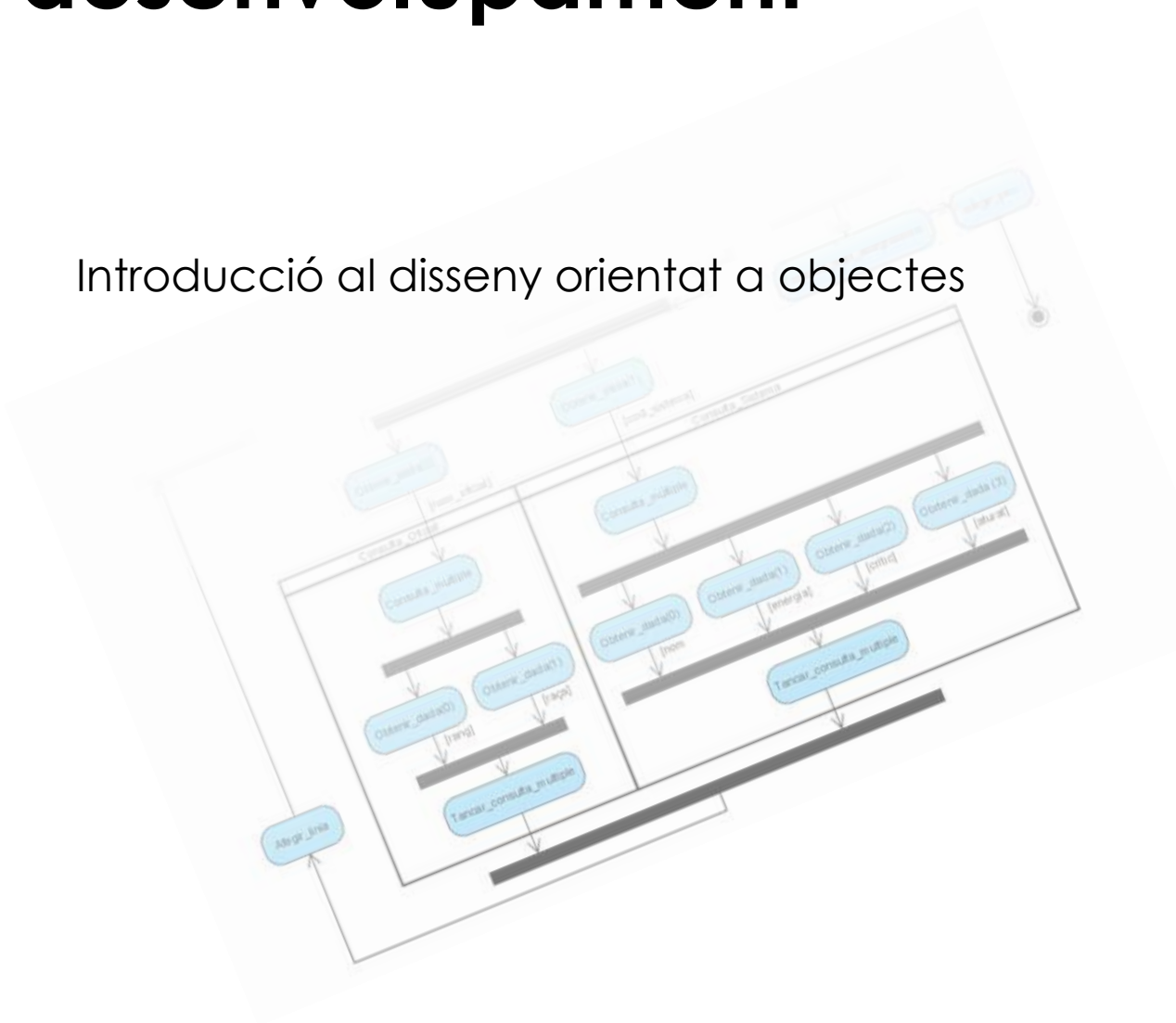
Desenvolupament d'Aplicacions Web

MANEL DÍAZ LLOBET

Mòdul 05
Entorns de desenvolupament

Introducció al disseny orientat a objectes



Contenido

1. EL LLENGUATGE UML I EL PATRÓ MVC

1.1. Definició:

UML (Unified Modelling Language) és un llenguatge per modelar/dissenyar aplicacions informàtiques. Les seves característiques principals són:

- **Basat en diagrames**: Es dissenya el sistema informàtic mitjançant diferents diagrames que aporten visions des de diferents punts de vista del sistema. Aquests diagrames estan formats per símbols i text que tenen significats específics.
- **Estàndard**: Hi ha varies versions de UML, totes compatibles entre si. Unifica diferents sistemes de disseny anteriors.
- **Orientat a Objectes**: tot el disseny es fa orientat a objectes per que la seva posterior implementació amb un llenguatge de programació sigui molt fàcil.

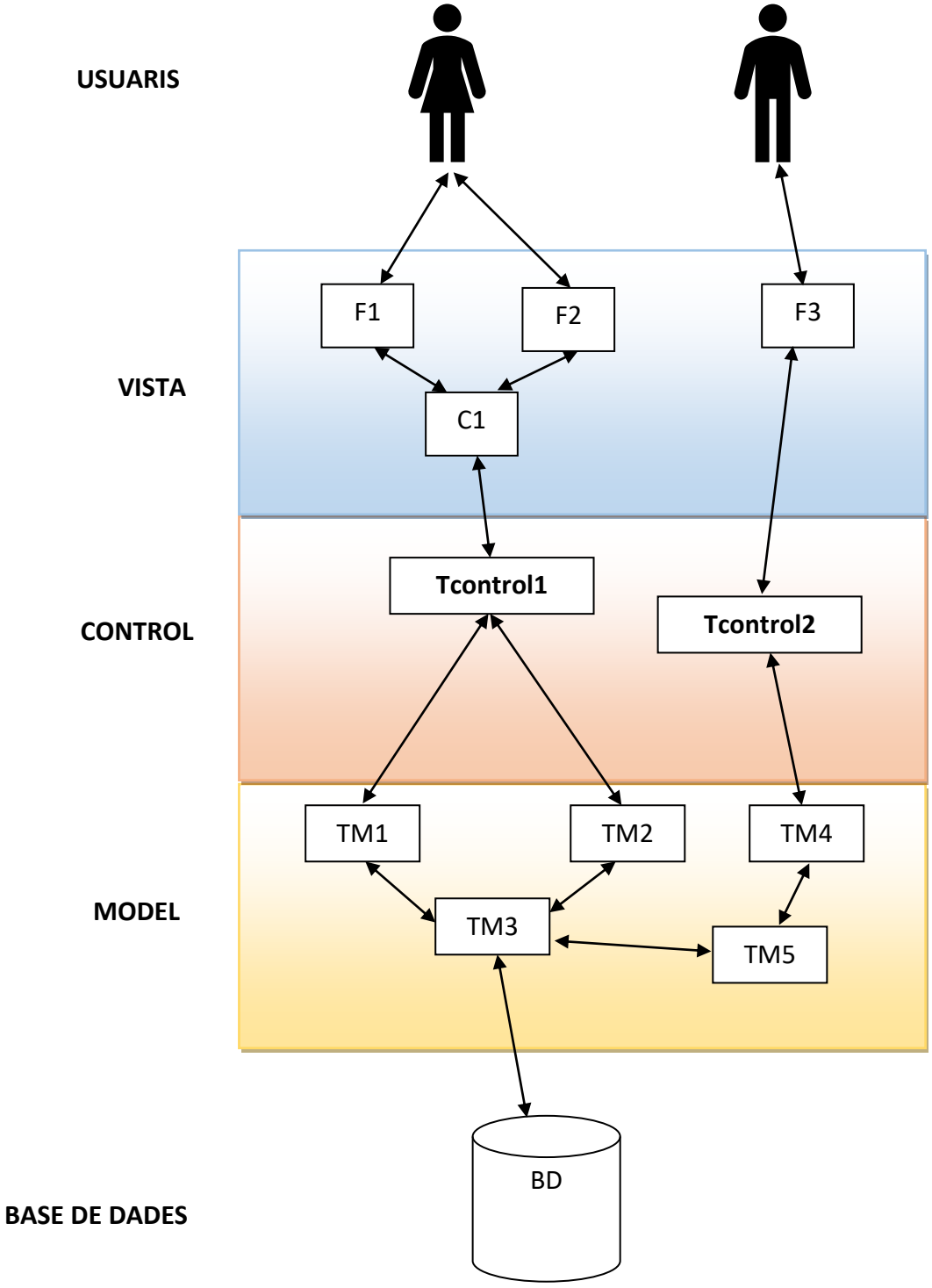
Hi ha diferents maneres de dissenyar amb UML.

La manera que veurem a classe serveix per aplicacions complexes de gestió, en les que la velocitat no és la prioritat.

Dintre de UML hi ha moltes maneres diferents de dissenyar. Nosaltres farem servir el patró de disseny **MVC (Model – Vista – Control)**, també anomenat **MODEL DELS TRES ESTEREOTIPS:**

1.2. Model dels tres estereotips

- **Classes de MODEL**: totes les classes necessàries per a accedir a la base de dades. Inclou també una classe especial per a centralitzar els accessos a la base de dades que han de fer les classes encarregades de gestionar les taules.
- **Classes de VISTA**: totes les classes necessàries per a comunicar-se amb la interfície d'usuari. Són les classes que implementen els filtres de dades introduïts per l'usuari i apliquen els formats necessaris a les sortides de dades. Son les finestres on l'usuari interactua i els mòduls de codi necessaris per gestionar aquesta interactuació.
- **Classes de CONTROLADOR**: totes les classes necessàries per a coordinar les peticions realitzades per les classes de VISTA, que reben peticions directament dels usuaris, i els mètodes de les classes de MODEL, que implementen la solució a aquestes peticions.


USUARIS
VISTA
CONTROL
MODEL
BASE DE DADES
F1
F2
F3
C1
Tcontrol1
Tcontrol2
TM1
TM2
TM4
TM3
TM5
BD

1.3. Funcionament de MVC

El funcionament del patró de disseny és el següent:

USUARIS: Es comunicant amb les finestres del estereotip VISTA. Trien la funcionalitat desitjada i introdueixen la informació necessària per realitzar aquesta funcionalitat. Al final del procés reben la informació produïda per la funcionalitat i un missatge de feedback informatiu.

VISTA: Consta de tots els fitxers relacionats amb la interfície d'usuari. Aquests fitxers són tant les finestres que es mostren a l'usuari, com els mòduls de codi associats a la gestió d'aquestes finestres. Rep les peticions dels usuaris i traslladen aquestes peticions a CONTROL

CONTROL: Rep les peticions de VISTA i analitza quina de les classes de MODEL és la més adient per realitzar la tasca encarregada. Una vegada ho sap, reenvia la petició a la classe de MODEL adient. Quan la classe de model ha realitzat la seva feina, rep la informació d'ella i la traspassa a la classe de VISTA que li ha demanat. Pot haver vàries classes a CONTROL (tcontrolStocs, tcontrolVendes, tcontrolClients) si el sistema informàtic disposa de varis subsistemes.

MODEL: Hi ha totes les classes que fan la feina efectiva. Cadascuna de les classes s'encarregarà de la gestió d'una taula de la base de dades. Les classes col·laboren entre elles per resoldre les peticions que fa CONTROL. Per tal d'accedir ordenadament a la base de dades, hi ha una classe encarregada de totes les funcionalitats que tenen a veure amb aquest accés. Una vegada la classe de MODEL encarregada de realitzar la funcionalitat demanada per CONTROL ha terminat la seva feina, li retorna els resultats a CONTROL.

Model de desenvolupament

UML consta d'una sèrie de diagrames. El mètode que farem servir no és el mètodes en cascada, si no el model en prototipatge, és a dir, es fa primer una primera versió del diagrama i després una segona versió que pot corregir la primera.

Es programa des del punt de vist del usuari (**diagrama de casos d'ús**), que vol el client. Després veure quines classes necessitem per poder fer el que vol el client (**diagrama de classes**). Després un **diagrama de flux** que ens descriu els passos pels que passa cada classe i com funciona cada classe.

2. DIAGRAMES DE UML

A UML hi ha diagrames que obligatòriament s'han de fer, i d'altres que són opcionals. S'han de fer els diagrames necessaris per que el sistema informàtic a implementar quedi totalment dissenyat. S'ha de començar a fer el disseny de les funcionalitats crítiques del programa, o sigui, el nucli de l'aplicació.

Per tal d'explicar millor tots els diagrames de UML d'aquest document es farà el disseny parcial d'una aplicació informàtica.

L'especificació de l'aplicació d'exemple és la següent:

ENUNCIAT

Es vol fer un disseny d'una aplicació per a la gestió d'alumnes i les seves notes. L'especificació del sistema és la següent.

L'administrador del sistema pot donar d'alta a nous alumnes. Les dades que es necessiten de cada alumne són el seu DNI, el seu nom complet, la seva adreça i la població on viu. Totes aquestes dades són obligatòries. No es pot donar d'alta a dos alumnes amb el mateix DNI.

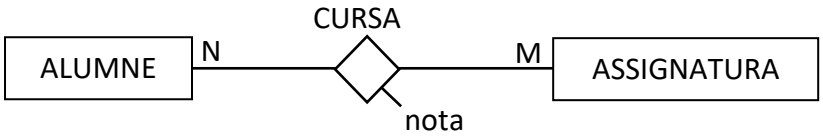
D'altre banda, l'administrador també pot donar de baixa un alumne, a partir del seu DNI. Cal comprovar que efectivament, l'alumne que es vol donar de baixa, estava prèviament donat d'alta a la base de dades.

L'administrador del sistema també ha de poder modificar les dades d'un alumne que s'han indicat prèviament (totes menys el DNI).

Un alumne pot consultar les seves dades. Per a això cal que introdueixi el seu DNI. Se li mostrà llavors totes les seves dades (nom complet, adreça i població). Si l'alumne te notes, també se li mostrarà un llistat amb totes les notes de totes les seves assignatures.

El model de dades del sistema és el següent:

Model conceptual EER:



Model lògic relacional:

```
ALUMNE (dni, nomComplet, adreça)
ASSIGNATURA (codi, nomAssignatura, horesSetmanals)
CURSA (dniAlumne, codiAssignatura, nota)
      dniAlumne és clau forania cap a dni d'ALUMNE
      codiAssignatura és clau forània cap a codi d'ASSIGNATURA
```

Model físic (SQL)

```
CREATE TABLE ALUMNE
(
      DNI char(9),
      nomComplet varchar(150),
      adreça varchar(150),
      PRIMARY KEY (DNI)
);
CREATE TABLE ASSIGNATURA
(
      codi char(8),
      nomAssignatura varchar(150),
      horesSetmanals integer,
      PRIMARY KEY (codi)
);
CREATE TABLE CURSA
(
      dniAlumne char(9),
      codiAssignatura char(8),
      nota integer,
      PRIMARY KEY (dniAlumne, codiAssignatura),
      FOREIGN KEY (dniAlumne) REFERENCES ALUMNE(DNI),
      FOREIGN KEY (codiAssignatura) REFERENCES ASSIGNATURA (codi)
);
```

2.1. Diagrama de casos d'ús

2.1.1. ***Funció:***

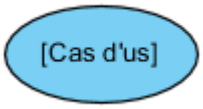
Serveix per dissenyar les funcionalitats de l'aplicació, els tipus d'usuari que la faran servir i un primer nivell de detall de la lògica interna de l'aplicació.

2.1.2. ***Informació necessària:***

La informació necessària per realitzar un diagrama de casos d'us surt dels requeriments funcionals de procés de l'especificació. Hi ha d'haver un cas d'us de primer nivell per cadascun dels requeriments funcionals de procés.

També és necessari conèixer la ***base de dades*** per poder dissenyar els casos d'us de segon nivell i superiors.

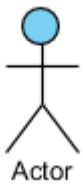
2.1.3. ***Elements del diagrama:***



CAS D'US: Funcionalitat requerida per un usuari o per un altre cas d'us, que resol un problema concret.

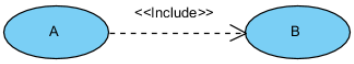
CAS D'US DE PRIMER NIVELL: Són els casos d'us directament connectats amb un ACTOR. Corresponen amb les opcions de menú de l'aplicació i amb tots els requeriments funcionals de procés de l'especificació

CAS D'US DE SEGON NIVELL I SUPERIORS: Són tots els casos d'us que no estan directament connectats amb els actors.

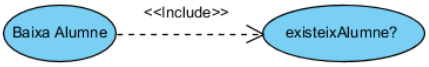


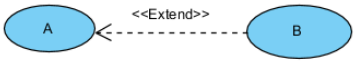
ACTOR: Perfil d'usuari de l'aplicació. Persona, sistema o altres que utilitza el sistema que s'està dissenyant. És una entitat externa que es comunica amb el sistema.
Els actors no es comunicant entre ells dintre del disseny del sistema. Tota la interacció entre actors (si és que hi és) es fa a través dels casos d'us.

CONNECTOR: Connecta als actors amb els casos d'us. No són dirigits (no tenen punta de fletxa) per que la comunicació entre els actors i els casos d'us és bidireccional

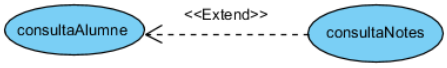


INCLUDE: Connecta dos casos d'us amb una relació d'inclusió. Dos casos d'us A i B estan relacionats amb un <include> quan A SEMPRE necessita a B per funcionar. Per exemple, per poder donar de baixa a un alumne, abans cal comprovar SEMPRE que l'alumne existeix a la base de dades:





EXTEND: Connecta dos casos d'us amb una relació d'extensió. Dos casos d'us A i B estan relacionats amb un <extend> quan A necessita A VEGADES a B per funcionar. Un dels casos on es far servir una relació <extend> és quan cal fer un tractament especial si es produeix un error. Per exemple, quan es consulta les dades d'un alumne, si aquest te notes, consultar també les seves notes:



2.1.4. ***Com es fa?***

Per tal de fer un diagrama de casos d'us, és necessari disposar de la llista de requeriments funcionals de procés de l'especificació a dissenyar.

- ✓ Per a cada perfil d'usuari que pugui fer servir l'aplicació, crear un actor.
- ✓ Per a cada requeriment funcional de procés, crear un cas d'us de primer nivell i relacionar-lo amb l'actor corresponent.
- ✓ Per a cada cas d'us de primer nivell, analitzar per sobre que funcionalitats addicionals obligatòries necessita per poder realitzar-se i fer els casos d'us de segon nivell adients, tot relacionant-los amb el cas d'us de primer nivell amb <<include>>
- ✓ Per a cada cas d'us de primer nivell, analitzar per sobre que funcionalitats addicionals ocasionals necessita per poder realitzar-se i fer els casos d'us de segon nivell adients, tot relacionant-los amb el cas d'us de primer nivell amb <<extend>>
- ✓ Dos casos d'us no poden tenir el mateix nom. En aquest cas, es dibuixa només un cas d'us i es fan arribar les relacions necessàries.
- ✓ Si per qüestions de claredat al diagrama cal repetir un cas d'us, el segon cas d'us portarà un * al començament del seu nom.

Analitzant l'especificació del nostre cas, tenim:

Segons l'enunciat, hi ha dos actors o usuaris:

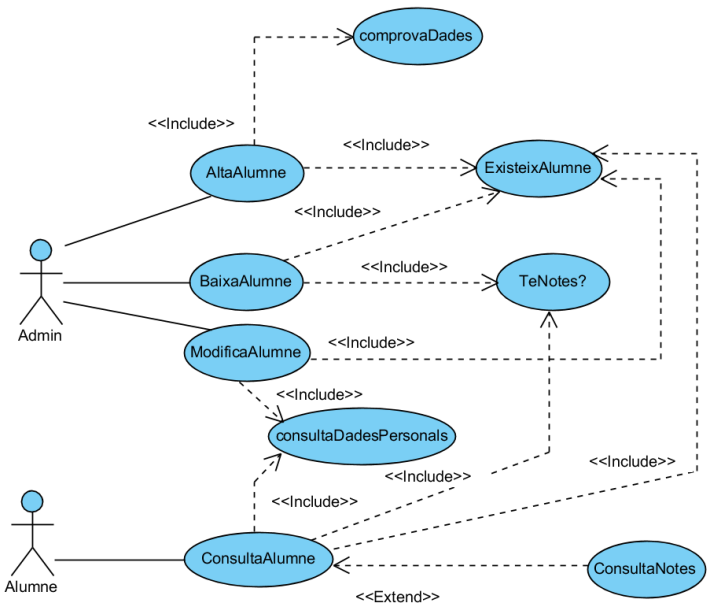
- Administrador: Pot donar d'alta alumnes, donar-los de baixa i consultar les seves dades personals. No pot consultar notes d'alumnes.
- Alumne: Només pot consultar les seves dades i si te notes, també les seves notes.

Ara, cal analitzar cadascuna de les funcionalitats a veure quines restriccions o regles tenen:

- altaAlumne: només es pot donar d'alta un alumne si s'han introduït totes les dades i l'alumne no està prèviament donat d'alta.
- baixaAlumne: per poder donar de baixa un alumne, cal comprovar que aquest estigui a la base de dades i no tingui cap nota.
- consultaAlumne: Si l'alumne està a la base de dades, es mostraran totes les seves dades personals. Si a més l'alumne te notes, també es mostrarà un llistat amb totes les notes de les seves assignatures.
- modificaAlumne: Per poder modificar les dades d'un alumne, primer cal mostrar per pantalla les seves dades personals. Si l'alumne no està a la base de dades, s'ha de donar un error.

Vist aquest anàlisi el diagrama de casos d'ús haurà de tenir:

- 2 actors
- 4 casos d'us de primer nivell
- 5 casos d'us de segon nivell o superior.



En aquest diagrama es pot veure tot l'anàlisi previ.

Com que hi ha 4 casos d'us de primer nivell, hi haurà 4 opcions al menú principal de l'aplicació.

En aquest diagrama no es pot veure l'ordre en que cal fer cadascun dels casos d'us de segon nivell o superior. Aquesta característica es visualitzarà en un altre diagrama.

2.2. Diagrama de classes

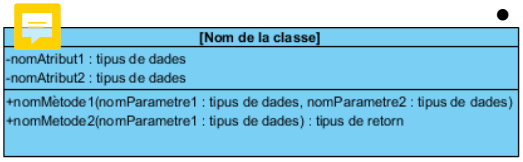
2.2.1. *Funció*

Mostra totes les classes necessàries per dissenyar tots els casos d'us. També mostra la relació que hi ha entre les classes del sistema. Aquestes relacions són de diferents tipus. Mostra l'estructura estàtica del sistema.

2.2.2. *Informació necessària*

La informació necessària per realitzar un diagrama de casos d'us surt dels ***requeriments funcionals de procés*** de l'especificació. També es necessita el ***diagrama de casos d'us***, ja que cada cas d'us haurà de ser dissenyat per al menys, un mètode d'una classe. Per tal de conèixer les classes a dissenyar, cal conèixer prèviament la ***base de dades*** (models conceptual, lògic i físic).

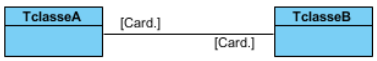
2.2.3. *Elements del diagrama*



CLASSE: Es representa amb un rectangle dividit en tres àrees:

- ***NOM***: Nom de la classe. Comença sempre per la lletra "T". A les classes encarregades de gestionar una taula de la base de dades, el nom de la classe és el mateix que el nom de la taula, precedit de la lletra "T". Altres noms especials de classes són TControl i TAccesbd.
- ***PROPIETATS***: llista de totes les propietats de la classe. Cal especificar el tipus de dades de cada propietat, així com la seva visibilitat (public, private, protected, ...)
- ***MÈTODES***: Llista de tots els mètodes de la classe, amb els seus paràmetres i tipus de dades dels paràmetres, tipus de dades de retorn en cas de les funcions i visibilitat dels mètodes (public, private, protected, ...)

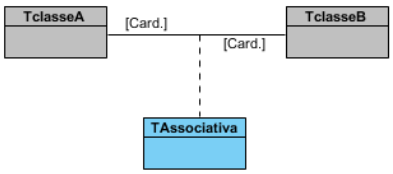
RELACIONS: És la forma en la que interactuen les classes per a dissenyar un cas d'us. Hi ha dels següents tipus:



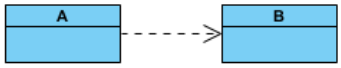
ASSOCIACIÓ: Dues classes estan relacionades mitjançant una associació (línia continua sense fletxes) si els entitats que representen a la base de dades també estan relacionades amb una interrelació la model EER. La cardinalitat de l'associació és la mateixa que la cardinalitat de la interrelació del model EER, però indicant si cal la cardinalitat mínima i la màxima.

Les cardinalitats d'una associació poden ser:

0, 0..1, 0..*, 1, 1..* o *; on * significa "molts"



CLASSE ASSOCIATIVA: És una classe que representa una relació binària N:M o una relació ternària del model EER de la base de dades. La classe es relaciona amb una relació d'associació entre altres dues classes (línia discontinua sense fletxes), per que només te sentit la seva existència si existeix també la relació entre les altres dues classes.



DEPENDÈNCIA: dues classes A i B estan relacionades mitjançant una relació de dependència (línia contínua amb una fletxa) si la classe A necessita a la classe B per funcionar. En altres paraules, si la classe A necessita fer un <include> de la classe B, A te una dependència de B.

2.2.4. Com es fa?

Per tal de fer un diagrama de classes, és necessari disposar de la llista de requeriments funcionals de procés de l'especificació a dissenyar. També cal tenir disponible tot el model de dades de l'aplicació, per que les classes de model es crearan gairebé totes, seguint aquest model de dades. En aquest document, només es farà el disseny de l'estereotip CONTROL i MODEL, deixant el disseny de l'estereotip VISTA per a un disseny de la interfície de l'aplicació.

2.2.4.1. PER A DISSENYAR LES CLASSES DE L'ESTEREOTIP **MODEL**:

- ✓ Per a cada **entitat** al model conceptual de la base de dades, cal crear una **classe** de MODEL.
- ✓ Per a cada **relació 1:N** del model conceptual de la base de dades (entitats febles incloses) cal crear una relació d'**associació** entre les classes corresponents, tot indicant la cardinalitat, segons el model de dades.
- ✓ Per a cada **relació N:M o ternària** del model conceptual de la base de dades, cal crear una **entitat associativa** per gestionar la taula intermèdia que surt al model lògic de la base de dades.
- ✓ Cada classe que es crea a MODEL te, com a mínim, tantes propietats com atributs te la taula que ha de gestionar a la base de dades. A més, aquestes propietats han de ser del mateix tipus que els atributs de la taula a la base de dades. Les classes poden tenir altres atributs diferents dels anteriors, per a gestions internes de la classe o l'aplicació.
- ✓ Cada classe de MODEL tindrà els mètodes necessaris per disenyar els casos d'us que li corresponguin. Per saber de quina classe de MODEL és un determinat mètode, cal fixar-se en la instrucció SQL que s'executa per fer la funció del mètode. Per exemple, el mètode ***altaAlumne*** executarà al final un INSERT INTO ALUMNE VALUES... , o sigui, que és un mètode de la classe ***TAlumne***
- ✓ Tots els mètodes de la classe TControl han d'estar en alguna de les classes de MODEL.
- ✓ Tots els casos d'us han d'estar en alguna de les classes de MODEL.

- ✓ Per tal de comunicar-se amb la base de dades de forma centralitzada, es crearà una classe anomenada **TAccesbd**, encarregada de fer la connexió i desconnexió amb la base de dades, i d'executar totes les instruccions SQL que demanin els diferents mètodes de les classes de MODEL.
- ✓ Totes les classes que facin servir a TAccesbd hauran de tenir una relació de dependència amb ella.

2.2.4.1.1 FUNCIONAMENT DE LA CLASSE TACCESBD

- Al **constructor** de la classe, cal passar-li les dades de connexió amb la base de dades. Comunament, aquestes dades són el nom del servidor, el nom de l'usuari de la base de dades, la paraula de pas i el nom de la base de dades. Aquests paràmetres poden canviar en funció del SGBD que es faci servir. Crea un objecte per poder connectar amb la base de dades.
- **connectarBD**: realitza la connexió amb la base de dades i retorna CERT si s'ha pogut realitzar. És un mètode intern a la classe.
- **consultaEnter, consultaString, consultaX**: fa una consulta d'una única dada del tipus de dades que indiqui la instrucció. Si cal fer la consulta d'un tipus de dades que no sigui ni enter ni string, caldrà programar un mètode nou (depenent del llenguatge). Retorna la dada consultada. Cal passar-li per paràmetre el SELECT a executar.
- **consultaMultiple**: inicia una consulta que ha de retornar més d'una dada. Una vegada realitzat amb èxit aquest mètode, la fila de dades actual és la primera fila de la consulta. Retorna CERT si s'ha pogut realitzar la consulta. Cal passar-li per paràmetre el SELECT a executar.
- **obtenirDada**: Mètode polimòrfic. Passant-li un índex o el nom d'un camp, retorna la dada situada en aquella posició o que tingui el nom del camp que se li passa, de la fila de dades actual. Retorna la dada actual.
- **seguentDada**: avança a la següent fila de dades. Retorna CERT si ha pogut avançar.
- **finalDades**: retorna CERT si s'ha arribat al final de les files de dades d'una consulta múltiple.
- **desconnectarBD**: desconnecta la connexió amb la base de dades. És un mètode intern a la classe.
- **executaSQL**: Executa una instrucció SQL que no sigui SELECT, o sigui, INSERT, UPDATE o DELETE. Retorna CERT si s'ha realitzat be l'execució de la instrucció.
- **El destructor** de la classe elimina l'objecte que connecta amb la base de dades.

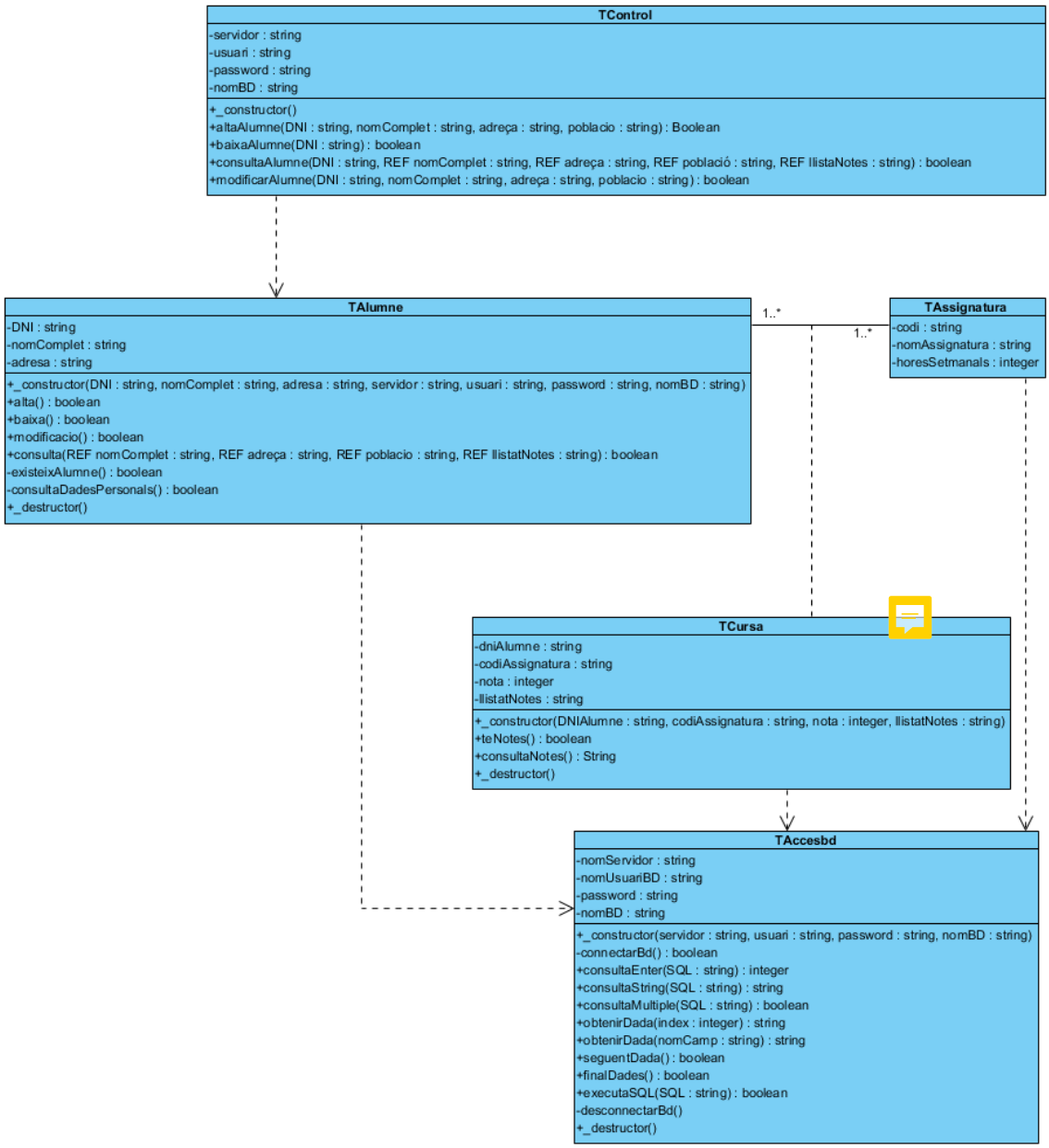
2.2.4.2. PER A DISSENYAR LES CLASSES DE L'ESTEREOTIP **CONTROL**:

- ✓ Si el sistema no és gaire complex, només hi haurà una classe a l'estereotip CONTROL. Aquesta classe serà **TControl** seguida d'una paraula identificativa del que controla.
- ✓ A la classe TControl en principi no hi ha cap propietat. Si es posa alguna propietat, seran les necessàries per al funcionament intern de l'aplicació. Per

exemple, hi pot haver les dades de connexió a la base de dades, per passar-les més tard a MODEL, i així tenir-les centralitzades en un sol punt.

- ✓ La classe TControl tindrà, com a mínim, tots els mètodes necessaris per disenyar els **cassos d'us de primer nivell** que hi hagi al diagrama de casos d'us. Pot tenir algun mètode més pel funcionament intern de l'aplicació.
- ✓ Per a dissenyar un cas d'us, TControl cridarà des d'un dels seus mètodes, al mètode de la classe encarregada del disseny del cas d'ús. Caldrà dibuixar una relació de dependència entre TControl i la classe afectada.

A l'exemple que estem seguint, aquest seria el diagrama de classes:



Regles bàsiques de gestió de dades:

ALTA: passar tota la informació que el sistema no pugui calcular o decidir.

BAIXA: passar només la informació necessària per identificar l'element a donar de baixa, és a dir, la clau primària de la taula.

CONSULTA: passar l'identificador de l'element a consultar i ***per referència*** la resta d'informació (dades) que es vulguin saber.

MODIFICACIÓ: passar l'identificador de l'element i totes les dades de l'element a modificar, ja siguin noves o velles.

2.3. Diagrames de Col·laboració (o de comunicació)

2.3.1. *Funció*

Mostren com col·laboren les diferents classes (cridant als seus mètodes) per tal d'implementar un cas d'us.
A un mateix disseny, hi ha molts diagrames de col·laboració. Al menys, hi ha d'haver un diagrama de col·laboració (o de seqüencia, com es veurà més endavant) per a cada cas d'ús de primer nivell.
Els diagrames de col·laboració mostren una execució particular d'un cas d'ús. Sempre cal fer un diagrama de col·laboració mostrant com funciona el sistema en el cas de que el cas d'ús es realitzi de forma correcta, sense errors. Si el cas d'us del que s'està fent el diagrama de col·laboració te un <<extend>>,caldrà fer un diagrama de col·laboració en cas de que no es faci l'<<extend>> i en cas de que si que es faci (*a l'exemple que s'està seguint, caldrà fer un diagrama de col·laboració en el cas de consultar un alumne que NO te notes, i un altre en el cas de consultar un alumne que SI te notes*).

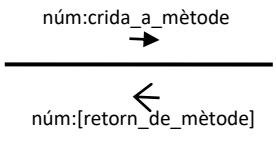
2.3.2. *Informació necessària*

La informació necessària per realitzar un diagrama de col·laboració surt dels ***requeriments funcionals de procés*** de l'especificació, per saber amb exactitud com dissenyar-lo. També es necessita el ***diagrama de casos d'us***, per veure els casos d'us de segon nivell i superiors que estan involucrats en el disseny. També intervé el diagrama de classes, ja que caldrà saber quins objectes de cada classe cal fer i quins mètodes (amb els seus paràmetres inclosos) cal crida en cada moment. Finalment, caldrà conèixer el model físic de la base de dades per tal de fer les instruccions SQL necessàries.

2.3.3. *Elements del diagrama*

ACTOR: És l'usuari que inicia el procés. A través de la interficie d'usuari (VISTA), crida a un dels mètodes de TControl

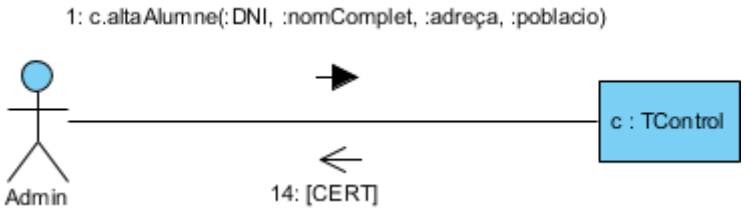
OBJECTE: és un objecte particular d'una classe. En aquest exemple l'objecte és ***alu1*** de la classe ***TAlumne***. Les classes són definició, els objectes, execució.



ENLLAÇ: Enllaça actors i objectes, o objectes entre si. Cada enllaç te una fletxa que indica una crida a un mètode, i una altre fletxa de retorn de la crida. Si el mètode és un procediment (no te cap valor de retorn) la fletxa de retorn es posa igualment, però sense informació. Tant les fletxes de crida

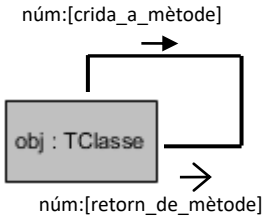
com de retorn, tenen un número d'ordre. Aquest número indica l'ordre d'execució de crides i retorns entre diferents mètodes dels objectes del diagrama.

Els mètodes sempre son de les classes a on apunta la fletxa de crida. Si la classe a on apunta la fletxa NO te el mètode que posa a la crida, llavors és un error. Aquest mètode ha de figurar al diagrama de classes, a la classe corresponent.

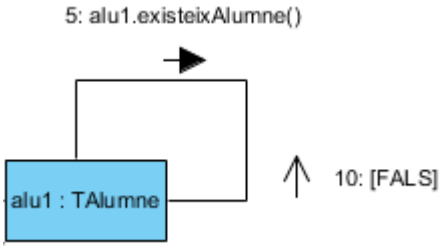


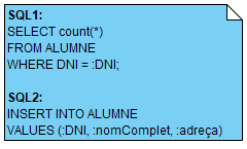
En aquest cas, tenim que l'actor ***ADMIN*** crida a un mètode de l'objecte ***c***, que és de la classe ***TControl***. Aquest mètode és ***altaAlumne***, i es crida amb els seus paràmetres. Com que és una crida, s'haurien de posar els valors reals dels paràmetres amb els que es crida. Per diferenciar aquests valors reals de la crida, de camps de la base de dades, es posa un símbol de dos punts (:) davant del nom del valor a la crida ***(:DNI, :nomComplet***, etc).

Com es pot veure, després de realitzar les crides i retorns que calgui per part de ***TControl***, aquest retornarà el resultat de l'execució del mètode ***altaAlumne***, que en aquest cas és ***[CERT]***



AUTOENLLAÇ: Un objecte pot cridar a mètodes de la seva pròpia classe. En aquest cas, es fa un enllaç entre la mateixa classe, per indicar que hi ha una crida i un retorn entre mètodes de la mateixa classe. En el cas de l'objecte ***alu1*** de la classe ***TAlumne***, per comprovar si un alumne està a la base de dades, és necessari cridar a ***existeixAlumne***, que es de la mateixa classe ***TAlumne***. En aquest cas, retorna ***[FALS]***



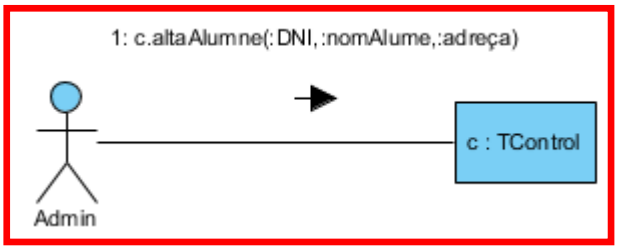


NOTA: És una nota informativa on s'han de posar totes les instruccions SQL que calgui executar durant l'execució dels mètodes del diagrama.

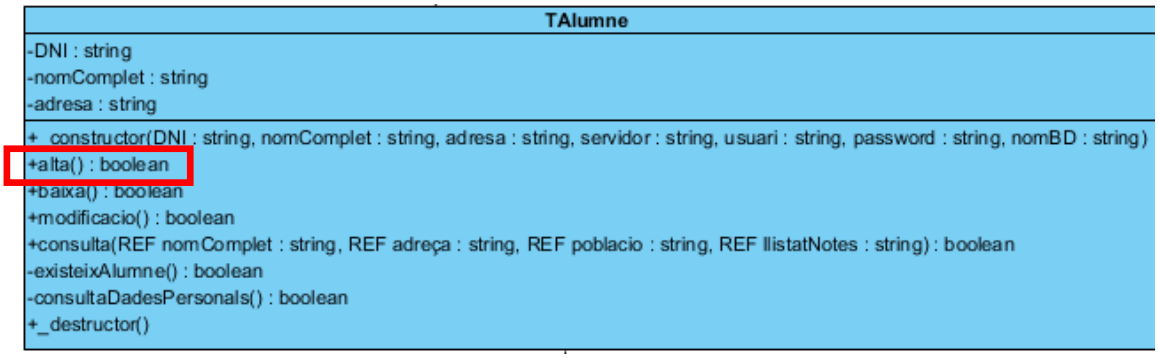
2.3.4. *Com es fa?*

En l'exemple que estem seguint el diagrama de col·laboració per al cas d'ús de primer nivell ***ALTA ALUMNE*** seria aquest:

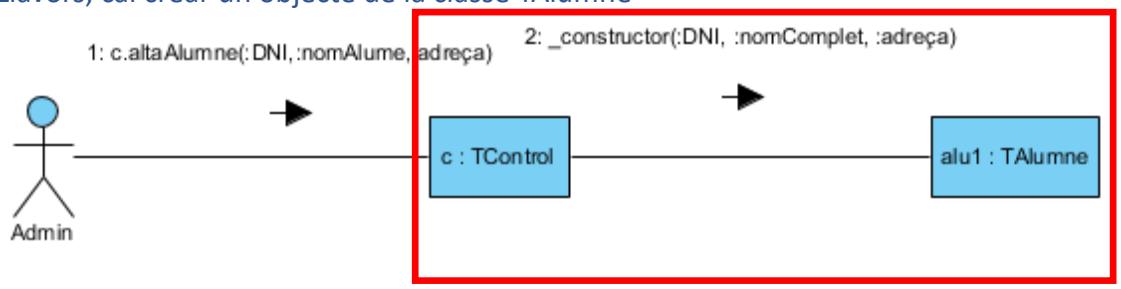
1.- Seleccionem l'actor que pot executar la funcionalitat, segons el diagrama de casos d'us i cridem al mètode de la classe TControl que implementa el cas d'ús altaAlumne. Per claredat del disseny, els mètodes de la classe TControl tenen el mateix nom que el cas d'us que implementen:



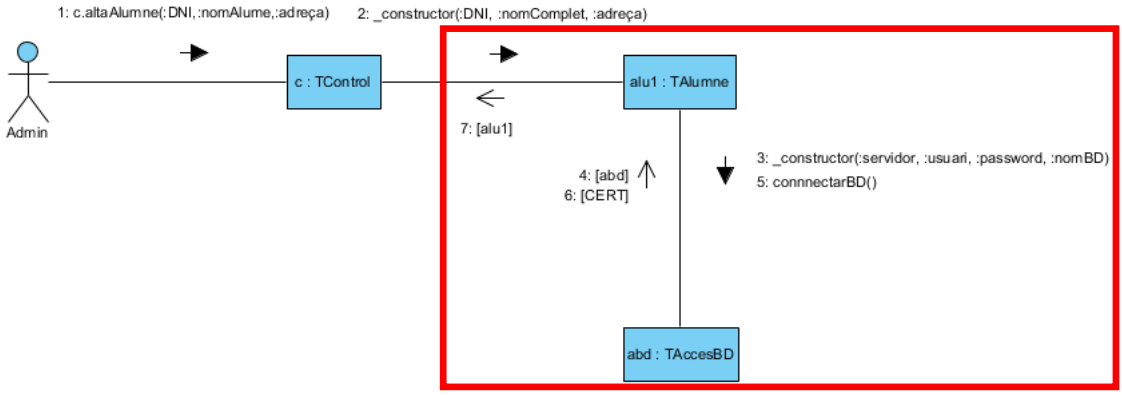
2.- En aquest moment, el codi de l'aplicació està dintre del mètode **altaAlumne() de TControl**. El següent pas és decidir quines de les classes de MODEL poden realitzar aquesta alta d'un alumne a la base de dades. Com es pot veure al diagrama de classes, la classe que pot fer aquest mètode és TAlumne:



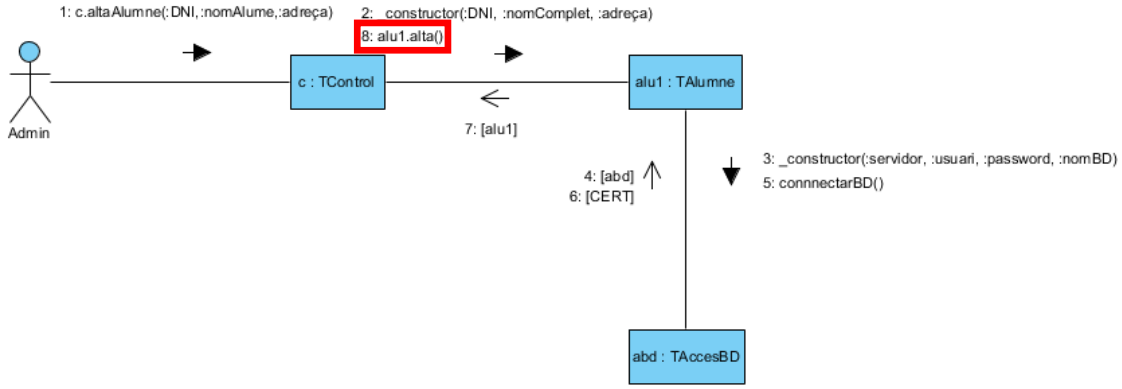
Llavors, cal crear un objecte de la classe TAlumne



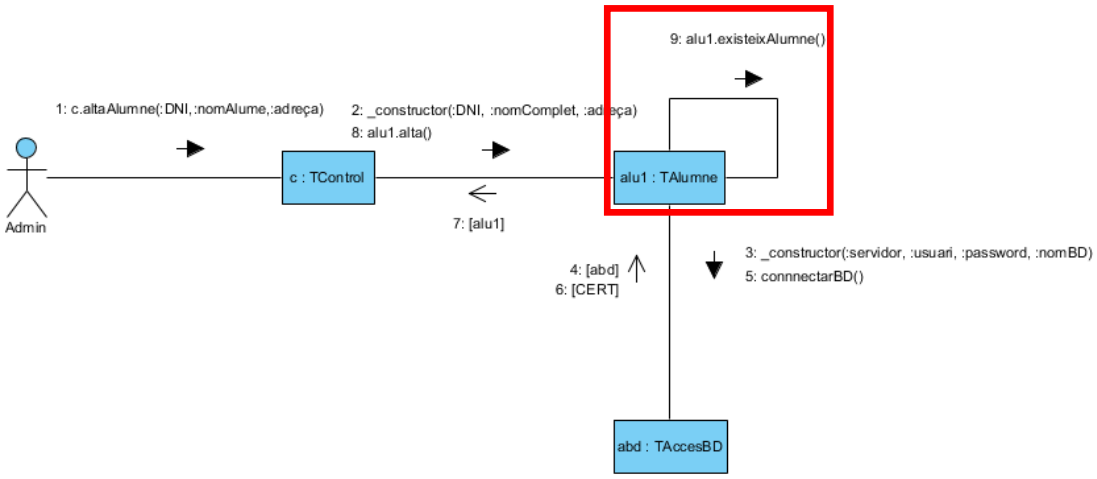
3.- El constructor de la classe TAlumne a la seva vegada crea un objecte per accedir a la base de dades (abd de la classe TAccesBD) i fa la connexió amb la base de dades. A partir d'aquest moment, ha es podran executar instruccions SQL. El constructor de la classe TAlumne retorna un objecte (abd):



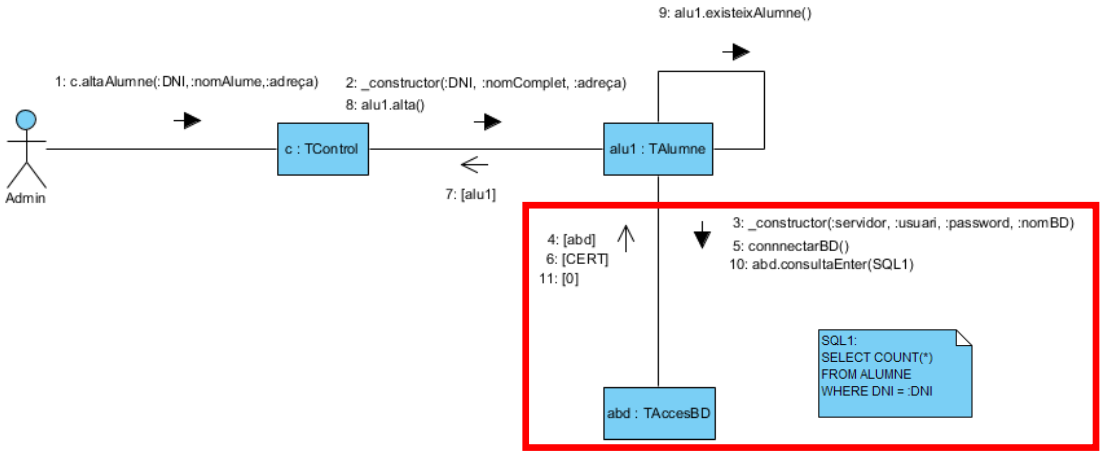
4.- En aquests moments, altaAlumne de TControl ja disposa d'un objecte alu1 de la classe TAlumne. Ja pot cridar al mètode alta():



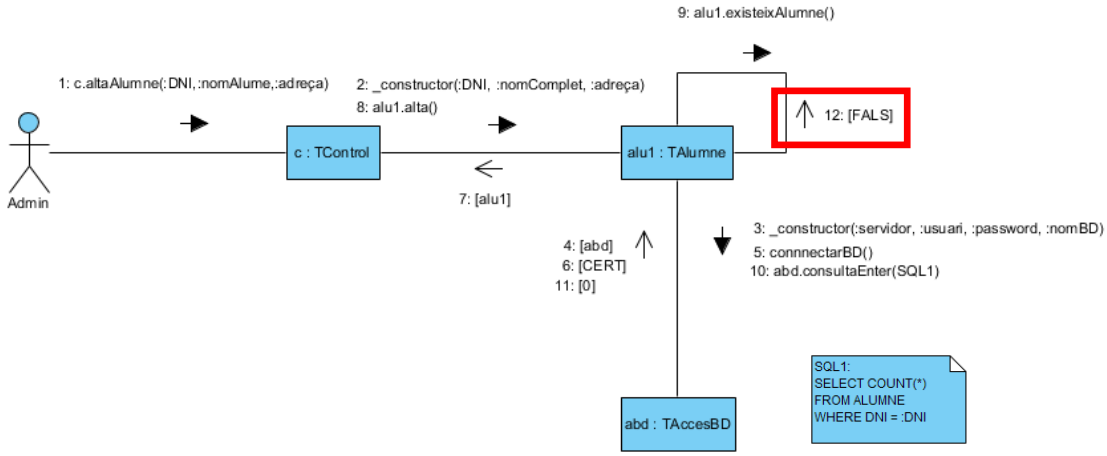
5.- Ara, el codi de l'aplicació està al mètode **alta() de la classe TAlumne**. Com es pot veure al diagrama de casos d'us, per poder donar d'alta un alumne a la base de dades, primer cal comprovar que l'alumne no existeixi a la base de dades. Per fer això, cal cridar al mètode existeixAlumne de la pròpia classe TAlumne:



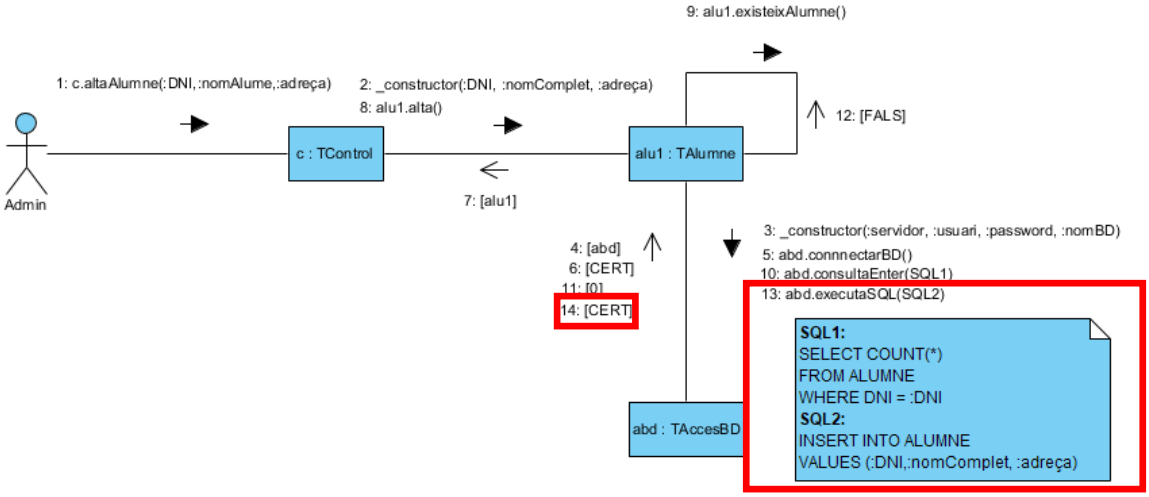
6.- El codi està dintre del mètode **existeixAlumne()**. Per comprovar l'existència (o no) de l'alumne a la base de dades, caldrà fer una consulta (SELECT). Cal cridar al mètode **consultaEnter() de TAccesBD.** Per saber si l'alumne existeix a la base de dades farem la següent instrucció: SELECT COUNT(*) FROM ALUMNE WHERE DNI = :DNI; Si retorna 0, és que no existeix.



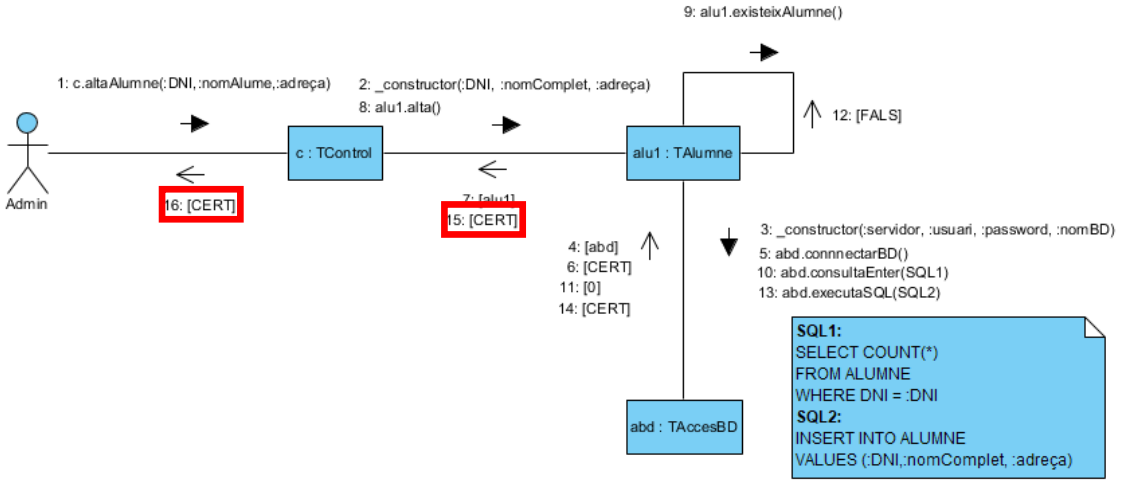
7.- El programa continua dintre del mètode **existeixAlumne() de TAlumne**. El resultat de fer la consulta a la base de dades és 0, amb el que aquest mètode ha de retornar FALSE al mètode que l'havia cridat, o sigui a alta() de TAlumne.



7.- Com que el mètode existeixAlumne() ja ha retornat el seu resultat i ha acabat, ara el codi està al mètode alta(), que ja sap que l'alumne no existeix. Per això, ara el mètode alta() ja pot executar la instrucció SQL que inserirà l'alumne a la base de dades. Cal executar el mètode executaSQL() de TAccesBD:



8.- Ja s'ha fet la inserció de les dades del nou alumne a alta(), amb el que cal que retorni el resultat a altaAlumne de Tcontrol, i aquest, a la seva vegada, ha d'informar a VISTA (o sigui, al actor, a través de VISTA) de que l'operació s'ha realitzat amb èxit:



9.- El diagrama de col·laboració del cas d'us ALTA ALUMNE queda així: