

Propietats dels objectes

- Cada expressió OCL:
 - s'escriu en el context d'una instància d'un tipus determinat
 - defineix una propietat d'aquesta instància
- Una propietat pot fer referència a:
 - atributs d'una classe d'objectes
 - navegació a través de les associacions

Propietats dels atributs d'una classe d'objectes:

- Propietat: edat d'una persona -- enter

context Persona inv:
self.edat
l'expressió ha de ser certa per a totes les instàncies de la classe
instància contextual de l'expressió (punt de partida)

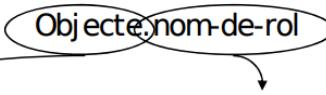
- Restricció de la propietat: "l'edat de les persones ha de ser superior o igual a zero"

context Persona inv: o, alternativament: context p:Persona inv:
self.edat ≥ 0 p.edat ≥ 0

Propietats dels objectes (II)

Propietats d'una navegació a través d'associacions:

Partint d'un objecte concret, podem navegar a través de les associacions del Model Conceptual per referir-nos a d'altres objectes i a les seves propietats

- Com es navega? 
objecte de partida nom de rol d'una associació de l'objecte

Resultat: conjunt d'objectes de l'altre extrem de nom-de-rol

- si no hi ha nom de rol especificat, es pot usar el nom de la classe d'objectes de l'altre extrem de l'associació (amb minúscules)

- Exemples de navegació:

context Empresa inv:
self.director
self.director.nom
self.empleat
self.empleat.espos
-- director de l'empresa -- Persona
-- nom del director -- String
-- empleats de l'empresa -- set(Persona)
-- esposos dels empleats -- set(Persona)

Col·leccions: conjunts, bosses i seqüències

Una col·lecció d'elements pot ser del tipus:

- Conjunt: cada element ocorre una única vegada a la col·lecció
- Bossa (multiconjunt): la col·lecció pot contenir elements repetits
- Seqüència: bossa on els elements estan ordenats

Exemple:

- "nombre de treballadors diferents que treballen per a una persona"

context Persona::num-treb:Integer derive:

self.empresesDirigides.empleat ->size()

Incorrecte: el resultat és una bossa i pot contenir repetits.

context Persona::num-treb:Integer derive:
self.empresesDirigides.empleat ->asSet() ->size()

Regles de navegació:

- *si la multiplicitat de l'associació és 1 el resultat és un objecte (o un conjunt d'un únic objecte).*
- *si la multiplicitat de l'associació és >1 el resultat és un conjunt.*
- *si es navega per més d'una associació i la multiplicitat d'alguna d'elles és >1 el resultat és una bossa, encara que de vegades pot ser un conjunt.*

Operacions bàsiques sobre col·leccions (I)

Select:

especifica un subconjunt de la col·lecció

- "persones majors de 50 anys que treballen a una empresa"

context Empresa inv:

self.empleat ->select(edat>50)

context Empresa inv:

self.empleat ->select(p | p.edat>50)

context Empresa inv:

self.empleat ->select(p: Persona | p.edat>50)

Collect:

especifica una col·lecció que es deriva d'una altra, però que conté objectes diferents

- "edats (amb repetits) dels empleats d'una empresa"

context Empresa inv:

self.empleat ->collect(dataNaixement)

versió simplificada: self.empleat.dataNaixement

Operacions bàsiques sobre col·leccions (II)

forAll: expressió que han de satisfer tots els elements

- "tots els empleats de l'empresa s'anomenen Jack"
context Empresa inv:
self.empleat ->forAll(nome='Jack')
- "la clau externa d'Empresa és el seu nom"
context Empresa inv:
Empresa.allInstances() ->forAll(e1,e2 | e1<>e2 implies e1.nom<>e2.nom)

Exists: condició que satisfà almenys un element

- "com a mínim un empleat de l'empresa s'ha de dir Jack"
context Empresa inv:
self.empleat ->exists(nome='Jack')

IsUnique: retorna cert si l'expressió s'avalua a un valor diferent per cada element de la col·lecció

- "la clau externa d'Empresa és el seu nom"
context Empresa inv:
Empresa.allInstances() ->isUnique(nome)

Ús d'OCL a l'esquema conceptual de dades

• Definició de restriccions textuais d'integritat

- "Les persones casades han de ser majors d'edat"
context Persona inv:
self.esposa ->notEmpty() implies self.esposa.edat >= 18
and self.espòs ->notEmpty() implies self.espòs.edat >= 18
- "Una empresa té com a màxim 50 empleats"
context Empresa inv:
self.empleat ->size() <= 50
- "Una persona no pot tenir alhora un espòs i una esposa"
context Persona inv:
not ((self.esposa ->notEmpty()) and (self.espòs ->notEmpty()))

• Definició d'atributs derivats

- "definició de l'atribut derivat nombred'Empleats"
context Empresa::nombred'Empleats : Integer
derive: self.empleat ->size()
- "definició de l'atribut derivat estàAturat"
context Persona::estàAturat : Boolean
derive: self.contractador->isEmpty()

Observació: Noteu la diferència en la sintaxi de la part del context segons si es tracta d'una restricció d'integritat (inv) o d'un atribut derivat (derive)

Navegació per classes associatives

Navegació a una classe associativa:

Es fa servir el nom de la classe associativa (amb minúscula)

- "els sous de les persones que treballen a la UPC han de ser més alts que 1000"
context Empresa inv:
(self.nom = 'UPC' implies self.treball ->forAll (t | t.salari > 1000))

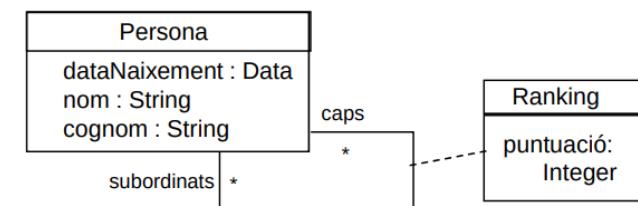
Navegació des d'una classe associativa:

S'usa el nom de rol de l'extrem cap on es vol navegar

Si no s'ha especificat nom de rol, s'usa el nom de la classe.

- "les persones que treballen no poden estar aturades"
context Treball inv:
self.empleat.estàAturat = false
- "les dues persones d'un casament no poden ser la mateixa"
context Casament inv:
self.esposa <> self.espòs

Navegació a classes associatives d'associacions recursives

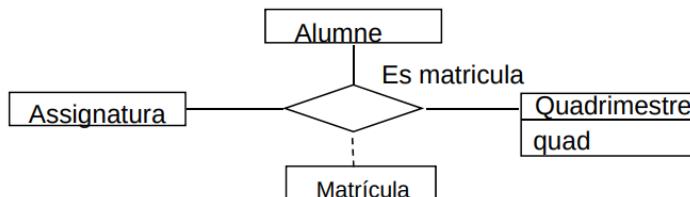


- Quan es navega cap a una classe associativa d'una associació recursiva hi ha dues possibilitats depenen de la direcció

Ex: es pot navegar de Persona a Ranking en direcció a caps o en direcció a subordinats

- Per distingir s'usa el nom de rol cap on es vol navegar entre claudàtors
 - "la suma de puntuacions que pertanyen a la col·lecció de subordinats ha de ser positiva"
context Persona inv:
self.ranking[subordinats] ->sum() > 0
 - "la suma de puntuacions que pertanyen a la col·lecció de caps ha de ser positiva"
context Persona inv:
self.ranking[caps] ->sum() > 0

Navegació per associacions ternàries (amb associativa)



- Navegació cap a la classe associativa o des de la classe associativa: Similar al cas de les associacions binàries.

context Alumne inv:

self.matrícula --matrícules d'un alumne

context Matrícula inv:

self.alumne -- alumne d'una matrícula

- Navegació d'una classe a una altra classe de l'associació.

context Alumne inv:

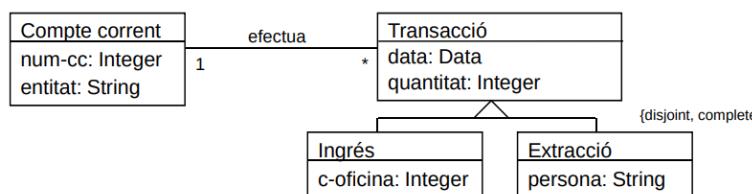
self.matrícula.assignatura -- assignatures de les que s'ha matriculat un alumne
(el resultat és una bossa)

- Navegació d'una classe a una altra classe de l'associació amb una condició de selecció que involucra la classe restant.

context Alumne inv:

self.matrícula->select(m | m.quadrimestre.quad='1Q-04-05').assignatura
-- assignatures de les que s'ha matriculat un alumne el quadrimestre '1Q-04-05'

Generalització / Especialització (herència)



Navegació:

- Accés directe a les propietats de la superclasse

context Ingrés inv:
self.compte-corrent.num-cc -- número de compte d'un ingrés

- Accés a les propietats definides al nivell de subclasse

context CompteCorrent inv:
self.transacció->select(oclIsTypeOf(Extracció)).oclAsType(Extracció).persona->asSet()
-- persones que han tret diners d'un compte corrent

Aspecte addicional:

- Selecció d'objectes que pertanyen a la subclasse

context CompteCorrent inv:
self.transacció->select(oclIsTypeOf(Ingrés)) -- ingressos d'un compte

Definició de variables: “let” i “def”

- Let:** Es fa servir en expressions que tenen subexpressions que s'usen més d'una vegada

- *“La suma de salari és menor que 1000 per les persones de menys de 40 anys i més gran o igual que 1000 per la resta”*

context Persona inv:

```

let ingressos : Integer = self.treball.salari ->sum() in
if self.edat <40 then ingressos <1000 else ingressos >=1000
endif
  
```

- Def:** Es pot reutilitzar en altres expressions ocl (com si fos un atribut de la classe)

context Persona

```
def: ingressos : Integer = self.treball.salari ->sum()
```

context Persona inv:

```

if self.edat <40 then ingressos <1000 else ingressos >=1000
endif
  
```

Com especificar en OCL: exemples

- “L'espòs i l'esposa d'un matrimoni han de ser majors d'edat”**

context Casament inv:
self.esposa.edat >=18 and self.espòs.edat >=18 -- és preferible a

context Persona inv:

self.esposa->notEmpty() implies self.esposa.edat >=18 and
self.espòs->notEmpty() implies self.espòs.edat >=18

- “Totes les persones han de ser majors d'edat”**

context Persona inv:
self.edat >=18 -- no és equivalent a

context Empresa inv:

self.empleat ->forAll (edat >=18)

- “Ningú no pot ser director i empleat d'una empresa”**

context Empresa inv:
not(self.empleat ->includes(self.director))

context Persona inv:

not(self.empresesDirigides.empleat ->includes(self))

-- quina és preferible en aquest cas?

- Definició de precondicions
Condicions que, de no cumplir-se, impedeixen l'execució de l'operació.
- Definició de postcondicions
Condicions que, un cop executada l'operació, són certes.
- Definició de bodys
Retorn d'una operació consultora

Definició d'expressions: "oclIsNew()" i "@pre"

- **oclIsNew():** Es fa servir, en una postcondició, per indicar que un objecte ha estat creat com a efecte de l'operació.
 - *"L'operació nouCasament ha de crear un objecte Casament"*
context Sistema::nouCasament(dni: String, ...):
pre: ...
post: Casament.allInstances()->exists(c| c.oclIsNew() and ...)
- **@pre:** Es fa servir, en una postcondició, per comprovar si un objecte existia abans d'executar l'operació.
 - *"L'operació nouCasament ha de crear una Persona amb el dni indicat si aquesta no existia"*
context Sistema::nouCasament(dni: String, ...):
pre: ...
post: if (not Persona.allInstances()@pre->exists(p| p.dni =dni))
then Persona.allInstances()->exists(p| ...)

Operació	Notació	Resultat
or	a or b	booleà
and	a and b	booleà
or exclusiu	a xor b	booleà
negació	not a	booleà
igualtat	a = b	booleà
desigualtat	a <> b	booleà
implicació	a implies b	booleà
if-then-else-endif	if a then b else b' endif	tipus de b o b'

Operacions estàndard de tipus string

Operació	Notació	Resultat
concatenació	string.concat(string)	string
tamany	string.size()	integer
substring	string.substring(int,int)	string
igualtat	string1 = string2	booleà
desigualtat	string1 <> string2	booleà

Operacions estàndard d'una classe d'objectes

Operació	Resultat
allInstances	retorna el conjunt de totes els elements de la classe d'objectes

Operacions estàndard de tipus col·lecció

Operació	Resultat
size()	nombre d'elements de la col·lecció
count(object)	nombre d'ocurrències de l'objecte
includes(object)	cert si l'objecte pertany a la col·lecció
includesAll(collection)	cert si els elements del paràmetre <i>collection</i> són a la col·lecció actual
excludes(object)	cert si l'objecte no pertany a la col·lecció
excludesAll(collection)	cert si els elements del paràmetre <i>collection</i> no són a la col·lecció actual
isEmpty()	cert si la col·lecció és buida
notEmpty()	cert si la col·lecció no és buida
sum()	suma de tots els elements
exists(expression)	<i>expression</i> és cert per algun element?
forAll(expression)	<i>expression</i> és cert per tots els elements?
isUnique(expression)	cert si <i>expression</i> avalua a un valor diferent per cada element de la col·lecció actual

Operacions estàndard (específiques) de tipus conjunt i bossa

Operació	Resultat
select(expression)	selecciona el subconjunt d'elements del conjunt o bossa actual per als quals <i>expression</i> és cert
reject(expression)	elimina el subconjunt d'elements del conjunt o bossa actual per als quals <i>expression</i> és cert
union(set)	resultat d'unir el conjunt o bossa actual amb el <i>set</i>
intersection(set)	resultat de la intersecció del conjunt o bossa actual amb el <i>bag</i>
union(bag)	resultat d'unir el conjunt o bossa actual amb el <i>bag</i>
intersection(bag)	resultat de la intersecció del conjunt o bossa actual amb el <i>bag</i>
asSet()	resultat d'eliminar repetits del conjunt o bossa actual

Artefactes usats a l'esquema del comportament del sistema

Diagrames de seqüència del sistema (DSS): Mostra les interaccions entre els actors i el sistema.

- **Punt de partida:** La seqüència d'esdeveniments en un escenari d'un cas d'ús
- **Idea bàsica:** L'actor genera esdeveniments cap al sistema, que exigeixen l'execució d'alguna operació com a resposta.
- **Objectiu/utilitat:** Permeten identificar les operacions del sistema i especificar la dinàmica dels escenaris.

Contractes de les operacions del sistema: Descriuen l'efecte de les operacions del sistema

- **Punt de partida:** Operacions identificades del sistema
- **Idea bàsica:** especificació Pre/Post per descriure els canvis d'estat que es produueixen com a conseqüència de l'execució
- **Objectiu/utilitat:** Permeten establir/fixar el comportament desitjat de les operacions

Només per veure l'aspecte dels DSS i els contractes:



Cas d'ús: Afegir-Persona

1. La persona indica al sistema el seu nom i la seva adreça
2. El Sistema enregistra el nom i l'adreça de la persona
3. La persona indica al sistema l'idioma que parla
4. El Sistema enregistra l'idioma parlat per la persona

Contractes:

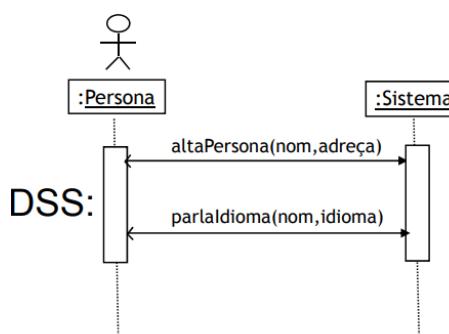
Operació: altaPersona(nom,adreça)

Precondicions:

Postcondicions:

- s'ha creat d'un objecte Persona amb el nom i l'adreça especificats

Sortida:



Operació: parladioma(nom,nomId)

Precondicions:

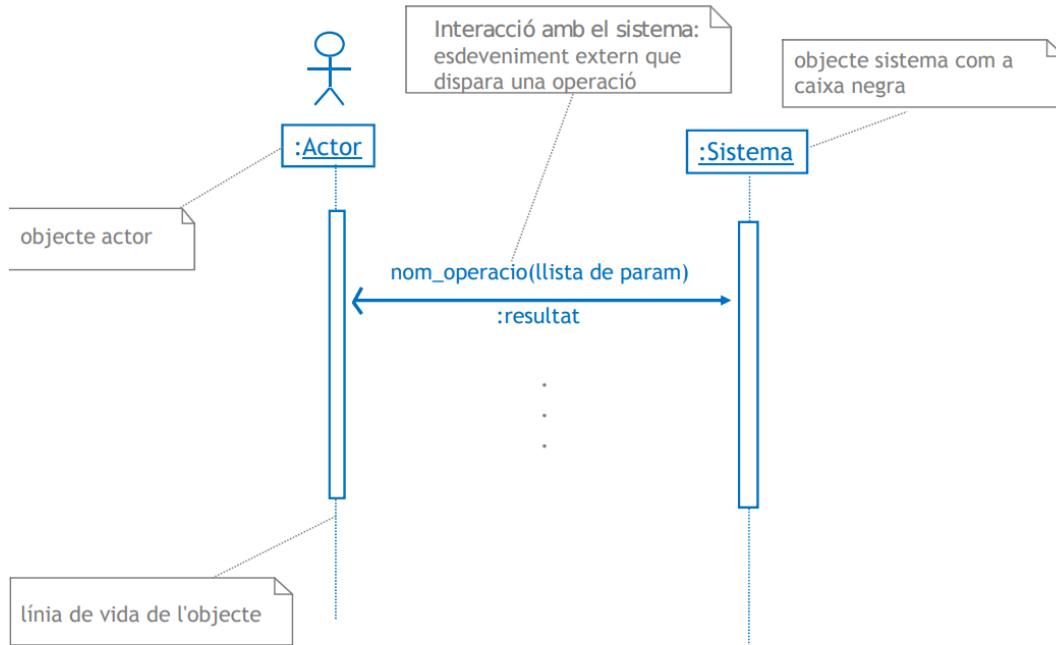
- l'idioma identificat per nomId existeix

Postcondicions:

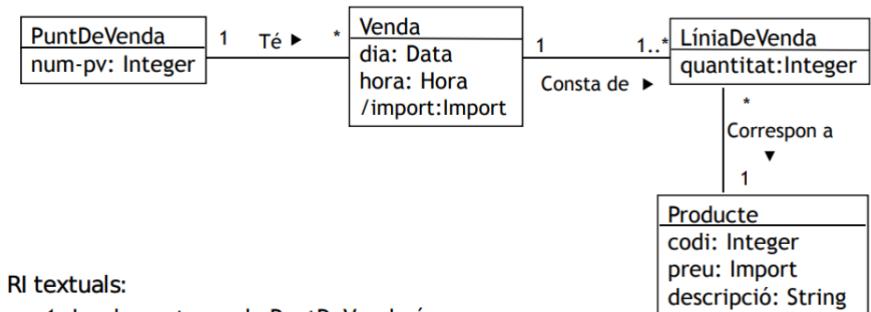
- creació d'una associació Parla entre la persona amb nom=nom i l'idioma nom=nomId

Sortida:

Elements bàsics d'un DSS:



Recordem l'Exemple del caixers i els punt de venda:



RI textuels:

- 1- La clau externa de PuntDeVenda és num-pv
- 2- La clau externa de Producte és codi
- 3- Un punt de venda no pot tenir més d'una venda amb el mateix dia i hora
(per tant una instància de venda queda únicament determinada per tres valors de num-pv, dia i hora)

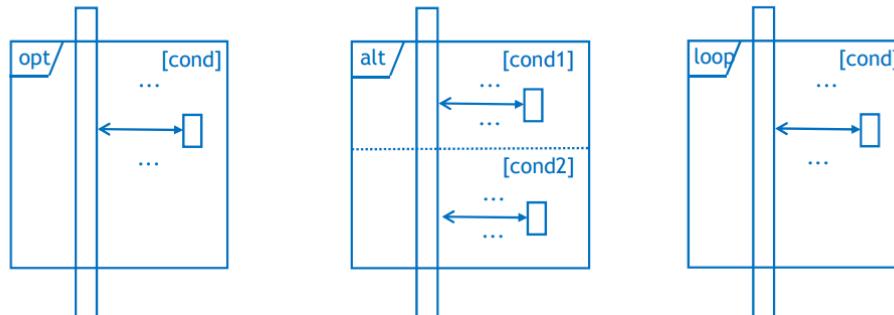
Informació derivada:

- 1- L'import d'una venda és igual al cost total dels productes de que consta

venda és un objecte del sistema que correspon a la venda que s'està realitzant
El valor d'aquest paràmetre és el mateix a totes les invocacions

Elements d'un DSS: frames

- Els *frames* permeten estructurar informació en els diagrames UML
- Tres tipus principals de *frames*:
 - execució opcional (*opt*)
 - execució alternativa (*alt*)
 - execució repetida (*/loop*)
- Els *frames* es poden aniar lliurement

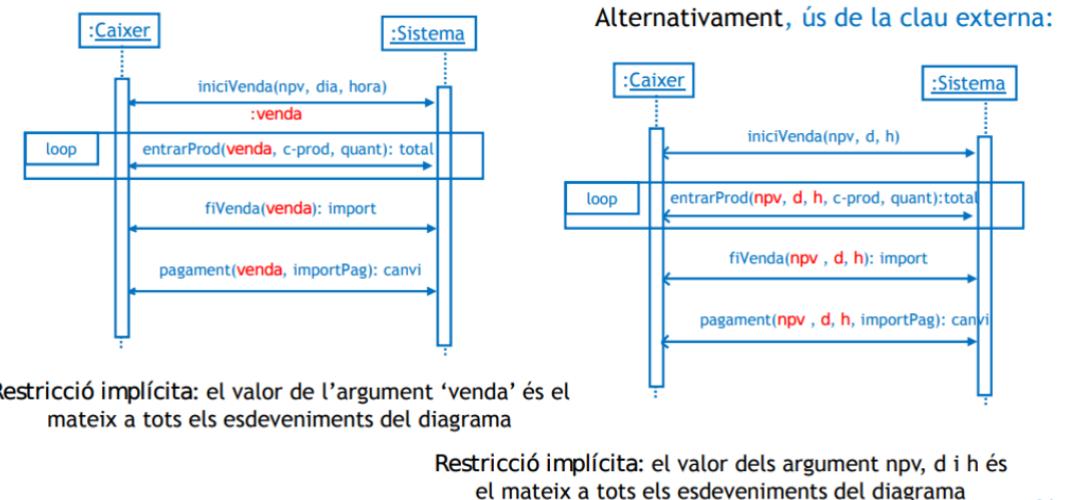


9

Compartició d'informació entre operacions d'un DSS

UML no precisa de quina manera les operacions d'un diagrama de seqüència del sistema poden compartir informació

Nosaltres ho especificarem mitjançant arguments addicionals de les operacions



10

12

Elements dels contractes de les operacions

Operació: nom i paràmetres de l'operació (Signatura de l'operació)

Precondicions:

Condicions que cal satisfer per poder executar l'operació.

Postcondicions:

Canvis d'estat que es produeixen com a conseqüència de l'execució:

- altes/baixes d'instàncies de classes d'objectes i d'associacions
- modificació d'atributs
- generalització o especialització d'un objecte
- canvi de subclasse d'un objecte

Sortida:

Descripció de la sortida que proporciona l'operació

Observeu el lligam que existeix entre els contractes de les operacions i l'esquema conceptual de les dades

Exemple: operació *iniciVenda*

Operació: iniciVenda(npv: Integer, dia:Data, hora:Hora): Venda

Precondicions:

Existeix un PuntDeVenda amb num-pv=npv

Postcondicions:

S'ha creat una instància V de Venda amb dia i hora

V s'ha associat amb la instància de PuntDeVenda amb num-pv=npv

Sortida: V

En OCL:

context: Sistema::iniciVenda(npv:Integer, dia:Data, hora:Hora): Venda

pre: PuntDeVenda.allInstances () ->exists (p|p.num-pv = npv)

post:

Venda.allInstances () ->exists (v|v.ocliIsNew() and v.dia=dia
and v.hora=hora and v.puntDeVenda.num-pv=npv and result =v)

Exemple: operació *entrarProd*

Operació: entrarProd(venda: Venda, c-prod: Integer, quant: Integer): Import

Precondicions:

Existeix un Producte amb codi = c-prod

Postcondicions:

S'ha creat una instància de *LíniaDeVenda L* amb quantitat=quant
venda s'ha associat amb L

L s'ha associat amb el *Producte* amb codi = c-prod

Sortida:

Es retorna l'import acumulat de la venda

En OCL:

context: Sistema::entrarProd(venda:Venda, codi:Integer, quant:Integer)

pre: Producte.allInstances () ->exists (p|p.codi = codi)

post:

LíniaDeVenda.allInstances () ->exists (l|l.ocliIsNew() and
l.quantitat = quant and l.venda = venda
and l.producte.codi = codi and result=venda.import)

Exemple: operació *fiVenda*

Operació: fiVenda(venda: Venda) :Import

Precondicions:

Postcondicions:

Sortida: es mostra l'import de la venda

En OCL

context: Sistema::fiVenda (venda:Venda) :Import

body: venda.import

Exemple: operació *pagament*

Name: pagament(venda: Venda, importPag: Import): Import

Precondicions:

importPag ≥ venda.import

Postcondicions:

Sortida: es mostra el canvi a retornar = importPag - venda.import

En OCL

context: Sistema::pagament(venda:Venda, importPag:Import) : Import

pre: importPag ≥ venda.import

body: importPag-venda.import

Redundància - Definició

Una especificació és redundant si un mateix aspecte del sistema està definit/descrit diverses vegades.

La redundància dificulta la modificabilitat de l'especificació perquè si varia aquell aspecte cal modificar tots els models que hi fan referència.

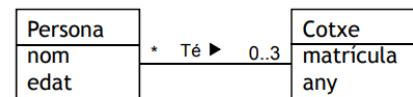
L'especificació no ha de ser redundant !!!

Redundància: Pre amb l'Esquema Conceptual de les Dades

L'esquema conceptual de les dades ja garanteix que les condicions imposades per les restriccions d'integritat (*estructurals, gràfiques i textuals*) mai es violaran



La precondició d'una operació no ha d'incloure les comprovacions necessàries per garantir que l'execució del cas d'ús no viola les restriccions d'integritat definides a l'esquema conceptual de les dades



Restriccions textuals:

1. Claus externes: (Persona,nom); (Cotxe,matrícula)
2. Les persones menors de 18 anys no poden tenir cotxe

context: Sistema::comprarCotxe (nom, mat)

pre:
Persona.allInstances()->exists(p|p.nom==nom
and p.edat>18)

Cotxe.allInstances()->exists(c|c.matrícula==mat)

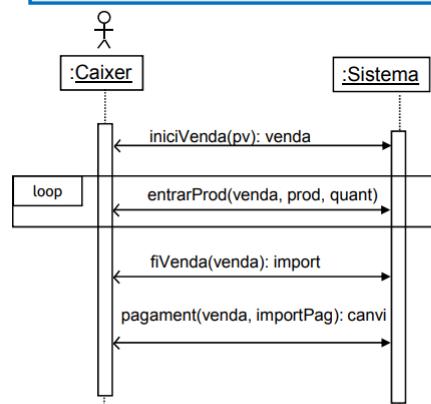
post:
Persona.allInstances()->select(p|p.nom==nom).
cotxe.matrícula->includes(mat)

Redundància: Pre amb el Diagrama de Seqüència

Els diagrames de seqüència ja defineixen un ordre d'invocació de les operacions



Les operacions no han d'incorporar informació per garantir que aquest ordre es compleixi



Operació: Pagament (venda, importPag): canvi

Precondicions:
- existeix venda "venda"
- la venda "venda" ha finalitzat

Postcondicions:

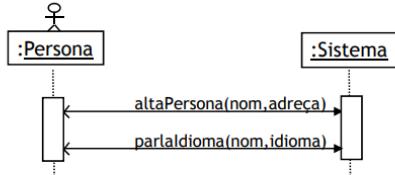
...

Sortida:

Exemple de diagrama de seqüència incorrecte



Cas d'ús: Afegir-Persona-1



Operació: altaPersona(nom,adreça)

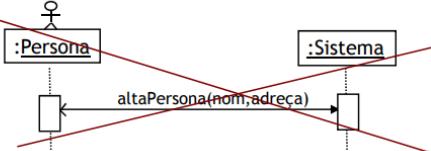
Precondicions:

Postcondicions:

- s'ha creat d'un objecte Persona amb el nom i l'adreça especificats

Sortida:

Cas d'ús: Afegir-Persona-2



Operació: altaPersona(nom,adreça)

Precondicions:

- l'idioma identificat per nomId existeix

Postcondicions:

- creació d'una associació Parla entre la persona amb nom=nom i l'idioma nom=nomId

Sortida:

Especificació en OCL de la sortida d'una operació

La sortida d'una operació acostuma a ser informació composta. O sigui, no elemental.

resumVendes(num-pv: Integer) retorna un llistat de vendes amb la informació:

Per cada Producte venut al PuntDeVenda num-pv mostrar el seu codi i el seu preu

resumVendes (num-pv: Integer) : ???

Definició d'informació composta en OCL:

Tuples: composició de diversos elements, possiblement de tipus diferents

Definició del tipus: TupleType(camp1:Tipus1, ..., campN:TipusN)

Assignació de valors: Tuple{camp1=valor1, ..., campN=valorN}

Col·leccions: Set, Sequence i Bag.

resumVendes (num-pv: Integer) :
Set(TupleType(codiprod:Integer, preu:Import))

Exemple d'operació de consulta en OCL

context: Sistema::resumVendes (npv:Integer) :
Set(TupleType(cprod:Integer, preu: Import))

pre: PuntDeVenda.allInstances () ->exists (p|p.num-pv=npv)

body:

```

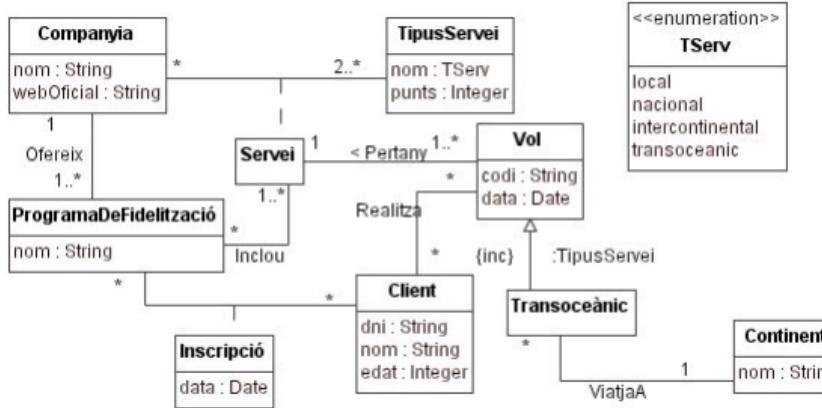
let prods: Set(Producto) =
  Producto.allInstances () ->
  select (p|p.liniaDeVenta.venda.puntDeVenda.num-pv->includes (npv))
in
prods->collect (p | Tuple{cprod = p.codi, preu = p.preu})
  
```

context Sistema::resumenVentas (num-pv: Integer) :
Set(TupleType(cod: Integer, npv: Import))

pre : PuntoDeVenta.allInstances () ->
exists (pdv | pdv.npv = num-pv)

body : let prods : Set(Producto) =
PuntoDeVenta.allInstances ()
→select (pdv | pdv.npv = num-pv)
·venta
Set(v)
Set(ldv)
result = prods →collect (p | Tuple{
cod = p.codigo,
npv = p.precio})

Considereu un sistema per un consorci de companyies aèries. Les companyies associades disposen de diversos tipus de servei (local, nacional, intercontinental i transoceànic) i ofereixen programes de fidelització als seus clients per a què s'hi inscriguin. S'enregistren també els vols que pertanyen a un servei i els clients que realitzen els vols. La part rellevant de l'esquema conceptual de les dades en UML es mostra a la figura següent:



Resticcions textuais

1. Claus externes: (Companyia, nom), (TipusServei, nom), (Continent, nom), (Vol, codi+data), (Client, dni)
2. Una companyia no pot oferir dos o més programes de fidelització amb el mateix nom
3. Els serveis inclosos a un programa de fidelització són de la companyia que ofereix el programa
4. Un client no es pot inscriure a més d'un programa de fidelització de la mateixa companyia

Suposarem que el sistema a desenvolupar no ha de donar d'alta instàncies de les classes Companyia, TipusServei, ProgramaDeFidelitzacio, Continent ni Client atès que hi ha un altre sistema que ho fa. El sistema ha de permetre efectuar les funcionalitats següents: alta de servei i vols, alta d'inscripció i consulta de clients sènior actius.

AltaServeiVols: Quan un directiu vol donar d'alta un servei i els seus vols li demana a un empleat del consorci que ho faci. L'empleat indica al sistema la informació necessària per donar d'alta un servei. A continuació, per cada vol que ha de pertànyer al servei, l'empleat dóna la informació necessària per donar d'alta el vol. Si es tracta d'un vol transoceànic, indica també el continent on viatja. Cal considerar que aquesta funcionalitat només es pot realitzar si la companyia del servei té menys de 20 programes de fidelització. Feu que la interacció per portar a terme aquesta funcionalitat requereixi més d'un esdeveniment.

AltaInscripció: Quan un client es vol inscriure a un programa de fidelització, ell mateix indica al sistema la informació necessària per identificar el programa de fidelització, el seu dni i la data de la inscripció. Només es pot fer aquesta funcionalitat si el client té com a mínim un vol que pertanyi a un servei inclòs al programa de fidelització on es vol inscriure i si la data de la inscripció és posterior o igual a l'actual. Feu que la interacció requereixi un sol esdeveniment.

ConsultaClientsSèniorActius: Quan un empleat vol fer una consulta de clients sènior actius, ell mateix introduceix al sistema un nom de companyia. El sistema mostra el llistat dels clients que tenen més de 40 anys i que han fet més de 3 vols transoceànics que pertanyen a serveis de la companyia donada. El llistat resultant mostra per cada client el dni i les dates de tots els vols que té enregistrats al sistema. Aquesta funcionalitat només s'executa si la companyia existeix.

Es demana que feu mitjançant la notació UML:

- Diagrames de seqüència i contractes en OCL de les operacions corresponents a les funcionalitats anteriors.

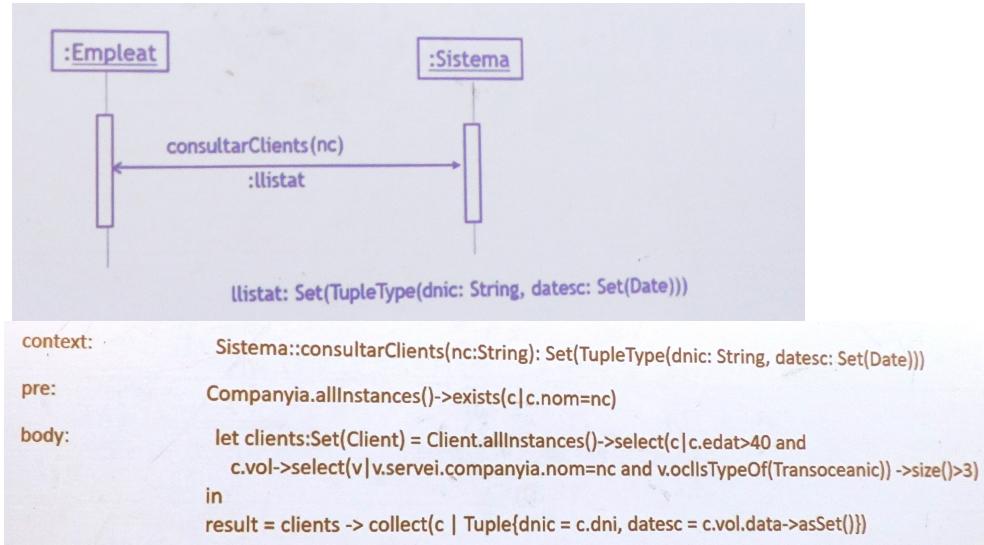


context: Sistema::altaServei (nc:String, nt:TServ): s: Servei
pre: Companyia.allInstances()->exists(c|c.nom=nc and c.programaDeFidelització->size() <20) and TipusServei.allInstances()->exists(t|t.nom=nt)
post: Servei.allInstances()->exists(s| s.ocllsNew() and s.companyia.nom=nc and s.tipusServei.nom=nt and result=s)

context: Sistema:: altaVol(s:Servei, cv:String, dv:Date, co:String)
pre: s.tipusServei.nom=TServ::transoceànic implies Continent.allInstances()->exists(c|c.nom=co)
post: Vol.allInstances()->exists(v| v.ocllsNew() and v.codi=cv and v.data= dv and v.servei=s and if s.tipusServei.nom=TServ::transoceanic then v.ocllsTypeOf(Transoceànic) and v.ocllsAsType(Transoceànic).continent.nom=co endif)



context: Sistema::altaInscripcio(nc:String, npf:String, dc:String, di:Date)
pre: di>=today() and Client.allInstances()-> exists(cl|cl.dni=dc and cl.vol.servei.programaDeFidelització-> exists(pf|pf.nom=npf and pf.companyia.nom=nc))
post: Inscripcio.allInstances()->exists(i| i.ocllsNew() and i.data=di and i.programaDeFidelització.nom=npf and i.programaDeFidelització.companyia.nom=nc and i.client.dni=dc)



Resticcions textuals

1. Claus externes: (Animal, nom), (Persona, nom), (Veterinari, numCol), (Data, data), (Espècie, nom), (Medicament, nom)
2. Per cada operació finalitzada, horaFi > horalini
3. Un Veterinari no pot fer dues operacions que se solapin temporalment.

El sistema a desenvolupar no ha de donar d'alta Veterinari, Espècie, Medicament ni Data ja que hi ha un altre sistema encarregat de fer-ho. En canvi, ha de proporcionar les funcionalitats següents:

Alta Animal: Quan l'auxiliar de clínica vol donar d'alta un nou animal a la clínica, ell mateix introduceix la informació necessària per fer-ho. És a dir, tota la informació requerida per a qualsevol animal. Si el propietari de l'animal no està registrat en el sistema, s'haurà de crear en aquell moment (però no com a veterinari).

Alta Operació Urgent: En els cas de les operacions urgents, aquestes són donades d'alta un cop ja han finalitzat. Per fer-ho, el veterinari indica al sistema totes les dades necessàries per crear una operació finalitzada. En aquest cas, podeu assumir que l'animal ja està donat d'alta al sistema. A més, aquesta funcionalitat només pot realitzar-se si el veterinari que realitza l'operació és especialista en l'espècie de l'animal que opera. Feu que la interacció necessària per dur a terme aquesta funcionalitat requereixi més d'un esdeveniment.

Consulta Gossos Perillósos no Esterilitzats: Podria ser el cas que l'auxiliar de clínica volgués consultar quins gossos agressius hi ha donats d'alta que no han estan sotmesos ni tenen programada una operació amb motiu "esterilització", per tal de recomanar als seus propietaris que la facin. Per fer-ho, l'auxiliar indica el nom de la població al sistema. El sistema retorna una llista amb el nom del gos, el nom del propietari i un boolé indicant si aquest últim també és veterinari, per tots aquells gossos agressius tals que: el propietari del gos viu a la població indicada per paràmetre, el gos no està esterilitzat (ni aquesta està programada), i el gos és mascle (en les femelles l'esterilització no millora l'agressivitat). Per poder dur a terme aquesta funcionalitat és necessari que hi hagi una persona com a mínim que visqui a la població indicada i que sigui propietària d'almenys un animal.



Context:

Sistema::AltaAnimal(nom_especie: String, nom_animal: String, sexe: TSexe, agressiu: boolean, nom_propietari: String, poblacio: String)

Pre:

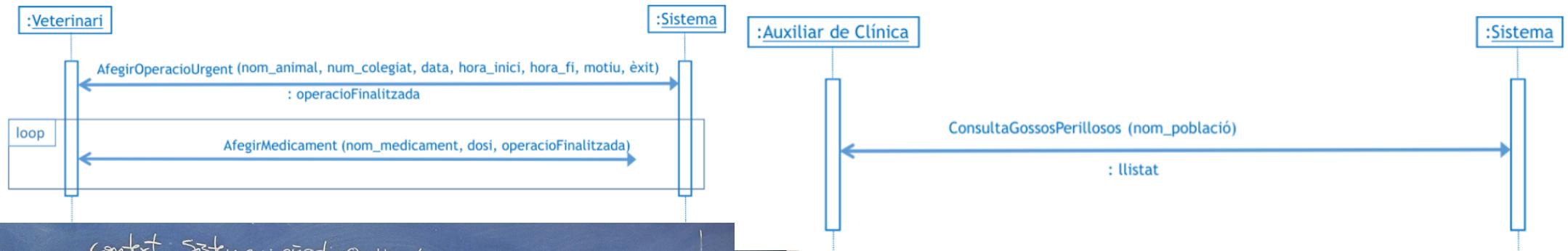
Especie.allInstances()->exists(e | e.nom = nom_especie)

Post:

```

If not (Persona.allInstances() @pre->Exists(p | p.nom = nom_persona) then
    Persona.allInstances()->Exists(p | p.nom=nom_persona
        and p.poblacio=nom_poblacio
        and p.ocliIsNew())
Endif
And
Animal.allInstances()->Exists(a | a.ocliIsNew()
    and a.nom=nom_animal
    and a.propietari.nom = nom_propietari
    and a.sexe = sexe
    and a.especie.nom = nom_especie
    and if (nom_especie='Gos') then
        a.ocliIsTypeOf(Gos)
        and a.oclaAsType(Gos).agressiu = agressiu
    endif )

```



context Sistema:: añadirOpUrg (—) : OperaciónFinalizada

pre : Veterinario.allInstances() → exists(v | v.numCol = num_colegiado
and v.especie.nombre → includes(

and
data ≤ today())
Animal.allInstances() → select(a |
a.nombre = nom_animal).
especie.nombre)) +

and
Animal.allInstances() → exists(a | a.nombre = nom_animal)

post : OperaciónFinalizada.allInstances() → exists(of |
of.ocellsNew() and of.exitosa = exit and
of.horaFin = hora.fi and of.horaInici = hora.inici and
of.motiu = motiu and
of.veterinari.numCol = num_colegiado and
of.fecha.fecha = data and
of.animal.nombre = nom_animal and
result = of)

pre : Medicamento.allInstances() → exists(m | m.nombre = nom_med)

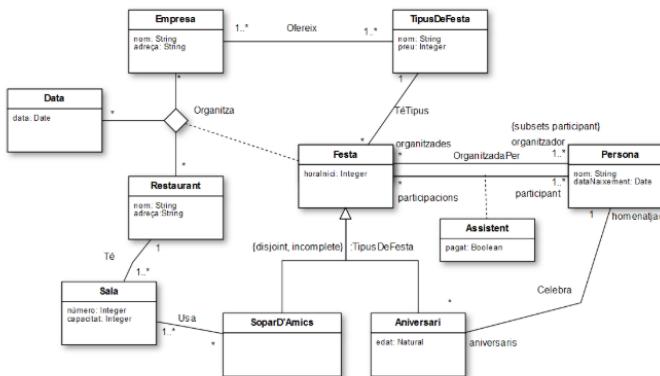
post : MedicamentoAdministrado.allInstances() → exists(ma |
ma.ocellsNew() and
ma.dosis = dosis and
ma.operacionFinalizada = of and
ma.medicamento.nombre = nom_med)

pre : Persona.allInstances() → exists(p |
p.población = nom_pob and
p.animal → notEmpty()
→ size() > 0

body : let permisChungos: Set(Perm) =

Permis.allInstances() → select(p |
p.agresivo and
p.sexo = TSexo::macho and
p.persona.población = nom_pob and
p.operación.motiu → excludes
Set(o) Set(st) ('esterilización'))

permisChungos → collect(pc | Tuple {
nom_p = pc.nombre,
nom_pProp = pc.persona.nombre,
es_vet = pc.propietario.adListyOF(Vet)}



El sistema a desenvolupar no ha de donar d'alta les dades d'**Empresa**, de **TipusDeFesta**, de **Persona** ni de **Data** ja que existeix un altre sistema que les gestiona.

1. Claus externes: (Empresa, nom); (TipusDeFesta, nom); (Data, data); (Restaurant, nom); (Persona, nom)

2. Un restaurant no pot tenir dues sales amb el mateix número

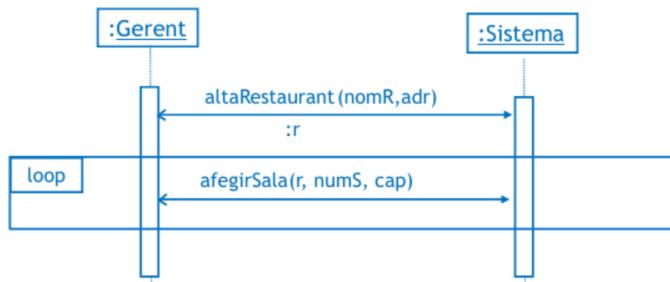
3. El tipus de festa d'una festa ha de ser ofert per l'empresa que organitza la festa

4. Una persona no pot assistir a dues o més festes que se celebren el mateix dia i amb la mateixa hora d'inici

5. La capacitat total de les sales usades per una festa SoparD'Amics ha de ser superior al número de participants a la festa

6. Les sales usades per una festa SoparD'Amics han de ser del restaurant on se celebra la festa

Alta de Restaurant: Quan el gerent del consorci vol donar d'alta un restaurant, introduceix al sistema tota la informació necessària per fer-ho. És a dir, la informació del restaurant i de totes les seves sales. Cal tenir en compte que caldrà crear les sales del restaurant sempre que sigui necessari. Aquesta funcionalitat només es podrà dur a terme si el consorci té menys de 100 restaurants i tots els restaurants tenen com a màxim 5 sales. Feu que la interacció necessària per dur a terme aquesta funcionalitat requereixi com a mínim dos esdeveniments.



Context Sistema::altaRestaurant(nomR: String, adr: String): Restaurant

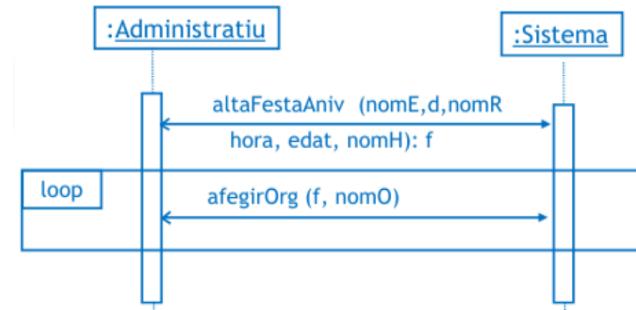
Pre: Restaurant.allInstances()>size()<100 and
Restaurant.allInstances()>forAll (r | r.sala->size()<=5)

Post: Restaurant.allInstances()>exists(r |
r.ocellsNew() and
r.nom=nomR and r.adreça=adr and
result=r)

Context Sistema::afegirSala(r: Restaurant, numS: Int, cap: Int)

Pre:
Post: Sala.allInstances()>exists(s |
s.ocellsNew() and
s.número=numS and s.capacitat=cap and
s.restaurant=r)

Alta Festa d'Aniversari: Quan un administratiu vol donar d'alta una festa d'aniversari, introduceix al sistema tota la informació necessària per fer-ho. És a dir, la informació de la festa d'aniversari, la dels seus organitzadors i la de la persona que hi celebra l'aniversari. Per a simplificar, suposem que quan es crea la festa els únics assistents seran els seus organitzadors i la persona que celebra l'aniversari, i que tots hauran pagat. Cal tenir en compte que aquesta funcionalitat només es pot dur a terme si els organitzadors ni la persona homenatjada no són assistents d'una altra festa que se celebra el mateix dia a la mateixa hora.



Context Sistema::altaFestaAniv(nomE: String, d: Date, nomR: String, hora: int, edat: Natural, nomH: String) : Aniversari

Pre: Empresa.allInstances()>exists (e | e.nom=nomE) and
Restaurant.allInstances()>exists (r | e.nom=nomR) and
TipusDeFesta.allInstances()>exists (tf | tf.nom="FestaAniversari") and
Persona.allInstances()>exists (p | p.nom=nomH)

Post: Aniversari.allInstances()>exists(f | f.ocellsNew() and
f.empresa.nom=nomE and f.data.data=d and f.restaurant.nom=nomR and
f.horaInici=hora and f.edat=edat and
f.tipusDeFesta.nom="FestaAniversari" and
f.homenatjada.nom=nomH and
Assistent.allInstances-> exists(a |
a.ocellsNew() and
a.participacions=f and a.participant.nom=nomH and
a.pagat) and
result=f)

context Sistema:: afegirOrg(f:Aniversari, nomO:String)

Pre: Persona.allInstances()>exists (p | p.nom=nomO)
Post: f.organitzador.nom->includes(nomO) and
Assistent.allInstances()-> exists(a |
a.ocellsNew() and
a.participacions=f and
a.participant.nom=nomO and
a.pagat)

Consulta Aniversaris Estranyos: Quan el gerent d'una empresa vol obtenir el llistat dels aniversaris estranyos, ell mateix introduceix al sistema el nom de l'empresa. El llistat mostra, per cada festa d'aniversari que s'ha fet a l'empresa, tal que la festa té més de 30 assistents però on la persona homenatjada no hi participa, la informació següent: nom de la persona homenatjada i el nom dels organitzadors de la festa. Aquesta funcionalitat només es pot portar a terme si l'empresa ha organitzat, com a mínim, una festa de cada tipus de festa que ofereix.



Context Sistema:: llistaAnivsEstranyos (nomEmp:String): Set(TupleType(nomH:String, nomsOrgs: Set(String))

Pre: Empresa.allInstances()->exists(e | e.nom=nomEmp and
e.festa.tipusDeFesta->includesAll(e.tipusDeFesta))

Body: let aniversaris:Set(Aniversari) = Aniversari.allInstances()->select(a |
a.empresa.nom=nomEmp and
a.assistent-> size()>30 and
a.participant -> excludes(a.homenatjada))
in
aniversaris->collect(a | Tuple{nomH=a.homenatjada.nom;
nomsOrgs=a.organitzador.nom})