

## DE L'ANÀLISI AL DISSENY ORIENTAT A OBJECTES

L'objectiu del present document és ampliar l'explicació del mòdul “Disseny Orientat a Objectes” per tal de clarificar quins són els passos necessaris per passar del diagrames de l'anàlisi (diagrames independents de la tecnologia a utilitzar per implementar el sistema) als diagrames de l'etapa de disseny (diagrames adaptats a les tecnologies d'implementació).

En concret es veuran els passos per al disseny del sistema considerant:

- 1 – un llenguatge de programació orientat a objectes.
- 2 – una base de dades relacional (que servirà per donar persistència als objectes de l'aplicació).

Aquest procés a vegades es coneix com a procés de normalització del disseny.

Les transformacions que s'expliquen no són totalment exhaustives però sí que permeten fer-se una idea prou detallada del procés.

### **1. Diagrama estàtic de disseny per una tecnologia orientada a objectes**

#### **1.0 Preprocés**

Abans de començar pròpiament les transformacions, es poden fer una sèrie de passos previs com la normalització de noms (punt 5.1 dels materials), la reutilització de classes (punt 5.2 y el document “Introducció a la reutilització de programari”) i la reconsideració de la necessitat d'algunes classes (en aquest ordre).

Per exemple, si una subclasse no té atributs ni relacions pròpies, es pot eliminar simplement afegint a la superclasse un atribut que permeti indicar si els objectes de la superclasse són o no del tipus de la subclasse (o de quina subclasse si n'hem eliminat varies). Si la subclasse té algun atribut i/o associació, podem afegir aquest atribut/associació a la superclasse.

Vegi's a la figura 1 el resultat d'eliminar la subclasse A1. És important fixar-se que, com a conseqüència de l'eliminació, l'atribut a1 i l'associació amb B de la superclasse són opcionals (sols aquells objectes d'A que siguin del tipus A1 tindran algun valor a l'atribut a1 i estaran relacionats amb algun objecte de B). Per ex. si pensem en A com a la classe Animal i en A1 com la subclasse AnimalEnCaptivitat, tindriem que l'atribut a1 podria representar el nom del propietari de l'animal. Després de la transformació, sols aquells animals de tipus “AnimalEnCaptivitat” tindran algun valor a l'atribut propietari.

L'atribut *tipus* permet identificar quins dels animals són AnimalsEnCaptivitat, quins són AnimalsSalvatges (suposant que AnimalSalvatge sigui una altra subclasse d'Animal)...

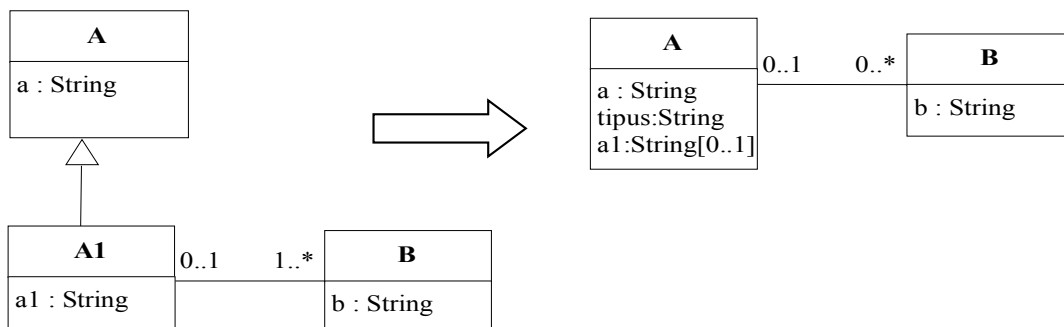


Figura 1

### 1.1 Adaptació de l'herència al nivell suportat pel llenguatge de programació

Aquest punt està explicat a l'apartat 5.3 dels materials

### 1.2 Transformació (“reificació”) de les classes associatives

El concepte de classes associatives no existeix en els llenguatges de programació per tant s'han de convertir a classes “normals”. Aquest procés es coneix amb el nom de reificació.

El procés (figura 2) és ben senzill. Es substitueix la classe associativa per una classe amb el mateix nom i amb els mateixos atributs. Aquesta nova classe es relaciona amb les classes que participaven en l'associació original. La cardinalitat de la nova classe respecte als participants és 1. Les cardinalitats inverses són les que hi havia a l'associació original.

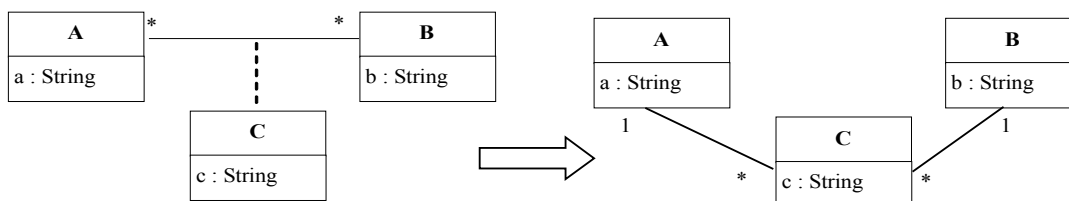


Figura 2

Caldrà controlar que no hi hagi dos objectes de C que es relacionin amb el mateix parell d'objectes d'A i B (cosa que prohibida en el diagrama original). Per ex. si A representa la classe Elecció, B la classe Persona i C l'associació Votar, en el diagrama transformat caldria controlar que no hi hagués dos objectes de votar que estiguessin relacionats amb la mateixa persona i elecció, sinó estariem permetent que una persona votés dues vegades en la mateixa elecció !!!

### 1.3 Eliminació de les associacions n-àries (n>2)

Les associacions n-àries (associacions amb més de dos participants, com per exemple les associacions ternàries) tampoc tenen una representació directa en els llenguatges O.O. El procés a seguir és exactament el mateix que en el cas anterior.

Per cada associació n-ària es crea una nova classe que es relacionarà amb els diferents participants de la associació original. Aquesta transformació s'il·lustra a la figura 3.

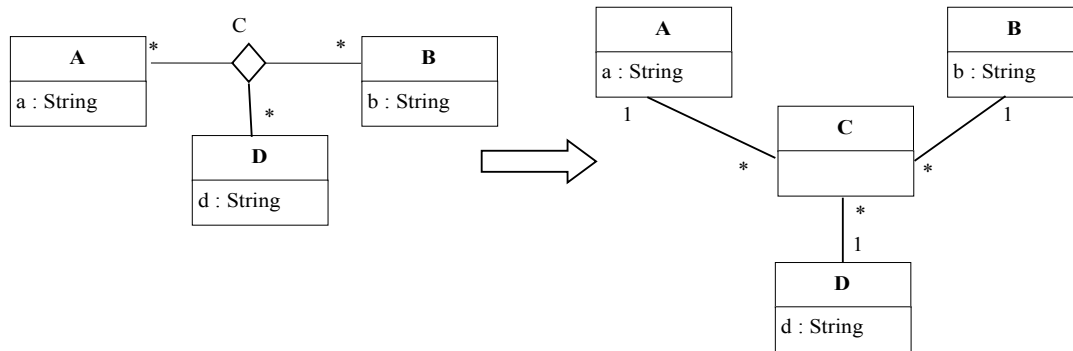


Figura 3

### 1.4 Implementació del disseny

Un cop fet aquests passos el disseny resultant sols conté herència simple (si el llenguatge de programació no admet herència múltiple) i relacions binàries entre classes. Per tant el disseny és directament implementable en el llenguatge de programació seleccionat simplement substituint les relacions per nous atributs a les classes participants que guardin (“apunten”) els objectes de l'altra classe amb les que l'objecte està relacionat.

Per ex, recuperant el disseny obtingut a la figura 1, el codi podria ser com el següent:

```
class A {  
    String a;  
    String a1;  
    String tipus;  
    B b[];  
}
```

```
class B {  
    String b1;  
    A a;
```

}

Fixeu-vos que la classe B té un atribut de tipus A per poder accedir a l'objecte d'A amb el què està relacionat. Semblantment, A té un atribut de tipus vector d'objectes de B. En aquest cas estem suposant que volem navegabilitat tant d'A cap a B com de B cap a A. Si, per exemple, sols volem que hi hagi navegabilitat d'A cap a B llavors B no tindria l'atribut a.

## **2. Disseny de la base de dades relacional**

A partir del diagrama de classes obtingut a la fase d'anàlisi, per a dissenyar la base de dades relacional, farem els següents passos:

### **2.1. Eliminació de l'herència**

Les bases de dades relacionals no ofereixen cap tipus de suport per l'herència, per tant inclús l'herència simple s'ha d'eliminar de l'esquema. El procés està explicat a l'apartat 6.2.3 dels materials

### **2.2. Obtenció del disseny de les taules**

Un cop fet el tractament de l'herència ja estem en condicions de crear el conjunt de taules relacionals que ens permetran guardar la informació del nostre sistema de forma permanent (fixeu-vos que a diferència del cas anterior, no cal que tractem explícitament les classes associatives ni les relacions n-àries).

Per fer-ho cal aplicar un conjunt de regles. Aquestes regles són una adaptació de les regles per a la transformació de models E/R que heu vist els qui heu fet l'assignatura Bases de Dades I.

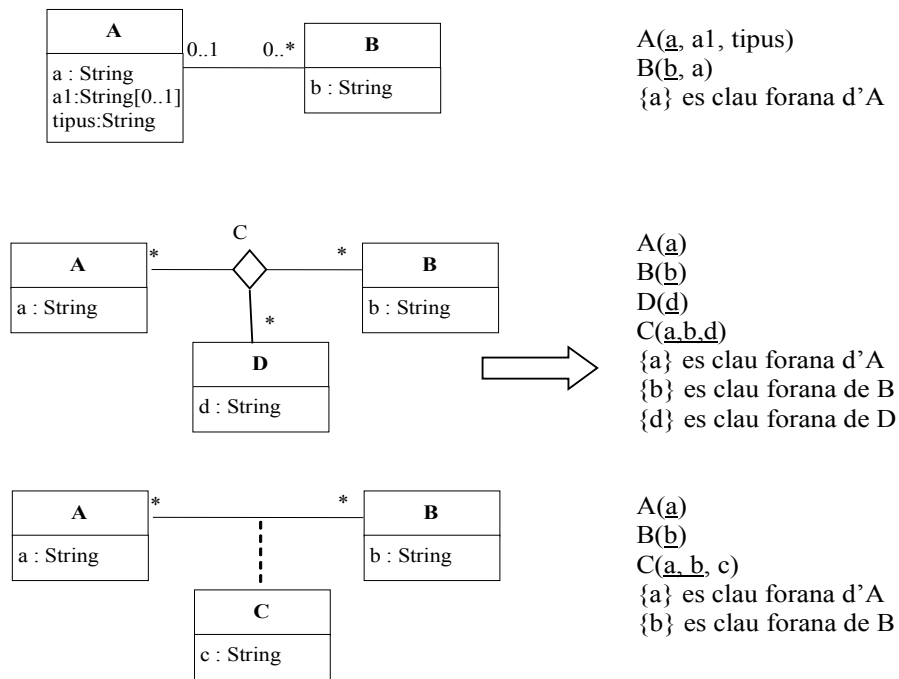
Les regles són:

1. Conversió de cada classe a una taula relacional. Els atributs de la classe passen a ser camps de la taula. Totes les taules necessiten una clau primària, que és un atribut o conjunt d'atributs que permeten identificar cada fila ("objecte") de la taula. Per exemple, el DNI (si suposem que no hi ha errors i per tant no hi ha DNIs repetits) podria ser un bona clau primària per una taula empleat. Si no trobem cap atribut o combinació d'atributs que ens faci el pes, es pot crear una clau artificial (afegint un nou camp "codi" a la taula).

2. Cada associació de tipus 1:\* (relacions on un extrem de la associació té cardinalitat 1 o 0..1 i l'altra un valor més gran que 1) es tracta afegint a la taula que representa la classe del costat \* el/s atribut/s que formen la clau primària de la taula que representa la classe del costat 1. Aquests atributs queden marcats com claus foranes. Una clau forana obliga a què el valor del nou atribut sigui vàlid, és a dir, que sigui un dels valors existents dins l'altra taula. L'altra taula queda inalterada ja que el model relacional no permet atributs amb valors múltiples (atributs de tipus vector).
3. Per cada associació \*.\* (relacions on ambdós extrems tenen una cardinalitat >1) es crea una nova taula que té com a atributs les claus primàries de les taules que representen les classes participants en la associació. Aquests atributs es marquen com a clau forana. La suma de tots ells es la clau primària de la nova taula.
4. De manera similar, cada associació n-ària es transforma en una nova taula amb n claus foranes. La clau primària es la suma de totes (o algunes) de les claus foranes. A vegades només amb la suma d'alguns dels atributs que fan de clau forana en tenim prou per identificar la taula, depèn de la cardinalitat de la associació en el disseny original.
5. El tractament de les classes associatives depèn de com sigui l'associació. Si és 1:\* els atributs de la classe associativa es poden afegir als atributs de la taula del costat \*. Si és \*.\* o n-ària s'afegeixen a la taula creada per representar l'associació.

Tot i que, com s'ha dit anteriorment, no es pretén proporcionar una visió exhaustiva de la transformació i per tant no es tindran en compte, calen altres consideracions per completar la transformació, com per exemple si els atributs poden tenir valors nuls o no (depèn de la cardinalitat mínima de les relacions), el tractament de les relacions 1:1 (que es poden tractar com si fossin 1:\* o \*.\* ) ...

Alguns exemples de l'aplicació de les anteriors regles es mostren a continuació. Enlloc de representar les taules a través de la sentència CREATE de SQL, s'opta per mostrar-les amb una sintaxi més concreta. Després del nom de la taula es posa entre parèntesis els seus camps. Els camps que formen la clau primària es mostren subratllats. A continuació indiquem quins dels camps són claus foranes.



**Figura 4**

### 3. Exemple

Per ajudar a la comprensió es resol el disseny del següent diagrama de classes obtingut a la fase d'anàlisi.

El diagrama representa un comerç que vol guardar la informació dels seus clients i treballadors. Dels clients es vol guardar el registre dels productes que ha comprat al llarg del temps (per simplificar no se'n guarda la quantitat). Molts dels productes que ven el comerç són de producció artesanal però d'altres són proporcionats per distribuïdors. Es vol saber quins són els distribuïdors de cada producte i quina és la millor oferta de cada un d'ells.

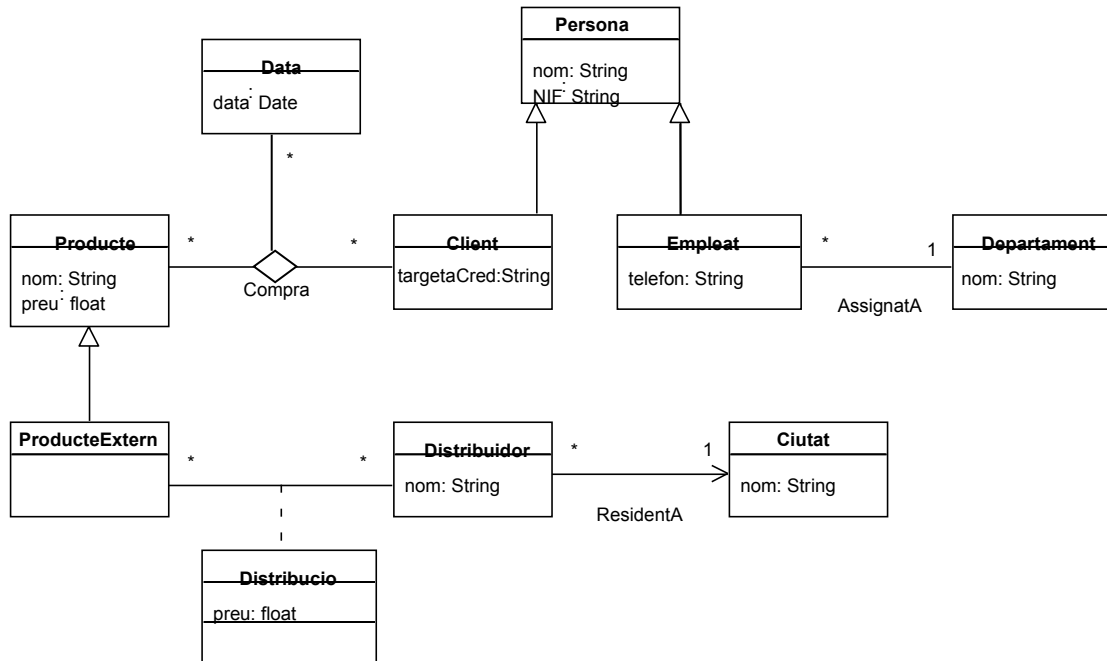
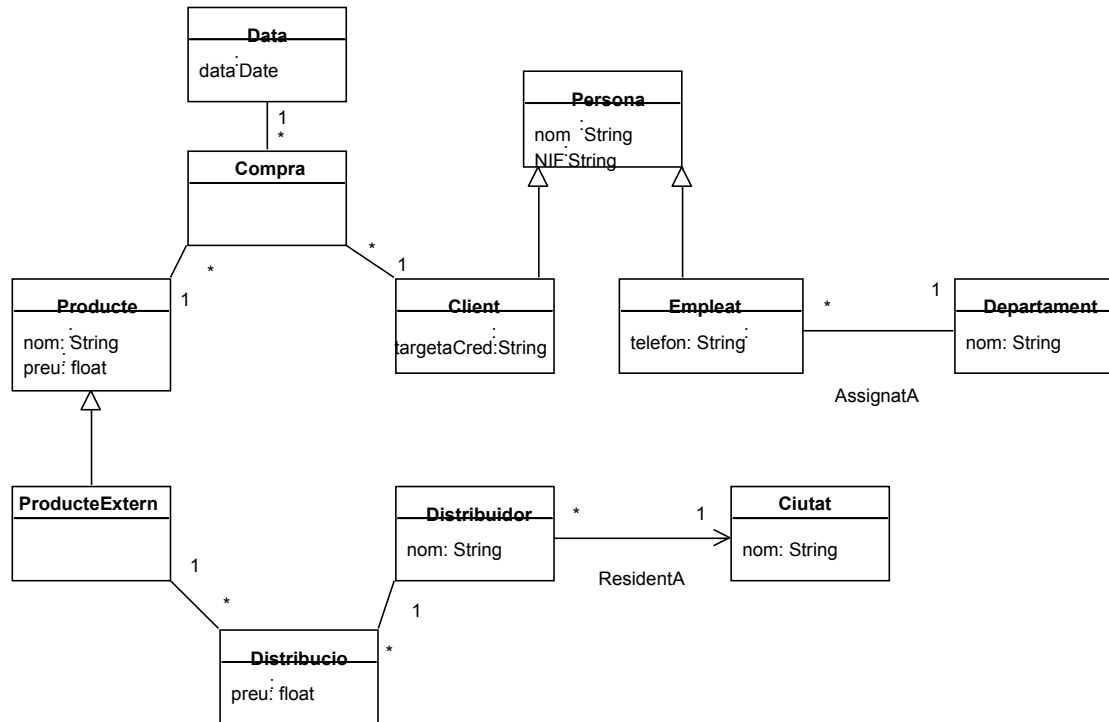


Figura 5

### 3.1. Diagrama estàtic de disseny per una tecnologia orientada a objectes

Segons l'explicat a l'apartat 1 per fer la transformació s'ha substituït la classe *Distribucio* per una nova classe relacionada amb *ProducteExtern* i *Distribuidor*. També s'ha reificat l'associació ternària *Compra*.



**Figura 6**

Un cop fet el diagrama de disseny, ja podem generar el codi Java corresponent. Fixeu-vos que a tot arreu s'assumeix navegació bidireccional menys en l'associació entre *Distribuïdor* i *Ciutat*. La classe *Data* no s'ha generat. Per optimitzar, s'ha substituït directament per un atribut de tipus *Date* dins la classe *Compra*. Per raons d'espai, no s'han inclòs el mètodes accessors *get* y *set* per cada atribut a cada classe que són necessaris per a accedir als atributs des de l'exterior de la classe.

```

public class Compra {
    private Producte producte;
    private Client client;
    private java.util.Date date;
}

public class Ciutat {
    private String nom;
}

public class Client extends Persona {
    private String targetaCred;
    private Compra compres[];
}

public class Departament {
    private String nom;
    private Empleat empleats[];
}
  
```

```

public class Distribucio {
    private float preu;
    private Distribuidor distribuïdor;
    private ProducteExtern producteExtern;
}

public class Empleat extends Persona {
    private String telefon;
    private Departament departament;
}

public class ProducteExtern extends Producte {
    private Distribucio distribucions[];
}

public class Persona {
    private String nom;
    private String NIF;
}

public class Producte {
    private String nom;
    private float preu;
    private Compra compres[] ;
}

public class Distribuïdor {
    private String nom;
    private Ciutat ciutat;
    private Distribucio distribucions[];
}

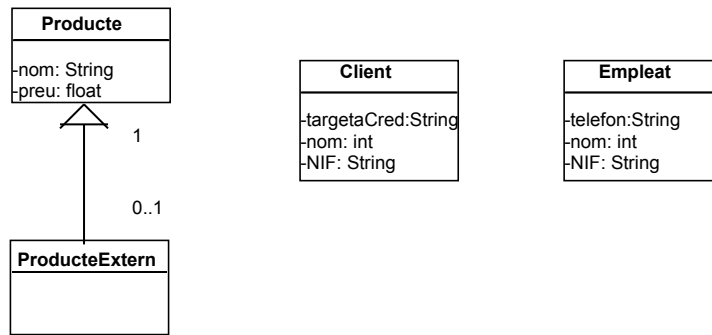
```

### 3.2. Disseny de la base de dades

En el diagrama inicial obtingut a la fase d'anàlisi (figura 5) hi ha les següents herències que cal tractar : Persona – Client, Persona-Empleat i Producte- ProducteExtern.

Pel primer cas hem optat per eliminar l'herència a través d'eliminar la superclasse i traspasar els seus atributs a cada subclasse (ja que la superclasse no participava en cap associació). Pel segon s'ha optat per mantenir tant la superclasse com la subclasse i per tant l'herència es substitueix per una associació 1:0..1 entre les dues. Així s'assegura que tot objecte de la subclasse (*ProducteExtern*) està relacionat amb un objecte de la superclasse (*Producte*), i al revés, que tot objecte de *Producte* està relacionat com a màxim amb un objecte d'*ProducteExtern*.

La resta d'elements del diagrama es mantenen igual.



**Figura 7**

Un cop fets aquests canvis, el conjunt de taules relacionals obtingut és el següent:

**Client**(NIF, nom, targetaCredit)

**Empleat**(NIF, nom, telefon, departament)

{departament} és clau forana a Departament (això implica que el valor de l'atribut *departament* de *Empleat* ha de ser igual a un dels noms de *Departament* existents a la base de dades).

**Departament**(nom)

**Producte**(nom, preu)

**Compra**(client, producte, data)

{client} és clau forana a Client

{producte} és clau forana a Producte

**ProducteExtern**(nom)

{nom} és clau forana a Producte (i en aquest cas també clau primària)

**Ciutat**(nom)

**Distribuidor**(nom, ciutat)

{ciutat} és clau forana a Ciutat

**Distribucio**(producteExtern, distribuidor, preu)

{producteExtern} és clau forana a ProducteExtern

{distribuidor} és clau forana a Distribuidor