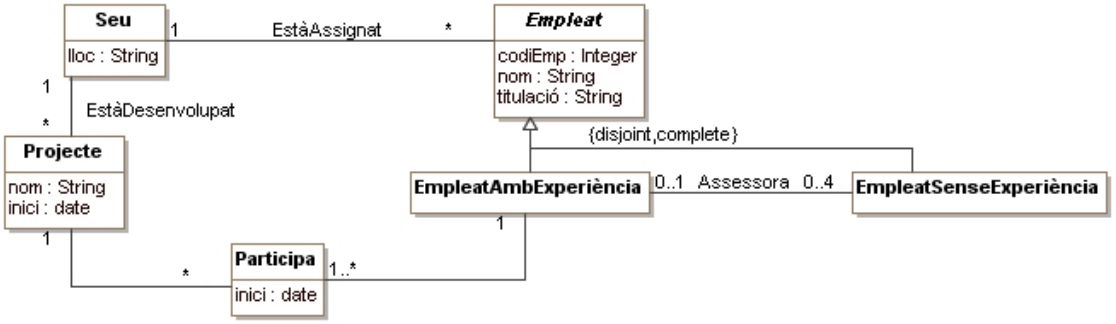
ARQUITECTURA DEL SOFTWARE

Unit 3.2: Domain Layer Design

Exercicis Resolts

Exercici 1 (Patrons)

Una empresa informàtica amb diferents seus a Catalunya ens ha demanat que li dissenyem una part d'un sistema software per a gestionar la participació dels seus empleats en els projectes que desenvolupa. De les seus de l'empresa, sabem la seva localització. Les seus de l'empresa desenvolupen projectes que són identificats pel seu nom. A més, també coneixem el nombre d'empleats amb experiència que participen en aquests projectes i la data d'inici del projecte. Dels empleats d'aquesta empresa sabem el codi d'empleat, el seu nom, la seva titulació i la seu a la que estan assignats. Els empleats poden ser de dos tipus: amb experiència i sense experiència. Els empleats sense experiència poden tenir l'assessorament d'un empleat amb experiència (que pot ser de la seva mateixa seu o d'una altra diferent). Els empleats amb experiència són els que participen en projectes, amb una data d'inici de participació en el projecte (la data-final de la participació no és rellevant per aquest disseny). Per simplificar podeu suposar que els empleats no canvien de tipus. Suposeu que ja s'ha fet una assignació de responsabilitats a capes i que com a resultat, es disposa d'un model de domini (perquè s'aplica Domain Model; a més, ja han aparegut algunes navegabilitats) i uns contractes d'operacions de capa de domini que es mostren tot seguit (considereu que les dues operacions són de casos d'ús diferents, amb el mateix nom de l'operació):



R.I. Textuals:
- Claus: (Seu, lloc); (Empleat, codi_emp); (Projecte, nom)
- No pot haver-hi dues participacions del mateix AmbExp en el mateix projecte.
- La data d'inici de la participació d'un empleat amb experiència en un projecte ha de ser posterior a la data d'inici del projecte.
- Els projectes en els que participa un empleat amb experiència han d'estar desenvolupats a la mateixa seu on l'empleat està assignat.

context CapaDeDomini::llistaAss (nomPr: String): Set(String)
exc *projecteNoExisteix*: El projecte identificat per *nomPr* no existeix
post result = retorna els noms dels empleats sense experiència que estan assignats a la seu on s'està desenvolupant el projecte amb *nomPr* i que són assessorats per empleats amb experiència que estan participant en el projecte *nomPr*

context CapaDeDomini::baixaEmp (codiEmp: Integer)
exc *empleatNoExisteix*: L'empleat identificat per *codiEmp* no existeix
post *eliminaAssignació:* elimina la instància de l'associació *EstàAssignat* entre l'empleat i la seu
post *eliminaParticipacióAssessorament*: si l'empleat és amb experiència, s'eliminen les seves associacions de participació, i les seves associacions d'assessorament amb empleats sense experiència. Si l'empleat és sense experiència s'elimina l'associació d'assessorament amb l'empleat amb experiència, si és el cas
post *eliminaEmpleat*: s'elimina la instància d'empleat

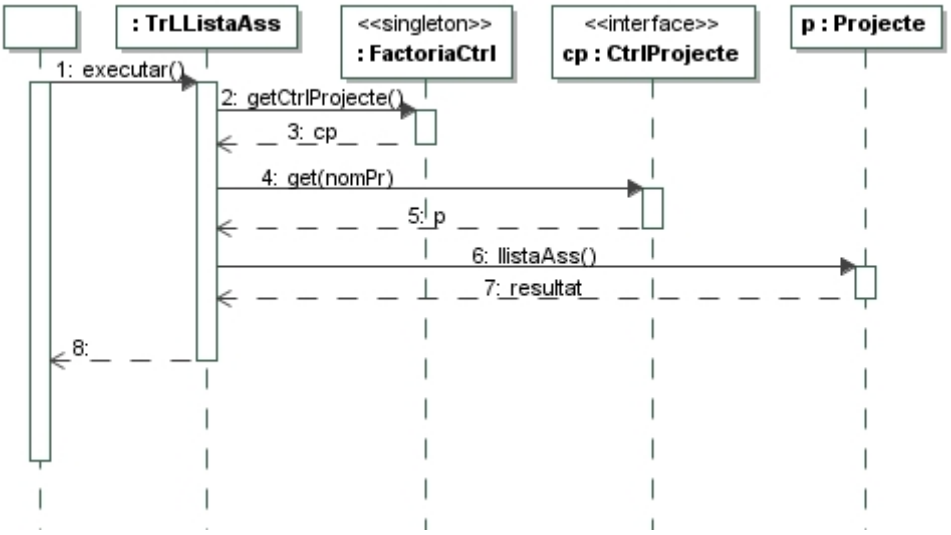
Es demana:

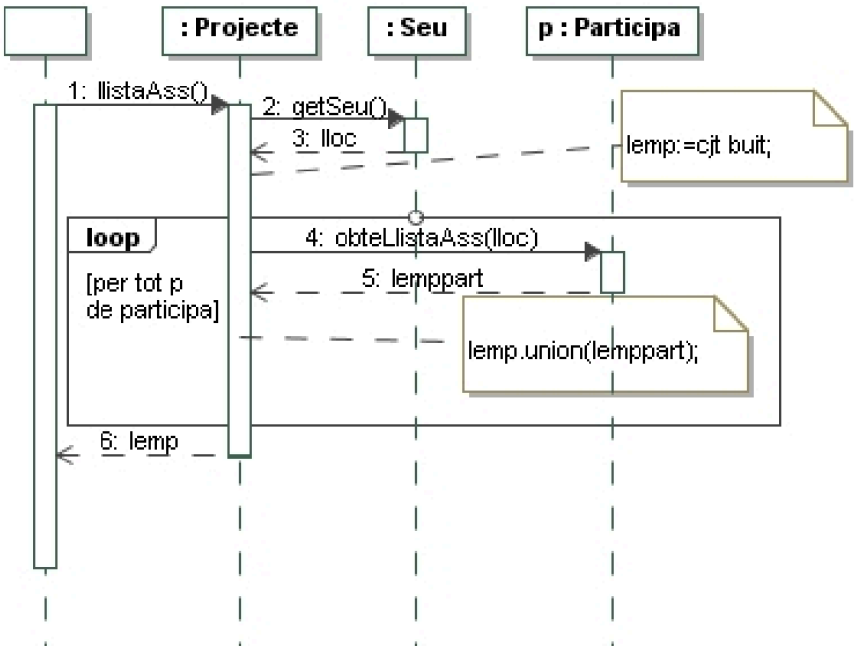
(a) Feu els diagrames de seqüència de les operacions anteriors. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència. Suposeu una navegabilitat inicial doble de l'associació *EstàAssignat*.

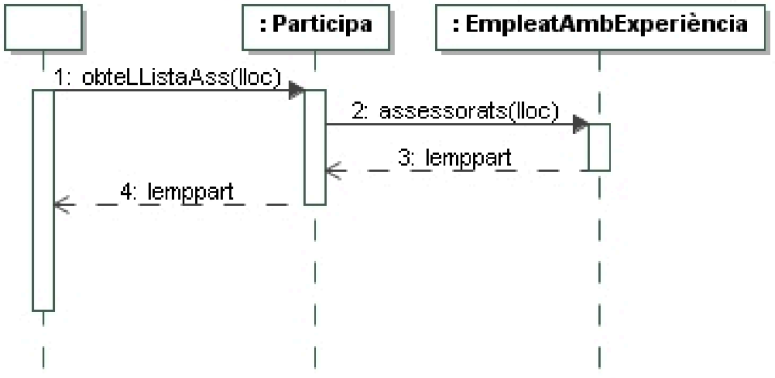
(b) Feu el diagrama de classes de la capa de domini.

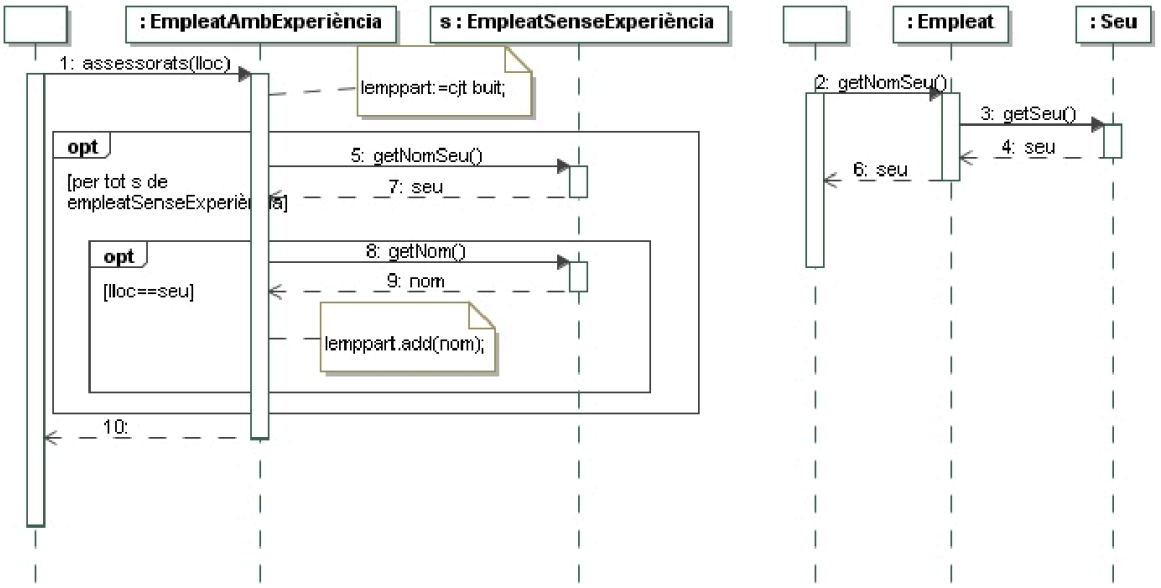
Solució

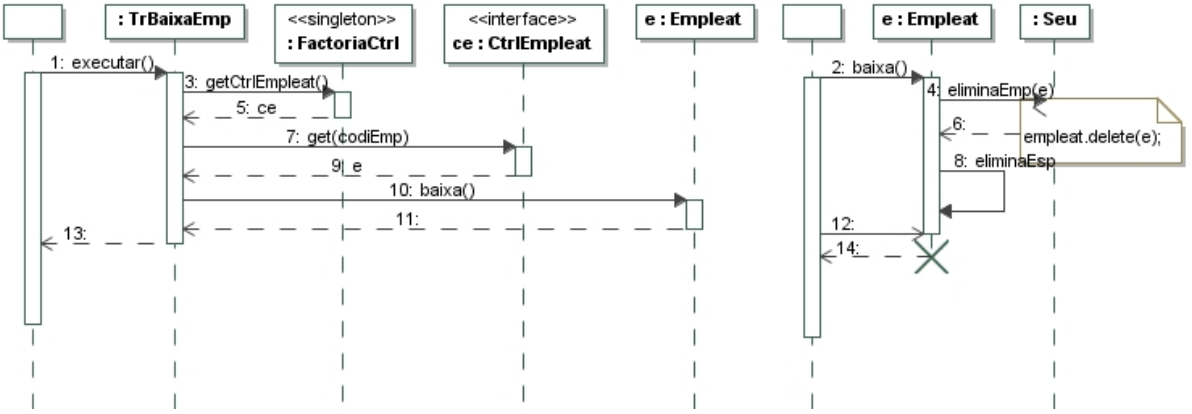
(a) Diagrama de seqüència corresponent a l'operació *llistaAss* i *baixaEmp*.

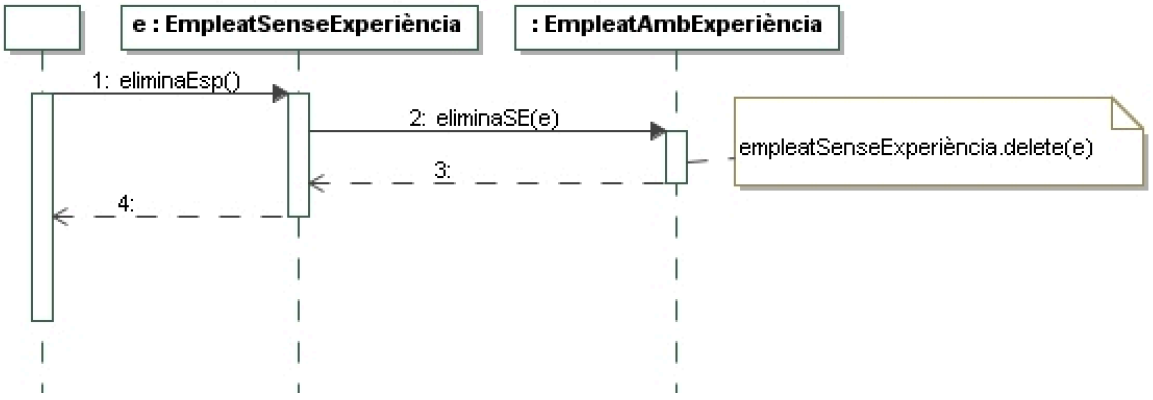


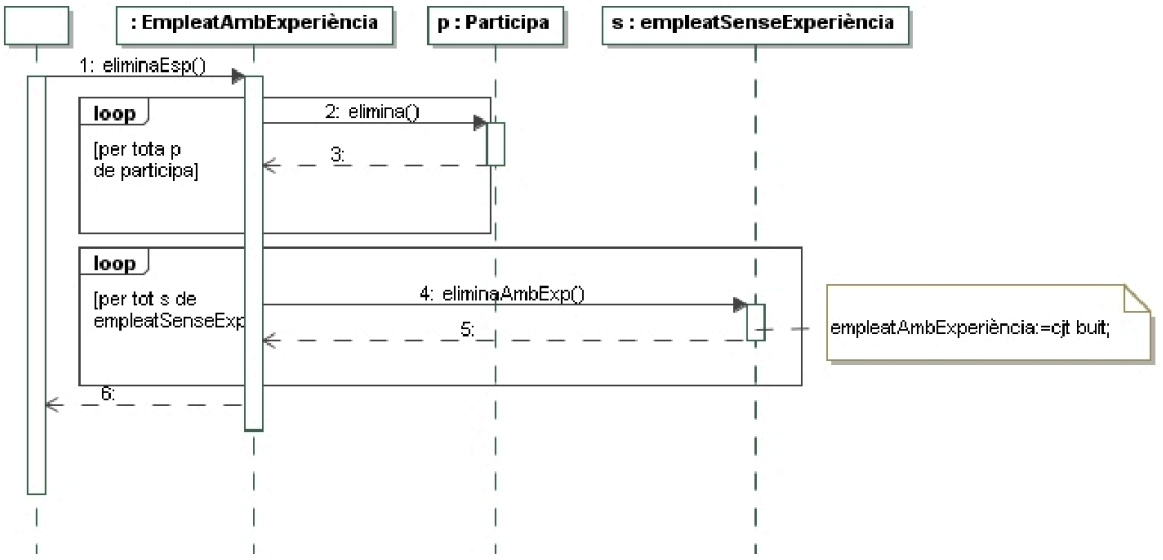


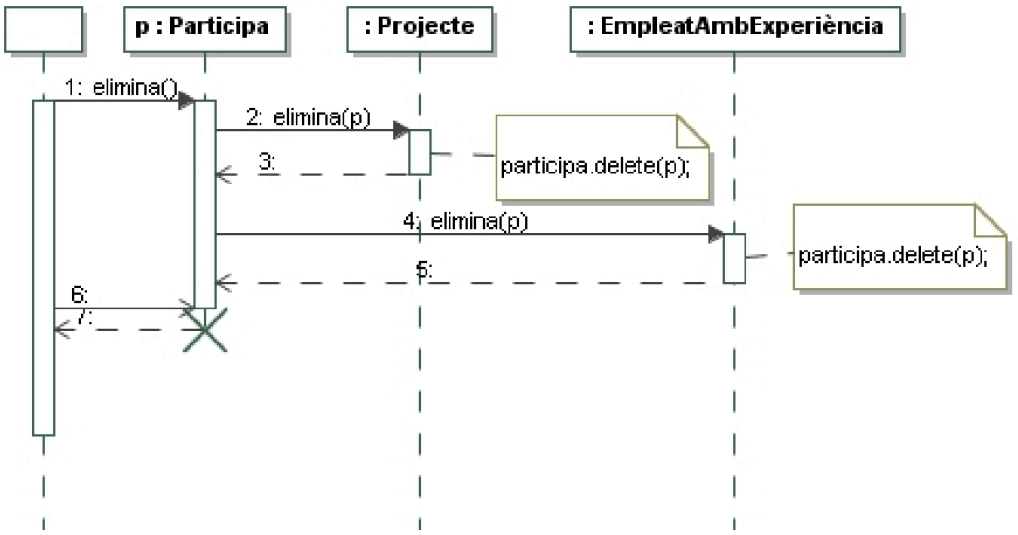



: EmpleatAmbExperiència
s : EmpleatSenseExperiència
1: assessorats(lloc)
lemppart:=cjt buit;
opt
[per tot s de empleatSenseExperièn ta]
5: getNomSeu()
7: seu
opt
[lloc==seu]
8: getNom()
9: nom
lemppart.add(nom);
10:
: Empleat
: Seu
2: getNomSeu()
3: getSeu()
4: seu
6: seu

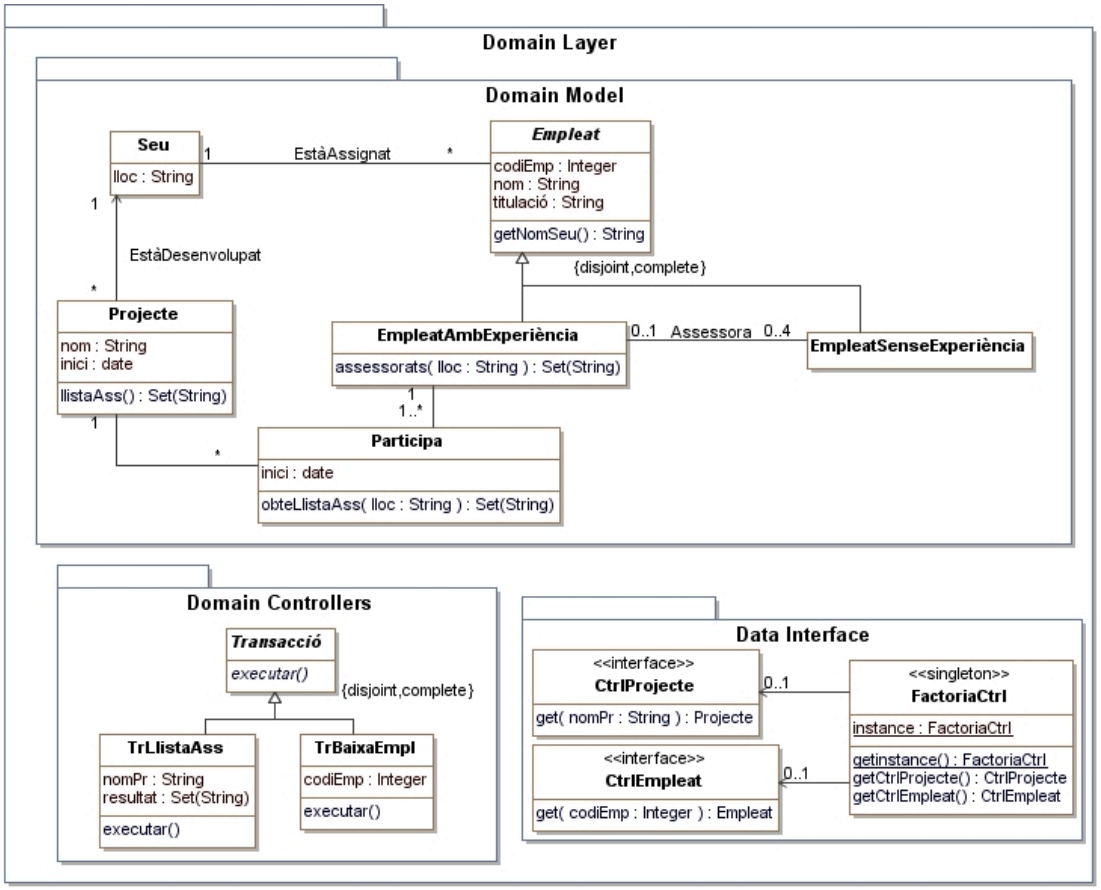

: TrBaixaEmp
<<singleton>>
: FactoriaCtrl
<<interface>>
ce : CtrlEmpleat
e : Empleat
1: executar()
3: getCtrlEmpleat()
5: ce
7: get(codiEmp)
9: e
10: baixa()
11:
13:
e : Empleat
: Seu
2: baixa()
4: eliminaEmp(e)
6:
empleat.delete(e);
8: eliminaEsp
12:
14:


e : EmpleatSenseExperiència
: EmpleatAmbExperiència
1: eliminaEsp()
2: eliminaSE(e)
empleatSenseExperiència.delete(e)
3:
4:





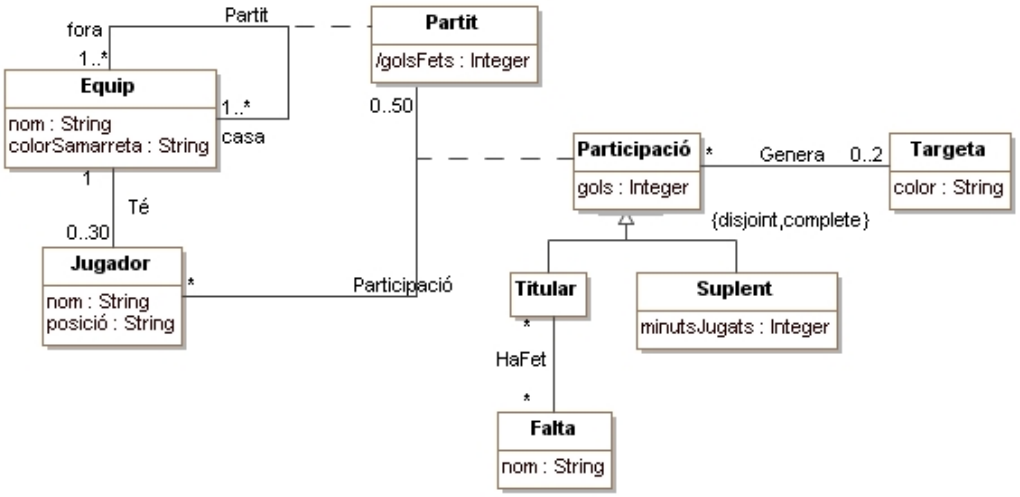
(b) Diagrama de clases de la capa de domini.



Exercici 3 (Patrons)

La Federació Internacional de Futbol ens ha demanat que dissenyem dos casos d'ús per gestionar la informació de les seves competicions. Aquest federació disposa de la informació dels partits que es juguen i dels jugadors de que disposa cada equip. De cada equip es coneix el nom, i el color de la samarreta titular amb la que juguen. Dels jugadors es disposa del seu nom i de la seva posició en el camp, i dels partits es coneix els gols totals fets pels dos equips. Per cada jugador que participa en un partit també es coneix els gols que ha fet en aquell partit, les targetes que li han mostrat i si ha jugat com a titular o suplent. Per les participacions com a suplent es coneix el nombre de minuts jugats i per les participacions com a titular les faltes fetes (es pot considerar que els jugadors suplents no fan faltes). Els casos d'ús que volem dissenyar permeten traspassar a un jugador a un altre equip i obtenir els jugadors que fan una determinada falta quan juguen a casa. A continuació disposeu de l'especificació feta per aquest sistema:

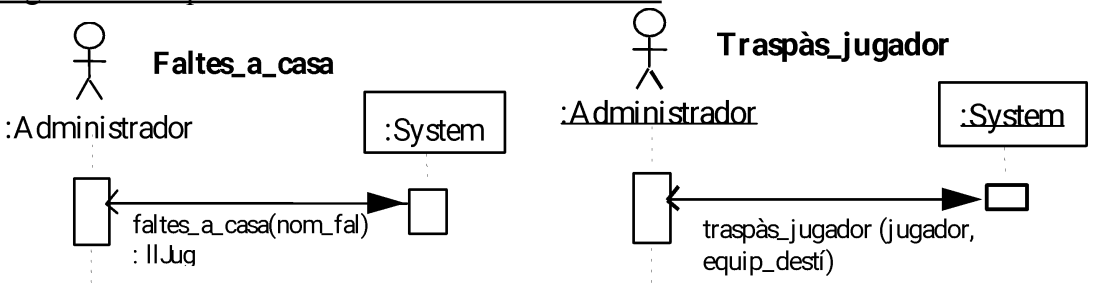
Esquema conceptual d'especificació:



R.I Textuals:
- Claus classes no associatives: (Equip, nom); (Jugador, nom); (Targeta, color); (Falta, nom_fal).
- Un equip no pot jugar amb ell mateix.
- Un jugador només pot participar en partits en els que juga el seu equip.

Info. derivada:
- golsfets: és el nombre de gols fets per tots els jugadors que han participat en un partit.

Diagrama de seqüència d'esdeveniments del sistema:



context faltes_a_casa(nom_fal: String): Set(String)
pre *existeixFalta:* existeix la falta *nom_fal*
post: result = noms de jugadors que han fet la falta *nom_fal* en algun dels partits jugats a casa

context traspàs_jugador(jug: String, equip_destí: String)
pre *existeixJugador:* existeix el jugador *jug*
pre *existeixEquipDestí:* existeix l'equip *equip_destí*
pre *jugadorNoJugaDestí:* el jugador *jug* no juga a l'equip *equip_destí*
post: 2.1 s'eliminen totes les associacions participa del jugador en el seu equip origen, tant si era suplent com titular
2.2 s'eliminen les associacions entre participació i targetes
2.3 s'eliminen les associacions entre la participació com a titular i les faltes fetes
2.4 s'elimina l'associació Té entre l'equip origen i el jugador
2.5 es crea un nova associació Té entre l'equip destí i el jugador

Es demana:

(a) Feu l'especificació de la capa de domini considerant un disseny extern sense cap tipus de desplegable (és a dir, per a cada operació, la informació s'entra directament en camps de text o numèrics, tota de cop), patró Domain Model a la

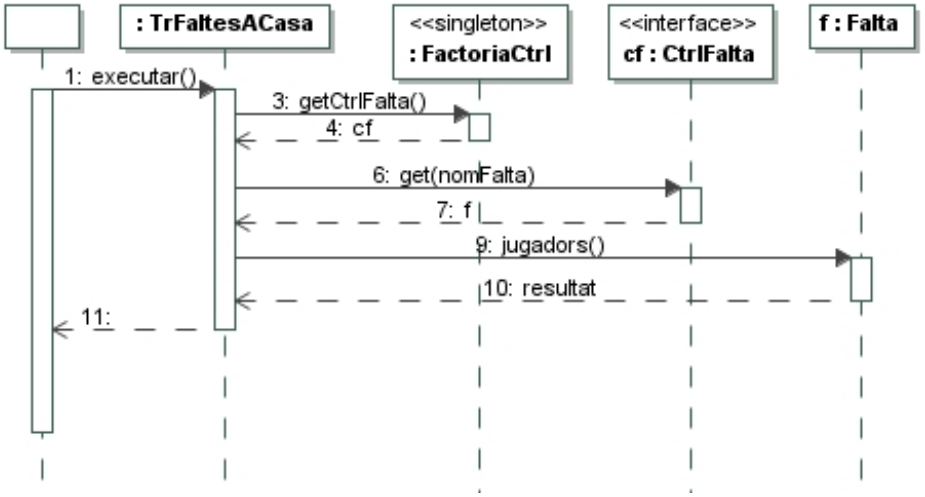
capa de domini, i assignació de totes les responsabilitats que apareixen en els contractes anteriors a la capa de domini, tret de les responsabilitats de clau que s'assignen a la capa de gestió de dades. Considereu que l'atribut derivat *golsfets* d'un partit és calculat.

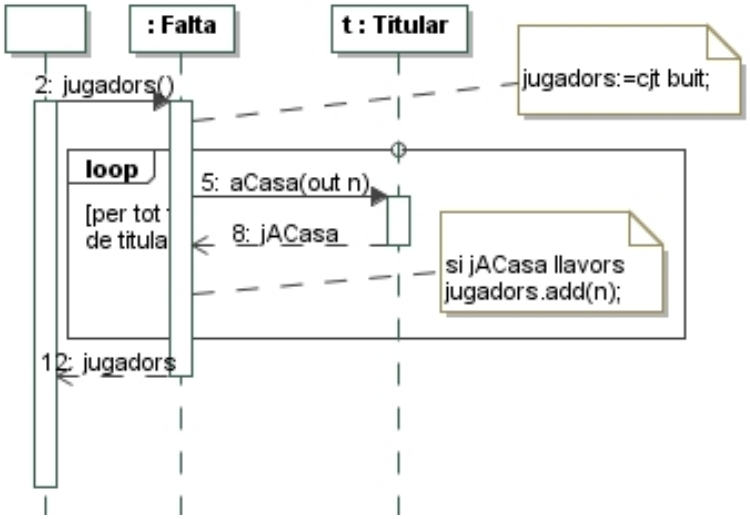
(b) Feu els diagrames de seqüència de les operacions resultants de la capa de domini. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència. Suposeu que la navegabilitat de les associacions que apareixen a la capa de domini és doble.

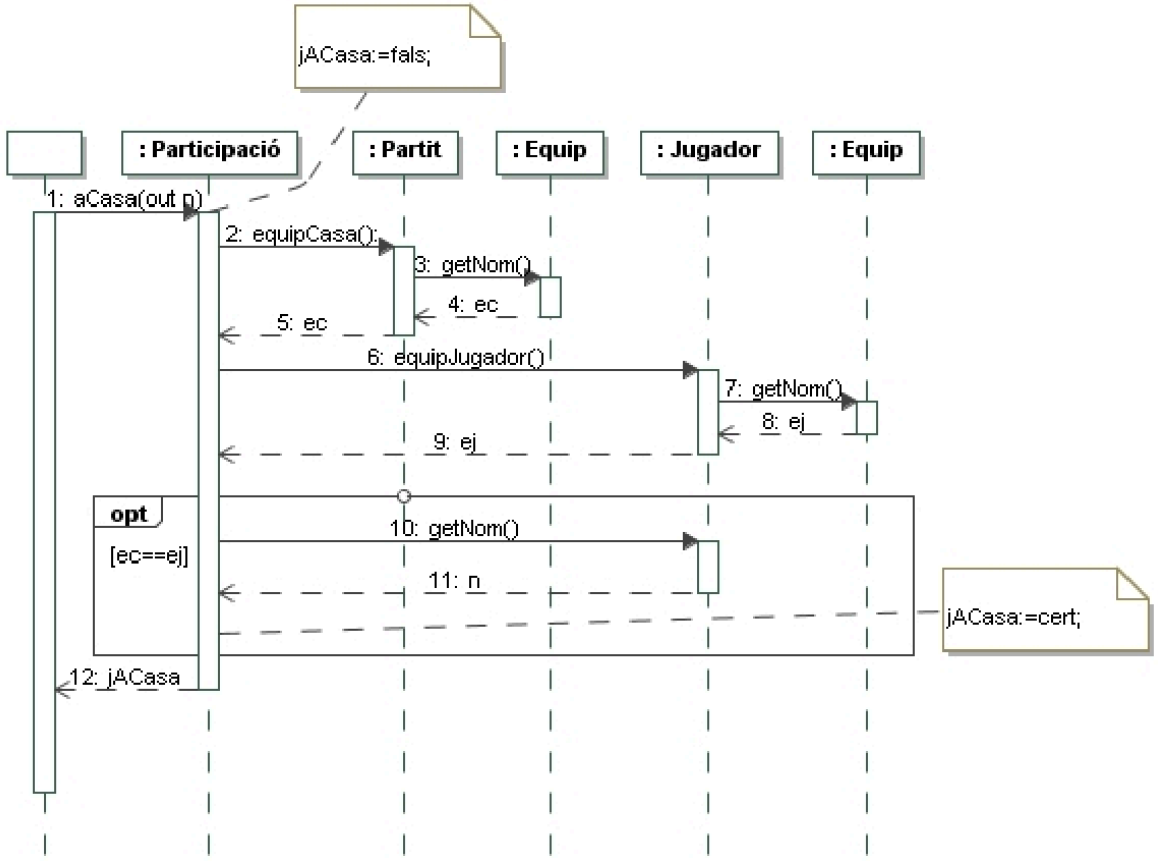
(c) Feu el diagrama de classes de la capa de domini.

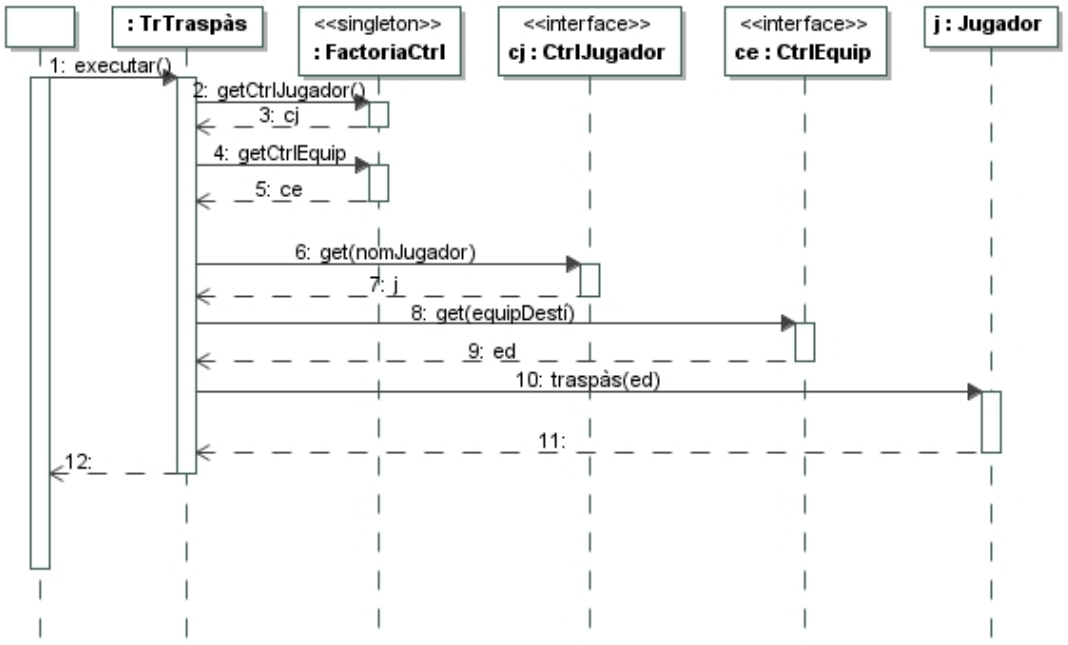
Solució

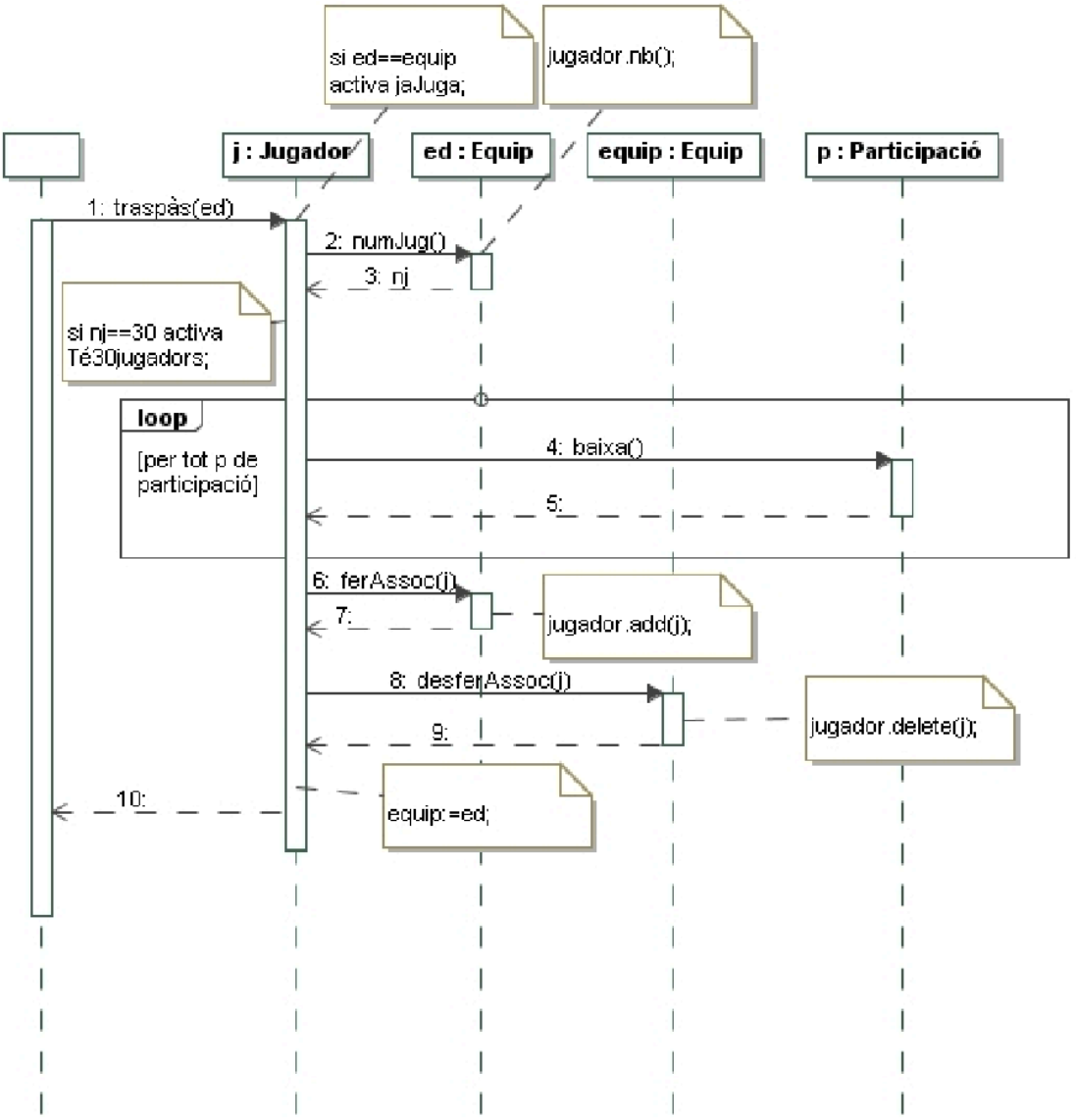
(a) Diagrama de seqüència corresponent a l'operació *faltes_a_casa* i *traspàs_jugador*.

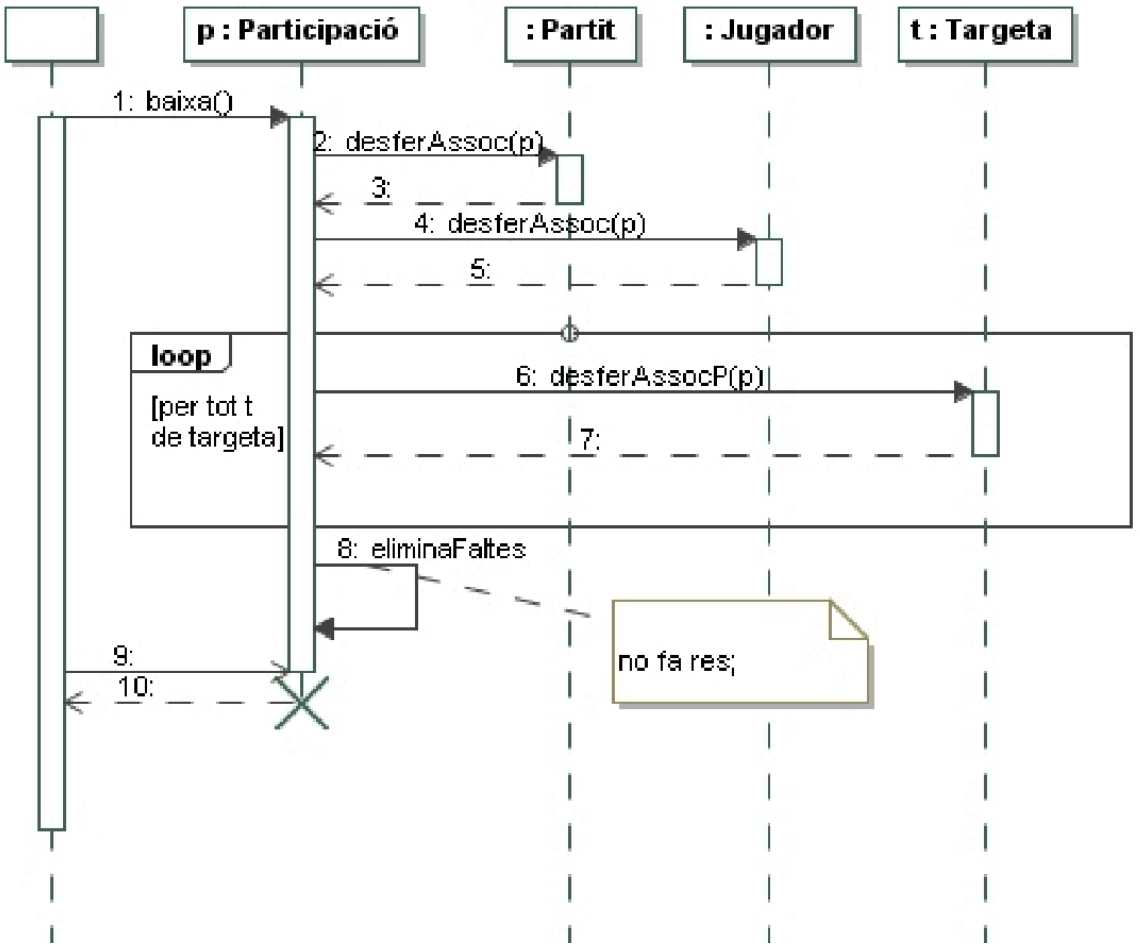


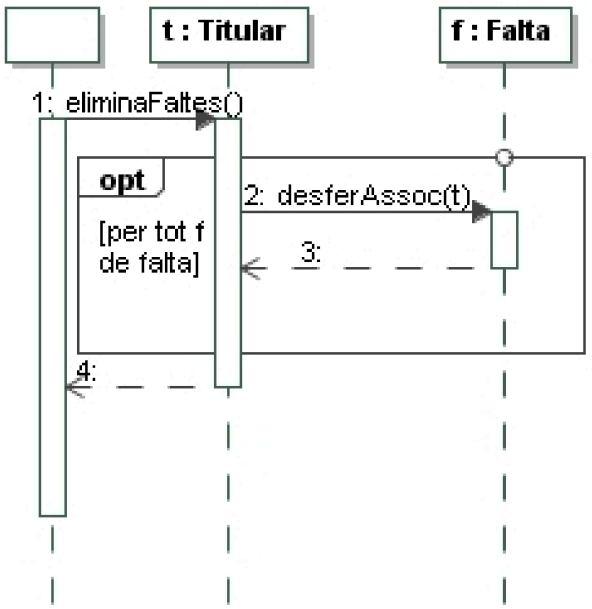



jACasa:=fals;
: Participació
: Partit
: Equip
: Jugador
: Equip
1: aCasa(out n)
2: equipCasa();
3: getNom()
4: ec
5: ec
6: equipJugador()
7: getNom()
8: ej
9: ej
opt
[ec==ej]
10: getNom()
11: n
jACasa:=cert;
12: jACasa

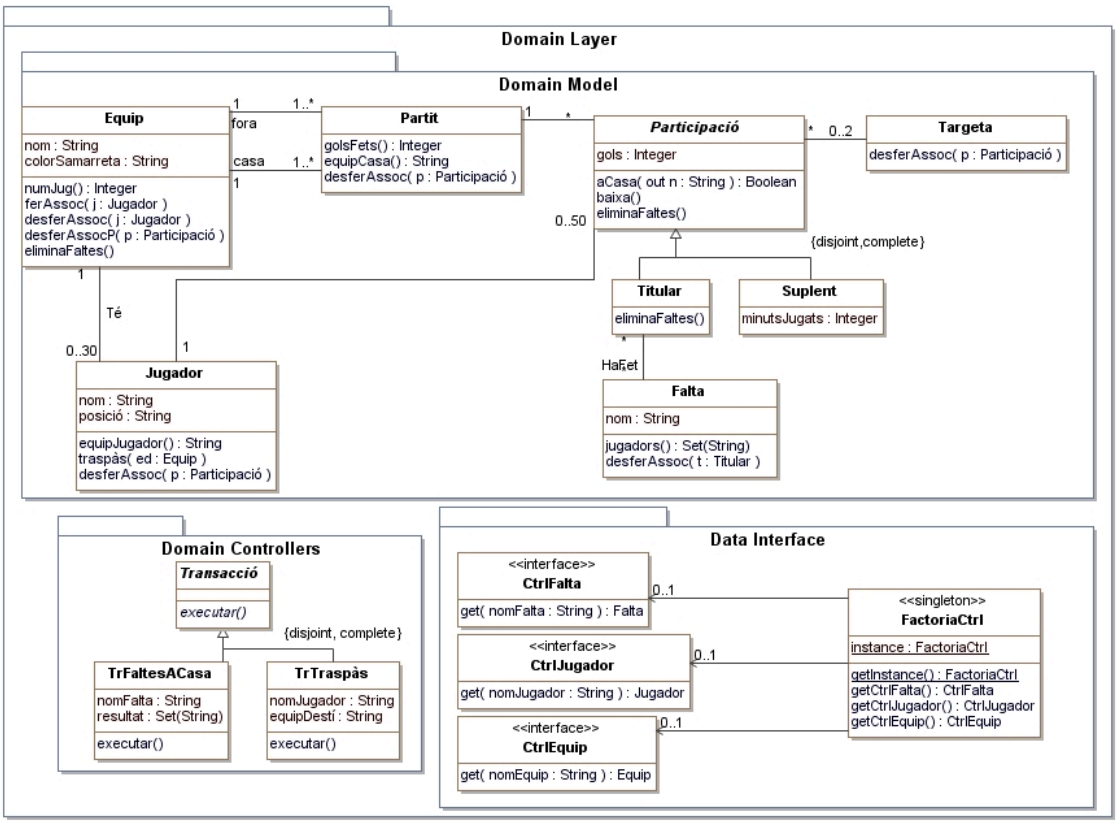

: TrTraspàs
<<singleton>>
: FactoriaCtrl
<<interface>>
cj : CtrlJugador
<<interface>>
ce : CtrlEquip
j : Jugador
1: executar()
2: getCtrlJugador()
3: cj
4: getCtrlEquip
5: ce
6: get(nomJugador)
7: j
8: get(equipDestí)
9: ed
10: traspàs(ed)
11:
12:


si ed==equip
activa jaJuga;
jugador.nb();
j : Jugador
ed : Equip
equip : Equip
p : Participació
1: traspàs(ed)
2: numJug()
3: nj
si nj==30 activa
Té30jugadors;
loop
[per tot p de
participació]
4: baixa()
5:
6: ferAssoc(j)
7:
jugador.add(j);
8: desferAssoc(j)
9:
jugador.delete(j);
10:
equip:=ed;



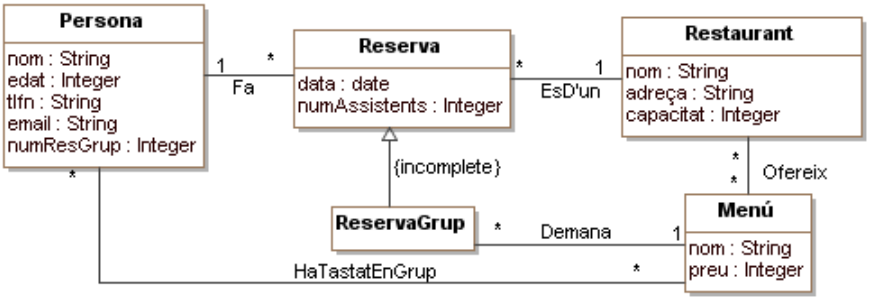


(b) Diagrama de clases de la capa de domini.



Exercici 6 (Serveis)

Una cadena de restaurants que ofereix menús per sopars ens ha demanat que li dissenyem una part d'un sistema software per gestionar les seves reserves. L'esquema conceptual (de l'especificació) es mostra a continuació:



R.I. Textuals:
- Claus: (Persona, nom); (Restaurant, nom); (Menú, nom); (Reserva, Persona::nom+data);
- Els menús de les reserves de grup han de ser menús oferts pel restaurant on s'ha fet la reserva.
- La capacitat d'un restaurant ha de ser més gran que el sumatori dels assistents de les reserves previstes per aquell restaurant en una data determinada.
- El numAssistents d'una reserva de grup ha de ser més gran que 5.
- Els menús que ha tastat en grup una persona són els mateixos que ha reservat.
- El numResGrup d'una persona és igual al número de reserves de grup que ha fet la persona.
- Tots els atributs de tipus integer han de ser positius.
- Altres restriccions no rellevants per l'exercici.

La capa de domini ofereix l' operació següent:

context novaReservaGrup(nomClient:String, nomRest:String, dr:date, nAss:Integer, nomMenú:String): Integer
pre *client-existeix*, *data-ok*, *més-5assistents*, *reserva-no-existeix*
exc *restaurant-no-existeix*: el restaurant amb *nomRest* no existeix.
exc *menu-no-ofert*: el restaurant *nomRest* no ofereix el menú *nomMenú*.
exc *restaurant-sense-lloc*: el restaurant *nomRest* no té disponibilitat pel *nAss* a la data *dr*.
post *reserva-grup-creada:* es crea una reserva de grup i es formen totes les associacions amb la persona, el restaurant i el menú.
post *numResGrup-incrementat:* s'incrementa el *numResGrup* de la persona *nomClient*.
post *haTastat-creat:* si la persona no ha tastat el menú, es crea una instància de *HaTastatEnGrup* entre la persona *nomClient* i el menú *nomMenú*.
post *result*= número de places disponibles del restaurant *nomRest* a la data *dr* (capacitat del restaurant – número de assistents (de les reserves) per la data *dr* - *nAss*).

La cadena de restaurants té un sistema CRM per gestionar la informació dels seus clients. Per integrar el sistema CRM a la nostra arquitectura s'ha definit el servei *SvCRM*. Cada restaurant disposa d'un servei que conté la informació del restaurant i dels menús que ofereix (dels menús només es guarda el nom). Podeu suposar que el servei d'un restaurant que té nom *nom* es diu *Svnom*. A més, es decideix llogar un servei genèric de gestió de reserves (aquest servei no distingeix entre les reserves de grup i les que no ho són). A continuació disposeu de les operacions proporcionades per cada servei (no es poden afegir operacions als serveis ni modificar les existents):

context SvCRM::obteDadesClient (inout dCl: DTOClient)
exc client-no-existeix: el client *dCl.nom* no existeix
post assigna l'edat, tlfn i email del client amb nom *dCl.nom* a *dCl.edat, dCl.tlfn* i *dCl.email*
**La classe DTOClient té com atributs nom, edat, tlfn i email.*

context Svnom::obteDadesRestaurant (): DTORestaurant
post retorna el nom, l'adreça, capacitat i menús que ofereix el restaurant a *dR.nom*, *dR.adreça*, *dR.capacitat* i *dR.menús*
**La classe DTORestaurant té com atributs nom, adreça, capacitat i un conjunt de noms de menús.*

context SvGestorReserves::creaReserva (dRes:DTOReserva)
exc reserva-existeix: la reserva de la persona *dRes.nom* i data *dRes.data* existeix
post es dona d'alta la reserva

context SvGestorReserves::obteDadesReserva(inout dRes:DTOReserva)
exc reserva-no-existeix: la reserva de la persona *dRes.nom* i data *dRes.data* no existeix
post retorna el nom del restaurant i el número de assistents de la reserva identificada per *dRes.nom* i *dRes.data*

context SvGestorReserves::obtenirOcupacióxData(inout dOcup: DTOOcupacióReserva)
post *dOcup.numAssistents*= sumatori dels assistents de les reserves del restaurant *dOcup.nomRes* en data *dOcup.dt*.
**La classe DTOReserva té com atributs nom persona, nom restaurant, data i número d'assistents.*
**La classe DTOOcupacióReserva té com atributs nom restaurant, data i número d'assistents.*

Es demana:

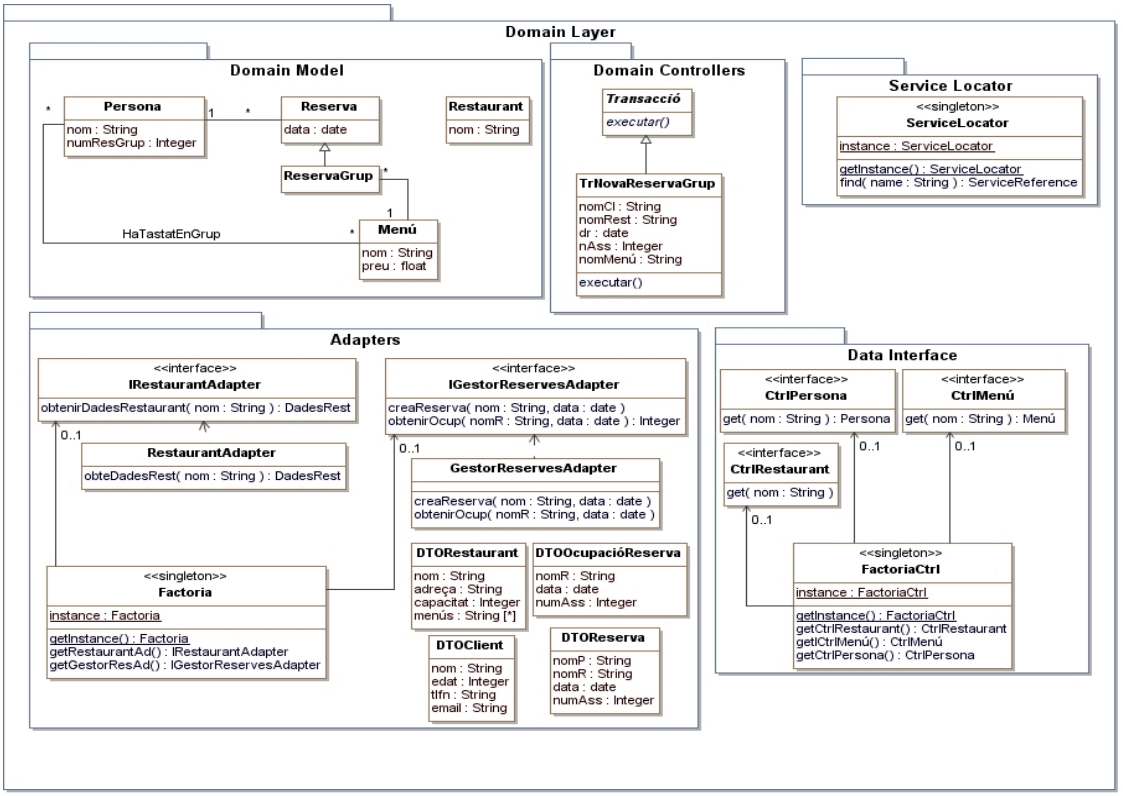
(a) Diagrama de classes de la capa de domini del sistema de reserves, incloent-ne els atributs (però no les operacions). Suposeu Domain Model, Data Mapper, i controlador Transacció. Es vol minimitzar (per sobre d'altres criteris) la informació redundant entre els serveis i el sistema que volem dissenyar. Justifica el diagrama de classes obtingut. Suposeu la navegabilitats doble de l'associació entre *Persona* i *Reserva*, la navegabilitat simple de *HaTastatEnGrup* de *Persona* a *Menú,* la navegabilitat simple de *EsD'un* de *Reserva* a *Restaurant* i la navegabilitat simple de *Demana* de *Reservagrup* a *Menú*.
(b) Diagrama de seqüència de l'operació *novaReservaGrup*.

Solució

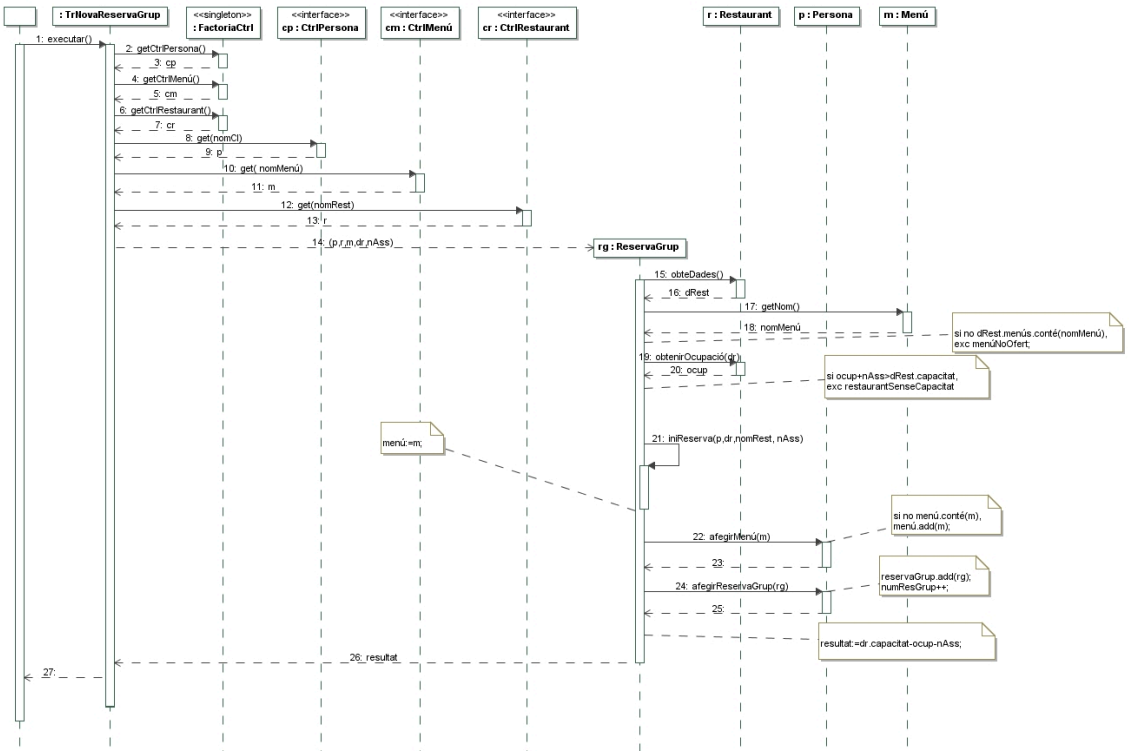
(a) Diagrama de classes

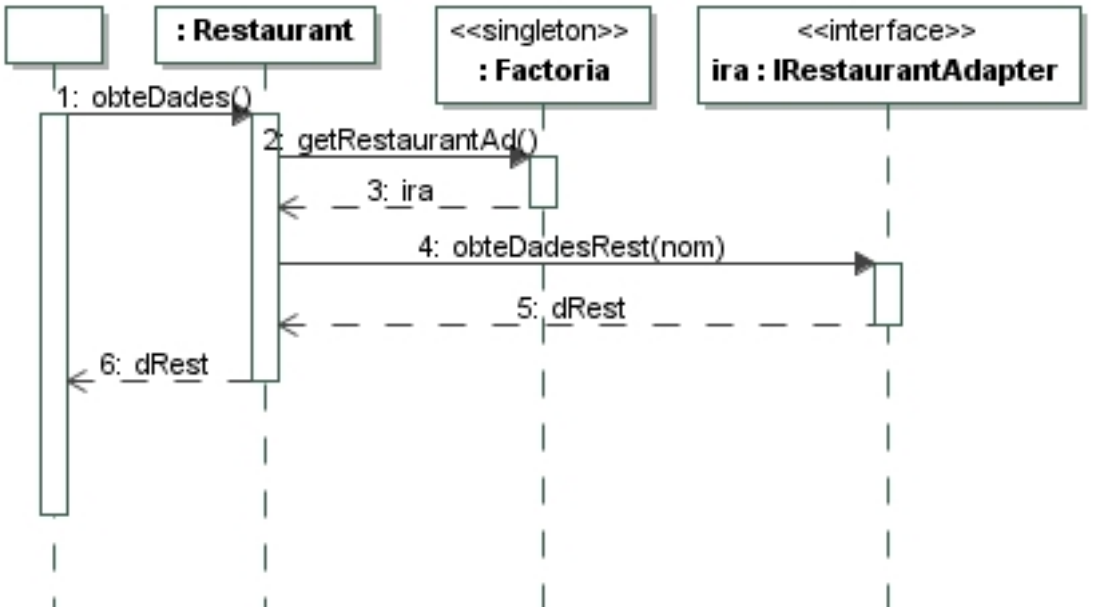
Classes del paquet Domain Model

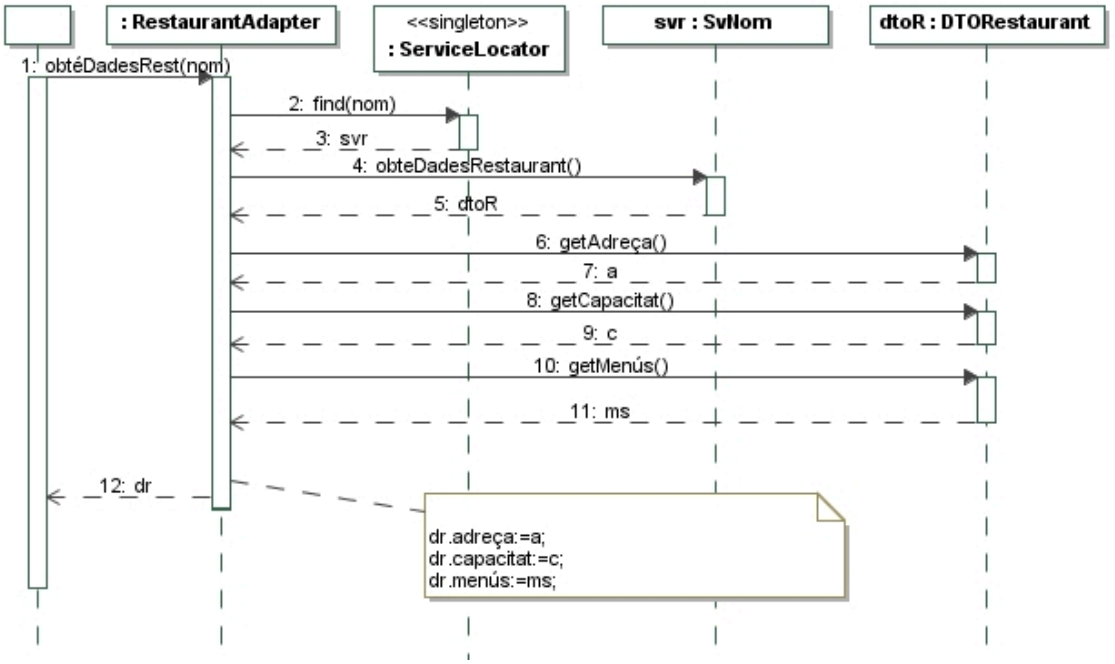
- La resta d'atributs de Persona queden sota la responsabilitat del servei CRM
- La responsabilitat de la informació dels restaurants i dels menús que ofereixen queda sota la responsabilitat del servei de cada restaurant.
- La informació de les reserves queda sota la responsabilitat del servei SvGestorReserves. Com el servei no ens proporciona la informació de totes les reserves decidim mantenir al nostre sistema la jerarquia (amb les claus) però sense la informació que guarda el servei.
- La classe Restaurant és necessària per comprovar l'existència del restaurant i per tenir registrats tots els restaurants de la cadena.

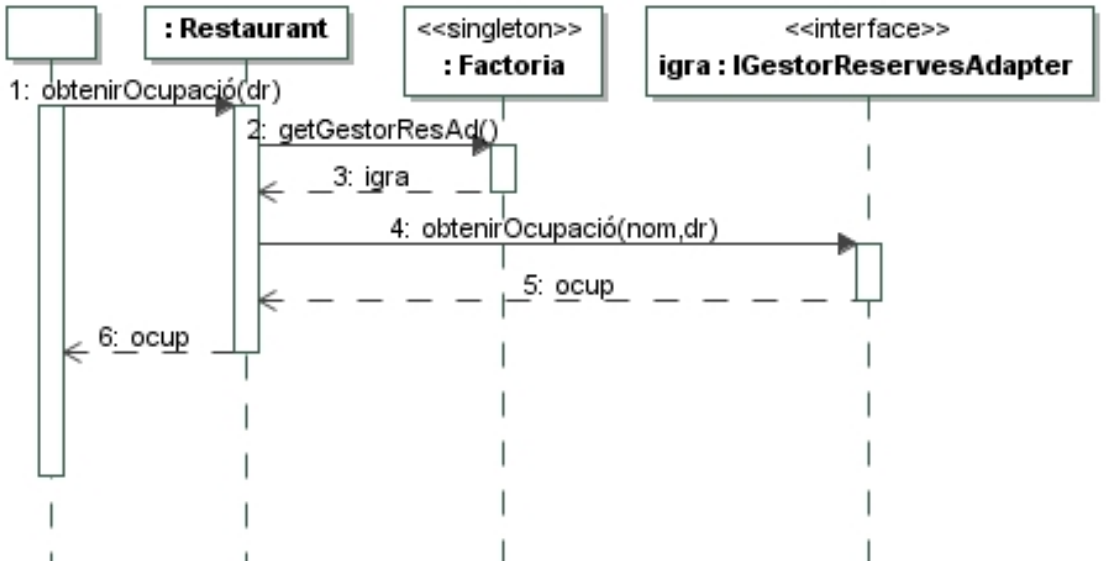


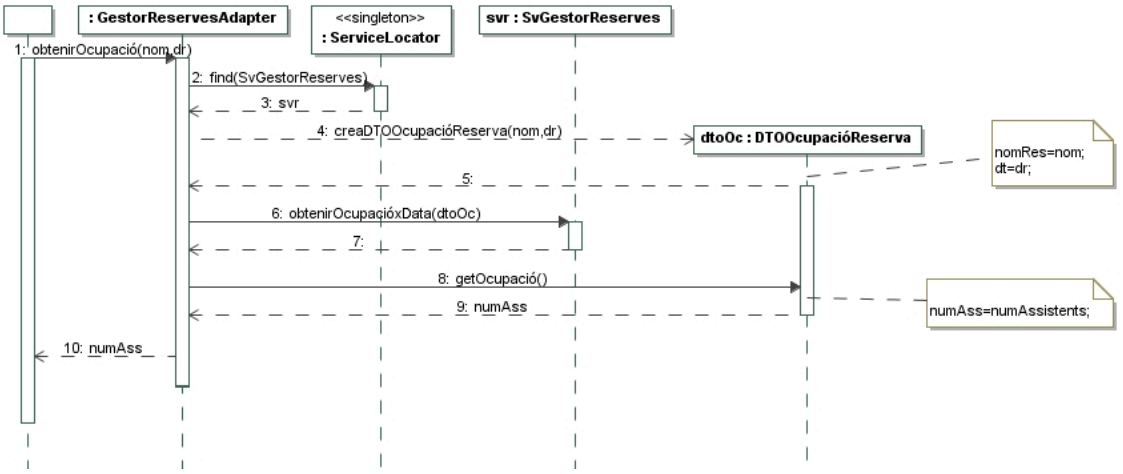
(b) Diagrama de seqüència

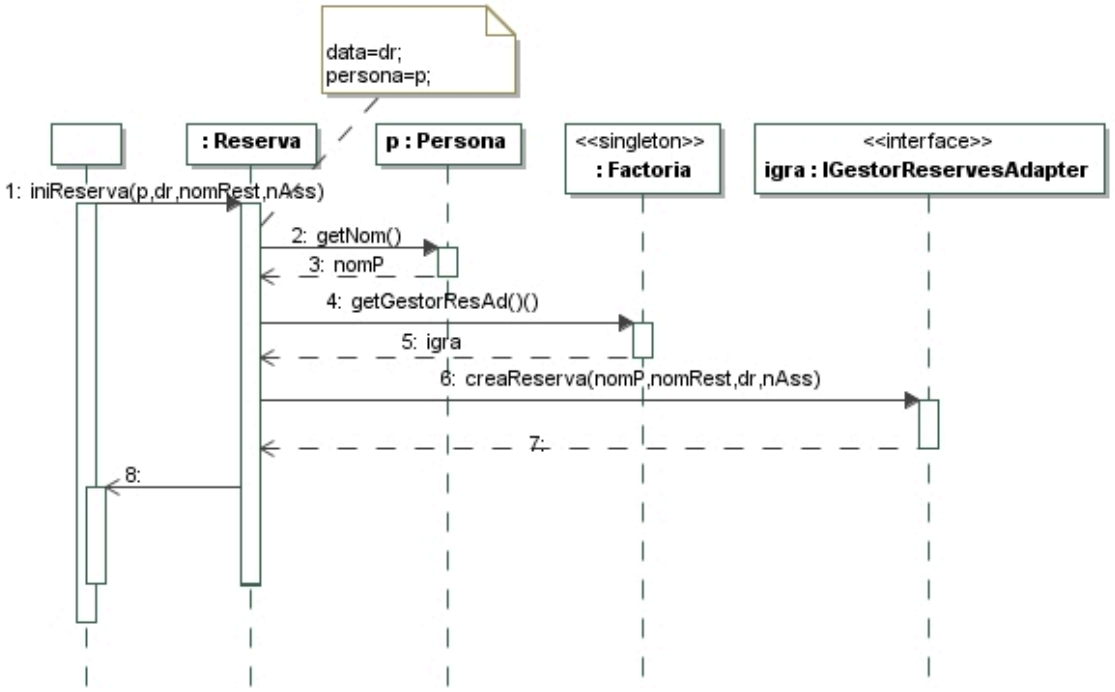


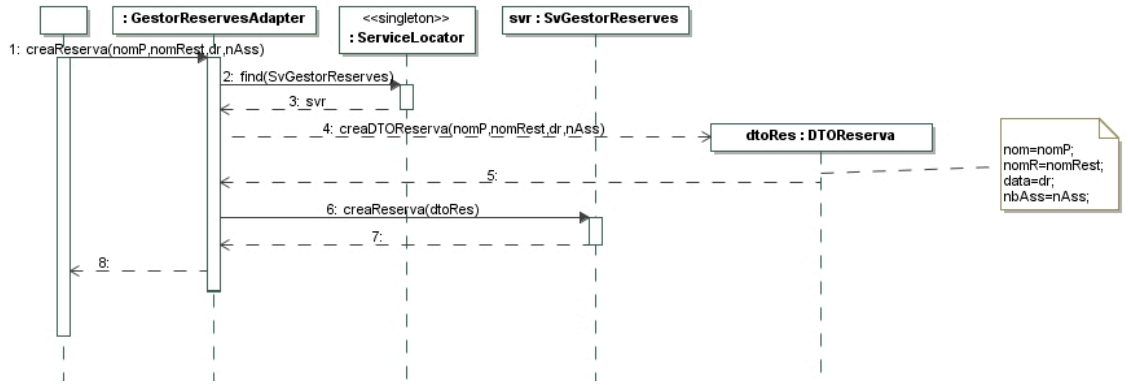

: Restaurant
<<singleton>>
: Factoria
<<interface>>
ira : IRestaurantAdapter
1: obteDades()
2: getRestaurantAd()
3: ira
4: obteDadesRest(nom)
5: dRest
6: dRest


: RestaurantAdapter
<<singleton>>
: ServiceLocator
svr : SvNom
dtoR : DTORestaurant
1: obtéDadesRest(nom)
2: find(nom)
3: svr
4: obteDadesRestaurant()
5: dtoR
6: getAdreça()
7: a
8: getCapacitat()
9: c
10: getMenús()
11: ms
12: dr
dr.adreça:=a;
dr.capacitat:=c;
dr.menús:=ms;


: Restaurant
<<singleton>>
: Factoria
<<interface>>
igra : IGestorReservesAdapter
1: obtenirOcupació(dr)
2: getGestorResAd()
3: igra
4: obtenirOcupació(nom,dr)
5: ocup
6: ocup

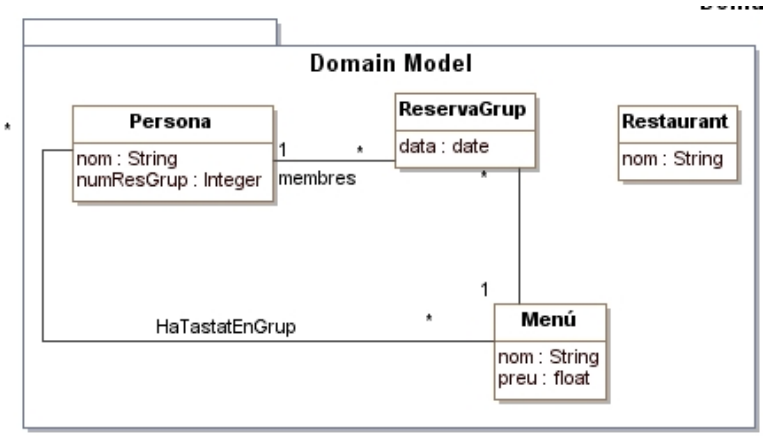

: GestorReservesAdapter
<<singleton>>
: ServiceLocator
svr : SvGestorReserves
1: obtenirOcupació(nom,dr)
2: find(SvGestorReserves)
3: svr
4: creaDTOOcupacióReserva(nom,dr)
dtoOc : DTOOcupacióReserva
nomRes=nom;
dt=dr;
5:
6: obtenirOcupacióxData(dtoOc)
7:
8: getOcupació()
9: numAss
numAss=numAssistents;
10: numAss


data=dr;
persona=p;
: Reserva
p : Persona
<<singleton>>
: Factoria
<<interface>>
igra : IGestorReservesAdapter
1: iniReserva(p,dr,nomRest,nAss)
2: getNom()
3: nomP
4: getGestorResAd()()
5: igra
6: creaReserva(nomP,nomRest,dr,nAss)
7:
8:



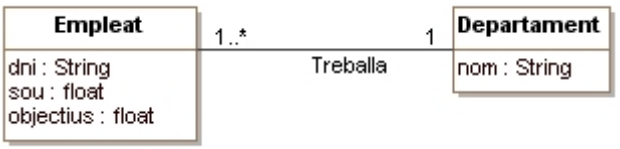
(c) Alternativa

Si el servei SVGestorReserves proporciones una operació per obtenir les dades de totes les reserves, el diagrama de classes del nostre sistema canviaria i per tant també el diagrama de seqüència.



Exercici 9 (Patrons)

Una empresa del sector farmacèutic ens ha demanat que li dissenyem un sistema per calcular el sou dels seus empleats. Aquesta empresa està formada per diferents departaments i cada departament té assignats els seus empleats. L'empresa vol poder canviar de forma dinàmica el càlcul dels sous dels seus empleats. En concret vol tenir la possibilitat de calcular el sou de cada empleat de diferents formes i poder variar aquest càlcul per cada empleat en funció de la productivitat i d'altres criteris que estableixi l'empresa. L'empresa estebleix tres formes de fer el càlcul del sou dels seus empleats: sou amb bonus (souEmpleat + bonus definit per l'empresa), sou amb objectius (souEmpleat*2 si l'empleat ha assolit uns objectius superiors a un valor establert per l'empresa i souEmpleat en cas contrari) o sou de crisis (souEmpleat - percentatge definit per l'empresa*souEmpleat). A continuació disposeu de l'esquema conceptual del sistema:



Restriccions textuals:
RT1. Claus: (Empleat, dni); (Departament, nom)

La capa de domini ofereix les següents operacions:

context CapaDeDomini::calcularSou (): Set(dniEmp:String, nomDept:String, sou:float)
exc *noHiHaEmpleats*: no hi ha empleats definits al sistema
post result = retorna el dni, el nom del departament de l'empleat dni i el sou a pagar per tots els empleats de l'empresa. La forma de càlcul del sou de cada empleat estarà definit per l'empresa.

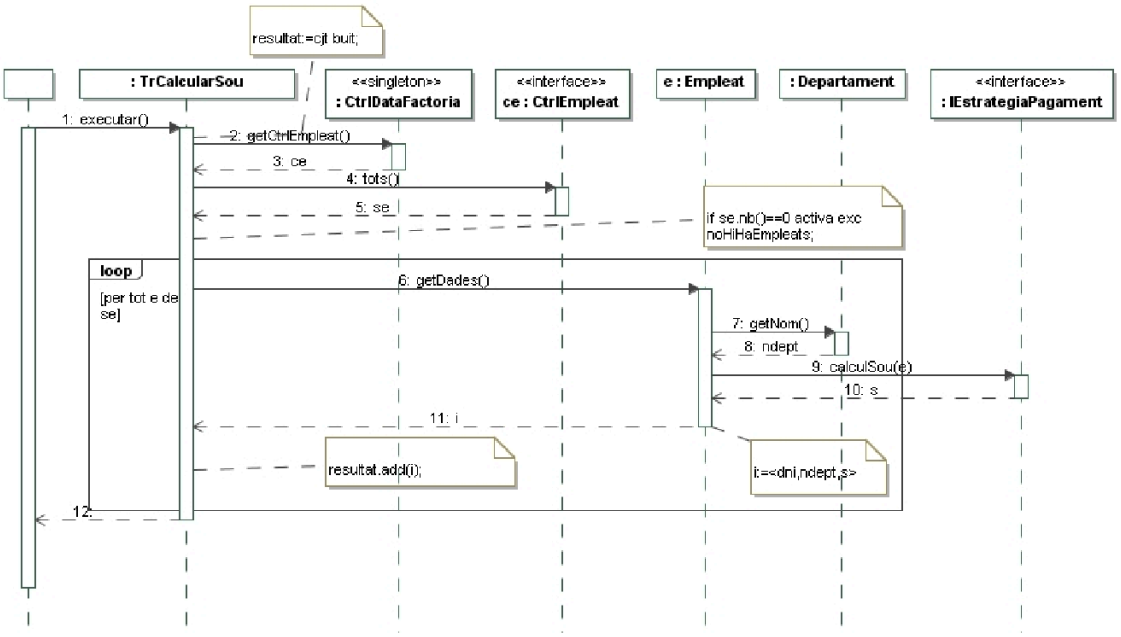
context CapaDeDomini::assignaCàlculSou (dniEmp:String,perc: Integer)
pre *percMésGranQue0*: el percentatge perc és més gran que 0
exc *noExisteixEmpleat*: no existeix l'empleat dniEmp
post *assigna*: s'assigna el càlcul del sou de l'empleat dni a pagament de crisi amb el percentatge perc.

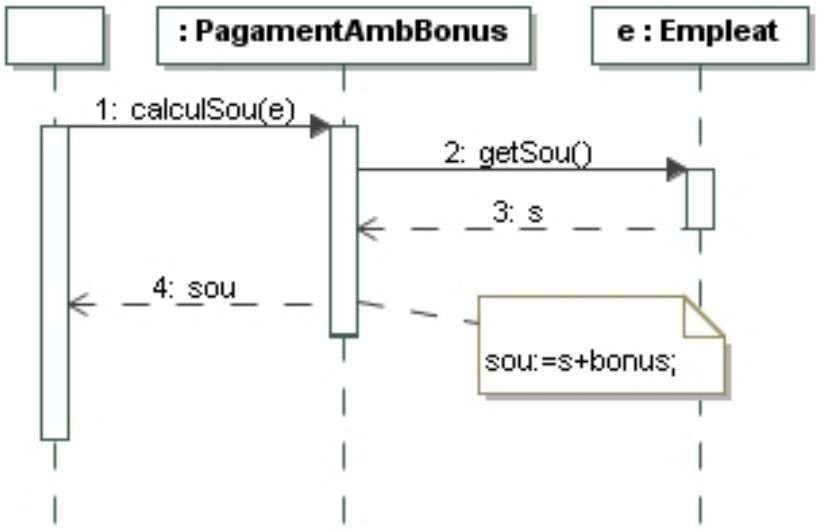
Es demana:

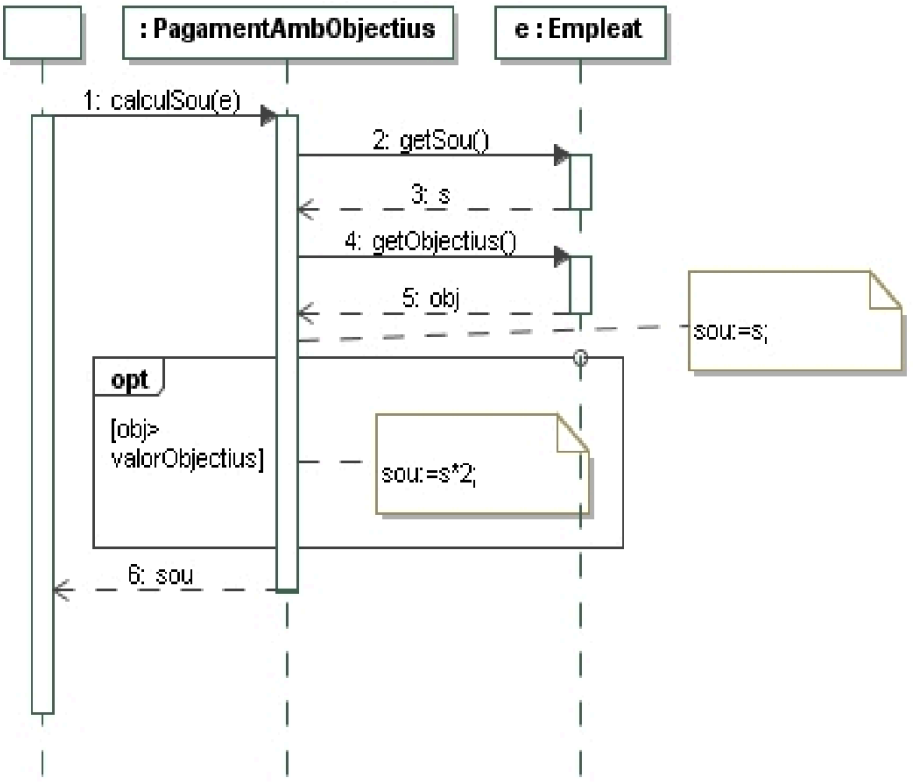
(a) Feu el diagrama de seqüència de les operacions anteriors. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència. No cal definir els algorismes d'ordenació. Podeu suposar l'existència d'una operació d'ordenació
(b) Feu el diagrama de classes de la capa de domini.

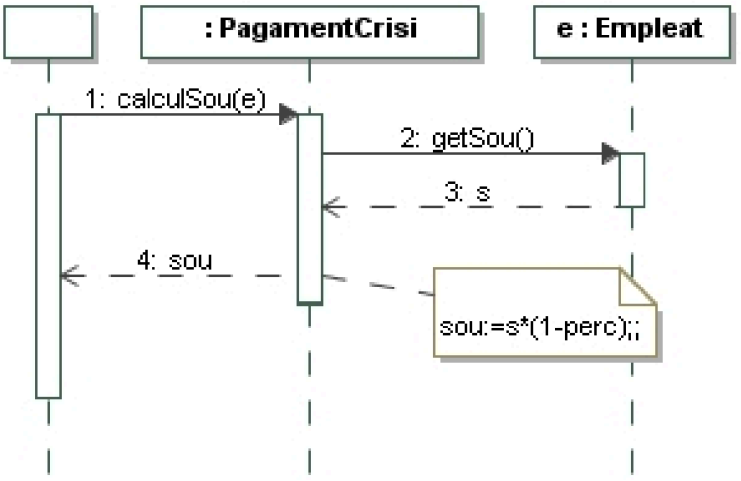
Solució

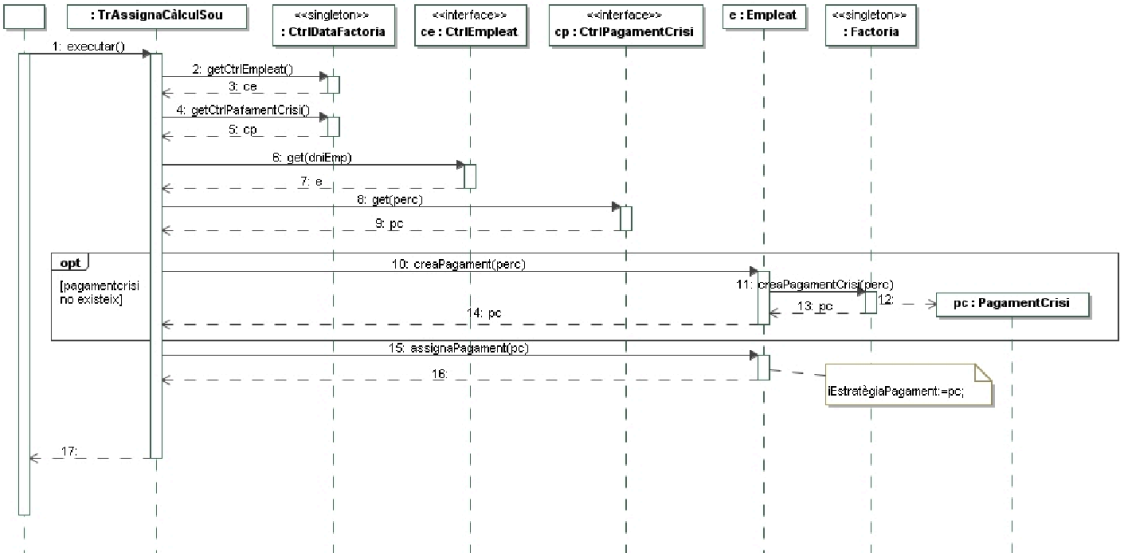
(a) Diagrames de seqüència



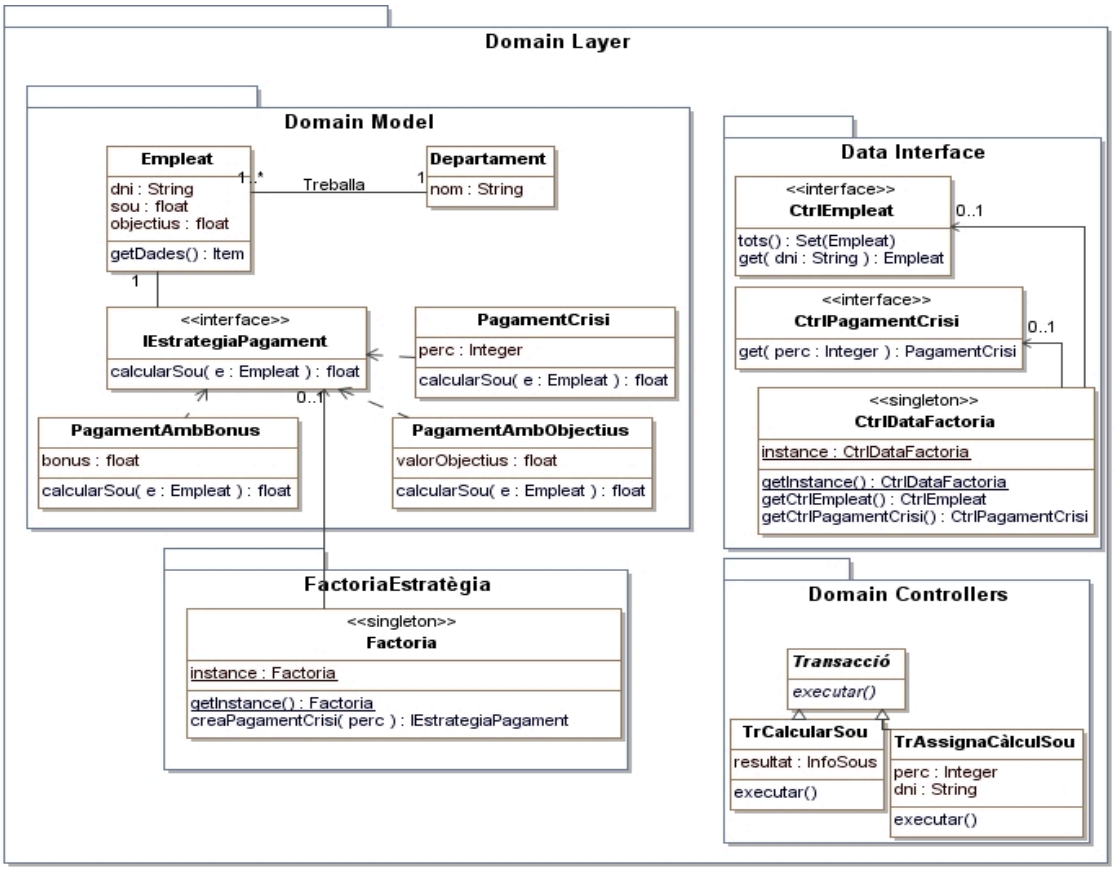



: PagamentAmbObjectius
e : Empleat
1: calculSou(e)
2: getSou()
3: s
4: getObjectius()
5: obj
sou:=s;
opt
[obj>
valorObjectius]
sou:=s*2;
6: sou


: PagamentCrisi
e : Empleat
1: calculSou(e)
2: getSou()
3: s
4: sou
sou:=s*(1-perc);;

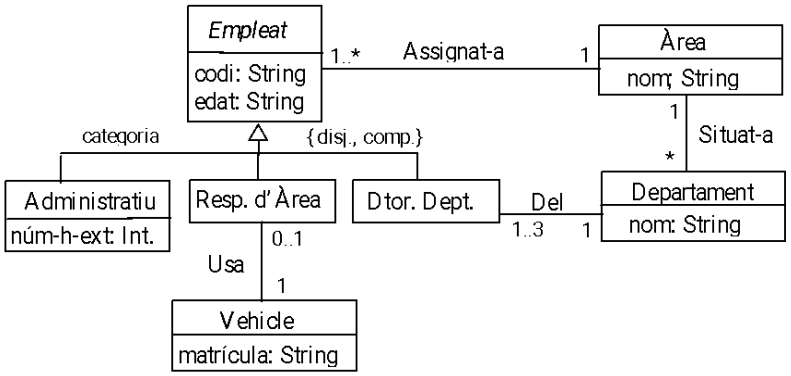


(b) Digrama de classes



Exercici 13 (Patrons)

Una empresa ens ha demanat que li dissenyem un sistema software per mantenir la informació de la categoria laboral dels seus empleats. Els empleats de l'empresa s'identifiquen per codi i estan assignats a una àrea geogràfica. Un empleat té una categoria laboral que pot ser: administratiu, responsable d'àrea o director de departament. Un empleat es pot contractar en qualsevol categoria però una vegada contractat només pot ascendir a la categoria superior (l'ordre de les categories és administratiu, responsable d'àrea i director de departament. Dels empleats que són administratius se'n guarda el nombre d'hores extres que han fet en un mes. Dels responsables d'àrea, se'n coneix el vehicle que poden utilitzar en els seus desplaçaments. Finalment, dels directors de departament cal enregistrar el departament que dirigeixen. A continuació disposeu de l'esquema conceptual del sistema:



R.I. Textuals:

- Claus de les classes no associatives: (Empleat, codi); (Àrea, nom); (Vehicle, matrícula); (Departament, nom).
- Un administratiu no pot fer més de 20 hores extres.
- No pot ser que hi hagi una àrea que no tingui assignat cap responsable d'àrea.
- Els directors de departament no poden tenir més de 30 anys.
- L'àrea d'assignació del director d'un departament ha de coincidir amb l'àrea on està situat el departament.

Considereu el cas d'ús d'especificació fer-director amb un únic esdeveniment amb el mateix nom.

context CapaDomini:: fer-director (codi-emp: String, nom-dept: String)
pre *empleatExisteix*: l'empleat amb codi-emp existeix.
pre *departamentExisteix*: el departament amb nom-dept existeix.
exc *empleatNoResp:* l'empleat amb codi-emp no és responsable d'àrea.
exc *massaDirectors:* el departament nom-dept ja té 3 directors.
exc *senseÀrea:* l'empleat és l'únic responsable d'àrea de l'àrea al que està assignat.
exc *massaGran:* l'empleat té més de 30 anys.
exc *diferentsÀrees:* l'àrea del departament nom-dept i la de l'empleat codi-emp són diferents.
post *ascendir*: l'empleat especificat passa a ser director de departament, del departament identificat per nom-dept. S'elimina l'associació Usa entre el responsable d'àrea i el vehicle. Es crea la instància de l'associació Del entre Departament i Dtor. Dept.

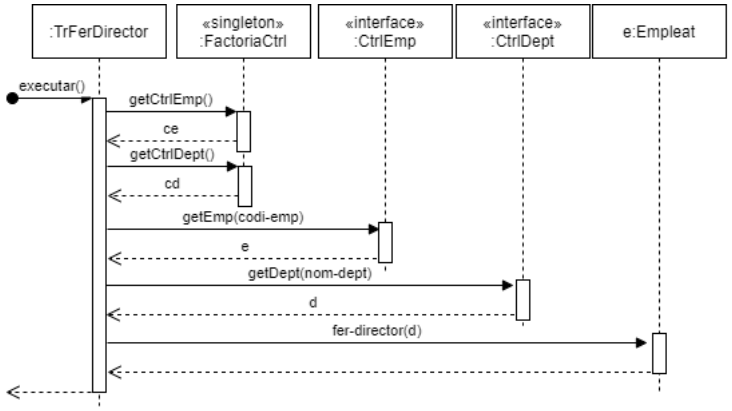
Es demana:

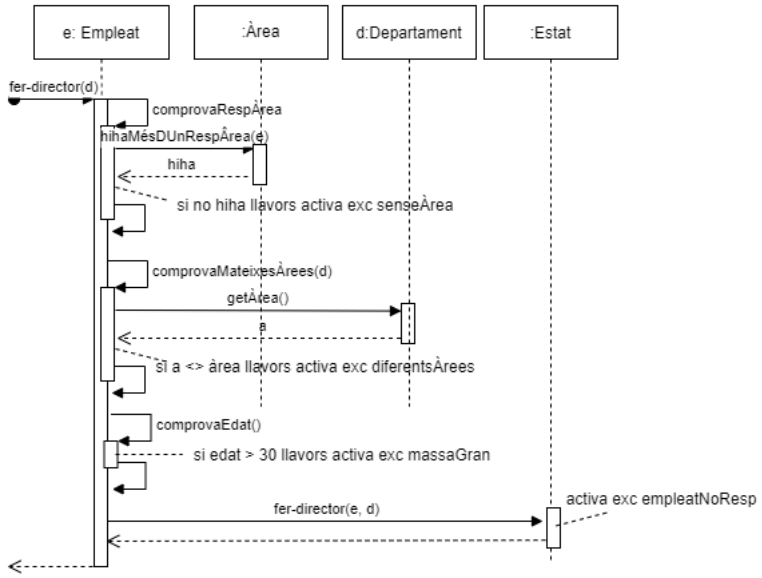
a) Diagrama de seqüència de l'operació del sistema *fer-director* , suposant que totes les navegabilitats són dobles. Cal que indiqueu amb comentaris tot el que no aparegui de forma explícita en els diagrames de seqüència.

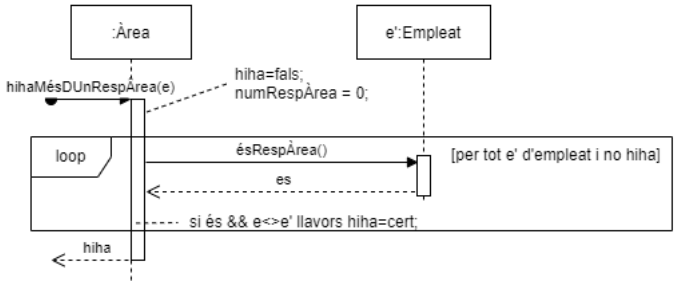
b) Diagrama de classes on es vegi l'aplicació del patró estat. No cal afegir les operacions.

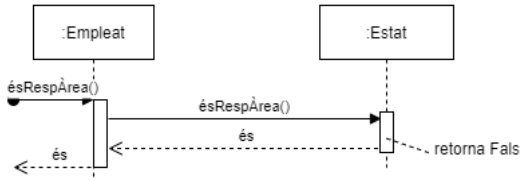
Solució

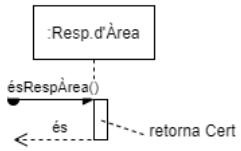
(a) Diagrames de seqüència

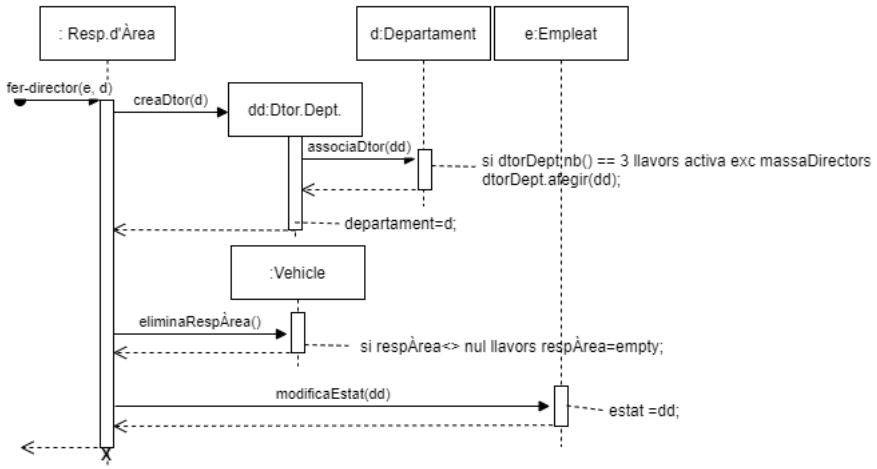




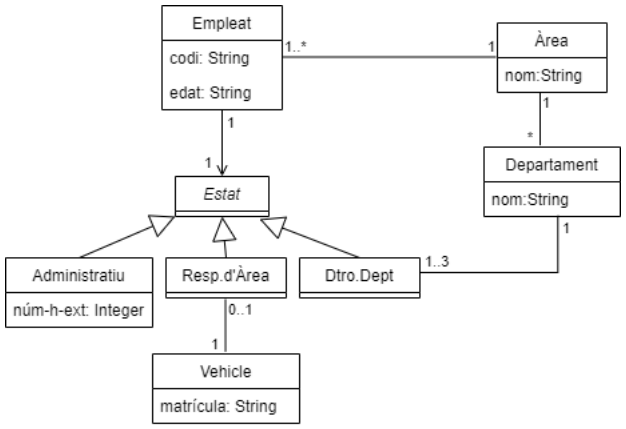








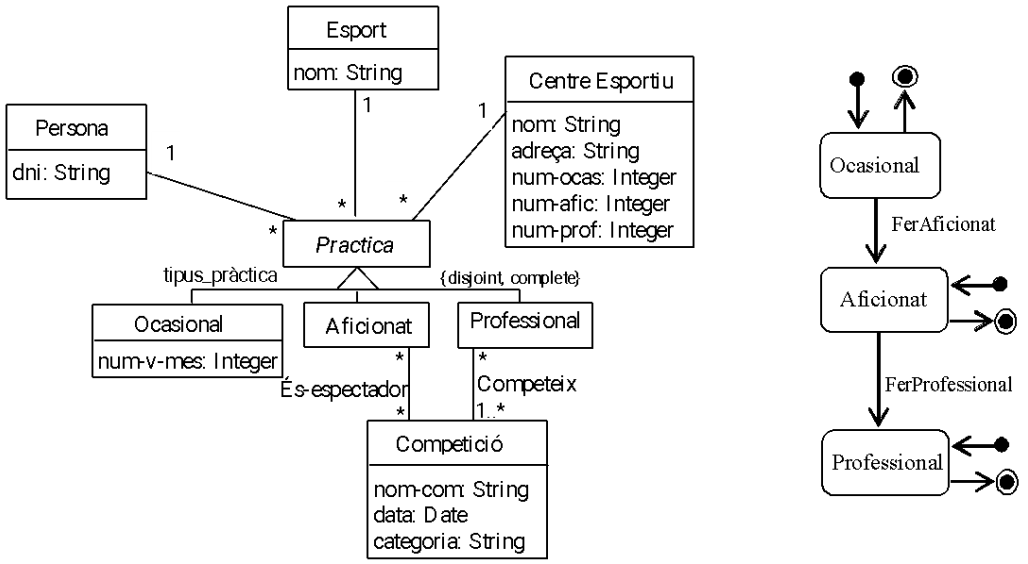
(b) Diagrama de classes



Exercici 14 (Patrons)

Una associació de centres esportius ens ha demanat que li dissenyem un sistema software per mantenir la informació de les persones que practiquen esports en els seus centres. Les persones poden practicar els esports de forma ocasional, com aficionats o com professionals. Les persones s'identifiquen per dni i els esports pel seu nom. Dels

centres esportius es guarda el nom, l'adreça i el nombre de persones que practiquen esport de forma ocasional, com aficionats i com professionals. De les persones que practiquen l'esport ocasionalment se'n guarda el nombre de vegades que hi van a fer esport en un mes. Dels aficionats, se'n coneix les competicions en les que són espectadors. Finalment, dels professionals cal enregistrar les competicions en les que competeixen. Les competicions s'identifiquen pel seu nom. A més també es guarda de les competicions la data de la celebració i la categoria. A continuació disposeu de l'esquema conceptual del sistema:



R.I. Textual:

- Claus classes no associatives: (Persona, dni); (Esport, nom); (Centre Esportiu, nom); (Competició, nom-com).
- No poden existir dues pràctiques per a la mateixa persona i esport.
- Una persona que practica un esport ocasionalment no pot anar més de quatre vegades al mes al centre esportiu.
- Una persona que practica un esport de forma professional no pot competir en competicions de categoria 'Aficionat'.
- De totes les pràctiques que té una persona, només una pot ser professional.
- En un centre esportiu es poden practicar com a molt 20 esports diferents.
- L'atribut num-ocas és igual al nombre de persones que practiquen esport en el centre de forma ocasional.
- L'atribut num-afic és igual al nombre de persones que practiquen esport en el centre de forma aficionada.
- L'atribut num-prof és igual al nombre de persones que practiquen esport en el centre de forma professional.

Considereu el cas d'ús d'especificació *FerProfessional* amb un únic esdeveniment amb el mateix nom. Després de fer l'assignació de responsabilitats a les capes disposem del contracte de l'operació de la capa de domini:

context CapaDomini:: *FerProfessional* (dni: String, nom-e: String, nom-com:String)
pre *competicióExisteix*: la competició amb nom nom-com existeix.
pre *personaExisteix*: la persona amb dni existeix.
pre *esportExisteix*: l'esport amb nom-e existeix.
exc *compd'Aficionat:* la competició nom-comp és de categoria 'Aficionat'.
exc *noAficionat:* la persona amb dni no practica l'esport amb nom-e com aficionat.
exc *personaJaTéPràcticaProfessional:* la persona amb dni ja té una practica com a professional.
post *passaAProfessional*: la persona amb dni passa a practicar l'esport nom-e de forma professional. S'elimina l' associació És_espectador entre l'aficionat i les competicions.

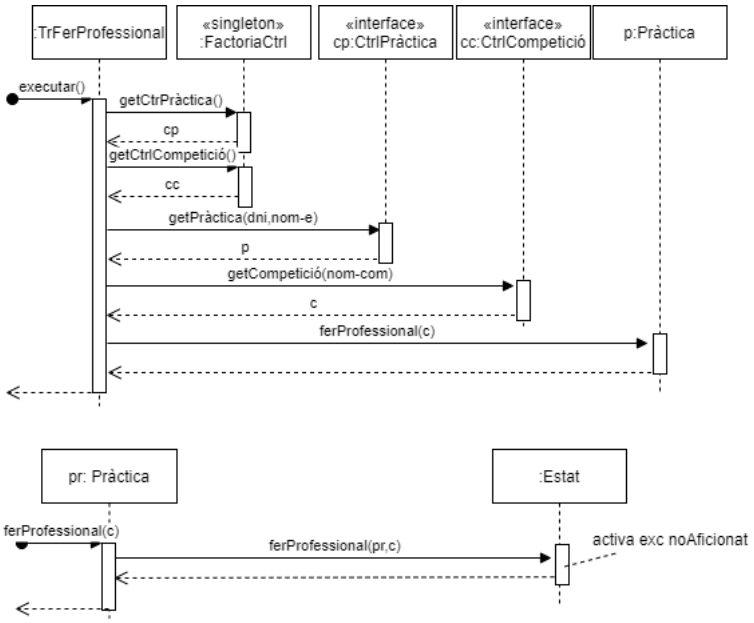
Es crea l'associació Competeix entre el professional i la competició amb nom_com. S'incrementa en 1 l'atribut num-prof del club esportiu de la pràctica i es decrementa en 1 l'atribut num-afic. S'eliminen les instàncies de l'associació Participa entre la persona i les competicions en les que era espectador com aficionat. Es crea la instància de l'associació Participa entre la persona i la competició nom-comp.

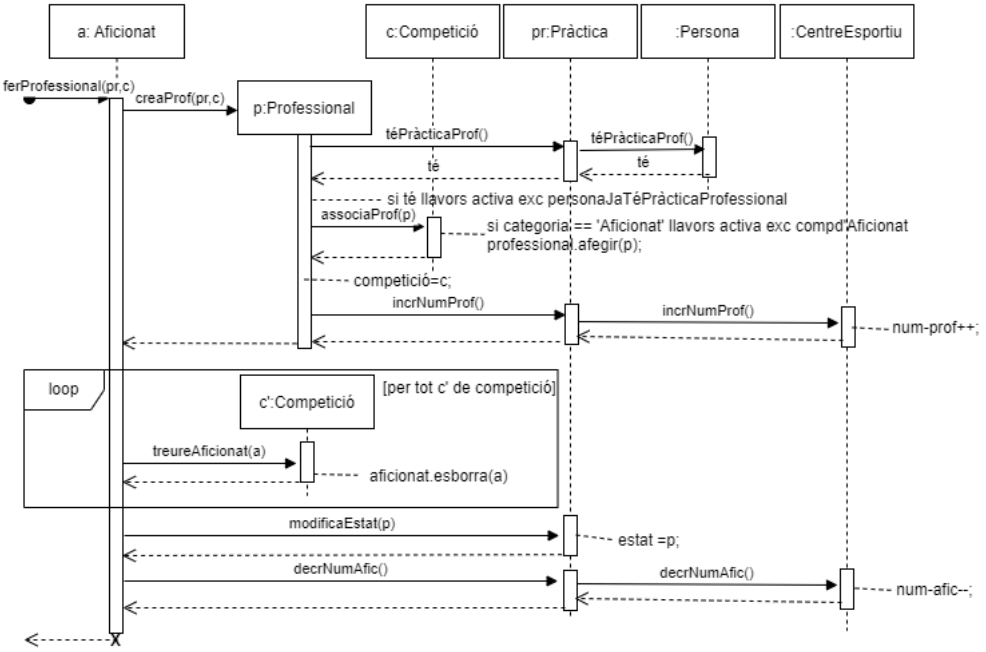
Es demana:

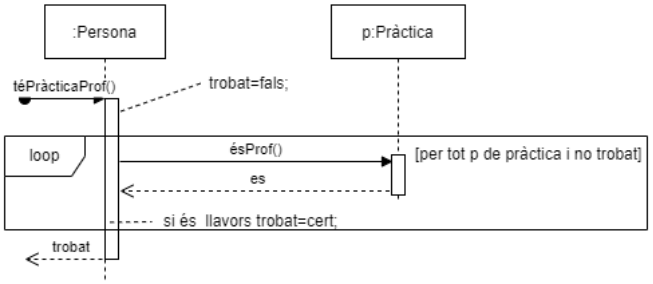
a) Diagrama de seqüència de l'operació *FerProfessional*, suposant que la navegabilitat de les associacions Competeix i És_espectador és doble. Cal que indiqueu amb comentaris tot el que no aparegui de forma explícita en els diagrames de seqüència.
b) Indiqueu la navegabilitat de la vostra solució (només paquet domain model).

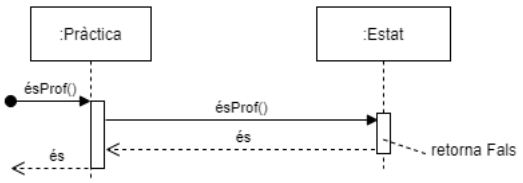
Solució

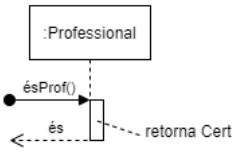
(a) Diagrames de seqüència



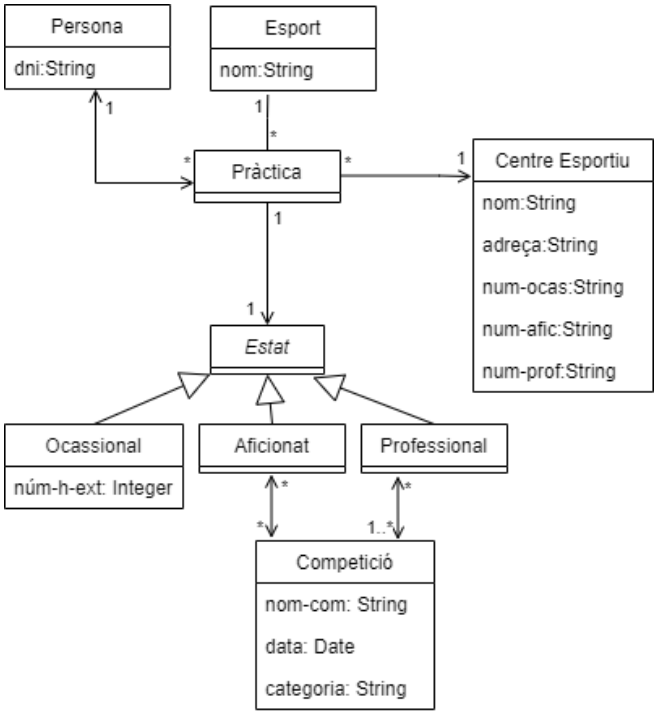

a: Aficionat
c:Competició
pr:Pràctica
:Persona
:CentreEsportiu
ferProfessional(pr,c)
creaProf(pr,c)
p:Professional
téPràcticaProf()
téPràcticaProf()
té
té
si té llavors activa exc personaJaTéPràcticaProfessional
associaProf(p)
si categoria == 'Aficionat' llavors activa exc compd'Aficionat
professional.afegir(p);
competició=c;
incrNumProf()
incrNumProf()
num-prof++;
loop
c':Competició
[per tot c' de competició]
treureAficionat(a)
aficionat.esborra(a)
modificaEstat(p)
estat =p;
decrNumAfic()
decrNumAfic()
num-afic--;


:Persona
p:Pràctica
téPràcticaProf()
trobat=fals;
loop
ésProf()
[per tot p de pràctica i no trobat]
es
si és llavors trobat=cert;
trobat


:Pràctica
:Estat
ésProf()
ésProf()
és
és
retorna Fals


:Professional
ésProf()
és
retorna Cert

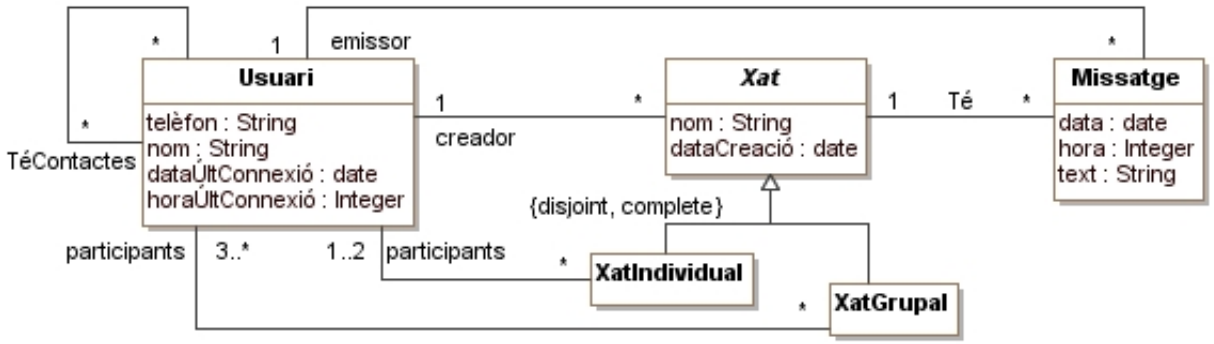
(b) Diagrama de classes



Exercici 18 (Patrons Factoria, Singleton i Adpatador)

Volem dissenyar un sistema software de missatgeria. Aquest software tindrà usuaris i els seus contactes. Cada usuari podrà crear un nombre indeterminat de xats. Els xats podran ser xats individuals i grupals. Els xats individuals tindran com a molt dos participants, el creador i un altre participant i els grupals tindran un mínim de 3 participants. Els xats tindran els missatges que envien els seus emissors (que han de ser participants del xat). A continuació disposeu d'un fragment de l'esquema conceptual del sistema a dissenyar:

Esquema conceptual de l'especificació:



R.I. Textuals:

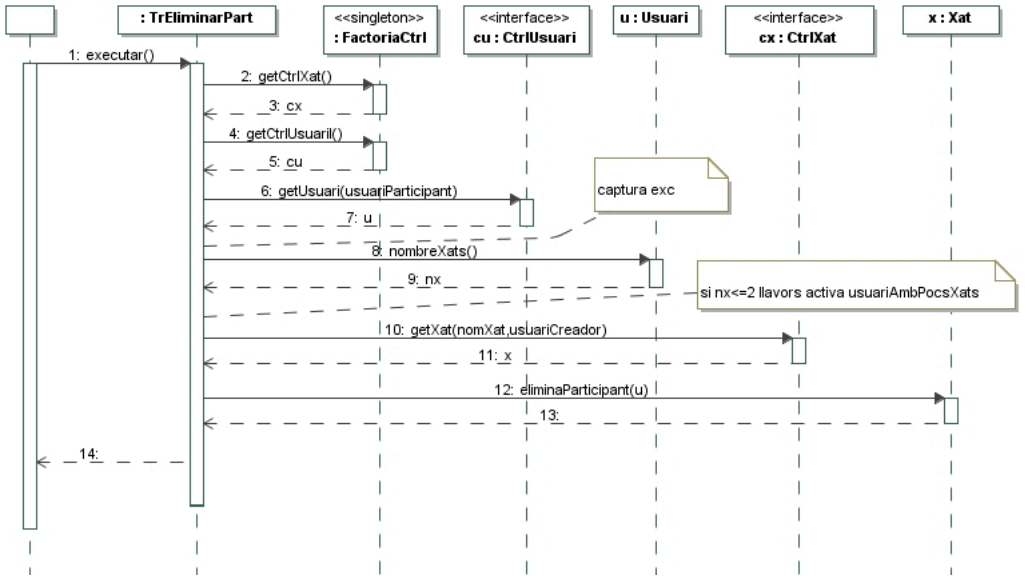
- Claus: (Usuari, telèfon); (Xat, Usuari::telèfon + nom); (Missatge, Xat::nom +Usuari::telèfon + data + hora);
- El creador d'un xat ha de ser un dels seus participants.
- L'emissor d'un missatge d'un xat ha de ser un dels participants del xat.
- La data d'un missatge ha de ser posterior a la data de la creació del xat.
- Els participants dels xats d'un usuari han de ser contactes de l'usuari del xat i ell mateix.
- Altres restriccions no rellevants pel problema

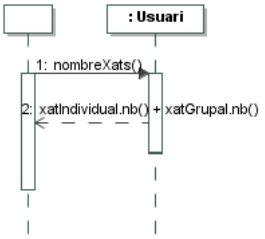
Volem dissenyar l'operació de la capa de domini *EliminarParticipantXat* per eliminar un dels participants del xat. A continuació disposeu del contracte de l'operació:

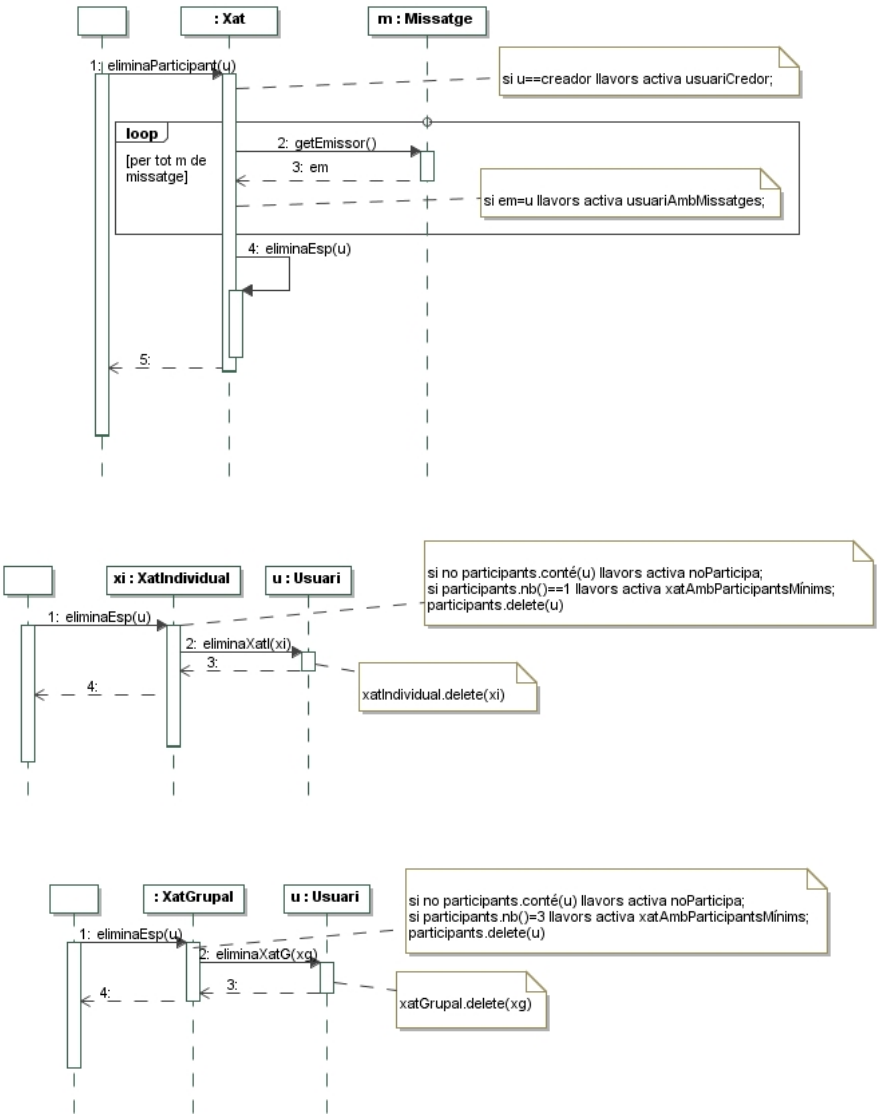
context CapaDomini :: EliminarParticipantXat (nomXat: String, usuariCreador: String, usuariParticipant: String)
exc *xatNoExisteix*: El xat identificat per *nomXat* i *usuariCreador* no existeix.
exc *noParticipa*: L'usuari *usuariParticipant* no és un participant del xat.
exc *usuariAmbMissatges*: L'*usuariParticipant* és emissor de missatges al xat.
exc *usuariCreador*: L'*usuariParticipant* és el creador del xat.
exc *usuariAmbPocsXats*: L'*usuariParticipant* participa en menys de 3 xats.
exc *xatAmbParticipantsMínims*: El xat té el nombre de participants mínim (3 si és grupal i 1 si és individual).
post *eliminaParticipant*: S'elimina el participant del xat.

Suposant que totes les navegabilitats són dobles, es demana:

- Diagrama de seqüència de l'operació EliminarParticipantXat de la capa de domini. Indiqueu clarament en el diagrama de seqüència els singleton, les interfaces i les operacions que són abstractes. Suposeu que totes les navegabilitats són dobles excepte les de les associacions entre els diferents tipus de xat i els seus participants. Aquestes navegabilitats van del tipus de xat cap a l'usuari.



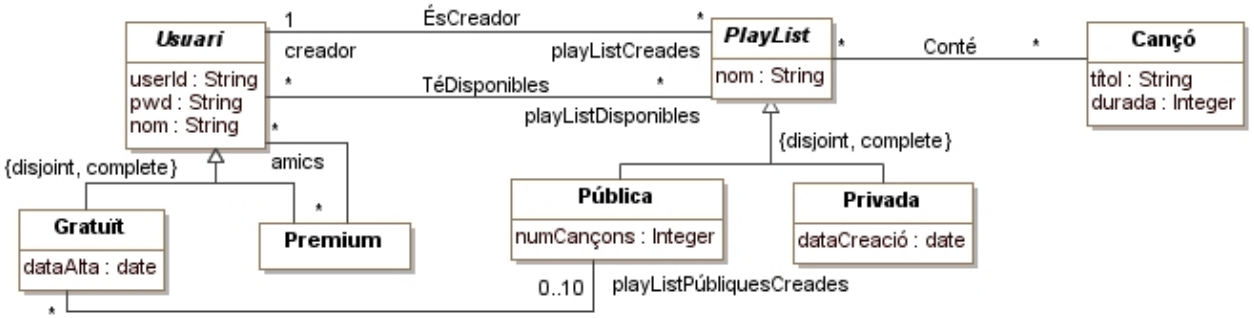




Exercici 19 (Patrons Factoria, Singleton i Adpatador)

Volem dissenyar un sistema software per gestionar la música d'un conjunt d'usuaris. Aquest software permetrà als seus usuaris crear playlists que podran ser públiques o privades. Les playlists tindran el seu usuari creador, un nom, i les seves cançons. Els usuaris podran ser gratuïts o premium. Els usuaris premium podran enregistrar altres usuaris amics. Cada usuari serà creador de les seves pròpies playlists però podrà tenir disponibles, a més, altres playlists dels seus amics. A continuació disposeu d'un fragment de l'esquema conceptual del sistema a dissenyar:

Esquema conceptual de l'especificació:



R.I. Textuals:
- Claus: (Usuari, userId); (PlayList, Usuari::userId del creador+ nom); (Cançó, títol);
- Un usuari premium no pot tenir com amic a sí mateix.
- Un usuari gratuït no pot tenir disponibles més de 20 playlists.
- Un usuari gratuït no pot tenir més de 30 cançons en les playlists que té disponibles.
- Les playlists que crea un usuari són un subconjunt de les playlists que té disponibles.
- Les playlists públiques creades per un usuari gratuït són un subconjunt de les playlists que ha creat.
- L'atribut numCançons d'una playlist pública és igual al nombre de cançons que té assignada la playlist.
- Altres restriccions no rellevants pel problema

Volem dissenyar dues operacions de la capa de domini. L'operació *assignarPlayList* de la classe *Usuari* que permet assignar una playlist com a disponible a un usuari i *crearICompartirPublicPlayList* per crear una playlist pública d'un usuari i assignar-la com a disponible als seus amics. A continuació disposeu del contracte de les dues operacions:

context Usuari :: assignarPlayList (pl: PlayList)
exc *playListAssignada*: La playlist *pl* ja la té disponible l'usuari *self*.
exc *gratuïtAmbTotesPL*: L'usuari *self* és gratuït i té disponibles 20 playlists.
exc *gratuïtiMoltesCançons*: L'usuari *self* és gratuït i el nombre de *cançons* del *pl* és més gran que 30.
post *assignacióPLDisponible*: S'assigna a l'usuari *self* la playlist *pl* com a playlist disponible.

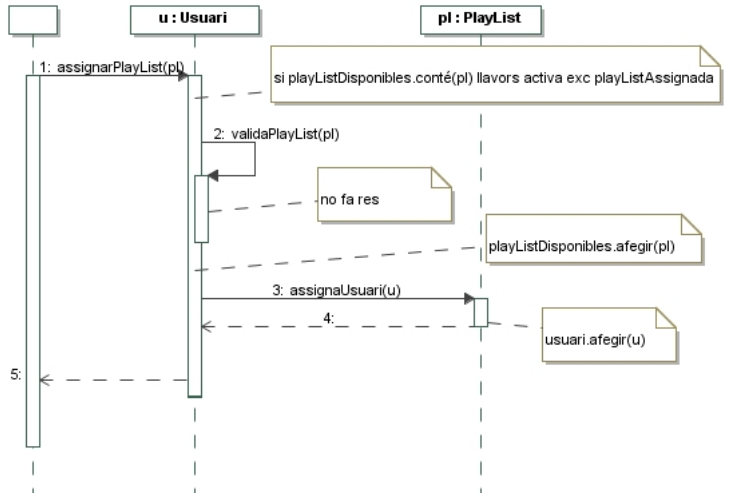
context CapaDomini :: crearICompartirPublicPlayList (userIdCreador: String, nomPL: String, cançons: Set(títol:String))
pre *cançonsExisteixen*: Les cançons del conjunt *cançons* existeixen i el nombre de cançons és més gran que 0.
exc *userNoExisteix*: L'usuari identificat per *userIdCreador* no existeix.
exc *playListExistent*: La playlist identificada per *userIdCreador* i *nomPL* ja existeix.
exc *gratuïtAmbTotesPL*: L'usuari identificat per *userIdCreador* és gratuït i té 20 playlists disponibles.
exc *gratuïtiMoltesCançons*: L'usuari identificat per *userIdCreador* és gratuït i el nombre de *cançons* és més gran que 30.
exc *gratuïtiMoltesPúbliquesCreades*: L'usuari identificat per *userIdCreador* és gratuït i ja té 10 playlists públiques creades.
post *creacióPL*: Es crea la playlist pública amb el nom, i s'assignen les cançons i l'usuari creador indicat als paràmetres.
post *assignacióPLDisponible*: S'assigna la playlist com a playlist disponible per a l'usuari creador.
post *assignacióPLPúblicaAGratuït*: Si l'usuari creador és gratuït llavors s'assigna la playlist creada a les *playListPúbliquesCreades* per l'usuari.
post *assignacióPLAmics*: Si l'usuari creador és premium, s'assigna la playlist creada a la llista de playlist disponibles de tots els seus usuaris amics que siguin premium i, si la

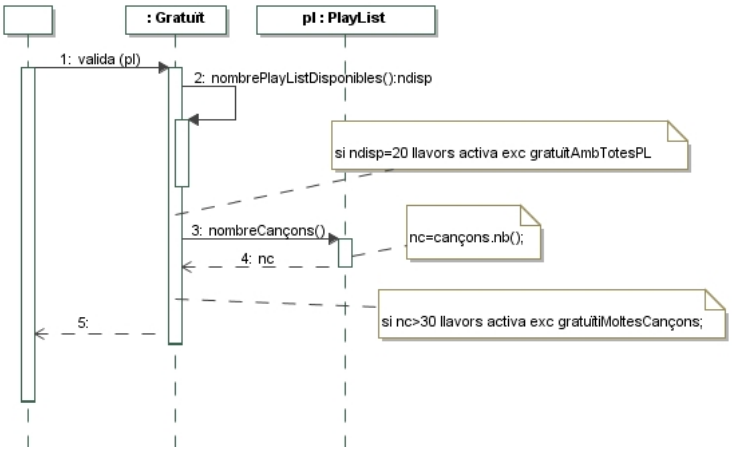
playlist pública creada té 30 o menys cançons, s'assigna la playlist també a tots els amics gratuïts que no tinguin 20 playlists disponibles.

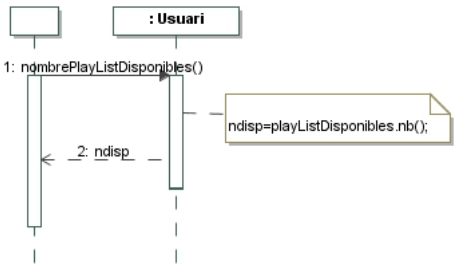
Suposant que les navegabilitats inicials de l'associació TéDisponibles és doble i la navegabilitat d'Usuari (rol creador) va cap a PlayList, la de Gratuït va cap a Pública i la de l'associació Conté va de PlayList a Cançó (recordeu que com a conseqüència del vostre disseny, si és necessari, podeu afegir noves navegabilitats), es demana:

a) Diagrama de seqüència de l'operació assignarPlayList de la classe Usuari. Indiqueu clarament en el diagrama de seqüència els singleton, les interfaces i les operacions que són abstractes.
b) Diagrama de seqüència de l'operació *crearICompartirPublicPlayList* de la capa de domini. Indiqueu clarament en el diagrama de seqüència els singleton, les interfaces i les operacions que són abstractes. Es valorarà especialment la reutilització de les operacions que dissenyeu. En concret, aquest sistema haurà de permetre que un usuari premium pugui crear una playlist pública sense compartir-la amb els seus amics.

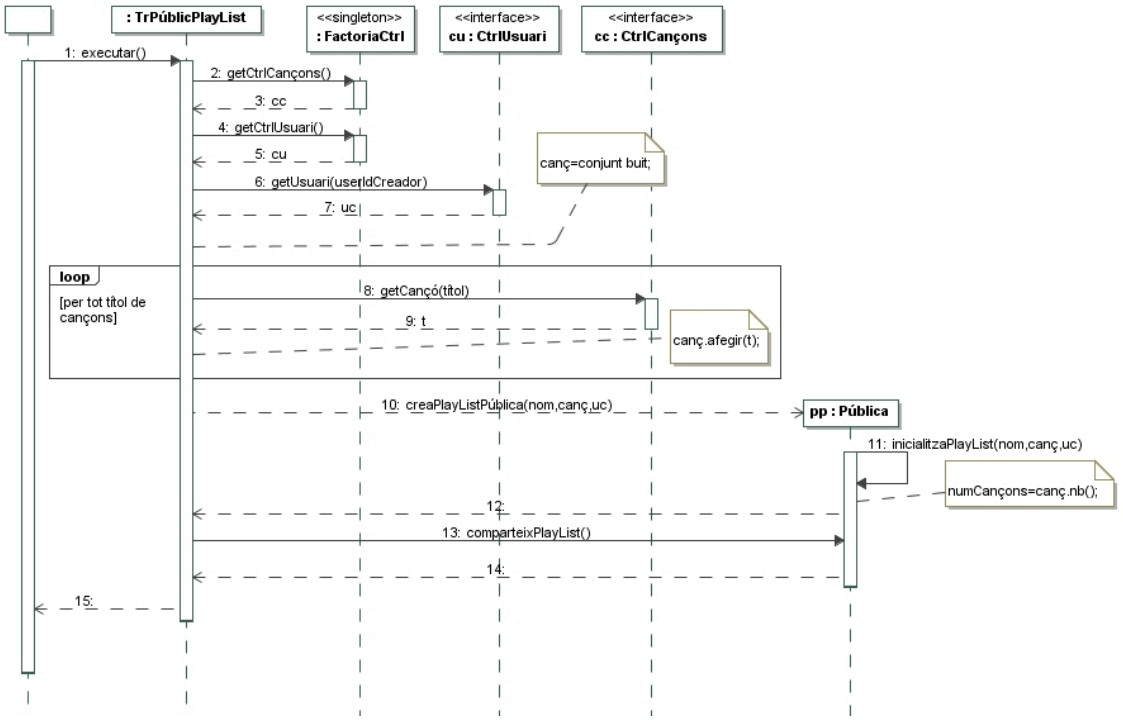
a)

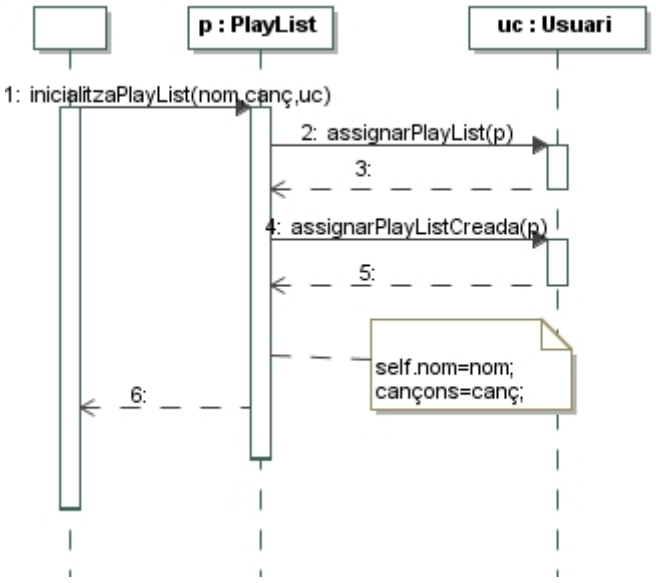


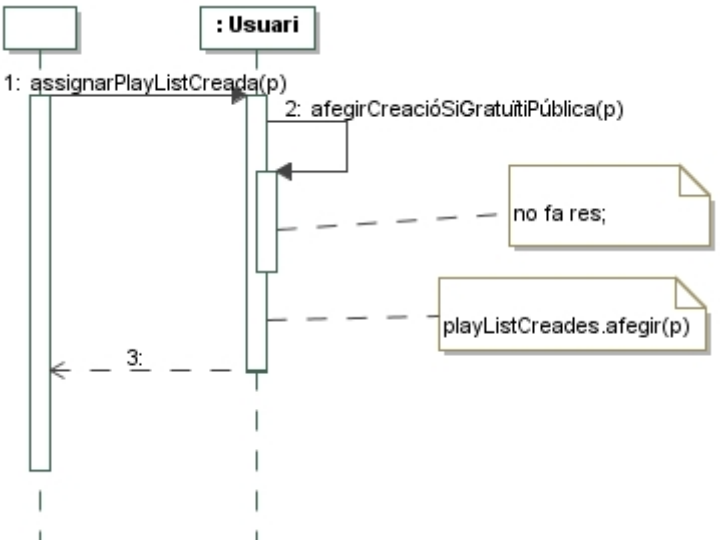


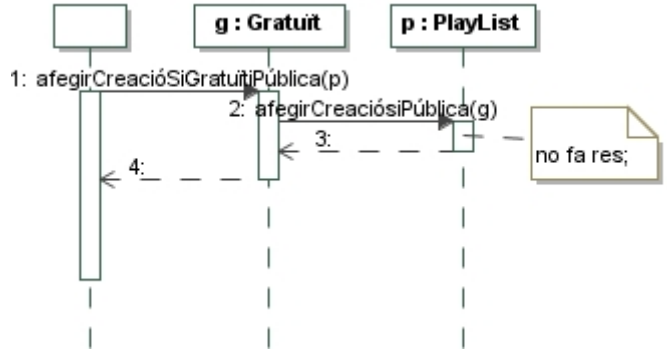

: Usuari
1: nombrePlayListDisponibles()
ndisp=playListDisponibles.nb();
2: ndisp

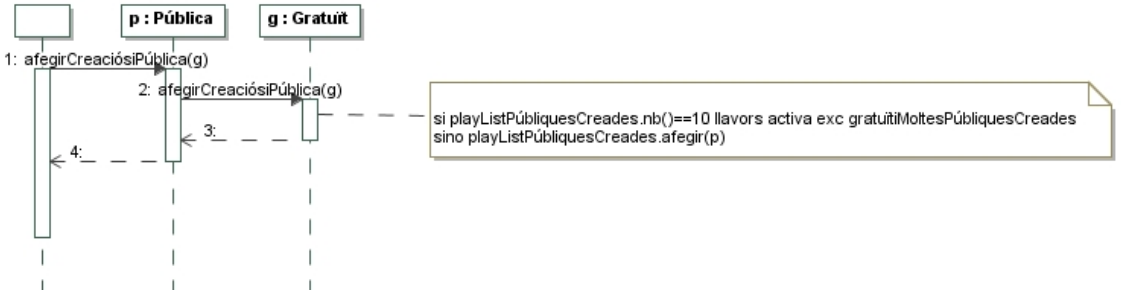
b)

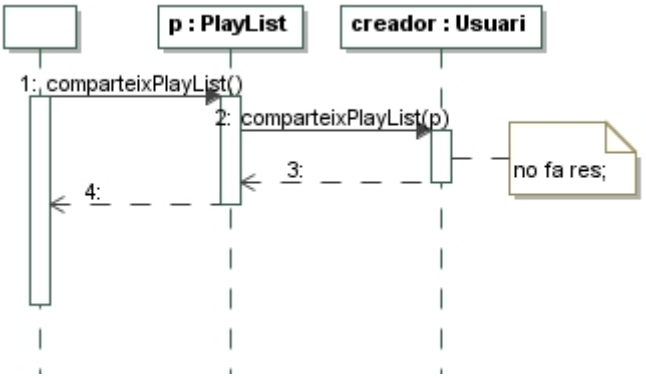

: TrPúblicPlayList
<<singleton>>
: FactoriaCtrl
<<interface>>
cu : CtrlUsuari
<<interface>>
cc : CtrlCançons
1: executar()
2: getCtrlCançons()
3: cc
4: getCtrlUsuari()
5: cu
6: getUsuari(userIdCreador)
7: uc
cançó=conjunt buit;
loop
[per tot títol de cançons]
8: getCançó(títol)
9: t
cançó.afegir(t);
10: creaPlayListPública(nom,cançó,uc)
pp : Pública
11: inicialitzaPlayList(nom,cançó,uc)
numCançons=cançó.nb();
12:
13: comparteixPlayList()
14:
15:

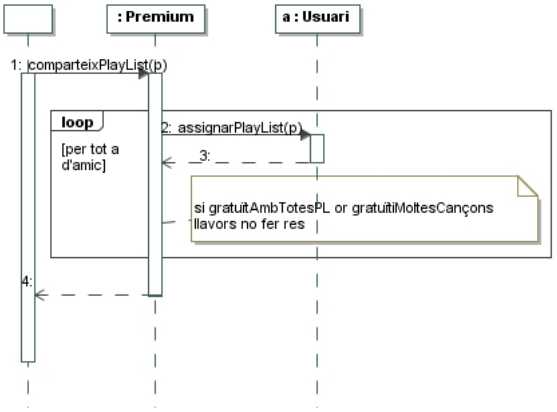

p : PlayList
uc : Usuari
1: inicialitzaPlayList(nom,cançó,uc)
2: assignarPlayList(p)
3:
4: assignarPlayListCreada(p)
5:
self.nom=nom;
cançons=cançó;
6:


: Usuari
1: assignarPlayListCreada(p)
2: afegirCreacióSiGratuïtiPública(p)
no fa res;
playListCreades.afegir(p)
3:


g : Gratuït
p : PlayList
1: afegirCreacióSiGratuïtiPública(p)
2: afegirCreaciósiPública(g)
3:
no fa res;
4:


p : Pública
g : Gratuït
1: afegirCreaciósiPública(g)
2: afegirCreaciósiPública(g)
3:
4:
si playListPúbliquesCreades.nb()==10 llavors activa exc gratuïtiMoltesPúbliquesCreades
sino playListPúbliquesCreades.afegir(p)


p : PlayList
creador : Usuari
1: comparteixPlayList()
2: comparteixPlayList(p)
3:
no fa res;
4:


: Premium
a : Usuari
1: comparteixPlayList(p)
loop
[per tot a d'amic]
2: assignarPlayList(p)
3:
si gratuitAmbTotesPL or gratuitiMoltesCançons
llavors no fer res
4: