Introducció al disseny orientat a objectes

Marcel García Vacas

Entorns de desenvolupament

Índex

Introducció

La fase de disseny, dins un projecte de desenvolupament de programari informàtic, es durà a terme després d'efectuar l'anàlisi de requeriments, i serà la fase que establirà les bases sobre com s'han de fer les coses a l'hora de desenvolupar l'aplicació. Un projecte ben planificat oferirà una fase de disseny que haurà de ser més costosa (en temps i recursos) que la pròpia fase de desenvolupament del programari. Si es fa ben fet, amb una codificació de programari automàtica o semiautomàtica, un bon disseny estalviarà molt temps d'haver de picar codi.

Aquesta fase de disseny s'haurà de dur a terme en concordança amb la metodologia escollida per al desenvolupament del projecte. Si s'ha apostat, per exemple, per una metodologia àgil, caldrà fer el disseny en funció d'aquest tipus de programació. Durant molts anys s'ha desenvolupat un disseny estructurat que dirigia cap a un tipus de programació estructurada i modular. De fet, en alguns projectes encara es fa servir aquest tipus de disseny. Igual que la programació estructurada va evolucionar cap a la programació orientada a objectes, l'evolució natural ha estat anar cap a una codificació de programari orientada a objectes, seguint, també, una anàlisi i un disseny orientats a objectes. UML permet elaborar un model del sistema que es vol automatitzar.

L'actual unitat formativa s'ha dividit en dos apartats. L'apartat "Diagrames estàtics" està enfocat a la introducció de tot allò relacionat amb el disseny de desenvolupament de programari, entrant en detall en el disseny orientat a objectes i, especialment, en els diagrames estàtics, que són els que identifiquen els elements de la modelització del sistema.

En l'apartat "Diagrames dinàmics", per la seva banda, es treballen els diagrames que indiquen el comportament del sistema i les seves interaccions. Aquests diagrames complementen els diagrames estàtics per oferir, conjuntament, un disseny complert de la solució que s'ofereix a un projecte informàtic.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Genera diagrames de classes valorant la seva importància en el desenvolupament d'aplicacions i emprant les eines disponibles en l'entorn.

- Identifica els conceptes bàsics de la programació orientada a objectes.
- Instal·la el mòdul de l'entorn de desenvolupament integrat que permet la utilització de diagrames de classes.
- Identifica les eines per a l'elaboració de diagrames de classes.
- Interpreta el significat de diagrames de classes.
- Traça diagrames de classes a partir de les especificacions de les mateixes.
- Genera codi a partir d'un diagrama de classes.
- Genera un diagrama de classes mitjançant enginyeria inversa.

2. Genera diagrames de comportament valorant la seva importància en el desenvolupament d'aplicacions i emprant les eines disponibles en l'entorn.

- Identifica els diferents tipus de diagrames de comportament.
- Reconeix el significat dels diagrames de casos d'ús.
- Interpreta diagrames d'interacció.
- Elabora diagrames d'interacció senzills.
- Interpreta el significat de diagrames d'activitats.
- Elabora diagrames d'activitats senzills.
- Interpreta diagrames d'estats.
- Planteja diagrames d'estats senzills.

1. Diagrames estàtics

El llenguatge unificat de modelització (UML) es basa en diagrames que estableixen les especificacions i documentació de qualsevol sistema. Aquests diagrames es fan servir en el desenvolupament de projectes informàtics per analitzar i dissenyar les necessitats dels usuaris finals de les aplicacions, amb la qual cosa es documenten els seus requeriments. Però també es podran utilitzar en molts més àmbits, com ara la modelització del funcionament d'una empresa, d'un departament o d'un sistema de qualsevol altre entorn.

Els diagrames que es poden fer servir per modelitzar sistemes es poden agrupar o classificar de moltes formes. Una d'aquestes classificacions diferencia entre els diagrames que corresponen a l'UML estàtic i els que corresponen a l'UML dinàmic. Els diagrames inclosos en els considerats de UML estàtic, també anomenats diagrames d'estructura, fan referència als elements que s'hauran de trobar en el sistema de modelització. Aquests diagrames fan una descripció del sistema sense tenir en compte les interaccions amb altres elements o el comportament que tenen els elements del sistema, només identifiquen aquests models i estableixen les seves característiques.

Dins aquest tipus de diagrames, el més important és el **diagrama de classes**, que especificarà totes les classes estimades al disseny orientat a objectes i que seran instanciades al codi de programació que es desenvoluparà a continuació.

En les pàgines corresponents a aquest apartat es treballaran els diagrames corresponents a l'UML estàtic, fent especial referència al diagrama de classes. També es mostra un exemple de com poder treballar amb un IDE (Eclipse) amb els diagrames UML.

1.1 Anàlisi i disseny orientat a objectes

La creació d'una aplicació informàtica es pot considerar una obra d'enginyeria. De fet, fa molts anys que es coneix amb el nom **enginyeria del programari** (o **enginyeria del *software***), considerat com una de les àrees més importants de la informàtica. L'enginyeria del programari engloba tota una sèrie de metodologies, tècniques i eines que faciliten la feina que comporten totes i cada una de les fases d'un projecte informàtic.

L'enginyeria del programari ha evolucionat de forma contínua (i ho continua fent a data d'avui), adaptant-se a les noves tecnologies i a les noves formes de desenvolupar programari. A la dècada dels 70, amb l'inici de la programació d'aplicacions com a feina generalitzada en les organitzacions, i l'aparició de la figura del programador, només hi havia un objectiu: que les aplicacions fessin el

que havien de fer optimitzant la velocitat i l'ús de memòria. En aquells primers programes ningú no es preocupava pel futur manteniment de les aplicacions per part d'altres programadors, ni tampoc que el codi fos fàcil d'entendre.

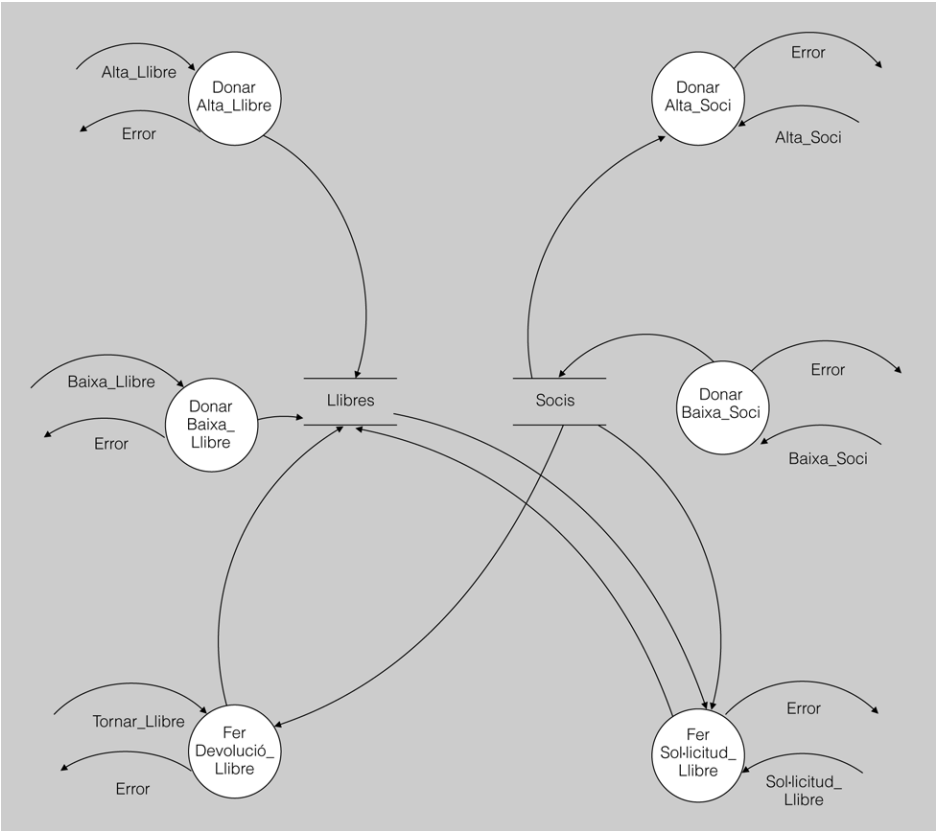
Amb els anys, les aplicacions demanen més funcionalitats i el desenvolupament d'aplicacions es fa més complex a mesura que millora el maquinari i el programari que utilitzen els programadors. La programació passa a ser programació estructurada, implementada en mòduls i més senzilla d'entendre i de mantenir.

A mitjan-finals dels anys 70 comencen a aparèixer les primeres metodologies. El fet s'elaborar el codi de programació a partir de les improvisacions o de les intuïcions del programador, sotmet el projecte a uns riscos, com ara la poca eficàcia o els errors en les funcionalitats que han de solucionar els requeriments de les especificacions.

Aquestes metodologies recullen l'experiència de molts projectes, oferint una sèrie de passos a efectuar per al correcte i exitós desenvolupament de les aplicacions informàtiques. Entre aquests passos es troben les fases en què es divideix un projecte, entre les quals hi ha la fase de disseny.

El disseny estructurat, o també anomenat disseny orientat al flux de dades, permet establir la transició del Diagrama de Flux de Dades (DFD) a una descripció de l'estructura del programa, que es representa mitjançant un diagrama d'estructures. Les eines que faciliten i automatitzen aquest tipus de disseny es coneixen com a eines CASE. A la figura 1.1 es pot observar un exemple de Diagrama de Flux de Dades.

FIGURA 1.1. Exemple DFD



L'evolució en l'enginyeria del programari sorgeix a partir dels anys 80, quan comença a aparèixer la programació orientada a objectes. La forma de programar ja no es basarà en les estructures i en la programació modular, sinó que la base seran les classes i els objectes. Igual que hi ha una evolució en la programació, aquesta evolució també es veu reflectida en les metodologies de gestió de projectes. Al cap d'uns anys d'agafar força la programació orientada a objectes i de fer-se un lloc entre els desenvolupadors d'aplicacions, van aparèixer:

- Modelització de sistemes orientats a objectes.
- Disseny orientat a objectes.
- Generació automàtica de Codi.
- Metodologies àgils.

En l'actualitat existeixen diferents tipus de tecnologies que s'utilitzen en el desenvolupament de programari, i les més utilitzades són:

- Metodologies estructurades.
- Metodologies orientades a objectes.
- Metodologies àgils.

Deixant la tercera de banda, en les altres dues (estructurades i orientades a objectes) es pot considerar que el procés de desenvolupament de programari té les mateixes fases. Segons diversos autors, pot haver-hi grans divergències entre les propostes de fases d'un projecte, però, en general, les proposades són aquestes:

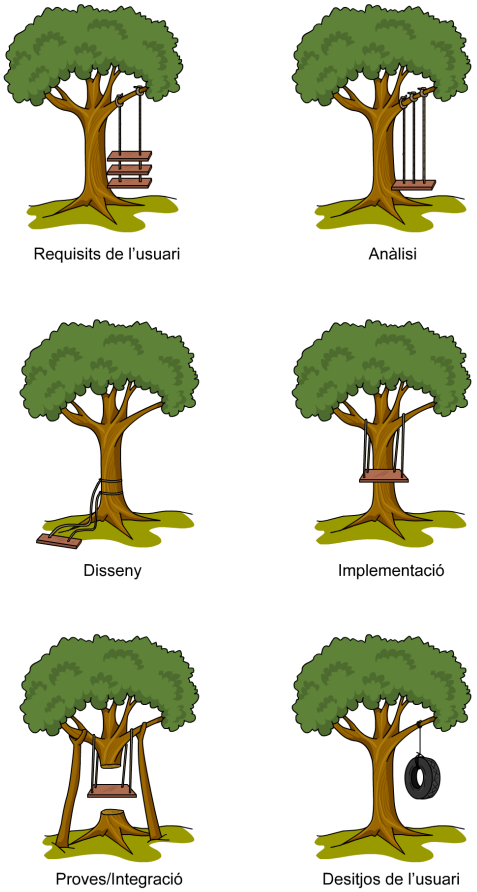
- **Estudi previ**, on es delimita l'àmbit del projecte de desenvolupament d'un programari i se'n determina la viabilitat, els riscos, els costos i el calendari.
- **Anàlisi de requeriments**, on es descriuen de manera adient els requisits del programari, que són les necessitats d'informació que el programari ha de satisfer. L'anàlisi ha de deixar molt clar el que es vol fer a partir de l'estudi d'un problema determinat. Els requeriments que cal recollir hauran ser funcionals i no funcionals. Caldrà que sigui independent de la tecnologia i del tipus de programació escollits per al desenvolupament del projecte.
- **Disseny**, on es projecta el programari amb vista a implementar-lo en un entorn tecnològic concret, definit per aspectes com ara el llenguatge de programació, l'entorn de desenvolupament i el gestor de bases de dades, de tots els quals en pot intervenir més d'un dins un sol projecte. A partir del que caldrà fer, definit a la fase anterior al disseny, es determina el com es farà. Caldrà tenir en compte les possibles conseqüències de les decisions presses en aquesta fase. El detall haurà de ser suficientment baix com perquè a partir d'aquestes indicacions es pugui desenvolupar el projecte.

- **Desenvolupament i proves**, on es porta a terme el desenvolupament del codi font que satisfaci les funcionalitats establertes a les fases anteriors. Aquesta fase no només es limita a la codificació manual, sinó que pot incloure la programació gràfica, la generació automàtica de codi i altres tècniques. En aquesta fase també s'inclouran les proves del programari. Algunes parts de la programació poden començar abans que el disseny estigui enllestit del tot, i també es poden provar parts del programari mentre encara se n'estan implementant d'altres.

- **Finalització i transferència**, on es prepara el programari per tal que estigui en explotació. Durant aquesta fase, és objecte de manteniment per eliminar errors no corregits durant la prova, introduir-hi millores i adaptar-lo a canvis tecnològics i organitzatius que es produeixin en l'entorn de treball dels usuaris.

Un model adequat evita trobar-se amb situacions com la que es mostra en la figura 1.2, on es poden observar, a mode d'exemple, les diferències d'enteniment entre les diferents parts i fases implicades en la gestió d'un projecte informàtic.

Figura 1.2. Errors a evitar en el desenvolupament d'un programari



1.2 Llenguatge unificat de modelització

UML és l'acrònim, en anglès, de *Unified Modeling Language*, és a dir, Llenguatge unificat de modelització. UML són un conjunt de notacions gràfiques que serveixen per especificar, dissenyar, elaborar i documentar models de sistemes i, en particular, d'aplicacions informàtiques. A partir d'aquestes notacions gràfiques o diagrames, l'analista i el dissenyador podran recrear les característiques que caldrà que tingui l'aplicació informàtica que els desenvolupadors hauran de crear posteriorment.

En l'actualitat és el llenguatge de modelització de sistemes més conegut i utilitzat. Gaudeix d'una acceptació gairebé universal i, entre altres beneficis, ha tingut l'efecte d'impulsar el desenvolupament d'eines de modelització gràfica del programari orientat a l'objecte. A més està reconegut i és recomanat per l'OMG.

OMG

Prové de l'anglès, Object Management Group, que és un consorci fundat l'any 1989 amb l'objectiu de fomentar l'ús de la tecnologia orientada a l'objecte mitjançant l'establiment d'estàndards.

Els avantatges de la notació UML són els següents:

- Està recolzada per la OMG (Object Management Group) com la notació estàndard per al desenvolupament de projectes informàtics.
- Es basa en una notació gràfica concreta i fàcil d'interpretar, essent completada amb explicacions escrites.
- A l'analista i/o al dissenyador els permet fer ús dels diagrames que considerin oportuns i amb el grau de detall que considerin en funció de les característiques del sistema.
- Permet tenir una visió global del sistema a implementar.
- Promou la reutilització.

Entre els desavantatges destaquen:

En la secció *Annexos* del web del mòdul hi podeu trobar més informació referent a la història i característiques de l'UML.

- UML no és una metodologia, és una notació.
- UML no és un llenguatge de programació.
- Pot resultar complex obtenir un coneixement complet de les possibilitats del llenguatge.

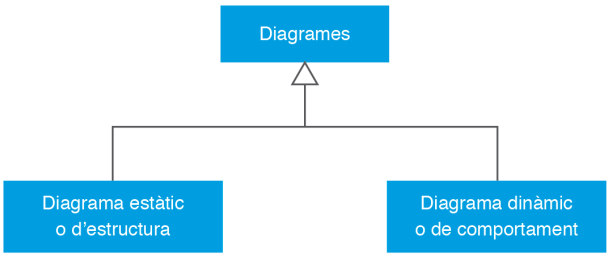
1.2.1 Diagrames

En el llenguatge de modelització UML, en la seva versió 2.4.1, existeixen un total de 14 tipus diferents de diagrames.

Com es pot observar a la figura 1.3, una possible classificació dels diagrames es fa en funció de la visió del model del sistema que ofereixen. Les dues visions diferents de model de sistema que poden representar els diagrames UML són:

- **Visió estàtica (o estructural)**: per oferir aquest tipus de visió s'utilitzen objectes, atributs, operacions i relacions. La visió estàtica d'un sistema dóna més valor als elements que es trobaran en el model del sistema des d'un punt de vista de l'estructura del sistema. Descriuen aspectes del sistema que són estructurals i, per tant, permanents (allò que el sistema té).
- **Visió dinàmica (o de comportament)**: per oferir aquest tipus de visió es dóna més valor al comportament dinàmic del sistema, és a dir, a allò que ha de passar en el sistema. Amb els diagrames de comportament es mostra com es modela la col·laboració entre els elements del sistema i els seus canvis d'estat. Representen allò que pot fer el sistema modelitzat.

FIGURA 1.3. Diagrames UML

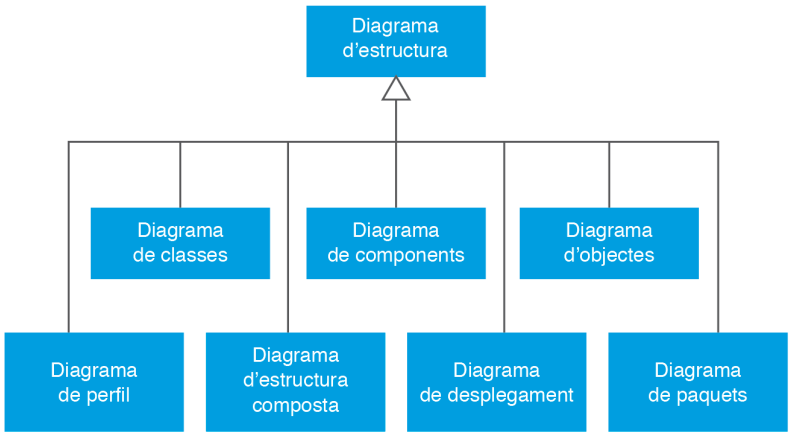


Dins els diagrames estàtics o diagrames d'estructura es poden trobar 7 tipus de diagrames, que són:

- **Diagrama de paquets**, que representa essencialment les relacions de diferents menes entre els continguts de diferents paquets d'un model.
- **Diagrama de classes**, que probablement molts consideren el diagrama principal, atès que descriu classificadors de tota mena i diferents tipus de relacions entre ells abans de fer-los servir en altres diagrames.
- **Diagrama d'objectes**, que representa instàncies de classificadors definits en un diagrama de classes previ i relacions entre elles.
- **Diagrama d'estructures compostes**, que descriu casos en què, o bé les instàncies d'un classificador tenen com a parts instàncies d'altres, o bé en el comportament executant d'un classificador participen instàncies d'altres.
- **Diagrama de components**, que és un diagrama de classes i alhora d'estructures compostes simplificat i més adient per a determinades tecnologies de programació.
- **Diagrama de desplegament**, que descriu la configuració en temps d'execució d'un programari especificat, normalment, per un diagrama de components.
- **Diagrama de perfil**, permet adaptar o personalitzar el model amb construccions que són específiques d'un domini en particular, d'una determinada plataforma, o d'un mètode de desenvolupament de programari...

A la figura 1.4 es pot observar un resum dels diagrames d'estructura.

Figura 1.4. Diagrames d'estructura o estàtics



Dins els diagrames dinàmics o diagrames de comportament es poden trobar 7 tipus de diagrames, que són:

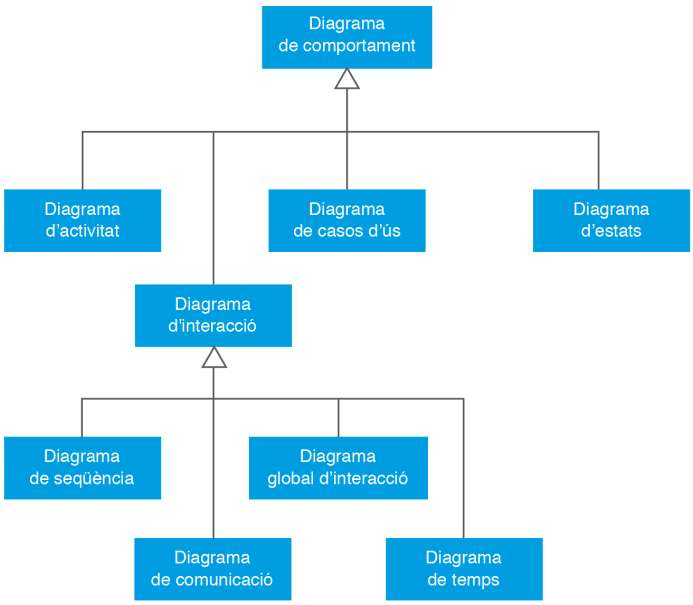
- **Diagrama de casos d'ús**, que considera els comportaments d'un sistema principalment des del punt de vista de les interaccions que té amb el món exterior.
- **Diagrama d'estats**, en el qual es descriuen les diferents situacions -estats- des del punt de vista dels seus comportaments, de les instàncies d'un classificador, alhora que les causes, condicions i conseqüències dels canvis d'una situació a una altra.
- **Diagrama d'activitats**, en què es descriu un comportament complex en forma de seqüències condicionals d'activitats components.

Diagrama d'interacció, que descriu comportaments emergents i té les variants següents:

- **Diagrama de seqüències**, que fa èmfasi en la seqüència temporal de les participacions de les diferents instàncies.
- **Diagrama de comunicacions**, que es basa directament en una estructura composta.
- **Diagrama de visió general de la interacció**, que dóna una visió resumida del comportament emergent.
- **Diagrama temporal**, que es basa en un diagrama d'estats previ o més d'un i alhora posa èmfasi en els canvis d'estat al llarg del temps.

A la figura 1.5 es pot observar un resum dels diagrames dinàmics, també anomenats diagrames de comportament.

FIGURA 1.5. UML: Diagrama de comportament o dinàmic



1.2.2 Diagrama de classes

Un dels diagrames referents de UML, classificat dins els diagrames de tipus estàtic, és el diagrama de classes. És un dels diagrames més utilitzats a les metodologies d'anàlisi i de disseny que es basen en UML.

Un diagrama de classes representa les classes que seran utilitzades dins el sistema i les relacions que existeixen entre elles.

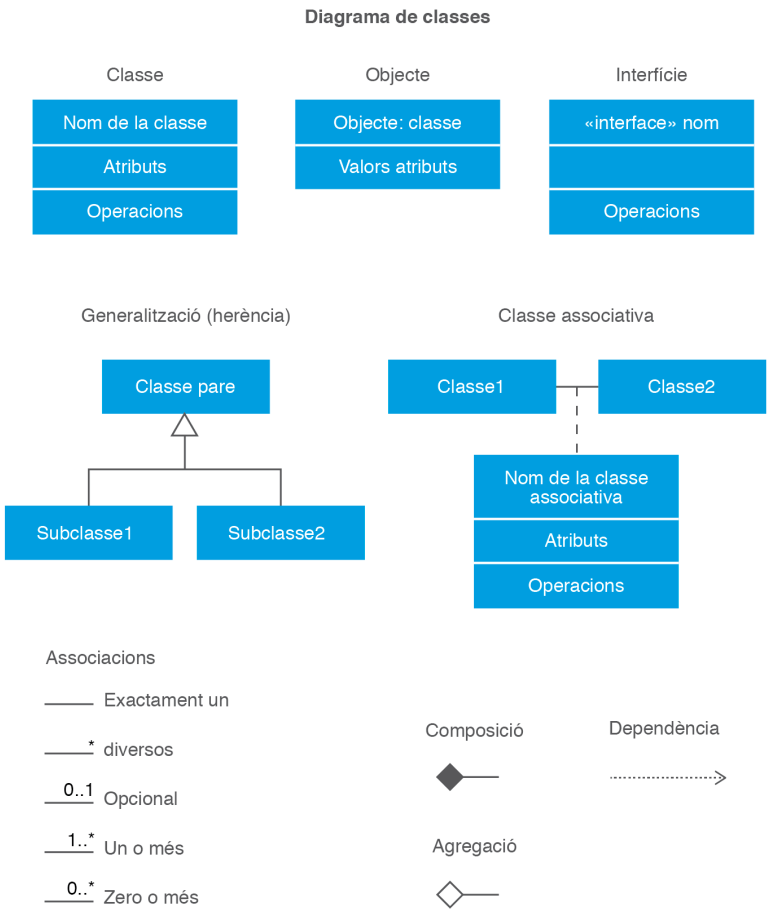
Aquest tipus de diagrames són utilitzats durant les fases d'anàlisi i de disseny dels projectes de desenvolupament de programari. És en aquest moment en què es comença a crear el model conceptual de les dades que farà servir el sistema. Per això s'identifiquen els components (amb els seus atributs i funcionalitats) que prendran part en els processos i es defineixen les relacions que hi haurà entre ells.

Un diagrama de classes porta vinculats alguns conceptes que ajudaran a entendre'n la creació i el funcionament en la seva totalitat. Aquests conceptes són:

- Classe, atribut i mètode (operacions).
- Visibilitat.
- Objecte. Instanciació.
- Relacions. Herència, composició i agregació.
- Classe associativa.
- Interfícies.

A la figura 1.6 es mostren els elements que poden pertànyer a un diagrama de classes.

Figura 1.6. Elements del diagrama de classes



Classes. Atributs i operacions

Una **classe** descriu un conjunt d'objectes que comparteixen els mateixos **atributs**, que representen característiques estables de les classes, i les **operacions**, que representen les accions de les classes.

Els atributs (també anomenats *propietats* o *característiques*) són les dades detallades que contenen els objectes. Aquests valors corresponen a l'objecte que instancia la classe i fa que tots els objectes siguin diferents entre si.

Tot atribut té assignat un tipus i pot tenir una llista formada per un valor o més d'aquest tipus, d'acord amb la multiplicitat corresponent, que han de pertànyer a aquest tipus i poden variar al llarg del temps.

Per exemple, un possible atribut de la classe persona podria ser el seu nom, atribut de tipus *string*.

Les **operacions** (també anomenades mètodes o funcionalitats) implementen les accions que es podran dur a terme sobre els atributs. Ofereixen la possibilitat d'aplicar canvis sobre els atributs, però també moltes altres accions relacionades amb l'objecte, com obrir-lo o tancar-lo, carregar-lo...

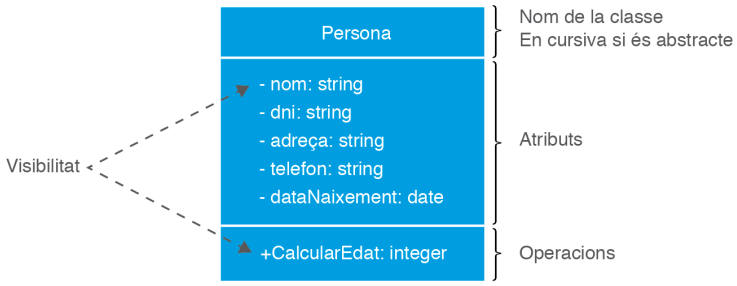
Cada **operació** té una signatura que en descriu els eventuals paràmetres i el seu valor de retorn. Els paràmetres tenen nom, tipus, multiplicitat i direcció (que indica si el paràmetre és d'entrada, de sortida, o d'entrada i sortida alhora). El valor de retorn, si n'hi ha, només té tipus i multiplicitat.

Les operacions poden tornar excepcions durant la seva execució. Una **excepció** és una instància d'un classificador que és el seu tipus. El fet que una operació torni una excepció normalment denota que s'ha produït una anomalia en l'execució.

Per exemple, un possible mètode de la classe `Persona` podria ser `CalcularEdat`, que retorna un enter.

A la figura 1.7, es mostra un breu exemple d'una classe i com es representaria.

Figura 1.7. Exemple de representació d'una classe



Aquesta representació mostra un rectangle que està dividit en tres files:

- A la primera s'indica el nom de la classe.
- A la segona s'indiquen els atributs que donen les característiques a la classe. Caldrà també indicar la seva visibilitat.
- A la tercera s'indiquen els mètodes o les operacions amb la seva declaració i visibilitat.

El codi que es generaria a partir de la classe `Persona` de la figura 1.7 seria:

```
1  Class Persona{
2  {
3       // atributs
4       private String nom;
5       private String dni;
6       private String adreca;
7       private String telefon;
8       private date dataNaixement;
9       // constructor
10      public Persona(String nom, String dni, string adreca, string telefon, date
            dataNaixement{
```

```
11        this.nom = nom;
12        this.dni = dni;
13        this.adreca= adreca;
14        this.telefon = telefon;
15        this.dataNaixement = dataNaixement;
16      }
17      // mètodes o operacions
18      public integer CalcularEdat() {
19           Date dataActual = new Date();
20           SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
21           String _dataActual = formato.format(dataActual);
22           String _dataNaixement = formato.format(dataNaixement);
23
24           String[] dataInici = _dataNaixement.split("/");
25           String[] dataFi = _dataActual.split("/");
26
27           int anys = Integer.parseInt(dataFi[2]) - Integer.parseInt(dataInici
                 [2]);
28           int mes = Integer.parseInt(dataFi[1]) - Integer.parseInt(dataInici
                 [1]);
29           if (mes < 0) {
30                anys = anys - 1;
31           } else if (mes == 0) {
32                int dia = Integer.parseInt(dataFi[0]) - Integer.parseInt(
                       dataInici[0]);
33                if (dia > 0) anys = anys - 1;
34           }
35           return anys;
36      }
37 }
```

La visibilitat

La **visibilitat** d'un atribut o d'una operació definirà l'àmbit des del qual podran ser utilitzats aquests elements. Aquesta característica està directament relacionada amb el concepte d'orientació a objectes anomenat *encapsulació*, mitjançant el qual es permet als objectes decidir quina de la seva informació serà més o menys pública per a la resta d'objectes.

Exemple de visibilitat

Una classe és un espai de noms per a les seves variables i mètodes; per exemple, un mètode de visibilitat # només pot ser invocat des de mètodes que formin part de la mateixa classe o d'alguna de les seves especialitzacions eventuals.

Les possibilitats per a la visibilitat, tant d'atributs com de mètodes, són:

- + en UML, o `public`, que vol dir que l'element és accessible per tots els altres elements del sistema.
- - en UML, o `private`, que significa que l'element només és accessible pels elements continguts dins el mateix objecte.
- # en UML, o `protected`, que vol dir que l'element només és visible per als elements del seu mateix objecte i per als elements que pertanyen a objectes que són especialitzacions.
- ~ en UML, o `package`, que només es pot aplicar quan l'objecte no és un paquet, i aleshores vol dir que l'element només és visible per als elements continguts directament o indirectament dins el paquet que conté directament l'objecte.

Per mostrar un exemple de la visibilitat, es mostra a continuació el codi de la classe `Habitació`, la classe `Persona` i la classe `Test`. Cadascuna de les classes té els seus atributs i els seus mètodes. Alguns d'aquests elements seran públics o privats, oferint les característiques que s'indiquen en finalitzar el codi.

```
public class Habitacio {
    private String dataEntrada;
    private String dataSortida;
    private Persona client;

    public Habitacio (String dataEntrada, String dataSortida, String nom) {
       this.dataEntrada = dataEntrada;
       this.dataSortida = dataSortida;
       client = new Persona(nom);
    }
    public String getDataEntrada() {
       return dataEntrada;
    }
    public String getDataSortida() {
       return dataSortida;
    }
    public Persona getClient() {
       return client;
    }
 }
class Persona{
    private String nom;

    public Persona(String nom) {
       this.nom = nom;
    }
    public String getNom() {
       return nom;
    }
    public void setNom(String nom) {
       this.nom = nom;
    }
}
public class Test {
    public static void main(String[] args) {
       Habitacio reserva = new Habitacio ("24/011/2012", "28/11/2012", "Joan
            Garcia");
       Persona client = reserva.getClient();
       String nom = reserva.getClient().getNom();
    }
}
```

Caldria fixar-se en el mètode `reserva.getClient()`, que únicament té visibilitat als atributs i als mètodes públics de la classe `Persona`, és a dir, pot consultar el nom de la persona a través del mètode públic `getNom` però no pot accedir a l'atribut privat `nom`.

D'altra banda, els mètodes `getNom` i `setNom` de la classe `Persona` tenen visibilitat als atributs i als mètodes privats de la classe, amb la qual cosa, poden accedir a l'atribut privat `nom`.

Objectes. Instanciació

Un objecte és una instanciació d'una classe. El concepte *instanciació* indica l'acció de crear una instància d'una classe.

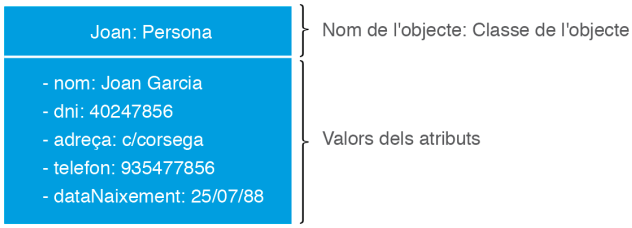
La creació d'una instància d'una classe es refereix a fer una crida al mètode constructor d'una classe en temps d'execució d'un programari.

> Un **objecte** es pot definir, llavors, com una unitat de memòria relacionada que, en temps d'execució, du a terme accions dintre un programari. És quelcom que es pot distingir i que té una existència pròpia, ja sigui de forma conceptual o de forma física.

Un objecte pot pertànyer a més d'una classe, però només d'una d'elles es poden crear objectes directament: la resta de classes representen només papers que pot exercir l'objecte.

Un possible objecte de la classe `Persona` podria ser el que es mostra a la figura 1.8.

Figura 1.8. Exemple d'un objecte de la classe "Persona"



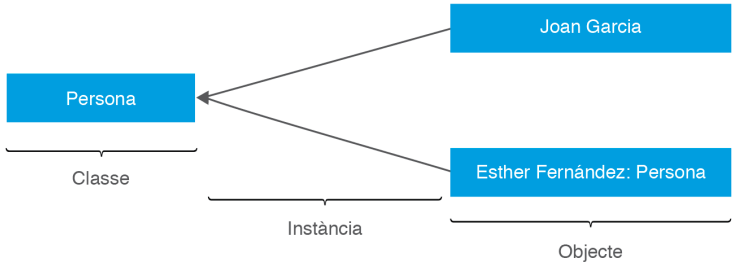
El codi per crear un objecte o instància es mostra tot seguit:

```
1 public static void main(String[] args) {
2       Persona Joan = new Persona("Joan Garcia","40782949","C/Casanoves 130",
            "93 2983924", 25/03/1977);
3 }
```

Un aspecte característic dels objectes és que tenen identitat, cosa que significa que dos objectes creats per separat sempre es consideren diferents, encara que tinguin els mateixos valors a les característiques estructurals.

En canvi, les instàncies de segons quins classificadors no tenen identitat i, alesho-res, dues instàncies amb els mateixos valors a les característiques estructurals es consideren com una mateixa instància. Aquest fet es representa a la figura 1.9.

Figura 1.9. Instanciació d'objectes de la classe "Persona"

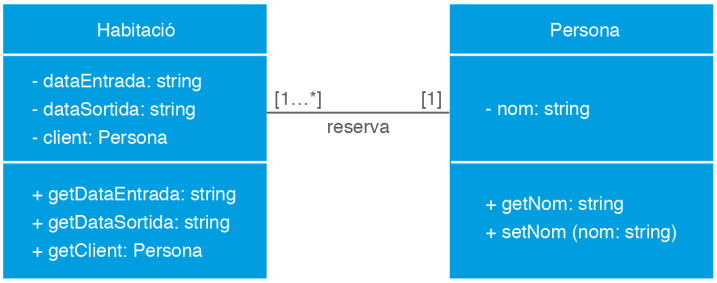


Relacions. Herència, composició, agregació

Les **relacions** són elements imprescindibles en un diagrama de classes. Per relació s'entén que un objecte `obj1` demani a un altre `obj2`, mitjançant un missatge, que executi una operació de les definides en la classe de l'`obj2`.

En l'exemple d'un client que reserva una habitació d'hotel es pot observar que intervenen dues classes, `Persona` i `Habitació`, que estan relacionades, on la classe `Habitació` consulta el nom del client corresponent a la classe `Persona`. Es veu representat a la figura 1.10.

Figura 1.10. Relació entre classes



Les relacions que existeixen entre les diferents classes s'anomenen de forma genèrica **associacions**.

Una **associació** és un classificador que defineix una relació entre diversos classificadors, que estableix connexions amb un cert significat entre les instàncies respectives.

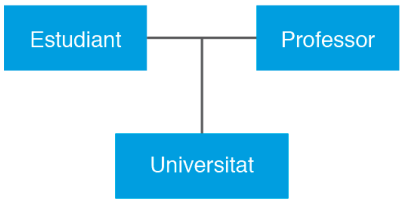
Classificador

Un classificador és un tipus els valors del qual tenen en comú característiques estructurals i de comportament, que són elements que es defineixen a nivell del classificador. Cadascun d'aquests valors és una instància del classificador.

Una associació té diversos **extrems d'associació**, a cadascun dels quals hi ha un classificador que té un cert paper en l'associació; un classificador pot ser en més d'un extrem de la mateixa associació amb papers diferents.

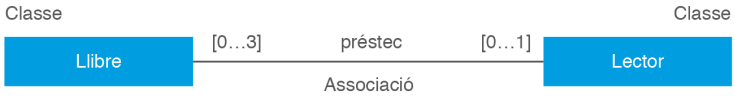
Una associació amb dos extrems es diu que és binària; seria el cas de l'exemple d'un client que reserva una habitació d'hotel. Una associació amb tres extrems es diu que és ternària. Es pot veure un exemple d'una associació ternària a la figura 1.11.

Figura 1.11. Associació ternària



La **multiplicitat**, representada per uns valors `<min....max>`, indica el nombre màxim d'enllaços possibles que es podran donar en una relació. Aquesta multiplicitat no podrà ser mai negativa, essent, a més a més, el valor màxim sempre més gran o igual al valor mínim. A la figura 1.12 es pot veure un exemple de la representació d'una associació entre dues classes amb la indicació de la seva multiplicitat.

FIGURA 1.12. Associació amb multiplicitat



Concretament, indica que, en l'associació `Préstec`, per a cada objecte de la classe `Lector` podrà haver-hi, com a mínim 0 objectes de la classe `Llibre` i, com a màxim, 3 objectes. En canvi per a cada objecte de la classe `Llibre` podrà haver-hi, com a mínim 0 i com a màxim 1 objecte de la classe `Lector`.

Es poden trobar diferents tipus de relacions entre classes. Aquestes es poden classificar de moltes formes. A continuació, es mostra una classificació de les relacions:

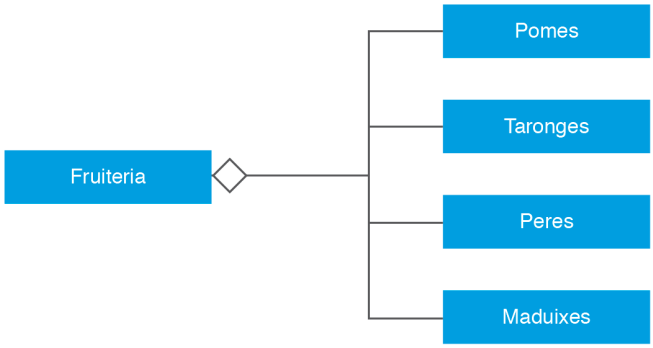
- Relació d'associació
- Relació d'associació d'agregació
- Relació d'associació de composició
- Relacions de dependència
- Relació de generalització

1) La **relació d'associació** es representa mitjançant una línia contínua sense fletxes ni cap altre símbol als extrems. És el tipus de relació (anomenada de forma genèrica associació) que s'ha estat explicant fins al moment. És un tipus de relació estructural que defineix les connexions entre dos o més objectes, la qual cosa permet associar objectes que instancien classes que col·laboren entre si.

2) Una **relació d'associació d'agregació** és un cas especial d'associació entre dos o més objectes. Es tracta d'una relació del tipus tot-part. Aquest tipus de relació implica dos tipus d'objectes, l'objecte anomenat base i l'objecte que estarà inclòs a l'objecte base. La relació indica que l'objecte base necessita de l'objecte inclòs per poder existir i fer les seves funcionalitats. Si desapareix l'objecte base, el o els objectes que es troben inclosos en l'objecte base no desapareixeran i podran continuar existint amb les seves funcionalitats pròpies. La relació d'associació d'agregació es representa mitjançant una línia contínua que finalitza en un dels extrems per un rombe buit, sense omplir. El rombe buit s'ubicarà a la part de l'objecte base. A la figura 1.13 es pot observar un exemple de relació d'associació d'agregació. L'objecte base és l'objecte anomenat `Fruiteria`. Els objectes inclosos a la fruiteria són: `Pomes`, `Taronges`, `Peres` i `Maduixes`. S'estableix

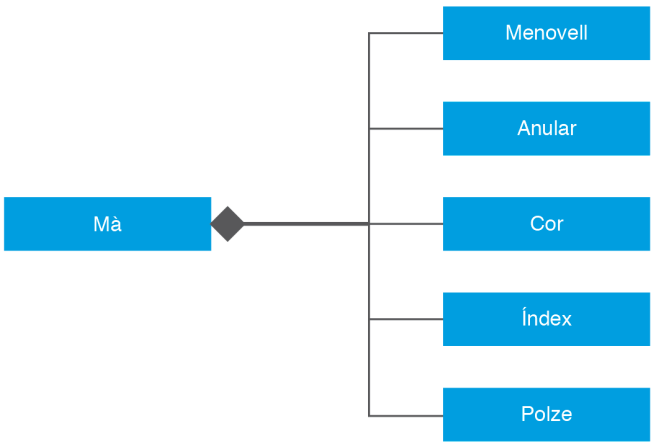
una relació entre aquestes classes del tipus tot-part, on les fruites són part de la fruiteria. Això sí, si la fruiteria deixa d'existir, les fruites continuen existint.

Figura 1.13. Relació d'associació d'agregació



3) Una **relació d'associació de composició** és també un cas especial d'associació entre dos o més objectes. És una relació del tipus tot-part. És una relació molt semblant a la relació d'associació d'agregació, amb la diferència que hi ha una dependència d'existència entre l'objecte base i l'objecte (o els objectes) que hi està inclòs. És a dir, si deixa d'existir l'objecte base, deixarà d'existir també el o els objectes inclosos. El temps de vida de l'objecte inclòs depèn del temps de vida de l'objecte base. La relació d'associació de composició es representa mitjançant una línia contínua finalitzada en un dels extrems per un rombe pintat, omplert. El rombe pintat s'ubicarà a la part de l'objecte base. A la figura 1.14 es mostra un exemple d'una relació d'agregació. L'objecte base `Mà` es compon dels objectes inclosos `Menovell`, `Anular`, `Cor`, `Índex` i `Polze`. Sense l'objecte `Mà` la resta d'objectes deixaran d'existir.

Figura 1.14. Relació d'associació de composició



4) Un altre tipus de relació entre classes és la **relació de dependència**. Aquest tipus de relació es representa mitjançant una fletxa discontínua entre dos elements. L'objecte del qual surt la fletxa es considera un objecte dependent. L'objecte al

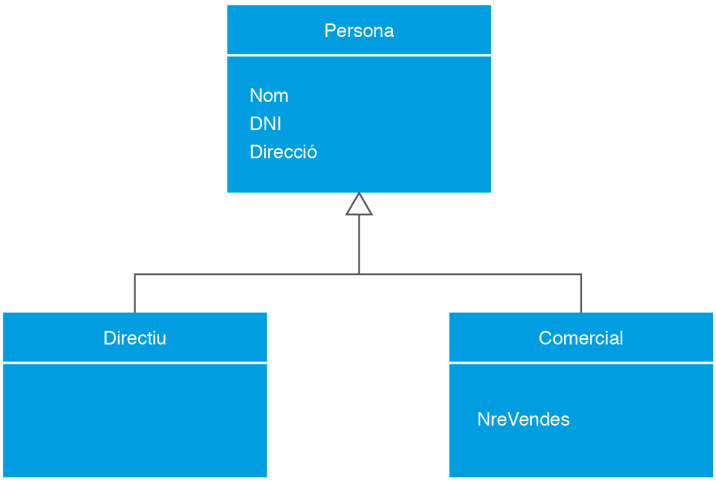
qual arriba la fletxa es considera un objecte independent. Es tracta d'una relació semàntica. Si hi ha un canvi en l'objecte independent, l'objecte dependent es veurà afectat.

5) La relació de generalització es dóna entre dues classes on hi ha un vincle que es pot considerar d'herència. Una classe és anomenada classe *mare* (o superclasse). L'altra (o les altres) són les anomenades classes *filles* o subclasses, que hereten els atributs i els mètodes i comportament de la classe *mare*. Aquest tipus de relació queda especificat mitjançant una fletxa que surt de la classe *filla* i que acaba a la classe *mare*. A la figura 1.15 es pot veure un exemple on l'element `actor` és independent. Hi ha una dependència normal de l'element `pel·lícula` en relació amb l'element `actor`, ja que `actor` es fa servir com a paràmetre als mètodes `afegir` i `eliminar` de `pel·lícula`. Si hi ha canvis a `actor`, l'objecte `pel·lícula` es veurà afectat.

Figura 1.15. Relació de dependència

L'herència es dóna a partir d'aquestes relacions de dependència. Ofereixen com a punt fort la possibilitat de reutilitzar part del contingut d'un objecte (el considerat superclasse), estenent els seus atributs i els seus mètodes a l'objecte *fill*. A la figura 1.16 es pot veure un exemple d'herència.

Figura 1.16. Relació d'herència

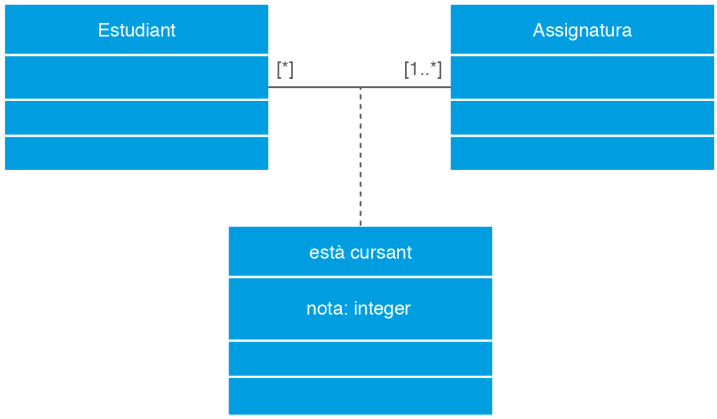


Classe associativa

Quan una associació té propietats o mètodes propis es representa com una classe unida a la línia de l'associació per mitjà d'una línia discontínua. Tant la línia com el rectangle de classe representen el mateix element conceptual: l'associació.

En el cas de l'exemple de la figura 1.17 ens trobem que la nota està directament relacionada amb les classes `Estudiant` i `Assignatura`. Cada un dels alumnes de l'assignatura tindrà una determinada nota. La manera de modelitzar l'UML aquesta situació és amb les classes associades.

Figura 1.17. Exemple de classe associativa



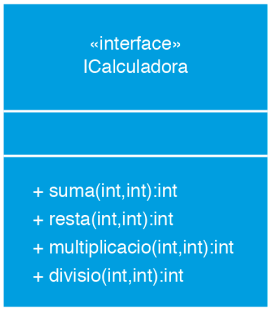
Interfície

Una **interfície** conté la declaració de les operacions sense la seva implementació, que hauran de ser implementades per una classe o component.

Arribats a aquest punt, ens podríem preguntar: Quina diferència hi ha entre una interfície i una classe abstracta?

Una interfície és simplement una llista de mètodes no implementats, així com la declaració de possibles constants. Una classe abstracta, a diferència de les interfícies, pot incloure mètodes implementats i no implementats. A la figura 1.18 es mostra un exemple d'una interfície representada en UML.

Figura 1.18. Interfície en UML



Aquesta interfície s'anomena ICalculadora, i conté els mètodes `suma`, `resta`, `multiplicacio` i `divisio`.

En el codi següent es mostra la definició d'aquesta interfície a partir de la figura 1.18.

```
interface ICalculadora {
    public abstract int suma (int x, int y);
    public abstract int resta (int x, int y);
    public abstract int multiplicacio (int x, int y);
    public abstract int divisio (int x, int y);
}
```

Una vegada definida es mosta com serà la utilització de la interfície definida en el codi següent.

```
public class Calculadora implements ICalculadora {
        public int suma (int x, int y){
            return x + y;
        }
        public int resta (int x, int y){
            return x - y;
        }
        public int multiplicacio (int x, int y){
            return x * y;
        }
        public int divisio (int x, int y){
            return x / y;
        }
}
```

1.2.3 Diagrama d'objectes

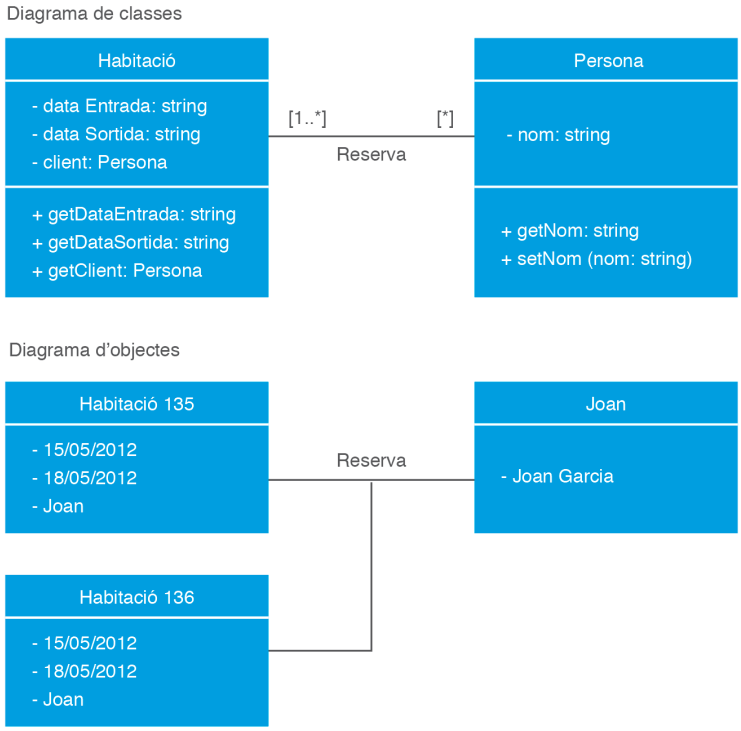
Un **objecte** és una instància d'una classe, per la qual cosa es pot considerar el diagrama d'objectes com una instància del diagrama de classes.

> El **diagrama d'objectes** només pot contenir instàncies i relacions entre objectes: enllaços i dependències que tinguin lloc entre instàncies. Els classificadors i les associacions respectives han d'haver estat definits prèviament en un diagrama de classes.

Sovint té la funció de simple exemple d'un diagrama de classes o d'una part d'ell.

A la figura 1.19 es mostra un diagrama de classes consistent en dues classes, la classe `Habitació` i la classe `Persona`. A partir d'aquest diagrama es crea i es mostra el diagrama d'objectes a la mateixa figura 1.19.

FIGURA 1.19. Diagrama d'objectes



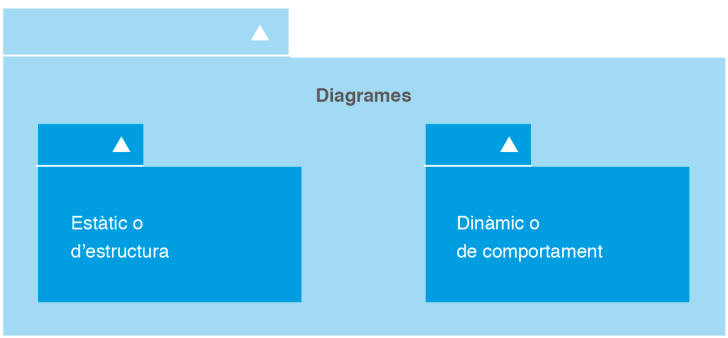
1.2.4 Diagrama de paquets

Els **paquets** permeten organitzar els elements del model en grups relacionats semànticament; un paquet no té un significat per si mateix.

> El **diagrama de paquets** serveix per descriure l'estructura d'un model en termes de paquets interrelacionats.

A la figura 1.20 es mostra un exemple de diagrama de paquets. S'hi ha representat un paquet, `Diagrames`, que conté dos paquets: `estàtic o d'estructura` i `dinàmic o de comportament`.

FIGURA 1.20. Diagrama de paquets



1.2.5 Diagrama d'estructures compostes

Una **estructura composta** és un conjunt d'elements interconnectats que col·laboren en temps d'execució per aconseguir algun propòsit.

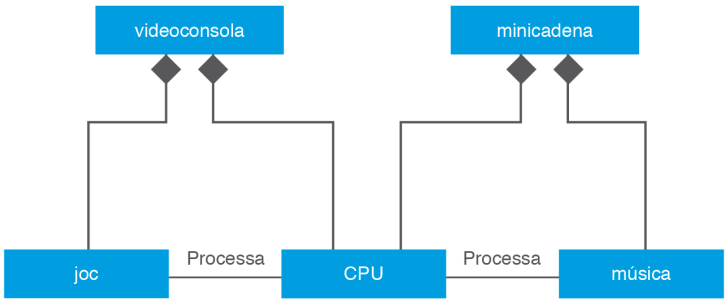
La finalitat del diagrama d'estructures compostes és descriure objectes compostos amb la màxima precisió possible. Es tracta d'un diagrama que complementa el diagrama de classes.

A continuació es descriurà, fent ús d'un exemple, el concepte del diagrama d'estructures compostes.

Si es vol modelitzar un sistema que té una videoconsola i una minicadena, ambdues estan compostes per una CPU (Unitat Central de Procés), però en la videoconsola la CPU processa el joc que serà visualitzat en algun dispositiu de sortida (com, per exemple, un televisor) i, en el segon cas, la CPU processa la música perquè pugui ser escoltada.

Una possible forma de modelitzar el sistema podria ser la que mostra la figura 1.21.

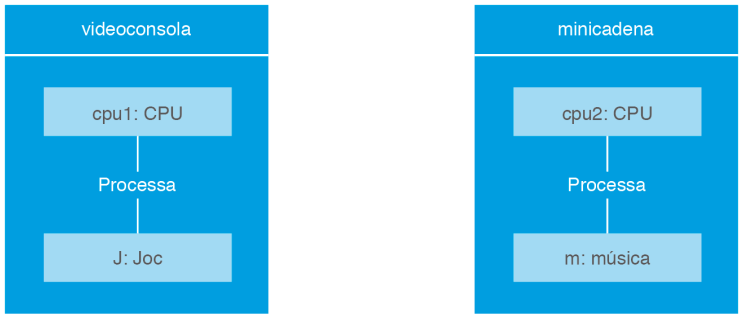
FIGURA 1.21. Diagrama de classes-composició



Però si es revisa amb detall el diagrama de la figura 1.21 es pot observar que té deficiències, ja que fàcilment es pot interpretar que la CPU de la minicadena pot processar música i jocs.

UML 2.0 ha introduït un nou diagrama, anomenat *diagrama d'estructura composta*, que permet concretar les parts que componen una classe. Aquest permetrà representar el sistema amb el diagrama que es mostra en la figura 1.22, que delimita l'abast de la videoconsola i de la minicadena.

FIGURA 1.22. Diagrama d'estructura composta



1.2.6 Diagrama de components

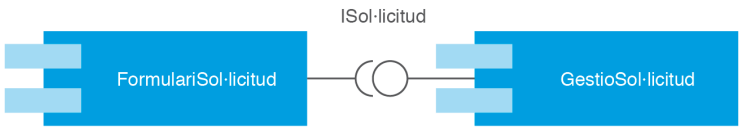
Interfície
Una interfície conté la declaració de les operacions sense la seva implementació, les quals hauran de ser implementades per una classe o component.

Un **component** és una peça del programari que conforma el sistema, peça que pot ser reutilitzable i fàcilment substituïble.

Un component sol fer ús d'una interfície que defineix les operacions que el component implementarà.

El **diagrama de components** mostra els components que conformen el sistema i com es relacionen entre si. A la figura 1.23 es pot veure un exemple d'un diagrama de components.

FIGURA 1.23. Diagrama de components



1.2.7 Diagrama de desplegament

El **diagrama de desplegament** descriu la distribució de les parts d'una aplicació i les seves interrelacions, tot en temps d'execució.

El diagrama de desplegament descriu la configuració d'un sistema de programari en temps d'execució, en termes de recursos de maquinari i dels components de programari, processos i objectes (en memòria o emmagatzemats en bases de dades, per exemple) que s'hi hostatgen.

Tot seguit es mostren diversos exemples de diagrames de desplegament.

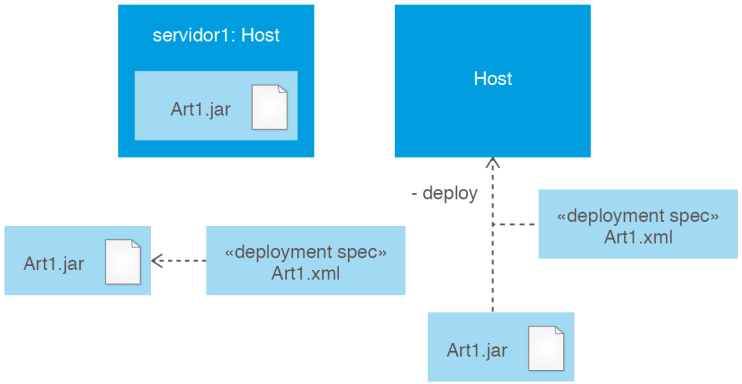
En la figura 1.24 hi ha els nodes `Servidor` i `Client`, considerats a nivell de classificador, i una línia de comunicacions. En aquesta, cada instància de Client

pot bescanviar informació només amb una instància de `Servidor`, mentre que una instància de `Servidor` en pot bescanviar informació almenys amb una instància de `Client`, però podrà fer-ho amb més.

Figura 1.24. Nodes i línia de comunicacions

Que un artefacte es desplegui dins d'un node es pot representar, o bé incloent el símbol de l'artefacte dins el del node, o bé amb el desplegament de l'artefacte cap al node. La figura 1.25 mostra un exemple de cada opció, la primera amb instàncies i la segona amb classificadors.

Figura 1.25. Un desplaçament i els seus elements



1.3 Modelio i UML: notació dels diagrames de classes

Per poder entendre millor els diagrames de classes i la seva notació, es mostra a continuació un exemple complet. A més a més, s'ha fet servir l'entorn de modelatge Modelio per a la creació d'aquests diagrames UML. Aquest entorn està especialitzat en el modelatge amb UML i altres llenguatges gràfics. També permet tant generar codi a partir dels diagrames com generar diagrames a partir de codi font. L'entorn Modelio ha estat implementat a partir de l'IDE Eclipse.

La pàgina web de l'entorn Modelio és www.modelio.org. Podeu descarregar-vos l'instal·lable des de l'apartat *Downloads*. Veureu que els requisits (que s'enllacen a la mateixa pàgina) no són gaire exigents. L'entorn Modelio pot utilitzar-se des dels sistemes operatius Windows, Linux i MacOS X. En aquest darrer cas, no existeix programa d'instal·lació i aquesta es limita a descomprimir els fitxers. Es farà servir el format d'exercici per oferir les indicacions, pas a pas, de com crear un diagrama de classes a partir dels requeriments d'un enunciat.

Enunciat de l'exemple

Les factures dels proveïdors d'una empresa poden ser de proveïdors de serveis o de proveïdors de productes o materials. Cadascuna de les factures, de qualsevol dels dos tipus, tenen en comú:

- El número de la factura.
- La data de la factura.
- L'import total –que es calcula de manera diferent en les unes que en les altres-.
- Les dades del client, que són el NIF i el nom, i que s'hauran de trobar en una classe a part.
- El detall de la factura (tant si és de materials com si és de serveis).

Tant per a les factures de serveis com per a les factures de productes o materials cal tenir guardada la darrera factura emesa, per tenir present el número de factura. Hi ha una llista de serveis amb la descripció i el preu per hora de cadascun.

En el detall de les factures haurà d'haver-hi:

- A les factures de serveis, a cada factura hi ha una llista de serveis amb la data de prestació, el nombre d'hores dedicades i el preu/hora per servei. En una factura hi pot haver més d'una prestació del mateix servei.
- A les factures de productes, per a cada producte (a cada factura n'hi ha almenys un) haurà d'haver-hi la descripció, el preu unitari i la quantitat.

1.3.1 Creació del projecte

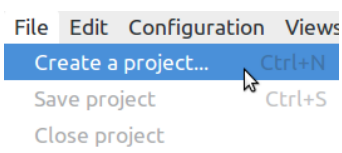
El primer cop que obriu el programa, us apareixerà una finestra de benvinguda. Podeu tancar-la després de, opcionalment, explorar les opcions que ofereix.

Un cop tancada aquesta finestra, cal seguir les següents passes:

- Seleccioneu l'opció del menú *File* / *Create a Project...* tal com es mostra a la figura 1.26.

De manera alternativa, també es pot crear un projecte fent clic a la icona de Crear Projecte

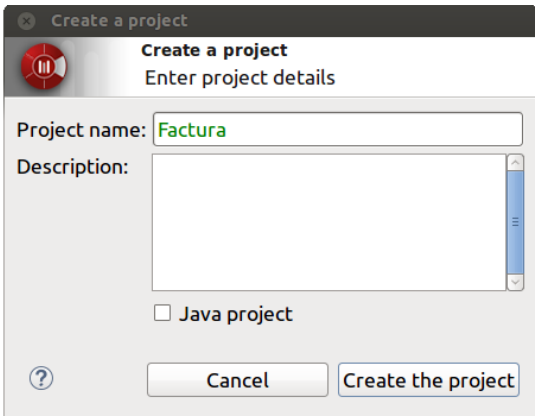
FIGURA 1.26. Opció Create a Project...



L'opció *Java project* activa el dissenyador Java. La seva finalitat és tenir sincronitzat el codi Java amb els diagrames UML. En aquest exemple no utilitzarem aquesta opció.

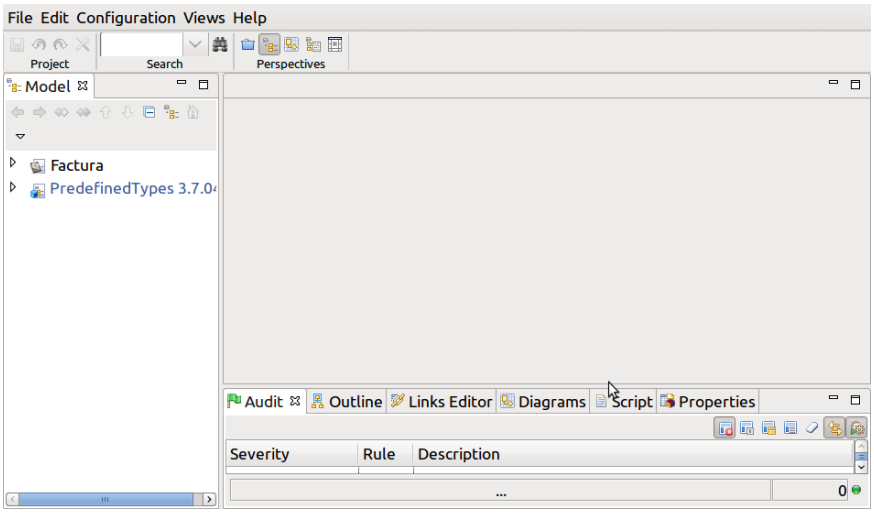
- Ens apareixerà una finestra com la que ens mostra la figura 1.27. Entreu-hi el nom, si voleu, una descripció i feu clic al botó *Create the projecte*

FIGURA 1.27. Definició del projecte



La pantalla resultant es mostra a la figura 1.28

FIGURA 1.28. Projecte nou

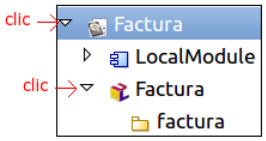


1.3.2 Creació del diagrama de classes

Un cop creat el projecte *Factura*, a la mateixa pantalla que mostra la figura 1.28, podem crear un diagrama de classes seguint les següents passes:

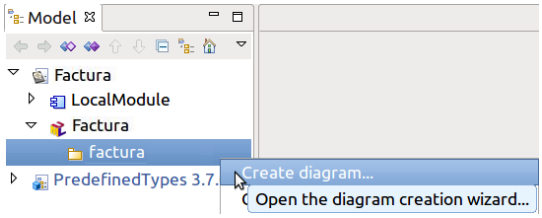
- Fer clic als triangles que hi ha a l'esquerra de les icones, de manera que aparegui la icona amb forma de carpeta que representa el projecte, tal com es mostra a la figura 1.29.

FIGURA 1.29. Accés a la carpeta que representa el projecte



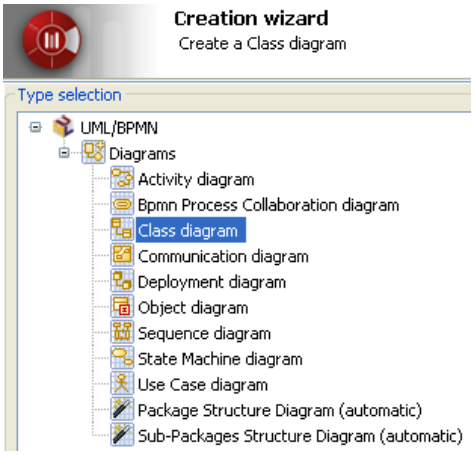
- Fer clic amb el botó secundari a sobre de la icona que representa el projecte (*Factura*, al nostre cas) i seleccionar *Create diagram...*, tal com es mostra a la figura 1.30

FIGURA 1.30. Creació d'un diagrama



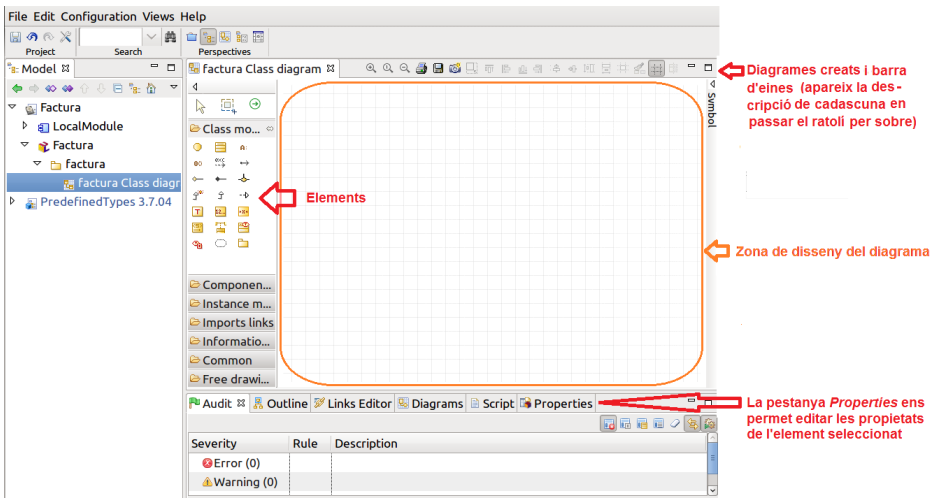
- Triar el tipus de diagrama que vulguem -*Class diagram*, al nostre cas- i fer clic a OK, tal com es mostra a la figura 1.31

FIGURA 1.31. Selecció del tipus de diagrama a crear



Ens apareix a la finestra principal una pestanya amb el nou diagrama. La figura 1.32 mostra les diferents seccions de la pantalla, un cop creat aquest nou diagrama.

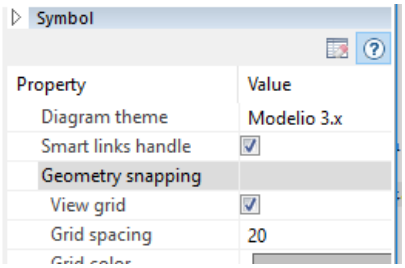
FIGURA 1.32. Seccions d'un diagrama



Per últim, és recomanable deshabilitar la característica *Smart links handle*, ja que no la utilitzarem i, a més, ens pot dificultar les accions. Es fa de la següent manera:

- Fer clic a una zona del diagrama on no hi hagi cap element.
- Fer clic a la columna *Symbol*, que es troba a la part dreta de la finestra, perquè es desplegui. La figura 1.33 ens mostra les propietats d'aquesta columna.
- Deshabilitar la propietat *Smart links handle* i, si volem, tornar a fer clic a la barra superior de la columna *Symbol* perquè es torni a plegar.

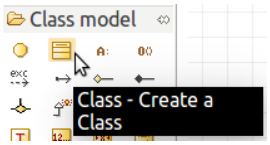
FIGURA 1.33. Propietats de la columna Symbol



1.3.3 Creació de classes. Relació d'herència

L'enunciat especifica: "cadascuna de les factures d'una empresa és, o bé una factura de serveis o bé una factura de productes". Podem observar que es tracta d'una generalització o herència, on la classe pare correspon a `Factura` i les classes *filles* a `Factura de serveis` i `Factura de productes`.

En primer lloc, cal crear les classes. Això es fa:

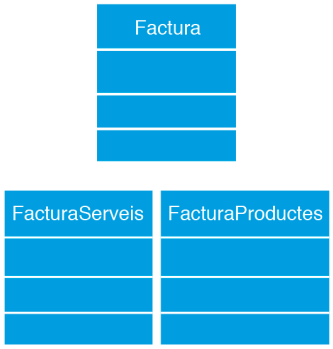


Icona Class

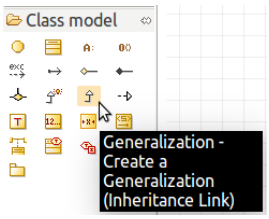
- Fent clic a sobre de la icona *Class*.
- Fent clic, a continuació, al punt del diagrama on volem inserir la classe.

Això cal fer-ho per a cadascuna de les tres classes. A la figura 1.34 es mostren aquestes classes.

FIGURA 1.34. Classes Factura, FacturaServeis i FacturaProducte



Queda pendent crear la relació d'herència entre les classes. Es fa de la següent manera:

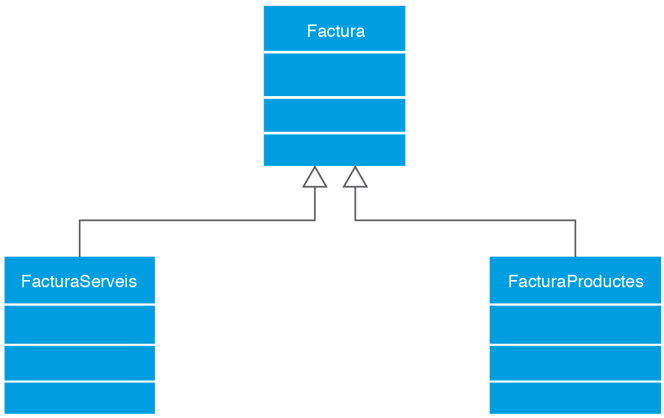


Icona Generalization

- Fent clic a la icona *Generalization*.
- A continuació, fent clic, **en aquest ordre** a una de les subclasses, per exemple a `FacturaServeis`, i, després, a la superclasse, `Factura`. Ens apareixerà una fletxa.
- Després, fent clic novament a la icona *Generalization*.
- Per últim, fent clic a l'altra subclasse, `FacturaProductes` i, a continuació, a la fletxa que ens ha aparegut al segon pas. Quan passem el ratolí per sobre, veureu que es posa de color verd; això significa que s'hi pot fer clic.
- Si hi hagués més subclasses, caldria repetir els dos passos anteriors per a cada subclasse.

El diagrama resultant és el que es mostra en la figura 1.35.

FIGURA 1.35. Relació d'herència entre classes



Per facilitar la realització dels diagrames, pot ser útil conèixer algunes operacions bàsiques que podem realitzar sobre els seus elements:

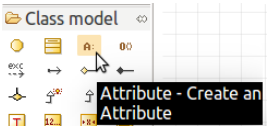
- **Seleccionar un element:** n'hi ha prou amb fer-hi clic a sobre o, si volem seleccionar-ne més d'un a la vegada, podem bé fer clic a sobre d'ells, un rera l'altre, mentre mantenim la tecla *Ctrl* polsada, bé assenyalar, tot arrosegant el ratolí, una àrea que inclogui a tots els elements a seleccionar.

- **Canviar el nom d'un element:** seleccionarem l'element al qual volem canviar el nom i, a continuació, farem clic a sobre del seu nom; una altra forma de fer-ho és seleccionar l'element i fer clic novament al mateix element; s'obrirà una finestra que ens permetrà canviar les propietats de l'element, entre elles, el nom.

1.3.4 Creació dels atributs d'una classe

Si continuem interpretant l'enunciat, ens indica: "les unes i les altres tenen en comú el número, la data i l'import". D'aquesta manera, els atributs de la classe `Factura` són: `número`, `data` i `import`.

Modelio permet afegir els atributs a les classes seguint els següents passos:



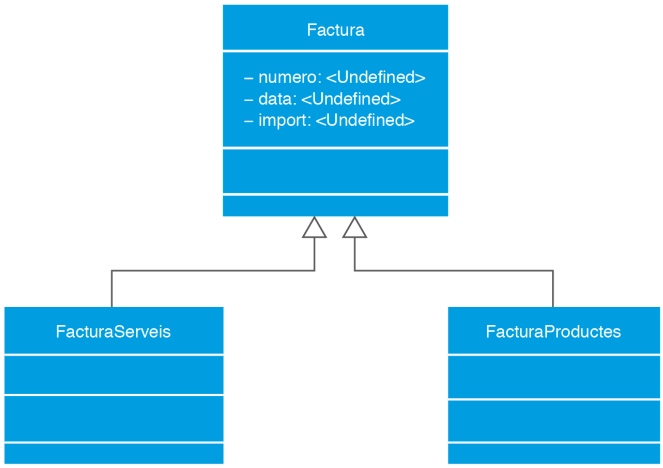
Icona Attribute

- Fer clic a sobre de la icona *Attribute*.

- A continuació, fer clic a sobre de la **classe** on volem afegir aquesta propietat.

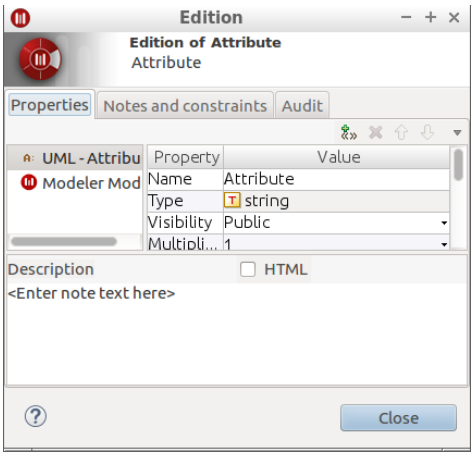
En la figura 1.36 es pot observar com ha de quedar el diagrama de classes.

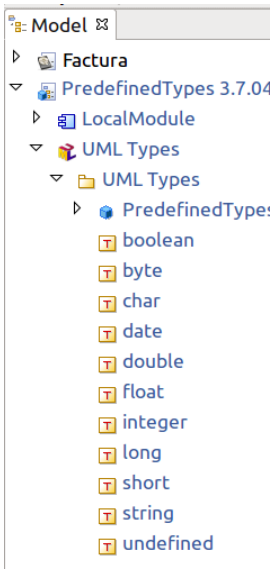
FIGURA 1.36. Atributs de la classe Factura



Ara cal especificar, per a cada atribut, el seu tipus i la seva visibilitat. Aquesta darrera, en principi, serà *private* a tots els casos. Per aconseguir-ho, cal fer doble clic a l'atribut que volem editar. Sortirà una finestra com la que mostra la figura 1.37.

FIGURA 1.37. Edició d'un Attribute





Tipus UML predefinits

A la pestanya *Properties* podrem editar el tipus, la visibilitat i altres característiques, que deixarem amb els seus valors per defecte. En fer-ho, cal tenir en compte el següent:

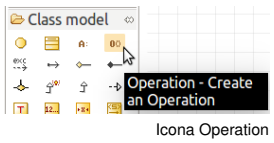
- **Tipus (Type):** podem escriure qualsevol dels tipus UML predefinits. Els podeu veure a la pestanya Model de la part esquerra de la pantalla.
- Després d'entrar el nom del tipus, cal picar la tecla d'introducció per a que el sistema validi el tipus i, a més, enregistri el canvi realitzat. També pot entrar-se el nom del tipus deixant el quadre en blanc o amb la o les primeres lletres del tipus i, a continuació, picar la combinació de les tecles *Ctrl* i *barra espaiadora*; això farà apareixer un desplegable amb els tipus que comencen amb les mateixes lletres que hem entrat.

1.3.5 Creació dels mètodes d'una classe

Modelio anomena *operations* al que per a nosaltres són mètodes.

L'enunciat ens indica que l'import es calcula de manera diferent en les unes que en les altres, és a dir el càlcul de l'import serà diferent per a la classe `FacturaServeis` i per a la classe `FacturaProductes`.

D'aquesta manera, el mètode `CalculImport()` de la classe `Factura` es definirà com un mètode abstracte que serà definit en cada una de les classes filles.



Icona Operation

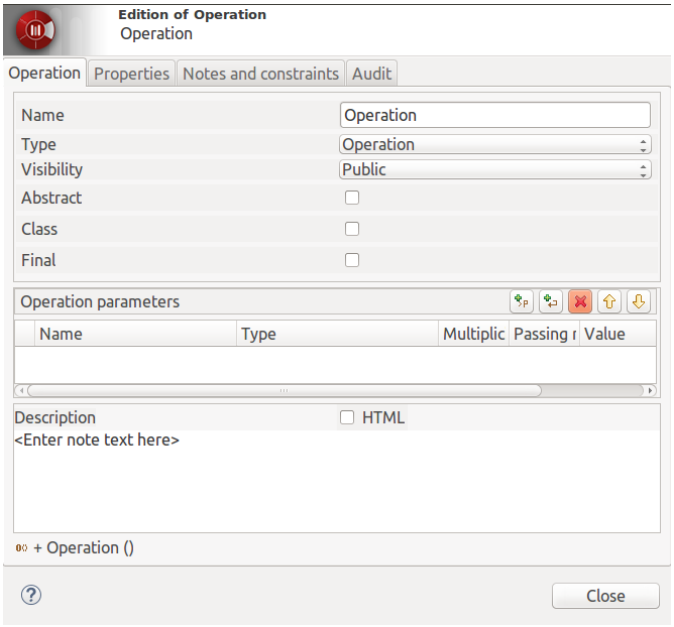
La definició dels mètodos es fa de manera similar a la dels atributs:

- Es fa clic a sobre de la icona *Operation*.
- Es fa clic a sobre de la classe on volem posar aquest mètode.

Ara cal indicar el nom del mètode, els paràmetres i el valor de retorn. Es fa des de la finestra d'edició. S'accedeix a ella fent doble clic a sobre del seu nom.

Ens apareixerà una finestra com la que indica la figura figura 1.38, amb la pestanya *Operation* oberta.

Figura 1.38. Edició d'un mètode

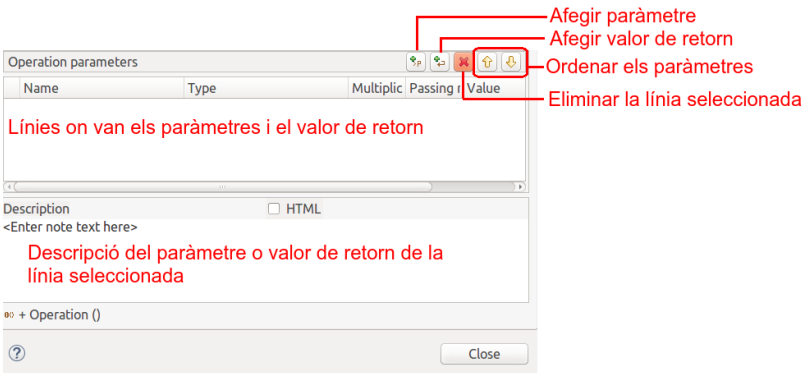


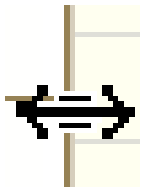
En ella podrem especificar:

- el nom
- la visibilitat
- si és abstracta
- si és final
- els paràmetres
- el valor de retorn

Les icones que apareixen a l'apartat *Operation parameters* ens permeten editar els paràmetres i el valor de retorn. Cadascun té la funció que es mostra a la figura 1.39.

Figura 1.39. Edició dels paràmetres i valor de retorn d'un mètode



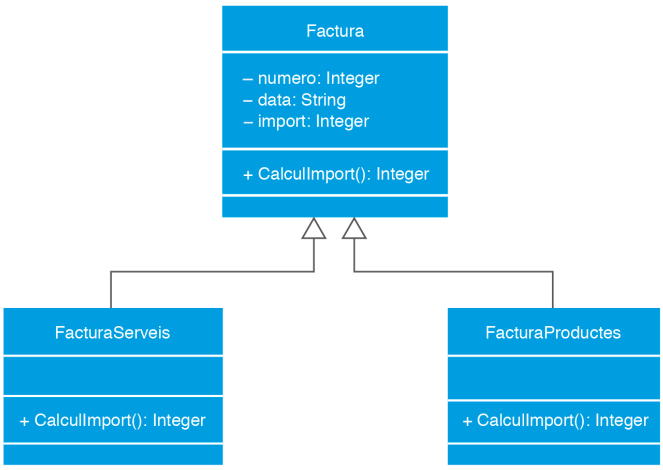


Forma del cursor del ratolí quan està a sobre d'un d'aquests requadres negres i indica que es pot reajustar la classe.

Després d'afegir un mètode a una classe, cal assegurar-se que aquesta té prou amplada per a mostrar tots els seus paràmetres; en cas contrari, cal reajustar la seva mida. Això es fa seleccionant la classe i arrossegant amb el ratolí un dels petits quadres negres que hi apareixen.

D'aquesta manera, el diagrama resultant analitzat fins al moment és el que es mostra en la figura 1.40.

FIGURA 1.40. Diagrama de classes

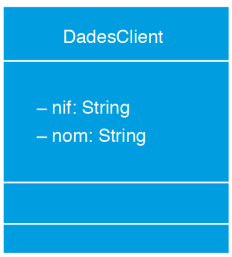


1.3.6 Agregació

Seguint amb l'enunciat, es diu: "les dades del client, que són el NIF i el nom, i són en una classe a part".

És necessari crear una nova classe anomenada `DadesClient`, que contindrà els atributs `NIF` i `nom`. Es pot observar el diagrama d'aquesta classe en la figura 1.41.

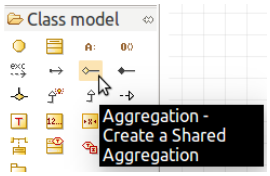
FIGURA 1.41. Classe DadesClient



El tipus d'associació entre la classe `Factura` i `DadesClient` és d'agregació, ja que la classe `DadesClient` és un objecte que únicament podrà ser creat si existeix l'objecte agregat `Factura` del qual ha de formar part.

Amb Modelio es crea seguint els següents passos:

- Fer clic a la icona *Aggregation* de la paleta.



Icona Aggregation

- Fer clic, en primer lloc, a la classe `Factura` i, després, a la classe `DadesClient`.

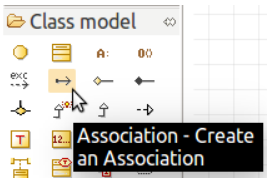
- Fer doble clic a la línia que representa l'agregació i editar, segons necessitem, les propietats de l'agregació (especialment, les propietats *navegable* i les cardinalitats).

- Si s'hagués d'incloure més classes a l'agregació, caldria:

  – Fer clic a la icona que representa una associació (*Association*).

  – Fer clic a la classe que volem afegir a l'agregació.

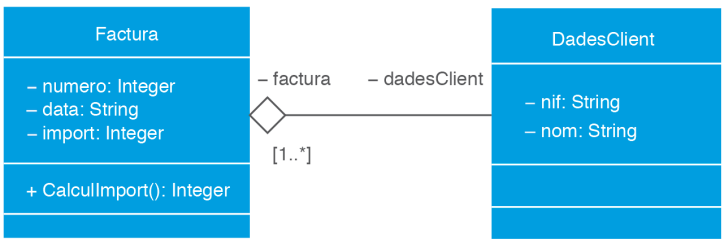
  – Fer clic a la línia que representa l'agregació.



Icona Association

El diagrama resultant analitzat fins al moment és el que es mostra en la figura 1.42.

Figura 1.42. Diagrama de classes

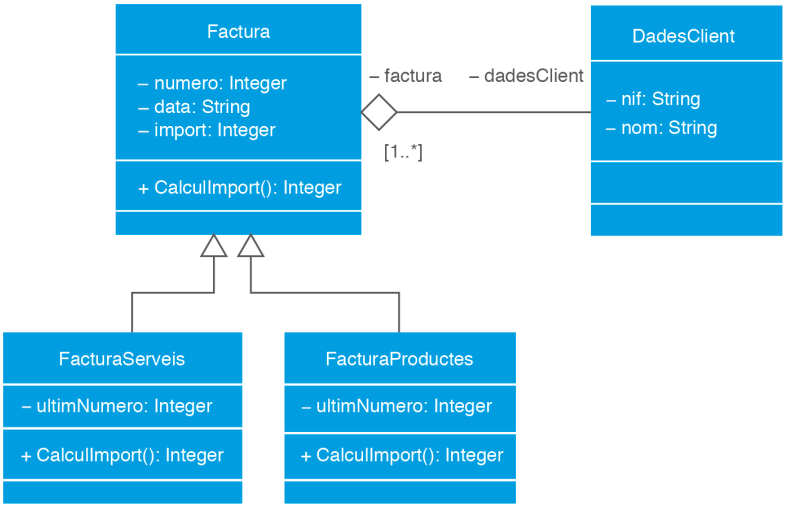


D'altra banda, l'enunciat especifica: "es guarda, d'una banda, l'últim número de factura de serveis i, de l'altra, l'últim número de factura de productes".

És necessari crear un nou atribut ultimNumero en la classe `FacturaServeis` i en la classe `FacturaProductes`.

El diagrama resultant analitzat fins al moment és el que es mostra en la figura 1.43.

FIGURA 1.43. Diagrama de classes

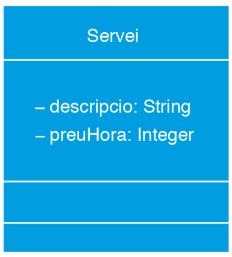


1.3.7 Classe associada

L'enunciat continua explicant: "Hi ha una llista de serveis amb la descripció i el preu per hora de cadascun".

Per tant, és necessari crear una nova classe anomenada `Servei` que contindrà dos atributs: `descripcio` i `preuHora` (preu per hora), com es pot observar en la figura 1.44.

FIGURA 1.44. Classe Servei



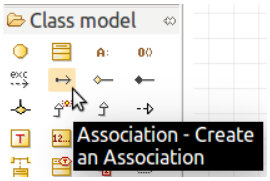
L'enunciat continua: "cada factura de serveis té una llista de serveis amb la data de prestació i el nombre d'hores, i en una factura hi pot haver més d'una prestació del mateix servei".

Per tant, és necessari crear una classe associada anomenada `Hores` que contindrà dos atributs: `dataPrestació` (data de prestació) i `nombre` (nombre d'hores), com es pot observar a la figura 1.45.

Figura 1.45. Classe Hores

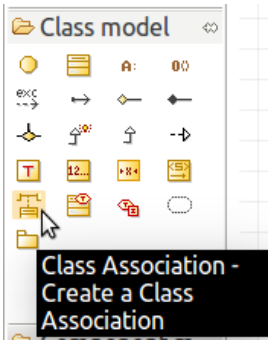
Ara cal crear l'associació entre les classes:

- Crearem una associació entre les classes `Servei` i `FacturaServeis` així:
  - Farem clic a la icona *Association*.
  - A continuació, farem clic a les dues classes consecutivament.



Icona Association

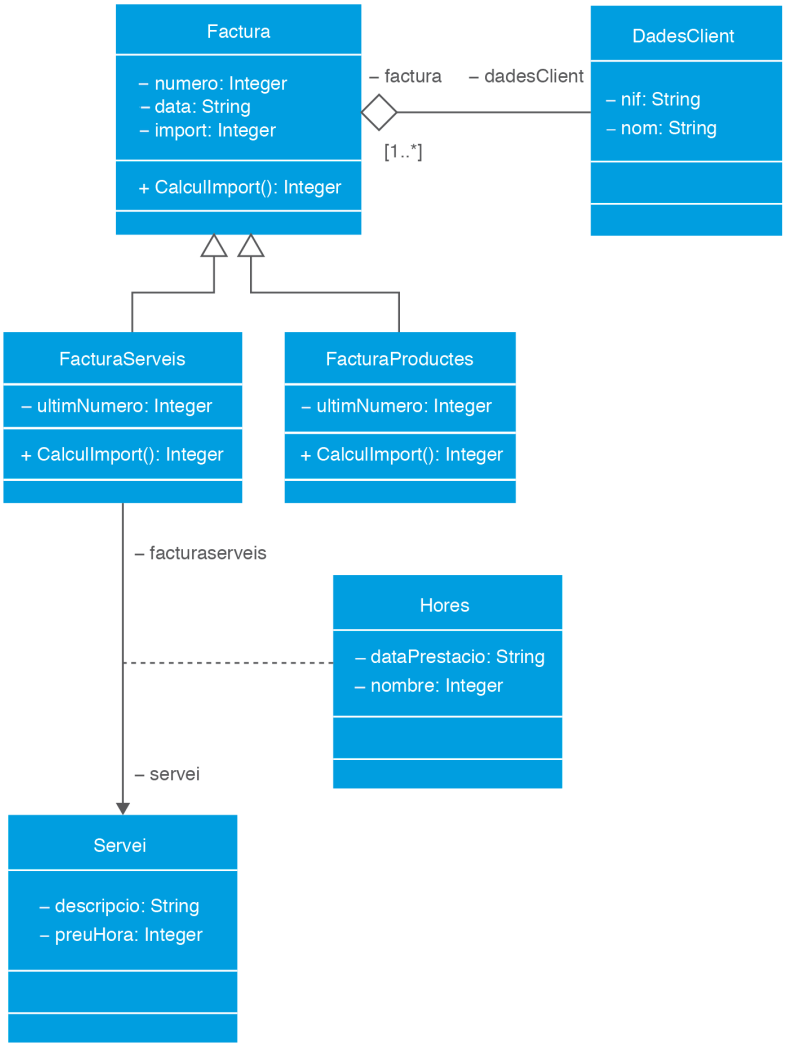
- Indicarem que la classe `Hores` és la classe associada:
  - Farem clic a la icona *classe associativa*.
  - Farem clic a la associació que acabem de crear.
  - Farem clic a la classe `Hores`.



Icona Classe associativa

El diagrama resultant és el que es mostra en la figura 1.46.

FIGURA 1.46. Diagrama de classes

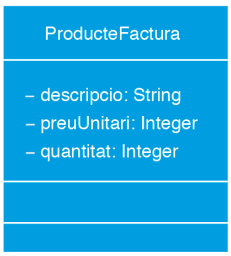


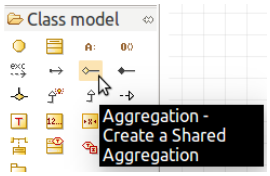
1.3.8 Composició

Continuant amb l'enunciat, aquest ens especifica: "Les factures de productes, per a cada producte (a cada factura n'hi ha almenys un) tenen la descripció, el preu unitari i la quantitat".

Per tant, és necessari crear una nova classe anomenada `ProducteFactura`, que contindrà els atributs `descripcio`, `preuUnitat` (preu unitari) i `quantitat`, com es pot observar en la figura 1.47.

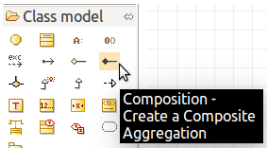
Figura 1.47. Classe ProducteFactura





Icona Aggregation

En aquest cas es tracta d'una composició, ja que és una relació més forta que l'agregació. `FacturaProductes` té sentit per ell mateix, però està compost de `ProducteFactura` (amb la descripció, preu i quantitat de la factura), aquesta relació és més forta que l'associació d'una agregació.

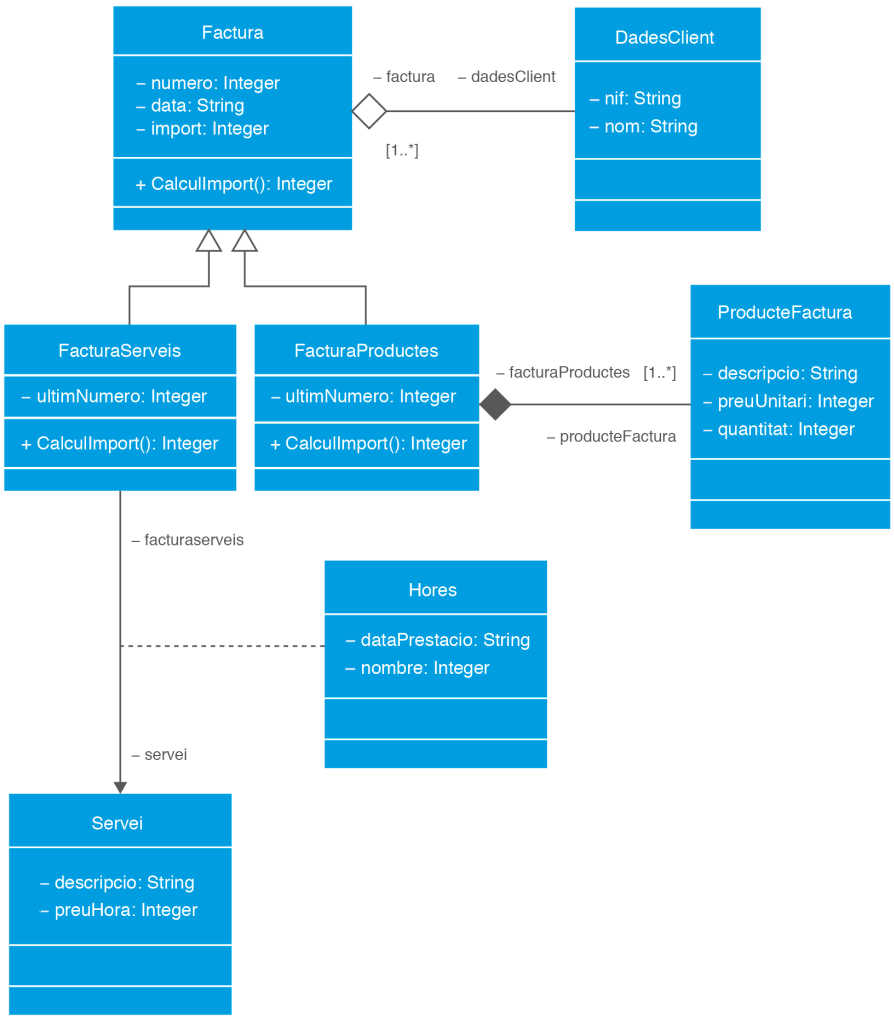


Icona Composition

Es crea igual que l'agregació, amb l'única diferència que, en lloc d'utilitzar la icona *Aggregation*, cal utilitzar la icona *Composition*.

El diagrama resultant de l'enunciat és el que es mostra en la figura 1.48.

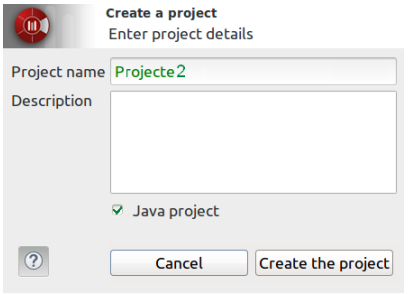
Figura 1.48. Diagrama de classes



1.3.9 Generació de codi a partir d'un diagrama de classes

Si volem poder generar codi a partir del diagrama de classes, en primer lloc, quan creem el projecte cal seleccionar l'opció *Java Project*, com es veu a la figura 1.49.

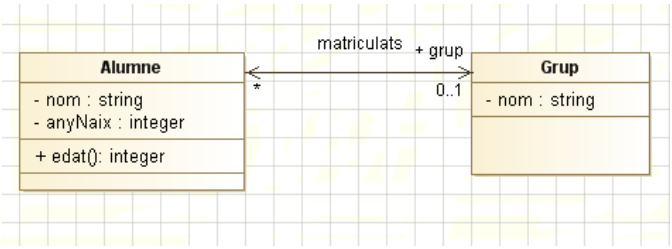
FIGURA 1.49. Creació d'un projecte que permet generar codi en Java



Es farà servir el format d'exemple per oferir les indicacions, pas a pas, de com generar codi a partir d'un diagrama de classes.

El diagrama de classes de partida serà el mostrat a la figura 1.50.

FIGURA 1.50. Diagrama de classes



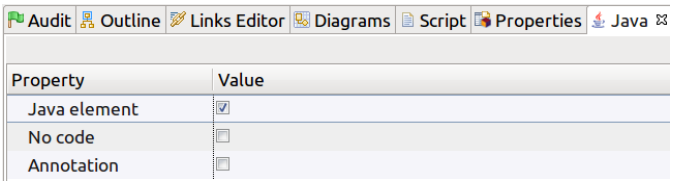
Atenció a les relacions

En fer aquest exemple, cal parar atenció a les cardinalitats. Cal fixar-se també que s'ha assenyalat l'associació com a navegable pels dos costats perquè es generin els membres que permetin aquesta doble navegació.

Un cop creat el diagrama, cal fer que tots els elements que apareixen es considerin com a *elements Java*. Es fa de la següent manera:

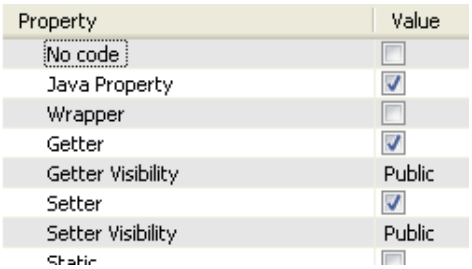
- **Classes**: cal seleccionar la classe i, a la pestanya *Java* de la part inferior, activar la casella *Java element*, com es veu a la figura 1.51.

FIGURA 1.51. Configuració d'una classe com a element Java



- **Propietats**: també cal seleccionar-les i configurar les caselles *Java Property*, *Getter*, *Setter*, *Getter visibility* i *Setter visibilty* de manera que quedi com a la figura 1.52.

Figura 1.52. Configuració d'una propietat com a element Java

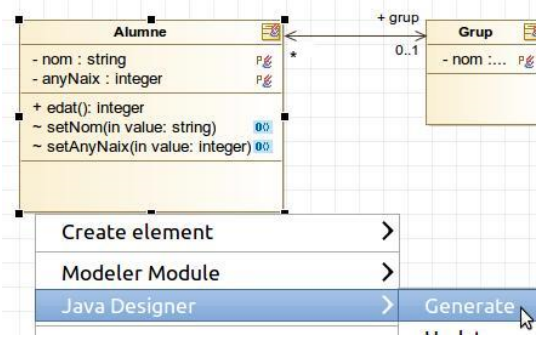


Després de definir les classes i les propietats com a *elements Java*, ho reflectiran, respectivament, les icones següents a la dreta del seu nom:

- **Associacions**: cal assenyalar-les com a *Java Property* dins de les seves propietats, de manera similar a com es fa amb els altres elements.

Per a **generar el codi** cal anar a **cada classe** i seleccionar, des del menú contextual (que apareix amb el botó secundari), l'opció *Java Designer* / *Generate*, com es veu a la figura figura 1.53.

Figura 1.53. Configuració d'una propietat com a element Java



Es pot veure el resultat tot seleccionant, al mateix menú contextual, *Java Desginer* / *Edit*. S'obrirà una pestanya amb el codi corresponent a la classe des de la qual hem triat l'opció.

A l'exemple, el resultat ha estat aquest el que es mostra a la figura 1.54.

Es pot veure que, tot i que el resultat és raonablement bo, cal fer algun retoc a mà a la classe `Grup`; concretament, l' `ArrayList` de la classe `Grup` no té nom, és públic i li falta el *getter*. A més, com és lògic, a la classe `Alumne` cal completar el mètode edat perquè calculi aquesta a partir d' `AnyNaix`.

Cal notar, també, que Modelio utilitza anotacions pròpies per incloure informació que necessita per gestionar internament els diferents elements.

FIGURA 1.54. Codi generat a partir del diagrama

```
import com.modeliosoft.modelio.javadesigner.annotations.mdl;
import com.modeliosoft.modelio.javadesigner.annotations.objid;

@objid ("ea148390-1b93-4ee6-b5ea-96e2e70982d2")
public class Alumne {
   @mdl.prop
   @objid ("b619ad4f-e045-4d54-90ec-a1e25a4a1662")
   private String nom;

   @mdl.propgetter
   public String getNom() {
      // Automatically generated method. Please do not modify this code.
      return this.nom;
   }

   @mdl.propsetter
   public void setNom(String value) {
      // Automatically generated method. Please do not modify this code.
      this.nom = value;
   }

   @mdl.prop
   @objid ("5f7344ef-9cbf-4474-a520-9c2760267bc5")
   private int anyNaix;

   @mdl.propgetter
   public int getAnyNaix() {
      // Automatically generated method. Please do not modify this code.
      return this.anyNaix;
   }

   @mdl.propsetter
   public void setAnyNaix(int value) {
      // Automatically generated method. Please do not modify this code.
      this.anyNaix = value;
   }

   @mdl.prop
   @objid ("13c5bbad-eb82-437f-8376-83a2fca94135")
   private Grup grup;

   @mdl.propgetter
   public Grup getGrup() {
      // Automatically generated method. Please do not modify this code.
      return this.grup;
   }

   @mdl.propsetter
   public void setGrup(Grup value) {
      // Automatically generated method. Please do not modify this code.
      this.grup = value;
   }

   @objid ("28f1391c-5152-4b82-890c-beb8c32fcae3")
   public int edat() {
   }

}
```

```
import java.util.ArrayList;
import java.util.List;
import com.modeliosoft.modelio.javadesigner.annotations.mdl;
import com.modeliosoft.modelio.javadesigner.annotations.objid;

@objid ("24a5fbd8-dccd-40e2-9826-8209eb4a06e6")
public class Grup {
   @mdl.prop
   @objid ("338ab93b-e6f7-400e-bf3a-f8196f24e84a")
   private String nom;

   @mdl.propgetter
   public String getNom() {
      // Automatically generated method. Please do not modify this code.
      return this.nom;
   }

   @mdl.propsetter
   public void setNom(String value) {
      // Automatically generated method. Please do not modify this code.
      this.nom = value;
   }

   @objid ("75030486-5d6f-4c37-b3ec-5cb131c6d884")
   public List<Alumne> = new ArrayList<Alumne> ();

}
```

1.3.10 Inserció de classes en un diagrama a partir del codi

En l'apartat *Generació de codi a partir d'un diagrama de classes* podeu trobar com crear un projecte definit com a *Java Project*.

Aquest apartat també seguirà el format d'exemple. En concret, afegirem dues classes a un projecte creat com a *Java Project*.

En concret, les classes que afegirem són les següents:

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Botiga {
5
```

```
  private String nom;

  private String adreca;

  private List<ClientHabitual>
 clients = new ArrayList<ClientHabitual>();

  public String getNom(){
    return nom;
  }

  public void setNom(String nom){
    this.nom=nom;
  }

  public String getAdreca(){
    return adreca;
  }

  public void setAdreca(String
 adreca){
    this.adreca=adreca;
  }

  public List<ClientHabitual>
 getClients(){
    return this.clients;
  }
}
```

```
public class ClientHabitual {

  private String nom;

  private String adreca;

  private Botiga botiga;

  public String getNom(){
    return nom;
  }

  public void setNom(String nom){
    this.nom=nom;
  }

  public String getAdreca(){
    return adreca;
  }

  public void setAdreca(String
 adreca){
    this.adreca=adreca;
  }

  public Botiga getBotiga(){
    return botiga;
  }

  public void setBotiga(Botiga
botiga){
    this.botiga=botiga;
  }

}
```

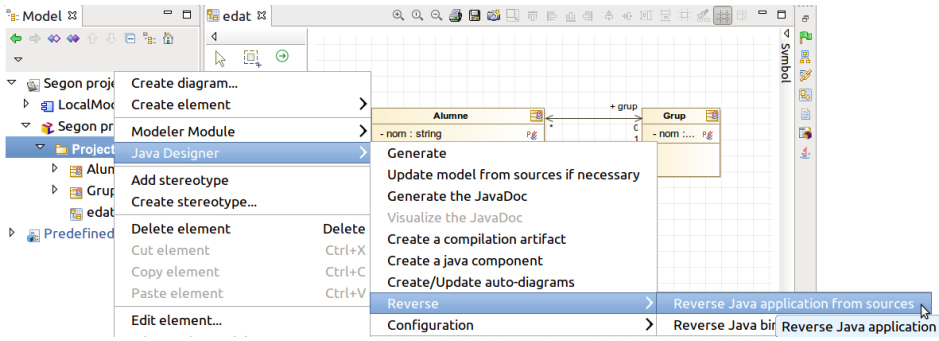
Per fer-ho, en primer lloc cal crear a la subcarpeta *src* del projecte un fitxer *.java* per a cadascuna de les classes, amb el contingut corresponent. La carpeta arrel del

projecte es troba seleccionant el projecte a l'explorador i clicant la icona *Project configuration*.

Icona Project configuration

A continuació, cal seleccionar l'opció *Java Designer* / *Reverse* / *Reverse Java application from sources* com es mostra a la figura 1.55.

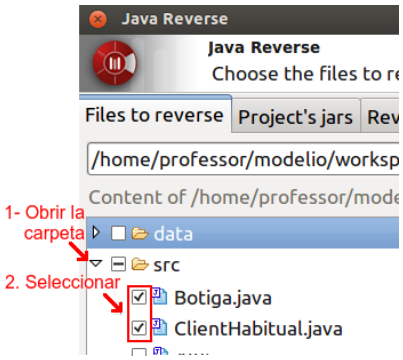
Figura 1.55. Opció per generar classes UML a partir de codi font



També poden importar-se fitxers *.class* o *.jar*.

A la pantalla que ens apareix, cal seleccionar els fitxers que contenen les classes que volem afegir al nostre projecte, com es mostra a la figura 1.56.

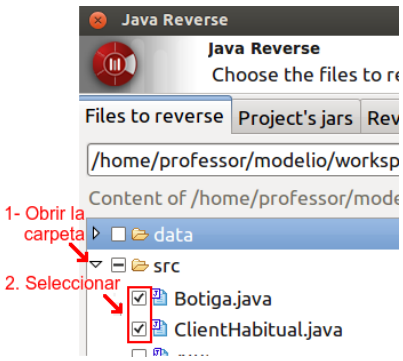
Figura 1.56. Selecció dels fitxers de les classes a incorporar al diagrama



Després, cal seguir l'assistent clicant al botó *Next* un parell de vegades, tot deixant les opcions que surten per defecte.

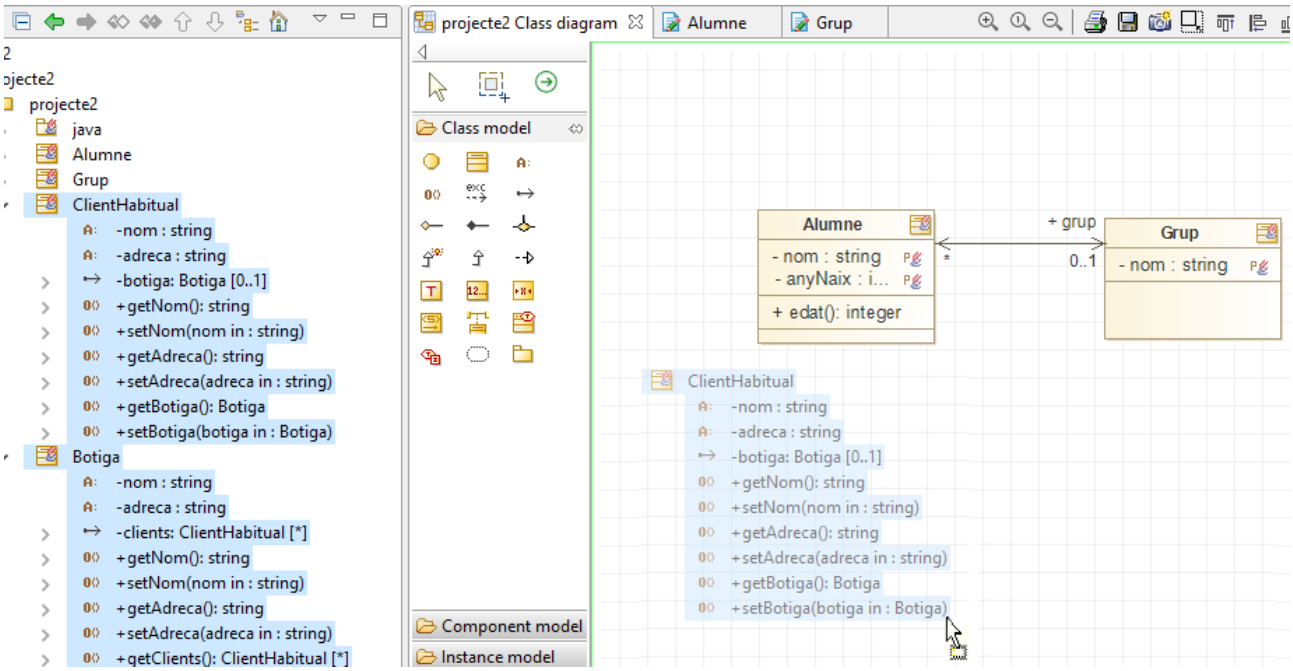
Per últim, clicant al botó *Reverse* ens sortirà una pantalla similar a la de la figura 1.57, que indica que tot ha anat bé:

Figura 1.57. Selecció dels fitxers de les classes a incorporar al diagrama



Es pot comprovar també que a la pestanya *Model* (a la part esquerra de la pantalla) es veuen les noves classes afegides al projecte, a més de les que ja tenia anteriorment. Ara només resta incorporar-les al diagrama de classes. Es pot fer desplegant-les, seleccionant-les totes (amb els seus membres inclosos) i arrossegant-les cap al diagrama. A la figura 1.58 es veu un resum del procés.

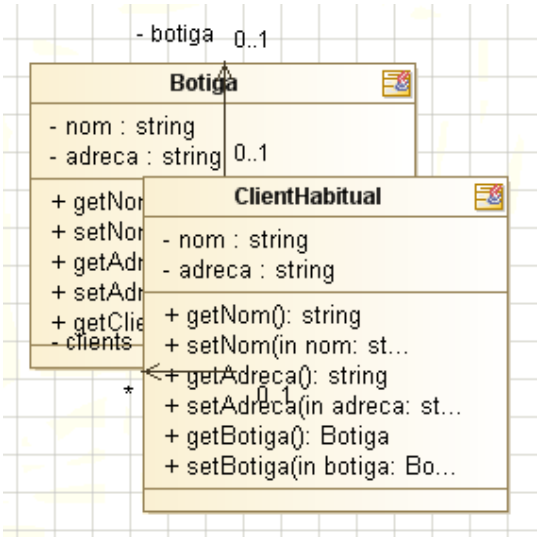
FIGURA 1.58. Incorporació de les classes al diagrama



Per seleccionar totes les classes que volem incorporar, n'hi ha prou amb fer clic a la primera, desplegar-les totes i, mantenint la tecla de majúscules premuda, fer clic al darrer element que volem incorporar.

Les noves classes presenten un aspecte similar al que es mostra a la figura 1.59.

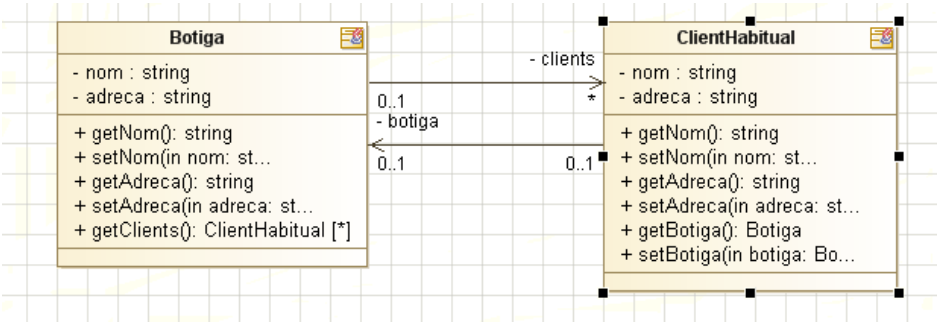
FIGURA 1.59. Classes incorporades al diagrama



Com es pot veure, les classes surten amuntegades. Podem moure-les i obtenir un resultat similar al mostrat a la figura 1.60.

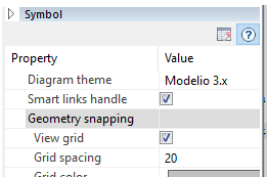
En aquest mateix apartat podeu trobar com fer les operacions bàsiques (moure, redimensionar...) amb els elements dels diagrames.

FIGURA 1.60. Resultat d'endreçar les classes.



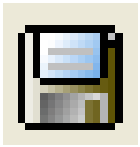
1.3.11 Operacions d'edició bàsiques

El diagrama de classes és el principal dels diagrames estàtiques. Modelio suporta també altres diagrames tant estàtics com dinàmics. Aquests poden tenir propòsits bastant diferents. No obstant, hi ha tota una sèrie d'operacions d'edició que són comunes a tots ells:



Propietats de la columna Symbol

- En primer lloc, a tots els diagrames és recomanable deshabilitar la característica *Smart links handle*, ja que no la utilitzarem i, a més, ens pot dificultar la resta de les accions. Es fa de la següent manera:
  - Fer clic a una zona del diagrama on no hi hagi cap element.
  - Fer clic a la columna *Symbol*, que es troba a la part dreta de la finestra, perquè es desplegui.
  - Deshabilitar la propietat *Smart links handle* i, si volem, tornar a fer clic a la barra superior de la columna *Symbol* perquè es torni a plegar.
- **Seleccionar un element**: n'hi ha prou amb fer-hi clic a sobre o, si volem seleccionar-ne més d'un a la vegada, podem:
  - Clicar a sobre d'ells, un rera l'altre, i mantenir a la vegada la tecla *Ctrl* polsada.
  - Assenyalar amb el ratolí una àrea que els inclogui, tot arrossegant el ratolí (de manera similar a com es seleccionen les icones d'una àrea a l'escriptori).
- **Esborrar un element**: cal seleccionar-lo i, a continuació, fer clic a la tecla *Supr*.


Icona Save diagrama in a file

- **Gravar el diagrama com un gràfic**: cal fer clic a la icona *Save diagram in a file*, que es troba a la barra d'eines de sobre del diagrama (al costat de la icona que representa una impressora i que permet imprimir el diagrama), i respondre al diàleg del sistema per desar fitxers.

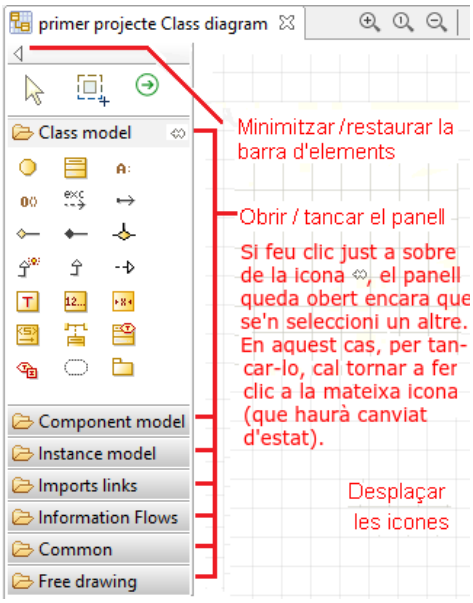

Icona Copy diagrama as grafic

- **Copiar el diagrama al porta-retalls com un gràfic**: cal fer clic a la icona *Copy diagrama as graphic*, que es troba a la barra d'eines de sobre del

diagrama i la imatge gràfica es copia automàticament al porta-retalls; a continuació pot enganxar-se en qualsevol programa d'edició.

- **Moure un element**: cal posar el ratolí a sobre de l'element i arrossegar-lo fins a portar-lo a la posició desitjada.
- **Canviar la forma d'un element**: cal seleccionar l'element, posar el ratolí a sobre d'un dels quadres negres que apareixen després d'haver fet la selecció i arrosegar-lo fins aconseguir la forma desitjada.
- **Connectar dos elements amb un enllaç**: normalment només cal seleccionar la icona corresponent a l'enllaç a la barra d'elements i, després, fer clic als elements que volem unir en l'ordre adient; quan s'està fent aquest segon pas, l'element sobre el que tenim el ratolí apareix de color verd si podem fer-hi clic i de color vermell en cas contrari.
- **Gestió de la barra d'elements**: podeu veure les operacions bàsiques a la figura 1.61. A tots els diagrames el funcionament és el mateix.

Figura 1.61. Elements per a gestionar la barra dels elements d'un diagrama



2. Diagrames dinàmics

El llenguatge unificat de modelització serveix per a la creació, especificació, construcció i documentació de models que representen tot tipus de sistemes. Aquesta modelització ha d'aportar una representació completa del sistema i es fa en termes d'orientació a objectes. Alguns dels sistemes que es poden representar mitjançant UML poden ser sistemes d'informació, sistemes de negocis, sistemes transaccionals (de gestió d'informació), sistemes distribuïts, sistemes estratègics, sistemes en temps real...

Alguns dels aspectes positius de fer servir la modelització de sistemes orientada a objectes són:

- Permet representar de forma més fidedigna el món real.
- Aquests són més fàcils d'entendre i de fer-hi ampliacions o modificacions.
- Tots els implicats parlen el mateix idioma (analistes, dissenyadors, desenvolupadors, verificadors...)
- Aporta més estabilitat pel fet que es poden fer petites ampliacions o modificacions sense haver de canviar tota la modelització.
- Permet la possibilitat de reutilitzar codi de forma senzilla.

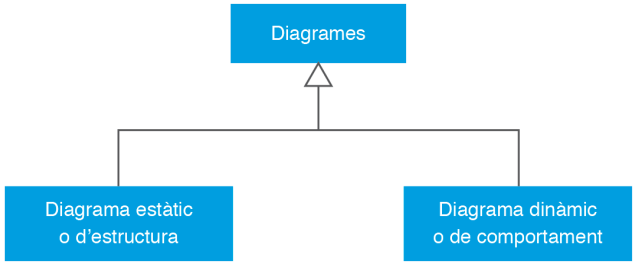
El llenguatge UML, en la seva versió 2.0, està compost de 13 diagrames (14 diagrames a partir de la versió 2.2), que es poden classificar seguint diversos criteris. Un d'ells pot ser en funció de les perspectives concretes o vistes des d'on es representen els sistemes. Per exemple, el diagrama de casos d'ús, que es considera un diagrama de la vista de requeriments o la vista de processos, engloba els diagrames de comportament.

Com es pot observar a la figura 2.1, una altra forma de classificació, que es fa servir en aquests materials, és la d'agrupar els diagrames en:

- Diagrames estàtics o diagrames d'estructura.
- Diagrames dinàmics o diagrames de comportament.

En la versió UML 2.2, i posteriors, es poden trobar set diagrames que pertanyen a cada una de les agrupacions de diagrames.

FIGURA 2.1. Classificació dels diagrames UML



2.1 Diagrames de comportament

Els diagrames considerats dinàmics o de comportament són els que representen el comportament dinàmic del sistema que s'està modelant. És a dir, indiquen les accions i processos que es duran a terme entre els elements del sistema, fixant-se en els seus moviments i en els efectes que tenen aquestes accions i activitats sobre els elements.

Els diferents diagrames de comportament descriuen el comportament de classificadors o, més sovint, de les seves instàncies, des de diferents punts de vista:

- Per una banda, representen o bé components executants o bé comportaments emergents.
- Per una altra, es destaca un d'aquests aspectes o un altre: interaccions amb l'exterior, o les situacions –anomenades estats– per les quals passa una instància durant la seva existència, o la seqüència temporal de les diferents parts d'un comportament...

Alguns d'aquests diagrames dinàmics es poden tornar a agrupar en funció de les seves funcionalitats. Concretament, dels set diagrames dinàmics n'hi ha tres que no s'agrupen, que tenen una entitat pròpia, i quatre més que es troben agrupats.

Els tres diagrames que no estaran agrupats són:

En l'apartat *Diagrama de casos d'ús* d'aquesta unitat es tracta amb detall el diagrama de casos d'ús.

- **Diagrama de casos d'ús**. Aquest diagrama identifica els diferents comportaments d'un sistema des del punt de vista de les seves interaccions amb el món exterior i descriu determinades relacions entre aquests comportaments.
- **Diagrama d'activitats** és el que descompon un comportament en activitats i representa els fluxos d'execució i d'informació entre aquestes activitats.
- **Diagrama d'estats** (en anglès State Machine Diagram). Aquest diagrama mostra els possibles canvis d'una situació a una altra de les instàncies del classificador de context i indica les causes i els comportaments que engeguen aquests canvis.

Els altres quatre diagrames estan agrupats en els anomenats diagrames d'interacció. Aquesta agrupació conté diagrames que representen un comportament emergent per mitjà de missatges entre les instàncies de classificadors d'una estructura interna o col·laboració i pot especificar restriccions temporals relatives a aquests missatges.

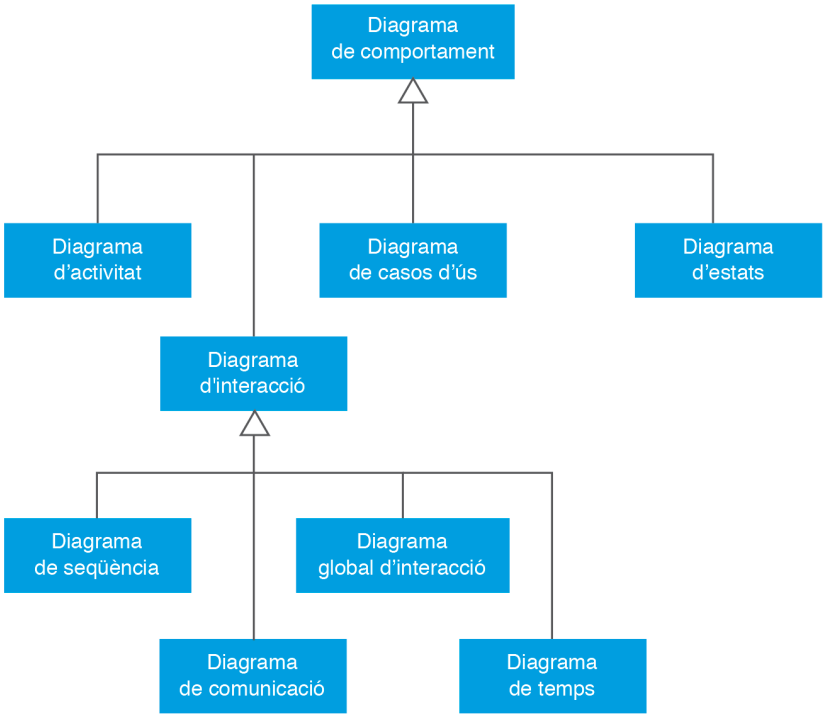
A l'apartat *2.3 Diagrames d'interacció* es tractaran amb detall els diagrames de seqüència i de col·laboració.

Els diagrames d'interacció són:

- **Diagrama de comunicacions**, que representa els missatges damunt els connectors d'una estructura interna o col·laboració
- **Diagrama de seqüència**, que posa èmfasi en l'ordre temporal dels missatges.
- **Diagrama de temps**, que representa una possible seqüència temporal de canvis d'estat d'una instància o de diverses instàncies que interactuen d'acord amb els diagrames d'estats respectius.
- **Diagrama general d'interacció**, que és un diagrama resumit que combina notacions dels diagrames de seqüències i dels d'activitats.

A la figura 2.2 es poden observar els diagrames de comportament pertanyents a l'UML Dinàmic.

FIGURA 2.2. Diagrama de comportament



2.1.1 Conceptes

Igual que en els diagrames d'estructura, cadascun dels diagrames de comportament de l'UML fa servir alguns tipus d'elements propis, a més d'altres de compartits per diversos diagrames de comportament. Els tipus d'elements compartits principals són:

- Senyal, que és una instància que es transmet d'un objecte a un altre.
- Missatge, que és una comunicació entre instàncies.
- Esdeveniment, que és un succés que pot tenir efectes damunt algun comportament.
- Activitat i acció.

Un **senyal** és una instància que un objecte o1 envia a un altre o2 i, com a conseqüència d'aquest enviament, s'executa un comportament asíncron que té o2 com a objecte de context.

Una **recepció de senyal** és una operació d'estereotip *signal* que indica que les instàncies del classificador dins el qual és definida (generalment, una classe o una interfície) reaccionen al senyal executant un comportament que és asíncron i, per tant, una recepció de senyal no pot tornar cap valor.

Un **missatge** és una comunicació entre instàncies de classificador, per la qual una (l'emissor) envia un senyal a l'altra (el destinatari) o li demana l'execució d'una operació.

Un **missatge síncron** és aquell que, quan s'emet, l'execució del comportament que l'ha emès roman aturada fins que rep un **missatge de resposta** del receptor; en canvi, quan s'emet un **missatge asíncron**, l'operació que l'ha emès es continua executant i no hi ha missatge de resposta. Un missatge asíncron pot consistir en l'enviament d'un senyal o la crida d'una operació; un de síncron només pot consistir en una crida d'una operació.

Un **esdeveniment** és un succés que es pot produir dins el sistema o el seu entorn, i quan té lloc pot provocar l'execució d'un comportament.

El fet que es produeixi un cert esdeveniment en un instant concret es diu **ocurrència d'esdeveniment**, i un conjunt de possibles ocurrències d'esdeveniment que tindrien el mateix significat i els mateixos efectes és un **tipus d'esdeveniment**. Un **disparador** és un element que especifica que una ocurrència d'esdeveniment del tipus indicat pot engegar un cert comportament. I inversament, tant l'engegada com la fi d'un comportament generen esdeveniments que poden engegar altres comportaments per mitjà dels disparadors corresponents.

Una **activitat** és una forma del comportament que es caracteritza per ser jeràrquica, en el sentit que pot ser constituïda per altres activitats; una activitat que no es pot descompondre és una **acció**.

2.1.2 Diagrama d'activitats

El **diagrama d'activitats** descriu les activitats que s'han de dur a terme en un cas d'ús, així com la manera de relacionar-se les activitats entre si per tal d'aconseguir un determinat objectiu. En altres paraules, el diagrama d'activitats descriu com un sistema implementa la seva funcionalitat.

Els diagrames d'activitats hereten conceptes dels diagrames de flux, essent un diagrama bastant fàcil d'interpretar.

Aquest diagrama està estretament vinculat a altres diagrames, com són el diagrama de classes (diagrama estàtic), el diagrama d'estats (diagrama dinàmic) i el diagrama de casos d'ús (diagrama dinàmic). Un diagrama d'activitats explica què està succeint entre diversos objectes o quin és el comportament d'un objecte.

Els diagrames d'activitats fan servir una sèrie d'elements, com ara:

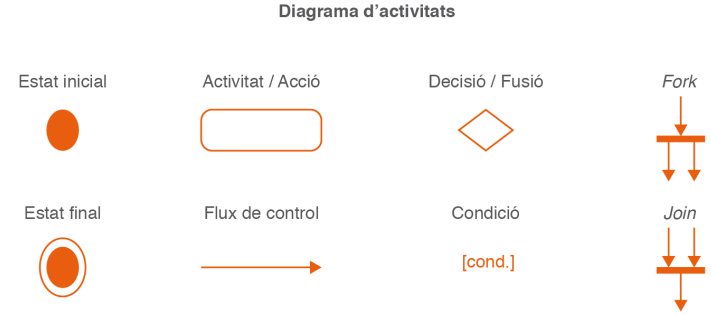
Diagrames previs

Els classificadors que s'esmentin en un diagrama d'activitats s'han d'haver descrit prèviament en un diagrama de classes, i si les instàncies d'algun d'ells tenen diferents estats possibles cal haver-ne fet el diagrama d'estats corresponent.

- **Estats inicials**, representats mitjançant un cercle omplert de color negre. Marquen l'inici de l'execució dels processos o activitats.
- **Estats finals**, representats mitjançant un cercle omplert de color negre amb una altra circumferència per sobre amb una petita distància sense omplir. Els estats finals indiquen el final de l'execució d'un procés o activitat.
- **Activitats o accions**, representades mitjançant un rectangle de cantonades arrodonides. Indiquen l'arribada a un node una vegada efectuada una acció o activitat.
- **Transicions o fluxos de control**, representats mitjançant fletxes. La seva direcció indica el node des del qual s'inicia l'activitat o l'acció fins a l'altre node, al qual s'arribarà una vegada finalitzada.

A la figura 2.3 es poden observar totes les representacions gràfiques dels elements que participen en els diagrames d'activitats.

Figura 2.3. Elements del diagrama d'activitats

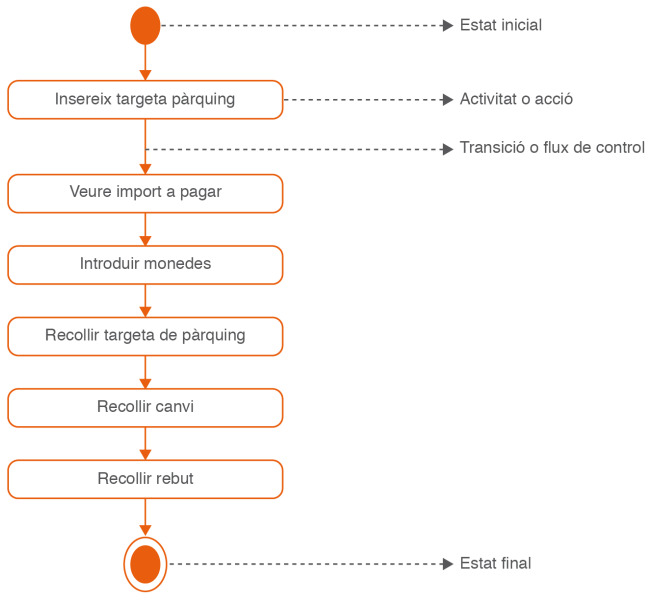


A la figura 2.4 es mostra un exemple de diagrames d'activitats. Es vol modelitzar el procediment de pagament per part d'un usuari de l'estada del seu cotxe a un pàrquing. Abans d'enretirar el cotxe del pàrquing, haurà d'abonar l'import que se li calcularà automàticament, en funció del temps que hagi estat estacionat el seu cotxe. Es mostra un diagrama d'activitats molt senzill en la figura 2.4, on totes les activitats es mostren a partir de transicions o fluxos de control seqüencials. A partir de l'estat inicial, i fins arribar a l'estat final, es passarà per sis activitats o accions:

- Inserir targeta pàrquing.
- Veure import a pagar.
- Introduir monedes.
- Recollir targeta pàrquing.
- Recollir canvi.
- Recollir rebut.

Es pot veure com les activitats defineixen, de forma molt clara, quina funcionalitat executaran.

FIGURA 2.4. Exemple de diagrama d'estats - activitats seqüencials



Altres elements importants en els diagrames d'activitats són els que fan referència a la possibilitat de donar alternatives als fluxos de control i a les activitats o accions. Alguns d'aquests elements són:

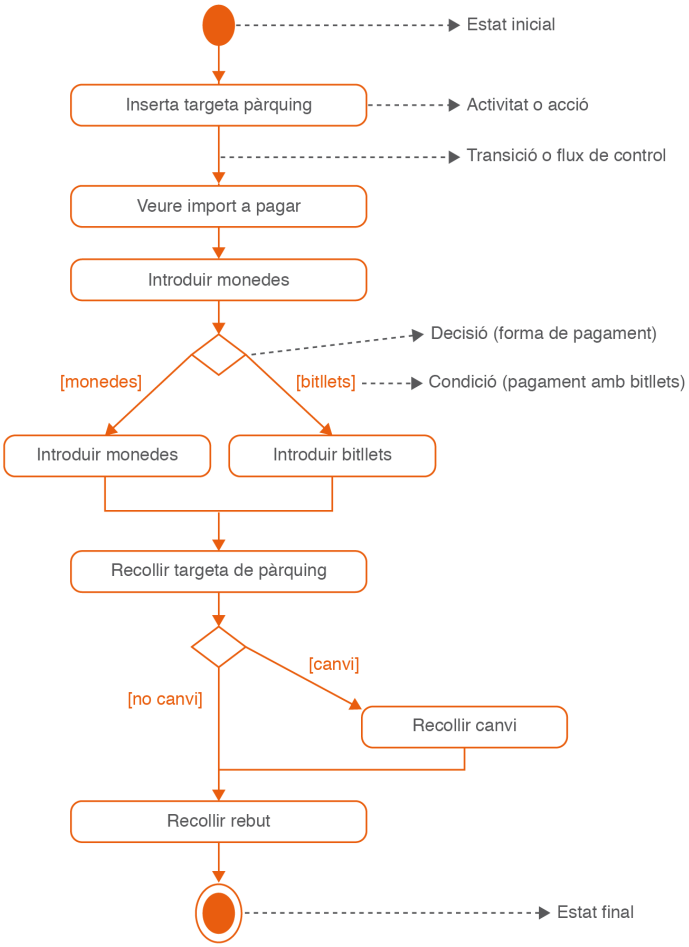
- **Decisió/fusió**, representat mitjançant un rombe regular. A partir d'un flux d'entrada, hi haurà dos o més fluxos de sortida, en funció de la condició marcada. Cada flux de sortida haurà d'estar indicat per una condició. En

el cas de la fusió, serveix per agrupar dos o més fluxos que arriben a un mateix punt de diverses decisions donades anteriorment a aquest punt en el diagrama d'activitats. La fusió tindrà dos o més fluxos d'entrada, però un únic flux de sortida.

- **Condició**, representat per un text que s'escriu entre els símbols `[ ... ]`. Aquesta condició representa la pregunta o preguntes que es farà el flux de control per, una vegada presa la decisió, saber per quin camí caldrà que continuï.

A la figura 2.5 es pot observar un exemple de condicions i bifurcacions a partir d'una ampliació de l'exemple anterior, el mostrat a la figura 2.4.

Figura 2.5. Exemple de diagrama d'activitats - alternativa



Què succeeix si l'usuari vol pagar la tarifa del seu pàrquing amb bitllets en comptes de fer-ho amb monedes? La modelització del sistema haurà de tenir en compte aquestes dues opcions. Què succeeix si l'usuari ha introduït l'import exacte a pagar i no ha de recollir canvi? Potser no és necessari passar per l'activitat Recollir canvi.

En aquests dos exemples es poden veure els elements de condició i de decisió/fusió.

Uns altres elements també importants en els diagrames d'activitats són els anomenats *fork* i *join*. *Fork* significa 'bifurcació' i *join* 'agrupació o unió'. Són dos elements estretament vinculats amb les bifurcacions, podent ser complementaris o substitutius d'aquestes. Les seves característiques són les següents:

- ***Fork***, representat per una línia negra sòlida, perpendicular a les línies de transició. Un *fork* representa una necessitat de ramificar una transició en més d'una possibilitat. La diferència amb una ramificació és que en el *fork* és obligatori passar per les diferents transicions, mentre que en la decisió es pot passar per una transició o per una altra. En altres paraules, les transicions de sortida d'un *fork* representen transicions que podran ser executades de forma concurrent.

- ***Join***, representat per una línia negra sòlida, perpendicular a les línies de transició. Un *join* fusiona dues o més transicions provinents d'un *fork*, és a dir, se sincronitzen en una única línia de flux. Totes les accions de les línies de flux prèvies al *join* han de completar-se abans que s'executi la primera acció de la línia posterior al *join*.

A la figura 2.6 es pot veure una altra evolució de l'exemple del pagament del pàrquing, afegint-hi aquesta vegada els elements *join* i *fork*. Concretament, si es parteix de la figura 2.5, es pot observar que a l'activitat *Recollir targeta del pàrquing* s'ha afegit un element de tipus *fork* que especifica que a partir d'aquella activitat s'haurà de passar per dues activitats, la de recollir el canvi, en el cas de ser necessari, i la de recollir el rebut. L'ordre en què es desenvoluparan serà indiferent.

Una vegada acabades aquestes dues activitats, s'ha afegit un element de tipus *join*, que recull la finalització de les dues activitats per, una vegada acabades, passar a la següent activitat, en aquest cas l'estat final.

Cal tenir en compte alguns condicionants que s'hauran d'acomplir a fi de crear un diagrama d'activitats correcte:

- No és obligatori que hi hagi un estat final. Per exemple, un procés que es desenvolupi de forma contínua mai acabarà.

- Podrà haver-hi diversos estats finals.

- Les activitats es podran descompondre en altres activitats.

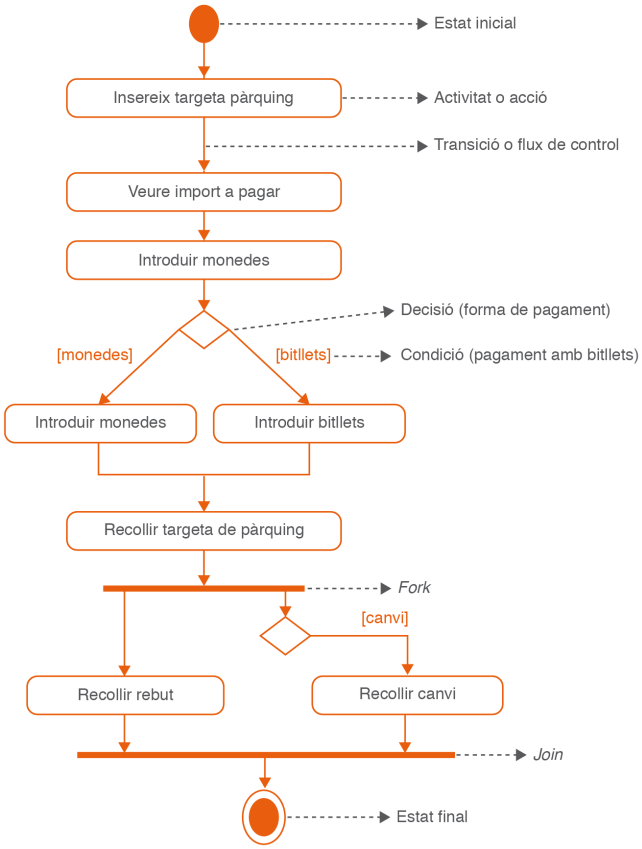
- Una acció és una activitat que no es pot descompondre en altres activitats.

- Un flux de control no podrà finalitzar mai en l'estat inicial.

- Cada activitat o acció ha de tenir, com a mínim, un flux de control d'entrada i un flux de control de sortida.

- Les condicions dels fluxos de sortida d'una mateixa decisió hauran de ser complertes i disjuntes.

FIGURA 2.6. Exemple diagrama de classes - 'fork' i 'join'



2.1.3 Diagrama d'estat

El **diagrama d'estat** representa el conjunt d'estats pels quals passa un objecte durant la seva vida en una aplicació en resposta a esdeveniments, juntament amb les seves respostes i accions. Un esdeveniment podria ser un missatge rebut, un error, un temps d'espera superior al planificat...

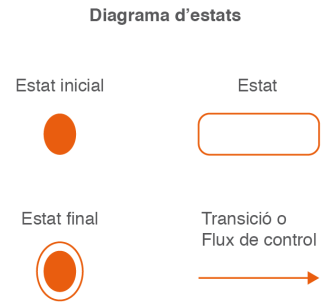
En la major part de les tècniques Orientades a Objectes, els diagrames d'estats es dibuixen per a una sola classe, mostrant el comportament d'un sol objecte durant tot el seu cicle de vida.

Com es pot veure a la figura 2.7, els elements d'un diagrama d'estats són molt similars al dels diagrama d'activitats. Resumint, podem trobar:

- **Estats inicials**, representats mitjançant un cercle omplert de color negre. Marquen l'inici del diagrama d'estats.
- **Estats finals**, representats mitjançant un cercle omplert de color negre amb una altra circumferència per sobre, amb una petita distància sense omplir. Els estats finals indiquen el final del diagrama d'estats. En un diagrama d'estats és possible que no hi hagi estats finals.

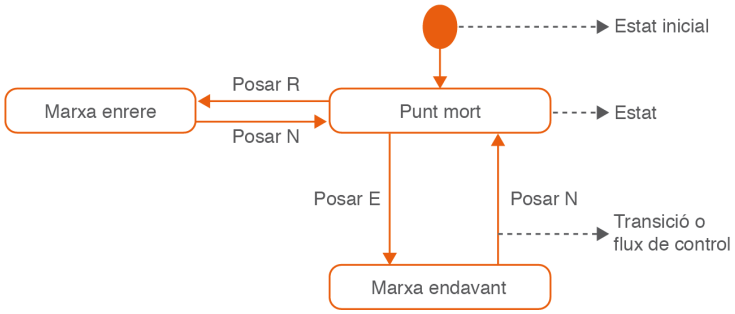
- **Estats**, representats mitjançant un rectangle amb les cantonades arrodonides. Un estat identifica una condició o una situació en la vida d'un objecte, durant la qual satisfà alguna condició, executa alguna activitat o espera que passi algun esdeveniment.
- **Transicions o fluxos de control**, representats amb una fletxa amb direcció que tindrà superposat el nom de l'esdeveniment que provocarà aquesta transacció entre estats. Tindrà sempre un inici i un final. Indica el pas d'un objecte d'un estat a un altre estat.

FIGURA 2.7. Elements del diagrama d'estats



A la figura 2.8 es mostra un petit exemple d'un diagrama d'estats. Es vol modelitzar, mitjançant un diagrama d'estats, el funcionament de les marxes d'un cotxe automàtic. Aquest tipus de cotxe podrà trobar-se en dos possibles estats: o està funcionant marxa endavant o està funcionat marxa enrere.

FIGURA 2.8. Diagrama d'estats - canvi de marxes d'un cotxe automàtic



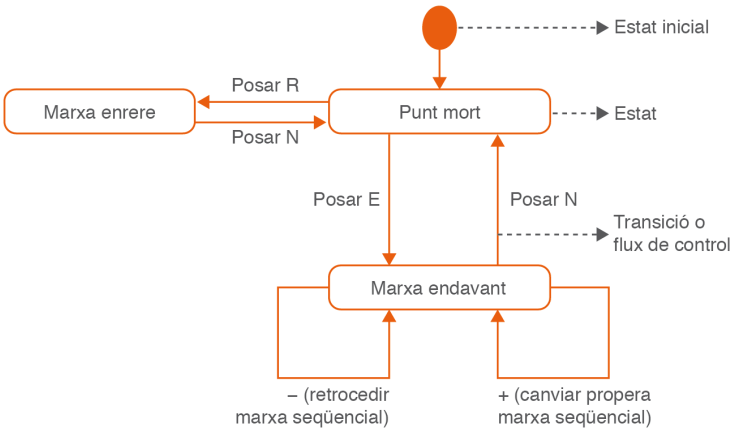
En l'exemple de la figura 2.8 es parteix d'un estat inicial i s'arriba a un estat anomenat *Punt Mort*. En aquest estat el cotxe no es mourà. A partir d'aquí es podrà posar la marxa enrere (mitjançant el flux de control anomenat *Posar R*) i es podrà tornar a l'estat *Punt Mort* amb el flux de control: *Posar N*. El mateix succeirà amb l'estat *Marxa endavant*.

Les transaccions es podran crear unint un mateix estat, com es pot veure a la figura 2.9.

En els cotxes automàtics el conductor pot deixar l'estat *Marxa endavant* activat.

En aquesta situació el cotxe va canviant automàticament de marxes sense l'acció del conductor. Però també s'accepta el poder augmentar o disminuir una marxa de forma seqüencial per part del conductor. En l'exemple es veu com l'estat *Marxa endavant*, a més de poder tornar a través de la transacció *Posar N* a l'estat *Punt Mort*, podrà evolucionar a partir de dues transaccions més (retrocedir marxa o canviar a la propera marxa) al mateix estat, amb la qual cosa modificarà les seves propietats.

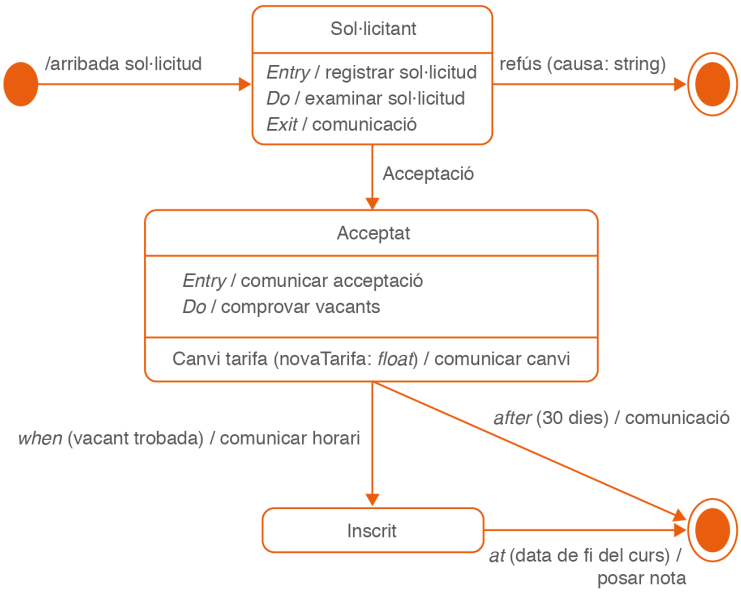
Figura 2.9. Diagrama d'estats - canvi de marxes d'un cotxe automàtic



Arribats a aquest punt, es proposarà un diagrama d'estats una mica més complicat, en el qual es mostra el comportament de l'estat davant un esdeveniment i les accions que efectua l'objecte.

La figura 2.10 és un diagrama d'estats que descriu la gestió de les inscripcions dels alumnes d'una acadèmia.

Figura 2.10. Diagrama d'estats - procés de matriculació



El diagrama comença amb l'estat inicial, representat amb la rodoneta; quan un alumne envia una sol·licitud d'inscripció, es produeix l'esdeveniment de senyal *arribada sol·licitud*, que provoca una transició cap a l'estat *Sol·licitant* i, com a conseqüència de l'arribada a aquest estat, s'executa l'activitat d'entrada *registrar sol·licitud* i, a continuació, l'activitat durant l'estat *examinar sol·licitud*, l'execució de la qual dura fins que es produeix algun dels esdeveniments que fan sortir d'aquest estat: si es produeix l'esdeveniment de senyal *refús*, que té com a atribut la seva *causa*, s'acaba tot el procés pel que fa a aquesta sol·licitud d'inscripció, mentre que si es produeix l'esdeveniment de senyal *acceptació*, l'estat de destinació és *Acceptat*; en tots dos casos, s'executa l'activitat de sortida *comunicació*.

Un cop a l'estat *Acceptat*, s'executa l'activitat d'entrada *comunicar acceptació* i, acte seguit, l'activitat a l'estat *comprovar vacants*, que s'interromp quan es produeix un dels dos esdeveniments següents:

- O bé l'esdeveniment de temps -que han passat 30 dies des de l'arribada a aquest estat-, i aleshores s'executa l'activitat *comunicació* i l'estat de destinació de la transició és l'estat final, que es representa per una rodona buida i una de plena concèntriques.
- O bé l'esdeveniment de canvi, que consisteix que passi a complir-se la condició *vacant trobada*.

En qualsevol cas, si abans de sortir de l'estat *Acceptat* es produeix l'esdeveniment de senyal *canvi tarifa*, que té com a atribut la *nova tarifa*, s'executa l'activitat *comunicar canvi*, d'acord amb la transició interna especificada.

Quan se surt de l'estat *Acceptat*, com a conseqüència d'aquest esdeveniment de canvi s'executa l'activitat *comunicar horari*.

Si un alumne és en l'estat *Inscrit* quan es produeix l'esdeveniment de temps d'arribada de la *data de la fi del curs*, es produeix una transició cap a l'estat final, alhora que s'executa l'activitat *posar nota*.

2.2 Diagrama de casos d'ús

Aquest diagrama és un dels més representatius i més utilitzats de l'anàlisi i disseny orientat a objectes. Mitjançant els diagrames de casos d'ús s'aconsegueix mostrar el comportament del sistema i com aquest es relacionarà amb el seu entorn.

El **diagrama de casos d'ús** identifica els comportaments executants d'un classificador generalment complex (per exemple, un programari) i especifica amb quins usuaris i altres entitats exteriors tenen interacció (en el sentit que en reben informació o els en donen o són engegats per aquestes entitats), i també certes relacions entre aquests comportaments.

El punt de vista que es fa servir per elaborar els diagrames de casos d'ús és el punt de vista de l'usuari final. Això implica que es tracta d'un diagrama molt convenient i utilitzat en els intercanvis inicials d'opinions amb els usuaris finals del sistema a modelitzar i, posteriorment, a automatitzar. Els mateixos usuaris el poden entendre i participar en la seva correcció. Amb aquest tipus de diagrama es podrà crear una documentació adequada per a les necessitats de la presa de requeriments.

Els diagrames de casos d'ús representen els diferents requeriments que fan servir els usuaris finals d'un determinat sistema. A partir de la utilització d'actors i de casos d'ús, aquest tipus de diagrames modelen les diferents funcionalitats que podrà oferir un sistema.

Un **cas d'ús** és una funcionalitat o un servei que ofereix el sistema a modelitzar als seus usuaris finals. És un conjunt d'interaccions seqüenciades que es desenvolupen entre els actors i el sistema per donar resposta a un esdeveniment que inicia un actor denominat principal.

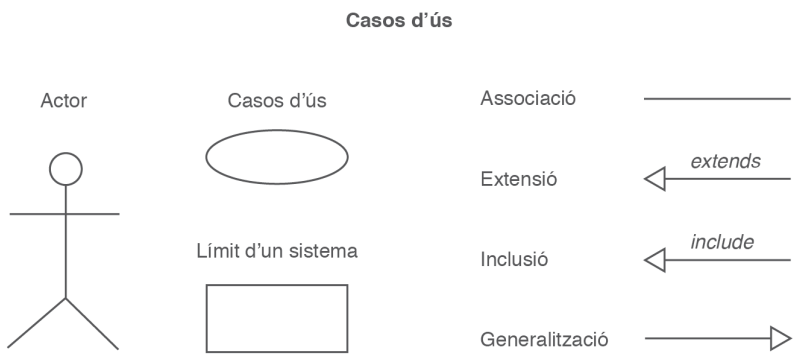
En la modelització d'un sistema podrà haver-hi molts casos d'ús. Això es deu al fet que cada cas d'ús haurà de donar resposta només a una característica concreta del sistema. Així, amb aquesta representació del sistema, l'usuari final podrà validar fàcilment que l'analista ha entès correctament el funcionament del sistema que ha de representar.

Els elements fonamentals del diagrama de casos d'ús són els següents:

- Escenari
- Actor
- Subjecte
- Cas d'ús o comportament
- Associació o relació

A la figura 2.11 es mostren les representacions gràfiques dels elements més importants d'un diagrama de casos d'ús.

FIGURA 2.11. Diagrama de casos d'ús



2.2.1 Escenari

Una entitat externa i els actors

A una sola entitat externa li poden correspondre diversos actors, si les interaccions que té amb el sistema de maquinari i programari són prou diverses per considerar que fa diferents grups de papers en relació amb ell.

Un escenari és un camí representat per un o més actors i un o més casos d'ús i les seves associacions. Es tracta d'un camí que podrà prendre un cas d'ús.

Un cas d'ús podrà tenir diversos escenaris possibles. Aquests escenaris podran representar casos d'èxit i casos on no s'acompleixen les expectatives.

2.2.2 Actor

Un actor és un conjunt de papers que fa una entitat física o virtual externa al subjecte en relació amb els seus casos d'ús; a partir d'aquests papers l'actor interactua amb el subjecte i cada paper té a veure amb un dels casos d'ús del subjecte. Tot allò que inicia un cas d'ús o respon a un cas d'ús també es considera un actor.

Els actors són classificadors instanciables. Un actor pot ser:

- Una persona (un usuari) que interactuï directament amb el subjecte (per tant, si un usuari interactua amb el subjecte per compte d'un altre, l'actor serà el primer usuari, no pas el segon).
- Un sistema informàtic extern en temps d'execució que rebi informació del subjecte o li'n doni.
- Un dispositiu físic que tingui un cert comportament propi i autònom en relació amb el subjecte i hi interactuï directament.
- *Temps*, *Rellotge*... quan un cas d'ús s'engega automàticament en una hora determinada.

Un actor també es defineix com un rol que farà un usuari en utilitzar un sistema. Amb el concepte de rol es vol determinar el fet que una mateixa persona física (un usuari final del sistema) podrà interpretar el paper d'actors diferents en diferents casos d'ús o, fins i tot, en un mateix cas d'ús. Un actor no serà una determinada persona, sinó que serà el paper que juga (la persona que sigui) quan utilitza el sistema.

2.2.3 Subjecte

El subjecte d'un diagrama de casos d'ús és el classificador al qual està associat el diagrama, que en representa els comportaments executants.

Pot ser qualsevol cosa que tingui comportaments: un programari, un sistema informàtic o físic en general, un subsistema, un component, una classe...

2.2.4 Cas d'ús

Un cas d'ús és un comportament executant del subjecte; és engegat d'una manera directa o indirecta per un actor i lliura uns resultats concrets a un actor o a diversos, o a un altre cas d'ús.

Un cas d'ús es representa mitjançant un verb que indica una acció, operació o tasca concreta. Aquesta operació s'activarà a partir d'una ordre donada per un agent extern al cas d'ús. Podrà ser un actor que faci una petició i generi el cas d'ús, o bé un altre cas d'ús que l'invoqui.

2.2.5 Una associació o una relació

Una associació o una relació és un vincle que es dóna entre un cas d'ús i un actor o entre dos casos d'ús.

Entre dos casos d'ús es podran establir diferents tipus de relacions:

- **Associació**: una associació és un camí de comunicació entre un actor i un cas d'ús. Aquesta comunicació implica que l'actor participa en el cas d'ús.
- **Generalització/especialització:** una generalització indica que un cas d'ús és una variant d'un altre, és a dir, podrà trobar-se en una forma especialitzada d'un altre cas d'ús existent. Tant la representació com el sentit poden semblar similars al concepte de subclasses en l'orientació a objecte. Aquest tipus de relació serveix per identificar comportaments semblants per part d'un o diversos casos d'ús. A partir d'una primera descripció més genèrica, l'especialització detalla els comportaments més específics a cada cas d'ús especialitzat. Això es deu al fet que el cas d'ús especialitzat pot variar qualsevol aspecte del cas d'ús base.

  La generalització és el mateix tipus de relació vist des del punt de vista de les subclasses que tenen elements comuns i que, a partir de les seves semblances, permeten crear una superclasse que contindrà aquests elements comuns.
- **Dependència d'inclusió**: el tipus de relació d'inclusió és un cas de dependència entre casos d'ús. Concretament, indica que un cas d'ús està inclòs en un altre. Això succeeix quan els casos d'ús comparteixen uns determinats elements. En aquest cas, el cas d'ús que està inclòs és el que tindrà tots els comportaments compartits. Aquest tipus de relació també es coneix com *use*, perquè hereta moltes de les característiques de l'antiga

relació *use* o utilitza. Es tracta d'una relació útil en casos en què es volen extraure comportaments comuns des de molts casos d'ús a una descripció individual. En aquest tipus de relació no hi ha paràmetres o valors de retorn. Quan un cas d'ús A és inclòs per un altre cas d'ús B, el cas d'ús que ha incorporat el comportament de l'altre, en aquest cas el cas d'ús B, haurà de ser utilitzat per si mateix. En cas contrari, es coneix com a cas d'ús abstracte.

- **Dependència d'extensió:** aquesta relació és un altre tipus de dependència. Ofereix un tipus d'extensió diferent a la relació de tipus generalització, d'una forma més controlada. Quan un cas d'ús s'estén a un altre, això significa que el primer pot incloure part del comportament de l'altre cas d'ús, aquell al qual s'està estenent. A més, podrà afegir-hi accions o comportaments. La forma de funcionar de la dependència serà la creació d'una sèrie de punts d'extensió per part del cas d'ús base. Si hi ha més d'un punt d'extensió, caldrà definir molt bé quin és el punt que s'ha estès. A partir d'aquests punts, el cas d'ús especialitzat només podrà modificar el comportament dels punts d'extensió que s'hagin creat. Aquest tipus de relació és una alternativa a l'ús de casos d'ús complexos. Amb les dependències d'extensió es podran controlar millor les diferents bifurcacions i els errors.

Un cas d'ús es podrà representar de dues formes diferents:

- Mitjançant un diagrama de casos d'UML, seguint la notació de la resta de diagrames UML.

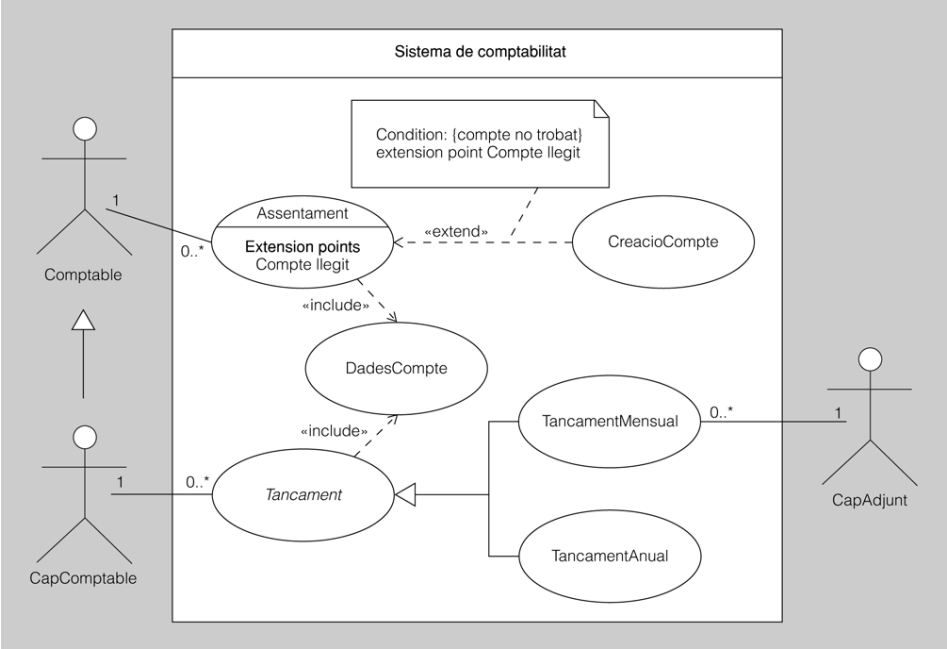
- Mitjançant un document detallat.

Diagrama de casos d'ús en UML

Cadascun dels elements que componen un diagrama de casos d'ús té una notació gràfica que permet la representació de cada cas d'ús de forma esquemàtica i fàcil d'entendre i d'interpretar per als usuaris finals.

A la figura 2.12 veiem el diagrama de casos d'ús d'una suposada aplicació de comptabilitat, representada pel subjecte anomenat *Sistema de comptabilitat*.

FIGURA 2.12. Cas d'ús - sistema de comptabilitat



L'actor *CapComptable* és una especialització de *Comptable*; *Comptable* és l'actor primari del cas d'ús *Assentament*, ja que no té cap més actor, i *CapComptable* és actor primari del cas d'ús *Tancament* i també, per herència entre actors, d'*Assentament*. El cas d'ús *CreacioCompte* estén el cas d'ús *Assentament* perquè quan s'intenta fer un assentament amb un compte inexistent cal crear aquest compte; s'ha indicat el punt d'extensió, *Compte llegit*, dins del cas d'ús estès, assentament, i la condició corresponent, *compte no trobat*. *CreacioCompte* no és executable independentment perquè no té cap actor ni, doncs, actor primari.

El cas d'ús *DadesCompte* consisteix en l'accés a les dades d'un compte, accés que és una part comuna als casos d'ús *Assentament* i *Tancament*, d'aquí la dependència include. *Tancament* és abstracte i *Tancament-Mensual* i *TancamentAnual* en són especialitzacions i, en conseqüència, n'hereten l'actor *CapComptable* i no cal associar-los-el explícitament. *TancamentMensual* té, a més de l'actor heretat, l'actor *CapAdjunt* (que pel diagrama no sé sap si també és primari o no). Les multiplicitats de les associacions són innecessàries per trivials: en cada execució del cas d'ús intervé una sola instància de l'actor i cada instància de l'actor pot participar en qualsevol nombre d'execucions del cas d'ús.

Document detallat de casos d'ús

La segona forma de representar els casos d'ús és presentar un document detallat. Aquest document podrà ser un substitutiu dels diagrames gràfics de casos d'ús, però també podrà ser un complement a aquells diagrames.

En el document detallat es mostrarà tot tipus d'informació referent al cas d'ús, aquesta vegada, però explicada amb text. Tant les comunicacions i les interaccions dels elements com els escenaris i actors involucrats queden descrits en aquest document detallat.

Es presenta en forma de taula completa amb dues columnes i tantes files com informacions es vulguin mostrar. Algunes d'aquestes informacions poden ser:

- **Cas d'ús**, que indicarà el nom del cas d'ús; serà com el títol del document detallat.
- **Actors**, que descriurà tots els actors, tant primaris com secundaris, que prendran part en el diagrama de casos d'ús.
- **Tipus**, que indica el tipus de cas d'ús que s'està detallant. Pot ser un tipus de flux bàsic o bé basat en algun altre cas d'ús (a partir d'una inclusió, una extensió o una generalització).
- **Propòsit**: caldrà indicar-hi l'objectiu, la raó de ser, del cas d'ús.
- **Resum del cas d'ús**: es durà a terme una descripció resumida del cas d'ús.
- **Seqüència normal o flux principal**: s'hi indicarà, seqüencialment, cada pas que es durà a terme juntament amb la seva acció corresponent i la seva descripció detallada. En funció de les accions dels actors, s'escollirà un subflux o un altre per a la continuació del cas d'ús.
- **Subfluxos**: són els considerats fluxos secundaris en les accions seqüencials d'un cas d'ús.
- **Precondicions**: igual que en programació, les precondicions estableixen els requisits que haurà de complir el cas d'ús per poder-se executar.
- **Excepcions**: són les situacions especials que hi pot haver durant l'execució d'un cas d'ús.
- **Urgència**: indicarà la celeritat amb què el cas d'ús s'haurà de desenvolupar i d'executar.
- **Freqüència**: indicarà el nombre de vegades que està prevista l'execució d'aquest cas d'ús.
- **Rendiment**: indicarà en quant de temps s'haurà d'executar el cas d'ús com les accions seqüencials.
- **Comentaris**: espai on es podran recollir altres consideracions referents al cas d'ús que es considerin oportunes.

A la figura 2.13 es pot veure com quedaria un document detallat d'un cas d'ús amb alguns dels camps citats anteriorment.

Figura 2.13. Document detallat d'un cas d'ús

Identificador	Codi identificatiu del cas d'ús		
Nom	Nom descriptiu		
Descripció	Breu descripció de l'objectiu del cas d'ús		
Actors	Identificació dels actors que intervenen en el cas d'ús		
Precondicions	Condicions que s'han de produir abans d'accedir al cas d'ús		
Seqüència normal	Pas	Acció	
	1	Descripció acció 1 (seqüencial)	
	2	Descripció acció 2 (seqüencial)	
	3	Descripció acció 3 (alternativa)	
		3a	Si condició acció 3a
		3b	Si condició acció 3b
	4	Descripció acció 4	
	...	...	
Seqüència alternativa o excepcions	Pas	Acció	
	e1	En el cas que es produeixi l'excepció1, s'haurà d'efectuar l'acció e2	
	e2	En el cas que es produeixi l'excepció2, s'haurà d'efectuar l'acció e3	
	...	...	
Postcondicions	Condicions que ha de complir una vegada executat el cas d'ús		
Notes	Comentaris		

Avantatges/inconvenients dels diagrames de casos d'ús

Els diagrames de casos d'ús ofereixen alguns punts forts que els converteixen en els més populars dels diagrames d'UML:

- Són molt adequats per comunicar-se amb els usuaris finals i, en definitiva, amb els clients.
- Ajuden a documentar les funcionalitats del que es coneix com a capses negres, que és com es consideren alguns dels casos d'ús d'un sistema.
- Ajuden a dividir i gestionar els projectes d'una mida molt extensa.
- Complementen la documentació del projecte informàtic, oferint una explicació senzilla del funcionament als usuaris.
- Permeten fer un seguiment objectiu del projecte.
- Ajuden en la tasca de verificació i validació que duen a terme els verificadors del programari desenvolupat en el projecte.

2.3 Diagrames d'interacció

Els anomenats diagrames d'interacció agrupen un conjunt dels diagrames classificats dintre els diagrames de comportament. De fet, els diagrames d'interacció es consideren un subtipus dels diagrames de comportament.

Cal recordar que els diagrames de comportament indicaran, dintre de la simulació o modelització del sistema representat, què haurà de succeir entre els elements definits als diagrames estàtics o d'estructura.

En els diagrames d'interacció es donarà més èmfasi a les relacions entre els elements dels sistema, concretament en el que té a veure amb el flux de dades i amb el flux de control.

Una **interacció** és un comportament emergent descrit en termes d'intercanvis de missatges entre els elements connectables d'una estructura composta.

Els diagrames d'interacció representen un comportament emergent per mitjà de missatges entre les instàncies de classificadors d'una estructura interna o col·laboració. Aquests diagrames poden especificar restriccions temporals relatives a aquests missatges.

Els diagrames d'interacció són:

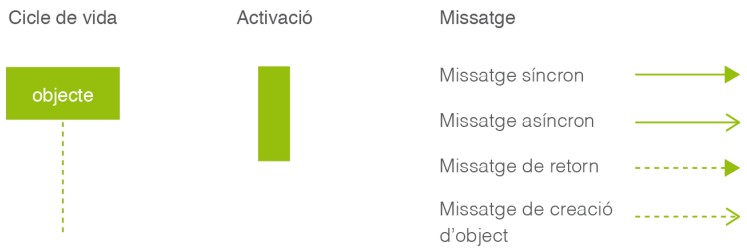
- **Els diagrames de seqüència**. Posen l'èmfasi en l'ordre temporal dels missatges entre els diferents elements de la modelització del sistema. Aquests missatges es coneixen com a missatges d'intercanvi entre objectes, el que realment són crides a mètodes de les classes. Aquests intercanvis es donen en un escenari concret en un temps delimitat.
- **Els diagrames de comunicació** (diagrames de col·laboració). Representen els missatges damunt els connectors d'una estructura interna o de col·laboració. Es tracta d'un tipus de diagrama molt similar als diagrames de seqüència pel que es refereix als missatges que s'intercanviaran els diferents objectes. La diferència rau en el fet que en els diagrames de seqüència l'important és l'ordre temporal d'aquests missatges, mentre que en els diagrames de comunicació es fa més èmfasi en la relació entre els objectes i el tipus de relació que hi ha.
- **Els diagrames de temps**. Representen els canvis d'estat que tindrà un objecte al llarg del temps. Aquests canvis d'estat dels objectes es produiran a partir d'esdeveniments que es produeixin al llarg del temps. Aquests diagrames estan estretament lligats als diagrames d'estats i als diagrames de seqüència.
- **Els diagrames de visió general de la interacció**. Defineixen les interaccions a partir d'una variant dels diagrames d'activitat. Ofereixen una visió general dels fluxos de control utilitzant interaccions en lloc d'activitats.

2.3.1 Diagrama de seqüència

El **diagrama de seqüència** descriu les interaccions entre un grup d'objectes mostrant de forma seqüencial les trameses de missatges entre objectes.

Els objectes interactuen entre si amb l'enviament de missatges. La recepció d'un missatge sol significar l'execució d'un mètode que s'especifica en el missatge, i es torna actiu l'objecte mentre dura l'execució del mètode.

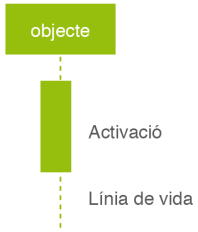
Figura 2.14. Elements del diagrama d'activitats



Els elements són els següents:

- Una **línia de vida** representa un element connectable que participa en una interacció enviant i rebent missatges (figura 2.15). Una línia de vida representa l'interval de temps en què existeix la instància o instàncies que formen part de l'element connectable, des de la seva creació fins a la seva destrucció, tot i que en general només durant una part o diverses parts d'aquest interval participen en la interacció que conté la línia de vida; aquestes parts s'anomenen **activacions** i representen els intervals de temps durant els quals s'està executant alguna operació que té com a objecte de context alguna de les seves instàncies.

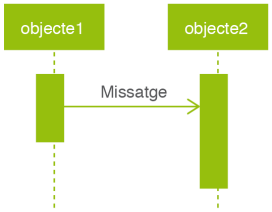
Figura 2.15. Línia de vida



- Un **missatge** és l'especificació de la comunicació entre objectes (figura 2.16). Un missatge es representa amb una fletxa de l'emissor cap al receptor; diferents tipus de missatge es representen amb diferents tipus de fletxa:
  - Un **missatge asíncron** s'indica amb una fletxa de línia contínua i punta oberta; és quan l'objecte no espera la resposta a aquest missatge abans de continuar.
  - Un **missatge síncron** es representa amb una fletxa de línia contínua i punta plena; és quan l'objecte espera la resposta a aquest missatge abans de continuar amb el seu treball.
  - Un **missatge de resposta d'un missatge síncron** s'indica amb una fletxa de línia discontínua i punta plena.

– Un **missatge que provoca la creació d'un objecte** s'indica amb una fletxa de línia discontínua i punta oberta; a més, l'extrem del missatge coincideix amb el començament de la línia de vida del receptor.

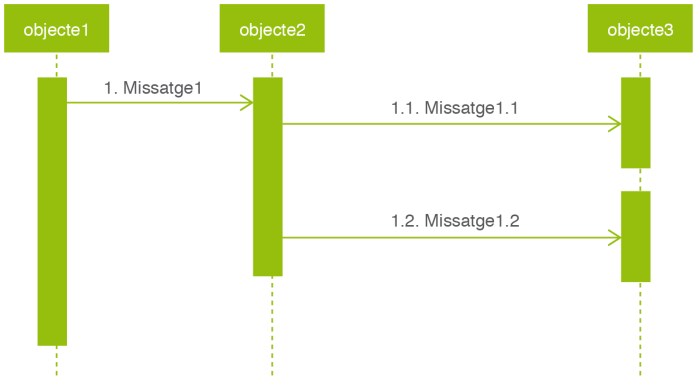
Figura 2.16. Missatge



Els missatges es poden numerar en format de llista multinivell. Si un missatge s'envia una vegada finalitzat el seu precedent es comptabilitza seqüencialment: 1,2,3... Però si el missatge s'envia abans que hagi finalitzat el precedent, s'incrementarà el nivell: 3.1, 3.2,...

A la figura 2.17 se'n pot veure un exemple representatiu.

Figura 2.17. Numeració dels missatges



A continuació, es mostra un exemple complet per al desenvolupament d'un diagrama de seqüència. Concretament, es proposa crear un diagrama de seqüència per a la modelització de l'elaboració d'una *vichyssoise*, sopa freda de puré de patata i porros. Aquesta recepta es durà a terme amb l'ajuda d'un robot de cuina.

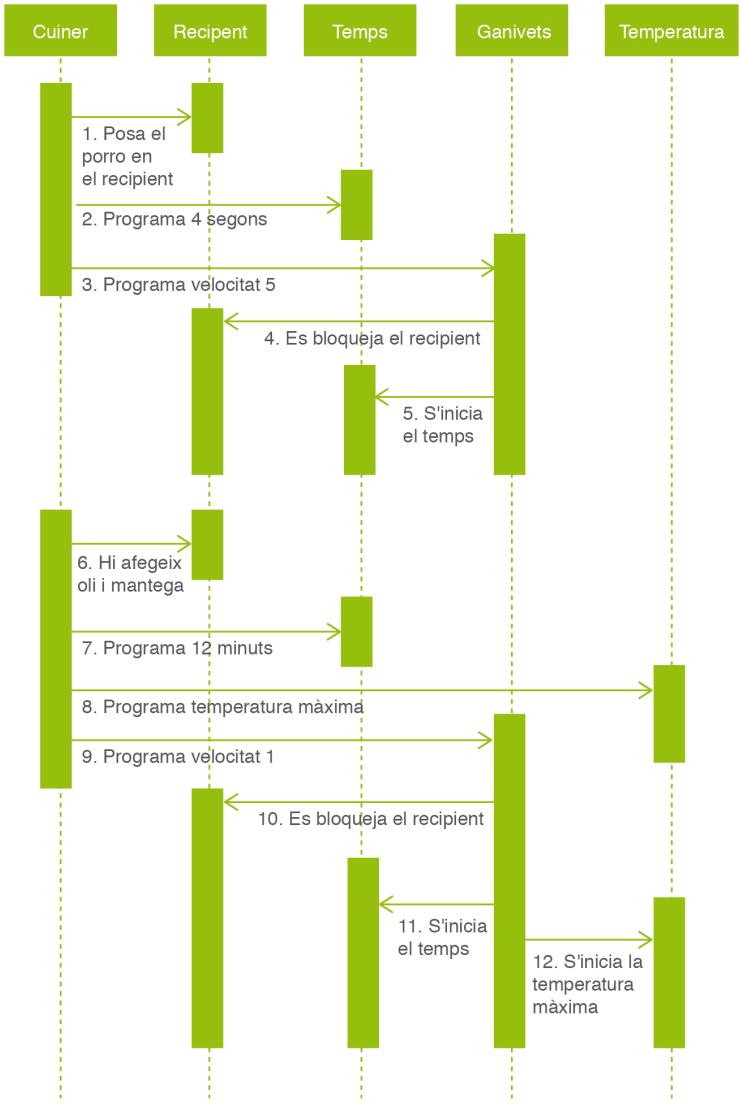
Inicialment, es mostren els passos que s'han de fer per a l'elaboració de la sopa:

El cuiner posa el porro en el recipient i programa 4 segons a velocitat 5. En especificar la velocitat dels ganivets, es bloqueja la tapa del recipient i el cronòmetre del robot comença a comptabilitzar el temps.

Un cop transcorregut el temps, es desbloqueja la tapa, i el cuiner hi pot col·locar l'oli i la mantega. Aquest programa velocitat 1 i 12 minuts a temperatura màxima. En especificar la velocitat dels ganivets, es bloqueja la tapa del recipient, el cronòmetre del robot comença a comptabilitzar el temps i s'activa la resistència

per proporcionar calor.

Figura 2.18. Exemple de diagrama de seqüència - robot de cuina



La recepta continuaria, però es demana el diagrama de seqüència fins a aquest punt.

A la figura 2.18 es pot veure un exemple de diagrama de seqüència que podria solucionar l'enunciat anterior.

2.3.2 Diagrama de comunicació

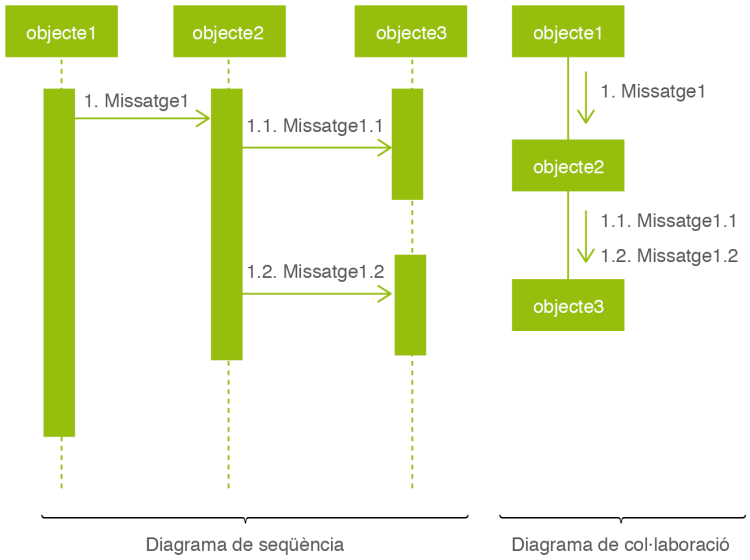
Aquest diagrama, en versions anteriors a la versió 2.0 de l'UML, es coneixia com a diagrama de col·laboració. Des de les darreres versions es coneix amb aquest nou nom: diagrama de comunicació.

La finalitat del **diagrama de comunicació** és representar la comunicació entre els elements del sistema que es vol modelitzar. Treballant en conjunt els elements podran acomplir els objectius del sistema.

El diagrama de comunicacions té l'aspecte general d'una col·laboració en la qual, al costat de cada connector, s'han afegit les fletxes corresponents als missatges que hi circulen durant la interacció. Aquestes fletxes són les mateixes que hi hauria en un diagrama de seqüències de la mateixa interacció pel que fa al tipus de línia i de punta, però són molt més curtes. La disposició vertical dels elements connectables és lliure i, per tant, no representa cap ordenació temporal; per mitjà de la numeració s'indica l'ordre d'emissió dels missatges i si aquesta emissió és seqüencial o en paral·lel.

A la figura 2.19 es mostra un diagrama de comunicació (antigament anomenat diagrama de col·laboració). També es mostra el diagrama de seqüència que hi està vinculat, per poder veure, d'aquesta manera, la seva integració.

FIGURA 2.19. Diagrama de seqüència vs. diagrama de col·laboració



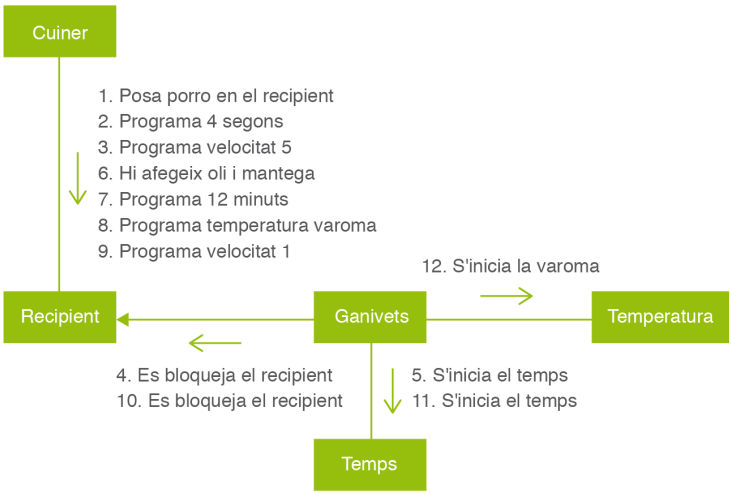
El diagrama de comunicacions, d'una banda, no permet representar els usos d'interacció i, de l'altra, només té representació per a alguns dels operadors d'interacció; en conseqüència, a la pràctica només és adequat per a la descripció d'interaccions relativament senzilles.

Fins a cert punt, un diagrama de comunicacions és semblant a un diagrama de seqüències vist des de dalt: de les línies de vida només se'n veuria la capçalera i, com que si hi hagués diversos missatges entre els mateixos dos elements connectables es veurien sobreposats. També hi ha autors que consideren que és un diagrama molt semblant al diagrama d'objectes, perquè mostra el context necessari per a una col·laboració entre objectes.

A la figura 2.20 es mostra el diagrama de comunicacions corresponent a l'exemple que s'ha exposat anteriorment: l'elaboració d'una sopa freda Vichyssoise. Es

recomana comparar el diagrama de seqüències anterior corresponent a aquest exemple, figura 2.18, amb el diagrama de comunicacions de la figura 2.20.

FIGURA 2.20. Diagrama de comunicacions - robot de cuina



2.3.3 Diagrama de temps

Els **diagrames de temps** (o diagrames temporals) representen una interacció entre diferents estats i en destaquen els aspectes temporals i, en especial, els canvis d'estat o de valor d'una línia de vida (és a dir, de les instàncies que conté) o de diverses línies al llarg del temps. Els diagrames de temps mostren els canvis de l'estat d'un objecte al llarg del temps com a conseqüència d'esdeveniments.

Generalment, en una màquina d'estats hi ha més d'un camí possible (és a dir, més d'una seqüència de transicions) entre l'estat inicial i l'estat final; un diagrama temporal correspon només a un d'aquests camins i, en conseqüència, en un diagrama temporal no hi pot haver parts opcionals o alternatives.

El diagrama de temps no té conceptes propis i les seves notacions pròpies són molt senzilles. En la seva forma més usual i detallada és un diagrama bidimensional, en el qual es representa una escala horitzontal de temps i els diferents estats o valors d'una línia de vida corresponent a diferents altures dins de la dimensió vertical; l'ordenació vertical dels estats no té cap significat. Un diagrama pot tenir diverses franges horitzontals que corresponen a diferents línies de vida que comparteixen una sola escala de temps.

La successió dels diferents estats d'una línia de vida al llarg del temps és una línia contínua feta de segments de recta, dels quals:

- Els segments horitzontals representen intervals de temps durant els quals la línia de vida roman en un cert estat.

- La resta de segments representa les transicions entre aquests estats.

Un **segment vertical** representa una transició que dura un temps negligible, mentre que un segment inclinat representa una transició que dura el temps indicat per la diferència de les abscisses dels seus extrems inicial i final.

Diagrames temporal i de seqüències

Atès que tots dos diagrames tenen una escala de temps i línies de vida, podríem dir que el diagrama temporal ve a ser un diagrama de seqüències sense fragments combinats en el qual damunt cada línia de vida es representen els canvis d'estat.

Tot seguit es mostra un exemple de diagrama de temps. En primer lloc, es mostren, a la figures figura 2.21 i figura 2.22, els diagrames d'estats de dues classes, la `Classe1` i la `Classe2`. La `Classe1` ofereix dos estats (`Estat A` i `Estat B`), mentre que la `Classe2` ofereix tres possibles estats (`Estat 1`, `Estat 2` i `Estat 3`).

Figura 2.21. Diagrama d'estats de "Classe1"

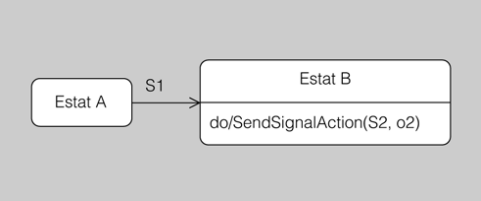
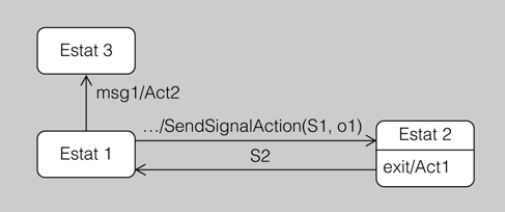


Figura 2.22. Diagrama d'estats de "Classe2"

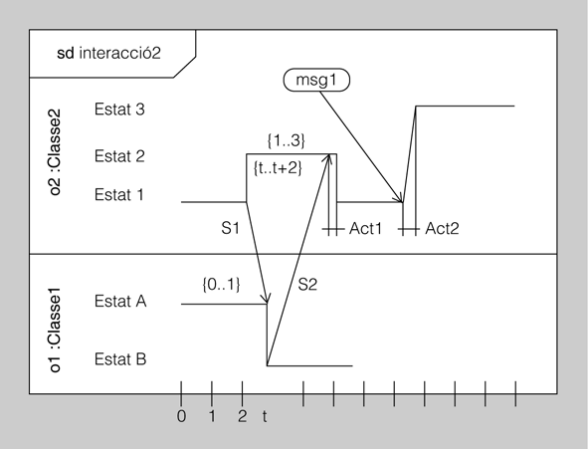


El diagrama de temps mostra una interacció entre l'objecte `o1` de la `Classe1` i l'objecte `o2` de la `Classe2`.

A la figura 2.23 es mostren les línies de vida de cadascun dels objectes, concretament dues franges temporals dins les quals es representen sengles línies de vida interrelacionades.

A la primera línia de vida hi ha dos estats i a la segona tres, l'ordre dels quals no té cap significat. Hi ha una escala de temps a la part de sota. Dels tres missatges que provoquen transicions el primer és conseqüència de la transició d'`Estat1` a `Estat2` (que provoca l'execució d'una activitat, l'acció estàndard `SendSignal Action(S1, o1)` la qual, d'acord amb els seus paràmetres, envia el senyal `S1` a l'objecte `o1`), el segon és conseqüència de l'entrada a `EstatB` i el tercer ve de l'exterior. El temps que passa entre l'arribada del segon missatge i la consegüent sortida d'`Estat2` és la durada de l'execució de l'activitat `act1`, i la durada de la transició d'`Estat1` a `Estat3` és la durada de l'execució de `act2`.

Figura 2.23. Diagrama temporal



Hi ha tres restriccions temporals:

- El missatge que transmet el senyal `S1` ha de durar com a màxim una unitat de temps.
- L'objecte `o1` ha de romandre a `Estat2` entre una i tres unitats de temps.
- El tercer missatge ha d'arribar no més tard de dues unitats de temps després de l'emissió del segon missatge.

La figura 2.24 és una versió més compacta del diagrama dels estats. En comptes de representar-hi els estats a diferents altures, se'n posen els noms damunt la línia de vida, dins dels intervals corresponents; aquest diagrama representa la primera línia de vida del de la figura 2.23.

Figura 2.24. Diagrama temporal simplificat

2.3.4 Diagrama de visió general de la interacció

El **diagrama de visió general de la interacció** aporta una visió global del flux de control de les interaccions.

Alhora de representar els models d'interacció d'un sistema pot sorgir el problema que siguin molt grans, amb la qual cosa és bastant immanejable el seu tractament. Amb els diagrames de visió general de la interacció, se segueix la filosofia "divideix i conqueriràs", ja que permeten modularitzar les interaccions, descomposant-

les en fragments més petits, i representar en un diagrama la relació entre els fragments.

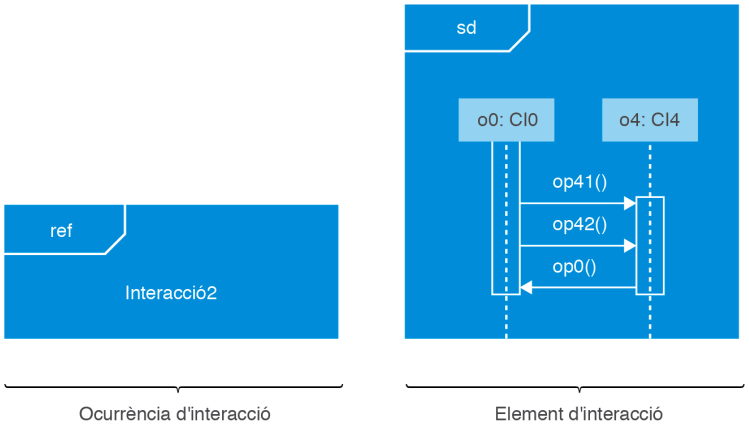
La representació del diagrama de visió general de la interacció fa ús de la nomenclatura del diagrama d'activitat incorporant dos conceptes nous:

- **Ocurrència d'interacció**: fa referència als diagrames d'interacció existents.
- **Element d'interacció**: es du a terme una representació de diagrames d'interacció existents dins un marc rectangular on se n'especifica el contingut.

En la secció *Annexos* del web del mòdul hi podeu trobar un exemple complert de diagrames de visió general de la interacció.

A la figura 2.25 es mostra la representació gràfica de cadascun dels dos conceptes.

FIGURA 2.25. Ocurrència d'interacció i element d'interacció



2.4 Modelio i UML: diagrames de comportament

UML és un llenguatge de modelatge de sistemes que es compon de molts diagrames. Cal comprendre quin és el significat de cadascun dels diagrames individualment, quins són els seus elements i com aquests es relacionen entre ells. Però per entendre d'una forma més global el que pot arribar a oferir UML cal observar un exemple complet, sobre el qual es desenvolupin tots els diagrames. Per fer aquest exemple global s'ha escollit un exemple clàssic, que es basa en el rentat de roba per part d'una rentadora.

Per a la creació d'aquest exemple de rentat de roba es fa servir l'entorn Modelio. Aquest entorn està especialitzat en el modelatge amb UML i altres llenguatges gràfics. També permet tant generar codi a partir dels diagrames com generar diagrames a partir de codi font. L'entorn Modelio ha estat implementat a partir de l'IDE Eclipse. La pàgina web de l'entorn Modelio és https://www.modelio.org/. Podeu descarregar-vos l'instal·lable des de l'apartat *Downloads*.

2.4.1 Diagrama de casos d'ús

A l'apartat "Diagrames estàtics" podeu trobar detallat el procediment per crear un projecte, un diagrama de classes i, també, com realitzar les operacions bàsiques amb **qualsevol tipus de diagrama**.

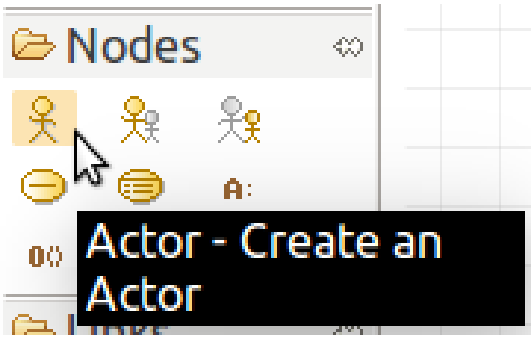
El cas d'ús descriu les accions d'un sistema des del punt de vista de l'usuari. En aquest cas, en Joan posa una rentadora amb l'objectiu que es renti la roba.

El diagrama de casos d'ús es crea de manera anàloga a com es creen els diagrames de classe, però seleccionant a l'últim pas *Use Case diagram* en lloc de *Class diagram*.

L'actor (en Joan) s'afegeix al diagrama:

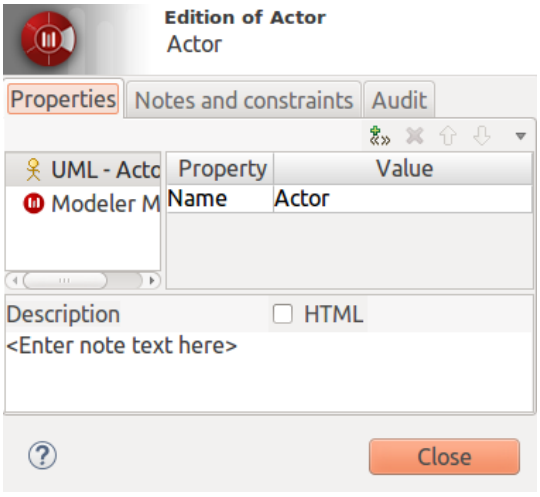
1. Fent clic a la icona *Actor*, a la secció *Nodes*, com es mostra a figura **??**.
2. Fent clic a continuació al punt del diagrama on volem inserir l'actor.

FIGURA 2.26. Icona //Actor//



Podem editar les seves propietats fent doble clic a sobre de la icona que representa l'actor. Ens apareixerà una finestra similar a la mostrada a la figura 2.27.

FIGURA 2.27. Propietats d'un //Actor//



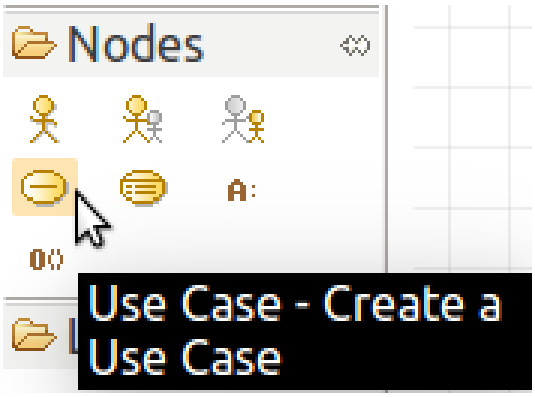
En aquest cas, només podem canviar el nom (*Name*). Hi posarem *Joan*.

Per especificar el cas d'ús *Rentar roba* cal:

1. Fer clic a la icona *Use Case*, a la secció *Nodes*, com es mostra a figura 2.28.

2. Fer clic a continuació al punt del diagrama on volem inserir el cas d'ús.

Figura 2.28. Icona //Use Case//



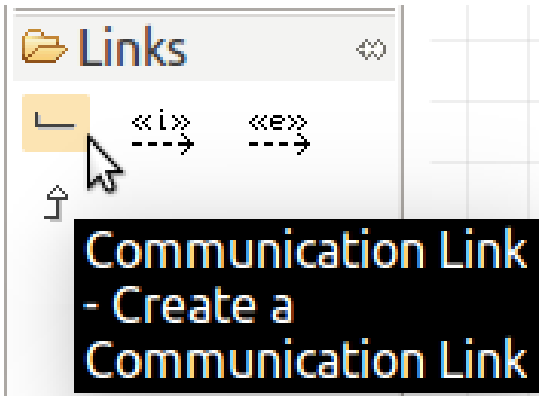
Igual que passava amb els actors, fent-hi doble clic ens apareixerà la finestra de les propietats on podem especificar el nom.

Ara cal enllaçar l'actor (en *Joan*) amb el cas d'ús (*Rentar roba*). Ho farem:

1. Fent clic a la *Communication Link*, a la secció *Links*, com es mostra a figura **??**.

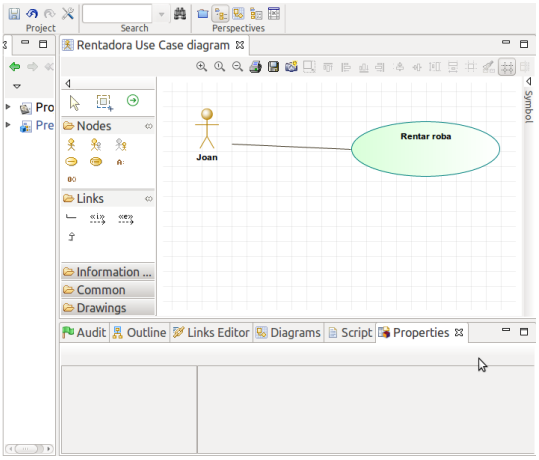
2. Fent clic a continuació a sobre de l'actor i, després, a sobre del cas d'ús.

Figura 2.29. Icona //Communication Link//



A la figura 2.30 es mostra el diagrama del cas d'ús resultant

FIGURA 2.30. Diagrama de cas d'ús: Rentar roba



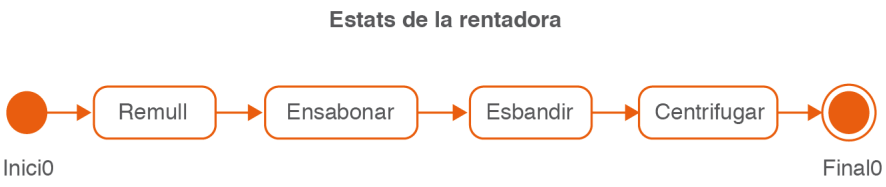
Normalment, cada cas d'ús va associat amb una explicació que sol contenir els següents apartats:

- **Nom**: `RentarRoba`.
- **Descripció**: Posar una rentadora amb l'objectiu que la roba bruta quedi neta.
- **Precondicions**: Roba bruta
- **Flux normal**: La rentadora neteja la roba.
- **Flux alternatiu**: No.
- **Postcondicions**: Roba neta.

2.4.2 Diagrama de transició d'estat

En qualsevol moment, un objecte es troba en un estat en particular. Una rentadora pot estar en la fase de remull, rentat o ensabonat, esbandida, centrifugat i apagada, i canviarà d'una a una altra, d'acord amb el diagrama d'estat que es mostra a la figura 2.31.

FIGURA 2.31. Diagrama de transició d'estat



El símbol de la part superior indica l'estat inicial, i el de la part inferior, el final.

Totes les icones que cal utilitzar en el diagrama d'estats són a l'apartat ***States***.

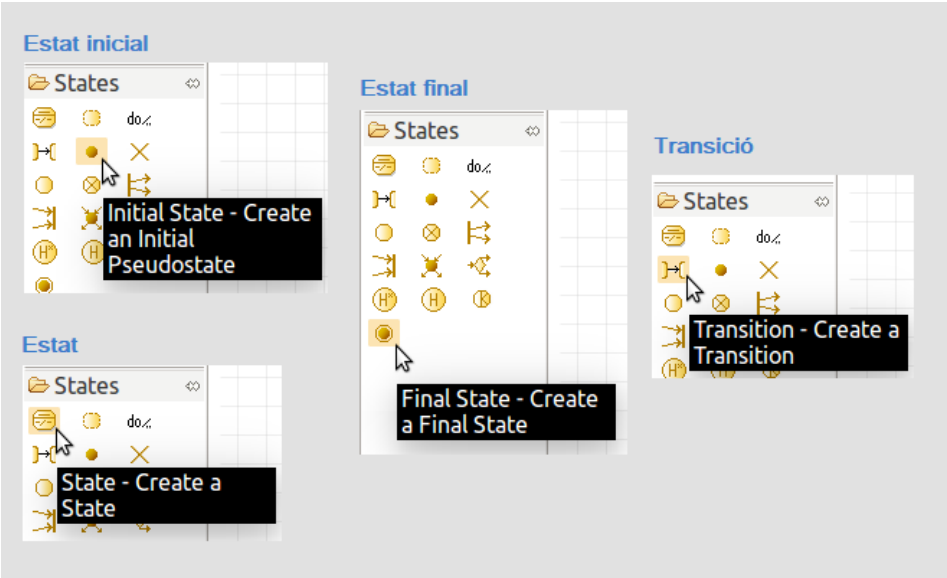
A Modelio, el diagrama de transició d'estats es crea de manera anàloga a com es crea la resta de diagrames, però seleccionant a l'últim pas *State Machine diagram* com a tipus de diagrama.

Un cop creat el diagrama, els estats s'hi afegeixen clicant la icona de la barra d'elements que representa el tipus d'estat que volem afegir (inicial, final o -simplement- estat) i, a continuació, clicant al lloc del diagrama on el volem situar.

Per afegir una transició, també es clica la icona que la representa i, a continuació, als dos estats del diagrama que volen unir-se. El nom de cada transició s'entra en la seva propietat *Guard*.

Podeu veure les icones a la figura 2.32.

FIGURA 2.32. Icones del diagrama de transició d'estats.

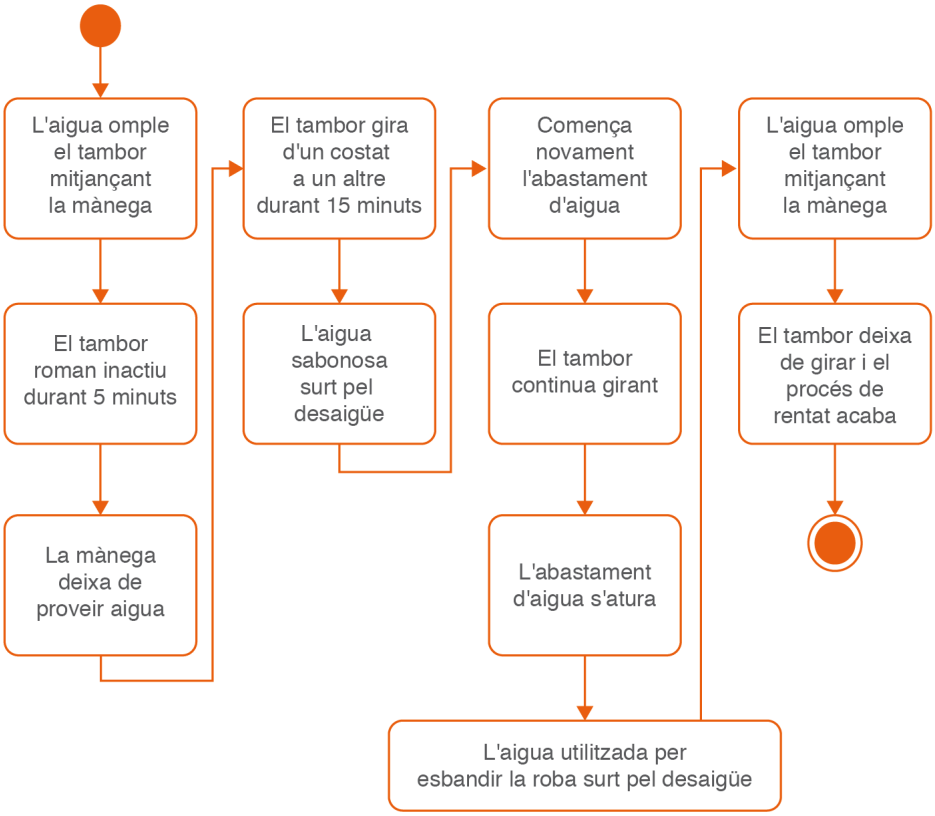


2.4.3 Diagrama d'activitats

Un diagrama d'activitats és una variació del diagrama d'estats UML. El diagrama d'activitats representa les activitats que passen dins un cas d'ús o dins el comportament d'un objecte.

A la figura 2.33 es mostra un possible diagrama d'activitats.

Figura 2.33. Diagrama d'activitats



Fixem-nos que cada una de les columnes podria correspondre a cada un dels estats del procés de rentat: remull, ensabonat, esbandit i centrifugat.

- **Estat de remull:**
  - L'aigua omple el tambor mitjançant la mànega.
  - El tambor roman inactiu durant cinc minuts.
  - La mànega deixa de proveir aigua.
- **Estat d'ensabonat o de rentat:**
  - El tambor gira d'un costat a un altre durant quinze minuts.
  - L'aigua ensabonada surt pel desaigüe.
- **Estat d'esbandit:**
  - Comença novament l'abastament d'aigua.
  - El tambor continua girant.
  - L'abastament d'aigua s'atura.
  - L'aigua utilitzada per esbandir la roba surt pel desaigüe.
- **Estat de centrifugat:**
  - El tambor gira en una sola direcció.
  - El tambor deixarà de girar i el procés de rentat acaba.

A Modelio, el diagrama d'activitats es crea de manera anàloga a com es crea la resta de diagrames, però seleccionant a l'últim pas *Activity diagram* com a tipus de diagrama.

Un cop creat el diagrama, els estats inicial i final i les accions s'afegeixen clicant a la icona de la barra d'elements corresponent al tipus d'element que volem afegir i, a continuació, clicant al lloc del diagrama on el volem situar-lo. Aquestes icones són a l'apartat *Control nodes*.

Per afegir un flux de control, també es clica en primer lloc la icona que el representa, que és a l'apartat *Flows*. A continuació, cal clicar consecutivament els dos elements del diagrama que volen unir-se. Els elements a unir tant poden ser estats com accions com *Forks*, *Joins* o nodes de decisió. Un cop afegit un flux de control al diagrama, es pot canviar la seva forma seleccionant-lo i arrossegant els quadres negres que apareixen en seleccionar-lo.

Per afegir un element *Fork* o *Join*, cal fer clic a la icona *Fork/Join*, que és dins de l'apartat *Control nodes*. Aquesta icona pot representar tant un *Fork* com un *Join* depenent dels fluxos que hi arriben i en surten. Un cop afegit aquest node al diagrama, es pot moure tot arrossegant-lo amb el ratolí. També es pot canviar la seva mida arrossegant novament amb el ratolí de la següent manera:

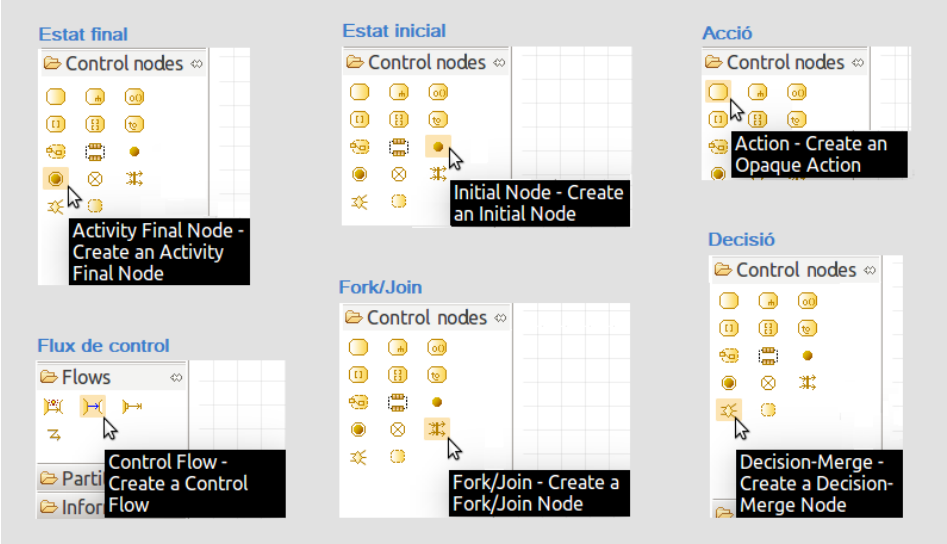
- Seleccionant l'element *Fork-Join*.

- Situant el ratolí en l'extrem que volem allargar o escurçar, entre els dos quadres negres que indiquen que l'element s'ha seleccionant, i arrossegant. El ratolí és a la posició correcta per arrossegar quan té forma de doble fletxa horitzontal. Si la doble fletxa no és horitzontal o el ratolí té una altra forma, no serà possible realitzar l'arrossegament.

Per últim, podeu afegir un node de decisió fent clic a la icona que representa aquests nodes i que es troba a l'apartat *Control nodes* i, a continuació, clicant el punt del diagrama on voleu situar-lo. Podem escriure la condició associada a cadascun dels fluxos de sortida a la seva propietat *Guard*.

Podeu veure les icones d'aquest tipus de diagrama a la figura 2.34.

Figura 2.34. Icones del diagrama d'activitats.



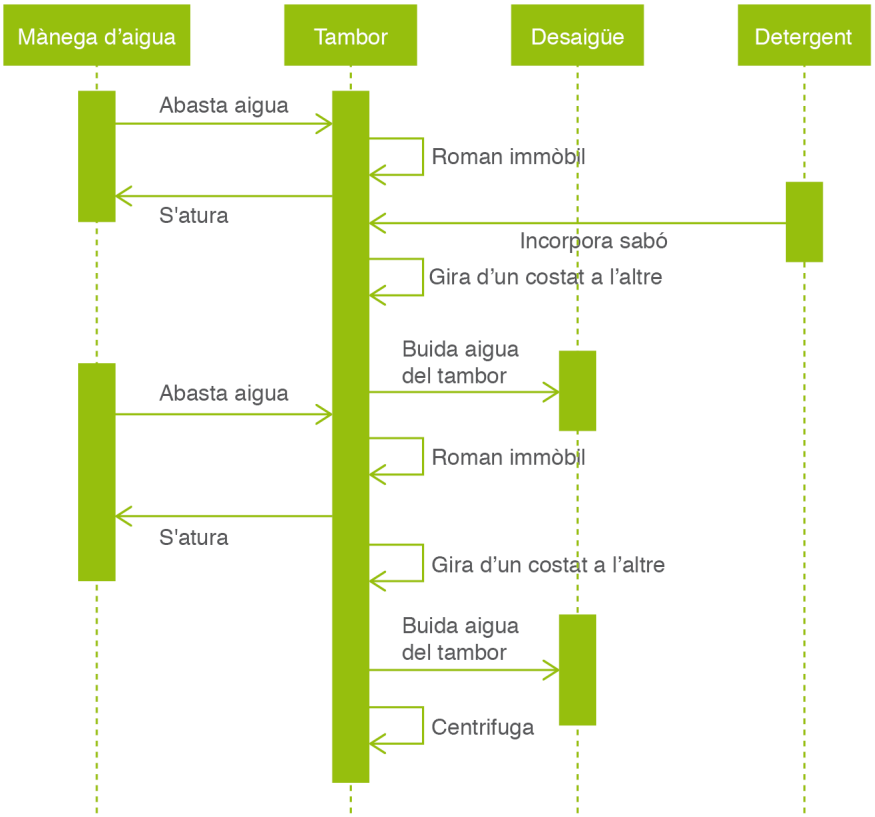
2.4.4 Diagrama de seqüències

En un sistema funcional els objectes interactuen entre si, i aquestes interaccions succeeixen amb el temps. El diagrama de seqüències UML mostra la mecànica de la interacció sobre la base del temps.

Entre els components de la rentadora es troben: una mànega d'aigua (per obtenir aigua fresca), un tambor (on es posa la roba), un sistema de desaigüe i el dispensador de detergent.

Què passarà quan invoqui al cas d'ús *Rentar roba*? Si donem per fet que s'han completat les operacions *afegir roba*, *afegir detergent* i *activar*, la seqüència seria més o menys la que mostra la figura 2.35.

Figura 2.35. Diagrama de seqüència



La seqüència de passos que segueix el procés de rentat és:

1. El sistema d'abastament d'aigua omple el tambor.
2. Durant 3 minuts no hi ha activitat al tambor.
3. El sistema d'abastament deixa de proporcionar aigua.
4. El sistema d'abastament proporciona sabó.
5. Durant 1 minut no hi ha activitat al tambor.
6. El sistema d'abastament deixa de proporcionar sabó.
7. Durant 12 minuts el tambor gira.
8. Comença a sortir aigua pel desaigüe fins que no deixa sabó.
9. El sistema d'abastament d'aigua omple el tambor.
10. Durant 10 minuts el tambor gira.
11. El sistema d'abastament deixa de proporcionar aigua.
12. El desaigüe rep l'aigua que es fa servir per esbandir.
13. Durant 4 minuts el tambor gira.
14. Es finalitza el procés. El tambor deixa de girar.

A Modelio, el diagrama de seqüències es crea de manera anàloga a com es crea la resta de diagrames, però seleccionant a l'últim pas *Sequence diagram* com a tipus de diagrama.

Les línies de vida s'afegeixen clicant la icona corresponent, que és a l'apartat *Nodes*, i, a continuació, fent clic al punt del diagrama on volem afegir-la. Podem donar-les nom tot modificant la propietat *Name*.

Els missatges s'afegeixen:

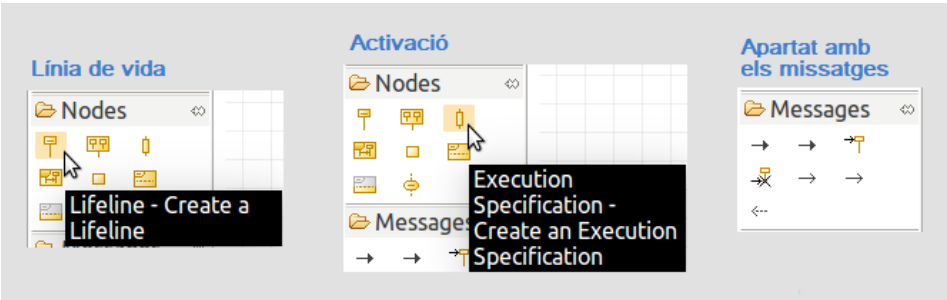
1. Fent clic a la icona corresponent al tipus de missatge que volem afegir, que serà a l'apartat *Messages*.
2. Fent clic a la línia de vida origen del missatge.
3. Fent clic a la línia de vida destinatària del missatge. Veureu que en la línia de vida destinatària s'afegeix automàticament un rectangle, que respresenta una activació. A més, si el missatge és síncron, es crea automàticament el missatge de resposta.

Les activacions també poden afegir-se directament fent clic primer a la icona que les representa, dins l'apartat *Nodes*, i, a continuació, fent clic a la línia de vida a la qual desitgem associar-la.

Pot canviar-se la mida d'una activació seleccionant-la i, a continuació arrosegant el missatge que arriba o surt de l'extrem que volem pujar o baixar. En cas que l'extrem no tingui cap missatge, cal arrossegar el costat superior o inferior de l'activació. El ratolí és a la posició correcta per arrossegar quan té forma de doble fletxa vertical. Si la doble fletxa no és vertical o el ratolí té una altra forma, no serà possible realitzar l'arrossegament.

Podeu veure les icones d'aquest tipus de diagrama a la figura 2.36.

Figura 2.36. Icones del diagrama de seqüències.

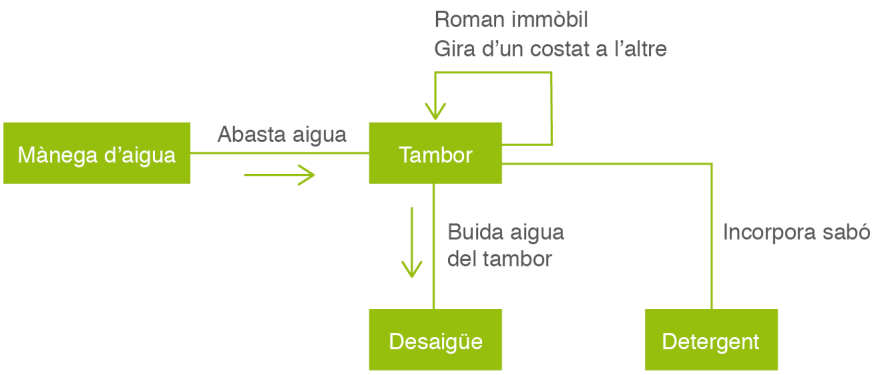


2.4.5 Diagrama de comunicació

Els elements d'un sistema treballen en conjunt per complir amb els objectius del sistema, i un llenguatge de modelització haurà de comptar amb una forma de representar això. El diagrama de col·laboracions UML està dissenyat amb aquesta finalitat.

A la figura 2.37 es pot observar un possible diagrama de comunicació.

Figura 2.37. Diagrama de comunicació



A Modelio, el diagrama de comunicació es crea de manera anàloga a com es crea la resta de diagrames, però seleccionant a l'últim pas *Communication diagram* com a tipus de diagrama.

Els objectes s'afegeixen fent clic en primer lloc a sobre de la icona *Communication Node* i, a continuació, al punt del diagrama on volem inserir-lo. Cada objecte quedarà representat com un rectangle. Podem modificar el seu nom des de la pestanya *Properties*.

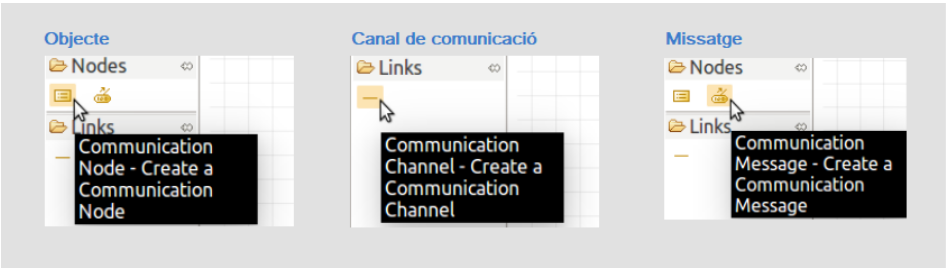
La inserció d'un missatge entre dos objectes es fa en dos passos:

1. Insertar un *canal de comunicació* entre tots dos objectes. Això es fa clicant la icona *Communication Channel* i, a continuació, clicant en els dos objectes que es comunicaran amb aquest missatge. L'ordre és indiferent.

2. Clicar la icona *Communication Message* i, a continuació, el canal que uneix els dos objectes que es comuniquen amb aquest missatge. El punt on es faci el clic és important, ja que determinarà el **sentit** del missatge: el missatge anirà des de l'objecte més proper al clic cap a l'objecte més llunyà al clic. El missatge resultant tindrà forma de fletxa, que indicarà el sentit del missatge, i apareixerà més propera a l'objecte emissor que no pas a l'objecte receptor.

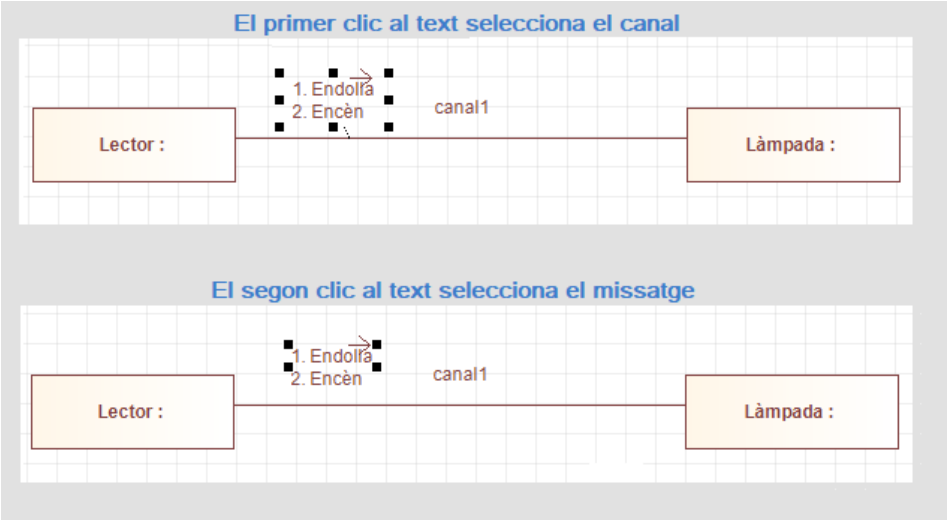
Podeu veure les icones d'aquest tipus de diagrama a la figura **??**.

Figura 2.38. Icones del diagrama de comunicació



Un canal admet més d'un missatge. Per representar-ho al diagrama cal repetir el pas *2* tantes vegades com ens calgui. Els missatges que van en el mateix sentit es representaran al voltant de la fletxa corresponent segons l'ordre d'inserció. Cadascun ve representat per una línia de text. Podem canviar el text que descriu el missatge seleccionant amb el ratolí la línia corresponent i modificant la seva propietat *Name*. Normalment, en clicar-lo el primer cop es seleccionarà tot el canal i els petits quadres negres que indiquen que l'element s'ha seleccionat apareixeran al voltant de tots els missatges que van en el mateix sentit. Per seleccionar només un missatge cal tornar a fer clic a sobre d'aquest; llavors els quadres negres només envoltaran el missatge triat per indicar que ara l'únic missatge seleccionat és aquest. A la figura 2.39 podeu observar la seqüència de clics que cal fer.

Figura 2.39. Selecció d'un missatge



A la figura 2.40 podeu observar un exemple de diagrama de comunicació realitzat amb Modelio:

Figura 2.40. Diagrama de comunicació elaborat amb Modelio