Especificació en UML: Esquema del comportament

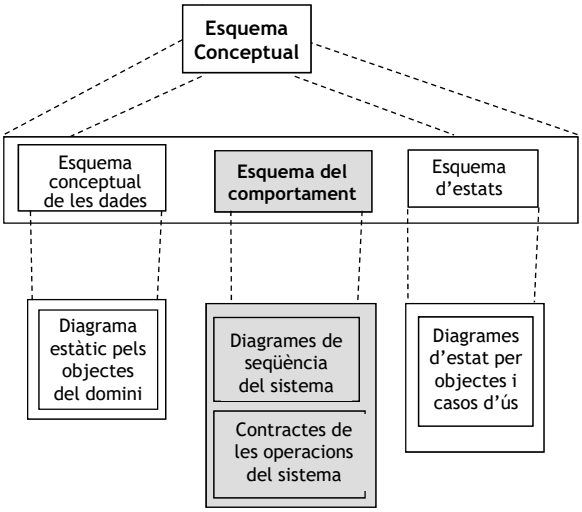
FIB

Departament d'Enginyeria de Serveis i Sistemes d'Informació
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Esquema del comportament en UML

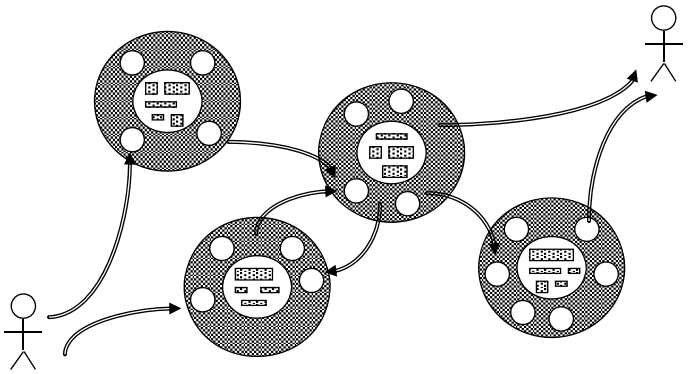
Especificació del comportament 'orientat a objectes'
Diagrames de seqüència del sistema
Contractes de les operacions del sistema
Redundància amb l'esquema conceptual de les dades
Interacció amb actors software (serveis) externs
Especificació en OCL de la sortida d'una operació
Diagrama d'estats
Bibliografia

Esquema conceptual (especificació)



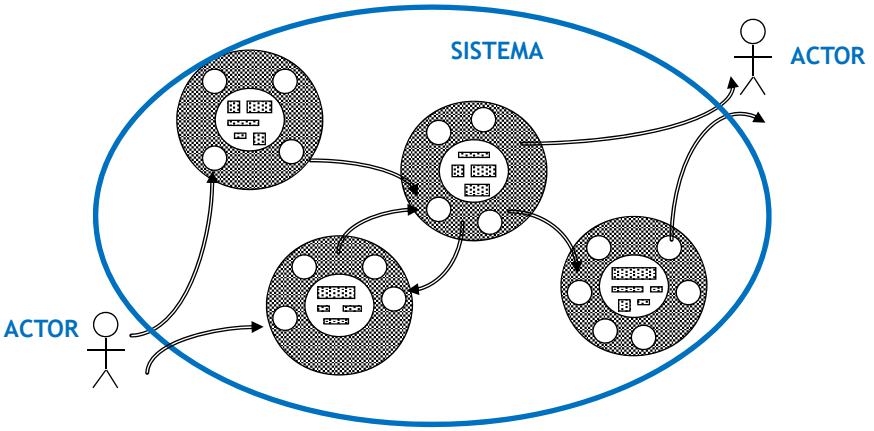
Descripció del comportament en OO
(visió dinàmica d'un sistema OO)

Els objectes es comuniquen (interactuen) mitjançant la invocació d'operacions d'altres objectes



"Especificació" del comportament en OO

- Considerem un tipus especial "sistema" que engloba tots els objectes i describim la interacció dels actors externs amb el sistema
- L'esquema del comportament del "sistema" ens permet especificar la dinàmica dels objectes descrits al *diagrama de classes* i dels *escenaris dels casos d'ús*.



Artefactes usats a l'esquema del comportament del sistema

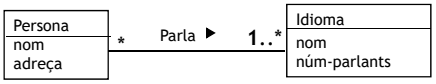
Diagrames de seqüència del sistema (DSS): Mostra les interaccions entre els actors i el sistema.

- 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 produeixen 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:



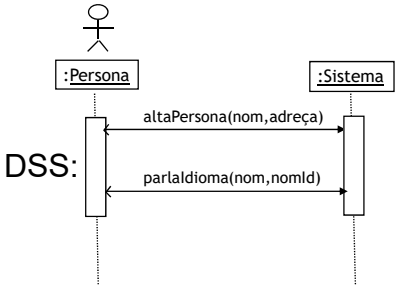
Cas d'ús: Afegir-Persona
1. La persona indica al sistema el seu nom i la seva adreça
2. El Sistema enregistra el nom i l'adreça de la persona
3. La persona indica al sistema l'idioma que parla
4. El Sistema enregistra l'idioma parlat per la persona

Contractes:

Operació: altaPersona(nom,adreça)
Precondicions:
Postcondicions:
- **s'ha creat d'un objecte Persona amb el nom i l'adreça especificats**

Sortida:

DSS:



Operació: parlaIdioma(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:

L'origen de molts mals

"El codi fa el que li demano, però no el que jo vull"

Raons habituals

- La majoria dels bugs no són d'escriptura de codi, sinó de "comprensió" dels requisits.
- El llenguatge natural és inherentment ambigu ("un usuari vàlid", "saldo suficient", "processar la comanda").
- Aquesta ambigüitat es tradueix directament en codi amb errors inesperats.

El Repte

Com podem definir el "voler" de forma precisa i inequívoca?

El nostre escenari: El compte bancari

Operació a especificar: `retirarDiners(quantitat: Real): Boolean`

L'especificació informal (i la seva fàbrica de bugs 🐛)

Requisit: "L'operació ha de permetre retirar diners del compte, actualitzant-ne el saldo."

Codi: Un programador (o IA) podria escriure aquest codi, per complir el requisit:

```
public void retirarDiners(double quantitat) {
    this.saldo = this.saldo - quantitat;
}
```

Però... Què passa si...?

- quantitat és -50€? saldo augmenta! (Forat de seguretat)
- No hi ha prou saldo? saldo queda en negatiu! (Política de descobert no desitjada)

El problema no és el programador, és l'especificació.
És massa ambigua per poder escriure codi robust.

Afegint estructura (pre/post)

Operació: `retirarDiners(quantitat: Real): Boolean`

Precondicions (Llenguatge Natural):

1. La quantitat ha de ser positiva.
2. El saldo ha de ser més gran o igual que la quantitat.

Postcondicions (Llenguatge Natural):

1. El saldo es redueix en la quantitat a retirada.

Millora:

- Molt millor, però... "positiva" és > 0 o >= 0?
- En sistemes complexos, el llenguatge natural continua tenint escletxes.

Contractes formals amb OCL

Per eliminar TOTA l'ambigüitat, fem servir un llenguatge formal, part de l'estàndard UML.

OCL (Object Constraint Language) ens permet escriure regles precises sobre els nostres models.

En OCL:

```
context Sistema::retirarDiners(compte: Compte,
                               quantitat: Real):Boolean
pre:
quantitat > 0 and compte.saldo >= quantitat
post:
compte.saldo = compte.saldo@pre - quantitat
```

I tot aquest esforç per a què?

Un contracte OCL no és un exercici acadèmic, és una eina d'enginyeria amb beneficis directes:

Claredat absoluta per al desenvolupador

- El programador que implementa retirarDiners té una guia inequívoca. No hi ha espai per a la interpretació. Sap exactament quines condicions comprovar i quin estat final ha de garantir.

Base sòlida per a la verificació (proves)

- El contracte ens diu exactament què s'ha de provar. Cada clàusula és una invitació a escriure una prova (test).
- Hi ha eines que permeten automatitzar la generació del codi de proves i del codi de producció.

Del contracte a la verificació

Observeu com cada línia del contracte ens "crida" un cas de prova que hem de crear.

Clàusula OCL	Casos de prova a implementar
pre: `quantitat > 0`	`test_retirar_quantitat_negativa_falla()` `test_retirar_quantitat_zero_falla()`
pre: `compte.saldo >= quantitat`	`test_retirar_mes_del_saldo_falla()`
post: `compte.saldo = compte.saldo@pre - quantitat`	`test_retirar_exitós_actualitza_saldo_correctament()` `test_retirar_saldo_exacte_es_exitós()`

Enfocament en l'especificació

- **Especificar el comportament és fonamental.** No és un extra, és el nucli del disseny de software robust i correcte.
- **L'OCL proporciona la precisió matemàtica** que el llenguatge natural no pot oferir, eliminant l'ambigüitat.
- **Els contractes** (pre/post) defineixen responsabilitats clares entre les diferents parts del software.
- **Com a gran benefici,** un bon contracte és la guia perfecta per a la verificació i validació, fent que pràctiques com el **TDD** siguin increïblement efectives i sistemàtiques.

OCL i TDD: Una aliança perfecta

TDD: Test Driven Development o desenvolupament dirigit per proves.

Tot i que el TDD no és el tema central, és l'aplicació més directa del valor dels contractes OCL.

El contracte OCL ens guia en el cicle "**Vermell**-**Verd**-**Refactor**".

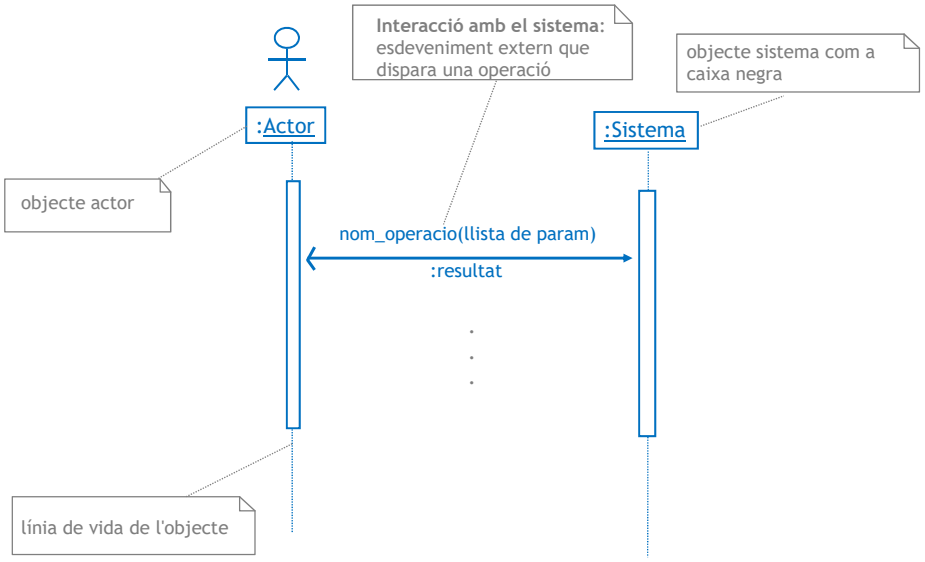
1. **VERMELL:** Tria una clàusula del contracte (`pre: quantitat > 0`). Escriu un test que la violi (`test_retirar_quantitat_negativa()`). **Verifica que falla.**
2. **VERD:** Escriu el codi **mínim** perquè aquest test passi (`if (quantitat <= 0) return false;`).
3. **REFACTOR:** El codi és net? Es pot millorar?
4. **REPETEIX:** Agafa la següent clàusula (`pre: saldo >= quantitat`) i crea el següent test.

El contracte OCL guia el procés de desenvolupament de forma sistemàtica.

Avui anem a veure:

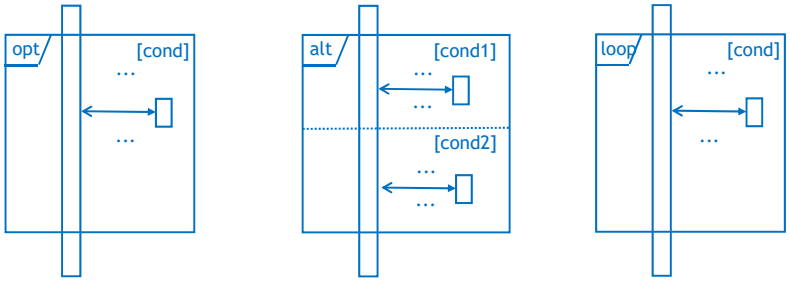
- Elements **dels** Diagrames de seqüència del sistema
- Elements dels Contractes de les operacions del sistema
- Redundància amb l'esquema conceptual de les dades

Elements bàsics d'un DSS:

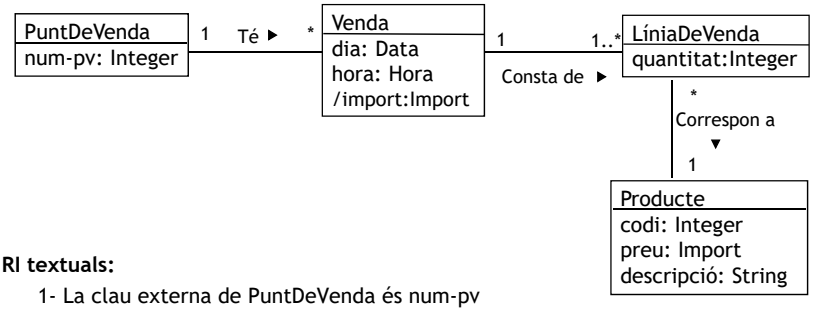


Elements d'un DSS: frames

- 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 aniuar lliurement



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



RI textuals:

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 unívocament 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

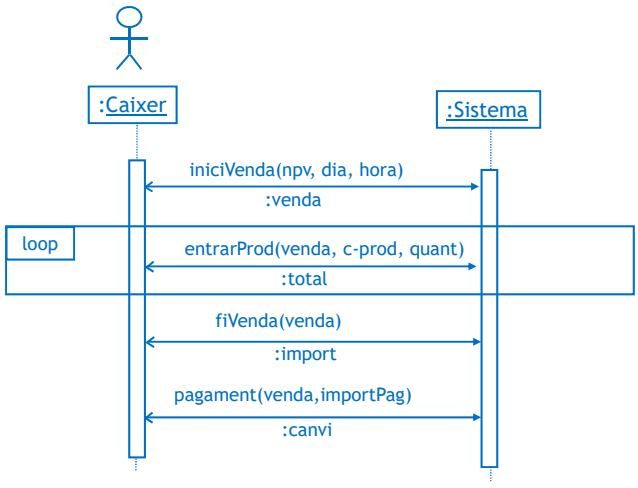
Exemple: cas d'ús Comprar Productes

escenari principal d'èxit

1. El Caixer indica al Sistema que comença una nova venda, amb el dia i l'hora actuals, en una caixa que té un número determinat
2. El Sistema enregistra el començament de la venda a la caixa
3. El Caixer entra al Sistema l'identificador de producte i la quantitat d'unitats
4. El Sistema enregistra la línia de venda corresponent al producte
5. El Sistema mostra el total acumulat

   Es repeteixen els passos 3 a 5 fins que s'han processat tots els productes
6. El Caixer indica al Sistema l'acabament de la venda
7. El Sistema presenta el total de la venda
8. El Caixer comunica al Client el total de la venda
9. El Client entrega al Caixer una quantitat de diners, potser més gran que el total de la venda
10. El Caixer indica al Sistema els diners que ha rebut
11. El Sistema mostra el canvi i imprimeix el rebut
12. El Caixer diposita els diners rebuts a la caixa i n'extreu el canvi
13. El Caixer dóna el canvi al Client
14. El Client se'n va amb els productes comprats

Exemple: Comprar Productes

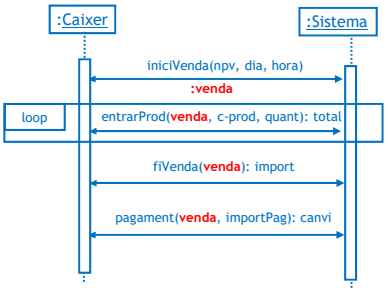


venda és un objecte del sistema que correspon a la venda que s'està realitzant
El valor d'aquest paràmetre és el mateix a totes les invocacions

Compartició d'informació entre operacions d'un DSS

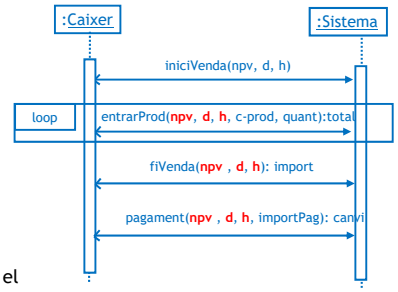
UML no precisa de quina manera les operacions d'un diagrama de seqüència del sistema poden compartir informació

Nosaltres ho especificarem **mitjançant arguments addicionals de les operacions**



Restricció implícita: el valor de l'argument 'venda' és el mateix a tots els esdeveniments del diagrama

Alternativament, ús de la clau externa:



Restricció implícita: el valor dels argument npv, d i h és el mateix a tots els esdeveniments del diagrama

Recordem:

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

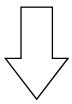
- Punt de partida: La *seqüència d'esdeveniments en un **escenari d'un cas d'us***
- Idea bàsica: L'actor genera esdeveniments cap al sistema, que exigeixen l'execució d'alguna operació com a resposta.
- Objectiu/utilitat: Permeten identificar les operacions del sistema i especificar la dinàmica dels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

Contractes: Esdeveniments i operacions

- **Esdeveniment** extern generat per un actor



La comunicació d'un esdeveniment del sistema provoca l'execució d'una operació del sistema amb el **mateix nom** i els **mateixos paràmetres**

- **Operació del sistema:** Operació interna que s'executa com a resposta a la comunicació de l'esdeveniment

Operacions del sistema

NO s'assignen a objectes concrets durant l'etapa d'especificació

Les operacions del sistema **s'agrupen** com operacions de la classe "**Sistema**", la qual usem per a representar el sistema software.

Exemple:

Sistema
iniciVenda(npv: Integer, d:dia, h:hora): venda
entrarProd(venda:Venda, c-prod: Integer, quant: Integer): total
fiVenda(venda: Venda): Import
pagament(venda: Venda, importPag: Import): Import

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:

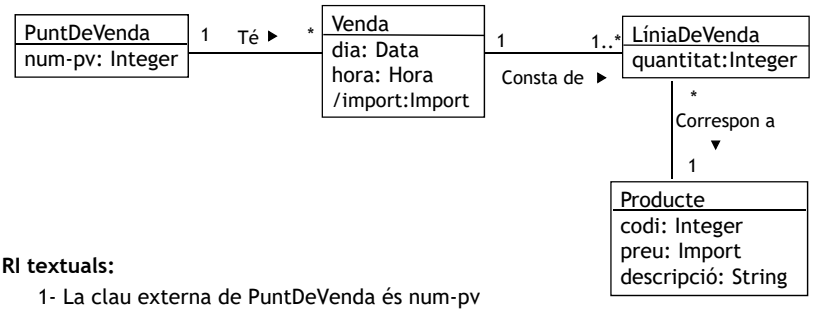
- altes/baixes d'instàncies de classes d'objectes i d'associacions
- modificació d'atributs
- generalització o especialització d'un objecte
- canvi de subclasse d'un objecte

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: esquema conceptual de partida



RI textuals:

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 unívocament 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

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.oclIsNew()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.oclIsNew() 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: `result = 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: `result = 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 !!!

Redundàncies possibles:

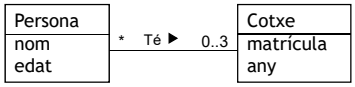
- Entre una precondició i l'esquema conceptual de les dades
- Entre una precondició i els diagrames de seqüència del sistema
- Entre una precondició i el diagrama d'estats (redundància que no veurem a IES)

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



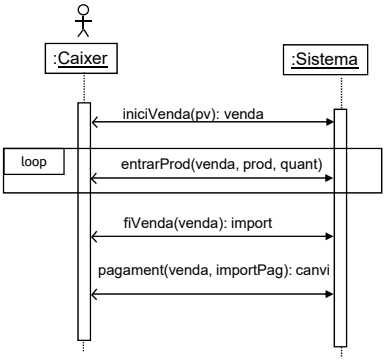
Restriccions textuals:
1. Claus externes: (Persona,nom); (Cotxe,matrícula)
2. Les persones menors de 18 anys no poden tenir cotxe

```
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

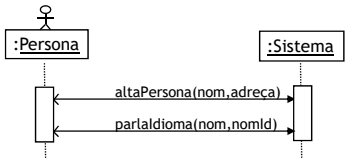


Operació: Pagament (venda, importPag): canvi
Precondicions:
- ~~existeix venda "venda"~~
- ~~la venda "venda" ha finalitzat~~

Postcondicions:
...
Sortida:

Exemple de diagrama de seqüència incorrecte

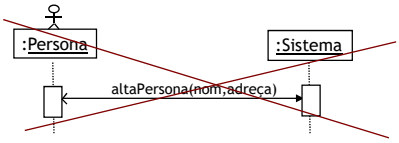
Cas d'ús: Afegir-Persona-1



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



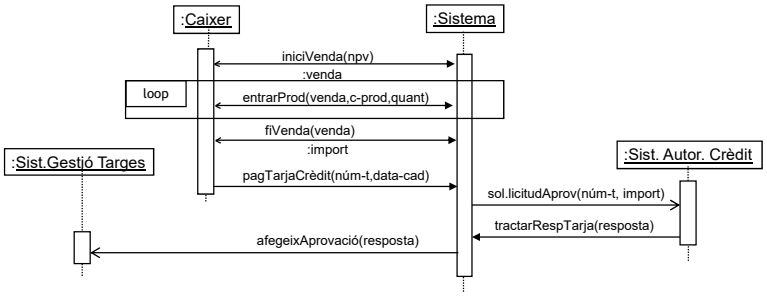
Operació: parlaldioma(nom,nomId)
Precondicions:
- l'idioma identificat per nomId existeix

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

Sortida:

Interacció amb actors software (serveis) externs

- De vegades un actor d'un cas d'ús pot ser un sistema software (servei) extern al sistema que estem desenvolupant.
- *Exemple: cas d'ús comprar productes amb tarja de crèdit*



La postcondició de l'operació *pagTarjaCrèdit* inclou la invocació del servei demanat al *Sistema d'Autorització de Crèdit*, mitjançant la frase:

'El sistema invoca l'execució de l'operació sol·licitudAprov al sistema d'autorització de crèdit i li passa núm-t i import com a paràmetres

Redundància - Interpretació dels contractes

Com s'han d'interpretar els contractes en relació a l'esquema conceptual?

Precondicions:

què ha de contenir l'Esquema Conceptual per **intentar executar una operació**

Restriccions d'integritat:

s'han de satisfer després de l'execució de totes les operacions d'un cas d'ús

Suposem que es rebutgen totes les operacions d'un cas d'ús si la seva execució viola (globalment) alguna restricció d'integritat de l'esquema conceptual de les dades

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))`

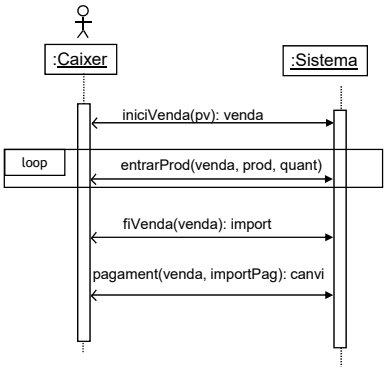
pre: `PuntDeVenda.allInstances()->exists(p|p.num-pv=npv)`

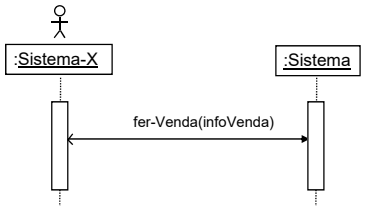
body:

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

Diagrama de seqüència: quants esdeveniments?

El nombre d'esdeveniments d'un diagrama de seqüència depèn de **com es produeix la interacció** entre els actors i el sistema software.



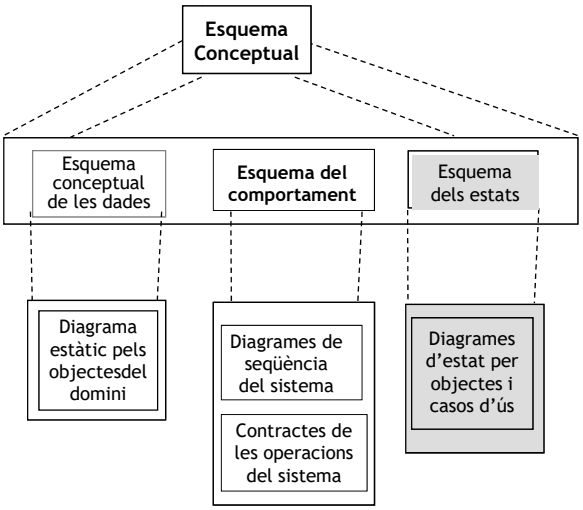


infoVenda: TupleType(num-pv:Integer,
ll-compra:Set(TupleType(codi-prod:Integer, quant:Integer)),
importPag:Import)

Ambdós diagrames suposen la **mateixa entrada d'informació** al sistema

Els dos diagrames de seqüència poden ser **correctes, segons les circumstàncies**

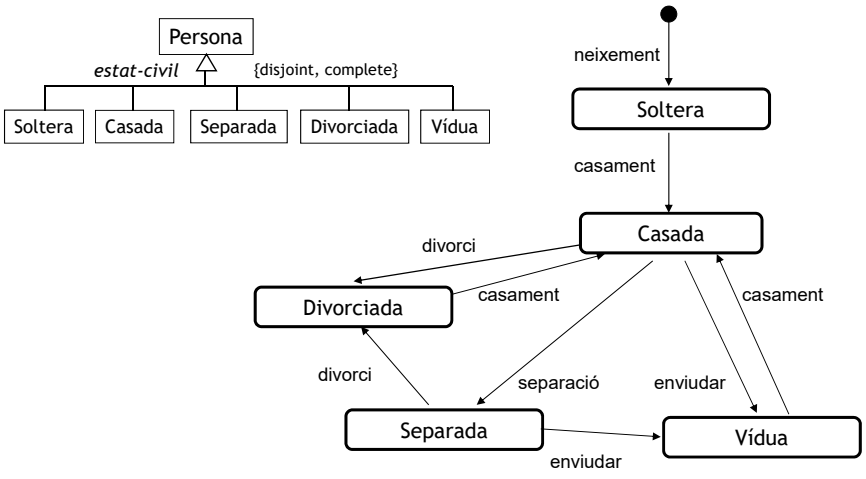
Esquema conceptual (especificació)



Esquema dels Estats

- **Objectius:**
  - crear diagrames d'estat per objectes i casos d'ús
- **Estats, transicions i esdeveniments:**
  - Estat:
    - condició d'un objecte o d'un cas d'ús en un moment del temps
  - Transició:
    - canvi d'estat com a conseqüència d'un esdeveniment
  - Esdeveniment:
    - tot allò que requereix una resposta del sistema software

Canvi d'estat civil d'una persona

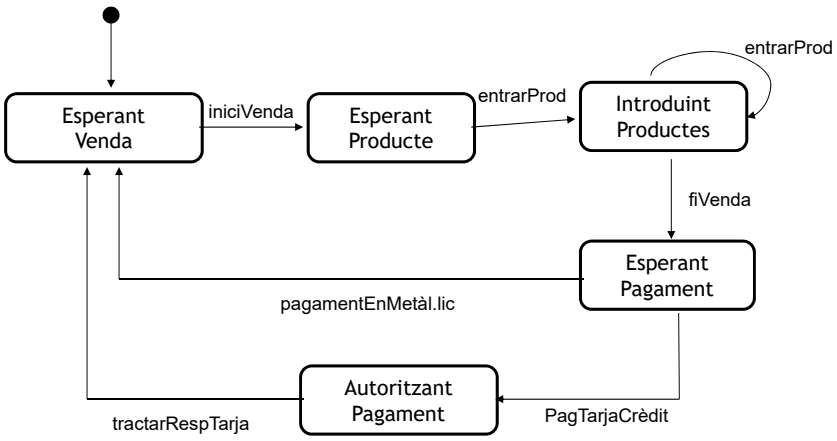


Ús dels diagrames d'estat

- Un diagrama d'estats es pot especificar per a una:
  - **Classe d'objectes**:
    - per descriure perquè els objectes canvien de subclasse
    - les subclasses d'un diagrama d'estats no tenen perquè aparèixer explícitament a l'esquema conceptual
    - per descriure classes d'objectes amb important "comportament dinàmic"
  - **Cas d'ús**:
    - per descriure la seqüència legal en la que els esdeveniments es poden produir al món real

      p.ex. en una compra de producte no es pot fer el pagament fins que no s'hagi tancat la venda.

Diagrama d'estats del cas d'us "comprar productes"



Bibliografia

- Larman, C. "*Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design*", Prentice Hall, 2005, (3ª edició)
- Rumbaugh, J.; Jacobson, I.; Booch, G. *"The Unified Modeling Language Reference Manual"*, 2ª edició, Addison-Wesley, 2004
- UML 2.3 Superstructure Specification
  http://www.omg.org/spec/UML/2.3/Superstructure/PDF
  OMG, 2010