Anàlisi UML

Jordi Pradel Miquel
Jose Raya Martos

PID_00171147

Universitat Oberta de Catalunya
www.uoc.edu

Índex

Introducció

Tal com hem vist al mòdul "Orientació a objectes", l'orientació a objectes és un paradigma que pot resultar molt útil no solament per a tasques de programació, sinó també per a tasques d'anàlisi. D'altra banda, al mòdul "Requisits" hem vist els requisits i la importància de fer una documentació de requisits de qualitat, i també els casos d'ús com una manera flexible de documentar requisits funcionals.

En aquest mòdul veurem com combinar l'ús de l'orientació a objectes i els casos d'ús per a fer una documentació de requisits fent servir UML. El tipus de documentació que elaborarem és força completa i pot arribar a ser formal, però pot ser útil per a tot tipus de metodologies.

En primer lloc veurem com el concepte d'orientació a objectes es pot aplicar a l'anàlisi i com UML constitueix un llenguatge estàndard per a crear models visuals de programari. També veurem una petita introducció al llenguatge i mostrarem els diferents tipus de diagrames que suporta.

A continuació veurem com podem fer servir UML per a complementar les especificacions textuals de casos d'ús. Veurem el diagrama de casos d'ús, que representa de manera visual els casos d'ús, els actors i les relacions entre aquests, i el diagrama d'activitats, que permet modelitzar visualment el comportament d'un cas d'ús com si fos un procés.

Els casos d'ús que haurem vist fins a aquest punt no estan encara enllaçats amb una interfície gràfica d'usuari. El pas següent serà, per tant, crear un model d'aquesta interfície, que ens permetrà resoldre dubtes d'usabilitat o d'arquitectura de la informació, entre altres.

Finalment veurem com podem fer modelització del domini fent servir UML. El model del domini serà una representació de les classes conceptuals del món real en el domini del sistema que estudiem.

Al llarg de tot el mòdul farem servir, com a fil conductor, exemples basats, tots, en un mateix sistema: una universitat que vol oferir un campus virtual per als seus alumnes i professors.

Objectius

Els objectius que l'estudiant ha d'haver assolit una vegada treballats els continguts d'aquest mòdul són:

1. Saber fer servir l'orientació a objectes per a fer anàlisi de programari per a sistemes d'informació.

2. Saber fer servir la notació UML per a documentar models d'anàlisi orientats a objectes.

3. Saber fer servir els casos d'ús per a fer anàlisi funcional de programari per a sistemes d'informació.

4. Saber fer servir els diagrames d'activitats per a documentar detalladament els casos d'ús complexos com a processos.

5. Saber modelitzar la interfície gràfica d'usuari del programari mitjançant casos d'ús concrets, esbossos de les pantalles i mapes navegacionals.

6. Saber fer modelització del domini mitjançant diagrames de classes UML.

1. Anàlisi orientada a objectes amb UML

1.1. Anàlisi orientada a objectes

Vegeu també

L'orientació a objectes s'estudia al mòdul "Orientació a objectes".

L'orientació a objectes va néixer com una eina de programació en què els programadors construeixen una abstracció del problema que estan mirant de resoldre en lloc de construir una abstracció de l'ordinador. Els programadors que fan servir orientació a objectes, per tant, creen classes de programari que representen aquells conceptes del món real que són rellevants al sistema.

A mesura que l'orientació a objectes va guanyar terreny, els analistes es van adonar que podia ser una eina molt útil per a construir els seus models d'anàlisi. A més, resultava molt útil fer servir models deliberadament semblants al programari que es construeix, ja que expressar l'anàlisi en termes fàcilment traslladables a programari orientat a objectes facilita no solament la tasca de disseny i programació sinó també el canvi, l'extensió i el manteniment del programari un cop desenvolupat.

1.2. El llenguatge UML

1.2.1. Motivació del llenguatge UML

El llenguatge UML és el llenguatge estàndard per a crear models visuals de programari.

Com a tal, UML és un llenguatge de propòsit general (intenta cobrir tots els possibles models de tots els tipus de programari) i visual (els models es representen, principalment, en forma de diagrames).

Un dels motius de l'èxit del llenguatge UML és que és independent del mètode de desenvolupament emprat. Així doncs, diferents mètodes de desenvolupament utilitzaran de manera diferent els diagrames proposats per UML.

Com a llenguatge, la utilitat principal de l'UML és permetre la **comunicació**. L'adopció per part de tota la indústria d'un mateix llenguatge permet que tothom pugui entendre els models creats per una altra persona o equip i, per tant, que es puguin discutir les propietats d'un sistema a partir dels models.

Segons Fowler (2004) podem distingir tres usos principals per al llenguatge UML:

- **Com a llenguatge per a fer un croquis**. Es tracta de fer un ús informal (normalment dibuixant a mà alçada) per tal d'explicar o explorar algun aspecte en concret d'un sistema informàtic. La clau aquí és la selectivitat: només tenim en compte els detalls que són rellevants per a la discussió que estem duent a terme en aquest moment i obviem la resta. D'aquesta manera, aconseguim maximitzar la relació entre l'esforç dedicat a la creació dels models i la utilitat obtinguda en canvi.

- **Com a llenguatge per a fer un plànol**. Construir models més detallats que ens permetin documentar el sistema amb prou detall per a facilitar-ne la implementació (fins i tot amb generació automàtica de codi) o per a capturar de manera visual els detalls sobre l'estructura i comportament d'un sistema existent (enginyeria inversa). La idea és poder aconseguir que un dissenyador creï els plànols del sistema i que, més endavant, els programadors només hagin d'escriure el codi, però no prendre decisions sobre com ha de ser el sistema per desenvolupar.

- **Com a llenguatge de programació**. Aquest seria el cas de les eines MDA. Es tracta de crear models tan detallats del sistema que el codi que es pugui generar de manera automatitzada no hagi de ser modificat (ni tan sols llegit) pels desenvolupadors. Aquest és un camp en què, actualment, s'estan fent molts esforços de recerca.

Vegeu també

Podeu trobar més informació sobre MDA al mòdul "Introducció a l'enginyeria del programari".

Una altra manera de caracteritzar l'ús de l'UML és tenint en compte què es vol modelitzar. Segons Fowler (2004) podem distingir entre dues perspectives:

- **Perspectiva conceptual**. El model representa una descripció dels conceptes del domini que estem estudiant.

- **Perspectiva de programari**. El model representa el sistema informàtic i, per tant, hi ha una correspondència directa entre els elements del model i les parts del sistema.

A l'hora d'estudiar el llenguatge UML caldrà decidir quin ús en voldrem fer sota aquestes dues classificacions, ja que en funció de la necessitat que tinguem, farem servir l'UML d'una o altra manera.

Pel que fa a la primera classificació, la majoria dels models que crearem seran croquis del que seria un sistema real. Això ens permetrà centrar-nos en els aspectes didàctics dels models sense haver-nos de preocupar pels detalls del sistema real. D'altra banda, això també ens permetrà no haver de dependre de cap eina CASE en concret.

Pel que fa a la segona classificació, durant l'anàlisi ens situarem en la perspectiva conceptual. Els models que fem fan referència als conceptes del domini que estem analitzant (l'espai del problema), però no al programari en si mateix (l'espai de la solució).

1.2.2. Origen del llenguatge UML

Per a comprendre el valor (i alguns dels defectes) de l'UML és important conèixer el seu origen. Abans de la publicació de l'UML cada mètode de desenvolupament utilitzava la seva pròpia notació. Això era un problema, ja que dificultava enormement la comunicació entre enginyers de programari.

Part del problema era que, mentre que totes les notacions eren més o menys similars, hi havia multitud de detalls que canviaven d'una a l'altra, la qual cosa feia encara més complicada la comunicació. Tot i que hi va haver alguns esforços d'estandardització, la reacció per part dels metodòlegs no va ser gaire positiva.

Aquesta situació va començar a canviar quan Jim Rumbaugh va passar a treballar, juntament amb Grady Booch, a la companyia Rational, en la creació del **mètode unificat**. Més endavant, Ivar Jacobson es va unir a la companyia i, entre els tres, van assentar les bases del que seria el llenguatge UML.

Durant la segona meitat de la dècada dels noranta va tenir lloc l'estandardització, per mitjà de l'OMG, del llenguatge UML. D'aquesta manera, UML deixava de ser un producte d'una sola companyia (Rational) per a passar a ser un estàndard obert que qualsevol altra companyia podia implementar.

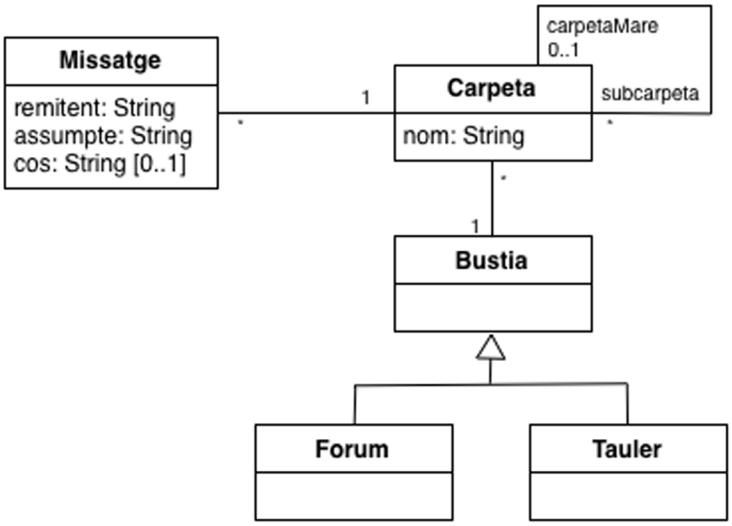
Aquest procés d'estandardització va ser clau perquè s'incorporés la feina d'altres metodòlegs i es pogués arribar a un estàndard que tota la indústria acceptés. Com a contrapartida, tenim un estàndard que, de vegades, pot semblar una mica inconsistent, ja que incorpora elements amb orígens molt diferents.

1.2.3. Tipus de diagrames UML

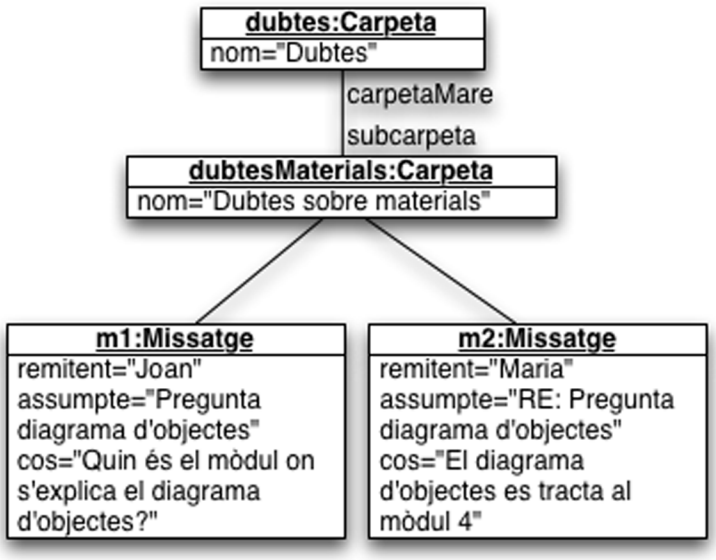
La versió 2 d'UML defineix 13 tipus de diagrama. A continuació veurem un exemple de cada tipus de diagrama. La idea no és estudiar els diagrames en detall sinó fer-nos una idea general del seu aspecte visual i la seva utilitat.

El **diagrama de casos d'ús** (*use case diagram*) serveix per a modelitzar quins són els casos d'ús del sistema i quins són els actors que hi estan relacionats.

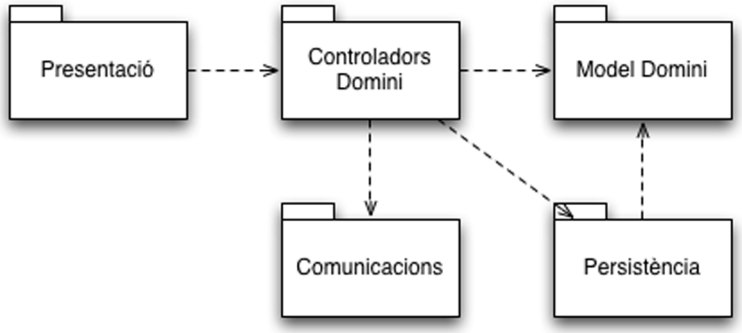
El **diagrama de classes** (*class diagram*) és, probablement, el diagrama més conegut de l'UML. Serveix per a modelitzar les classes d'objectes i les seves relacions, fent èmfasi en els aspectes estructurals més que no pas en el comportament.



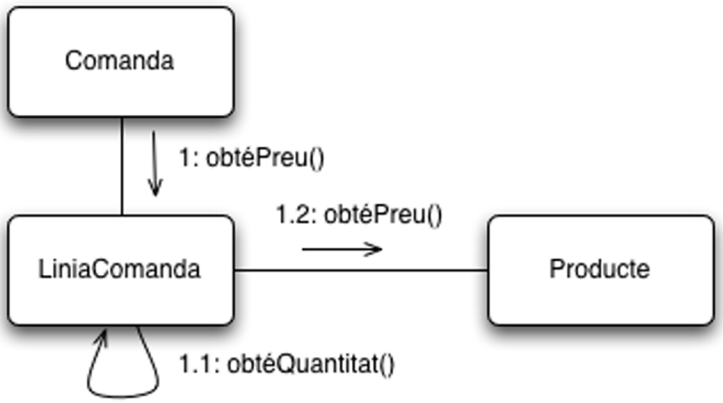
La versió 2 del llenguatge UML va oficialitzar per primer cop el **diagrama d'objectes** (*object diagram*). Aquest és un diagrama similar al de classes en què el que representem no són classes sinó intàncies de classe (objectes).



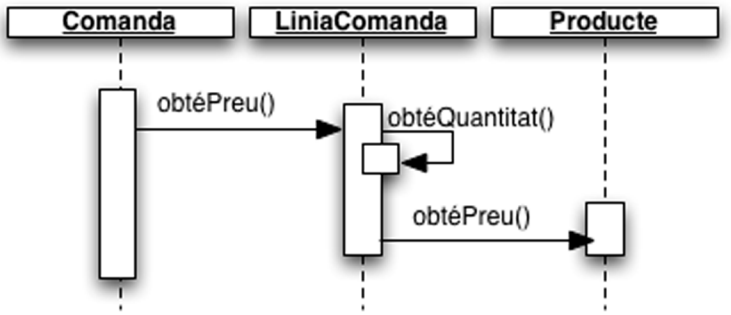
El **diagrama de paquets** (*package diagram*) ens permet modelitzar les dependències entre paquets. Un paquet és un grup d'elements UML com ara classes o casos d'ús. Per tant, podem fer servir la notació de paquets en altres diagrames. Així doncs, un paquet d'un diagrama de paquets pot representar un subsistema, un grup de classes, o qualsevol altra agrupació d'elements. Al diagrama de paquets mostrarem els paquets i les dependències entre aquests.



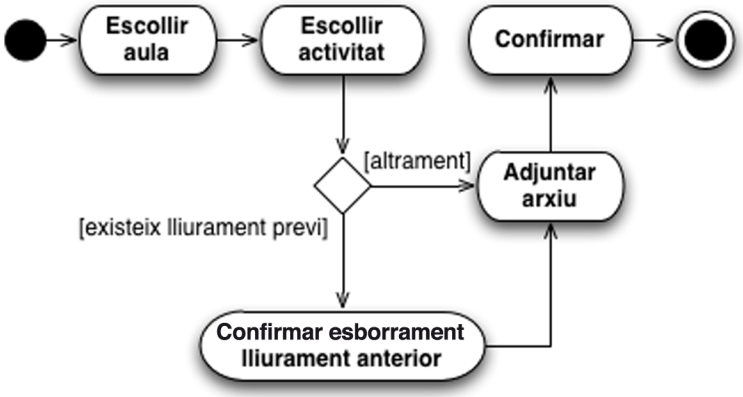
El **diagrama de comunicació** (*communication diagram*) (anomenat *diagrama de col·laboració* a la versió 1 d'UML) ens permet modelitzar la interacció entre diversos objectes fent èmfasi en l'aspecte estructural (quins objectes estan "connectats" a quins altres i hi col·laboren).



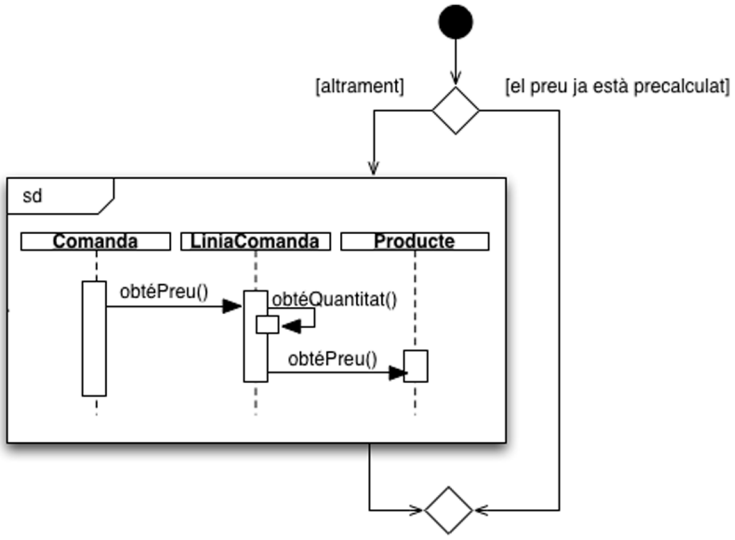
El **diagrama de seqüència** (*sequence diagram*) és similar al de comunicació en el sentit que modelitza el mateix (la interacció entre diversos objectes) però, en aquest cas, fent èmfasi en la seqüència temporal.



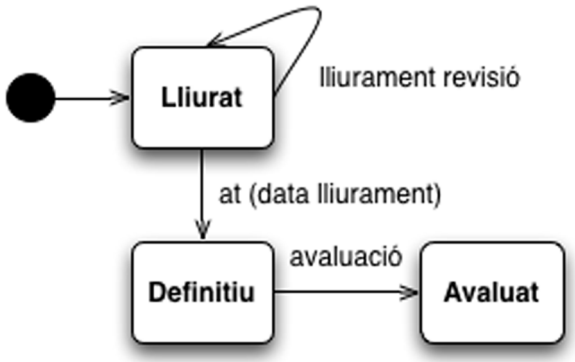
El **diagrama d'activitats** (*activity diagram*) s'utilitza, habitualment, per a descriure processos de manera similar als diagrames *flow-chart*.



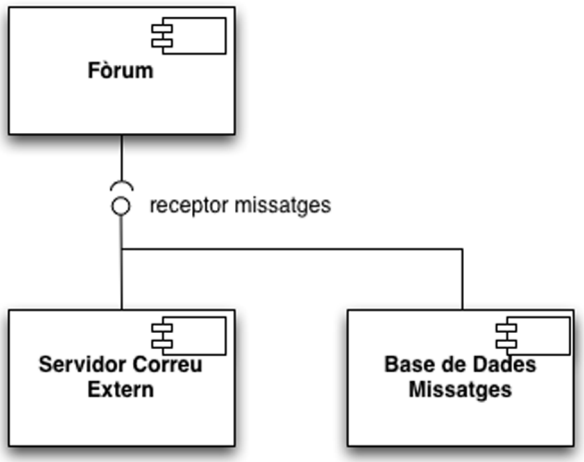
Els dos diagrames anteriors (seqüència i activitats) els podem combinar en un **diagrama de visió general d'interacció** (*interaction overview diagram*). El podem veure com un diagrama d'activitats en què hem substituït les activitats per diagrames de seqüència. Aquest és un tipus de diagrama que va introduir per primer cop la versió 2 d'UML i no està tan difós com els altres.



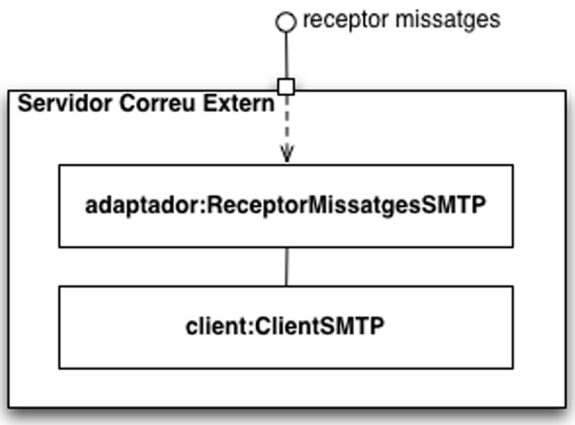
El **diagrama d'estats** (*state machine diagram*) modelitza estats i transicions. La idea del diagrama d'estats és modelitzar com afecten a un objecte els diferents esdeveniments que poden tenir lloc al sistema



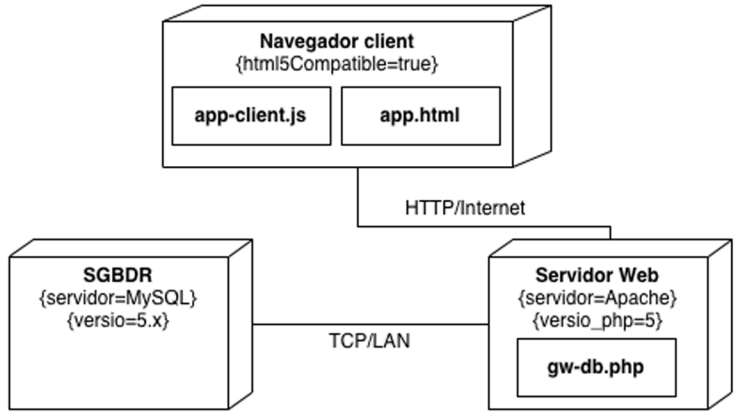
El **diagrama de components** (*component diagram*) modelitza els diferents components que formen part del nostre sistema i les interfícies que fan servir per a comunicar-se entre si.



El **diagrama d'estructura composta** (*composite structure diagram*) ens permet modelitzar l'estructura interna en temps d'execució d'una classe o component.



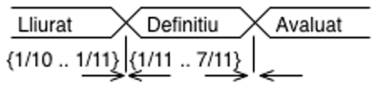
El **diagrama de desplegament** (*deployment diagram*) modelitza la distribució física, en temps d'execució, dels diferents artefactes de programari.



El **diagrama de temps** (*timing diagram*) és molt habitual en el món de l'electrònica i, a partir de la seva versió 2, forma part del conjunt de diagrames estàndard d'UML, tot i que la seva aplicabilitat al desenvolupament del programari es limita a casos molt concrets, com ara el programari de temps real.

Programari de temps real

El programari de temps real és aquell que es fa servir en sistemes en què hi ha restriccions de temps real, és a dir, hi ha un límit de temps entre que es produeix un esdeveniment i que el sistema hi dóna resposta.



1.3. Exemple

Al llarg d'aquest mòdul farem servir un exemple de domini que anirem desenvolupant en petits exemples al llarg del mòdul.

Prendrem com a exemple una universitat per a la qual volem desenvolupar un sistema d'informació que gestioni la informació dels cursos impartits, les matrícules dels estudiants i les activitats que fan, incloent-hi les qualificacions.

La universitat té un seguit d'assignatures, per a cadascuna de les quals s'imarteix una edició (que també anomenem *curs*) cada semestre. Per a cada curs d'una assignatura es creen un seguit de grups, cadascun dels quals té un professor i una sèrie d'alumnes matriculats.

Alguns cursos són presencials i, per a aquests, volem saber l'horari setmanal i, per a cada franja horària del curs, l'aula on s'imparteix. Les aules estan situades en edificis que, alhora, pertanyen a un determinat campus.

Hi ha, també, un campus virtual, on cada grup disposa d'un tauler –una bústia de missatges on el professor pot escriure però els alumnes només poden llegir– i un fòrum –una bústia de missatges on tant el professor com els alumnes poden participar. Els missatges, dins el tauler o fòrum, es poden agrupar per carpetes.

Els professors proposen, cada curs, unes activitats comunes a tots els grups de l'assignatura –com ara exàmens o pràctiques. La qualificació de cada alumne en una assignatura es calcularà a partir de les qualificacions que tregui a cada activitat entregada.

2. Model de casos d'ús

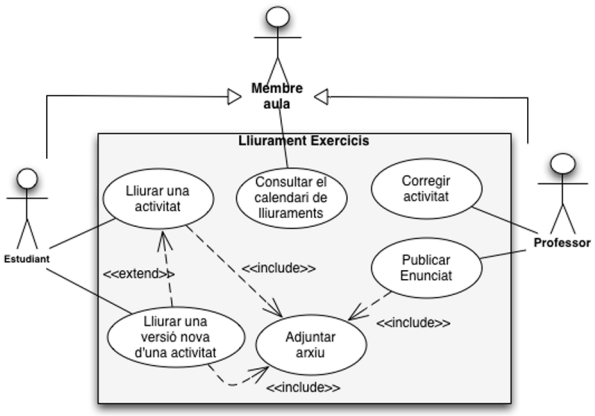
Anteriorment hem vist la tècnica dels casos d'ús per a la documentació de requisits. També hem vist que la notació més habitual per a documentar els casos d'ús és la documentació textual. En aquest apartat veurem com podem utilitzar el llenguatge UML per a complementar la documentació dels casos d'ús mitjançant diagrames.

Vegeu també

Hem vist la tècnica dels casos d'ús per a la documentació de requisits al mòdul "Requisits".

2.1. El diagrama de casos d'ús

El primer diagrama que veurem és el diagrama de casos d'ús. La notació del diagrama és molt senzilla; de fet, a continuació veurem un exemple d'aquest diagrama on s'utilitza tota la notació que necessitarem en aquest mòdul.



Cal fixar-se, però, que aquest diagrama no conté informació sobre quin és el comportament del sistema (per exemple, què passa si un estudiant intenta lliurar una activitat fora de termini).

El diagrama de casos d'ús complementa, però en cap cas no substitueix, la descripció textual dels casos d'ús, ja que no inclou informació sobre quin és el comportament del sistema.

Vegeu també

Podeu trobar més informació sobre els casos d'ús i la seva especificació textual al mòdul "Requisits".

Quina és, doncs, la utilitat d'aquest diagrama? Per una banda, ens permet relacionar visualment els actors i els casos d'ús i, per l'altra, ens dóna una visió ràpida de quina és la funcionalitat que el sistema ofereix als diferents actors.

2.1.1. Actors

Al diagrama d'exemple podem veure tres actors: "Estudiant", "Professor" i "Membre aula", que es representen amb una mena de ninot. Això és perquè, habitualment, els actors seran persones (els usuaris del sistema) tot i que, fins i tot en el cas en què un actor sigui un sistema extern, el representarem amb el mateix símbol.

La fletxa que va d'Estudiant a Membre aula (i la que va de Professor a Membre aula) significa que aquest actor és una especialització de l'actor *Membre aula*. A efectes pràctics, això vol dir que tot el que es diu sobre l'actor *Membre aula* és aplicable a l'actor *Estudiant* (i també a l'actor *Professor*).

L'avantatge de l'ús de la relació de generalització/especialització entre actors és que ens permet simplificar el diagrama, ja que elimina la necessitat de representar per repetit allò que és comú a més d'un actor (com ara les seves relacions amb els casos d'ús). Així, a l'exemple, no ens cal representar la relació d'estudiants i professors amb el cas d'ús "Consultar el calendari de lliuraments", ja que l'hem representada a l'actor més general.

2.1.2. La frontera del sistema

El requadre amb l'etiqueta "Lliurament exercicis" representa la frontera del sistema: tot el que queda dintre del requadre forma part del sistema, mentre que tot el que queda fora del requadre (en aquest cas, els actors) no en forma part.

L'etiqueta ens pot ajudar a aclarir l'àmbit dels casos d'ús: quin sistema o subsistema estem tenint en compte.

2.1.3. Els casos d'ús

Els casos d'ús es representen mitjançant el·lipses que inclouen un text: el nom del cas d'ús. D'aquesta manera podem relacionar els casos d'ús que apareixen al diagrama amb els casos d'ús que apareixen a la documentació dels requisits.

Per a facilitar la comprensibilitat del diagrama, és molt convenient evitar barrejar casos d'ús de nivell d'abstracció diferent al mateix diagrama. Per exemple, al nostre diagrama d'exemple, tots els casos d'ús (menys "Adjuntar arxiu", del qual parlarem més endavant) són del mateix nivell (usuari).

Si volem representar gràficament els casos d'ús en què es descompon un altre cas d'ús, sempre ho podem fer en un diagrama a part.

2.1.4. Relacions entre casos d'ús i actors

Les línies que veiem entre els casos d'ús i els actors indiquen que hi ha alguna relació entre aquests. Un actor pot estar relacionat amb un cas d'ús perquè en sigui l'actor principal o bé perquè en sigui un actor de suport.

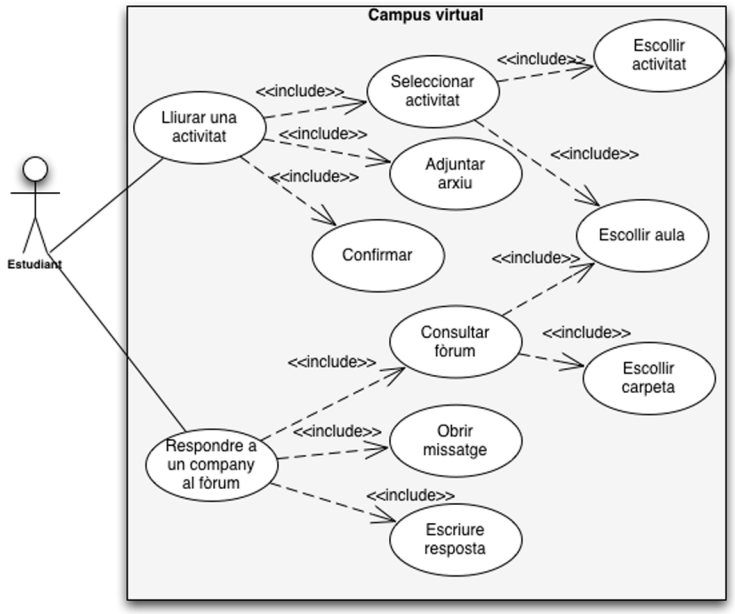
En el nostre exemple, podem veure que qualsevol membre de l'aula (i això inclou els estudiants i els professors) pot consultar el calendari de lliuraments, que els estudiants poden lliurar una activitat o bé lliurar una versió nova d'una activitat i que els professors poden corregir una activitat o publicar un enunciat.

2.1.5. Relacions entre casos d'ús: inclusió

Al diagrama d'exemple podem veure una relació d'inclusió entre els casos d'ús "Lliurar una activitat" i "Adjuntar arxiu" i una altra entre "Publicar enunciat" i, de nou, "Adjuntar arxiu". La manera de llegir aquesta relació és, per exemple, que "Lliurar una activitat" inclou "Adjuntar arxiu", tal com indica el sentit de la fletxa.

Com ja hem vist, les relacions d'inclusió, sovint, es donen entre casos d'ús de diferent nivell. Tenint en compte el que hem dit anteriorment sobre barrejar casos d'ús de diferent nivell al mateix diagrama, és discutible la conveniència de mostrar el cas d'ús al mateix diagrama que la resta, ja que la presència del cas d'ús al diagrama pot confondre més que no pas ajudar els seus destinataris.

Un error molt habitual és intentar representar el flux d'esdeveniment del cas d'ús fent servir relacions d'inclusió per a descompondre'l en objectius més concrets:

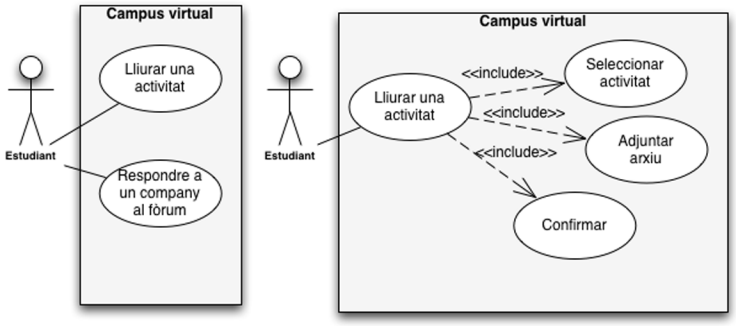


El problema amb aquest tipus de descomposició és que es corre el risc d'intentar substituir la descripció textual, cosa que no es pot fer, ja que és més difícil de llegir i entendre i, alhora, no aporta tota la informació que conté la descripció. Per exemple, no tenim cap manera d'indicar la seqüència dels esdeveniments ni de relacionar l'escenari principal amb les extensions: primer adjuntem un arxiu i després seleccionem l'activitat o a l'inrevés?

El primer que cal tenir en compte és que el diagrama de casos d'ús no mostra la seqüència d'esdeveniments d'un cas d'ús; la distribució a l'espai dels casos d'ús no té cap significat i no implica, per tant, que un cas d'ús vagi abans de l'altre.

A més, per assegurar-nos que els diagrames són fàcils d'entendre, no hauríem de tenir grans diferències pel que fa al nivell dels seus objectius. Cal evitar casos, doncs, com el de l'exemple en què tenim, per exemple, un cas d'ús de nivell d'usuari "Lliurar una activitat" i els casos d'ús a escala de subtasca "Escollir aula" i "Escollir activitat".

En tot cas, si ho considerem necessari, sempre podem fer servir més d'un diagrama i agrupar així els casos d'ús segons el criteri que considerem més adient. D'altra banda, també cal tenir en compte que un cas d'ús pot aparèixer en més d'un diagrama sense que això impliqui cap duplicitat, ja que la descripció textual només serà en un lloc.



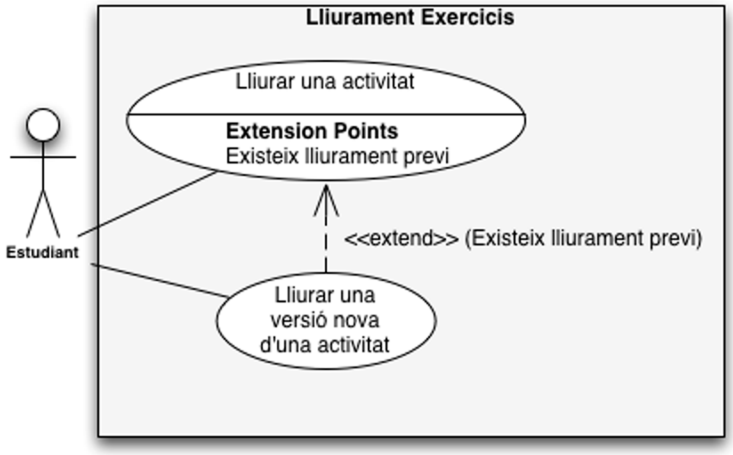
En aquest cas hem creat dos diagrames. En el primer hem agrupat els casos d'ús pel nivell del seu objectiu mostrant només els de nivell usuari. En el segon, en canvi, ens centrem en la descomposició d'un cas d'ús i mostrem, per tant, el cas d'ús i els de nivell immediatament inferior al seu que hi estan relacionats. Podríem fer un tercer diagrama que mostri la descomposició de "Respondre a un company del fòrum", un altre per a "Seleccionar activitat", etc.

2.1.6. Relacions entre casos d'ús: extensió

A l'exemple inicial podem trobar un exemple d'extensió entre els casos d'ús "Lliurar una activitat" i "Lliurar una versió nova d'una activitat". En aquest cas, el sentit de la fletxa ens indica que "Lliurar una versió nova d'una activitat" és una extensió de "Lliurar una activitat".

Com hem vist anteriorment, aquesta relació es fa servir poc. Habitualment, en lloc de crear un nou cas d'ús "Lliurar una nova versió d'una activitat" hauríem afegit aquesta extensió al cas d'ús "Lliurar una activitat" modificant-ne, per tant, la descripció textual.

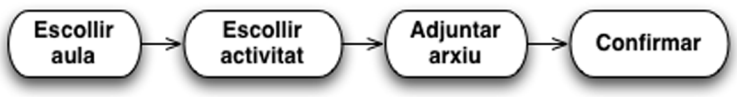
Opcionalment, UML permet indicar l'anomenat *punt d'extensió*, que és un text que identifica en quin punt de l'escenari principal es produeix l'esdeveniment que provoca el comportament alternatiu: d'aquesta manera podem identificar el punt on s'inicia el nou comportament sense necessitat de fer referència a un número de pas concret, facilitant així el manteniment del model de requisits. El problema, però, és que és necessari identificar, *a priori*, tots els possibles punts d'extensió del cas d'ús i, per això, és una pràctica en desús.



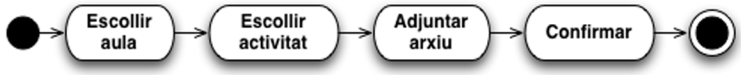
2.2. El diagrama d'activitats

El diagrama d'activitats combina diferents idees del món de la modelització de processos i, per tant, ens pot ajudar a documentar el comportament del sistema si el veiem com un procés. La modelització d'activitats fa èmfasi en la seqüència i la coordinació de les possibles accions per documentar més que no pas en qui és responsable d'executar-les.

L'element bàsic d'aquest diagrama és l'activitat, que representa el fet que algú fa quelcom. L'altre element bàsic són les transicions o fluxos, que representen el fet que es passa d'una activitat a la següent. A continuació, tenim un diagrama d'activitats molt senzill per al cas d'ús "Lliurar una activitat".



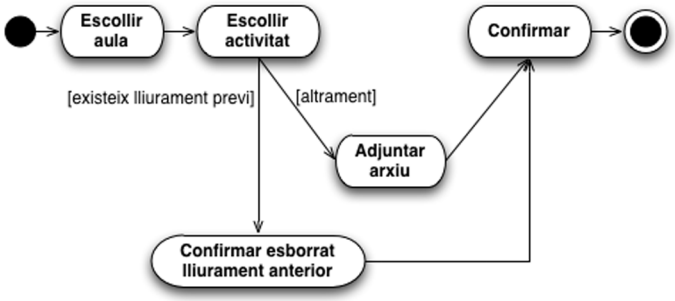
En aquest cas, cada pas del cas d'ús l'hem representat com una activitat i les diferents transicions ens donen una idea sobre quina és la seqüència en la qual tenen lloc les diferents activitats. S'acostuma a facilitar la lectura de la seqüència, indicant-ne l'inici i el final:



En aquest exemple podem veure fàcilment que el procés comença per "Escollir aula", tal com indica el punt negre, i el final es produeix després de "Confirmar", tal com indica la transició cap al punt negre dins un cercle. Tot i que, en aquest cas, només n'hi ha un, podem posar tants símbols de final com vulguem al nostre diagrama.

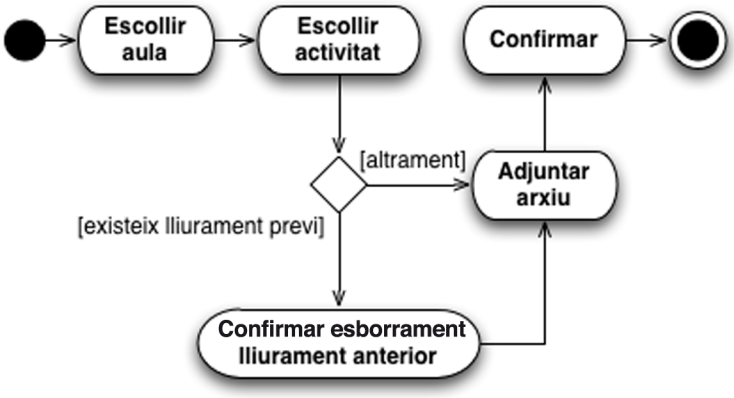
2.2.1. Lògica condicional

Un dels avantatges del diagrama d'activitats és que podem representar la lògica condicional que ens porta a l'execució dels diferents escenaris. Així doncs, continuant amb el cas anterior, podríem fer el diagrama següent:



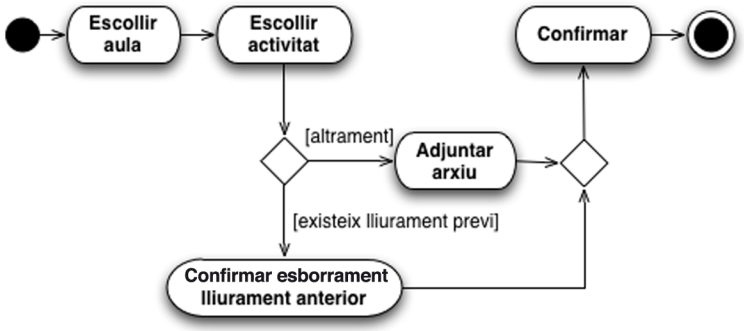
En aquest diagrama hem introduït un nou element de notació, la condició, també anomenada *condició de guarda*, que es representa mitjançant el text entre claudàtors a la transició. La condició *existeix lliurament previ* ens indica que aquesta transició només es produeix quan la condició és certa. Per tant, "Escollir activitat" té dues transicions possibles: una es produeix quan hi ha un lliurament previ i l'altra quan no és així.

Si volem fer més explícit el punt en què hi ha una presa de decisió, el podem representar mitjançant l'element gràfic de decisió, que es representa com un rombe:



La decisió té un flux d'entrada i diversos fluxos de sortida, cadascun amb una condició. Només un dels fluxos de sortida (aquell per al qual la condició es compleixi) serà el que es prengui en un escenari concret. Per això, les diferents condicions haurien de ser mútuament excloents.

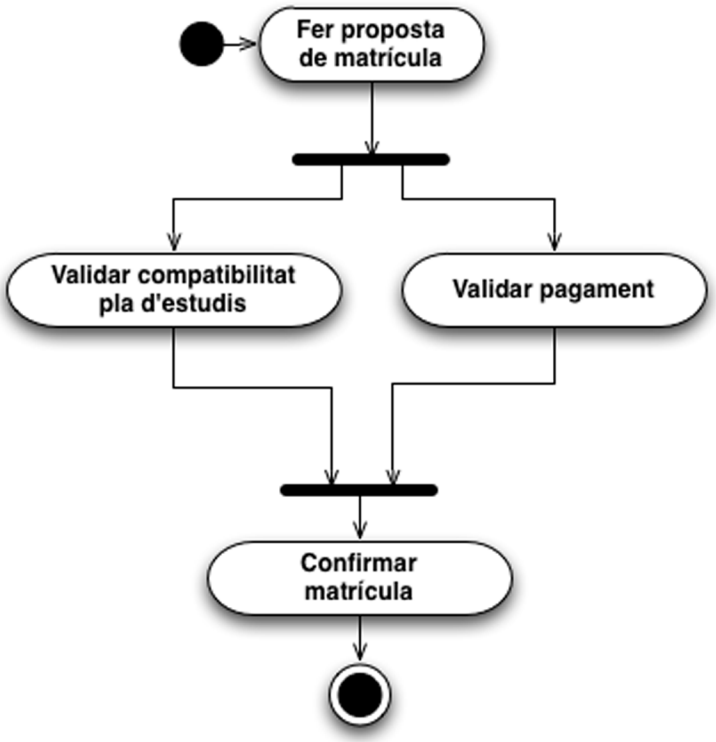
Un altre element condicional que podem fer servir és la fusió (*Merge*), que es representa com un rombe (igual que la decisió) però que té diversos fluxos d'entrada i només un de sortida, que es prendrà quan s'arribi per algun dels fluxos d'entrada:



Com hem vist, les decisions i fusions són elements opcionals que ens permeten fer èmfasi en la lògica condicional. Habitualment, però, només farem servir, amb aquest propòsit, les decisions.

2.2.2. Paral·lelització

Amb els elements de lògica condicional, els diferents camins són excloents: O bé se'n pren un o bé es pren l'altre, però sempre hi ha un únic camí (i, per tant, una única activitat) executant-se en un moment concret. De vegades, però, ens pot interessar poder indicar que dos camins s'executen de manera paral·lela:



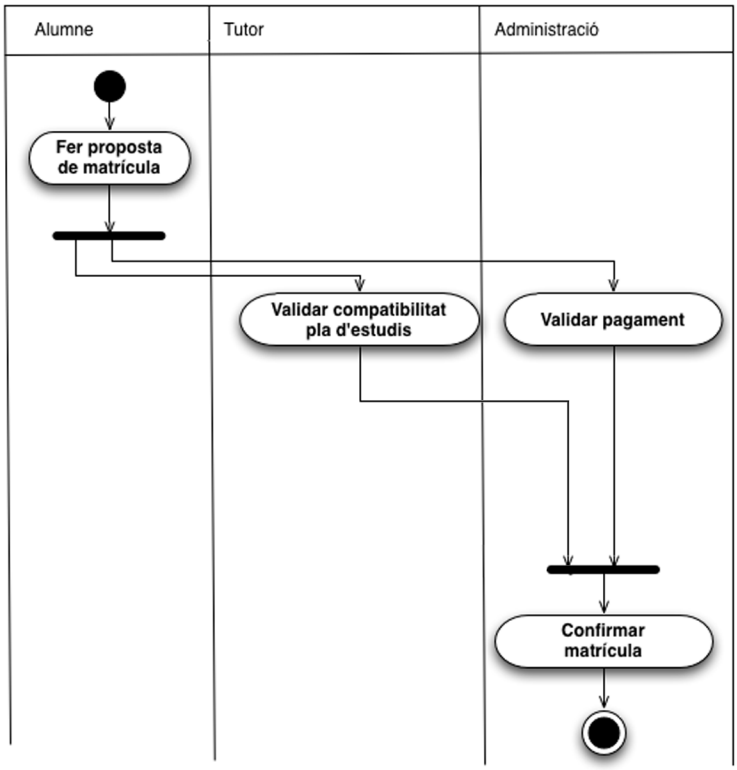
En aquest exemple, un cop s'ha fet la proposta de matrícula, s'inicien dos fluxos paral·lels: un que comprova la compatibilitat amb el pla d'estudis i un altre que comprova que el pagament es faci correctament. Tots dos s'executen en paral·lel, ja que no cal esperar que un acabi per a començar l'altre. La primera línia gruixuda del diagrama és un exemple de bifurcació (*fork*); cada bifurcació té un flux d'entrada i diversos fluxos de sortida.

L'element contrari a la bifurcació és la unió (*join*). Aquest element (l'altra barra gruixuda) té diversos fluxos d'entrada i un de sortida que, a diferència de la fusió, no s'executa fins que no s'han executat tots els fluxos d'entrada.

Així doncs, en el nostre exemple, no es confirma la matrícula fins que no s'ha validat la compatibilitat amb el pla d'estudis i s'ha validat el pagament.

2.2.3. Organització: carrils

Finalment, hi ha un element de notació que ens pot ajudar a organitzar les activitats. Els carrils (*swimlanes*) es fan servir, típicament, per a indicar quines activitats fa cadascun dels actors del procés (o, en el nostre cas, del cas d'ús) que estem documentant. S'indiquen, gràficament, de la manera següent:



En aquest cas, veiem que l'alumne és el responsable de fer la proposta de matrícula, el tutor de validar-ne la compatibilitat amb el pla d'estudis i el departament d'administració de validar-ne el pagament i confirmar-la.

2.2.4. Altres elements de notació

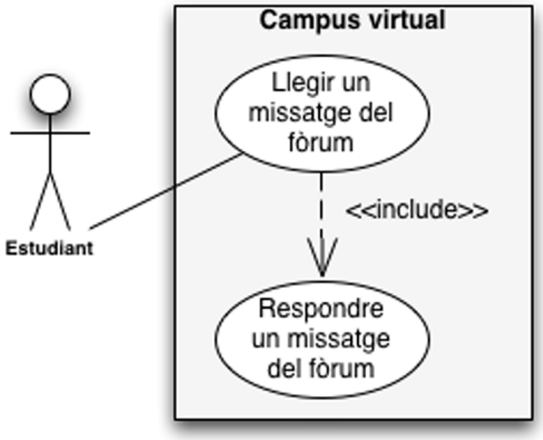
El diagrama d'activitats suporta altres elements de notació (com, per exemple, els senyals o les subactivitats) que no veurem en aquest mòdul, ja que serveixen per a modelitzar comportaments més complexos que els que trobem típicament en un cas d'ús.

2.3. El model resultant

El model de casos d'ús resultant ha de contenir una especificació basada en casos d'ús especificats textualment, tal com es va explicar al mòdul "Requisits". Aquesta es complementarà amb els diagrames UML que siguin oportuns per a ajudar a comprendre el model.

A continuació veurem un exemple del model de casos d'ús resultant per a l'exemple de les universitats. Per brevetat, ens centrarem, però, en només dos casos d'ús del sistema.

El diagrama de casos d'ús mostra visualment els actors, els casos d'ús, les relacions de generalització/especialització entre casos d'ús, quins actors intervenen en cada cas d'ús i, si n'hi ha, les relacions entre casos d'ús. En el nostre reduït exemple, mostraria els dos casos d'ús:



Com s'ha comentat, però, els diagrames de casos d'ús només complementen l'especificació textual d'aquests, que també ha de formar part del model de casos d'ús.

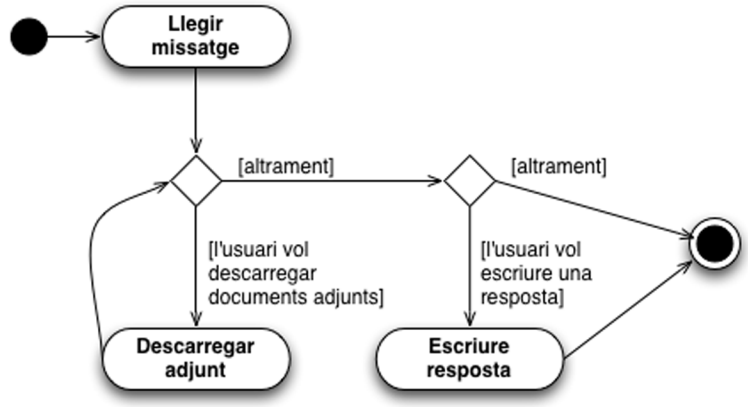
Cas d'ús: llegir un missatge del fòrum
Actor principal: usuari
Àmbit: campus virtual
Nivell d'objectiu: usuari
***Stakeholders* i interessos**:
Usuari: vol llegir el missatge
Professor responsable de l'aula: vol llegir el missatge i saber quins estudiants l'han llegit
Precondició: l'usuari s'ha d'haver identificat al sistema
Garanties mínimes: el sistema enregistrarà l'intent de lectura del missatge
Garanties en cas d'èxit: el sistema mostrarà a l'usuari el contingut del missatge i n'enregistrarà la lectura.
Escenari principal d'èxit:
1. L'usuari indica quin missatge vol llegir.
2. El sistema enregistra l'intent de lectura del missatge per part de l'usuari.
3. El sistema valida que l'usuari tingui accés al fòrum.
4. El sistema mostra el tema i el contingut del missatge.
5. El sistema enregistra que l'usuari ha llegit el missatge.

Extensions:
1a. L'usuari tanca l'ordinador
1a1. El sistema enregistra la lectura del missatge encara que l'usuari potser no l'hagi vist.
5a. L'usuari vol escriure una resposta a un missatge
L'usuari respon al missatge al fòrum
5b. La base de dades no està disponible (error de servidor)
5b1. El sistema indica a l'usuari que no ha estat possible recuperar el missatge i li demana que ho torni a provar passada una estona.
5c. L'usuari vol baixar documents adjunts
5c1. L'usuari indica quin document adjunt vol baixar.
5c2. El sistema baixa el document adjunt a l'ordinador de l'usuari.
5c3. Si l'usuari vol baixar més documents, tornem al pas 5c1.

Cas d'ús: respondre un missatge del fòrum
Actor principal: usuari
Àmbit: campus virtual
Nivell d'objectiu: usuari
***Stakeholders* i interessos**:
Usuari: vol respondre un missatge que ha llegit
Professor responsable de l'aula: vol fer el seguiment dels debats que es produeixen a la seva aula
Precondició: l'usuari s'ha d'haver identificat al sistema
Garanties mínimes: el sistema enregistrarà la resposta al missatge
Garanties en cas d'èxit: el sistema enregistrarà la resposta al missatge
Escenari principal d'èxit:
1. L'usuari indica que vol escriure una resposta a un missatge.
2. El sistema demana el tema i el contingut de la resposta, suggerint RE:<Tema del missatge al qual responem> com a tema del missatge
3. L'usuari modifica el tema si vol i introdueix el contingut de la resposta.
4. El sistema enregistra la resposta amb el tema i contingut introduïts, assigna la data actual com a data de publicació del missatge i l'associa al missatge original.

Extensions:
3a. L'usuari introdueix un tema o una resposta buits
3a1. El sistema indica que no es pot deixar buit el tema ni la resposta i tornem al punt 3.

En alguns casos, ens pot interessar documentar un o més casos d'ús vistos com a processos. Els diagrames d'activitats ens permetran mostrar visualment la seqüència de passos que fan els actors, la lògica condicional, el paral·lelisme, si n'hi ha, i, fins i tot, com diferents actors intervenen en un mateix cas d'ús (en forma de carrils). Al nostre exemple podríem documentar els dos casos d'ús mitjançant un diagrama d'activitats:



En realitat, però, només val la pena crear un diagrama d'activitats quan es vol documentar com a procés un cas d'ús (o un conjunt de casos d'ús) més complex que els d'aquest exemple. De fet, la majoria de casos d'ús de nivell d'usuari són prou simples per a no necessitar cap diagrama que els complementi. Altres casos d'ús, però, especialment alguns de nivell d'organització, poden ser més complexos i fer-se més comprensibles si els complementem amb un diagrama d'activitats.

3. Modelització de la interfície

El model de casos d'ús que hem vist fins ara se centra a recollir els objectius dels actors envers el sistema. Per aquest motiu, no hem inclòs, en cap moment, detalls sobre la interfície d'usuari (no volem que els detalls de la interfície d'usuari ens distreguin del que ens interessa: els objectius dels actors).

D'altra banda, tenim tota una sèrie de requisits, com ara la usabilitat o l'arquitectura d'informació que, en ser els casos d'ús independents de la interfície d'usuari, no queden recollits en aquest model.

Necessitem, per tant, un altre model que ens documenti, abans de dissenyar la interfície d'usuari, aquests requisits, i que ens ajudi a entendre com els diferents usuaris del sistema hi interactuaran.

Aquest model de la interfície d'usuari ens ha de donar indicacions sobre el següent:

- De quina manera el sistema presenta la informació als usuaris (mitjançant una interfície gràfica, un sistema de veu, etc.).
- Com navega l'usuari a través de la informació que li mostra el sistema per tal d'aconseguir els seus objectius.
- Com es relaciona el comportament indicat al model de casos d'ús amb la interfície d'usuari.

3.1. Casos d'ús concrets

S'anomenen **casos d'ús essencials** els casos d'ús que descriuen la interacció entre actors i sistema de manera independent de la tecnologia i de la implementació. En contraposició, anomenarem **casos d'ús concrets** aquells que tenen en compte la tecnologia i la implementació.

Així doncs, a partir d'un mateix model de casos d'ús essencials, és possible arribar a diversos models de casos d'ús concrets, segons quina sigui la tecnologia que utilitzem per a la interfície amb els usuaris i quin sigui l'intercanvi d'informació entre usuaris i sistema. En endavant, suposarem que el sistema per desenvolupar es comunica amb els seus usuaris mitjançant una interfície gràfica basada en pantalles.

Per a decidir el contingut de les diferents pantalles s'ha de tenir en compte l'opinió de diversos especialistes. A continuació en descrivim alguns:

- **Especialista en interacció**. La seva finalitat és aconseguir que els usuaris del sistema aconsegueixin complir els seus objectius. Per a fer-ho, estudia les diferents tasques per a aconseguir un objectiu concret (per exemple, què cal fer per a matricular un alumne a la universitat) i la manera en què es duen a terme.
- **Especialista en usabilitat**. Aplica el mètode científic a l'estudi i l'observació de la manera en què els usuaris fan servir el sistema per tal de fer que l'ús sigui còmode i intuïtiu.
- **Arquitecte d'informació**. S'encarrega d'organitzar la informació de manera que sigui fàcil de trobar i d'utilitzar. Per exemple, és l'encarregat d'assegurar-se que els estudiants poden accedir ràpidament al fòrum de la seva aula i que no han de navegar per un munt de pantalles abans d'arribar-hi.

Així doncs, podem escriure una versió concreta (sense tenir en compte les possibles extensions) del cas d'ús "Llegir un missatge del fòrum".

Cas d'ús: llegir un missatge del fòrum
Actor principal: usuari
Escenari principal d'èxit:
1. El sistema mostra la pantalla *Llista de missatges* amb els missatges de la carpeta *Rebuts* del fòrum.
2. L'usuari selecciona un missatge de la llista.
3. El sistema enregistra l'intent de lectura del missatge per part de l'usuari.
4. El sistema valida que l'usuari tingui accés al fòrum.
5. El sistema mostra la pantalla *Llegir missatge*.
6. El sistema enregistra que l'usuari ha llegit el missatge.

Extensions:
[...]
Pantalles:

Pantalla	Dades mostrades	Dades introduïdes
Llista de missatges	Llista de missatges. Per cada missatge: Marca llegit/no llegit Nom autor Data i hora Tema	Missatge seleccionat
Llegir missatge	Tema del missatge Contingut del missatge Autor Data	

3.2. Models de les pantalles

Una tècnica molt útil en aquesta fase és crear models de les pantalles. Els models de les pantalles ens ajuden a fer-nos una idea de com serà la interfície gràfica d'usuari final sense necessitat de preocupar-nos dels detalls concrets del disseny gràfic (colors, tipus de lletra, etc.).

Els models de les pantalles se centren en els conceptes per mostrar i en l'estructura de la pantalla més que no pas en com serà finalment. Per exemple, no es tenen compte els colors, ni els elements exactes d'interfície gràfica que es faran servir. Per això moltes vegades es fan, fins i tot, manualment (o simulant un dibuix a mà alçada).

Aquests models ens ajuden a explorar possibles models d'interacció i d'organització de la informació i també a detectar requisits sobre quina informació cal mostrar a cada pantalla.

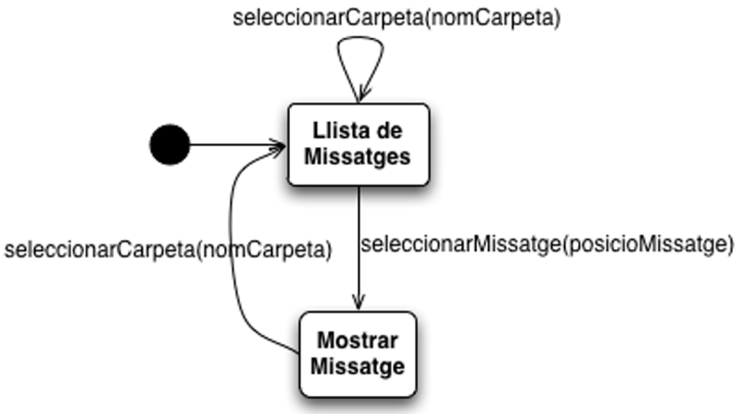
L'objectiu, doncs, dels models de les pantalles és deixar clar:

- Quina informació es mostra.
- La distribució de la informació a la pantalla.
- Quines accions pot prendre l'usuari a partir de la informació mostrada.
- Quin és el procés que segueix l'usuari per a completar el cas d'ús.

3.2.1. Diagrama d'estat de les pantalles (mapa navegacional)

El mapa navegacional és un model que ens dóna informació sobre quin és el flux entre pantalles que pot seguir l'usuari. Així doncs, la finalitat del mapa navegacional és donar una visió general de quines accions es poden fer a cada pantalla i en quins casos es passa d'una pantalla a una altra.

Per a fer el model del mapa navegacional podem fer servir una versió molt simplificada del diagrama d'estats UML:



El diagrama d'estats UML és molt similar al d'activitats i, per tant, és molt fàcil entendre'n la notació. Els estats representen les diferents pantalles (i el punt negre indica l'estat inicial) i les fletxes representen esdeveniments als quals el sistema ha de reaccionar.

Al diagrama de l'exemple, hi ha dues pantalles (*Llista de Missatges* i *Mostrar Missatge*) i tres esdeveniments:

- seleccionarCarpeta(nomCarpeta) a Llista de Missatges
- seleccionarMissatge(posicioMissatge) a Mostrar Missatge
- seleccionarCarpeta(nomCarpeta) a Llista de Missatges

Com podem veure, el mapa navegacional ens dóna una visió molt resumida de la interacció entre l'usuari i el sistema, ja que ens diu quina informació aporta l'usuari al sistema (el nom de la carpeta o la posició del missatge).

3.3. Contractes de les operacions del sistema

En el cas que necessitem elaborar una documentació formal dels requisits del sistema, els models vistos fins ara no seran suficients, ja que no formalitzen quin és el comportament del sistema davant de cada esdeveniment; només indiquen quina transició de pantalles es produeix.

Si necessitem aquest nivell de formalisme, podem fer servir operacions o esdeveniments de sistema. El que fem en aquest cas és identificar operacions a partir dels esdeveniments del mapa navegacional i escriure, en un llenguatge més o menys formal, un contracte que estableixi quin és el comportament del sistema quan es crida aquesta operació.

Un contracte consta de tres parts:

- **La signatura**. Ens diu el nom de l'operació, la informació que rep d'aquell que la crida (els paràmetres d'entrada) i la informació que es retorna a qui ha cridat l'operació (els paràmetres de sortida o el resultat).

- **Les precondicions**. Les obligacions que ha de complir aquell qui crida l'operació per tal que el sistema executi correctament l'operació. En el cas que aquestes precondicions no es compleixin, suposarem que el sistema rebutja l'esdeveniment i que no es produeix cap canvi.

- **Les postcondicions**. Les obligacions a què es compromet el sistema quan s'invoca l'operació. Són condicions que el sistema es compromet que siguin certes un cop executada l'operació.

Exemple de contracte

obtenirNombreMissatgesCarpeta(nomCarpeta:String):Integer

pre: el nom de la carpeta es correspon a una carpeta del fòrum

post: el resultat és el nombre de missatges que hi ha a la carpeta amb independència de si s'han llegit o no

Aquest senzill exemple ens permet veure com el contracte documenta:

- El nom de l'operació (obtenirNombreMissatgesCarpeta).
- La informació que aporta l'usuari (nomCarpeta de tipus String, és a dir, un text).
- La informació que retornarà el sistema (en aquest cas, un nombre enter).
- Les obligacions de l'usuari (donar un nom de carpeta que existeixi).
- Les obligacions del sistema (calcular el nombre de missatges de la carpeta i retornar-lo).

Com ja hem mencionat, podem escriure els contractes fent servir un llenguatge més formal, com ara OCL[1], el llenguatge estàndard de restriccions d'UML.

(1)*OCL* són les sigles d'*object constraint language*, en català, llenguatge de restriccions d'objectes.

Exemple de contracte en OCL

context System::obtenirNombreMissatgesCarpeta(nomCarpeta:String):Integer

pre: Carpeta.allInstances()->exists(c|c.nom=nomCarpeta)

post: result = Carpeta.allInstances()->select(c|c.nom=nomCarpeta).missatge->size()

4. Model del domini

Un model del domini és una representació de classes conceptuals del món real en un domini d'interès. Aquest model ha de servir a les persones que intervenen en el desenvolupament de programari per a entendre millor el domini del sistema que han de desenvolupar.

Per a documentar un model del domini en UML es faran servir un o més diagrames de classes, mitjançant els quals es representaran les classes conceptuals (o classes del domini), els seus atributs i les seves associacions.

4.1. Convencions en els diagrames UML

Abans de començar amb la descripció dels diagrames del model conceptual, establirem algunes convencions que farem servir pel que fa al nom de classes, atributs i associacions.

UML no posa cap restricció respecte de quins noms són vàlids i quins no però normalment preferim fer servir noms que més endavant, quan es passi a fer tasques de disseny i de programació, es puguin conservar. D'altra banda, seguir una determinada convenció de noms ajuda a donar coherència al conjunt d'artefactes que elaborem durant el desenvolupament de programari.

Per aquesta raó, en aquests materials seguirem les convencions següents:

- Farem servir noms en minúscules en general. Per a les classes, les associacions i els tipus dels atributs, la primera lletra estarà en majúscula, com si es tractés d'un nom propi.

  Exemple

  No farem servir noms de classe com *ALUMNE* o *alumne* sinó *Alumne*. Els noms d'associació seran com *Enregistra* o *EsMatricula*, i els dels tipus d'atribut seran com *Integer* o *Boolean*.

  Els noms d'atribut seran en minúscules, com ara nom o cognoms.

- No farem servir noms que continguin espais, sinó que separarem les paraules posant en majúscula la primera lletra de cada paraula.

  Exemple

  Com a nom d'atribut farem servir, per exemple, *dataNaixement* en lloc de *data de naixement*.

- No farem servir caràcters fora del rang d'ASCII, com ara els signes de puntuació, els accents, la ç, etc.

Exemple

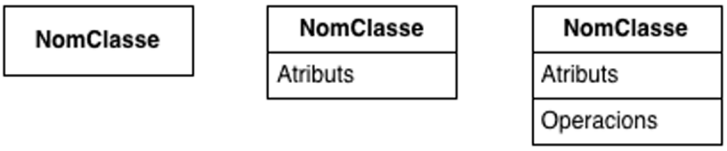
En lloc de *data d'inscripció* farem servir *dataInscripcio*.

4.2. Classes

4.2.1. Notació UML

En UML una classe s'indica mitjançant una caixa amb tres compartiments, dos dels quals són opcionals.

Representació de les classes en el llenguatge UML



El primer compartiment és per al nom de la classe i és l'únic compartiment obligatori. El segon compartiment conté els atributs de la classe i, tot i que és opcional, en els models del domini el farem servir. El tercer compartiment conté les operacions i també és opcional; en aquest cas, però, no el farem servir als models del domini.

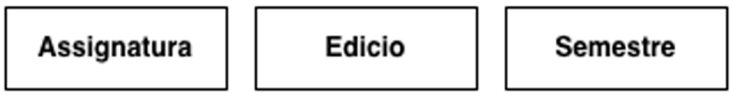
4.2.2. Tècniques de modelització

Identificació de classes del domini

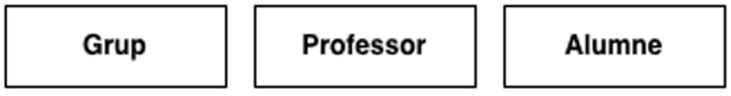
Per a identificar classes del domini cal fer una llista d'aquells conceptes de l'espai del problema que són rellevants per al sistema que s'analitza.

Exemple

En l'exemple de partida de la universitat, ens diuen que la universitat ofereix un seguit d'assignatures de les quals s'imparteix una edició cada semestre.

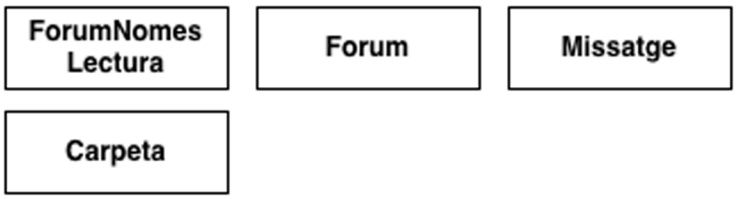


Sabem, també, que per a cada edició d'una assignatura es creen grups, cadascun dels quals té un professor i una sèrie d'alumnes matriculats.

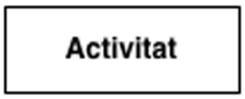


Alguns cursos són presencials i, per a aquests, en volem saber l'horari setmanal i, per a cada franja horària del curs, l'aula on s'imparteix. Les aules estan situades en edificis que, alhora, pertanyen a un determinat campus.

Hi ha un campus virtual on cada grup disposa d'un tauler i un fòrum que contenen missatges agrupats per carpetes.



Els professors proposen activitats.



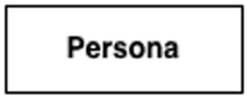
Una manera de completar la llista inicial de classes del domini és la tècnica, usada per diversos autors, de fer servir categories de classes conceptuals. Consisteix a repassar una llista de categories típiques de classes per a identificar classes d'aquella categoria que ens poden haver passat per alt.

Coad, Lefebvre i Luca (1999) proposen la llista de categories següent:

- **Participants, llocs i coses**. Classes que representen persones o organitzacions del domini (participants), llocs i coses. Els magatzems de què disposa una empresa, els productes que ven o les persones que hi treballen o que hi compren materials, per exemple, pertanyen a aquesta categoria.

Exemple

En l'exemple de les universitats, les persones que hi intervenen (estudiants, professors i qualsevol altre personal implicat) són exemples clars de participants.



D'altra banda, aules, edificis i campus són exemples clars de llocs.

- **Rols**. Classes que representen els rols que poden tenir les persones o organitzacions en les activitats que es desenvolupen en el domini analitzat. Els rols de comprador o venedor que poden tenir les persones en molts negocis són un bon exemple.

Exemple

En el nostre exemple els dos rols més importants són l'alumne i el professor, que ja havíem identificat.

- **Moments o intervals**. Classes que representen moments en el temps o intervals de temps, com ara un lloguer, una venda, etc. Tot sovint, aquests moments o intervals tenen un conjunt de parts.

Exemple

En l'exemple de les universitats, les diverses edicions d'una assignatura –ja identificades– són un exemple clar d'interval.

- **Descripcions**. Classes que representen la típica descripció de tipus catàleg d'una certa "cosa", típicament, un producte. Així, per exemple, distingim un cotxe concret, que té la seva matrícula i número de bastidor, del model de cotxe, que és una descripció de tipus catàleg associada a aquell cotxe concret.

Exemple

L'Assignatura es pot veure com una descripció de les diverses edicions que se'n fan, ja que ens indica el nombre de crèdits, que és el mateix per a totes les edicions d'una assignatura.

Larman (2005) proposa una llista de categories menys genèrica:

- **Transaccions, línies de transacció i productes o serveis**: les transaccions, típicament de negoci. Un exemple clàssic són les vendes d'un supermercat. Cada venda té un conjunt de línies de venda en què cada línia té un producte venut, un nombre d'unitats i un preu total de línia.

- **Esdeveniments importants**, sovint en una data o lloc que és rellevant, com per exemple un vol o un passi d'una pel·lícula.

- **Catàlegs i descripcions de coses**: les descripcions, en el mateix sentit que en parla Coad, es poden agrupar en catàlegs.

- **Contenidors físics o lògics i els elements que contenen**, com ara magatzems i els productes que hi desem o un tauler d'escacs i les caselles que conté.

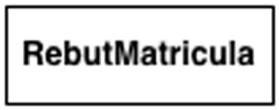
Exemple

En el nostre exemple, el campus i els edificis són contenidors físics d'altres llocs.

- **Sistemes externs** amb els quals interactuarà el sistema que estem analitzant.
- **Enregistraments** de treballs, contractes, afers legals, etc., com ara els rebuts.

Exemple

En el nostre exemple, els estudiants pagaran la seva matrícula i tindrem un rebut d'aquest pagament.



- Instruments financers com un xec, efectiu, etc.
- Agendes, manuals, documents, etc.

Exemple

En l'exemple, l'horari dels cursos presencials, que ja hem identificat com una classe, es pot veure com una agenda.

La llista de categories de Larman inclou, a més, un seguit de categories molt coincidents amb la llista de Coad: llocs, objectes físics i rols d'organitzacions i persones.

Una altra tècnica útil per a identificar classes del domini consisteix a reutilitzar models existents. Tot i que no podrem reutilitzar tal qual l'anàlisi feta per a un sistema existent, ja que si poguéssim fer-ho, probablement, no ens caldria desenvolupar un sistema a mida nou, sinó que podríem reutilitzar directament el sistema existent, sí que podem fer servir una anàlisi existent com a punt de partida per a identificar classes en el nostre model.

En aquest sentit, l'obra de Coad i altres (1999) proposa una sèrie de models genèrics que poden ser útils com a punt de partida, i Fowler (1997) proposa fer servir el concepte de patró per a documentar patrons d'anàlisi que podem aplicar adaptant-los al domini concret que estem analitzant.

Vegeu també

Podeu trobar més informació sobre els patrons aplicats a l'enginyeria del programari al mòdul "Introducció a l'enginyeria del programari".

Nomenclatura de les classes

A l'hora de donar nom a les classes del domini que s'identifiquen convé tenir en compte el que s'anomena la **regla del cartògraf**. Es tracta del següent:

- Fer servir la terminologia que fan servir les persones i experts del domini que s'està modelitzant, igual que un cartògraf fa servir la toponímia local al lloc que està cartografiant.

Exemple

A l'exemple hem fet servir un nom de classe força estrany, que no té res a veure amb el que fa servir la gent que treballa i estudia a la universitat d'exemple: ForumNomesLectura. Convé, doncs, que ens fixem en la nomenclatura que fa servir la gent que estudia i treballa a la universitat per a la qual desenvolupem el projecte. En aquest cas, dels fòrums on només algunes persones autoritzades poden escriure, en diuen *Taulers*.

- Excloure aspectes irrellevants del domini que estem analitzant igual que els cartògrafs fan una abstracció de la realitat i només representen aquells trets del terreny que interessen en un mapa.

Exemple

En el nostre cas, hem identificat una classe *RebutMatricula* que no és rellevant en el sistema que estem analitzant, ja que l'abast d'aquest sistema no inclou el pagament i l'elaboració de rebuts de les matrícules dels estudiants.

- No afegir res que no sigui realment al domini que analitzem.

Errada freqüent: classes de domini, no de programari

Cal tenir clar que l'objectiu del model del domini és documentar conceptes del món real, no classes de programari. Per tant, no tindrem classes que representin artefactes de programari com ara bases de dades, botons o finestres.

Per la mateixa raó, hi ha certs conceptes de l'orientació a objectes que no farem servir quan fem anàlisi. Així, per exemple, no assignarem a les classes d'aquest model responsabilitats de comportament i no els assignarem operacions. Tampoc no farem servir el concepte de visibilitat.

4.3. Atributs

4.3.1. Notació UML

Dins la caixa que representa una classe, al compartiment dels atributs, cada atribut s'indica textualment escrivint-ne el nom i, opcionalment, dos punts seguits del tipus de l'atribut.

Exemples d'atributs

Els següents són alguns exemples d'atributs:

```
nom: String
dataNaixement: Data
alçada: Longitud
```

El llenguatge UML defineix un seguit de tipus primitius estàndard que podem fer servir per a indicar els tipus dels atributs. Però, en general, podem fer servir altres tipus d'atributs sempre que quedi prou clar, pel nom del tipus, a què ens referim.

Vegeu també

Al subapartat 4.3.2 es dóna una llista de tipus d'atributs típics que ens poden facilitar la identificació d'atributs en les classes del domini.

UML permet, opcionalment, indicar la multiplicitat d'un atribut afegint, després del tipus, i entre claudàtors, un indicador d'aquesta, que pot ser:

Multiplicitat d'un atribut

La multiplicitat d'un atribut indica quines cardinalitats són vàlides per a aquell atribut; és a dir, quina restricció hi ha respecte al nombre de valors que pot tenir un atribut.

- "0..1" per a indicar que un atribut és univaluat però opcional.
- "1" per a indicar que un atribut és univaluat i obligatori. Aquesta opció és la que considerarem per defecte i, per tant, no indicarem aquesta multiplicitat de manera explícita.
- "*" per a atributs multivaluats ("1..*" si com a mínim han de tenir un valor). Una manera equivalent és indicar "0..*", però com que "*" és més curt, farem servir aquesta segona manera.
- Altres multiplicitats, com per exemple "3..5" (entre 3 i 5 valors).

Exemples d'atributs tenint en compte la seva multiplicitat

Els següents són alguns exemples d'atributs tenint en compte la seva multiplicitat:

- nom: String
- dataNaixement: Data [0..1]
- telefon: Telefon[1..*]

En el primer cas, com que no hem indicat cap multiplicitat, entenem que la multiplicitat és 1; és a dir, que el nom és obligatori i univaluat.

4.3.2. Tècniques de modelització

Identificació d'atributs

A mesura que anem identificant classes de domini, els podem anar afegint atributs que indiquin la informació rellevant dels objectes d'aquella classe. Per a identificar de manera més completa els possibles atributs d'una classe pot ser interessant repassar una llista dels tipus d'atributs més freqüents:

- **Nombres**: sovint ens convé saber si es tracta de nombres naturals (enters positius, que anomenarem *Nat*), enters (*Int*), o nombres no enters en general (*Real*).
- **Textos**: *Strings* (cadenes de caràcters sense format) i textos amb format.
- **Booleans**: cert o fals, sí o no.
- **Informació temporal**: com ara Data (1 de gener de 1980), Hora (12.43), Moment (1 de gener de 1980 a les 12.43), Durada (63 minuts)

- **Quantitats**: com ara Import (per exemple 23 €), Longitud (com ara 23,5 m), Temperatura (15°), Massa, etc.

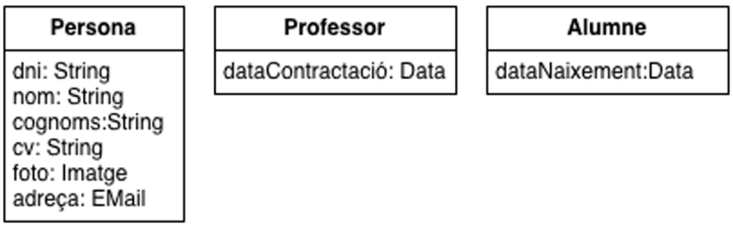
- **Altres valors sense entitat pròpia**: Color (com ara el color RGB 70,0,130), Coordenades (com ara 41°23'N 2°11'E), CodiPostal, NumeroTelefon, etc.

Exemple d'identificació d'atributs

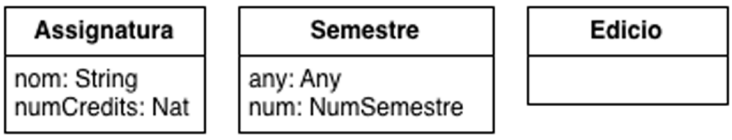
Després de parlar dels detalls amb els *stakeholders*, en el nostre exemple, hem identificat atributs a cada classe.

Les persones tenen nom, cognoms i DNI. Com que hi haurà un campus virtual, ens diuen que també en volem tenir un text amb un petit currículum i una foto, i també una adreça de correu electrònic de contacte.

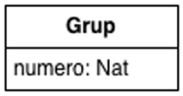
Dels professors, en volem saber també la data de contractació, i dels alumnes, la de naixement.



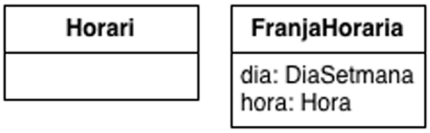
Les assignatures tenen un nom i un nombre de crèdits. Cada semestre té l'any i el número de semestre en què s'imparteix (primavera o tardor). De les edicions que es fan d'una assignatura cada semestre, de moment, no en tenim atributs.



Els grups que té cada curs tenen un número.



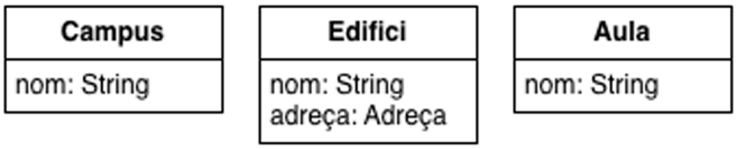
Dels horaris, no en tenim cap atribut, però de les franges horàries que els formen, sí. En un horari, una franja horària pot ser, per exemple, el dilluns a les 15 h o el dimecres a les 9 h. Suposarem que si en un horari hi ha dues hores seguides de classe, vol dir que hi ha dues franges horàries.



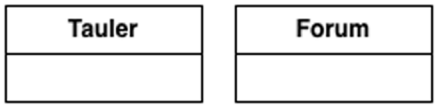
Les aules, edificis i campus tenen tots un nom (Aula C, edifici Shakespeare, campus Llull, per exemple). Dels edificis, volem saber-ne l'adreça postal.

Atributs o associacions

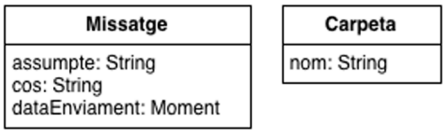
Una edició correspon a la impartició d'una assignatura en un semestre. Aleshores, per què no posem els atributs *assignatura: Assignatura* i *semestre: Semestre* a la classe *Edicio*? Ho veurem al subapartat 4.4.



De taulers i fòrums, no en tenim atributs.

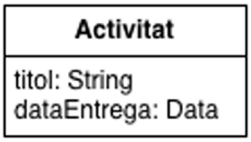


Els missatges tenen un assumpte, un cos i una data i hora d'enviament. No tenen destinatari perquè són missatges enviats a un tauler o un fòrum, no missatges de correu electrònic. Les carpetes en què s'organitzen els missatges tenen un nom.



Per a la data i hora d'enviament, en lloc de fer servir dos atributs separats, s'ha decidit fer servir un atribut que representi el moment exacte (data i hora) en què es va enviar el missatge.

Les activitats de les assignatures tindran un títol i una data d'entrega:

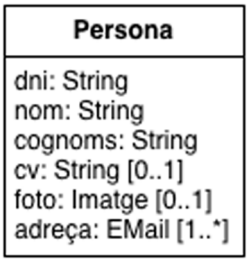


Atributs opcionals i atributs multivaluats

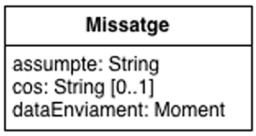
Per a la majoria d'atributs, caldrà decidir, tan sols, si l'atribut és **opcional** o no ho és. Però alguns atributs poden ser, a més, **multivaluats**.

Exemples d'atributs opcionals i multivaluats

Suposem, per exemple, que les persones poden no indicar el seu currículum i poden no tenir fotografia. I, a més, que poden tenir múltiples adreces de correu electrònic, però que com a mínim n'han de tenir una:



D'altra banda, podem considerar que el cos d'un missatge és opcional perquè amb l'assumpte, de vegades, n'hi ha prou.

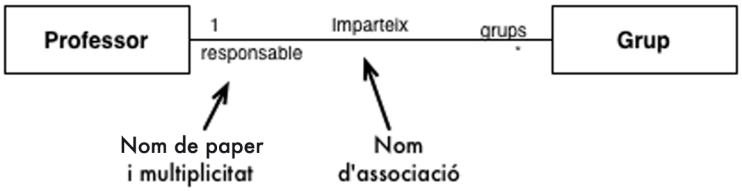


4.4. Associacions

4.4.1. Notació UML

Una associació, en UML, s'indica mitjançant una línia contínua entre les dues classes associades, i pot incloure un nom, noms de rol i multiplicitats, entre d'altres elements.

Notació UML de les associacions



Els noms de rol de l'associació indiquen el rol que té cada classe a l'associació i, per tant, és una manera de referir-se a les instàncies associades a una instància des d'aquesta. Així, a l'exemple de la figura, des de Grup, el professor associat s'anomena *responsable*, mentre que des de Professor, el conjunt de grups que té associats s'anomena *grups*.

La multiplicitat dels extrems d'associació s'indica fent servir la mateixa nomenclatura que la dels atributs. Tot i que UML considera, si no s'indica el contrari, que la multiplicitat per defecte és 1, en aquests materials sempre s'indicarà explícitament la multiplicitat de cada extrem d'associació.

4.4.2. Tècniques de modelització

Noms d'associació i de rols

Normalment fem servir noms d'associació que es puguin llegir formant una frase del tipus *NomClasse-NomAssociacio-NomClasse*, en què el nom de l'associació és un verb que permet formar una frase amb sentit.

Exemples de noms d'associació

Alguns exemples de noms d'associació i la frase que formen poden ser:

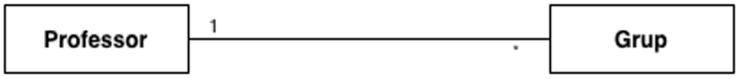
- Alumne EsMatriculaDe Assignatura
- Professor Imparteix Grup
- Estudiant Entrega Activitat

Per als noms dels rols d'associació, en canvi, fem servir noms semblants als dels atributs.

Tant el nom d'associació com els de rol són totalment opcionals i, de fet, indicar tant el nom d'associació com el de rol sol ser força redundant. Per aquesta raó moltes vegades no s'indica el nom de les associacions. En aquests materials, indicarem un nom de rol quan sigui especialment interessant indicar-lo.

En el cas de no indicar nom de rol, UML assumeix que el nom de rol és el nom de la classe, però començant en minúscula. En el nostre cas, a més, preferirem passar-lo a plural si aquell extrem de l'associació és multivaluat, ja que els noms de rol seran, així, més naturals.

Exemple d'associació amb els noms de rol implícits

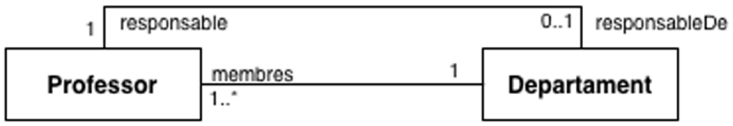


En aquest cas, els noms de rol serien *professor* i *grups*

Caldrà, però, indicar, com a mínim, un nom de rol, en aquells casos en què hi pugui haver confusió. Així, per exemple, serà necessari indicar com a mínim un nom de rol si tenim dues associacions entre les dues mateixes classes.

Exemple de dues associacions entre les mateixes classes

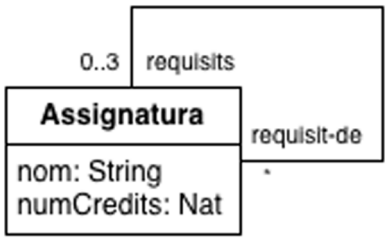
Imaginem que volem conèixer el departament al qual pertany un professor i quin professor és responsable de cada departament. En aquest cas ens cal fer servir els noms de rol per a distingir el professor responsable d'un departament d'aquells que en són membres.



En el cas de les associacions recursives, també és especialment important indicar clarament el rol de cada extrem d'associació.

Exemple d'associació recursiva

Suposem, per exemple, que una assignatura pot tenir requisits: altres assignatures que cal haver cursat abans de cursar l'assignatura en qüestió. Posem per cas que una assignatura pot tenir fins a 3 requisits però que pot ser requisit de qualsevol nombre d'altres assignatures.



Identificació d'associacions

A mesura que anem identificant classes del domini i poblant-les amb els seus atributs també podem anar identificant les associacions existents entre aquestes que, com sempre, siguin rellevants per al sistema que analitzem.

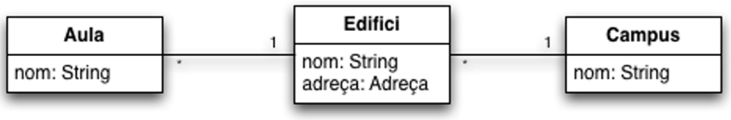
Identificarem una associació sempre que descobrim que a l'espai del problema les instàncies de dues classes poden estar associades, de tal manera que recordar l'associació entre aquestes és rellevant.

Per a identificar les associacions al model del domini un bon punt de partida és estudiar les classes ja identificades i plantejar quines poden ser les associacions entre aquestes.

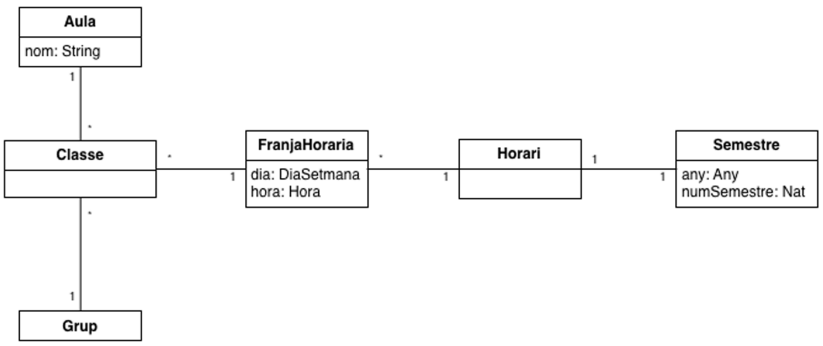
Exemple d'identificació d'associacions

Algunes associacions del nostre cas d'exemple es poden identificar d'entrada.

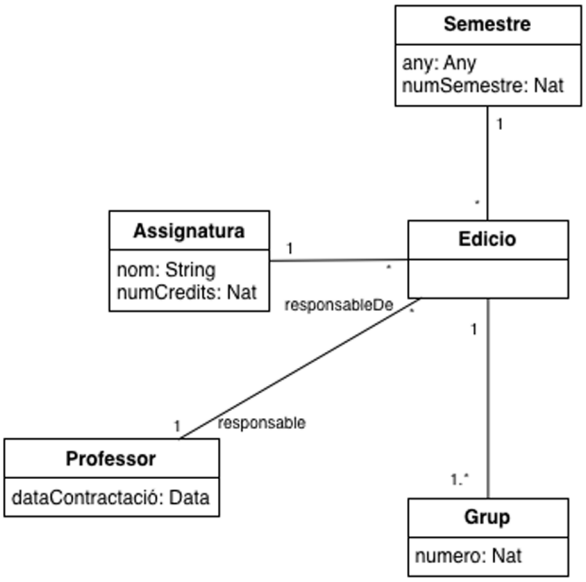
Cada aula és en un edifici que, alhora, és en un campus. Així, per exemple, podríem tenir una aula de nom *112*, situada en un edifici anomenat *B* que, al seu torn, està ubicat en un campus anomenat *Llull*.



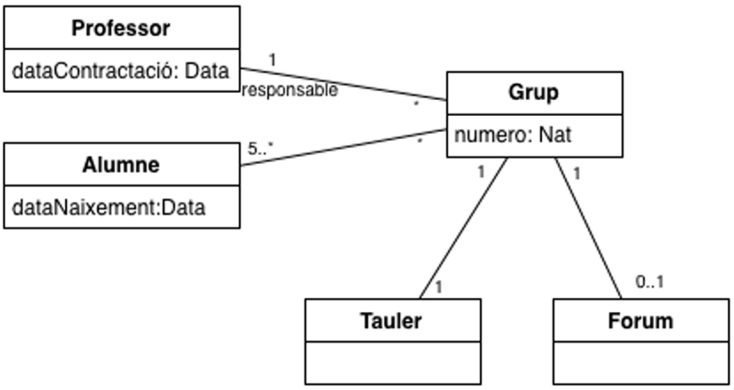
Cada semestre té un horari que està format per un conjunt de franges horàries. A cada franja horària es programen qualsevol nombre de classes. Així, per exemple, en un determinat horari, la franja horària del dilluns a les 9 h podria tenir programada una classe del grup 2 d'Enginyeria del Programari a l'aula 112 abans mencionada. Com que no l'havíem detectada abans, introduïm ara la classe *Classe*, que representa una d'aquestes classes programades i que té associada l'aula i grup que s'han programat.



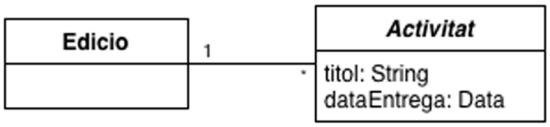
Cada edició d'una assignatura correspon a un semestre, té un professor responsable i un o més grups.



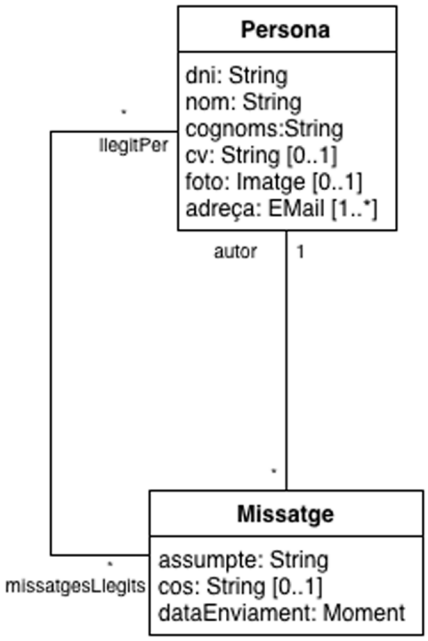
Cada grup té un professor i alguns alumnes. Suposem, per exemple, que en un grup hi ha d'haver, com a mínim, 5 alumnes. A cada grup assignem, també, un tauler i, opcionalment, un fòrum.



Cada edició d'una assignatura té una sèrie d'activitats:



De cada missatge, en voldrem saber l'autor i qui l'ha llegit i qui no:

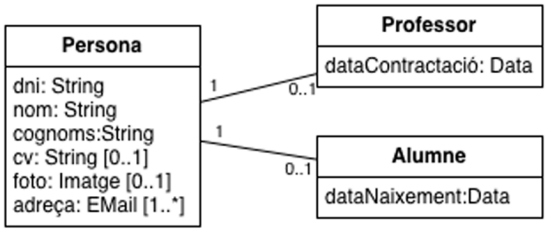


A partir d'aquí, es pot millorar la identificació fent servir, com ja s'ha mencionat en el cas de les classes, llistes de categories d'associació.

La llista de categories de classes del domini de Coad, Lefebvre i Luca (1999) proposa les associacions típiques següents:

- Els participants, coses o llocs poden tenir associada una descripció
- Cada participant, però també les coses o llocs, pot tenir associats qualsevol nombre de rols, en què cada instància de rol correspon a un únic participant.

  En el nostre exemple, les persones poden tenir el rol de Professor o d'Alumne (o tots dos). Una persona pot ser professor o no ser-ho i, per tant, la multiplicitat de l'associació de *Persona* amb el rol *Professor* tindrà multiplicitat 0..1. Igual passarà amb la classe *Alumne*.



- Els moments o intervals poden tenir associats diversos rols corresponents als participants, llocs i coses que hi intervenen (sota un rol determinat, com ara comprador o venedor).

Una altra llista de categories d'associació ens la proposa Larman (2005). Algunes de les categories més interessants d'aquesta llista són:

- Les transaccions poden tenir associades altres transaccions, com ara l'associació entre una venda i el seu pagament.

- Algunes transaccions tenen associades línies de transacció, com ara les línies de venda d'una venda d'un supermercat.

- Les transaccions o línies de transacció poden tenir associat un producte o descripció de producte, com ara l'associació entre la línia de venda del supermercat i la descripció del producte venut.

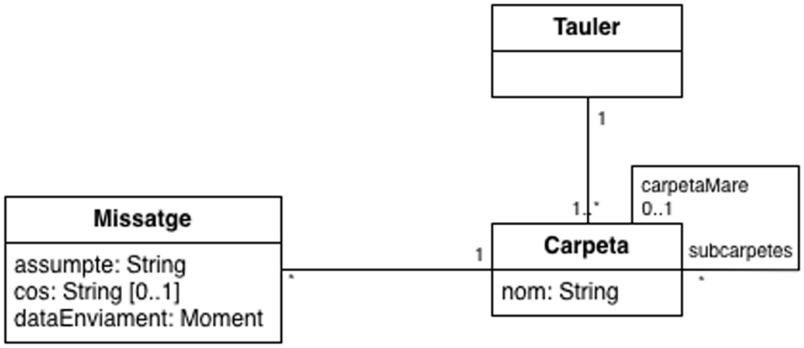
- Les transaccions poden tenir associats participants per mitjà dels seus rols.

- Algunes classes representen objectes que estan compostos lògica o físicament per objectes d'una altra classe, com els seients d'una sala de cinema o els pisos d'un edifici.

- Ens pot interessar la relació de contenidor-contingut entre instàncies, com és el cas d'un cotxe i la plaça d'aparcament on està aparcat.

Exemple

Aquest és el cas dels taulers. Cada tauler contindrà un conjunt de carpetes; com a mínim una, on aniran a parar els missatges per defecte. Aquestes no seran compartides, sinó que cada carpeta estarà al tauler on es va crear i només en aquell. Dins de cada carpeta hi haurà missatges; de nou, un missatge només podrà estar en una carpeta. Les carpetes poden, però, contenir altres carpetes. Per tant, cada carpeta té una carpeta mare (tret de les carpetes arrel) i pot tenir qualsevol nombre de subcarpetes.



Els fòrums funcionaran de manera semblant als taulers.

- Si tenim productes però també descripcions de producte, tindrem una associació entre aquests, com la que hi pot haver entre una edició d'una assignatura i la seva assignatura, que ja hem identificat prèviament.

Classes associatives

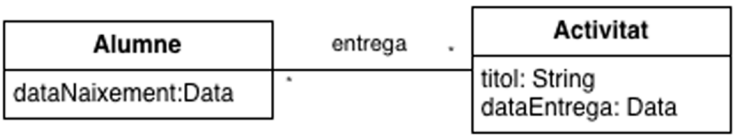
Les classes associatives permeten afegir atributs i associacions a les instàncies d'associació. La representació en UML d'una classe associativa és la d'una classe amb una línia discontínua sense fletxes que la connecta amb l'associació corresponent.

Vegeu també

Les classes associatives s'estudien al mòdul "Orientació a objectes".

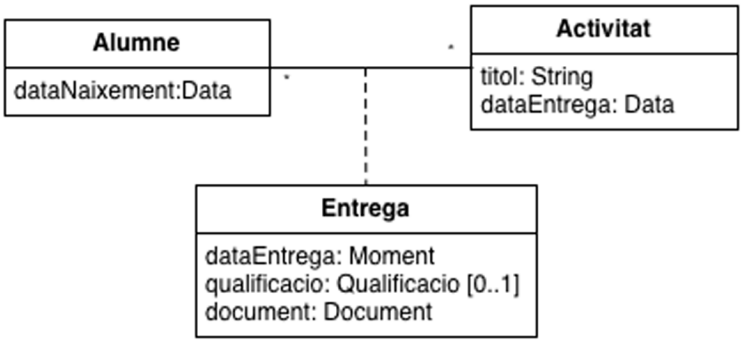
Exemple de classes associatives

Suposem, per exemple, que volem saber quins alumnes entreguen quines activitats.



Però, a més, volem fer un seguiment de la data i hora d'entrega, de les qualificacions que es posa a cada alumne de cada activitat i tenir el document entregat per l'alumne; és a dir, de cada entrega, volem tenir més informació en forma d'atributs. Una manera de representar aquesta informació és afegint una classe associativa per a les entregues.

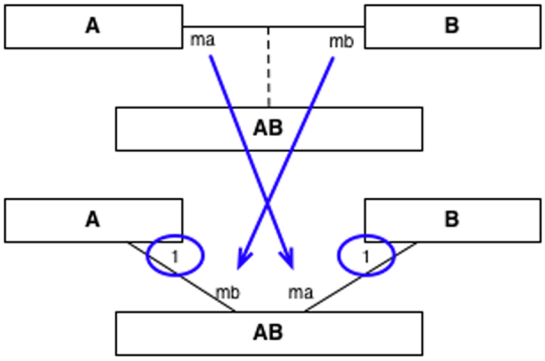
Representació de les entregues com una classe associativa



Hem indicat que la qualificació és de tipus Qualificació perquè no volem entrar en detalls encara sobre si és un nombre del 0 al 10, un nombre del 0 al 100, una lletra (*A, B, C, D*), etc. D'altra banda, hem indicat la qualificació com a opcional, perquè només aquelles entregues ja corregides tindran qualificació.

Alguns autors prefereixen no usar classes associatives, ja que introdueixen més conceptes i símbols gràfics dels estrictament necessaris.

Tota classe associativa es pot representar, de fet, com una classe normal que tingui, al seu torn, dues associacions amb les dues classes que associava.

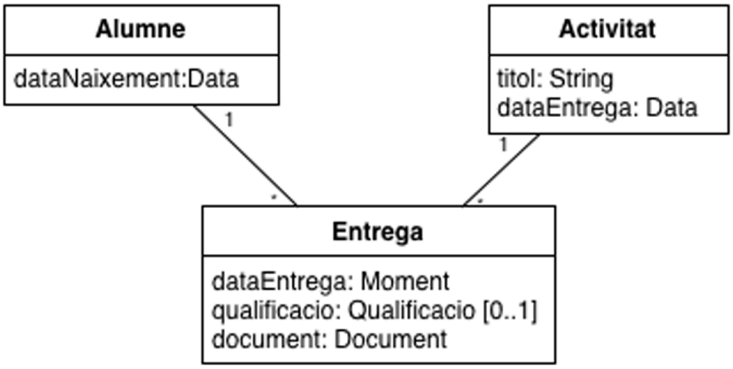


Les multiplicitats als dos extrems d'associació més allunyats de la classe originalment associativa sempre tenen multiplicitat 1, ja que, per definició, cada instància de la classe AB associa una i només una instància de la classe A i una i només una de la B.

Les multiplicitats als extrems d'associació propers a la classe AB són les que hi havia a l'associació original: cada instància d'A té associades *mb* instàncies de B i, per a cada associació, tindrà una instància d'AB; per definició, doncs, cada instància d'A tindrà associades *mb* instàncies d'AB.

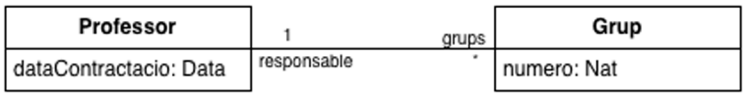
Representació de les entregues com una classe no associativa amb dues associacions

En el nostre exemple, les entregues es poden representar també sense classes associatives.



Associacions o atributs de tipus de classes del domini

Tant les associacions com els atributs acaben definint propietats de les classes. Formalment, en UML, una classe té un conjunt de propietats format pels atributs de la classe i els extrems d'associació de les associacions de la classe. Prenguem l'exemple d'associació entre un professor i els grups que imparteix:



Noms de rol d'associació identificatius

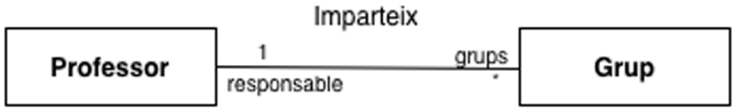
Els noms de les propietats de les classes han de ser únics. Per tant, de la mateixa manera que no podem tenir dos atributs d'una mateixa classe que es diguin igual, tampoc no podem tenir dos noms de rol en dues associacions d'una mateixa classe que defineixin propietats amb el mateix nom o un nom de rol que defineixi una propietat amb el mateix nom que un atribut.

Aquesta associació defineix dues propietats noves que no són atributs:

- La classe *Grup*, a més de la propietat *numero* de tipus Nat, té una propietat *responsable* de tipus Professor amb multiplicitat 1.
- La classe *Professor*, a més de la propietat *dataContractacio* de tipus Data, té una propietat *grups* de tipus Grup amb multiplicitat *.

Per tant, en realitat, els dos models següents són molt semblants:

Representació més adequada: en forma d'associació



Representació menys adequada: en forma d'atributs

La diferència està en el fet que la primera representació és més visual i ens indica que les propietats *grups* i *responsable* són inverses. És a dir, que si canviem el responsable d'un grup, els grups d'aquest responsable i del que hi havia abans hauran canviat.

Per tant, per a relacionar classes del domini preferirem sempre associacions i no atributs.

Errada freqüent: ús de "claus foranes"

Un cas particular i molt típic d'ús d'atributs quan caldria fer servir associacions és el de les claus foranes. Quan es dissenya una base de dades, les associacions s'emmagatzemen en forma d'atributs que contenen identificadors de registres de la taula relacionada.

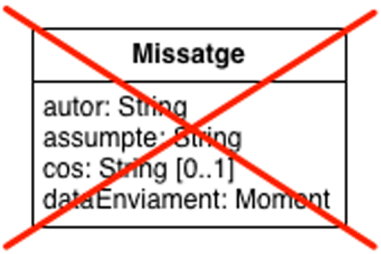
A causa d'això, alguns analistes fan servir atributs semblants en els seus models UML, ja sigui a més de l'associació corresponent o en lloc d'aquesta.

Com hem vist anteriorment, tots dos casos són un error, ja que cal representar l'associació com a tal i no com a atribut.

Exemple

Si, per exemple, haguéssim dit que l'autor d'un missatge és un atribut de tipus String que conté el DNI de l'autor, estaríem cometent aquest error:

Representació incorrecta d'una associació fent servir claus foranes

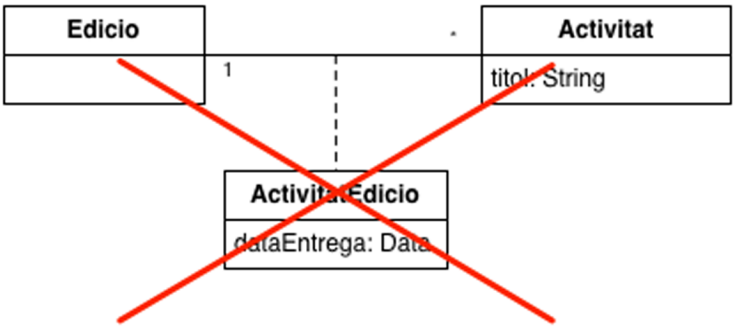


Errada freqüent: Classes d'associació amb atributs erronis

Un error habitual és fer servir classes associatives per a afegir atributs a una associació que, en realitat, pertanyen a una de les classes associades. Això passa, especialment, en associacions amb una multiplicitat d'1 en un dels extrems.

Exemple

Un exemple d'aquest error seria pensar que la data d'entrega d'una activitat és un atribut de l'associació entre Activitat i Edicio. Com que cada activitat només es fa en una edició, aquesta data depèn de l'activitat però no de l'edició.



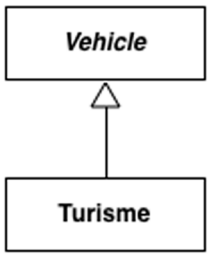
4.5. Herència

4.5.1. Notació UML

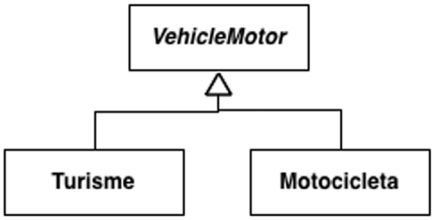
La notació UML de l'herència, que UML anomena *generalització*, és una línia contínua des de la subclasse cap a la superclasse amb un triangle a la punta.

Exemple

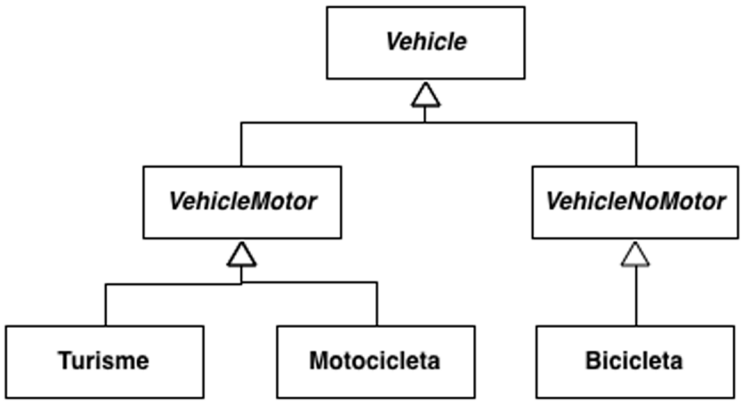
Exemple de notació UML de la generalització:



En el cas que una classe tingui més d'una subclasse, solem agrupar les línies de generalització en una única línia.



En UML, les classes abstractes (aquelles en què totes les instàncies han de ser instància d'alguna de les seves subclasses) s'indiquen posant-ne el nom en cursiva.



4.5.2. Tècniques de modelització

Identificació de l'herència: generalització i especialització

Els processos que ens permeten identificar l'herència són la generalització i l'especialització.

Vegeu també

Els processos de generalització i especialització s'estudien al mòdul "Orientació a objectes".

En un moment donat, examinant una determinada classe, podem detectar una possible subclasse per **especialització** quan ens trobem algun dels casos següents:

- La subclasse candidata té atributs addicionals d'interès (que no tenen totes les instàncies de la classe examinada).

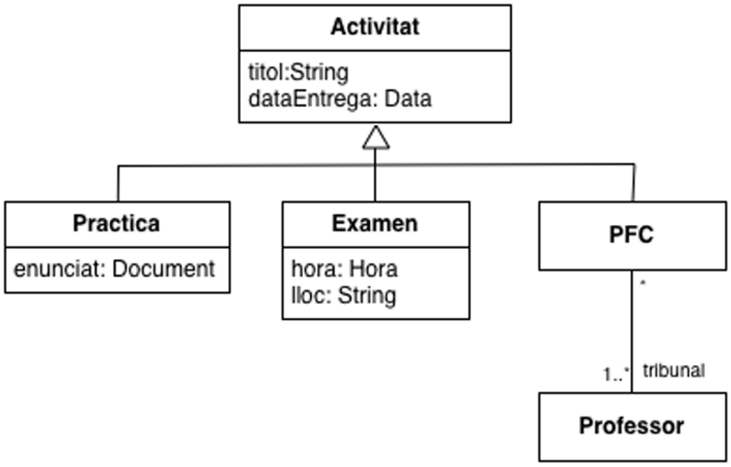
- La subclasse candidata té associacions addicionals d'interès (que no tenen totes les instàncies de la classe examinada).

- La subclasse candidata es comporta de manera diferent en un sentit rellevant. Pot ser que es faci servir o s'hi operi de manera diferent o bé que representi una persona, animal o sistema que té un comportament propi diferent.

Exemple d'especialització

En el nostre cas, ens podem fixar en les activitats. Suposem que tenim activitats que són pràctiques (s'han d'entregar al campus virtual), d'altres que són exàmens presencials i, finalment, projectes de final de carrera.

A part de la informació comuna a tota activitat, de les pràctiques, com que es fan pel campus virtual, en voldrem tenir l'enunciat. Dels exàmens ens caldrà desar l'hora i el lloc. Els projectes de final de carrera (que es poden considerar l'única activitat de cada edició de l'assignatura de projecte de final de carrera), tenen un tribunal format per un o més professors. Per tant, podem especialitzar les activitats de la manera següent:

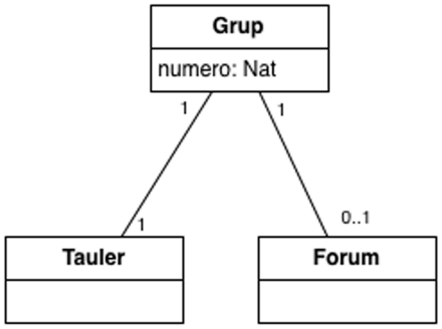


Pel que fa a la **generalització**, podem detectar-ne una oportunitat si es dóna algun dels casos següents:

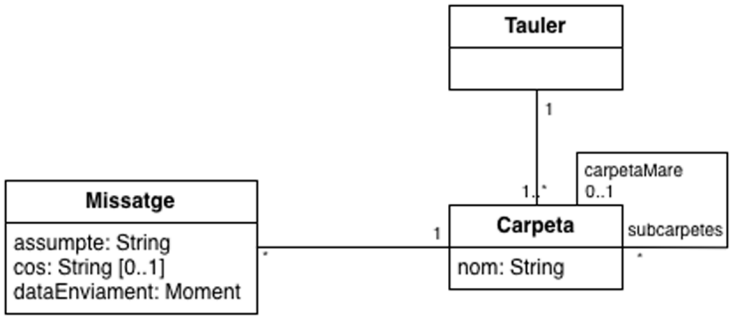
- Les subclasses potencials representen variacions d'un mateix concepte.
- Les subclasses potencials tenen un o més atributs en comú que es poden expressar com a atributs de la nova superclasse.
- Les subclasses potencials tenen una o més associacions en comú que es poden representar com a associacions de la nova superclasse.

Exemple de generalització

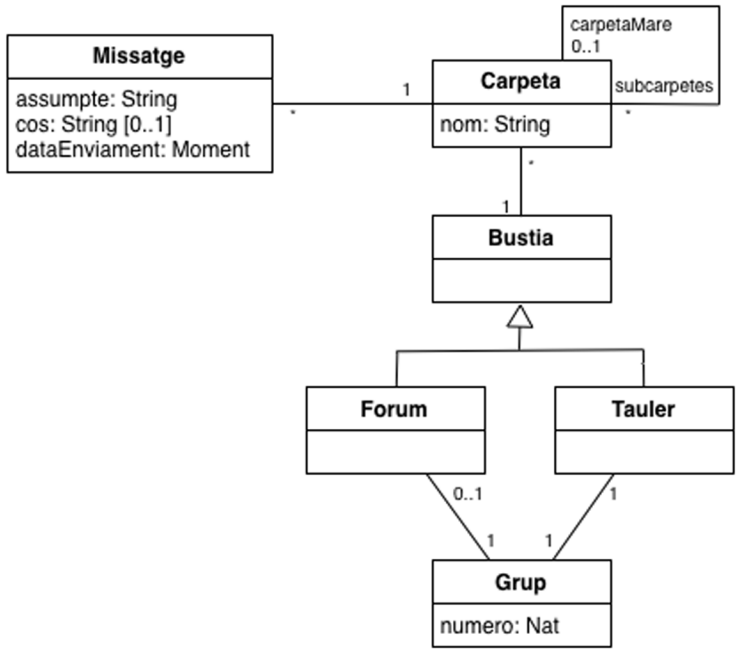
Seguint amb el nostre exemple, fins ara hem identificat que cada grup d'un curs té un tauler i, opcionalment, un fòrum:



També hem vist que el tauler conté missatges organitzats en carpetes:

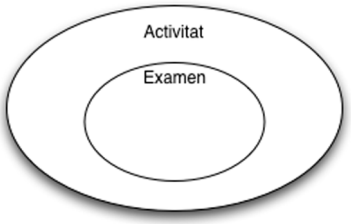


Però resulta que els fòrums també tenen missatges organitzats en carpetes. Per tant, fòrums i taulers són bústies de missatges amb algunes característiques particulars de funcionament i algunes associacions particulars (el tauler és obligatori en un curs, però el fòrum no ho és) i podem fer una generalització per a abstraure les parts en comú a tots dos:



Errada freqüent: Herències falses

La relació d'herència es pot estudiar sota la teoria de conjunts com la pertinença de determinades instàncies a determinats conjunts. Si representem cada classe com un conjunt de les instàncies d'aquella classe, la relació d'herència entre Activitat i Examen, per exemple, es pot representar de la manera següent:

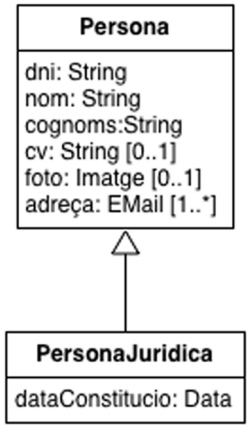


Per tant, tota instància d'*examen* ha de ser també una instància d'*activitat* i, tot el que hem definit de les activitats (associacions i atributs) ha de ser 100% cert per a *exàmens*, també.

Cal evitar, doncs, les falses herències, en què alguna instància d'una pretesa subclasse no ho és de la superclasse o en què algun atribut o associació d'una pretesa superclasse no és aplicable a totes les seves subclasses.

Exemple

Suposem, per exemple, que ens interessa modelitzar les persones jurídiques. Com que tenim ja una classe *persona*, tot sembla indicar, *a priori*, que una persona jurídica serà una subclasse de *persona*:



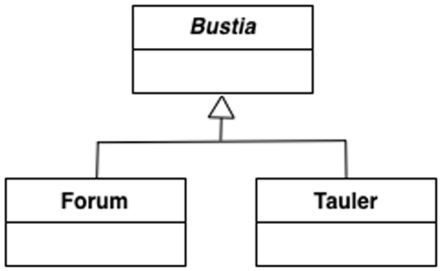
El problema és que les persones jurídiques no tenen cognoms, CV, ni foto, i en lloc de DNI tenen un CIF. Per tant, en realitat, no és cert que tot allò que podem dir de les persones també sigui cert de les persones jurídiques.

Distinció entre classes abstractes i classes concretes

Quan modelitzem una herència, cal plantejar-se si tota instància de la superclasse ha de ser, forçosament, instància d'alguna subclasse o, al contrari, pot ser que alguna instància de la superclasse no ho sigui de cap subclasse. En el primer cas, la superclasse es definirà com a abstracta.

Exemple de classe abstracta

En el cas de la generalització de taulers i fòrums en bústia de missatges, ens trobem que no tindrem mai una bústia de missatges que no sigui un tauler o un fòrum, ja que estem suposant que el sistema que modelitzem no ofereix bústies de cap altra mena. Per tant, caldria indicar que la classe *Bustia* és abstracta posant-ne el nom en cursiva.

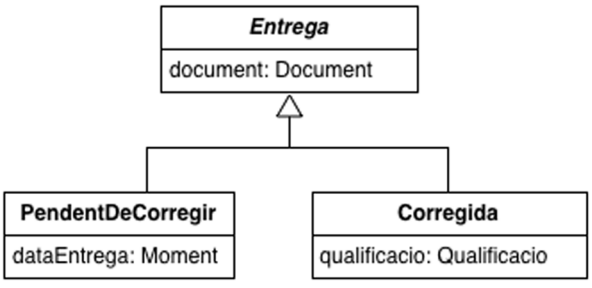


De manera semblant, la classe *Activitat* també serà abstracta.

Modelització de l'estat dels objectes

Molt sovint ens trobem amb una classe els objectes de la qual poden estar en diversos estats. Així, per exemple, les entregues d'activitats per part dels alumnes es poden considerar pendents, fetes i corregides. Per a les entregues ja fetes ens pot ser d'interès la data en què es va entregar, mentre que per a les ja corregides ens interessarà la qualificació atorgada.

Una manera de representar aquests possibles estats és fer servir l'herència:



Amb aquest model, estem dient que les instàncies d'entrega, quan es creen, són instàncies de PendentDeCorregir que, un cop corregides passen a ser instàncies de Corregida. Aquesta herència és, per tant, dinàmica.

Diem que una herència és dinàmica quan una instància de la superclasse, al llarg de la seva vida, pot canviar de subclasse.

Forma més natural d'entendre l'herència

L'herència és un concepte de l'orientació a objectes i, per tant, és originari dels llenguatges de programació. Atès que gairebé cap llenguatge de programació no permet que una instància canviï de classe al llarg de la seva vida, la manera més natural d'entendre l'herència és com a herència estàtica.

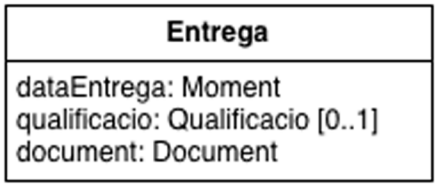
Aquest enfocament té l'inconvenient que el concepte d'herència dinàmica és poc natural i, per tant, poc entenedor.

Un altre problema habitual en aquest ús de l'herència és que pot induir a errors respecte de quins atributs són rellevants en cada estat. El model anterior, per exemple, sembla prou natural, però hem comès l'error de considerar que per a les entregues corregides no ens interessa saber la data en què es van entregar.

Una manera alternativa de representar aquesta situació és fer servir un atribut que indiqui l'estat de l'objecte (discriminador) i fer opcionals aquells atributs que només prenen valor en determinats estats, tal com ja havíem fet.

Exemple

Representació de la classe *Entrega* sense reflectir, al diagrama de classes, els seus estats possibles:

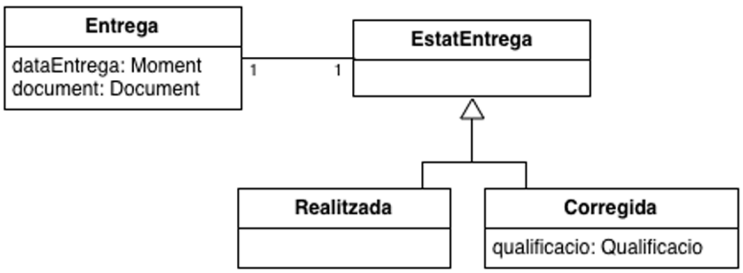


En aquest cas, l'atribut *qualificació* pot actuar com a discriminador ja que, per definició, una entrega corregida és aquella que té qualificació. Un altre cas habitual seria fer servir un atribut booleà (*corregida: boolean*) o d'un tipus específic per a casos amb més de dos estats (*estatEntrega: EstatEntrega*).

Una altra solució consistiria a afegir una classe que representi l'estat. En aquest cas caldrà anar amb compte de no cometre el mateix error respecte de l'atribut *dataEntrega*:

Exemple

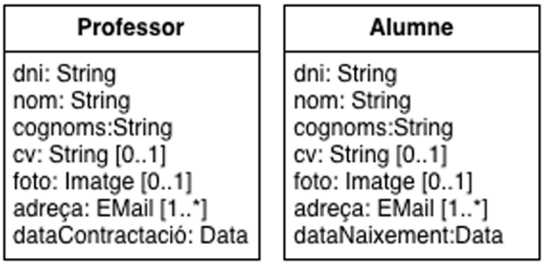
Representació de la classe *Entrega* reflectint els seus possibles estats mitjançant una classe associada:



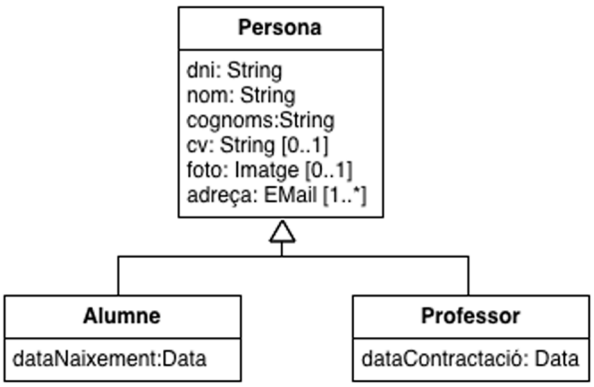
Modelització del rol d'objectes i persones

Una situació habitual és que un objecte, especialment si representa una persona del món real, pugui tenir un rol determinat dins el domini analitzat. És el cas, per exemple, dels professors i alumnes d'una universitat.

Suposem, per exemple, que s'han identificat professors i alumnes de la facultat com a dues classes d'objectes:



En aquesta situació, tal com s'ha explicat, trobem una oportunitat molt clara de generalitzar les parts comunes en una classe *Persona*:



Aquesta solució, però, planteja el problema ja plantejat anteriorment de les herències dinàmiques. Una mateixa persona podria passar d'alumne a professor i, per tant, l'herència es pot considerar dinàmica.

Però, a més, presenta una problemàtica addicional: suposem que una persona pot ser, alhora, Alumne i Professor (per exemple, perquè és alumne d'uns estudis i professor d'uns altres). En tal cas, un mateix objecte de classe *Persona* hauria de pertànyer a totes dues subclasses alhora.

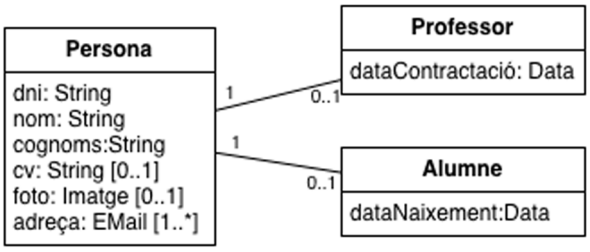
Diem que una herència és *overlapping* quan una mateixa instància de la superclasse pot pertànyer a més d'una subclasse alhora.

Per raons semblants al cas de les herències dinàmiques, considerem que les herències *overlapping* no són naturals; són confuses, difícils d'entendre i, per tant, no considerem el seu ús gaire recomanable.

Un solució millor consisteix a modelitzar els rols com a classes pròpies associades a la classe que pot tenir aquests rols. És, de fet, el que proposen Coad, Lefebvre i Luca (1999) quan ens proposen els rols com una categoria de classes a l'hora d'identificar-les, tal com s'ha explicat al subapartat 4.2.2.

Exemple

Modelització dels possibles rols de les persones al nostre sistema com a classes associades. En aquest cas, una persona pot tenir el rol de Professor, el d'Alumne, tots dos, o cap dels dos.



4.6. Informació derivada i regles d'integritat

4.6.1. Regles d'integritat

Una de les funcionalitats bàsiques del programari per a sistemes d'informació consisteix a desar informació de manera persistent que permeti fer consultes i guiar la presa de decisions en el dia a dia de l'empresa, organització o persona que fa servir el sistema d'informació.

Exemple

El programari per a la universitat, per exemple, permetrà fer consultes dels expedients dels alumnes i permetrà guiar l'avaluació dels estudiants, ja que, per exemple, pot proporcionar al professor, a l'hora d'introduir les qualificacions, una llista dels estudiants matriculats al seu grup.

Perquè aquesta informació sigui útil serà important mantenir-ne la integritat: caldrà evitar situacions en què la informació és contradictòria o incompleta.

En el cas que ens ocupa serà important evitar, per exemple, situacions en què es qualifica una entrega d'una activitat a un alumne que no està matriculat d'aquell curs, o situacions en què no es disposa de la qualificació d'una activitat que es va corregir.

El model del domini, per tant, ha de documentar quines restriccions d'integritat han de satisfer les dades que gestiona el sistema. Els diagrames de classes UML ens permeten representar gràficament algunes d'aquestes restriccions d'integritat. És el cas, per exemple, de les multiplicitats.

Exemple

En el fragment de diagrama mostrat, que és una part del que ja hem analitzat, hem documentat diverses restriccions d'integritat. Diem, per exemple, que un grup ha de tenir com a mínim 5 alumnes i que, per tant, si es donés el cas que tenim un grup amb menys de 5 alumnes, el sistema ho hauria de considerar una situació errònia. De manera semblant, indiquem que cada grup té un i només un professor i que, per tant, no es pot donar el cas que un grup tingui dos professors o que no en tingui cap.

Però moltes altres restriccions d'integritat no poden ser representades gràficament en el llenguatge UML. Per a aquestes, caldrà redactar un document que les recopili.

Claus de les classes del domini

Un subconjunt especialment important de les restriccions d'integritat que caldrà documentar són les claus de les classes del domini. Gairebé cada classe del domini tindrà, típicament, un atribut o conjunt d'atributs que n'identifica les instàncies de manera única. Per tant, caldrà que el sistema garanteixi que no hi pot haver dues instàncies d'una mateixa classe en què la clau sigui la mateixa.

Exemple

En el cas que ens ocupa, si, per exemple, el sistema detectés que s'està donant d'alta una persona amb el mateix DNI que una persona existent, caldria que indiqués que es tracta d'un error: o bé la persona ja s'ha donat d'alta al sistema amb anterioritat o bé hi ha una errada al DNI de la nova persona que s'està donant d'alta. En qualsevol dels dos casos, permetre l'alta de dues persones amb el mateix DNI voldria dir trencar la integritat de les dades.

Diem, doncs, que la clau de *persona* és el DNI. Al document on hi ha la llista de les claus de les classes del domini, indicarem, per tant:

Les classes del domini tindran les claus següents:

- Persona: dni
- Assignatura: nom
- Campus: nom

En el cas dels semestres, l'identificador serà l'any i el número de semestre, cosa que indicarem com:

- Semestre: any + num

Els edificis, en principi, s'identificaran per *nom*. Però hi podria haver dos edificis que es diguessin igual, sempre que estiguessin en campus diferents, ja que continuaria sense haver-hi confusió possible. Per tant, el nom de l'edifici l'identifica dins el campus. Podríem considerar, doncs, que la clau d'edifici és nom del campus + nom de l'edifici. Per a simplificar, però, direm que la clau d'edifici és campus + nom (és a dir, un cop identificat un campus, l'edifici s'identifica per *nom*). De manera semblant, la clau de la classe *Aula* estarà formada per l'edifici i el nom.

- Edifici: campus + nom
- Aula: edifici + nom

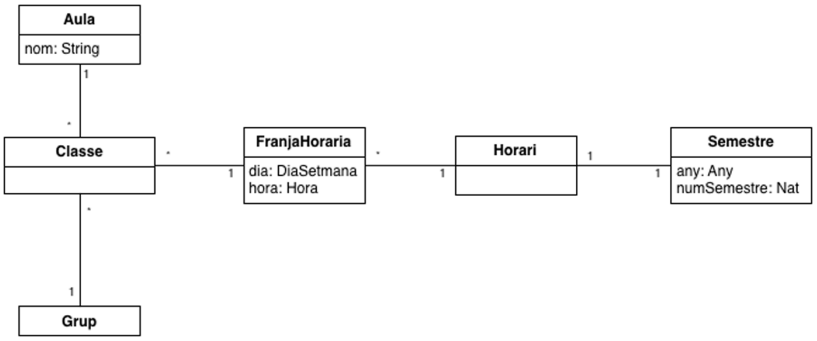
DNI com a clau

Tot i que el DNI sembla una clau òbvia per a identificar persones a l'Estat espanyol, en realitat, hi ha tota una problemàtica associada als nombres repetits que nosaltres no tindrem en compte però que ens podria afectar en sistemes reals.

Cada semestre té el seu propi horari. Per tant:

- Horari: semestre

Però el cas de les franges horàries mereix una atenció especial:



En un horari, la franja horària s'identificarà pel dia de la setmana i l'hora:

- FranjaHoraria: horari + dia + hora

Pel que fa a les classes, en una mateixa franja hi pot haver més d'una classe sempre que es compleixin una sèrie de condicions: no pot hi haver dues classes del mateix grup a la mateixa franja horària, i no hi pot haver dues classes a la mateixa aula i la mateixa franja horària. Per tant, en aquest cas, tenim dos identificadors:

- ClasseProgramada: franjaHoraria + aula, franjaHoraria + grup

Cada edició d'una assignatura es correspon a un semestre d'una assignatura. I, en una edició, els grups s'identifiquen per número. Per tant:

- Edicio: semestre + assignatura
- Grup: edicio + numero

Cada tauler té associat només un grup i és l'únic tauler del grup. Igual passa amb els fòrums, en cas que el grup en tingui. Per tant:

- Tauler: grup
- Forum: grup

Taulers i fòrums tenen una superclasse *Bustia*, però no tenim cap manera d'identificar les bústies per si mateixes. Per tant, la classe *Bustia* no té clau.

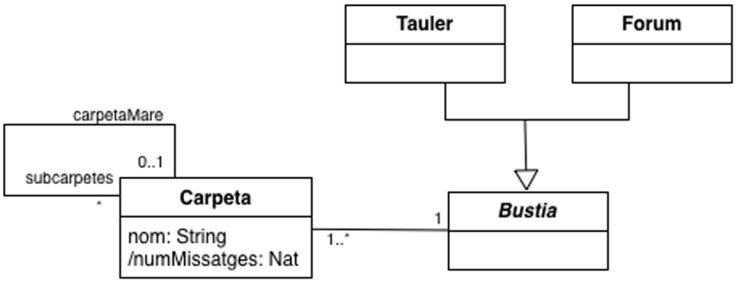
Les activitats d'una edició d'una assignatura han de ser identificades per *títol*. Òbviament, hi podria haver dues activitats amb el mateix títol sempre que siguin de diferents edicions d'assignatura.

- Activitat: edicio + titol

Cada instància de rol de professor o d'alumne es correspon a una única persona i és aquesta persona el que les identifica:

- Professor: persona
- Alumne: persona

El cas de les carpetes de les bústies també cal mirar-lo amb atenció:



És clar que no hi pot haver dues carpetes amb el mateix nom dins una mateixa carpeta. Per tant, sembla que la clau de carpeta ha de ser carpetaMare + nom. Però sí que hi pot haver dues carpetes amb el mateix nom que no tinguin mare, sempre que estiguin en bústies diferents. La clau és, doncs, bústia + nom? No, perquè sí que es poden tenir dues carpetes diferents, de la mateixa bústia, que tinguin el mateix nom, sempre que tinguin mares diferents.

En aquest cas, per tant, no podem trobar cap clau.

Finalment, pel que fa als missatges, podríem tenir dos missatges exactament iguals (amb el mateix assumpte, cos i moment d'enviament) dins la mateixa carpeta. Per tant l'única manera d'identificar un missatge és per la posició que ocupa dins la carpeta.

- Missatge: carpeta + posició a carpeta

Altres restriccions d'integritat

A part de les restriccions de clau, convé documentar de manera textual qualsevol altra restricció d'integritat que el llenguatge UML no ens permeti expressar gràficament.

Exemple

En el nostre exemple trobarem algunes restriccions que ens caldrà documentar:

- Per a tota activitat, la data d'entrega ha de ser una data dins el semestre de l'edició a la qual està associada l'activitat.
- Les carpetes formen una jerarquia vàlida de mares-subcarpetes (no es poden formar cicles tals que una carpeta té com a mare una filla, o una filla de la seva filla, etc.).
- No hi pot haver més d'una carpeta sense mare amb el mateix nom dins la mateixa bústia
- No hi pot haver més d'una carpeta amb la mateixa carpeta mare i el mateix nom.

4.6.2. Informació derivada

De vegades ens trobem amb atributs o associacions que poden ser d'interès però que, en realitat, són derivats a partir d'altres atributs i associacions; és a dir, que el seu valor es pot deduir a partir d'informació ja modelitzada.

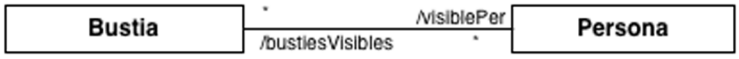
Exemple

Suposem, per exemple, que de les persones ens interessa saber la data de naixement i l'edat. Direm que l'edat és derivada perquè es pot calcular a partir de la data de naixement.

Un altre exemple d'atribut derivat podria ser el nombre de missatges en una carpeta. Tot i que encara no l'hem posat com a atribut de la classe *Carpeta*, sabem que és rellevant però que es pot derivar de l'associació entre Missatge i Carpeta.

Les associacions també poden ser derivades. Com a exemple, podem pensar en la llista de bústies visibles per una persona. Si la introduïm com a associació entre Persona i Bustia tindrem una associació que ens dóna informació rellevant però que és derivada, ja que una persona té visibles els fòrums i taulers dels grups en els quals participa com a professor o com a alumne.

UML permet indicar que un atribut és derivat posant una barra (/) davant el seu nom. De manera semblant, podem indicar que una associació és derivada posant una barra davant el seu nom o el nom dels rols.



De manera semblant al que passa amb les restriccions d'integritat, UML no ofereix una manera gràfica d'expressar com es deriva la informació derivada. Per tant, convé documentar-ho textualment.

En el nostre exemple, ens caldria documentar:

- *Carpeta::numMissatges* és el nombre de missatges associats a la carpeta
- *Persona::bustiesVisibles* és el conjunt format pel tauler i el fòrum (per a aquells que en tinguin) de tots els grups als quals la persona està associada com a professor o com a alumne.

4.7. Classes i atributs: elements avançats

4.7.1. Tipus de dades

Fins ara hem fet servir, als atributs, tipus com ara String, Nat, Data, però també quantitats o altres valors sense entitat pròpia com ara Adreça, NumSemestre o Qualificacio. Aquests tipus els anomenem *tipus de dades*.

Un tipus de dades és un conjunt de valors per als quals la identitat única no té sentit; és a dir, un conjunt de valors que es comparen per valor i no per identitat.

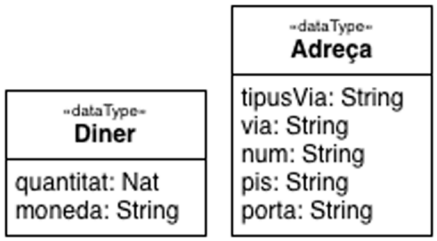
Si pensem en el nombre 15 o en l'hora 13:46, no podem pensar en termes d'identitat. No té sentit distingir entre dues instàncies diferents del nombre 15 o de les 13:46, ni pensar en termes d'esborrar un nombre 15 o tenir-ne 25 còpies; i tampoc no té sentit pensar en una llista de totes les hores. Per tant, els nombres i les hores són dos exemples de tipus de dades.

En canvi, encara que dos missatges d'una carpeta tinguin el mateix autor, assumpte, cos i data d'enviament, els dos missatges no són el mateix, ja que tenen una identitat pròpia. Si esborrem un dels missatges l'altre no s'esborrarà; no és el mateix tenir 2 còpies del missatge que tenir-ne 25; si fem una llista dels missatges d'una carpeta ens apareixeran dos missatges iguals un sota l'altre.

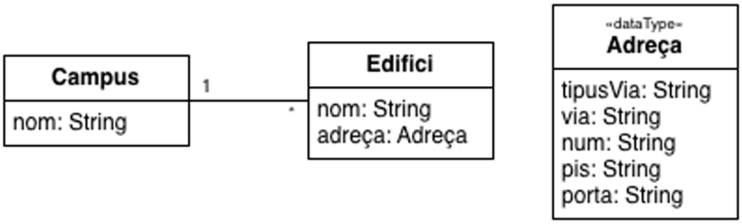
Una altra diferència és que els valors pertanyents a tipus de dades són, des d'un punt de vista conceptual, immutables. No té sentit modificar les 13:46. En tot cas, si teníem una reunió a les 13:46, la podem canviar a les 14:00 canviant el valor 13:46 pel valor 14:00. Si realment modifiquéssim el valor 13:46, totes aquelles reunions que comencin o acabin a les 13:46, els missatges enviats a les 13:46, etc., tindrien un canvi en algun dels seus atributs.

Els tipus de dades poden ser primitius (com els Strings, els nombres, els booleans o les dates) o poden ser estructurats, constituïts, al seu torn, per diversos camps, com ara les quantitats (23 €), els colors (RGB 70,0,130), etc. En el cas dels tipus de dades estructurats ens pot interessar representar-los visualment al diagrama de classes UML per a indicar quins són els camps que els componen.

Per a representar un tipus de dades en UML es representa un classificador amb l'estereotip *dataType*.



Fins i tot quan representem un tipus de dades de manera gràfica, preferirem modelitzar les propietats d'aquest tipus com a atributs i no pas com a associacions.



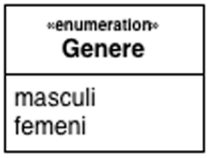
Enumeracions

Una enumeració és un tipus de dades especial en què donem la llista exacta de valors que formen el tipus de dades.

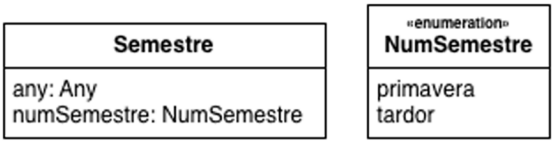
Exemple d'enumeració

Si estiguéssim interessats en el gènere de les persones definiríem, a la classe *persona*, un atribut *gènere*. Aquest no és, òbviament, un String, ni tampoc un booleà (encara que només pugui prendre dos valors, aquests no són *cert* ni *fals*).

Per a definir una enumeració només cal fer una llista dels valors vàlids. UML ofereix una notació gràfica en forma de classificador amb l'estereotip *enumeration* i amb la llista de valors en lloc dels atributs.



A l'exemple del campus el que sí que teníem era un atribut per al número de semestre. Tenint en compte que aquest només pot ser 1 o 2 o, si fem servir la nomenclatura pròpia de la universitat d'exemple, primavera o tardor, podríem haver definit el semestre de la manera següent:



4.7.2. Atributs: notació UML avançada

Tot i que la majoria no es fan servir en el model del domini, els atributs UML tenen altres elements que es poden indicar segons mostra la sintaxi completa:

```
visibilitat nom: tipus multiplicitat = valor-per-defecte {propietats}
```

Model de domini

Per a fer el model del domini, no farem servir la visibilitat ni el valor per defecte, que, per tant, no s'indicaran; i molt rarament es faran servir propietats.

Els elements dels quals encara no havíem parlat són:

- `visibilitat`. Un sol caràcter que indica la visibilitat de l'atribut i que pot ser "+" per a visibilitat pública, "-" per a privada, "#" per a protegida i "~" per a visibilitat de paquet. Normalment, durant l'anàlisi, no indicarem cap visibilitat als atributs dels nostres models.

- `valor-per-defecte`. El valor per defecte que pren l'atribut quan es crea un objecte en cas que no s'indiqui un valor inicial per a aquest.

- `propietats`. Permet indicar propietats addicionals a l'atribut. Un exemple de propietat és {readOnly}, que indica que un atribut no pot ser modificat un cop s'ha creat l'objecte o {ordered}, que, per a un atribut multivaluat, indica que l'ordre dels valors és rellevant i que, per exemple, no és el mateix tenir els valors [933455667, 654433221] que els valors [654433221, 933455667].

Un altre concepte d'orientació a objectes que no fem servir durant l'anàlisi és l'àmbit. En UML els atributs amb àmbit de classe van subratllats, mentre que els atributs amb àmbit d'instància no hi van.

4.8. Associacions: elements avançats

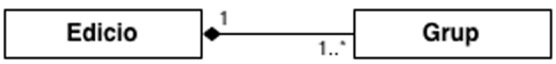
4.8.1. Composició

Algunes associacions representen una associació part-tot més forta del que és habitual. Per a representar aquest fet, UML defineix el concepte de composició.

Una composició UML és una associació entre dues classes (la composta i la classe component) tal que:

- Si destruïm una instància composta, tots els seus components també són destruïts.
- No hi pot haver instàncies de la classe component que no formin part d'una i només una instància composta.
- No podem agafar una instància component i canviar-la de compost

UML permet representar les composicions amb un rombe negre a l'extrem de la classe composta:



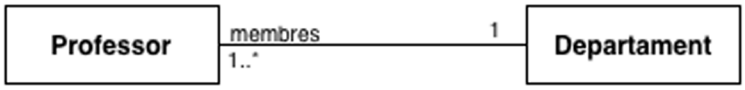
Exemple de composició

Cada edició representa, com ja s'ha vist, un semestre d'una assignatura, i està formada per un o més grups. Si eliminéssim una edició tots els seus grups s'han d'eliminar; no té sentit tenir un grup que no sigui de cap edició; i, finalment, no té sentit agafar un grup i canviar-lo d'edició. Per tant, podem dir que l'associació entre Edicio i Grup és una composició en què Edicio és la classe composta i el Grup la classe component, ja que una edició està composta per diversos grups.

UML també defineix un tipus d'associació que anomena *agregació*. Una agregació és una associació amb un cert significat part-tot però sense cap restricció o semàntica especial (a diferència de la composició). És per això que els autors mateixos d'UML i Larman recomanen no fer servir agregacions en els nostres models. Malgrat això, UML en permet representar (igual que les composicions però amb el rombe blanc).

4.8.2. Associacions amb repeticions o amb ordre

UML considera que els extrems d'associació són, per defecte, sense repeticions i sense ordre.



Així, en aquest exemple, un departament pot tenir un cert nombre de professors associats, però no hi ha cap ordre en aquest conjunt; dit d'una altra manera, no té sentit distingir entre el conjunt de professors [Maria, Joan] i el conjunt [Joan, Maria].

A més, en aquest exemple, el conjunt de membres d'un departament no pot tenir repeticions. Un mateix professor no pot ser membre més d'un cop del mateix departament; o, el que és el mateix, no té sentit dir que els membres d'un departament són [Maria, Maria, Joan] (si entenem que això vol dir que una única professora anomenada Maria és membre del departament dues vegades).

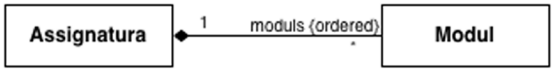
Si en un extrem d'associació l'ordre dels objectes és rellevant, cal indicar-ho, en UML, fent servir la paraula clau *ordered* entre claus.

Observació

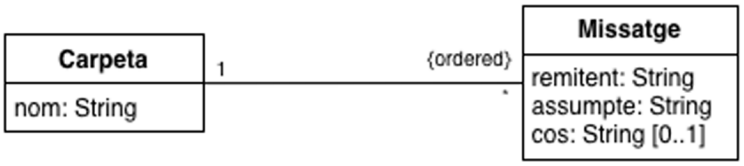
S'ha considerat que l'associació de l'assignatura amb els seus mòduls és una composició, perquè si eliminem una assignatura hem d'eliminar els mòduls que la formen, no podem tenir un mòdul que no pertanyi a cap assignatura i, finalment, no té sentit parlar de moure un mòdul a una altra assignatura. Vegeu el subapartat 4.8.1.

Exemple

Si volem conèixer el temari de cada assignatura, podem dir que una assignatura té un conjunt de mòduls. Però l'ordre dels temes d'una assignatura és rellevant, ja que no és el mateix un temari format pels mòduls 2,1,3 que dir que està format pels mòduls 1,2,3.



Encara un altre exemple. En una carpeta els missatges tenen un ordre que és rellevant. Tot i que s'ordenen per data d'arribada, si dos missatges arribessin exactament al mateix temps, és important saber quin va abans que quin altre.



Normalment, durant l'anàlisi, no trobarem utilitat a fer servir associacions que permetin repeticions. UML, però, permet indicar-ho fent servir una notació semblant al cas anterior però amb la paraula clau *non-unique*.

4.8.3. Notació UML avançada

Els altres elements de les associacions UML són:

- **Visibilitat dels extrems d'associació**. Un sol caràcter davant del nom de rol, que indica la visibilitat de l'extrem d'associació, i que pot ser "+" per a visibilitat pública, "-" per a privada, "#" per a protegida i "~" per a visibilitat de paquet.

- **Propietats**. Com en el cas dels atributs, permeten indicar propietats addicionals dels extrems d'associació com les ja vistes *readonly* (que també és aplicable als extrems d'associació), *non-unique* o *ordered*.

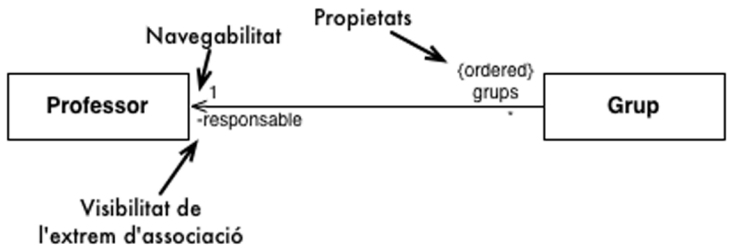
- **Navegabilitat**. De vegades volem que una associació només sigui navegable en un dels seus sentits, és a dir, que només una de les classes tingui una propietat lligada a l'associació. En el cas de l'exemple hem indicat que l'associació és navegable de Grup cap a Professor –i, per tant, la classe *Grup* té una propietat *professor*– però que no ho és de Professor a Grup –i, per tant, la classe *Professor* no té cap propietat anomenada *grups*.

Navegabilitats dobles

Tot i que UML no indica cap valor per defecte per a les navegabilitats, durant l'anàlisi, quan elaborem el model del domini, suposarem que les navegabilitats són sempre dobles i, per tant, no les indicarem als diagrames.

Exemple

Notació UML completa de les associacions

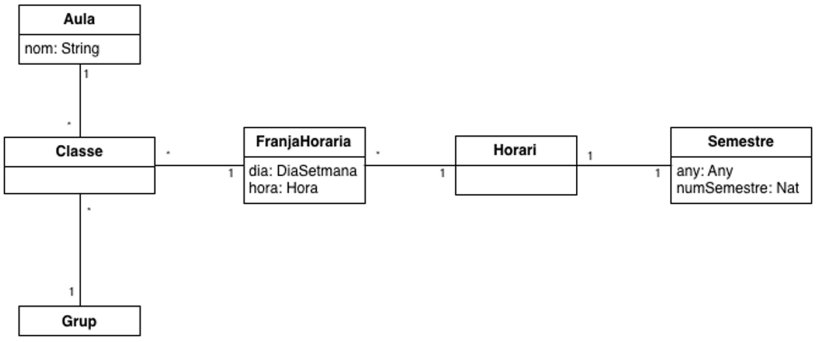


4.8.4. Associacions ternàries (i *N*-àries en què *N* > 2)

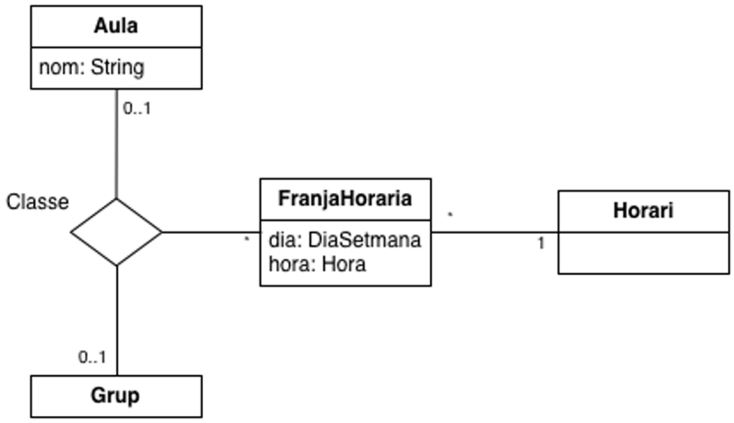
En UML una associació pot tenir més de 2 extrems d'associació. Anomenem **associació ternària** les associacions amb 3 extrems d'associació i **associació quaternària** les que en tenen 4.

Exemple d'associacions ternàries

Vam modelitzar els horaris com un conjunt de franges horàries. A cada franja es programaven un conjunt de classes, cada una de les quals associava una aula a un grup.



Aquest mateix concepte s'hauria pogut expressar com una associació ternària que representés les classes.



L'associació *Classe* és una associació ternària, ja que té tres extrems d'associació: el grup, l'aula i la franja horària (cada instància de l'associació associa una aula, un grup i una franja horària determinades). En aquest sentit, doncs, la solució és força semblant a l'anterior.

Les multiplicitats d'una associació ternària funcionen de manera una mica especial. El 0..1 al costat d'aula indica que no hi pot haver més d'una classe programada per al mateix grup i la mateixa franja horària. No vol dir, per tant, que puguem tenir una classe que tingui grup però no aula.

De manera semblant, el 0..1 al costat de grup indica que no hi pot haver més d'una classe programada a la mateixa aula i per a la mateixa franja horària.

En casos molt determinats les associacions ternàries (i altres associacions no binàries) permeten expressar de manera gràfica les restriccions de clau. En aquest sentit, són més expressives que fer servir només classes i associacions binàries.

Ara bé, s'ha de tenir en compte que l'ús de ternàries és menys entenedor, sobretot pel que fa les seves multiplicitats.

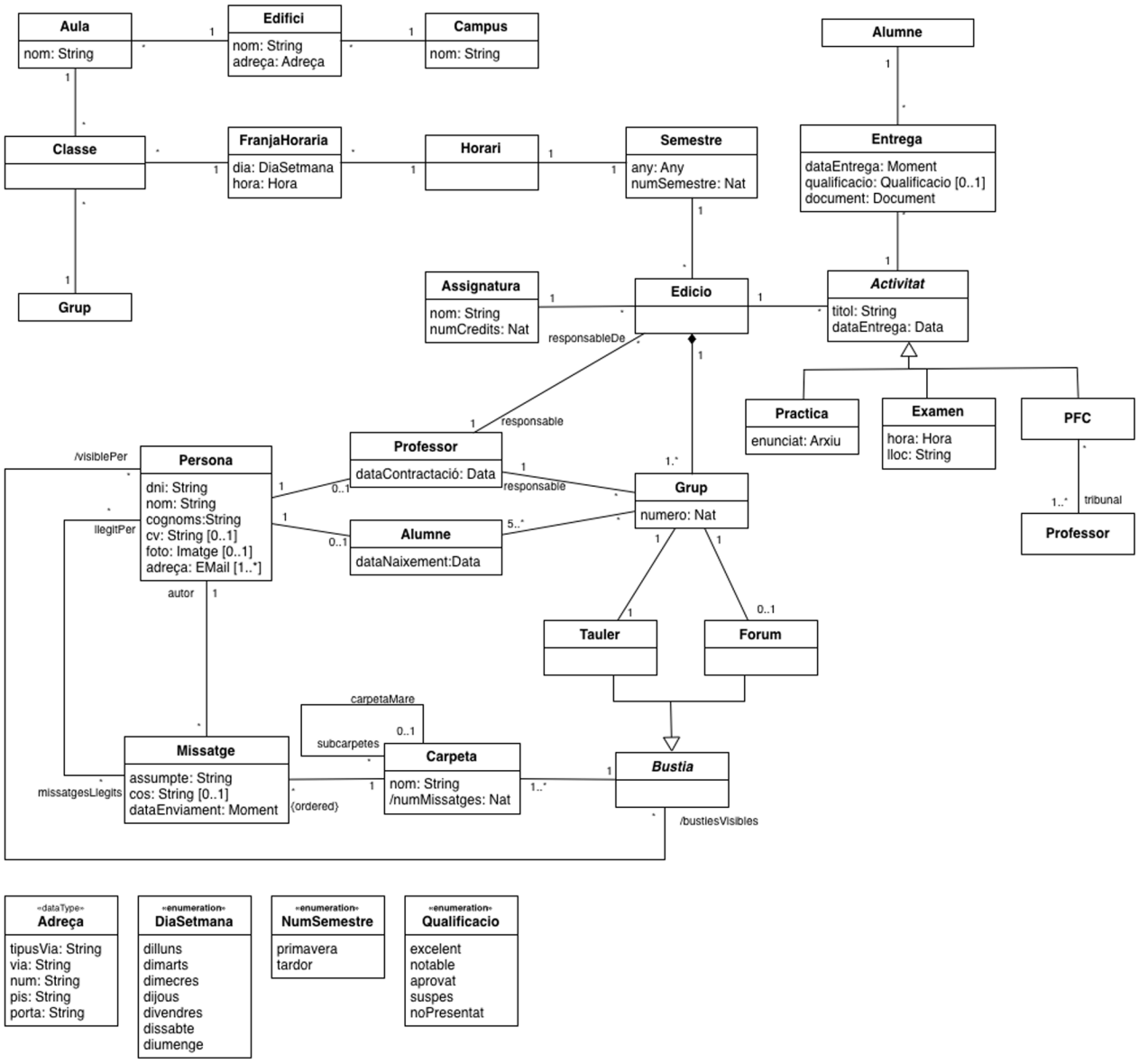
Atès que la nostra finalitat en fer modelització del domini és ajudar a entendre el problema per resoldre a les persones implicades en el desenvolupament de programari, tret que totes les parts implicades entenguin molt bé l'ús correcte d'aquest tipus d'associacions, no és aconsellable fer-les servir.

D'altra banda, la majoria d'eines de modelització no suporten les associacions no binàries o ho fan només parcialment.

4.9. El model resultant

A continuació es mostra el model del domini resultant de l'anàlisi del cas d'exemple de la universitat. Algunes classes s'han representat més d'un cop per facilitar la lectura del diagrama; en tal cas, només una de les representacions de la classe té el compartiment dels atributs per a evitar repetir innecessàriament aquesta informació.

Aquest no pretén ser un model fidel a la realitat sinó un exemple de model UML que representaria el model conceptual del campus virtual que hem discutit. No hem tingut en compte, per exemple, els requisits ni els mòduls que formen el temari de les assignatures, tot i haver-ne parlat.



Model conceptual del campus virtual

Restriccions de clau:

- Persona: dni
- Assignatura: nom
- Campus: nom
- Semestre: any + numSemestre
- Edifici: campus + nom
- Aula: edifici + nom
- Horari: semestre
- FranjaHoraria: horari + dia + hora
- Classe: franjaHoraria + aula, franjaHoraria + grup
- Edicio: semestre + assignatura
- Grup: edicio + numero
- Tauler: grup
- Forum: grup
- Activitat: edicio + titol
- Professor: persona
- Alumne: persona
- Missatge: carpeta + posició a carpeta::missatges

Altres restriccions d'integritat:

- Per a tota activitat, la data d'entrega ha de ser una data de l'any del semestre del curs al qual està associada l'activitat.
- Les carpetes formen una jerarquia vàlida de mares-subcarpetes (no es poden formar cicles tals que una carpeta té com a mare una filla, o una filla de la seva filla, etc.).
- No hi pot haver més d'una carpeta sense mare amb el mateix nom dins la mateixa bústia
- No hi pot haver més d'una carpeta amb la mateixa carpeta mare i el mateix nom.

Informació derivada:

- *Carpeta::numMissatges* és el nombre de missatges associats a la carpeta

- *Persona::bustiesVisibles* és el conjunt format pel tauler i el fòrum (per a aquells que en tinguin) de tots els grups als quals la persona està associada com a professor o com a alumne.

Resum

En aquest mòdul hem vist com podem utilitzar els casos d'ús i l'orientació a objectes per a fer anàlisi en UML de programari per a sistemes d'informació.

El llenguatge UML és el llenguatge estàndard que ens permet crear models visuals del programari i es pot fer servir des de diverses perspectives, de les quals nosaltres fem servir la conceptual.

El model de casos d'ús consisteix a fer l'anàlisi basada en casos d'ús, fer-ne la descripció textual tal com hem vist al mòdul "Requisits" i fer-ne diagrames de casos d'ús UML que mostren els casos d'ús, els actors i les relacions entre aquests. Cal triar amb cura com agrupem els casos d'ús en diagrames per tal d'elaborar un document el més entenedor possible.

El diagrama d'activitats UML ens permet documentar de manera visual el comportament d'un cas d'ús vist com un procés. Ens permet indicar la seqüència d'activitats que es duen a terme, branques condicionals, paral·lelisme i, fins i tot, com diversos participants interactuen en el procés (mitjançant carrils).

Però sovint també voldrem modelitzar la interfície del sistema per a un determinat cas d'ús. Per a això podem redactar casos d'ús concrets, que tenen en compte la tecnologia i la implementació. Podem representar les pantalles d'un cas d'ús fent-ne models que mostrin els elements principals que les formen; el diagrama d'estats UML, a més, ens permet representar visualment les transicions que es produeixen entre aquestes.

El model del domini és un diagrama de classes UML que representa les classes conceptuals del món real en un domini d'interès fent servir l'orientació a objectes. Per a elaborar-lo primer cal identificar les classes i afegir-los atributs i associacions. Podem fer servir la relació d'herència per a especialitzar classes (quan trobem casos concrets que s'aparten de la definició general que teníem fins al moment) o per a fer generalització (quan trobem dues classes amb elements en comú). El diagrama de classes permet, a més, fer servir altres elements, com ara la informació derivada, les enumeracions o la composició, per a representar alguns conceptes particulars de manera força precisa i formal.

Activitats

1. Feu una possible especificació textual breu per als casos d'ús "Lliurar una activitat", "Lliurar una versió nova d'una activitat" i "Adjuntar arxiu" de l'exemple del subapartat 2.1. Tingueu especialment en compte que cal reflectir les relacions entre casos d'ús mostrades al diagrama.

2. Refeu el diagrama de l'exemple del subapartat 2.1 per a no utilitzar la relació *extend*, tal com es recomana al subapartat 2.1.6. Refeu l'activitat anterior tenint en compte el nou diagrama.

3. Suposeu que disposem del cas d'ús següent:

Preparar inici curs (nivell usuari)

El professor d'un grup vol preparar l'inici de curs. Primer escull el grup del qual vol preparar l'inici de curs i, després, ha d'escriure un missatge de benvinguda al tauler i, si la seva assignatura té fòrum, organitzar les carpetes del fòrum. El missatge de benvinguda al tauler i l'organització de les carpetes es poden fer en qualsevol ordre, i no es pot donar el cas d'ús per finalitzat fins que el professor no hagi fet totes dues coses (tret que el grup no tingui fòrum, cas en el qual només cal escriure el missatge de benvinguda).

Feu el diagrama d'activitats d'aquest cas d'ús.

4. Suposeu que disposem del cas d'ús següent:

Preparar edició assignatura (nivell usuari)

Un membre del personal acadèmic, per ordre del cap d'estudis, vol donar d'alta una nova edició d'una assignatura. Escull l'assignatura i un professor responsable, i el sistema enregistra la nova edició. A continuació, l'usuari indica si en aquesta assignatura es fa servir fòrum o no i crea un o més grups de l'assignatura. El sistema assigna, a cada grup, un nombre de grup consecutiu (que l'usuari pot canviar). Per a cada grup, l'usuari assigna un professor. El sistema enregistra els nous grups i els crea un tauler a cada un i, si s'ha indicat que es fan servir, un fòrum a cada un.

a) Escriviu una especificació concreta i detallada d'aquest cas d'ús
b) Feu el diagrama d'activitats d'aquest cas d'ús.
c) Proposeu un model d'interfície gràfica d'usuari que inclogui esbossos de les pantalles i mapa navegacional.

5. Volem desenvolupar un sistema per a fer la gestió de projectes mitjançant la metodologia àgil Scrum (o, si més no, una versió simplificada).

Cada projecte, que s'identificarà per nom, tindrà un equip de com a mínim dues persones identificades, també, per un nom, i estarà format per un conjunt d'iteracions (com a mínim una), identificades, dins el projecte, per un número consecutiu. De cada iteració en voldrem saber, a més, la data d'inici, la data de la reunió de retrospectiva (amb què finalitza la iteració) i la velocitat prevista (un nombre corresponent al nombre de punts d'estimació que es preveu, en iniciar la iteració, que pugui acomplir).

Cada projecte té també un conjunt d'històries d'usuari anomenat *backlog*. Les històries s'identifiquen (dins el projecte) per un títol, tenen un estat (pendent, en desenvolupament o finalitzada) i una estimació (un dels nombres del *planning poker*, no s'inclou el caràcter interrogant). Aquest conjunt d'històries s'ordenarà, dins el projecte, per prioritat.

A mesura que avança el projecte, a les iteracions es van assignant històries d'usuari no finalitzades, de tal manera que algunes iteracions tindran un conjunt d'històries d'usuari (de les del projecte) ordenades per prioritat.

Durant el desenvolupament, cada dia laborable, per a cada projecte, es farà una reunió (el *daily scrum*) en què, entre altres coses, es decidirà quants punts d'estimació es considera que falten per a acabar la iteració. El sistema ha de desar el valor proporcionat cada dia, a partir del qual es farà un gràfic de seguiment anomenat *burndown chart*.

En acabar una iteració, les històries d'usuari de la iteració que no s'hagin acabat s'associaran a una nova iteració i es donarà la iteració per acabada. La velocitat real d'una iteració es calcula com la suma de les estimacions de les històries d'usuari que, per a aquella iteració, s'han finalitzat.

Feu el model del domini del sistema descrit. Més concretament:

a) Feu el diagrama de classes del model del domini

b) Indiqueu la informació derivada i regles d'integritat addicionals que calguin.

6. Volem desenvolupar un sistema de gestió del catàleg per a una cadena de roba amb diverses botigues distribuïdes per tot el país.

Cada botiga té un nom, que la identifica, i una adreça. El catàleg consisteix en un conjunt de models, cadascun dels quals s'identifica per un nom i una marca, com ara el model 309 de Tangerine Jeans. De cada model hi ha diverses variants, cadascuna de les quals s'identifica, dins el model, per un nom i una talla, com ara la variant Deep blue, talla 32, del model mencionat abans. De les marques, a més del nom que les identifica, ens cal conèixer una adreça de correu electrònic de contacte.

Els preus dels models poden variar segons la temporada i la botiga. Per això quan s'indica el preu d'un model en una botiga s'indica el període de validesa del preu. Un preu es considera vigent si la data actual està dins el període de validesa, i no ha de poder passar que dos preus d'un mateix model a la mateixa botiga s'encavalquin. Si, en una botiga i una data donada, un model no té preu, es considera que no està disponible per a la venta.

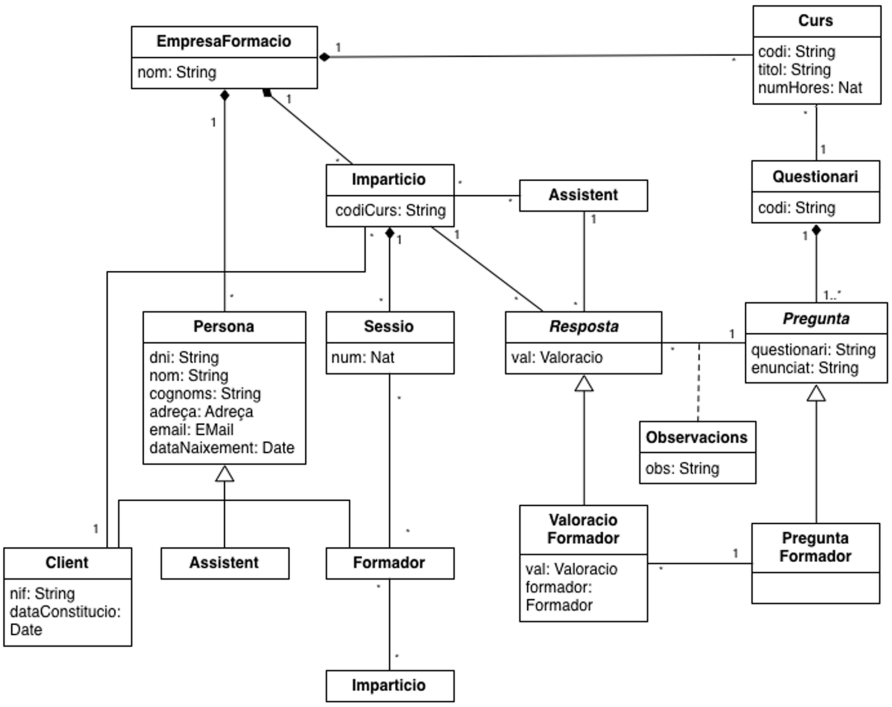
Alguns preus són de descompte. En tal cas, cal desar un preu, com sempre, i un percentatge de descompte.

Feu el model del domini del sistema descrit. Més concretament:

a) Feu el diagrama de classes del model del domini.
b) Indiqueu la informació derivada i les regles d'integritat addicionals que calguin.

7. Volem desenvolupar un sistema informàtic per a una empresa de formació anomenada *Training SA* per a gestionar els cursos que imparteix, els formadors que els imparteixen i els qüestionaris de valoració de curs que passen als assistents.

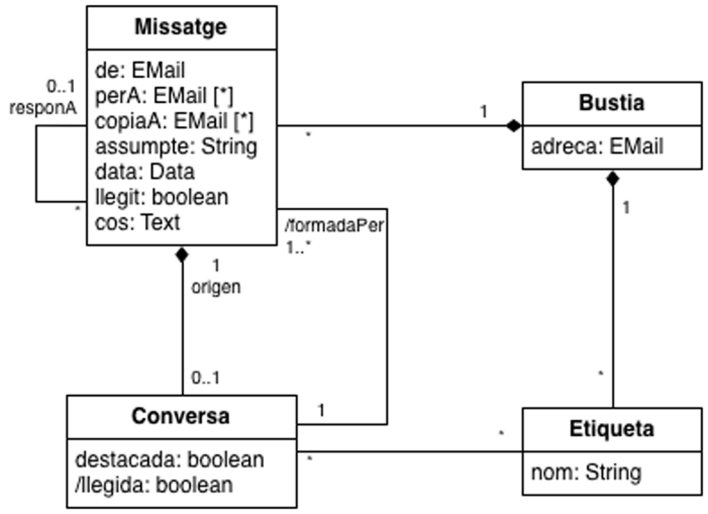
Tenim un model del domini ja elaborat:



Aquest model mostra que cada curs té un qüestionari associat que està compost per un conjunt d'almenys una pregunta; algunes preguntes són sobre el conjunt del curs però algunes altres són només sobre el formador. Quan s'imparteix un curs es vol saber quins formadors l'imparteixen (i, en concret, quins imparteixen cada sessió) i quins assistents han assistit a cada sessió (es passa llista). En finalitzar el curs es passa el qüestionari i cal conèixer les respostes (valoració i observacions) que fan els assistents; per a les preguntes sobre el formador hi ha d'haver una resposta per a cada formador dels que hagin impartit el curs.

Feu una crítica del model del domini proposat tot indicant com es poden corregir els defectes que hi trobeu.

8. Disposem d'un sistema de correu electrònic que agrupa els correus per converses, en què una conversa és un correu que no respon a cap altre correu i tot l'arbre de respostes directes o indirectes que en pengen; a més, el sistema permet etiquetar les converses. Disposem, també, d'un model del domini per al sistema en qüestió:



Claus de les classes:

- Bustia: adreca, Etiqueta:nom

Restriccions d'integritat i informació derivada:

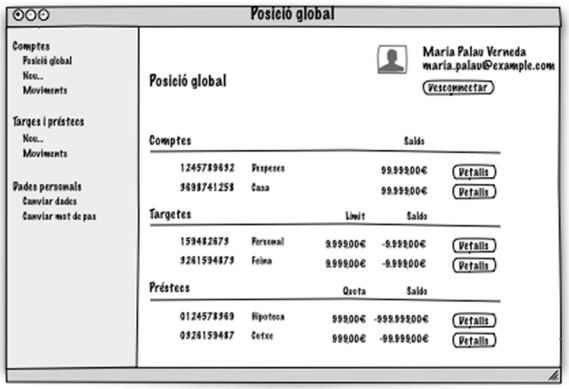
- Els missatges que no són una resposta, i només aquests, han de ser origen d'una conversa.
- Per a una conversa *llegida* és fals si i només si *llegit* és fals per a tots els missatges que formen la conversa.
- Una conversa està formada pel seu missatge origen i per les respostes a qualsevol dels missatges que forma part de la conversa.
- Un missatge només pot ser resposta d'un altre que estigui a la mateixa bústia.

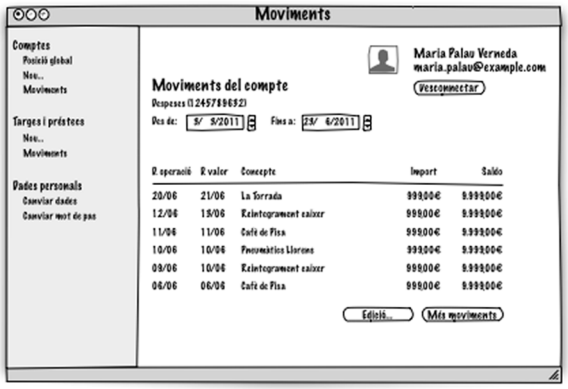
Responeu les preguntes següents:

a) Pot ser que un missatge ni tingui destinataris (perA) ni vagi amb còpia a ningú (copiaA)?
b) Pot ser que un missatge no tingui cos?
c) Què passa si esborrem una bústia?
d) I si esborrem una etiqueta?
e) Com podríem representar la primera restricció d'integritat de manera gràfica en UML?
f) La multiplicitat del rol d'associació *formadaPer* és 1..*. Per què no és *?
g) Quines diferències hi hauria si en lloc de modelitzar l'etiqueta com una classe l'haguéssim modelitzada com un atribut *etiqueta*:*String* de la classe *Conversa*?

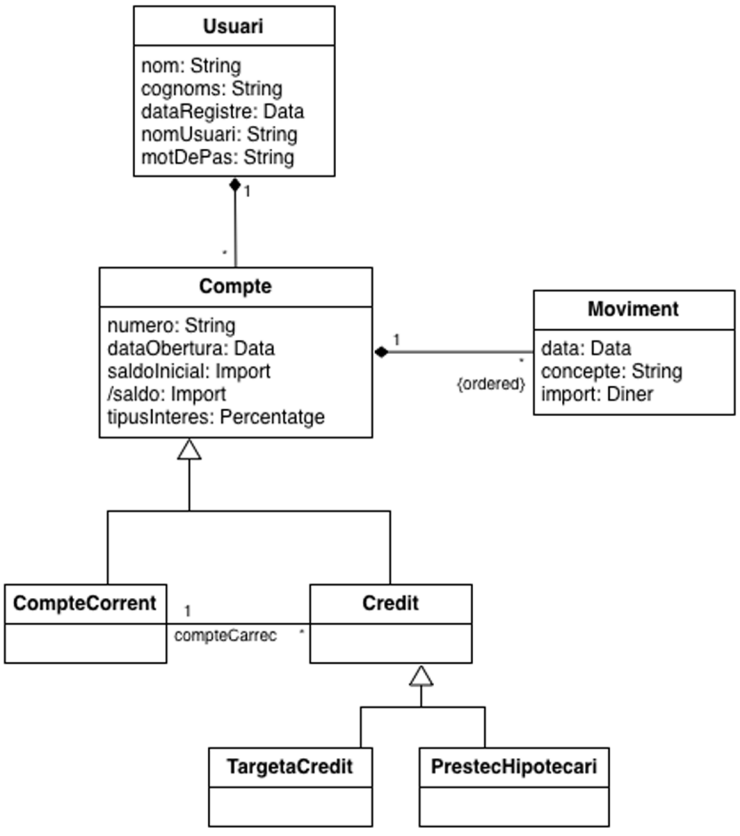
9. Volem desenvolupar un sistema per a la gestió personal de finances. Els usuaris s'enregistren proporcionant tot de dades i, un cop registrats, poden donar d'alta comptes, que poden ser comptes corrents, targetes de crèdit o préstecs hipotecaris. El sistema els permet gestionar els moviments de cada compte.

Disposem dels models de dues de les pantalles:





També tenim un model del domini parcial que voldríem completar i corregir:



Claus de les classes:

- Usuari: nomUsuari, Compte: usuari + numero, Moviment: ordre dins el compte

Restriccions d'integritat

- L'usuari d'un compte de crèdit ha de ser el mateix que l'usuari del compte de càrrec associat.
- El saldo d'un compte es calcula com saldoInicial + suma de tots els imports de tots els moviments associats al compte

Comproveu la validesa del model del domini assumint que els models de les pantalles són correctes. Indiqueu les errades i mancances que hi trobeu.

Exercicis d'autoavaluació

1. Què és el llenguatge UML?

2. Quins són els tres usos principals per al llenguatge UML?

3. Quina és la relació entre el diagrama de casos d'ús i la descripció textual d'aquests?

4. Què són els punts d'extensió d'un cas d'ús?

5. Com podem indicar la lògica condicional en un diagrama d'activitats?

6. Per què cal fer un model de la interfície d'usuari?

7. Què són els casos d'ús essencials?

8. Quin és l'objectiu dels models de les pantalles?

9. Què és el mapa navegacional?

10. Què és el model del domini?

11. Indiqueu tres categories típiques de classes conceptuals.

12. Quines indicacions ens dóna la "regla del cartògraf"?

13. Què indica una multiplicitat "*" en un atribut?

14. Per què no fem servir atributs per a representar relacions entre classes?

15. En quins casos ens hem de plantejar si cal dur a terme un procés d'especialització d'una classe?

16. Què són les restriccions d'integritat?

17. Què és la informació derivada?

18. Quina és la principal diferència entre un tipus de dades i una classe?

Solucionari

Activitats

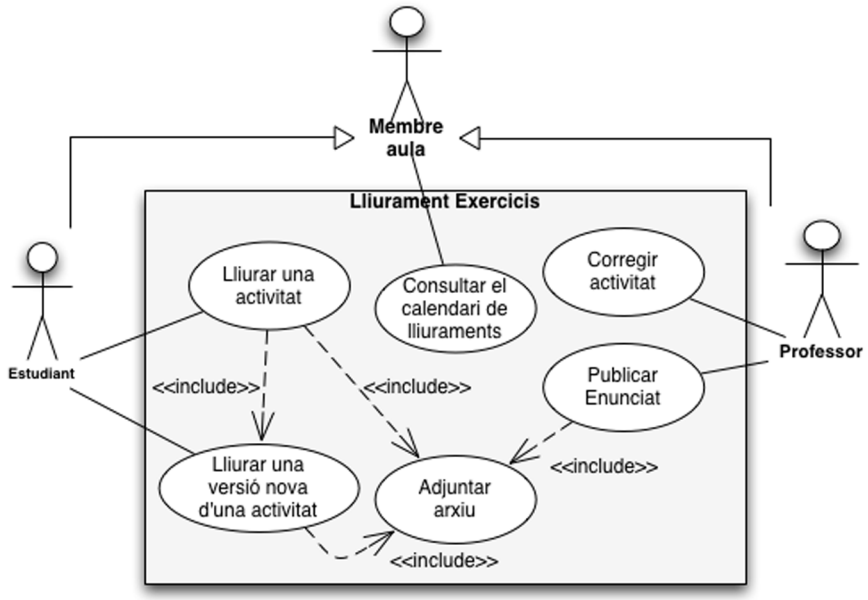
1.

a) Cas d'ús: lliurar una activitat
L'estudiant demana lliurar una activitat. El sistema li mostra una llista amb les assignatures que està cursant, de les quals l'estudiant n'escull una. A continuació el sistema mostra una llista d'activitats de l'assignatura seleccionada i l'usuari en selecciona una i adjunta l'arxiu de l'entrega. El sistema enregistra l'entrega, incloent-hi la data i hora en què s'ha fet.

b) Cas d'ús: lliurar una versió nova d'una activitat
Aquest cas d'ús estén el cas d'ús Lliurar una activitat quan l'estudiant va a adjuntar l'arxiu d'entrega i resulta que ja havia fet l'entrega prèviament. El sistema demana confirmació i l'usuari confirma que vol lliurar una nova versió i adjunta un nou arxiu. El sistema enregistra la nova versió i la nova data i hora d'entrega.

c) Cas d'ús: adjuntar arxiu
L'usuari vol adjuntar un arxiu i el sistema li mostra un navegador del seu espai de carpetes, situat, inicialment, a la seva carpeta d'usuari. L'usuari navega per l'espai de carpetes fins que troba l'arxiu que vol adjuntar, el selecciona i ho confirma. El sistema desa l'arxiu adjuntat.

2.

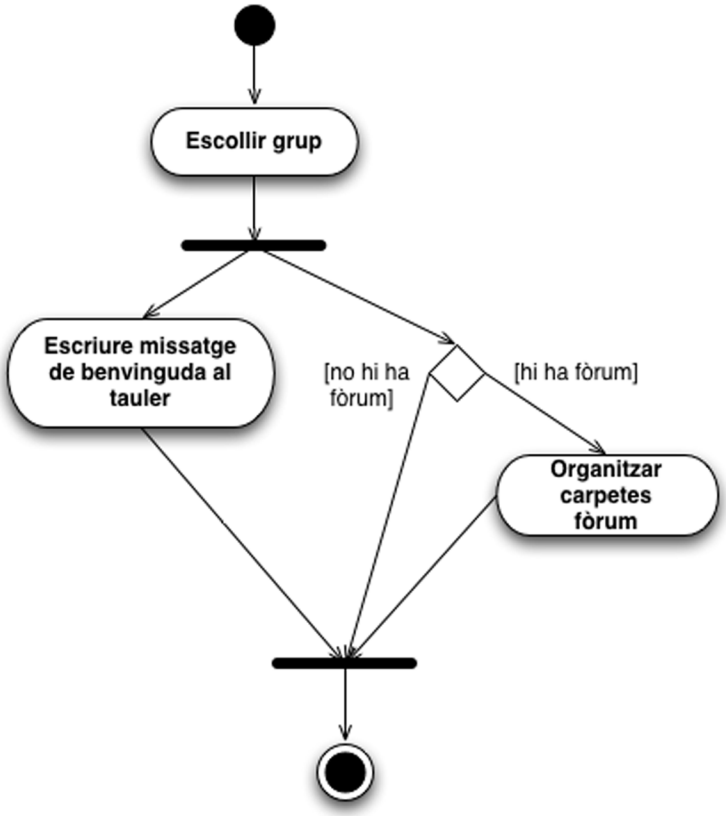


L'especificació textual del cas d'ús *Adjuntar arxiu* no canvia. La dels altres dos casos d'ús, però, sí que ho fa:

a) Cas d'ús: lliurar una activitat. L'estudiant demana lliurar una activitat. El sistema li mostra una llista amb les assignatures que està cursant, de les quals l'estudiant n'escull una. A continuació el sistema mostra una llista d'activitats de l'assignatura seleccionada i l'usuari en selecciona una i adjunta l'arxiu de l'entrega. El sistema enregistra l'entrega, incloent-hi la data i hora en què s'ha fet. Si l'estudiant ja havia fet l'entrega, l'estudiant pot lliurar una versió nova d'una activitat.

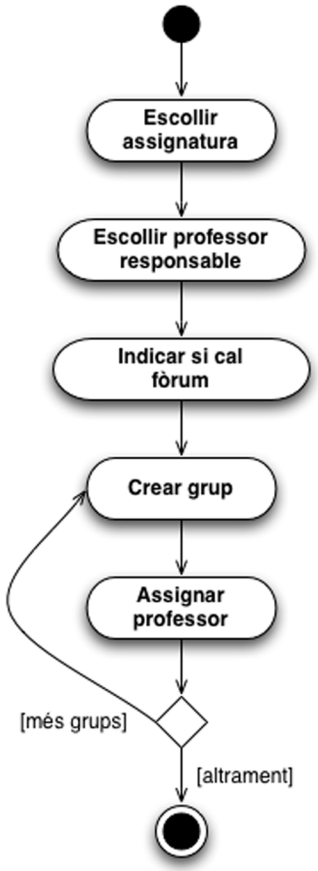
b) Cas d'ús: lliurar una versió nova d'una activitat. El sistema demana confirmació i l'usuari confirma que vol lliurar una nova versió i adjunta un nou arxiu. El sistema enregistra la nova versió i la nova data i hora d'entrega.

3.



S'ha fet servir una bifurcació per a indicar que escriure el missatge de benvinguda i organitzar les carpetes al fòrum (si cal) es fan en paral·lel, i una unió per a indicar que el procés no pot acabar fins que no s'han fet totes dues activitats. I s'ha fet servir una decisió per a indicar que només cal organitzar les carpetes si hi ha fòrum; en cas contrari es va directament a la unió i, per tant, el procés s'acaba tan bon punt s'hagi escrit el missatge de benvinguda.

4. El diagrama d'activitats del cas d'ús es mostra a continuació:



Un possible cas d'ús concret podria ser el següent:

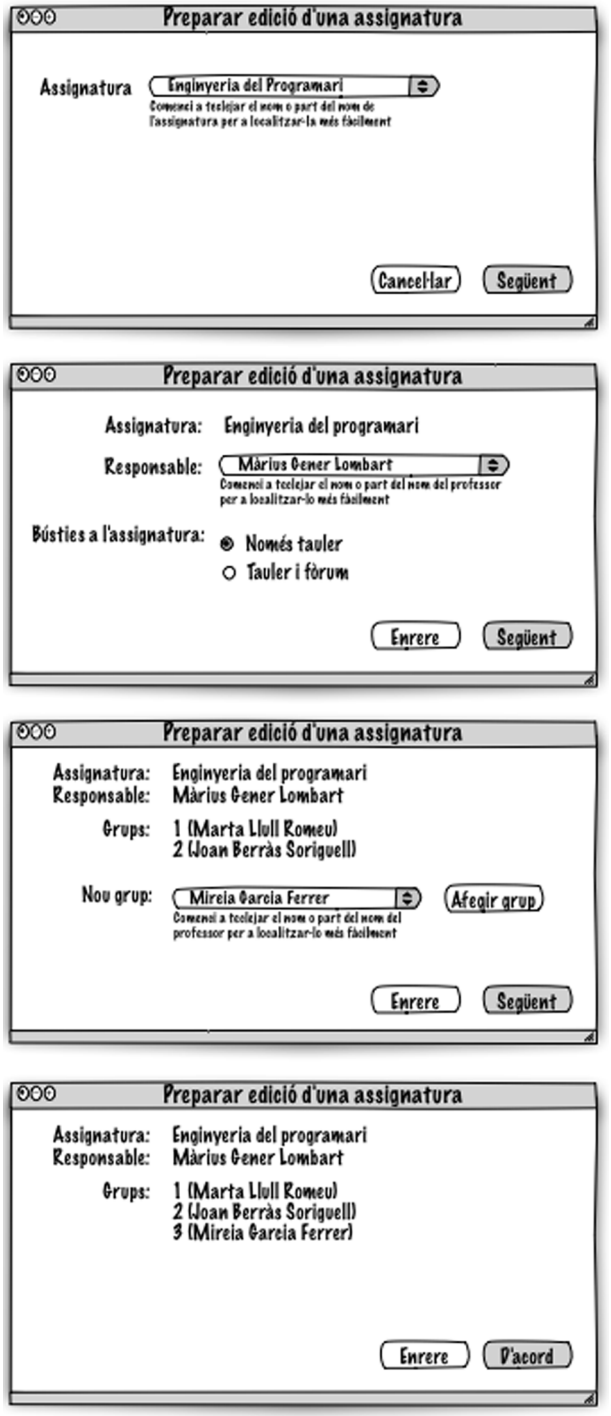
Cas d'ús: preparar edició assignatura
Actor principal: usuari
Escenari principal d'èxit:
1. El sistema mostra la pantalla *Escollir assignatura* amb un desplegable de les assignatures que encara no tenen edició en el curs actual.
2. L'usuari selecciona una assignatura i prem *Següent*.
3. El sistema mostra la pantalla *Responsable i bústies* amb un desplegable de professors que poden fer de responsable de l'assignatura.
4. L'usuari selecciona un professor, escull una configuració de bústies i prem *Següent*.
5. El sistema mostra la pantalla *Grups* amb un desplegable de professors que poden portar un grup de l'assignatura per a la creació de nous grups.
6. L'usuari selecciona un professor per al nou grup i prem *Afegir grup*.
7. Els passos 5 i 6 es repeteixen fins que l'usuari no vol crear més grups, moment en què prem *Següent*.
8. El sistema mostra la pantalla *Confirmació* amb totes les dades de la nova edició de l'assignatura.
9. L'usuari prem *D'acord* i el sistema enregistra la nova edició de l'assignatura i els seus grups.

Extensions:
2a. L'usuari prem *Cancel·lar* i el cas d'ús finalitza sense canvis
***a** En qualsevol moment a partir del pas 3, l'usuari pot prémer *Enrere* i el cas d'ús torna a la pantalla anterior
(***a** indica extensions múltiples; en aquest cas, és vàlid per a tots els passos del 3 al 9).
Pantalles:

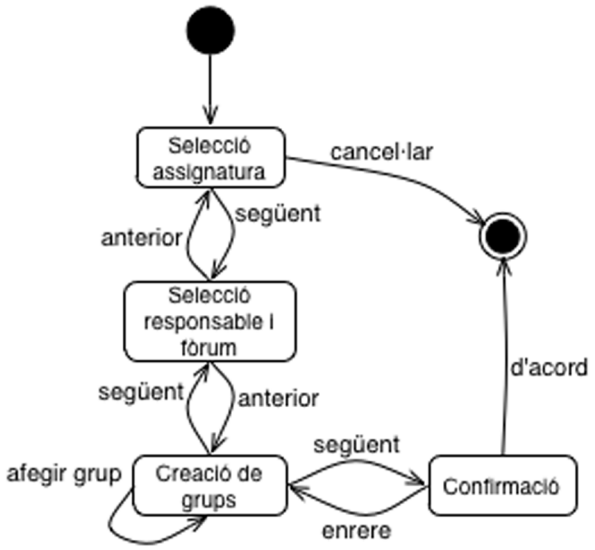
Pantalla	Dades mostrades	Dades introduïdes
Escollir assignatura	Llista de noms d'assignatura	Assignatura seleccionada
Responsable i bústies	Assignatura seleccionada Llista de professors que en poden ser responsables	Professor seleccionat Només tauler / Tauler i fòrum
Grups	Assignatura, responsable i opcions de bústies - Número - Professor responsable Llista de professors que poden portar un grup	Professor seleccionat com a responsable d'un grup
Confirmació	Assignatura, responsable, opcions de bústies Llista de grups. Per a cada un: - Número - Professor responsable	-

Com es pot veure, s'ha decidir que el professor responsable i la configuració de bústies d'aquesta edició es fan a la mateixa pantalla. Així mateix, s'ha introduït una pantalla de confirmació al final.

Els esbossos de les pantalles són els següents:

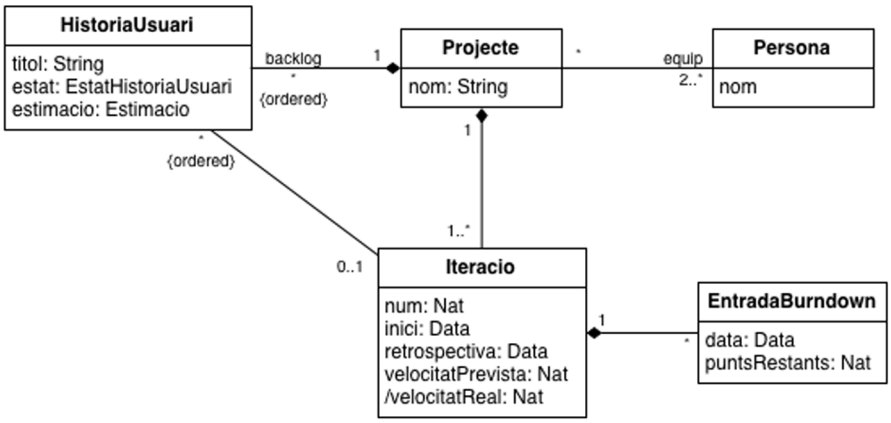


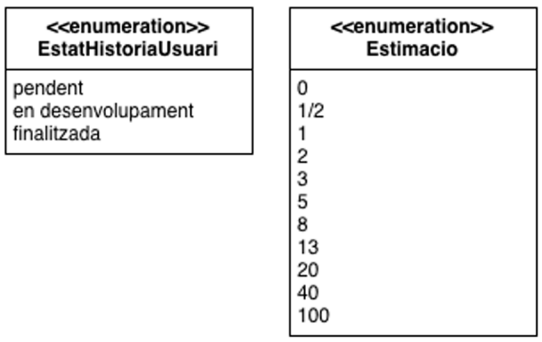
El mapa navegacional corresponent és el següent:



A cada pantalla correspon un estat del diagrama. Les transicions entre estats estan etiquetades amb l'esdeveniment que les provoca, que és una acció de l'usuari a la pantalla, com ara prémer un botó.

5.





Restriccions d'integritat:

- Claus: Projecte: nom, Persona: nom, Iteracio: projecte + nom, HistoriaUsuari: projecte + títol, EntradaBurndown: iteració + data.
- La data d'inici d'una iteració ha de ser anterior a la data de retrospectiva.

- La data de totes les entrades de *burndown* d'una iteració ha de ser entre les dates d'inici i de retrospectiva d'aquesta.
- Les històries d'usuari associades a una iteració han de pertànyer al projecte al qual pertany la iteració.

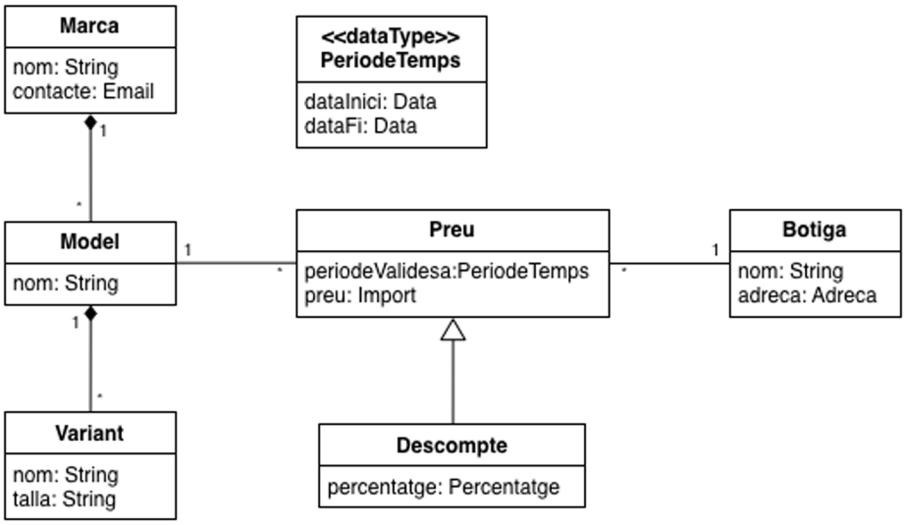
Informació derivada:

- La velocitatReal d'una iteració és la suma de les estimacions de les històries d'usuari finalitzades associades a aquella iteració.

Notes:

- S'ha considerat que hi ha diverses associacions que són composicions. Així, un projecte estarà compost del seu conjunt d'històries d'usuari i del seu conjunt d'iteracions. Aquestes, al seu torn, estaran compostes d'un conjunt d'entrades de *burndown*. Per tant, entre altres coses, podem afirmar que si esborrem un projecte també n'esborrarem les històries d'usuari, les iteracions i les entrades de *burndown*.
- S'ha fet servir {ordered} per a indicar que la llista d'històries d'usuari d'un projecte està ordenada i que l'ordre importa. Igualment per a la llista d'històries d'usuari d'una iteració.

6.



Restriccions d'integritat:

- Claus: Botiga: nom, Marca: nom, Model: marca + nom, Variant: model + nom + talla
- No hi pot haver més d'un preu del mateix model a la mateixa botiga amb períodes de validesa que s'encavalquin.

Notes:

- S'ha modelitzat el preu amb descompte com una subclasse. Per la seva simplicitat, però, s'hauria pogut indicar, senzillament, el percentatge de descompte com un atribut opcional de preu.
- Els tipus de dades Email, Adreca, Percentatge i Import no s'han modelitzat amb més detall, o bé perquè eren prou clars (Email, Percentatge i Import) o bé perquè a l'enunciat no en tenim més informació (Adreca).

7.

- D'entrada, al model mostrat no s'ha indicat cap tipus de restriccions d'integritat addicionals, ni tan sols les de clau. Caldria una nota addicional que informés sobre aquestes restriccions.
- La classe *ValoracioFormador* té un atribut de tipus Formador. Aquest, en tot cas, hauria de ser una associació.
- La classe *Imparticio* té un atribut *codiCurs* de tipus String que sembla una mena de clau forana cap a Curs. Caldria indicar que una Imparticio té associat un Curs mitjançant una associació com a tal i no a un atribut que contingui el codi del curs associat.
- El model presentat té una classe *EmpresaFormacio* que no és necessària. La presència d'aquesta classe sembla indicar que les empreses de formació i el seu nom són rellevants per al nostre sistema informàtic. Però essent, com és, per a una única empresa de forma-

ció, conèixer-ne el nom i tenir la possibilitat de tenir més d'una empresa de formació no és rellevant.

- La classe associativa *Observacions* no té gaire sentit. Si, en respondre, un assistent afegeix observacions a la resposta, aquestes serien un atribut de la classe *Resposta*. No té sentit modelitzar-les com un atribut d'una classe associativa, sobretot perquè la multiplicitat al costat de Pregunta és 1 i, per tant, cada instància de Resposta tindrà una única instància d'Observacions associada.
- La classe *Pregunta* s'ha indicat com a abstracta, quan en realitat només té una subclasse. Tret que totes les preguntes siguin, efectivament, de formador, la classe *Pregunta* ha de ser concreta. Igual passa amb la classe *Resposta*.
- La classe *Persona* no s'ha indicat com a abstracta, tot i que sembla que totes hagin de ser clients, assistents o formadors. Si, efectivament, no hi pot haver persones rellevants al problema que no pertanyin a alguna de les 3 subclasses, aleshores s'hauria d'indicar que la classe *Persona* és abstracta (posant-ne el nom en cursiva).
- La classe *Client* no sembla una subclasse de Persona correctament modelitzada. Si un client té un NIF i una data de constitució és perquè deu ser una empresa o algun altre tipus d'organització. Per tant, no té sentit que diguem que és una subclasse de Persona.
- La classe *Formador* té una associació amb la classe *Sessió* que indica les sessions en les quals participa cada formador. També té una associació amb la classe *Imparticio* que deu indicar els formadors d'una impartició d'un curs. Aquestes dues associacions haurien d'estar relacionades: o bé els professors d'una impartició haurien de ser derivats a partir dels professors de cada una de les sessions o bé hi hauria d'haver una restricció d'integritat que ens indiqui que tots els professors associats a una sessió també ho han d'estar a la impartició corresponent.
- De manera semblant al cas anterior, una ValoracioFormador té associada una PreguntaFormador i, com tota Valoracio, també una Pregunta. L'associació amb PreguntaFormador sobra (i cal indicar mitjançant una restricció d'integritat textual que la pregunta associada a una ValoracioFormador serà una PreguntaFormador).

8.

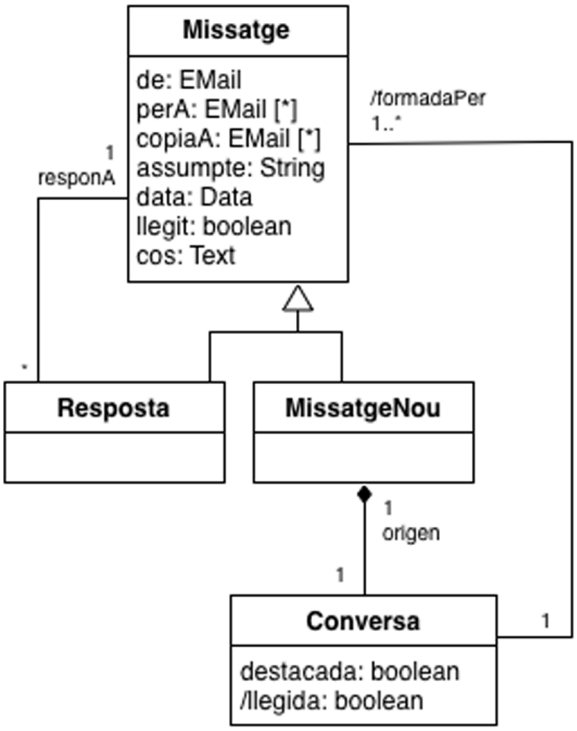
a) Segons el model de què disposem, sí.

b) No. Per al cos no s'ha indicat cap multiplicitat i, per tant, assumim que la multiplicitat és 1; és a dir, que és un atribut obligatori i univaluat.

c) Que totes les etiquetes i missatges de la bústia també s'esborraran, ja que una bústia està composta per etiquetes i per missatges. En esborrar aquells missatges que siguin origen d'una conversa, també s'esborraran les converses.

d) Res, ja que una etiqueta no és un objecte compost. Depenent del cas d'ús i de com es defineixi, probablement senzillament es desassociarà l'etiqueta de les converses on estigués associada i de la bústia on es va crear, i s'esborrarà.

e) Podríem haver fet servir l'especialització per a distingir entre els missatges que són resposta (i, per tant, tenen 1 a la multiplicitat de responA) i els que no són resposta (i, per tant, són origen d'1 i només una conversa):



f) Perquè segons la definició d'aquesta associació derivada, tota conversa està formada pel seu missatge origen i per les respostes a qualsevol missatge de la conversa. Per tant, com a mínim hi haurà el missatge que n'és l'origen.

g) D'entrada l'atribut hauria de ser multiavaluat: etiqueta:String[*]. A més, no tindríem les etiquetes com a entitats; així, per exemple, estaríem donant a entendre que no té sentit gestionar les etiquetes: crear-les, enumerar-les, etc. Dues converses podrien tenir una mateixa etiqueta només "per coincidència".

9.

- A les pantalles figura l'adreça de correu electrònic de l'usuari. Caldria afegir-la com a atribut de la classe Usuari.
- A les pantalles, cada compte, a més del número, té un nom. Caldria, doncs, afegir l'atribut *nom*:*String* a la classe *Compte*.
- A les pantalles, les targetes tenen, a més del saldo, un límit. Caldria afegir-lo com a atribut de la classe *TargetaCredit*.
- Els préstecs de les pantalles poden ser hipoteques, però poden ser d'altres tipus. Segurament la classe *PrestecHipotecari* del model del domini s'hauria d'anomenar *Prestec*, ja que no solament representa préstecs hipotecaris. Si més endavant es veiés que hi ha motius per a especialitzar més, es podrien tenir subclasses de préstec.
- Els préstecs tenen, a més del saldo, una quota, que falta com a atribut de la classe *Prestec* mencionada abans.
- Els moviments de les pantalles tenen dues dates: una d'operació i una de valor. Per tant, en lloc de l'atribut *data*, la classe *Moviment* hauria de tenir els atributs *dataOperacio* i *dataValor*. Tot i que les pantalles enumeren un saldo després de cada moviment i que aquest es podria representar al model del domini, aquest saldo es pot derivar i, per tant, no es pot considerar incorrecte el fet que no aparegui explícitament com a atribut de la classe *Moviment*.
- Hi ha molta informació al model del domini que no apareix a les pantalles. Segurament no és incorrecta, però s'hauria de verificar amb la resta de pantalles. És el cas, per exemple, de la majoria dels atributs de la classe *Usuari*, de diversos atributs de *Compte*, del compte de càrrec dels crèdits, etc.

Exercicis d'autoavaluació

1. El llenguatge UML és el llenguatge estàndard per a crear models visuals de programari. (Vegeu el subapartat 1.2.1.)

2. Els tres usos principals per al llenguatge UML són: com a llenguatge per a fer un croquis, com a llenguatge per a fer un plànol i com a llenguatge de programació. (Vegeu el subapartat 1.2.1.)

3. El diagrama de casos d'ús complementa, però en cap cas no substitueix, la descripció textual dels casos d'ús, ja que no inclou informació sobre quin és el comportament del sistema. (Vegeu el subapartat 2.1.)

4. El punt d'extensió és un text que identifica en quin punt de l'escenari principal es produeix l'esdeveniment que provoca el comportament alternatiu. (Vegeu el subapartat 2.1.6.)

5. Ho podem fer mitjançant la condició de guarda i, opcionalment, una decisió (que es representa mitjançant un rombe). (Vegeu el subapartat 2.2.1.)

6. Tenim tota una sèrie de requisits, com ara la usabilitat o l'arquitectura d'informació, que, en ser els casos d'ús independents de la interfície d'usuari, no queden recollits en aquest model. (Vegeu l'apartat 3.)

7. S'anomenen *casos d'ús essencials* els casos d'ús que descriuen la interacció entre actors i sistema de manera independent de la tecnologia i de la implementació. (Vegeu el subapartat 3.1.)

8. L'objectiu dels models de les pantalles és deixar clar quina informació es mostra, la distribució de la informació a la pantalla, quines accions pot prendre l'usuari a partir de la informació mostrada, que és el procés que segueix l'usuari per a completar el cas d'ús. (Vegeu l'apartat 3.2.)

9. El mapa navegacional és un model que ens dóna informació sobre quin és el flux entre pantalles que pot seguir l'usuari. (Vegeu el subapartat 3.2.1.)

10. Un model del domini és una representació de classes conceptuals del món real en un domini d'interès. (Vegeu l'apartat 4.)

11. Coad proposa: participants, llocs i coses, rols, moments o intervals i descripcions. (Vegeu el subapartat "Identificació de classes del domini" dins del 4.2.2.)

12. Fer servir la terminologia que fan servir les persones i experts del domini, excloure aspectes irrellevants del domini i no afegir res que no sigui realment al domini que analitzem. (Vegeu el subapartat "Nomenclatura de les classes" dins del 4.2.2.)

13. Indica que és opcional (mínim 0 valors) i multivaluat (màxim * valors). (Vegeu el subapartat 4.3.1.)

14. Tot i que tant els atributs com els extrems d'associació són propietats de la classe, la notació de les associacions és més visual i, per tant, la preferim a la dels atributs per a representar la relació entre dues classes. (Vegeu l'apartat 4.4.2.)

15. Ens hem de plantejar l'especialització si hi ha un subconjunt de les instàncies de la classe tal que té atributs o associacions addicionals d'interès que no tenen totes les instàncies de la classe examinada, o bé es comporten diferent de manera rellevant. (Vegeu el subapartat "Identificació de l'herència: generalització i especialització" dins del 4.5.2.)

16. Són el conjunt de regles que han de complir les dades que emmagatzema un sistema per tal d'evitar situacions en què la informació és contradictòria o incompleta. (Vegeu el subapartat 4.6.1.)

17. Els atributs o associacions derivats són aquells tals que el seu valor es pot deduir a partir d'informació ja modelitzada. (Vegeu el subapartat 4.6.2.)

18. Els valors d'un tipus de dades, a diferència dels objectes, no tenen identitat única. Per tant, es comparen per valor i no pas per identitat. (Vegeu el subapartat 4.7.1.)

Glossari

bifurcació *f* Element gràfic del diagrama d'activitats d'UML representat com una línia horitzontal gruixuda amb un flux d'entrada i múltiples fluxos de sortida. Quan s'hi arriba, tots els fluxos de sortida es produeixen de manera paral·lela.

cardinalitat (d'un atribut) *f* Nombre de valors que una determinada instància té en un atribut.

cardinalitat (d'una associació) *f* Nombre d'instàncies que una determinada instància té associades per mitjà d'una associació concreta.

carril *m* Element gràfic del diagrama d'activitats d'UML que permet organitzar les activitats segons quin actor del procés les du a terme.

cas d'ús concret *m* Cas d'ús que descriu la interacció entre actors i sistema tenint en compte la tecnologia i la implementació. Poden contenir, per exemple, detalls sobre com l'actor fa servir la interfície oferta pel sistema.

cas d'ús essencial *m* Cas d'ús que descriu la interacció entre actors i sistema de manera independent de la tecnologia i de la implementació.

clau (d'una classe de domini) *f* Atribut o conjunt d'atributs que identifica les instàncies de la classe de manera única, de tal manera que no hi pot haver més d'una instància amb els mateixos valors en aquell o aquells atributs.

composició *f* Associació amb un fort sentit tot-part, de tal manera que si destruïm una instància composta tots els seus components també es destrueixen, no hi pot haver instàncies de la classe component que no formin part d'una i només una instància composta i, finalment, una instància component no pot canviar d'un compost a un altre.

contracte (d'una operació) *m* Documentació detallada d'una operació que inclou la seva signatura i les precondicions i postcondicions.

decisió *f* Element gràfic del diagrama d'activitats UML representat com un rombe i que representa una presa de decisió.

diagrama d'activitats *m* Diagrama estàndard d'UML utilitzat per a representar processos.

diagrama d'estats *m* Diagrama estàndard d'UML utilitzat per a modelitzar estats i transicions entre aquests.

diagrama d'estructura composta *m* Diagrama estàndard d'UML utilitzat per a modelitzar l'estructura interna en temps d'execució d'una classe o component.

diagrama d'objectes *m* Diagrama estàndard d'UML utilitzat per a representar objectes.

diagrama de casos d'ús *m* Diagrama estàndard d'UML utilitzat per a modelitzar els casos d'ús del sistema, els actors i les relacions entre aquests.

diagrama de classes *m* Diagrama estàndard d'UML utilitzat per a modelitzar les classes d'objectes i les seves relacions.

diagrama de col·laboració *m* Nom del diagrama de comunicació a la versió 1 d'UML.

diagrama de components *m* Diagrama estàndard d'UML utilitzat per a modelitzar els components que formen part d'un sistema.

diagrama de comunicació *m* Diagrama estàndard d'UML utilitzat per a modelitzar la interacció entre diversos objectes fent èmfasi en l'aspecte estructural.

diagrama de desplegament *m* Diagrama estàndard d'UML utilitzat per a modelitzar la distribució física dels diferents artefactes de programari en temps d'execució.

diagrama de paquets *m* Diagrama estàndard d'UML utilitzat per a representar els paquets i les dependències entre aquests.

diagrama de seqüència *m* Diagrama estàndard d'UML utilitzat per a modelitzar la interacció entre diversos objectes fent èmfasi en la seqüència temporal.

diagrama de temps *m* Diagrama estàndard d'UML utilitzat per a modelitzar el comportament dels objectes al llarg del temps fent èmfasi en les restriccions temporals.

diagrama de visió general d'interacció *m* Diagrama estàndard d'UML que combina els diagrames d'activitats i els de seqüències.

enumeració *f* Tipus de dades en què el conjunt de valors és una llista finita de valors que s'enumera en la definició del tipus de dades.

fusió *f* Element gràfic del diagrama d'activitats d'UML representat com un rombe amb diversos fluxos d'entrada i només un de sortida. El flux de sortida es produeix quan s'arriba per algun dels fluxos d'entrada.

herència dinàmica *f* Herència en què una instància d'una classe, al llarg de la seva vida, pot canviar d'una classe de l'herència a una altra.

herència *overlapping* *f* Herència en què una mateixa instància de la superclasse pot pertànyer a més d'una subclasse alhora.

mapa navegacional *m* Model que documenta els fluxos entre pantalles de la interfície gràfica d'usuari. Es representa mitjançant un diagrama d'estats UML en què cada pantalla és un estat i cada transició representa la transició entre pantalles produïda per un esdeveniment, típicament un gest de l'usuari.

multiplicitat *f* Restricció que indica quines cardinalitats són vàlides per a un atribut o associació.

navegabilitat *f* Propietat d'una associació binària que fa que només estigui definida en un sentit i que, per tant, només defineixi una propietat en una de les dues classes que hi participen. L'altra classe continuarà ignorant de l'existència de l'associació.

object constraint language *m* Llenguatge formal estàndard de restriccions: llenguatge estàndard d'UML per a l'expressió de restriccions que es pot fer servir, entre d'altres coses, per a escriure precondicions i postcondicions dels contractes de les operacions.
Sigla **OCL**

Object Management Group *m* Consorci americà sense ànim de lucre, creat el 1989, que té per objectiu l'estandardització i promoció de la modelització orientada a objectes.
Sigla **OMG**

postcondició (d'una operació) *f* Obligació a què es compromet el sistema quan s'invoca una operació en forma de condició que es compromet que sigui certa un cop executada l'operació.

precondició (d'una operació) *f* Condició que s'ha de satisfer necessàriament en el moment d'invocar l'operació. Si la condició no és certa, el sistema rebutja la petició i l'operació no s'executa.

propietat (d'una classe) *f* Nom genèric per a referir-se a atributs i a extrems d'associació de la classe.

punt d'extensió *m* Punt concret en la seqüència de passos d'un escenari (principal o extensió) d'un cas d'ús al qual es dóna un nom per tal que els casos d'ús que l'estenen puguin definir aquell punt com a punt on entra en joc l'extensió. En desús.

signatura (d'una operació) *f* Nom i informació sobre les dades que rep (paràmetres d'entrada) i la informació que retorna.

tipus de dades *m* Conjunt de valors per als quals la identitat única no té sentit, sinó que es comparen per valor.

unió *f* Element gràfic del diagrama d'activitats d'UML representat com una línia horitzontal gruixuda amb múltiples fluxos d'entrada i un flux de sortida. El flux de sortida només es produeix quan han arribat tots els fluxos d'entrada.

Bibliografia

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language*. Addison-Wesley.

Aquesta obra és una molt bona introducció curta a l'UML. Té un enfocament pragmàtic més que no pas normatiu i és molt aclaridora. Mostra l'ús de tots els diagrames, amb exemples, i dóna consells personals de l'autor sobre com i quan cal utilitzar cada part de l'especificació. Evidentment, però, no entra en detalls.

Larman, C. (2005). *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Prentice Hall.

Larman cobreix, en aquesta obra, l'anàlisi i el disseny de sistemes informàtics fent servir el llenguatge UML. Dóna un enfocament àgil i un altre d'UP, i el llibre està escrit de manera iterativa i incremental. Cobreix tot el que s'explica en aquest mòdul i força més, ja que entra en detalls de disseny i implementació.

Bibliografia complementària

Booch, G.; Rumbaugh, J.; Jacobson, I. (2005). *The unified modeling language user guide*. Addison-Wesley.

En aquesta obra els autors d'UML escriuen una guia d'ús del llenguatge de caràcter didàctic, més que no pas de referència. L'obra se centra en el llenguatge UML, però ens dóna consells i suggeriments sobre quan cal usar una part de l'estàndard o una altra.

Coad, P.; Lefebvre, E.; De Luca, J. (1999). *Java Modeling in Color With UML: Enterprise Components and Process*. Prentice Hall.

Tot i que ja fa força anys que va ser publicat i malgrat que el seu títol dóna a entendre que es tracta d'un llibre tècnic, aquesta obra tracta, bàsicament, sobre la modelització de sistemes d'informació fent servir orientació a objectes. Proposa un enfocament basat en un model d'anàlisi que es reutilitza adaptant-lo a les necessitats concretes de cada problema.

Cockburn, A. (2001). *Writing Effective Use Cases*. Addison-Wesley.

Tot i que no aprofundeix en l'ús d'UML, Cockburn continua essent la bibliografia de referència bàsica pel que fa a casos d'ús. Respecte d'aquest mòdul és especialment interessant la discussió que fa sobre relacions entre casos d'ús i sobre l'abast i nivell dels casos d'ús.

Diversos autors (2010). *Unified Modeling Language™ (UML®)*. Object Management Group.

Aquesta és l'especificació d'UML. Tot i el seu llenguatge formal i normatiu és, evidentment, la font més fiable en cas de dubte sobre el llenguatge UML.

Referències bibliogràfiques

Fowler, M. (1997). *Analysis Patterns. Reusable Object Models*. Addison-Wesley.

Fowler proposa l'ús del concepte de patró per a reutilitzar solucions d'anàlisi sobretot pel que fa a la modelització del domini. A part de proporcionar alguns patrons realment útils a l'hora de modelitzar sistemes d'informació, també és interessant veure els exemples de models que elabora aplicant aquests mateixos patrons.

Génova, G.; Lloréns, J.; Martínez, P. (2001). "Semantics of the minimum multiplicity in ternary associations in UML". *The 4th International Conference on the Unified Modeling Language*.

http://www.ie.inf.uc3m.es/ggenova/pub-uml2001.html

En aquest article els autors diuen que l'especificació d'UML és ambigua a l'hora d'explicar la cardinalitat de les associacions no binàries i proposen una clarificació.