

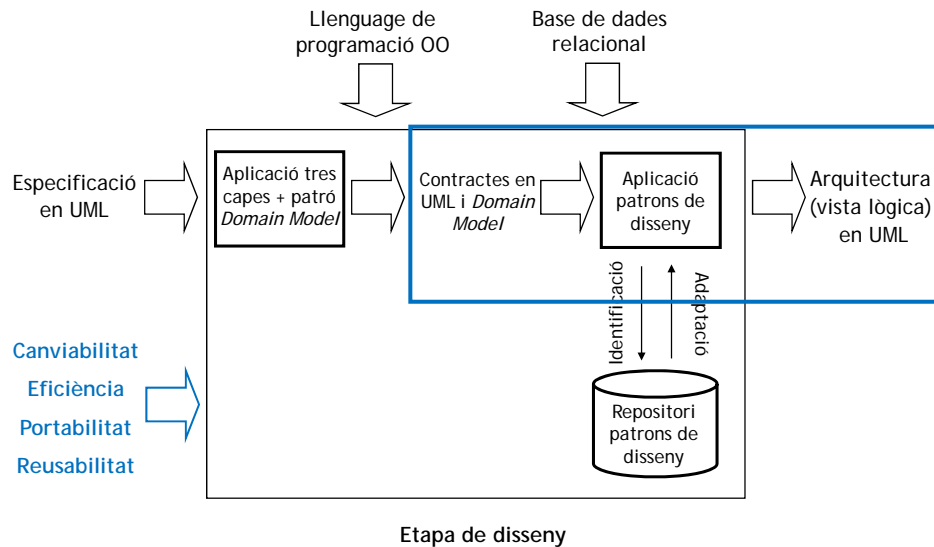
Patrons de disseny



Patrons de disseny

Introducció als patrons de disseny
Patró controlador
Patrons d'assignació de responsabilitat a objectes
Patró estat
Bibliografia

Etapa de disseny a l'assignatura IES



Concepte de patró

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

Christopher Alexander, arquitecte (1977)

Context

- Situació en la què es presenta el problema de disseny

Problema

- Descripció del problema a resoldre
- Enumeració de les forces a equilibrar

Solució

- Aspecte estàtic: impacte en el diagrama de classes del disseny
- Aspecte dinàmic: establiment del comportament de les noves operacions

Catàlegs de patrons de disseny

Patrons que determinen l'estructura general de les capes

Proposats per Fowler (2003):

- Capa de domini:
 - Gran influència en la distribució de responsabilitats a capes
 - *Domain Model, Transaction Script*
- Capa de dades:
 - Determinen els serveis que ofereix la capa de dades
 - *Data Mapper, Pasarel·la Fila, Enregistrament Actiu*

5

Catàlegs de patrons de disseny

Patrons d'aplicació general

Proposats per Gamma *et al.* (1995) i Larman (2005):

Controlador

Patrons d'assignació de responsabilitats a objectes

Estat

Plantilla

Observador

Representant

etc...

*són els que
estudiarem a IES*

6

Patró controlador

Patró controlador: descripció general

Context:

- Els (sub)sistemes software reben esdeveniments
 - . Ex: la capa de domini d'un SI rep esdeveniments externs
- Un cop interceptats aquests esdeveniments, algun objecte del sistema ha de rebre'ls i executar les accions corresponents

Problema:

- Quin objecte és el responsable de rebre un esdeveniment?

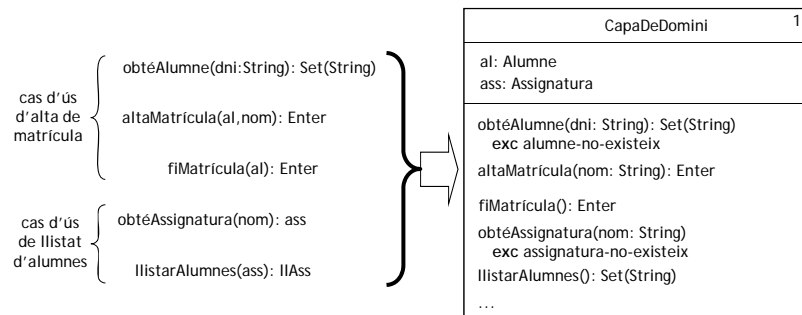
Solució:

- Assignar aquesta responsabilitat a un controlador
 - . Els clients del sistema desconeixen l'estructura interna del sistema
- Un controlador és un objecte d'una certa classe
 - . El controlador delega sobre un o més objectes del sistema el tractament de l'esdeveniment
- L'objecte que tracta l'esdeveniment no té coneixement sobre l'existència o el tipus de controlador
- Variants analitzades:
 - . Façana: Un objecte que representa tot el sistema
 - . Cas d'ús: Un objecte que representa una instància d'un cas d'ús
 - . Transacció (*Command*): Un objecte que representa una instància d'esdeveniment

Controlador façana

Aspecte estàtic

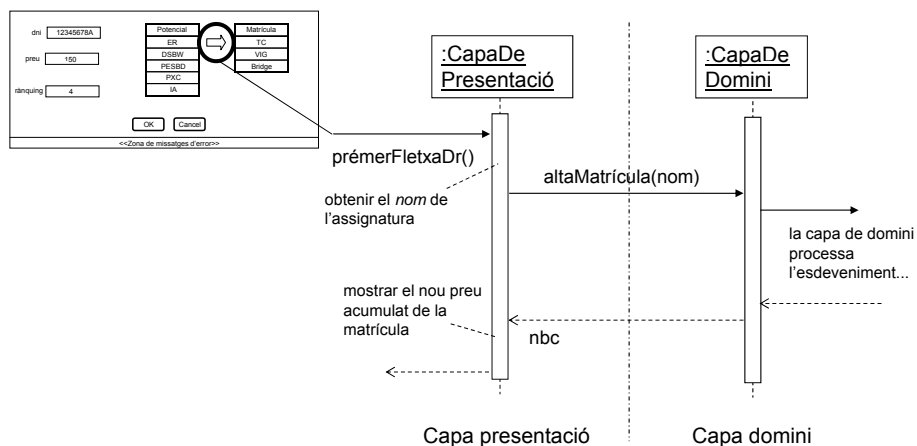
- Classe *singleton*
 - tantes operacions com esdeveniments ha de capturar el sistema
 - eventualment, poden incloure's atributs per compartir informació
- Controladors inflats si hi ha molts esdeveniments → poca cohesió



9

Controlador façana

Aspecte dinàmic



10

Controlador cas d'ús

S'associa un *controlador cas d'ús* a cada cas d'ús definit al sistema

Aspecte estàtic: tantes noves classes com casos d'ús té el sistema

- cada classe declara les operacions del diagrama de seqüència corresponent
- eventualment, poden incloure's atributs per compartir informació (estat del cas d'ús)

ControladorAltaMatrícula	ControladorListaAlumnos
al: Alumne	ass: Assignatura
obtéAlumne(dni: String): Set(String) exc alumne-no-existeix altaMatrícula(nom: String): Enter fiMatrícula(): Enter	obtéAssignatura(nom: String) exc assignatura-no-existeix l·listarAlumnos(): Set(String)

Aspecte dinàmic: similar al cas anterior

- en no ser *singleton*, cal crear-los i destruir-los quan es necessiten

Millora la cohesió del sistema

11

Controlador transacció

Aspecte estàtic (1)

S'introdueix una classe concreta per cada operació del sistema (transacció)

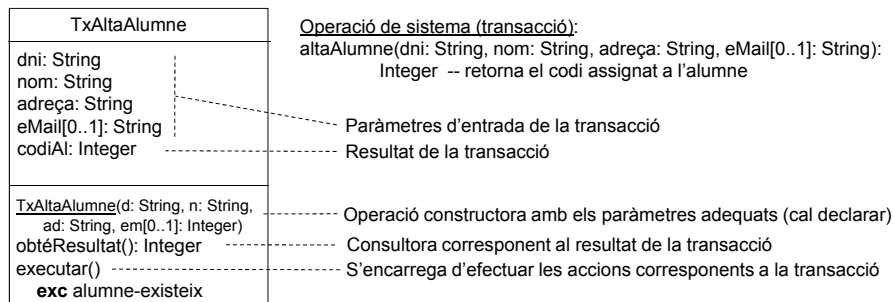
Cada paràmetre de l'operació dóna lloc a un atribut de la classe

- Si l'atribut és *out* o *inout*, s'afegeix una operació per consultar el seu valor
- L'operació constructora de la classe té tants paràmetres com paràmetres *in* i *inout* té l'operació

Si hi ha resultat, també es declara un atribut del tipus del resultat

- S'afegeix una operació per consultar el seu valor

S'afegeix una operació **executar()** que s'encarrega d'executar la transacció



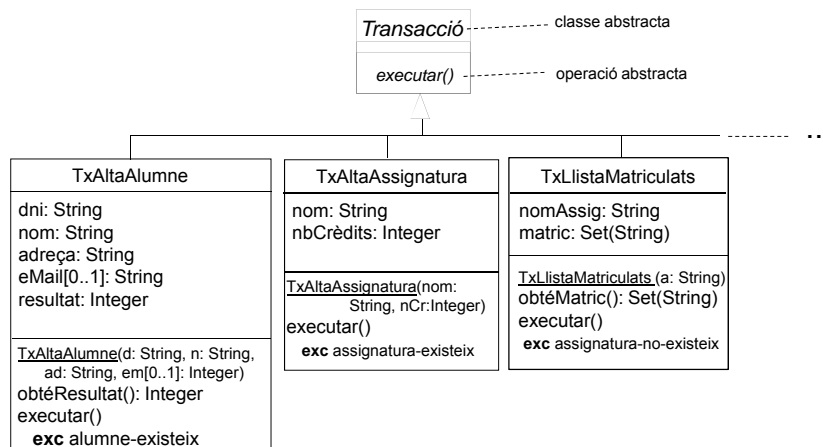
12

Controlador transacció

Aspecte estàtic (2)

S'introdueix una classe abstracta que actua de superclasse de tots els controladors transacció del sistema

- Declara l'operació d'executar la transacció com a abstracta
- Proporciona una vista unificada a les classes clients dels diferents tipus de transaccions



13

Controlador transacció

Aspecte dinàmic (1)

Es crea un objecte transacció a cada ocurrència de l'esdeveniment corresponent

L'operació constructora inicialitza convenientment els atributs corresponents als paràmetres *in* i *inout*

Tot seguit, s'invoca l'operació *executar* de l'objecte creat

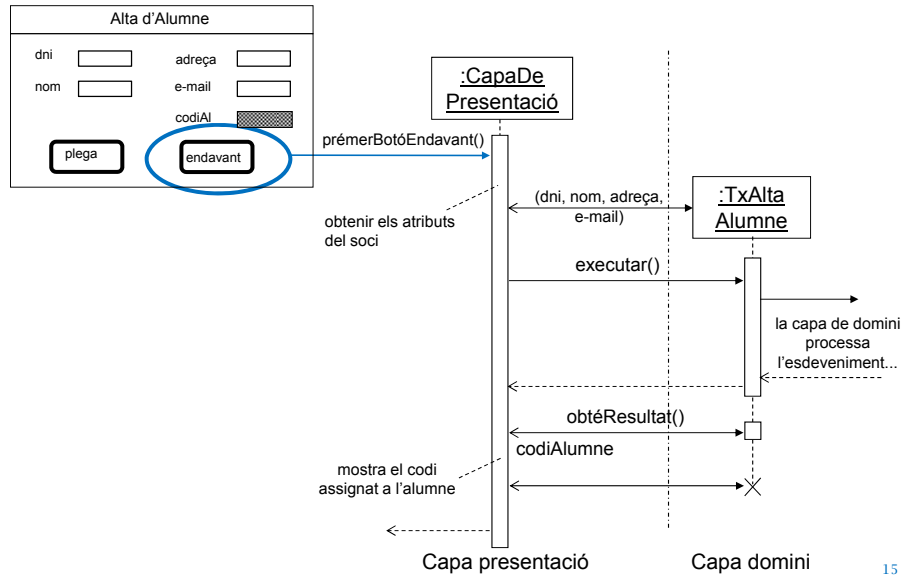
A continuació, es poden consultar els resultats de l'execució

S'acostuma a destruir l'objecte transacció una vegada recollits els resultats

14

Controlador transacció

Aspecte dinàmic (2)



15

Patrons d'assignació de responsabilitats a objectes

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.

En el context del patró *Domain Model*, les responsabilitats d'un objecte consisteixen a:

- 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.

17

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 (excepcions i postcondicions de les operacions)
- No sempre existeix un únic expert, sinó que poden existir diversos experts parcials que hauran de col·laborar.

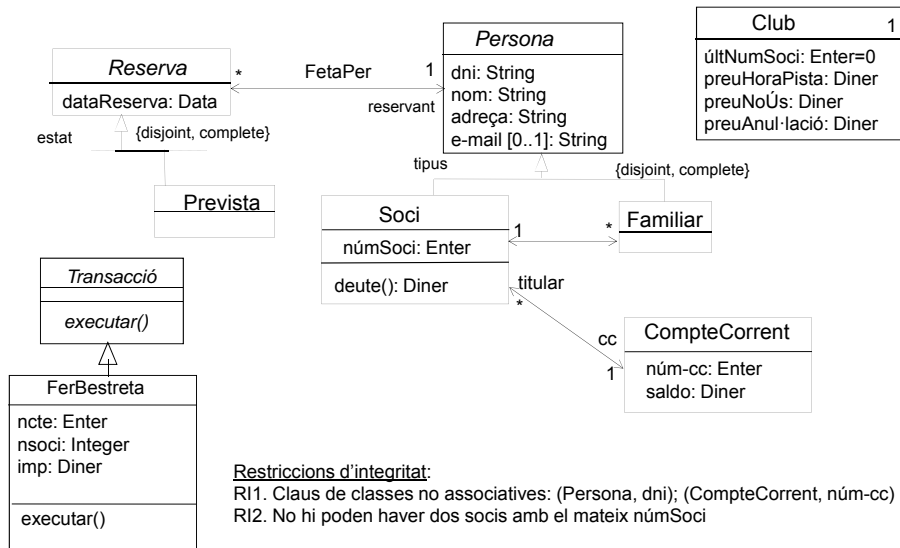
Beneficis:

- Es manté l'encapsulament → baix acoblament
- Conducta distribuïda entre les classes que tenen la informació → alta cohesió

18

Patró expert: exemple

Diagrama de classes i controlador



19

Patró expert: exemple

Contracte de fer-bestreta

context fer-bestreta (ncte: Enter, nsoci: Enter, imp: Diner)

-- el Soci nsoci fa un ingrés al compte corrent ncte en previsió del pagament per --
 l'ús de les seves reserves previstes i les dels seus familiars.

exc:

soci-no-existeix: no existeix un Soci s amb numSoci=nsoci ————— **CtrlSocis**
 compte-incorreccte: el CompteCorrent amb num-cc=ncte no és del Soci s ————— **Soci**
 massa-poc: L'import 'imp' es inferior al 50% del cost d'usar les reserves ————— **Soci + (Club +**
 previstes del Soci s i dels seus Familiars **Persona + Reserva)**
 (num_res_prev*preuHoraPista*0,5).

post: s'incrementa el saldo del CompteCorrent amb l'import especificat 'imp' ————— **CompteCorrent**

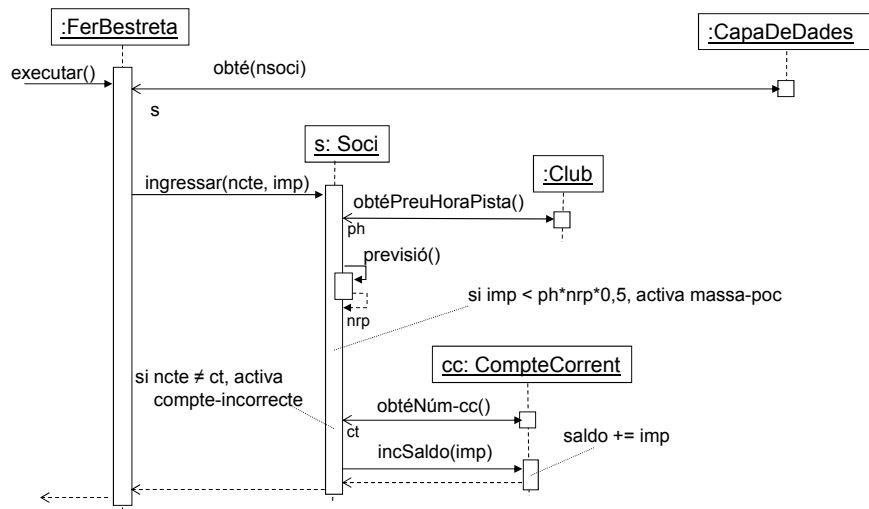
Experts



20

Patró expert: exemple

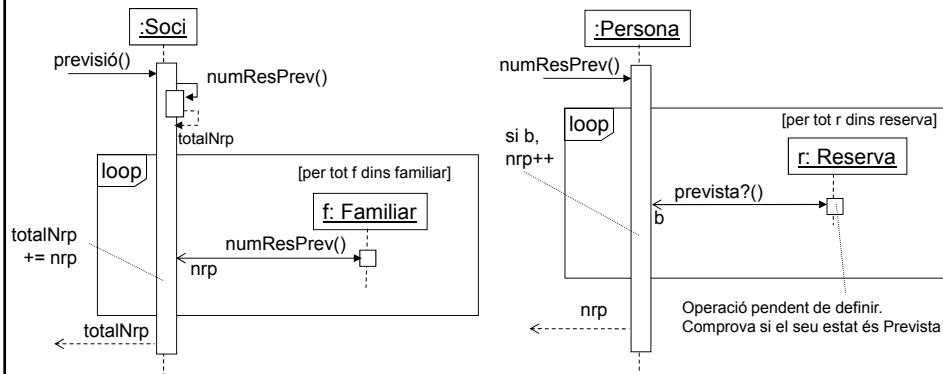
Diagrama de seqüència (I)



21

Patró expert: exemple

Diagrama de seqüència (II)



22

Patr3 expert: exemple

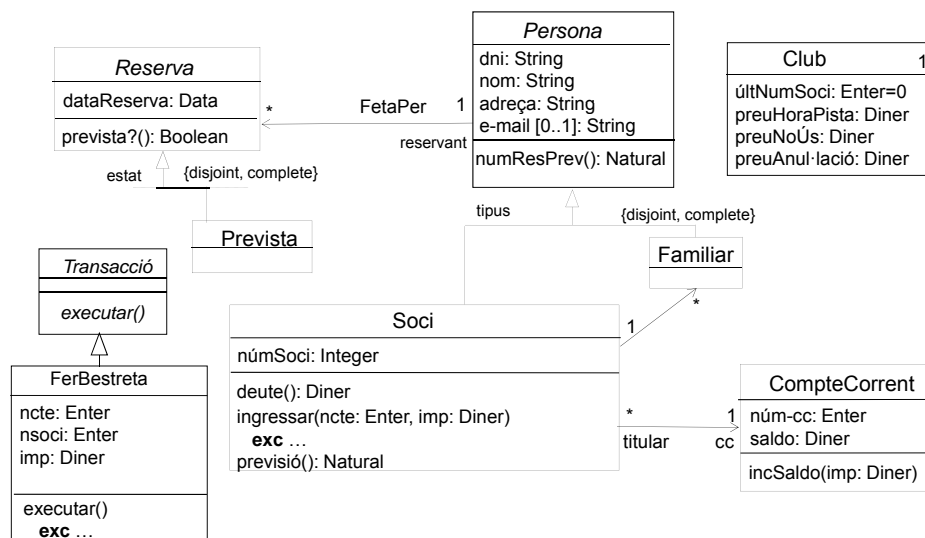
Anàlisi

	Diagrama de Seqüència
Expert	FerBestreta delega tota la feina a Soci que ja té responsabilitats relacionades
Acoblament (afegit)	Dins la capa de domini: FerBestreta acoblat amb Soci i Soci acoblat amb Club Entre capes: FerBestreta acoblat amb capa de dades
Cohesió	No es redueix la cohesió de les classes, ja que s'aprofiten les responsabilitats de gestió degudes a la navegabilitat de les associacions
Reusabilitat	Diverses operacions definides amb possibilitat de reutilització

23

Patr3 expert: exemple

Diagrama de classes de disseny



24

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 de A
 - . B conté objectes de A
 - . B enregistra instàncies d'objectes de A
 - . B usa molt objectes de A
 - . B té les dades necessàries per inicialitzar un objecte de A (B té els valors dels paràmetres del constructor de A)

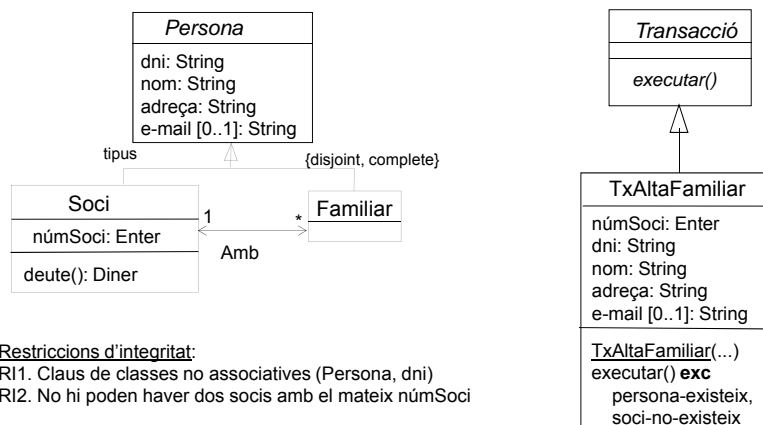
Beneficis:

- Acoblament baix

25

Patró creador: exemple

Diagrama de classes de disseny



Suposem que la navegabilitat entre soci i familiar és doble

26

Patró creador: exemple

Contracte altaFamiliar

context altaFamiliar (númSoci: Enter, dni: String, nom: String, adreça: Enter, e-mail: String[0..1])

exc persona-existeix: ja existeix una Persona amb aquest dni

soci-no-existeix: no existeix un Soci amb aquest numSoci

post:

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

Creador

Té les dades per inicialitzar Familiar

(A) Transacció

(B) Soci

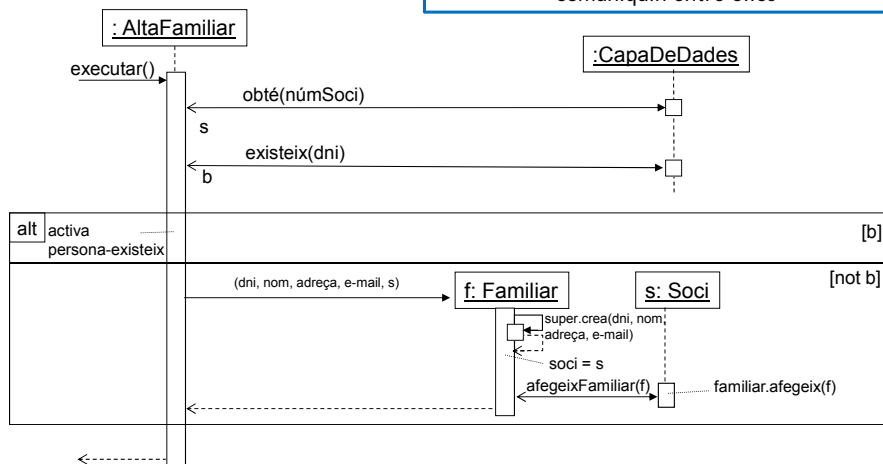
Enregistra Familiars

27

Patró creador: exemple

Diagrama de seqüència (A)

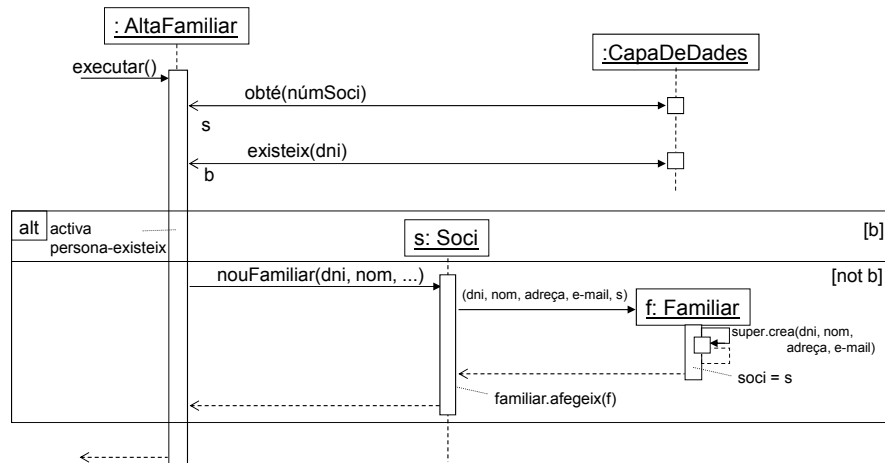
El controlador és qui fa tota la interacció amb la capa de gestió de dades
volem que les classes del domini només es comuniquin entre elles



28

Patr  creator: exemple

Diagrama de seq ncia (B)



29

Patr  estat

Patró Estat

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.

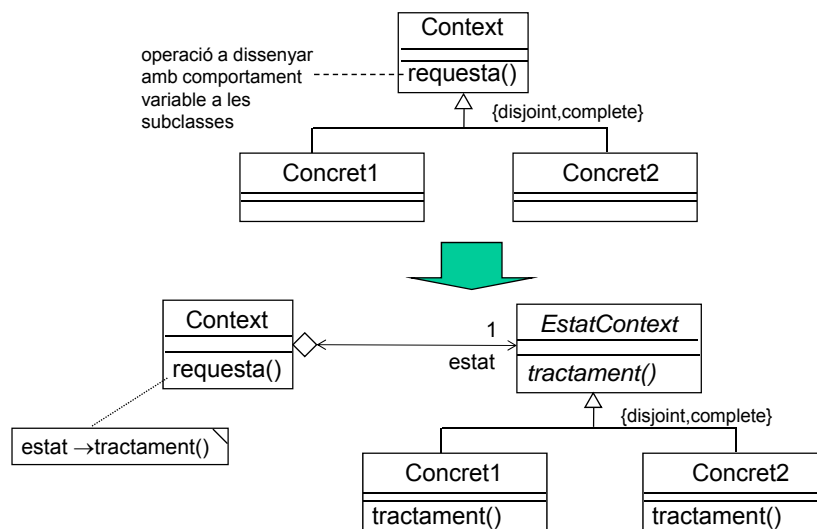
Solució:

- 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*.

31

Estat

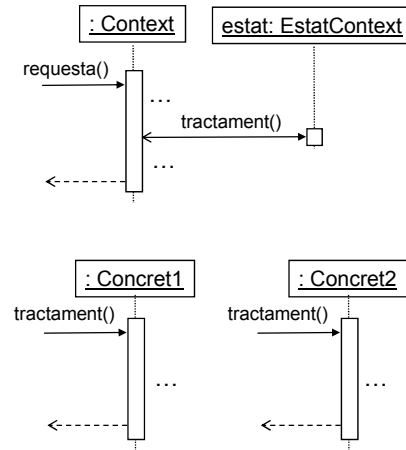
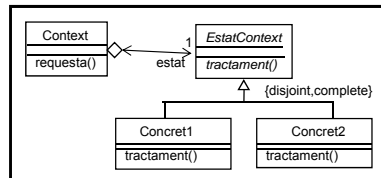
Aspecte estàtic



32

Estat

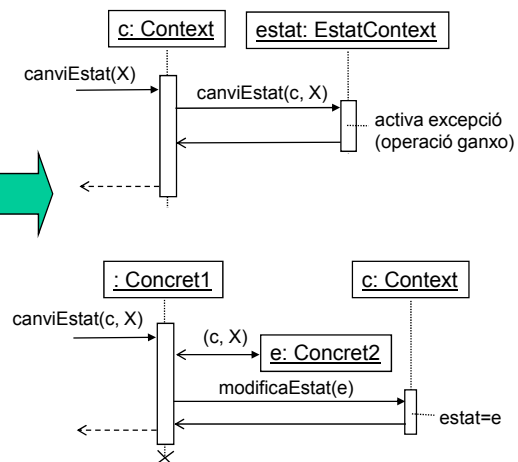
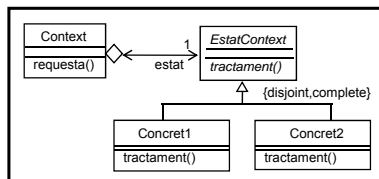
Aspecte dinàmic



33

Estat

Aspecte dinàmic, canvi estat

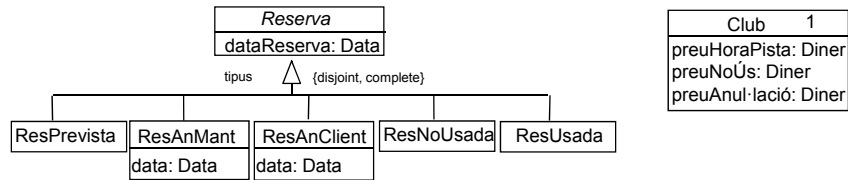


34

Estat

Exemple 1, problema

- Esquema conceptual d'especificació:



- Es vol dissenyar les operacions *anMant* i *cost* de la classe d'objectes *Reserva*.
- L'operació *anMant* passa una reserva prevista a anul·lada per manteniment.
- L'operació *cost* calcula el cost d'una reserva:
 - $\text{cost} = 0$ si la reserva està prevista o anul·lada per manteniment.
 - $\text{cost} = \text{preuHoraPista}$ (de Club) si la reserva està usada.
 - $\text{cost} = \text{preuNoÚs}$ (de Club) si la reserva està no usada.
 - $\text{cost} = \text{preuAnul·lació}$ (de Club) si la reserva està anul·lada pel client.

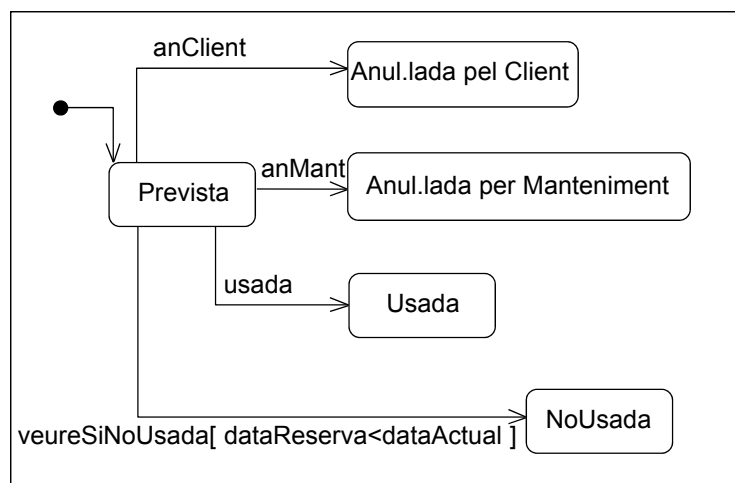


Aplicació del patró Estat

35

Estat

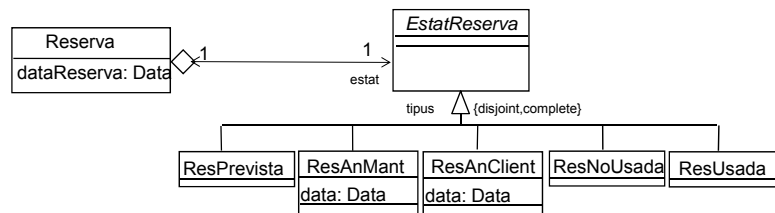
Exemple 1, diagrama d'estats de Reserva



36

Estat

Exemple 1, aplicació del patró

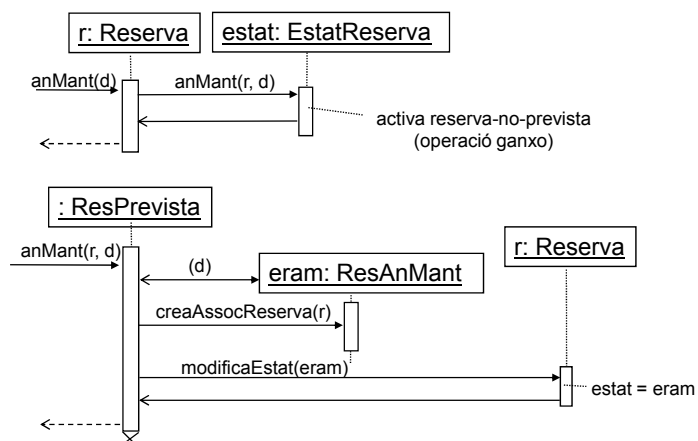


37

Estat

Exemple 1, disseny operació *anMant*

Operació *anMant*(d: Data)

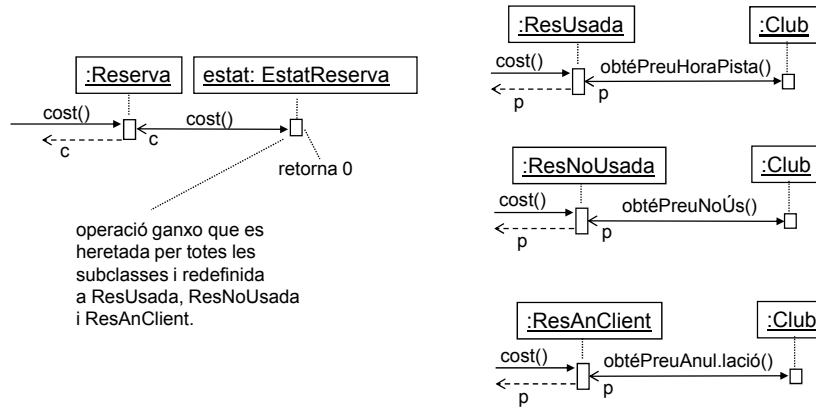


La resta de diagrames de seqüència per les operacions *anClient*, *usada* i *noUsada* serien similars.

38

Estat

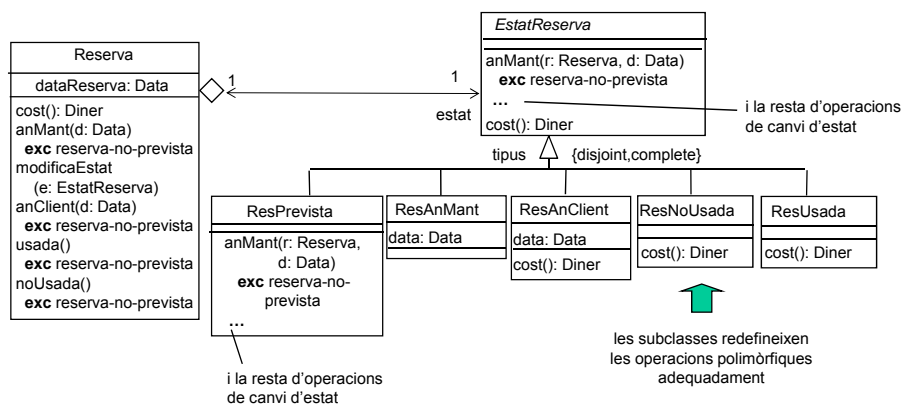
Exemple 1, disseny operació *cost*



39

Estat

Exemple 1, diagrama de classes resultant del disseny



40

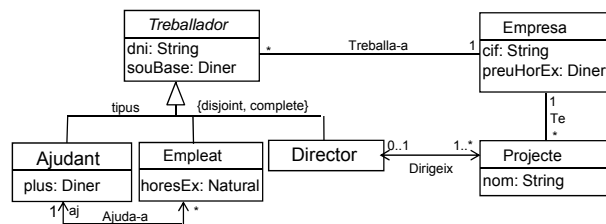
Conseqüències

- Descomposa i localitza en un sol lloc la conducta dels diferents estats:
 - La conducta de cada estat està en una operació diferent.
- Es poden definir fàcilment nous estats, sense alterar els existents.
- Fa explícites les transicions d'estat:
 - Cada estat té associat un objecte estat diferent.

41

Estat

Exemple 2, problema



R.I. Textuals:

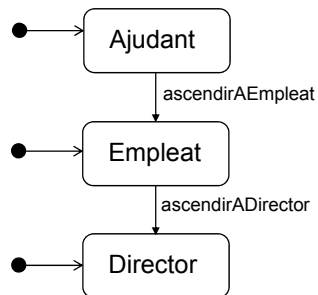
- Claus classes no associatives: (Treballador, dni); (Empresa, cif); (Projecte, nom)
- Un director dirigeix projectes de l'empresa on treballa

Es vol dissenyar l'operació que ascendeix un empleat a director.
Aplicarem també el patró controlador, variant transacció

42

Estat

Exemple 2, diagrama d'estats de Treballador



43

Estat

Exemple 2, contracte d'ascendirADirector

context ascendirADirector (dni: Enter, nomProj: String)

-- l'empleat amb *dni* passa a ser director i comença a dirigir el projecte amb nom *nomProj*

exc:

treballador-no-existeix: El treballador amb dni *dni* no existeix

projecte-no-existeix: El projecte amb nom *nomProj* no existeix

no-empleat: El treballador amb dni *dni* no és empleat

projecte-no-empresa: El projecte amb nom *nomProj* no és de l'empresa on treballa l'empleat

projecte-ja-dirigit: El projecte amb nom *nomProj* ja és dirigit per un director

post:

2.1 El treballador passa a ser director

2.2 S'elimina l'associació *Ajuda-a* entre l'empleat i l'ajudant

2.3 Es forma l'associació *Dirigeix* entre el director i el projecte amb nom *nomProj*

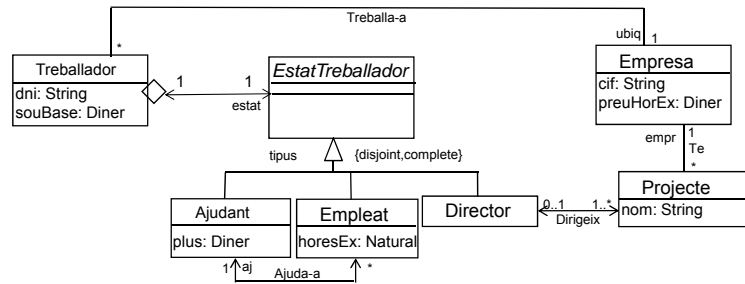


Aplicació del patró Estat

44

Estat

Exemple 2, aplicació del patró

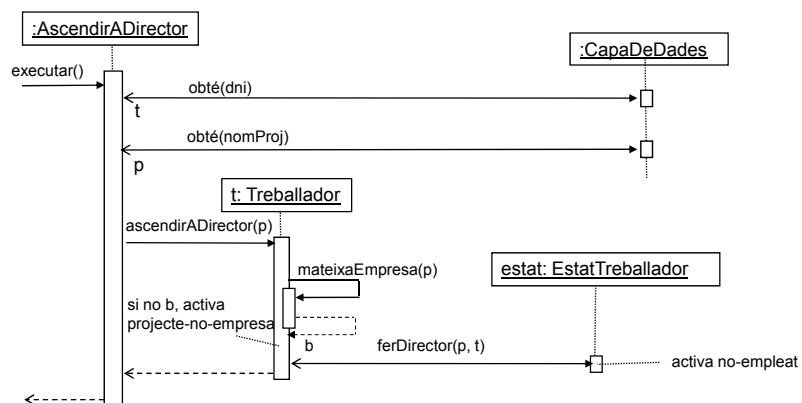


45

Estat

Exemple 2, disseny operació ascendirADirector (1)

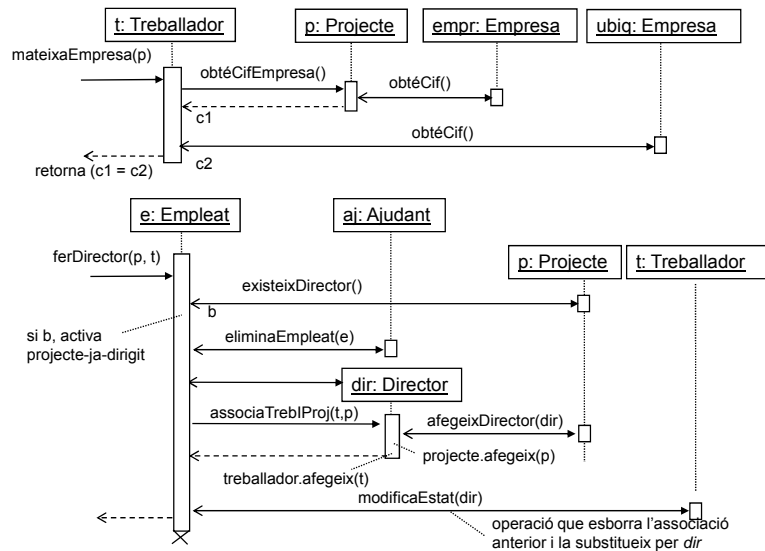
Operació `ascendirADirector(dni: String, nomProj: String)`



46

Estat

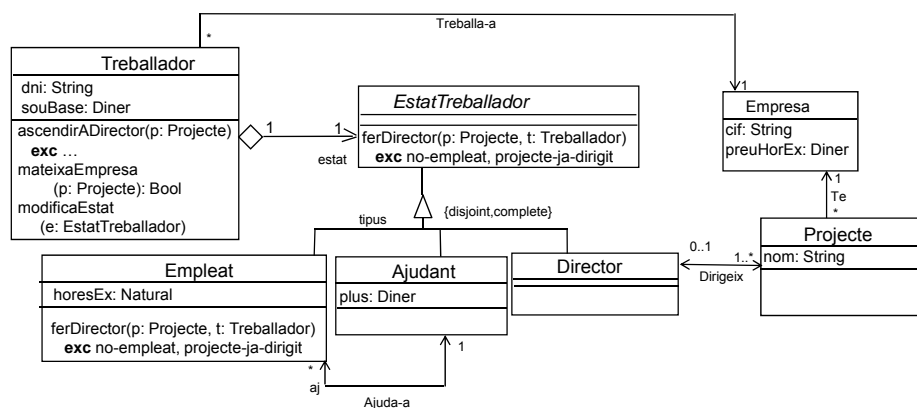
Exemple 2, disseny operació ascendirADirector (2)



47

Estat

Exemple 2, diagrama de classes resultant del disseny



48

Bibliografia

- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. "*Design Patterns*", Addison-Wesley, 1995
- Larman, C. "*Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design*", Prentice Hall, 2005, (3ª edició).
- Fowler, M., *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003
- Martin, R.C., "*Agile Software Development: Principles, Patterns and Practices*", Prentice Hall, 2003.