les característiques del Model Objecte de ODMG excepte:
 - Relacions.
 - Extents.
 - Claus.
 - No hi ha literals estructurats.

Java binding: Java ODL i Java OML

- Java ODL ens permet descriure l'esquema de la BD com un conjunt de classes Java utilitzant sintaxi Java a excepció dels elements que no suporta el Model Objecte de Java.
- Java OML és la sintaxi utilitzada per crear, esborrar, identificar, referenciar, obtenir i modificar valors i invocar mètodes d'objectes persistents. Totes les operacions de Java OML són invocades en els objectes Java apropiats com si fossin objectes transitoris.
 - La persistència dels objectes és per referència.

Java binding: JavaOQL

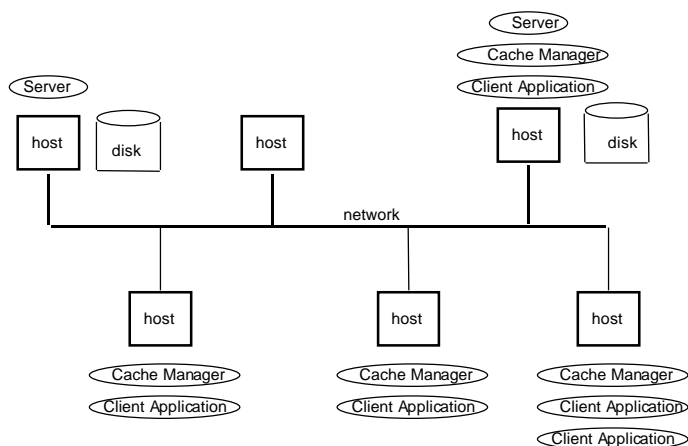
- Totes les funcionalitats de l'OQL estan disponibles mitjançant Java binding. Aquesta funcionalitat pot ser usada:
 - Mètodes de consulta a les classes col.lecció.
 - Consultes utilitzant classes OQLQuery

```

class OQLQuery {
    public OQLQuery() {}
    public OQLQuery(String Query){...}
    public create(String Query){...}
    public bind(Object parameter){...}
    public Object execute() throws ODMGException{...}
};

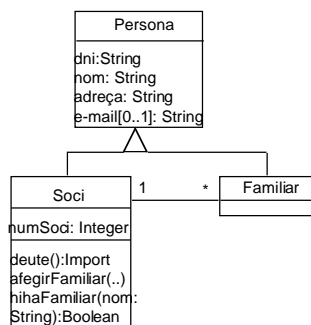
```

Java binding: Arquitectura d'ObjectStore



Java binding Exemple d'utilització del Java binding amb ObjectStore/1

Definició de l'esquema



```

abstract class Persona {
    String nom;
    String adreça;
    String email;

    // Constructor:
    public Persona(String nom, String adreça, String email) {
        this.nom = nom; this.adreça = adreça; this.email = email;
    }

    public String obtenir_nom() { return nom; }
    public String obtenir_adreça() { return adreça; }
    public String obtenir_email() { return email; }
    public void posar_nom(String nom) { this.nom = nom; }
    public void posar_adreça(String adreça) { this.adreça = adreça; }
    public void posar_email(String email) { this.email = email; }
}

class Familiar extends Persona {
    Soci s;
    public Familiar(String nom, String adreça, String email, Soci s) {
        super(nom, adreça, email);
        this.s = s;
        s.posar_familiar(this);
    }
}

```

Java binding

Exemple d'utilització del Java binding amb ObjectStore/2

```
import COM.odi.*;
import COM.odi.coll.*;
class Soci extends Persona {
    int numSoci;
    Set familiars;

    // Constructor:
    public Soci(String nom, String adreca, String email, int numSoci, Placement placement) {
        super(nom, adreca, email);
        this.numSoci = numSoci;
        familiars = NewCollection.createSet(placement);
    }
    public int obtenir_numSoci() { return numSoci; }

    public void posar_familiar(Familiar f) {
        familiars.insert(f);
    }
}
```

Java binding

Exemple d'utilització del Java binding amb ObjectStore/3

Definició dels programes

```
import COM.odi.*;
public
class Exemple {
    public static void main(String argv[]) {
        String dbName = argv[0];
        // Aquesta línia inicialitza ObjectStore
        ObjectStore.initialize(null, null);
        Database db = createDatabase(dbName); // operació de la classe Database
        readDatabase(db);
    }

    static Database createDatabase(String dbName) {
        // Es crea una BD cada vegada que es crida a l'aplicació
        try {
            Database.open(dbName, ObjectStore.OPEN_UPDATE).destroy();
        } catch (DatabaseNotFoundException e) {
        }
        // Crida al mètode de creació de la BD.
        Database db = Database.create(dbName, ObjectStore.ALL_READ | ObjectStore.ALL_WRITE);
        // Comença una transacció d'actualització
        Transaction tr = Transaction.begin(ObjectStore.UPDATE); // operació de la classe Transaction
        // Creació dels soci
        Soci pep = new Soci("Pep", "C/Balmes", "pep@lsi.upc.es", 1, db); // operació de creació d'un objecte a la BD
        Soci josep = new Soci("Josep", "C/Sants", "josep@lsi.upc.es", 2, db);
        // Creació de dos punts d'entrada a la BD
        db.createRoot("Pep", pep); // operació de punts d'entrada a la BD
        db.createRoot("Josep", josep);
    }
}
```

Java binding

Exemple d'utilització del Java binding amb ObjectStore/4

```
Familiar maria = new Familiar("Maria", "C/Balmes", "maria@lsi.upc.es", pep);
Familiar joan = new Familiar("Joan", "C/Pelai", "joan@lsi.upc.es", pep);
Familiar pere = new Familiar("Pere", "Rda. Universitat", "pere@lsi.upc.es", josep);
// Final de la transaccio. Es guarden els objectes persistents a la BD.
tr.commit();
return db;
}
static void readDatabase(Database db) {
    // Comença una transaccio de lectura
    Transaction tr = Transaction.begin(ObjectStore.READONLY);
    // Utilitzem els punts d'entrada per obtenir els objectes.
    Soci pep = (Soci)db.getRoot("Pep");
    Soci josep = (Soci)db.getRoot("Josep");
    //Final de la transaccio de lectura
    tr.commit();
}
}
```

Bibliografia

- *Object Database Standard: ODMG 2.0*
Catell R.G.G, Barry Douglas, Bartels Dirk et al.
Morgan Kaufmann Publishers, Inc.