Creació de taules d'una base de dades relacional a partir del diagrama de classes d'anàlisi d'UML

Josep Lluís Figueras Gómez
Enginyeria en Informàtica

Anna Queralt Calafat

10 de Gener de 2005

Dedicatòria i agraïments

Dedico aquest projecte a la meva família, Roseta, Mercè, Xavi... i als amics de Cyclops, Dolors, Carles, Ignasi...

Un record pel meu avi Josep, mai podré agrair-te prou tot allò que de tu vaig aprendre.

Agraeixo molt a la meva consultora Anna tota la seva col·laboració i paciència durant l'elaboració d'aquest projecte.

Resum

És ben sabut que les úniques etapes del cicle de vida del programari que necessàriament s'han d'especificar són les de recollida de requeriments i l'anàlisi o especificació del programari. La resta (disseny, implementació i prova) es poden generar de forma més o menys automàtica a partir de l'anàlisi. En aquest PFC hem pretès veure la viabilitat de la construcció automàtica de codi SQL a partir de diagrames de classes d'anàlisi UML. S'ha estès l'eina de modelatge UML Poseidon amb un plug-in de manera que, amb una interfície molt simple, es pot obtenir molt ràpidament l'esquema bàsic d'una base de dades incloent taules, les seves columnes, claus primàries i foranes i també els disparadors (i les claus úniques) necessaris per garantir les restriccions de cardinalitat de les associacions.

Ha calgut estudiar com fer la traducció d'UML a SQL. No presenta cap dificultat la traducció de classes a taules i poca la d'atributs a columnes, però la traducció d'associacions a taules-relació, claus foranes, claus úniques o disparadors té una certa complexitat. S'ha de tenir en compte que les associacions de vegades tenen la forma de classe associativa. S'ha fet un estudi exhaustiu de la traducció de les associacions cas per cas. També s'han tractat les relacions d'herència.

L'interès d'aquest PFC és doble: per una banda, l'estudi teòric de la transformació d'UML a SQL, i per altra banda el mateix plug-in, que permet obtenir molt ràpidament un esquema bàsic d'una base de dades a partir de diagrames de classes d'anàlisi UML. Això darrer pot ser útil per analistes i dissenyadors i pot ajudar a la creació de prototipus. La integració total del plug-in en una eina de modelatge UML és també un avantatge.

Existeix un plug-in de traducció UML a SQL que porta ja incorporat l'eina Poseidon (accessible amb Generation/SQL) que té moltes mancances que s'han pogut superar amb el plug-in construït. Com a clara innovació respecte al plug-in de Gentleware hi ha el tractament dels extrems fixats de les multiplicitats als extrems de les associacions amb disparadors i claus úniques, el tractament de l'herència, el tractament de les multiplicitats múltiples als extrems de les associacions, el tractament dels valors per defecte, una major intel·ligència en la generació (no repetir informació o no representar un concepte dues vegades) i una major adequació a l'estàndard SQL dels tipus UML. No es controlen el errors de generació de codi deguts a que s'han posat estereotips o valors etiquetats no adequats o incoherents o bé no s'han posat i són necessaris; encara que el plug-in de Gentleware ho intenta fer, no detecta errors evidents (com la manca de clau primària) i dóna error per coses que no se n'han de considerar (com definir una longitud a un String), com s'explica més endavant. En resum, pensem que és una eina clarament superior, a la que caldria millorar la interfície i anar ampliant-la amb altres coses, com la possibilitat de tractar la persistència, les claus úniques definides al model o les restriccions del tipus check, de controlar o solucionar quan sigui possible els errors de generació, etc.

Índex de continguts

Índex de figures

1. Introducció

1.1. Justificació del PFC i context en el qual es desenvolupa: punt de partida i aportació del PFC

Com sabem, el cicle de vida del programari consta de les fases de recollida de requeriments, anàlisi, disseny, construcció, proves (i més tard també podem incloure-hi el manteniment). Però, les úniques etapes del cicle de vida del programari que necessàriament s'han d'especificar són les de recollida de requeriments i l'anàlisi o especificació del programari. La resta (disseny, implementació i prova) es poden generar de forma més o menys automàtica a partir de l'anàlisi, possibilitat que s'inclou en algunes eines CASE.

Aquest PFC es troba emmarcat dins la Generació Automàtica de codi. De codi n'hi ha de vàries menes i existeixen actualment diversos generadors, amb més o menys eficàcia i eficiència en la generació. Típicament, a partir d'un diagrama en un llenguatge de modelatge (UML, per exemple), es genera codi que "materialitza" els conceptes i comportaments que representa el diagrama (per exemple, un diagrama de col·laboració). El codi podria obtenir-se també a través d'especificacions textuals (que segueixin o no una estructura gramatical).

En aquest PFC es pretén estendre una eina existent per a que pugui generar l'esquema d'una base de dades SQL a partir de diagrames de classes d'anàlisi. S'ha fet servir l'eina Poseidon (de l'empresa Gentleware), la qual s'ha estès per a que es puguin obtenir les instruccions SQL necessàries per crear la base de dades corresponent a un diagrama de classes UML. Existeix ja un plug-in d'aquestes característiques incorporat a l'eina Poseidon, però és molt simple i no és capaç de fer coses tan bàsiques com crear taules-relació com a conseqüència de relacions molts a molts.

1.2. Objectius del PFC

L'objectiu general d'aquest projecte és mostrar la viabilitat de generar codi SQL un cop definides les classes d'anàlisi i estudiar la manera de fer la traducció d'UML a SQL. En particular, s'ha generat codi de base de dades SQL per crear les taules i les seves relacions a partir de diagrames de classes d'anàlisi UML. Per a fer-ho, s'ha afegit a l'eina Poseidon una extensió. Només s'ha tingut en compte la part estàtica i no pas el comportament (si bé ha calgut generar disparadors SQL per tal que el codi generat es correspongués al màxim amb l'especificació).

Ha calgut estudiar quines transformacions intermèdies calien per traduir una especificació a model relacional, les quals estan en funció del tipus d'element UML concret i de com està relacionat amb els altres elements. Per això s'ha fet una anàlisi per casos de com s'ha de traduir cada element. Aquesta és una de les claus del PFC.

El projecte inclou els següents elements bàsics:
- Planificació temporal
- Anàlisi de la traducció UML a SQL element per element
- Breu explicació de com s'estén l'eina, indicant el problemes que han sorgit i com s'han solventat
- Desenvolupament d'una extensió que tradueix d'UML a SQL
- Documentació de l'extensió
- Jocs de prova

1.3. Enfocament i mètode seguit

Com que es pretenia fer l'estudi de la traducció dels diagrames de classes d'anàlisi UML a taules SQL, ha calgut identificar abans que res què es pot trobar en un diagrama d'aquesta mena. Paral·lelament, s'ha vist què pot aparèixer en les sentències de creació de taules en SQL. S'ha fet llavors un mapeig d'UML a SQL. L'estudi realitzat s'ha inclòs a la present memòria. Era important fitar la feina, decidir què es faria i què no. Es va decidir mapejar només atributs de tipus simples i considerar que tots els atributs són persistents. Les classes també s'han considerat totes com a persistents.

De forma paral·lela s'ha estudiat com es fa un plug-in amb l'eina Poseidon i com s'hi incorpora. Al mateix temps també, s'ha estudiat quina tecnologia utilitzar per fer el plug-in. Es coneixia que hi ha un API que permet l'accés als diagrames de Poseidon. Un cop accedits, s'ha estudiat la manera de generar els scripts SQL corresponents.

En una segona fase, aclarida la feina que s'ha de fer i com fer-ho, s'ha escrit el plug-in, s'han fet els jocs de prova corresponents i s'ha documentat l'esmentat plug-in.

El tractament de l'herència no es va contemplar al principi, però s'ha pogut afegir al final. Tampoc es pensaven tractar agregacions i composicions, però al final s'han tractat com a associacions simples; en una fase posterior es podria pensar quin tractament cal donar a aquestes relacions estructurals.

Durant tota la duració del PFC s'ha escrit la present memòria.

1.4. Planificació del projecte

Aquí s'inclou la planificació temporal de totes les tasques anteriors.

La relació de tasques finalitzades amb les entregues és la següent:

- PAC1 (Pla de treball, 22/09/2004)
- Planificació (15/9/2004 al 22/9/2004,: 6 dies)

- PAC2 (02/11/2004)
- Anàlisi de com es tradueix UML a SQL (23/9/2004 al 2/11/2004: 29 dies)
- Estudiar tecnologia més adequada (30/9/2004 al 13/10/2004: 10 dies)
- Investigació i explicació d'extensió de l'eina (23/9/2004 al 6/10/2004: 10 dies)

- PAC3 (09/12/2004)
- Desenvolupament plug-in UML (22/10/2004 al 9/12/2004: 35 dies)
- Jocs de prova (23/11/2004 al 9/12/2004: 13 dies)

- PAC4 (Memòria i presentació, 10/01/2005)
- Documentació plug-in (10/12/2004 al 20/12/2004: 7 dies)
- Preparar presentació (20/12/2004 al 31/12/2004: 10 dies)
- Escriptura de memòria (23/9/2004 al 5/1/2004: 75 dies)

El temps dedicat s'ha ajustat força al que s'havia planificat. Fins i tot, la Documentació del plug-in es va poder lliurar a la PAC3, abans del que s'havia previst.

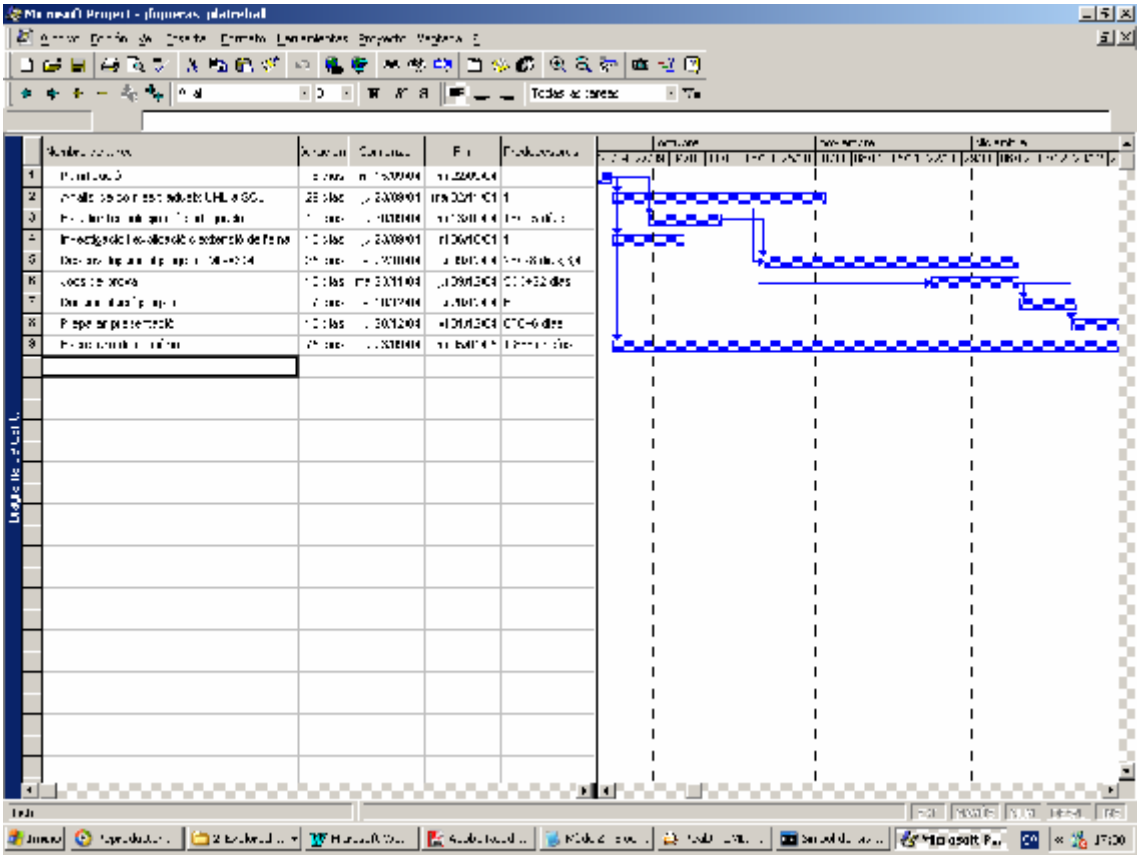


Fig. 1: Planificació temporal

1.5. Productes obtinguts

El producte d'aquest PFC és el codi corresponent al generador de codi DDL/SQL. Es tracta d'un conjunt de classes Java, concretament

Classe de frontera:

- ClassesUMLaTaulesPane.java

Classe de control:

- Logic.java

Classes d'entitat:

- Associacio.java,
- Atribut.java
- Cardinalitat.java
- Classe.java
- ClauForana.java
- ClauPrimaria.java
- ClauUnica.java
- Columna.java
- Disparador.java
- Estereotip.java
- Generalitzacio.java
- Participant.java
- Taula.java
- ValorEtiquetat.java

Classes d'utilitat:

- Configuracio.java
- TAssociacio.java
- TClasse.java
- TGeneralitzacio.java
- TTaula.java

Classe d'instal·lació del plug-in:

- ApiDemoPlugin.java: Serveix únicament per instal·lar el plug-in.

Interfície:

- GrupColumnes.java

Aquestes classes i interfície estan explicades a l'*Annex A: Documentació*.

El fitxer resultant de la compilació de tot plegat es diu *classesumlataules.jar*. Més endavant s'explica com obtenir-lo mitjançant la compilació amb l'eina *ant*, i també com instal·lar el plug-in.

1.6. Breu descripció dels altres capítols de la memòria

En l'apartat *2. Anàlisi de com es tradueix del diagrama de classes UML a taules SQL* trobem una introducció prèvia on s'explica, de mode introductori, què podem trobar en un diagrama de classes d'anàlisi UML i què hi pot haver en una taula d'una base de dades SQL (o sigui, d'on partim i on volem anar). Se situa el diagrama de classes d'anàlisi dins del model estàtic d'UML, s'enumeren els components que pot tenir i es detalla la sentència de creació de taules en SQL. Posteriorment, s'explica com traduir el diagrama de classes UML a taules SQL element a element (classes, atributs, taules, associacions, agregacions, composicions i generalitzacions). La traducció de les associacions és la que mereix un tractament més llarg per la seva major complexitat; se'n donen alguns exemples.

En l'apartat *3. Estudi de la tecnologia més adequada per generar codi DDL-SQL* s'explica la forma d'accedir al model de diagrames de l'eina Poseidon amb la seva pròpia API (però també es comenta l'alternativa d'accés per XMI). S'estudia com passar des d'aquesta API fins a codi DDL/SQL de forma general, creant en aquest punt els diagrames de classes d'anàlisi de l'aplicació. Es decideix quins valors etiquetats i estereotips es faran servir per poder generar el codi.

En l'apartat *4. Investigació i explicació de l'extensió de l'eina* s'explica com s'instal·len els plug-ins amb l'eina Poseidon i la manera de compilar-los amb l'eina *ant*.

En l'apartat *5. Desenvolupament del plug-in UML a SQL* s'inclouen els diagrames de casos d'ús. els de classes de disseny i els de seqüència de l'aplicació. Tambés'inclouen diversos jocs de prova del plug-in construït.

Segueixen a aquests apartats una *Valoració econòmica* del treball realitzat, les *Conclusions*, el *Glossari* i la *Bibliografia*. S'acaba la memòria amb un annex on s'inclou la *Documentació* del plug-in i un altre on s'explica com instal·lar el plug-in.

2. Anàlisi de com es tradueix del diagrama de classes UML a taules SQL

2.1. Introducció

2.1.1. El diagrama de classes d'anàlisi d'UML

El diagrama de classes dins del model estàtic d'UML

El model estàtic d'UML és aquell on es descriuen les relacions entre les classes i els objectes. És "estàtic" perquè mostra totes les possibles relacions al llarg del temps, no les que són vàlides en un moment determinat. Es composa del diagrama de classes i el diagrama d'objectes. El *diagrama de classes* pot contenir classes (i objectes) i relacions entre elles; el *diagrama d'objectes* conté objectes i relacions entre ells i es fa servir per fer exemples del diagrama de classes.

Un diagrama de classes mostra l'estructura estàtica de les classes en un domini (part de la realitat que considera una aplicació). S'hi mostren classes i les relacions entre elles (associació, agregació, herència o ús).

El model estàtic s'utilitza en tot el cicle de vida, fent servir diferents tipus d'objectes. En l'anàlisi es consideren objectes del món de l'usuari (factures, productes, clients...), mentre que en el disseny es consideren objectes de la tecnologia informàtica (pantalles, gestors de disc...).

Els llenguatges de programació suporten (d'una manera més o menys completa) les relacions d'herència però no distingeixen entre associacions, agregacions o usos; per això, en passar del disseny a la programació caldrà fer transformacions.

El diagrama estàtic de disseny i el d'anàlisi

El diagrama estàtic de disseny sol tenir moltes més classes que el d'anàlisi (factor típic: 5 a 1). Però la fase de programació definirà encara més classes, moltes d'elles amb un paper purament instrumental.

La revisió del diagrama estàtic de disseny té en compte els aspectes següents: la normalització dels noms, la reutilització de classes, l'adaptació de l'herència al llenguatge de programació, l'increment de la velocitat i la reducció del trànsit de missatges mitjançant l'agrupació de classes, la millora del rendiment, l'addició de classes temporals per guardar resultats intermedis, i la revisió des del punt de vista de la cohesió i l'acoblament.

Algunes diferències destacables entre el diagrama de classes de disseny i el d'anàlisi són que el primer admet associacions amb direcció, classes de control o frontera i operacions assignades a les classes i el segon no. El de disseny no pot tenir associacions ternàries ni classes associatives, contràriament al diagrama d'anàlisi.

En el nostre PFC només considerarem el diagrama de classes d'anàlisi.

Elements presents en un diagrama de classes d'anàlisi

Una classe és un tipus de *classificador*, el qual és l'entitat bàsica del model estàtic. Un classificador és quelcom més genèric que una classe; és un conjunt els elements del qual s'anomenen instàncies. Una *classe* és la descripció d'un conjunt d'objectes que comparteixen els mateixos atributs, operacions, relacions i semàntica. Es representa amb un rectangle i té un nom. Una classe pot tenir *atributs*, que són propietats de les classes (identificades amb un nom), que descriuen un rang de valors que poden prendre les instàncies. Les *instàncies* són realitzacions concretes d'una abstracció, entitats a la quals se'ls pot aplicar un conjunt d'operacions i que tenen un estat que guarda l'efecte de les operacions. Les instàncies tenen sempre entitat pròpia, de manera que es diferencien entre sí malgrat que tots els valors dels atributs puguin ser exactament iguals. Una classe pot tenir també *operacions*, que són implementacions de serveis que es poden demanar a qualsevol objecte de la classe per a mostrar un comportament; aquestes operacions s'implementen amb *mètodes*.

Hi ha tres menes de classes:

- *Classes d'entitat*: Representen objectes del domini. Modelen entitats o esdeveniments del món real dels quals el programari ha de fer servir informació. Molts d'aquests objectes han de ser persistents.
- *Classes de frontera*: Representen la interfície de l'usuari amb la pantalla, o sigui la interfície d'usuari entre un cas d'ús i un actor. Representen objectes gràfics complexos (finestres, botons...) i normalment tenen pocs o cap atribut.
- *Classes de control*: Són objectes interns del programari (no visibles pels usuaris, no modelen res del món real) i no persistents.

Com que tractem el diagrama de classes d'anàlisi, en el nostre PFC totes les classes es consideraran com a classes d'entitat (totes les altres classes es tractaran de la mateixa manera). Per la mateixa raó, en els diagrames no haurien d'aparèixer operacions a les classes (si n'hi ha, seran ignorades).

Hi ha diverses menes de relacions entre classes que es representen al diagrama de classes d'anàlisi:

- *Associació*: És una relació estructural que especifica que els objectes d'un element es connecten als objectes d'un altre.
  Existeix una mena especial de classes, les *classes associació*, les quals incorporen les propietats d'una associació.
- *Agregació*: És un cas d'associació binària en què un element és "part" d'un altre (agregat). Si els continguts no tenen sentit sense el contenidor i no poden formar part de més d'un contenidor, parlem de *composició*; altrament d'*agregació simple*.
- *Herència*: És un mecanisme pel qual elements més específics incorporen l'estructura i el comportament d'elements més generals. Entre classes, una *subclasse* comprèn un subconjunt dels objectes de la *superclasse*, els quals tenen tots els atributs de la superclasse i alguns addicionals específics de la subclasse. En el diagrama de classes de disseny també s'hereten (i es poden sobreescriure) les operacions.
- *Dependència*: Una relació de dependència expressa que un element del model (el client) depèn per a la seva implementació o funcionament d'un altre element (el subministrador).

En el nostre PFC considerarem les associacions (incloses les agregacions i composicions, que es tracten de la mateixa manera), les classes associació i les relacions d'herència. La direcció de les associacions, si n'hi ha, serà ignorada, ja que no hauria d'aparèixer en un diagrama de classes d'anàlisi.

2.1.2. La creació de taules en SQL

La sentència CREATE TABLE

El DDL (Data Definition Language) és el subconjunt del llenguatge SQL destinat a la creació o modificació dels objectes de la base de dades (taules, vistes, procediments, disparadors...). El DML (Data Manipulation Language) està destinat, contràriament, a la manipulació de les dades (amb operacions ara com INSERT, UPDATE, DELETE o SELECT). El DDL pot canviar força d'un SGBD a un altre. Aquí l'estudi se centra en l'SQL estàndard oblidant les particularitats dels SGBDs concrets.

La sentència CREATE TABLE serveix per crear l'estructura d'una taula, permetent definir les columnes que té i les restriccions que han de complir aquestes columnes. Una restricció consisteix en la definició d'una característica addicional que té una columna o un grup de columnes. Les restriccions de columna apareixen dins de la definició d'una columna després del tipus de dades i afecten únicament a la columna que s'està definint. Les restriccions de taula es defineixen després de definir totes les columnes de la taula i es fan sobre una columna o un grup de columnes.

La sentència CREATE TABLE té la següent sintaxi principal:

```
CREATE TABLE [usuari.]nom_taula(
{dades_columna | restriccions de taula}
  [,{dades_columna | restriccions de taula}]...
)

dades_columna:
        nom tipus_de dades [DEFAULT expressió]  [restricció_de_columna]
```

També s'hi poden incloure definicions de nivell físic (com ara decidir a quin TABLESPACE s'encabirà una taula), però no tenen interès per aquest PFC i no les tractarem.

Restriccions associades a taules

Les restriccions es guarden al catàleg, és interessant que els donem un nom, però això no és d'interès per aquest PFC.

- UNIQUE : no es poden repetir valors en tuples diferents

```
[CONSTRAINT nom] UNIQUE (columna[,columna]...)
```

- PRIMARY KEY: les columnes que en formen part formen la clau primària, que com a conjunt han de ser UNIQUE i individualment han de ser NOT NULL.

```
[CONSTRAINT nom] PRIMARY KEY (columna[,columna]...)
```

- FOREIGN KEY: les columnes que en formen part formen una clau forana contra la clau primària de la taula REFERENCES.

```
[CONSTRAINT nom] FOREIGN KEY (columna[,columna]...)
REFERENCES [usuari.]nom_taula  (columna[,columna]...)
```

- CHECK permet condicions lògiques entre columnes

[CONSTRAINT nom] CHECK (condició_amb_diversos_camps)

No es tractaran els CHECKS en aquest PFC ja que no són necessaris per a traduir associacions i classes associatives. En tot cas, tindrien sentit si es traduïssin restriccions d'integritat addicionals a les que permet expressar el model UML.

Restriccions associades a les columnes

- NOT NULL: la columna no admet valors nuls.

[CONSTRAINT nom] NOT NULL

- UNIQUE, PRIMARY KEY i CHECK: Són les mateixes restriccions que les aplicades a les taules, i tenen el mateix significat.

[CONSTRAINT nom] UNIQUE

[CONSTRAINT nom] PRIMARY KEY

[CONSTRAINT nom] CHECK (condició)

Tampoc tractarem els CHECKS com a restriccions associades a les columnes en aquest PFC, igual que a l'apartat anterior.

Valors per defecte

- DEFAULT: No són restriccions de columna, es tracta dels valors que cada columna té si no se li especifica cap valor en el moment de la creació de la tupla.

DEFAULT valor_per_defecte_de_columna

Els tractarem en el PFC.

Tipus de dades en SQL

En aquest projecte només es tractaran tipus simples. Existeixen moltes variacions en els tipus de dades que es poden assignar a una columna als SGBD comercials. Només s'inclouen els tipus que es tractaran en aquest PFC, procurant que siguin el més estàndard possible:

- Cadenes de longitud fixa: CHAR, CHARACTER
- Cadenes de longitud variable: VARCHAR
- Números amb precisió i escala: NUMERIC, TINYINT, BIGINT, SMALLINT, INTEGER, DOUBLE, REAL
- Dates: DATE, TIME
- Booleans: BIT

2.2. Com traduir del diagrama de classes d'anàlisi d'UML a taules SQL

2.2.1. Introducció

La primera idea que se'ns acudiria de forma completament natural és la que seguirem: mapejar classes sobre taules, atributs sobre columnes, tipus sobre tipus de dades i associacions sobre relacions. Inicialment es mapejarà tot, ignorant el fet que les classes o els atributs poden tenir el valor etiquetat *persistent* amb el valor *false*, que indicaria que no s'haurien de mapejar.

2.2.2. Mapeig de classes

El mapeig de classe a taules normalment serà una a una. El mapeig podria ser diferent per raons lligades a la base de dades; no tindrem en compte aquestes raons i el mapeig serà sempre de una a una.
Podrien aparèixer també taules com a conseqüència de relacions molts a molts (taules-relació) i de classes associació (que són associacions que tenen atributs, les quals podem considerar com a autèntiques classes).
Totes les classes es consideraran persistents.

2.2.3. Mapeig d'atributs i de tipus

El mapeig d'atributs a columnes és senzill. Podria ser que alguns atributs no fossin persistents, o bé que calguessin columnes a les taules (per raons lligades a la base de dades) que no fossin a les classes. Tot això no ho tractarem, considerarem tots els atributs com a persistents i no ens preocuparem de raons de rendiment de les bases de dades.

El mapeig de tipus a tipus de dades és senzill. Aquí s'inclou la taula de conversió que seguirem entre els tipus UML de l'eina Poseidon (que són realment tipus Java) i els tipus de dades SQL.

Tipus UML	**Tipus de dades SQL**
BigDecimal	NUMERIC
BigInteger	BIGINT
Boolean	BIT
Byte	TINYINT
Char	CHAR
Character	CHARACTER
Date	DATE
Double	DOUBLE
Float	REAL
Integer	INTEGER
Int	INTEGER
Long	BIGINT
Short	SMALLINT
String	VARCHAR
Time	TIME

Fig. 2: Correspondència entre tipus UML i tipus de dades SQL

S'utilitzaran els següents modificadors de tipus:

- *length*: Per a tipus alfabètics, la longitud màxima de l'atribut
- *precision*: Per a tipus numèrics, la longitud màxima de l'atribut
- *scale*: Per a tipus numèrics, la longitud de la part decimal de l'atribut (inclosa dins la *precision*).
- *default*: Valor per defecte de l'atribut.

L'usuari ha de definir aquests modificadors correctament. Si en defineix algun que no correspon o en deixa de definir algun, el resultat no serà l'esperat (per exemple, els tipus similars a Integer no tenen *scale*; també, si es defineix la *scale* és obligatori definir la *precision*). No es controlaran els d'errors de generació per mal ús dels valors etiquetats o estereotips, de manera que els scripts generats poden no ser correctes. Pensem que l'usuari, que en principi coneix bé les sentències DDL/SQL, veurà fàcilment els errors comesos, i que en qualsevol cas l'SGBD li proporcionarà l'error quan intenti crear els objectes de bases de dades corresponents. Per exemple, si l'usuari veu PRIMARY KEY() s'adonarà que no ha definit la clau primària de d'una taula.
Si es continua aquest PFC podria incloure's com a opció detectar errors o no de generació i el grau de gravetat d'aquests per a ser explicitats. El que no és lògic és que si definim un valor etiquetat *length* a un atribut *atrib* de tipus String es donguin els següents errors que dóna el plug-in de Gentleware:

atrib is not valid SQL data type.
atrib - String only takes 0 parameter(s).

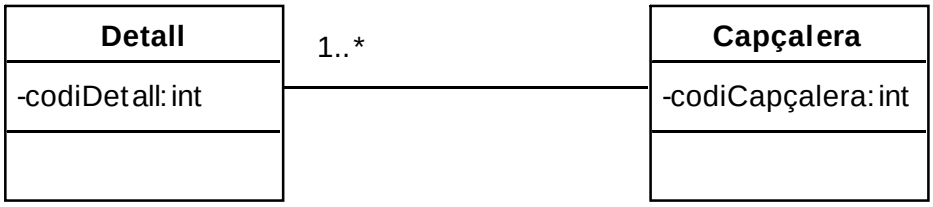
Aquests errors no haurien d'aparèixer. Un String no té longitud en un diagrama UML, però té sentit que la definim com el valor etiquetat *length* si volem fer una traducció a SQL (no està modificant essencialment la semàntica del diagrama).

El fet que existeixen altres tipus de dades, que l'usuari en pugui definir de propis i que alguns dels sistemes gestors de bases de dades (SGBD) més populars s'apartin de l'SQL estàndard (per exemple, Oracle, que usa intensivament el tipus de dades VARCHAR2 en comptes de VARCHAR), faria recomenable facilitar que l'usuari pugui canviar o ampliar aquesta taula de traducció (es podria fer amb un fitxer de configuració, però també posant la taula ben localitzada al codi perquè es pugui canviar fàcilment i recompilar després). Si utilitza algun tipus de dades no inclòs s'inclourà tal qual a l'script, però amb un comentari d'avís a l'usuari de que les sentències DDL podrien no ser correctes.

2.2.4. Mapeig d'associacions

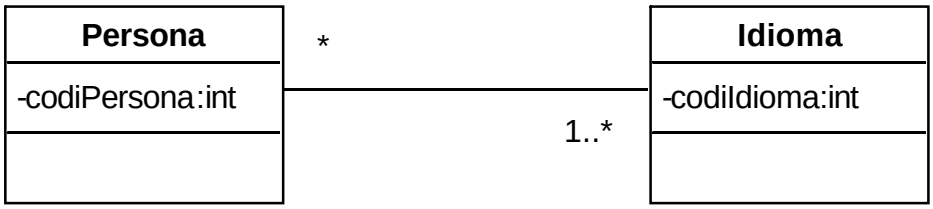
Exposarem primer els fonaments teòrics de la transformació d'associacions. Tenint en compte que parlem de diagrames d'anàlisi, no tindrem en compte la navegabilitat de les associacions. Seguidament comentarem què fer en cas de trobar-nos amb cardinalitats múltiples als extrems de les associacions i posarem alguns exemples.

Per situar-nos, si tenim el següent diagrama de classes d'anàlisi UML



podem endevinar que en SQL tindríem una taula Capçalera i una altra taula Detall, la qual tindria una clau forana obligatòria contra Capçalera.

Si tenim aquest altre diagrama



la traducció a SQL seria una taula de relació entre Persona i Idioma, que tindria com a columnes les claus primàries d'aquestes dues taules, que també serien clau primària de la taula de relació i claus foranes contra la taula de la qual provenen.

Anem a formalitzar tot això i a completar-ho amb respostes a preguntes com: què passa si s'intenta esborrar l'últim idioma que parla una persona de la taula de relació? Per fer-ho, farem servir la següent notació (referida al darrer exemple):

Persona(0..*)-(1..*)Idioma

on com es veu seguim la posició dels elements al dibuix, posant les classes als extrems, enmig les cardinalitats i, per separar-les, un guió que simula l'associació. Si hagués cardinalitats múltiples en un extrem, les separaríem amb comes dins del parèntesi corresponent.

2.2.4.1. Com mapejar les associacions sobre diversos elements de base de dades

Primerament, comentarem què es considera que representen les següents cardinalitats:

- {0..*}: cardinalitat inferior = 0 *i* (cardinalitat superior = "*" *o* cardinalitat superior > 1)
- {1..*} : cardinalitat inferior >= 1 *i* (cardinalitat superior = "*" *o* cardinalitat superior > 1)
- {0..1}: cardinalitat inferior = 0 *i* cardinalitat superior = 1
- {1}: cardinalitat inferior = 1 *i* cardinalitat superior = 1

Seguidament, establirem una relació d'ordre entre la cardinalitat dels extrems de les relacions, així:

{0..*} < {1..*} < {0..1} < {1}

Quant més a la dreta de l'expressió és una cardinalitat més força té per imposar les seves "regles".
Suposem que tenim una relació binària, ordenada de manera que el primer extrem és més *petit* (segons la relació d'ordre anterior) que el segon (si fossin iguals, podríem

decidir que el petit seria el de menor nom alfabèticament, o bé deixar la tria en mans de l'usuari). Un cop ordenats així els extrems, les regles de tractament de les associacions serien les següents, segons la cardinalitat que tingui el segon extrem:

- {1}: Cal crear una clau forana obligatòria a la taula del primer extrem de l'associació.
- {0..1}: Cal crear una clau forana opcional a la taula del primer extrem de l'associació.
- {1..*}: Cal crear una nova taula de relació entre les taules que corresponen als dos extrems de l'associació.
- {0..*}: Cal crear una nova taula de relació entre les taules que corresponen als dos extrems de l'associació.

S'observa que la relació d'ordre {0..*} < {1..*} és la menys clara de totes i fins i tot es podrien considerar iguals les dues cardinalitats. De tota manera, {1..*} obligarà més endavant a crear alguns disparadors i {0..*} no, de manera que és més exigent i per tant té més força.

Tot l'anterior val si no hi ha una classe associativa a la relació. Si n'hi ha, cal convertir la classe associativa en una taula de relació C entre les taules que corresponen als dos extrems de l'associació. Es podria considerar llavors, si seguim la notació anunciada anteriorment que

A(X) – (Y)B amb classe associativa C (on X,Y són les multiplicitats o conjunts de multiplicitats dels extrems de la relació) és equivalent a A(1) – (X)C(Y) – (1)B, i per tant la traducció hauria de ser la combinació de les traduccions de les relacions A-C i C-B, on a més la clau primària de C ha d'estar formada per les claus foranes d'A i de B.

La conseqüència d'això és que la taula de relació C contindrà les claus primàries de les altres dues taules, que formaran la seva pròpia clau primària, i seran clau forana contra les taules d'on provenen.

Posteriorment, tant si l'associació té una classe associativa com si no, per a cadascun dels extrems de l'associació:

a) Si la cardinalitat inferior d'un extrem de la relació (k) és igual o superior a 1, cal crear un disparador d'esborrat que eviti que existeixin files de l'altre extrem de la relació que es relacionin amb menys de k files de l'extrem que estem considerant. Això és completament innecessari en els extrems de relacions que són apuntats per una clau forana des de l'altre extrem (això indica que la cardinalitat superior d'aquest extrem és igual a 1 i la coherència ja queda assegurada per la integritat referencial causada per la clau forana).

b) Si la cardinalitat superior de l'extrem de la relació (k) és superior a 1, cal crear un disparador d'inserció o actualització que eviti que existeixin files de l'altre extrem de la relació que es relacionin amb més de k files de l'extrem que estem considerant.

c) Si la cardinalitat superior de l'extrem de la relació (k) és igual a 1, cal crear una clau única a la taula que correspon a la classe de l'extrem que estem considerant formada per la clau forana que s'haurà creat contra l'altre extrem. Això no té sentit en els extrems de relacions que són apuntats per una clau forana des de l'altre extrem (això indica que la cardinalitat inferior d'aquest extrem és 0 o 1 i la coherència ja queda assegurada per la pròpia la clau forana); apart, la columna sobre la que s'hauria de fer una clau única ni tan sol existeix a la taula corresponent a l'extrem de la relació que estem considerant.

Cal aclarir que quan es parla de “claus foranes des de l’altre extrem de la relació” ens referim justament a això, està clar que si existeix una classe associativa la seva taula corresponent apuntarà amb claus foranes als dos extrems de la relació (però això no té cap rellevància per la creació de disparadors o claus úniques que acabem de comentar).

2.2.4.2. Transformació de cardinalitats múltiples

Si tenim l'associació

$A(C_{a1}, C_{a2} ...C_{aN})$ - $(C_{b1}, C_{b2} ...C_{bN})B$, on C_{Ki} vol dir cardinalitat i-èssima de la classe K,

per fer la transformació a taules a cada extrem caldrà simplificar les cardinalitats fins a convertir-les en una de sola que sigui el menys restrictiva possible. Així, la cardinalitat “resum” d’un extrem sempre serà la més *petita* en el sentit definit en l’apartat anterior. L’única excepció és la combinació {1..*, 0..1} que no donarà {1..*} (que és la més petita) sinó {0..*} (que inclou també el zero de 0..1).

Els límits fixats caldrà guardar-los tots per poder construir disparadors posteriorment. Així, la combinació {3..5, 9..*} és equivalent segons el que s’ha comentat a l’apartat anterior a {1..*, 1..*} que dóna com a resultat {1..*} (ja que és la cardinalitat més petita). Caldrà guardar els límits inferiors 3 i 9 i el superior 5 per poder construir disparadors que controlin que si el nombre de files és menor que 3 o és entre 5 i 9 cal llençar una excepció.

2.2.4.3. Alguns exemples de transformació d’associacions

En aquest PFC només tindrem en compte les associacions *binàries*. Aquí inclourem alguns exemples de transformació d’aquest tipus d’associacions, que segueixen sempre els fonaments teòrics explicats anteriorment. Als jocs de prova es pot obtenir més informació per casos concrets no coberts en aquesta secció.

Farem servir la notació classe(cardinalitat) - (cardinalitat)classe que recorda el dibuix d’una associació entre classes, i on el guió representa la pròpia associació.

<u>Cardinalitats bàsiques sense classe associativa</u>

Partirem del següent diagrama:

L’estereotip PRIMARY KEY marca cada atribut com a integrant de la clau primària de la taula que es correspondria amb la classe on són els esmentats atributs. També hagués estat possible generar automàticament claus primàries "artificials” per les taules, però sembla millor donar el control a l’usuari sobre aquesta assignació.

Afegirem tot seguit una associació entre les classes A i B i anirem canviant les seves multiplicitats als extrems.

Aquestes són les taules que es generen si no hi ha cap associació formen la solució general, i no les inclourem quan no variïn en cada cas particular respecte a aquesta solució:

```
CREATE TABLE A(
atrib_a1 INTEGER(4),
PRIMARY KEY(atrib_a1)
);

CREATE TABLE B(
atrib_b2 VARCHAR(5),
atrib_b1 INTEGER(7),
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

Els disparadors s'han fet genèrics, seguint l'estil del SGBD Oracle. Caldria adaptar-los al SGBD que s'utilitzi en cada moment. Hem suposat que estem en un entorn transaccional, de manera que un cop llençada l'excepció el programa que la recull pot tirar enrera la transacció (ROLLBACK). Això no es pot fer dins d'un disparador, ja que el disparador no coneix la transacció on es dispara i podria fer perdre dades pendents de confirmar importants, apart de que molts SGBDs simplement no permeten ROLLBACKs en un disparador.

a) *Associació A(0..*) – (0..*)B*

La solució és crear una taula de relació R que tingui les columnes de les claus primàries d'A i de B, que formaran claus foranes contra A i B respectivament i formaran la clau primària d'R.

```
CREATE TABLE R(
A_atribut_a_1 INTEGER(4),
B_atribut_b_1 VARCHAR(5),
B_atribut_b_2 INTEGER(7),
PRIMARY KEY(A_atribut_a_1, B_atribut_b_1, B_atribut_b_2),
FOREIGN KEY(A_atribut_a_1)
REFERENCES A(atribut_a_1),
FOREIGN KEY(B_atribut_b_1, B_atribut_b_2)
REFERENCES B(atribut_b_1, atribut_b_2)
);
```

b) *Associació A(1..*) – (1..*)B*

La solució és crear una taula de relació R que tingui les columnes de les claus primàries d'A i de B, que formaran claus foranes contra A i B respectivament i formaran la clau primària d'R. Cal afegir un trigger a la taula R per evitar que existeixin files d'A que no es relacionin amb cap fila de B a través de R i un altre per evitar que existeixin files de B que no es relacionin amb cap fila d'A a través de R.

```
CREATE TABLE R(
A_atribut_a_1 INTEGER(4) ,
B_atribut_b_1 VARCHAR(5) ,
B_atribut_b_2 INTEGER(7) ,
PRIMARY KEY(A_atribut_a_1, B_atribut_b_1, B_atribut_b_2),
FOREIGN KEY(A_atribut_a_1)
REFERENCES A(atribut_a_1),
FOREIGN KEY(B_atribut_b_1, B_atribut_b_2)
REFERENCES B(atribut_b_1, atribut_b_2)
);

CREATE TRIGGER R_1
AFTER DELETE ON R
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM R
  WHERE B_atrib_b2 = :old.B_atrib_b2
     AND B_atrib_b1 = :old.B_atrib_b1;
```

```
 IF (numFiles < 1) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = R, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TRIGGER R_2
AFTER DELETE ON R
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM R
 WHERE A_atrib_a1 = :old.A_atrib_a1;
 IF (numFiles < 1) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = R, files = ' || to_char(numFiles));
 END IF;
END;
```

c) Associació A(0..1) – (0..1)B

La millor solució és crear una clau forana opcional a la taula A contra la clau primària de la taula B i una clau UNIQUE a A per evitar que existeixin files de B que es relacionin amb més d'una fila d'A.

Es podria crea una clau forana a cada taula contra l'altra, però això té el problema que caldria que si una fila d'A apunta a una fila de B, la fila de B apunti a aquesta fila d'A i no a cap altra. Això es descarta per que és difícil de mantenir i duplica informació.
Alternativament, es podria crear una taula de relació C amb dues claus foranes, una apuntant a la taula A i una a la taula B, essent la clau primària la concatenació d'aquestes dues. Es descarta perquè afegeix una nova taula al model, i a més caldrien disparadors per evitar que existissin files d'A que es relacionessin amb més d'una fila de B i files de de B que es relacionessin amb més d'una fila d'A.

Les aparicions d'A podrien substituir-se per B i les de B per les d'A, quedant un model simètric i correcte. Es podria demanar a l'usuari quin model dels dos prefereix.

```
CREATE TABLE A(
atribut_a_1 INTEGER(4) ,
B_atribut_b_1 VARCHAR(5) ,
B_atribut_b_2 INTEGER(7) ,
PRIMARY KEY(atribut_a_1),
FOREIGN KEY(B_atribut_b_1, B_atribut_b_2)
REFERENCES B(atribut_b_1, atribut_b_2),
UNIQUE(B_atribut_b_1, B_atribut_b_2)
);
```

d) Associació A(1) – (1)B

La solució és crear una clau forana obligatòria a la taula A contra la clau primària de la taula B i una clau UNIQUE a A per evitar que existeixin files de B que es relacionin amb més d'una fila d'A. Caldrà un trigger a la taula A per evitar que existeixin files de B que no es relacionin amb cap fila d'A.
Les aparicions d'A podrien substituir-se per B i les de B per les d'A, quedant un model simètric i correcte. Es podria demanar a l'usuari quin model dels dos prefereix.

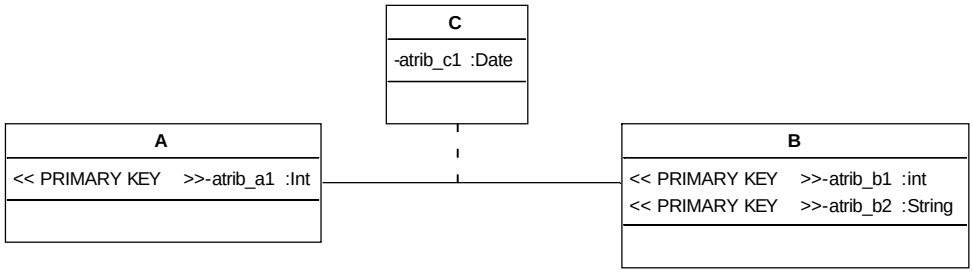
```
CREATE TABLE A(
atribut_a_1 INTEGER(4)  ,
B_atribut_b_1 VARCHAR(5)  NOT NULL,
B_atribut_b_2 INTEGER(7)  NOT NULL,
PRIMARY KEY(atribut_a_1),
FOREIGN KEY(B_atribut_b_1, B_atribut_b_2)
REFERENCES B(atribut_b_1, atribut_b_2),
UNIQUE(B_atribut_b_1, B_atribut_b_2)
);

CREATE TRIGGER A_1
AFTER DELETE ON A
FOR EACH ROW
```

```
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM A
 WHERE B_atrib_b2 = :old.B_atrib_b2
    AND B_atrib_b1 = :old.B_atrib_b1;
 IF (numFiles < 1) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = A, files = ' || to_char(numFiles));
 END IF;
END;
```

Cardinalitats bàsiques amb classe associativa

La solució general és transformar la classe associativa C en una taula de relació C que tingui a més les columnes de les claus primàries d'A i de B, que formaran claus foranes contra A i B respectivament i formaran la clau primària de C. Segons les cardinalitats dels extrems de l'associació potser caldrà afegir alguna clau única o un disparador. Les taules A i B no es modifiquen i no les escriurem en els exemples posteriors.

Partirem del següent diagrama:



Anirem canviant les cardinalitats als extrems de l'associació entre A i B.

Aquest diagrama genera les següents taules (sense tenir en compte les cardinalitats als extrems de l'associació). Considerarem aquest exemple com l'exemple general:

```
CREATE TABLE A(
atribut_a_1 INTEGER(4) ,
PRIMARY KEY(atribut_a_1)
);

CREATE TABLE B(
atribut_b_1 INTEGER(7) ,
atribut_b_2 VARCHAR(5) ,
PRIMARY KEY(atribut_b_1, atribut_b_2)
);

CREATE TABLE C(
atrib_c_1 DATE ,
A_atribut_a_1 INTEGER(4) ,
B_atribut_b_1 VARCHAR(5) ,
B_atribut_b_2 INTEGER(7) ,
PRIMARY KEY(A_atribut_a_1, B_atribut_b_1, B_atribut_b_2),
FOREIGN KEY(A_atribut_a_1)
REFERENCES A(atribut_a_1),
FOREIGN KEY(B_atribut_b_1, B_atribut_b_2)
REFERENCES B(atribut_b_1, atribut_b_2)
);
```

Els disparadors s'han construït de la mateixa manera que les relacions sense classe associativa.

a) Associació A(0..) – (0..*)B amb classe associativa C*
S'aplica la solució general.

b) *Associació A(1..*) – (1..*) B amb classe associativa C*

A més d'aplicar la solució general, cal afegir un trigger a la taula C per evitar que existeixin files d'A que no es relacionin amb cap fila de B a través de C i un altre per evitar que existeixin files de B que no es relacionin amb cap fila d'A a través de C.

```
CREATE TRIGGER C_1
AFTER DELETE ON C
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM C
 WHERE B_atrib_b2 = :old.B_atrib_b2
    AND B_atrib_b1 = :old.B_atrib_b1;
 IF (numFiles < 1) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TRIGGER C_2
AFTER DELETE ON C
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM C
 WHERE A_atrib_a1 = :old.A_atrib_a1;
 IF (numFiles < 1) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
 END IF;
END;
```

c) Associació A(0..1) – (0..1)B amb classe associativa C

A més d'aplicar la solució general, caldrà una clau UNIQUE a C per evitar que existeixin files de B que es relacionin amb més d'una fila d'A a través de C i una altra clau UNIQUE per evitar que existeixin files d'A que es relacionin amb més d'una fila d'A a través de C.

```
CREATE TABLE C(
atrib_c_1 DATE ,
A_atribut_a_1 INTEGER(4),
B_atribut_b_1 VARCHAR(5),
B_atribut_b_2 INTEGER(7),
PRIMARY KEY(A_atribut_a_1, B_atribut_b_1, B_atribut_b_2),
FOREIGN KEY(A_atribut_a_1)
REFERENCES A(atribut_a_1),
FOREIGN KEY(B_atribut_b_1, B_atribut_b_2)
REFERENCES B(atribut_b_1, atribut_b_2),
UNIQUE(A_atribut_a_1),
UNIQUE(B_atribut_b_1, B_atribut_b_2)
);
```

d) Associació A(1) – (1)B amb classe associativa C

A més d'aplicar la solució general, caldrà una clau UNIQUE a C per evitar que existeixin files d'A que es relacionin amb més d'una fila de B a través de C. Caldrà una altra clau UNIQUE a C per evitar que existeixin files de B que es relacionin amb més d'una fila d'A a través de C. Caldrà un trigger a la taula C per evitar que existeixin files de B que no es relacionin amb cap fila d'A a través de C i un altre per evitar que existeixin files d'A que no es relacionin amb cap fila de B a través de C.

```
CREATE TABLE C(
atrib_c_1 DATE ,
A_atribut_a_1 INTEGER(4),
B_atribut_b_1 VARCHAR(5),
```

```
B_atribut_b_2 INTEGER(7),
PRIMARY KEY(A_atribut_a_1, B_atribut_b_1, B_atribut_b_2),
FOREIGN KEY(A_atribut_a_1)
REFERENCES A(atribut_a_1),
FOREIGN KEY(B_atribut_b_1, B_atribut_b_2)
REFERENCES B(atribut_b_1, atribut_b_2),
UNIQUE(A_atribut_a_1)
UNIQUE(B_atribut_b_1, B_atribut_b_2) ,
);

CREATE TRIGGER C_1
AFTER DELETE ON C
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM C
 WHERE B_atrib_b2 = :old.B_atrib_b2
    AND B_atrib_b1 = :old.B_atrib_b1;
 IF (numFiles < 1) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TRIGGER C_2
AFTER DELETE ON C
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM C
 WHERE A_atrib_a1 = :old.A_atrib_a1;
 IF (numFiles < 1) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
 END IF;
END;
```

Variants de les cardinalitats bàsiques

S'identifiquen algunes variants de les cardinalitats bàsiques, rarament utilitzades i que expliquem de forma abreviada. N'hi ha moltes i es podrien posar molts casos, però segueixen sempre l'esquema general.

a) Associació A(k..m) – (n..o)B, on k i n són enters més grans que 1, i m i o són enters tals que m>k i o>n sense classe associativa

Cal un trigger a la taula R per evitar que hi hagi més de o files d'A que es relacionin amb una fila de B a través de R, i un altre trigger a la taula R per evitar que hi hagi més de m files de B que es relacionin amb una fila d'A a través de R. També cal afegir dos triggers més per controlar els límits inferiors: un a la taula R per evitar que hi hagin menys de k files de B que es relacionin amb una fila d'A a través de R, i un altre també a la taula R per evitar que hi hagi menys de n files d'A que es relacionin amb una fila de B a través de R.

b) Associació A(k..m) – (n..o)B, on k i n són enters més grans que 1, i m i o són enters tals que m>k i o>n amb classe associativa

Cal un trigger a la taula C per evitar que hi hagi més de o files d'A que es relacionin amb una fila de B a través de C, i un altre trigger a la taula C per evitar que hi hagi més de m files de B que es relacionin amb una fila d'A a través de C. També cal afegir dos triggers més per controlar els límits inferiors: un a la taula C per evitar que hi hagin menys de k files de B que es relacionin amb una fila d'A a través de C, i un altre també a la taula C per evitar que hi hagin menys de n files d'A que es relacionin amb una fila de B a través de C.

2.2.5. Mapeig d'agregacions i composicions

Per UML la diferència entre una associació "normal" i les agregacions i les composicions és una qüestió de grau, les darreres no són més que una associació amb més força. Les agregacions i les composicions les hem considerat com a simples associacions sense tenir en compte si el major acoblament entre classes que produeixen mereix un tractament diferenciat; això és una tasca que s'hauria de fer en un futur.

2.2.6. Mapeig de generalitzacions

L'herència podria mapejar-se de les següents maneres:

a) Una taula per cada subclasse que contingui també tots els atributs de la superclasse. Això vol dir duplicar els atributs de la classe mare a cada taula de les subclasses.

b) Una única taula per la superclasse que contingui tots els atributs de totes les subclasses.

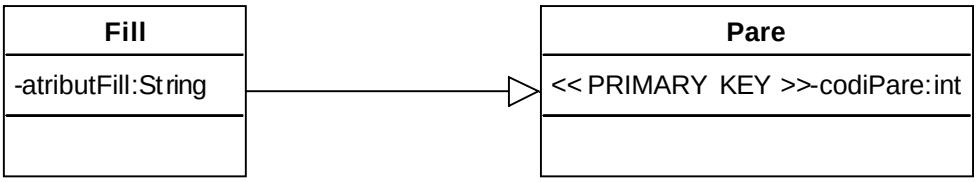
Això vol dir que hi haurà molts valors nuls (els que corresponguin a les subclasses diferents de la classe de la instància que estem considerant).

c) Una taula per la superclasse i una taula per cadascuna de les subclasses, totes amb els atributs propis de la classe a que corresponen.

Això s'implementa de la mateixa manera que una associació fill(0..1)-(1)pare per cada generalització, però amb la particularitat que la taula fill agafa la clau primària de la taula que correspon a la classe pare, contra la que també forma una clau forana. És la solució més econòmica en espai i més mantenible, de manera que serà la solució adoptada.

L'herència s'implementa sense restriccions, ja que l'eina Poseidon no les deixa posar de forma explícita. La primera és *disjoint* (una instància de la superclasse no pot ser-ho de dues subclasses alhora) / *overlapping* (el contrari); la segona és *complete* (una instància de la superclasse ha de ser instància com a mínim d'una de les subclasses) / *incomplete* (el contrari).

Exemple:




```
CREATE TABLE Pare(
codiPare INTEGER(5),
PRIMARY KEY(codiPare)
);

CREATE TABLE Fill(
Pare_codiPare INTEGER(5),
atributFill INTEGER(5),
PRIMARY KEY(Pare_codiPare),
FOREIGN KEY(Pare_codiPare)
REFERENCES Pare(codiPare)
);
```

3. Estudi de la tecnologia més adequada per generar codi DDL/SQL

S'ha utilitzat el llenguatge de programació Java conjuntament amb l'eina ant per compilar (des de la línia de comandes). L'editor usat per editar els fonts en Java ha estat el Jext, que destaca per la seva comoditat. El fitxer generat es diu *classesumlataules.jar*. S'ha obtingut de l'empresa propietària de Poseidon (Gentleware) la següent key per poder utilitzar aquest nom, en la versió 0.1:

MmqVk9yxZ7+jHm1UYgJVJG5jkAkDiv17pUpRxh7TbfXMUMCq8lKa5DVMO8QjLzf0
+R9wlTDHEvTEn3FhD23JMH/zCgjIwePBCp3czCNrYjCQXTot93ukKQhWD+72YBSv
FJ07cNtCNn7LAd00d0a53jtndAytmV7xdRRrJsYqWwY

Aquesta key no té data de caducitat. De tota manera, no s'ha aconseguit que funcioni sinó és dins de l'embolcall de l'aplicació exemple *apidemo*, de manera que es mantenen alguns noms relacionats amb aquesta aplicació.

Com el que pretenïem era traduir d'un model plasmat en un diagrama a un altre model plasmat en text, el més natural ha semblat seguir els següents passos:

1) Accedir al model plasmat en diagrames
2) Crear un model basat en classes on estigui plasmada la informació rellevant del model basat en diagrames
3) Transformar el model basat en classes en un model basat en taules preparat adequadament per plasmar-se en codi
4) Transformar el model basat en taules en codi

Tot l'anterior s'ha fet amb una interfície d'usuari adequada.

3.1. Accedir al model plasmat en diagrames

PRIMERA OPCIÓ: XMI

UML defineix un format estàndard d'intercanvi per models UML que s'anomena XMI, de la família d'XML. De fet, Poseidon guarda els projectes en XMI comprimit juntament amb informació gràfica i altres informacions del projecte. Es tracta del format *zuml*. Es pot obtenir el fitxer XMI descomprimint el fitxer d'extensió zuml (amb l'eina unzip). També es pot exportar el projecte al format XMI amb File\Export Project to XMI.
Si es té o es fabrica un generador de codi extern (possiblement sense interfície gràfica) que treballa amb entrada amb format XMI, és una bona idea usar aquesta opció ja que pot servir per diverses eines CASE que són capaces de generar XMI.
De tota manera, se sap que no totes les eines implementen aquest format de forma suficientment estàndard, i l'intercanvi pot no funcionar amb algunes d'elles.

SEGONA OPCIÓ: API DE POSEIDON

L'eina Poseidon té un API que permet accedir fàcilment al model, si bé està documentat de forma irregular i part de la documentació només és accessible on-line. Hi ha documentació amb format UML i altra en format Javadoc. Cal tenir en compte que els *projectes* s'organitzen en un únic *model*, malgrat que això es planteja canviar-ho en el futur. Un model pot contenir uns quants diagrames (màxim de 8 en la versió

d'avaluació de Poseidon utilitzada), però no és possible accedir als elements presents en un diagrama concret, s'accedeix als elements de tot el model.

DISCUSSIÓ SOBRE LES OPCIONS

Hi ha dues opcions molt clares en aquest PFC per obtenir la informació del model. La primera seria que l'usuari exportés el projecte a XMI i seguidament llencés el generador de codi extern (segurament des de la línia de comandes) per obtenir el codi desitjat. La segona seria que integrat a l'eina Poseidon hi hagués un plug-in que accedís a la informació del model mitjançant l'API de Poseidon i ell mateix generés el codi. L'opció triada és la darrera. El seu principal avantatge és la integració total amb l'eina i la seva facilitat d'ús. En principi, només pot generar codi per l'eina Poseidon, si bé es pot estructurar perquè canviant-ne certes parts pugui servir per altres eines. Es podria pensar també que el format XMI es presta (per la seva pròpia estructura) a poder usar analitzadors lèxics i sintàctics per facilitar la implementació del generador i això és cert; de tota manera, la generació d'scripts SQL per crear taules és senzilla i presenta poca dificultat de programació directa en Java.

L'API DE POSEIDON: L'ACCÉS AL MODEL

L'API de Poseidon és prou completa, només destacarem el que és d'interès per aquest PFC.

- Classe *PoseidonProjectConnector*

És una classe per manejar els projectes. La gran majoria dels seus mètodes són estàtics. Cada projecte té associat un nom de fitxer. Un projecte conté informació del model, la informació gràfica dels diagrames i informació relativa al fitxer que té associat. Actualment, l'únic projecte suportat és el que conté informació UML.
Un projecte conté informació sobre un model (sobre més d'un en el futur).

D'aquesta classe ens poden interessar els següents mètodes:

- public static *getCurrentProject*() : PoseidonProject
  retorna el projecte obert actualment a l'eina Poseidon (només es permet tenir-ne un d'obert en un moment determinat).

- public static *getFilename*() : String
  retorna el nom del fitxer associat al projecte actual, incluint l'extensió.

- public static *getModel*() : Object
  retorna el primer (actualment l'únic) model UML del projecte actual. La classe retornada realment és Model. No funciona com diu la documentació, perquè ho faci cal passar-li el número de model, així: *getModel*(0).

- Interfície *Model*

Interfície per tractar el model. D'aquesta interfície ens pot interessar el següent mètode:

- public *getOwnedElement*(): Collection
  retorna tots els elements del model.

Els elements de la darrera Collection poden ser de diverses classes o interfícies. Nosaltres ens interessarem per les interfícies *AssociationClass*, *UmlClass* i *UmlAssociation*. La segona representa una classe UML, la tercera una associació UML i

la primera és una classe associativa, que no és altra cosa que la fusió de les altres dues classes anteriors.

- Interfície *UmlClass*

Representa les classes UML. D'aquesta interfície ens poden interessar els següents mètodes:

- public *getName*(): String
  retorna el nom de la classe

- public *getFeature*(): List
  retorna una llista amb els elements estructurals de la classe (interfície *Feature*). De la interfície *Feature*, només ens interessaran els atributs (*Attribute*), no prestarem atenció als mètodes, operacions, etc. Això és perquè en un diagrama de classes d'anàlisi no hi ha d'haver ni operacions ni mètodes.

- public *getStereotype()*: Collection
  retorna una llista amb els estereotips de la classe, que són de la interfície *Stereotype*.

- public *getTaggedValue*(): Collection
  retorna una llista amb els valors etiquetats de la classe, que són de la interfície *TaggedValue*.

- Interfície *Attribute*

Representa els atributs d'una classe. D'aquesta interfície ens poden interessar els següents mètodes:

- public *getName*(): String
  retorna el nom de l'atribut

- public *getStereotype*: Collection
  retorna una llista amb els estereotips de l'atribut, que són de la interfície *Stereotype*.

- public *getTaggedValue*(): Collection
  retorna una llista amb els valors etiquetats de l'atribut, que són de la interfície *TaggedValue*.

Per obtenir el tipus de l'atribut s'han de fer servir les interfícies *Classifier* i *StructuralFeature*.

- Interfície *Classifier*

Interfície pare de tots els classificadors, incloses *UmlClass*, *AssociationClass*, *DataType*, etc. Ens interessa perquè és el tipus que retorna el mètode per obtenir el tipus de dades d'un atribut.

- Interfície *StructuralFeature*

Pare directe de la interfície *Attribute*. D'aquesta interfície ens pot interessar el següent mètode:

- public *getType*(): Classifier
  retorna el tipus, particularment d'un *Attribute*.

- Interfície *Stereotype*

Representa els estereotips associats a classes o atributs. D'aquesta interfície ens pot interessar els següent mètode:

- public *getName*(): String
  retorna el nom de l'estereotip.

- Interfície *TaggedValue*

Representa els valors etiquetats associats a classes o atributs. D'aquesta interfície ens poden interessar els següents mètodes:

- public *getType()*: TagDefinition
  retorna un *TagDefinition*, que representa el nom o clau del valor etiquetat.

- public *getDataValue*(): Collection
  retorna una llista d'Strings amb els valors del valor etiquetat. En aquest PFC considerarem que només en pot tenir un (encara que l'estàndard UML permet que en tingui més), ja que no farem servir valors etiquetats multivalor.

- Interfície *TagDefinition*: representa la definició d'un valor etiquetat. D'aquesta interfície ens pot interessar els següent mètode:

- public *getName*(): String
  retorna el nom del valor etiquetat.

- Interfície *UmlAssociation*

Representa les associacions UML. D'aquesta interfície ens poden interessar els següents mètodes:

- public *getName*(): String
  retorna el nom de l'associació. No la farem servir perquè hi ha associacions a les quals l'usuari no assigna cap nom, i n'hi poden haver també amb el nom repetit.

- public *getStereotype()*: Collection
  retorna una llista amb els estereotips de l'associació, que són de la interfície *Stereotype*.

- public *getTaggedValue*(): Collection
  retorna una llista amb els valors etiquetats de l'associació, que són de la interfície *TaggedValue*.

- public *getConnection*(): List
  retorna una llista dels elements que estan connectats per l'associació, que són de la interfície *AssociationEnd*. Com que només tractem associacions binàries, ignorarem qualsevol participant que no siguin els dos primers.

- Interfície *AssociationEnd*

Representa els extrems de les associacions UML, amb la informació associada. D'aquesta interfície ens poden interessar els següents mètodes:

- public Classifier *getParticipant*():
  retorna el valor del participant, que és un classificador, en el nostre cas una classe.

- public *getName*(): String
  retorna el rol que fa l'extrem de l'associació dins de l'associació.

- public *getMultiplicity*(): Multiplicity
  retorna la multiplicitat (cardinalitat) en aquell extrem de l'associació.

- Interfície *Multiplicity*

Representa la multiplicitat (cardinalitat) en aquell extrem de l'associació. D'aquesta interfície ens pot interessar el següent mètode:

- public *getRange*(): Collection
  retorna una llista del rangs de la *Multiplicity*, que són de la interfície *MultiplicityRange*.

- Interfície *MultiplicityRange*

Representa cardinalitat inferior i superior en aquell extrem de l'associació. D'aquesta interfície ens poden interessar els següents mètodes:

- public *getLower*(): int
  retorna el valor inferior de la cardinalitat d'un extrem de l'associació.

- public *getUpper*(): int
  retorna el valor superior de la cardinalitat d'un extrem de l'associació.

- Interfície *AssociationClass*

Representa les classes associatives UML. Hereta de les dues interfícies anteriors com era d'esperar: una classe associativa no és més que una associació amb propietats de classe o bé una classe amb propietats d'associació. Les tractarem com una classe i com una associació (dos elements separats però relacionats).

- Interfície *Generalization*

Representa les relacions d'herència. D'aquesta interfície ens poden interessar els següents mètodes:

- public *getChild*(): GeneralizableElement
  retorna el fill d'una relació d'herència

- public *getParent*(): GeneralizableElement
  retorna el pare d'una relació d'herència

Els *GeneralizableElement* només es tindran en compte si són *UmlClass* tots dos, sinó seran ignorats.

3.2. Crear un model basat en classes on estigui plasmada la informació rellevant del model basat en diagrames

De tota la informació anterior podrem extreure tot el necessari per construir el model de text prèvia transformació en un model de taules.

Com veiem hi ha dos móns que estan separats però que es relacionen. D'una banda tenim les classes, amb el seus atributs, generalitzacions i també amb els estereotips i valors etiquetats. De l'altra, les associacions, amb els seus participants i les seves cardinalitats i les classes que associen. El següent diagrama de classes d'anàlisi representa un possible model basat en classes que conté la informació rellevant recollida del model. Cal tenir en compte que els diagrames s'han fet amb la mateixa

eina Poseidon, que té la particularitat que les multiplicitats 1 als extrems de les associacions simplement no les escriu, per tant si no es visualitza cap cardinalitat s'assumeix que és 1.

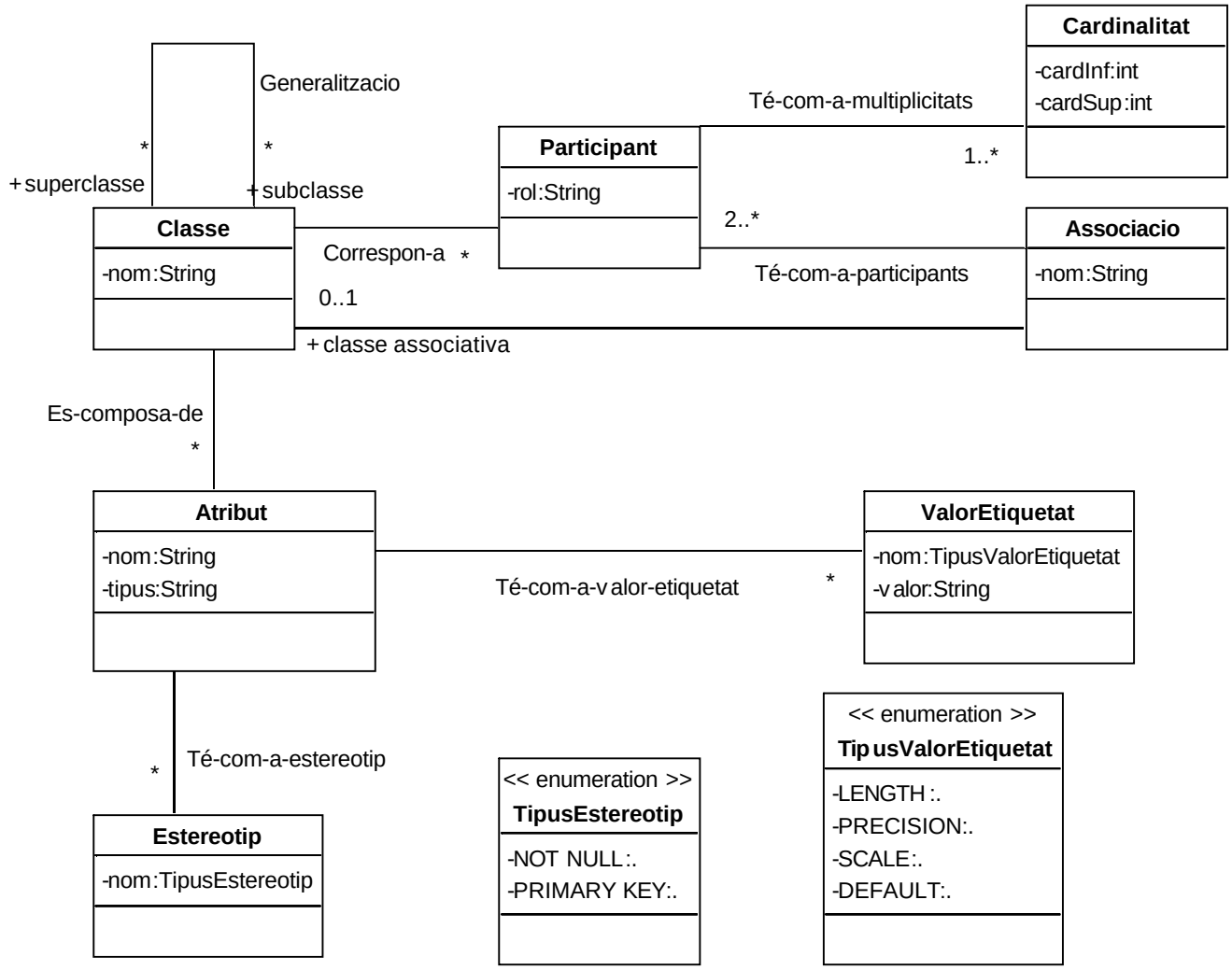


Fig. 3: Diagrama de classes d'anàlisi del model basat en classes

3.3. Transformar el model basat en classes en un model basat en taules preparat adequadament per plasmar-se en codi

Les transformacions que cal fer serien:

⇨ **Transformar classes en taules**
No ofereix cap complicació, una classe es transforma en una taula. Els estereotips de nivell de classe s'ignoraran ja que no aporten informació rellevant per les taules tal com hem plantejat aquest PFC, igualment que els valors etiquetats. Totes les classes, independentment del seu tipus, es consideren persistents.

⇨ **Transformar atributs en columnes i tipus en tipus de dades**

La transformació d'atributs en columnes és trivial. Es consideren tots com a persistents. Per transformar de dades a tipus de dades farem servir la taula de conversió que s'ha explicitat a l'apartat *2. Com traduir del diagrama de classes d'anàlisi d'UML a taules SQL*. Els tipus que s'apartin d'aquesta relació no es podran traduir; el que es farà es escriure'ls directament sense traduir al codi SQL generat, incloent, a més, un comentari per fer evident aquest fet.

Farem servir els següents valors etiquetats per modificar el tipus de dades bàsic de les columnes

Valors etiquetats d'atribut	**Significat a les columnes**
length	Longitud dels tipus de dades alfabètics
precision	Longitud total dels tipus de dades numèrics
scale	Longitud de la part decimal dels tipus de dades numèrics

Fig. 4: Valors etiquetats usats per modificar el tipus de dades de les columnes

Farem servir els següent valor etiquetat per modificar el valor per defecte de les columnes

Valors etiquetats d'atribut	**Significat a les columnes**
default	Valor per defecte de l'atribut

Fig. 5: Valors etiquetats usats per modificar el valor de les columnes

Farem servir els següents estereotips per modificar restriccions de columna

Estereotips d'atribut	**Significat a les columnes**
NOT NULL	NOT NULL
PRIMARY KEY	El camp forma part de la clau primària

Fig. 6: Estereotips usats com a restriccions de columna

⇨ **Transformar associacions en claus foranes, taules de relació, disparadors o claus úniques**

No farem servir estereotips ni valors etiquetats per modificar el caràcter de les associacions.

Algunes relacions acabaran generant claus foranes, creant taules de relació o bé claus úniques o disparadors. Cal veure el subapartat *Mapeig d'associacions sobre relacions* dins de l'apartat *2. Com traduir del diagrama de classes d'anàlisi d'UML a taules SQL* per veure el tipus de transformacions que es poden realitzar.

⇨ **Transformar l'herència**

Les relacions d'herència s'implementaran de la mateixa manera que les associacions fill(0..1) – (1)pare, però amb la particularitat que la taula corresponent a la classe fill agafa la clau primària de la taula corresponent a la classe pare. És la manera més eficient de tractar-la, si bé es podria fer una taula per cada subclasse que inclogués tots els atributs de la superclasse, o bé tenir una sola taula amb tota la informació de les subclasses i la de la superclasse.

El nou model de classes d'anàlisi seria el següent

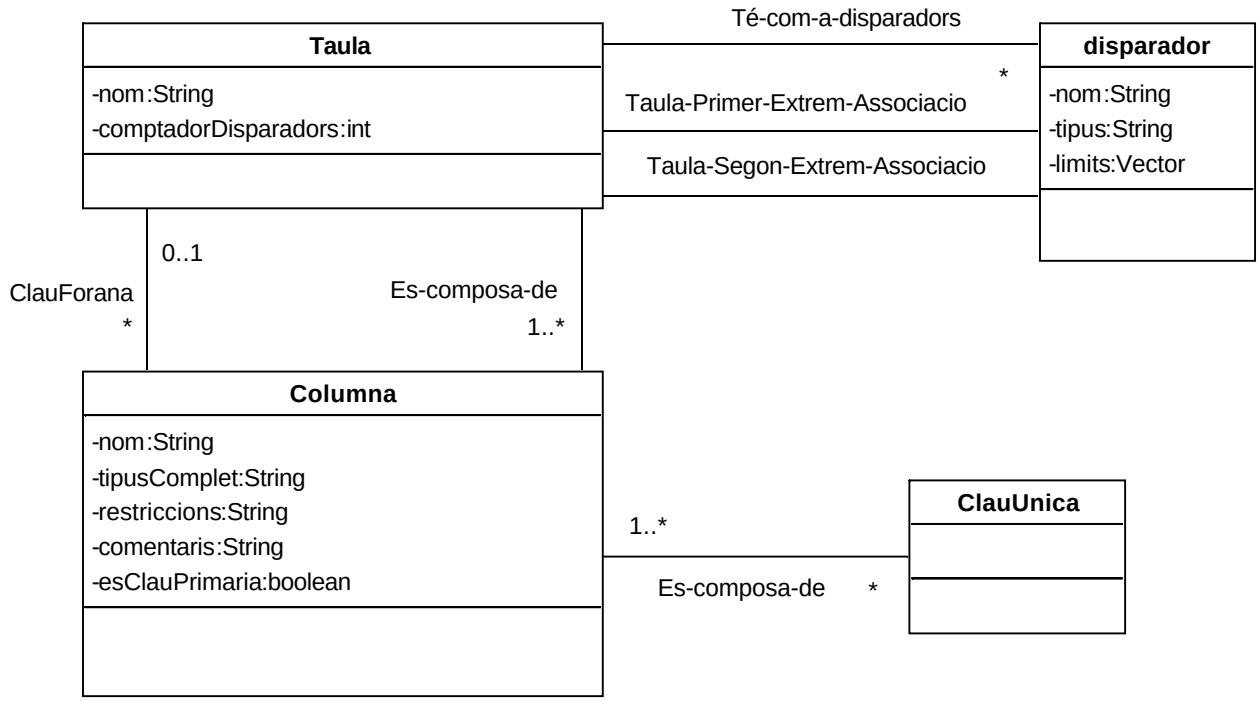


Fig. 7: Diagrama de classes d'anàlisi del model basat en taules

S'observa que tota la informació és ara en Taules i classes associades.
Les classes Taula, Columna, Disparador i ClauUnica representen el mateix concepte que en el món de les bases de dades relacionals. Els disparadors es creen com a conseqüència d'una associació que té dues classes als extrems que acaben generant dues taules (Taula-Primer-Extrem-Associacio i Taula-Segon-Extrem-Associacio); cal conèixer aquestes, ja que la segona servirà per construir la join dins la condició WHERE del disparador i la primera aclarirà si ens trobem davant d'una relació reflexiva. Si hi ha taula-relació, el disparador s'aplicarà a aquesta i no a cap taula dels extrems de l'associació.

3.4. Transformar el model basat en taules en codi

El model que hem vist a l'apartat anterior es pot ja plasmar en text d'una manera senzilla, tenint en compte que ja conté, en un format adequat, la informació necessària per a ser convertida en text.

4. Investigació i explicació de l'extensió de l'eina

Poseidon és una eina de modelatge amb el llenguatge UML, derivada d'Argo UML, que és una versió de codi obert de gran popularitat per modelar. A diferència d'aquest, pretèn dirigir-se a un mercat professional, de manera que està suportat per l'empresa Gentleware. Una de les seves fites és que sigui el màxim d'usable i que modelar sigui el més senzill possible, de manera que és natural que permeti extensions.

Poseidon permet l'extensió de les seves funcionalitats mitjançant plug-ins, encara que no en totes les seves edicions. L'edició que s'ha fet servir (la Professional 2.5.1.) porta incorporats alguns plug-ins i, el que és més important, permet als usuaris de l'eina d'afegir-ne de nous.

Seguidament explicarem com construir un plug-in.

Requeriments de maquinari i programari

Cal usar el *Java Development Kit (JDK)* en la seva versió 1.4 com a mínim. Poseidon simplement no pot treballar amb versions anteriors del JDK. Qualsevol editor de textos serveix per editar el codi font en Java. Per compilar farem servir l'eina *ant*, ja que simplifica notablement el procés (si bé no és estrictament necessari).

És fonamental afegir a la variable d'entorn CLASSPATH el subdirectori lib del directori on s'hagi instal·lat l'eina Poseidon.

No cal fer servir cap sistema operatiu en particular, pot ser Windows (98 o superior), Linux o UNIX.

Poseidon és una eina força exigent de recursos. L'ordinador necessita almenys 256 MB de memòria RAM per executar-la de manera "suau". S'ha provat amb una màquina amb 128 MB de RAM i 550MHz de velocitat de processador, i l'eina s'executava amb una lentitud desesperant.

Compilar: l'eina ant

L'eina *ant* està basada en Java. És una eina per compilar codi, similar a *make* però superant les limitacions d'aquest. En comptes d'un model basat en shell-scripts (massa dependent dels sistemes operatius en particular), ant fa servir classes Java. Enlloc d'escriure comandes de shell, els fitxers de configuració estan basats en XML (evitant problemes com els tabuladors als makefiles). Proporciona independència del sistema operatiu en particular on s'executa.

Perquè ant funcioni, és recomenable definir la variable d'entorn *ANT_HOME* i afegir el seu subdirectori *\bin* a la variable d'entorn PATH.

Per compilar cal escriure prèviament un fitxer XML anomenat precisament *build.xml*, que té un contingut especial. En aquest fitxer cal definir un tag *project* amb l'atribut *default* que conté el *target* que es farà servir per defecte, i opcionalment un *name* (nom) i un *basedir* (directori a partir del qual es calculen tots els paths).
Cada projecte té un o més targets, els quals són un joc de tasques que es vol executar. Es pot triar quan s'executa ant quin target es vol executar, si no es fa s'executarà el *default* del project.
Un *target* pot dependre d'altres targets. N'hi hauria d'haver un per compilar i un altre per crear una distribució. La distribució depèn de la compilació prèvia, ant resol aquestes dependències. Els targets tenen l'atribut *depends*, el qual especifica l'ordre en

què els targets s'executaran. Els atributs *if* i *unless* permeten condicionar la seva execució. Pot incloure també un atribut *description*.
Sol existir un target *init* del qual depenen els altres targets, amb tasques d'inicialització.
Les *tasques* són trossos de codi que es poden executar. Poden tenir molts atributs (que representen arguments). Hi ha algunes tasques que estan predefinides. Poden tenir un identificador (atribut *id*), i sempre tenen un *name* (nom).
Els projectes poden tenir un conjunt de *propietats*. Les que proporcionen accés a propietats dels sistema estan predefinides. Es poden declarar propietats fora de qualsevol target.

Aquí s'inclou un fitxer build.xml d'exemple.

```
<project name="MyProject" default="dist" basedir=".">
  <description>
    simple example build file
  </description>
 <!-- set global properties for this build -->
 <property name="src" location="src"/>
 <property name="build" location="build"/>
 <property name="dist"  location="dist"/>

 <target name="init">
  <!-- Create the time stamp -->
  <tstamp/>
  <!-- Create the build directory structure used by compile -->
  <mkdir dir="${build}"/>
 </target>

 <target name="compile" depends="init"
     description="compile the source " >
  <!-- Compile the java code from ${src} into ${build} -->
  <javac srcdir="${src}" destdir="${build}"/>
 </target>

 <target name="dist" depends="compile"
     description="generate the distribution" >
  <!-- Create the distribution directory -->
  <mkdir dir="${dist}/lib"/>

  <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
  <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
 </target>

 <target name="clean"
     description="clean up" >
  <!-- Delete the ${build} and ${dist} directory trees -->
  <delete dir="${build}"/>
  <delete dir="${dist}"/>
 </target>
</project>
```

Existeixen altres elements com token filters, path-like structures, command-line arguments i references que fan encara més rica aquesta completa eina.

Un cop haguem compilat el projecte tindrem un fitxer comprimit Java (jar), que serà el que haurem d'incloure a l'eina Poseidon.

Instal·lar

Primerament cal copiar el fitxer jar generat al subdirectori lib del directori d'instal·lació de Poseidon a l'ordinador. Després entrem a l'eina Poseidon i ens situem a Help/License Manager. Allà escrivim o enganxem la clau, l'afegim amb *Add* i tanquem seguidament el License Manager.
Anem a Plug-Ins/Plug-Ins panel, premem *Add*, busquem el fitxer jar que volem instal·lar (que acabem de deixar al subdirectori lib del directori d'instal·lació de Poseidon) i premem *Install*. Si tot ha anat bé, el plug-in sortirà a la llista de plug-ins instal·lats. Tanquem la finestra amb *Close* i ja tenim el plug-in instal·lat.

Si posteriorment es vol desinstal·lar el pluggin cal anar igualment a Plug-Ins/Plug-Ins panel i llavors prémer *Remove* i *Close*. S'hauran d'eliminar manualment els fitxers que s'afegiren durant el procés d'instal·lació. Finalment, cal anar al License Manager i eliminar la License Key.

5. Desenvolupament del plug-in UML a SQL

5.1. Anàlisi

Els diagrames de classes vistos anteriorment formarien part de l'anàlisi.
Apart, només d'identifica un cas d'ús, que s'exposa seguidament.

5.1.1. Diagrames de casos d'ús

Cas d'ús: "Traduir un model de diagrames de classes d'anàlisi UML a codi SQL per generar taules "

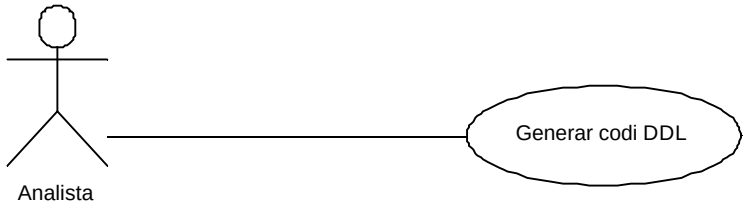


Fig. 8: Cas d'ús traduir un model de diagrames de classes d'anàlisi UML a codi SQL per generar taules

- *Resum de la funcionalitat*: Proporcionar codi SQL per generar taules a partir de diagrames de classes UML, tot integrat dins l'eina Poseidon.
- *Paper dins el treball de l'usuari*: Obtenir d'una manera ràpida l'esquelet de la base de dades associada a l'aplicació que s'està construint.
- *Actors*: Analista
- *Casos d'ús relacionats*: Cap (dins l'àmbit d'aquest PFC)
- *Precondició*: El sistema pot accedir a un projecte amb alguns (o fins i tot cap) diagrames de classes d'anàlisi facilitats per l'analista.
- *Postcondició*: L'analista té disponible el codi SQL necessari per generar les taules que corresponen als diagrames de classes d'anàlisi del projecte proporcionat al sistema.
- *Descripció detallada*: L'analista fa una sol·licitud per generar codi DDL/SQL. Seguidament, el sistema li mostra el codi SQL necessari per generar les taules que corresponen als diagrames de classes d'anàlisi del projecte proporcionat al sistema.

Curs normal dels esdeveniments

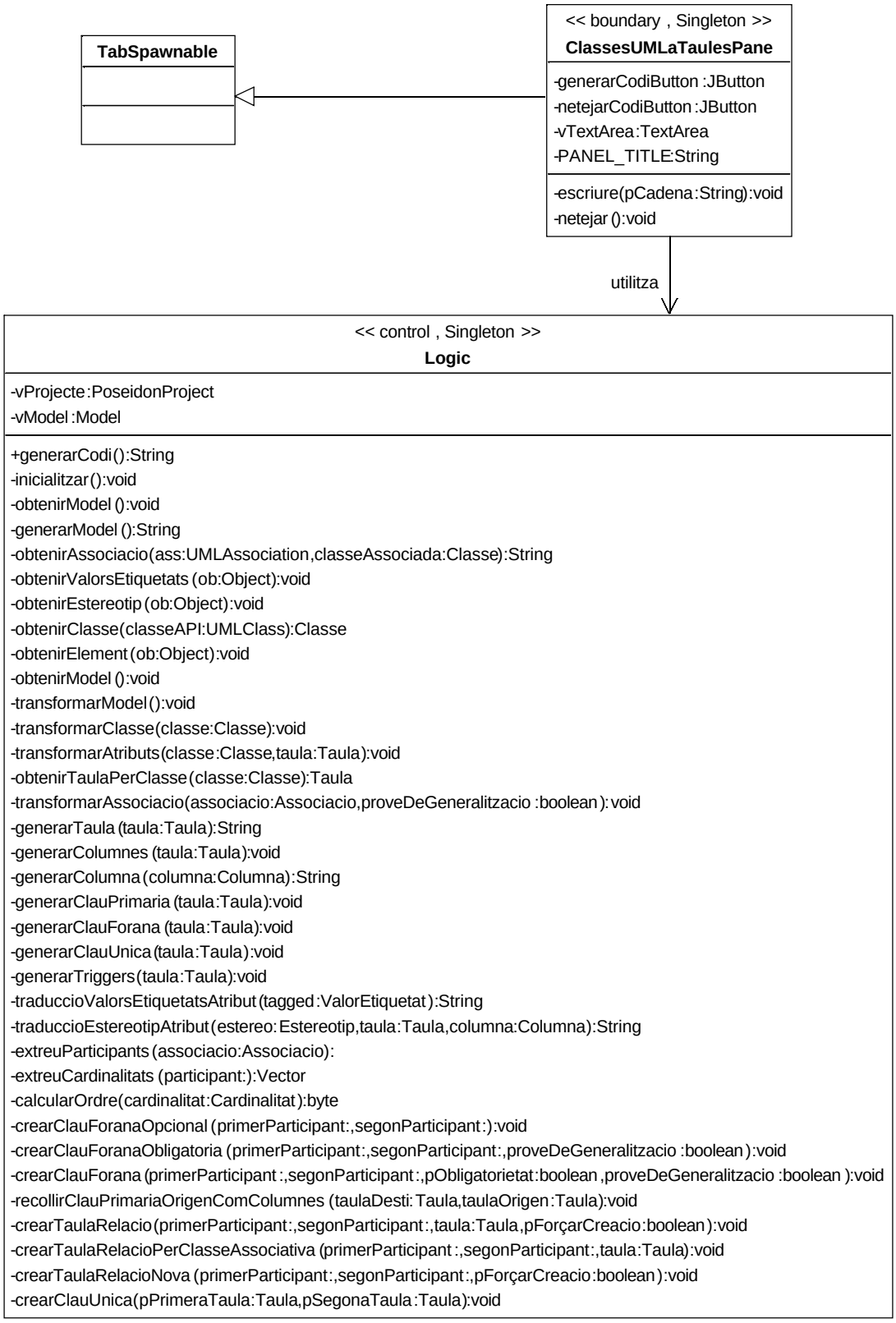
Acció de l'actor	*Acció del sistema*
1. L'analista fa una sol·licitud per generar codi DDL.	
	2. Mostra el codi SQL necessari per generar les taules que corresponen als diagrames de classes d'anàlisi del projecte proporcionat al sistema.

Curs alternatiu
No existeix. Si no hi ha informació (classes i associacions) per generar codi, simplement no es generarà res. No hi ha control d'errors, de manera que si la informació és incorrecta o incompleta el codi DDL/SQL generat no serà correcte.

5.2. Disseny

5.2.1. Diagrames de classes

5.2.1.1. Diagrama de classes principal



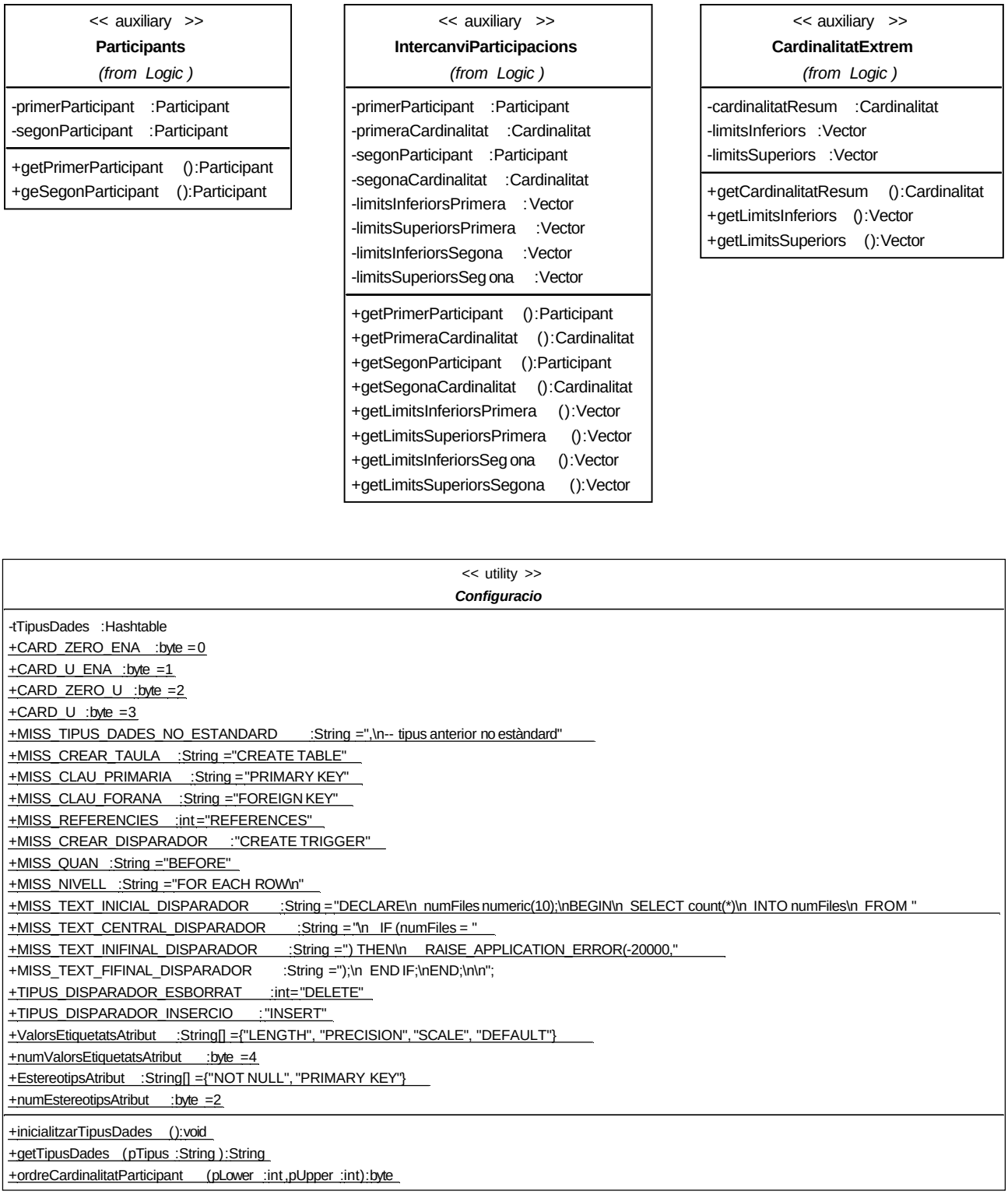


Fig. 9: Diagrama de classes de disseny principal

Observem que la classe ClassesUMLaTaulesPane és la classe de presentació identificada a l'anàlisi. Es tracta d'una pestanya i per això hereta de TabSpawnable. L'entrada que estimularà l'inici del cas d'ús és la pulsació d'un botó, la qual servirà per afegir el codi generar a una àrea de text. S'ha decidit en aquest punt afegir un botó per netejar la citada àrea de text.

La pulsació del botó esmentada abans crearà una instància de la classe de control Logic, que és la que contindrà la lògica de l'aplicació: obté la informació del model, la transforma i la retorna com a codi generat. Les classes Participants, IntercanviParticipacions i CardinalitatExtrem són auxiliars i es defineixen dins de la classe Logic. Les dues primeres serveixen per intercanviar participacions (de manera que l'extrem que tingui la cardinalitat més "petita" sigui el primer a tractar-se), i la tercera serveix per convertir les possibles cardinalitats múltiples d'un extrem d'una associació en una única cardinalitat.

La informació de suport de l'aplicació estarà continguda en la classe Configuració, la qual contindrà coses com missatges, constants i alguna informació bàsica com ara la manera de transformar tipus UML en tipus de dades SQL.

Aquesta és la descripció concreta de les classes més importants:

- *ClassesUMLaTaulesPane*: Classe de presentació per generar codi DDL/SQL a partir de diagrames de classes UML de l'eina Poseidon.
- *Logic*: Classe que conté la lògica de l'aplicació. Genera codi DDL/SQL a partir de diagrames de classes UML.
- *Configuracio*: Classe d'utilitat amb informació bàsica per l'aplicació, missatges, etc. Permet fer canvis ràpidament sobre característiques bàsiques de l'aplicació, canviar els textos dels missatges, etc.

5.2.1.2. Diagrama de classes del model de classes

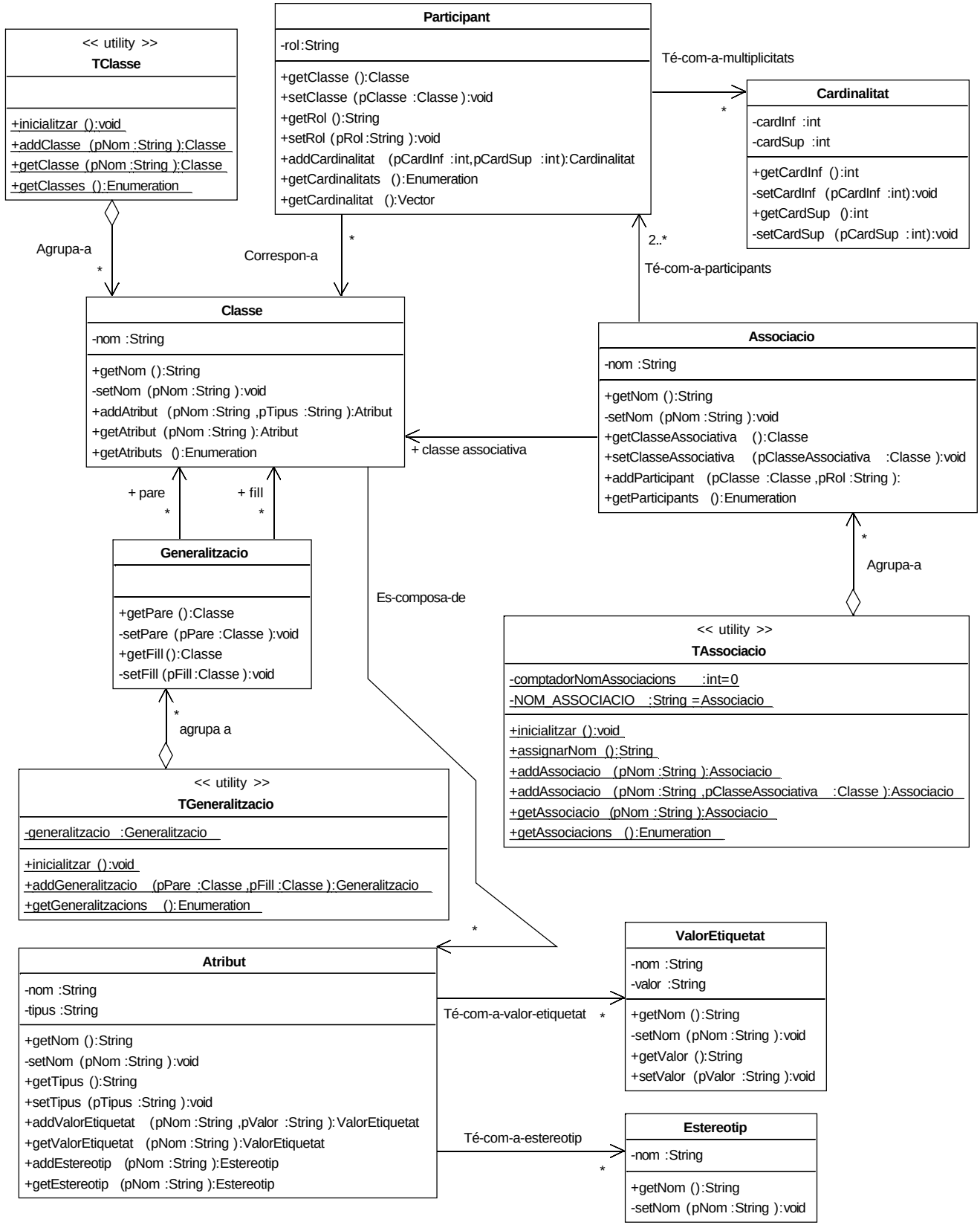


Fig. 10: Diagrama de classes de disseny del model de classes

Observem la presència de les classes TClasse i TAssociacio, les quals controlen la col·lecció de Classe i Associacio respectivament per cohesionar més el model i evitar atapeir encara més la classe de control Logic. Encara que només tractarem associacions binàries, es considera que la relació entre Participant i Associacio és 2..*.

Aquesta és la descripció concreta de les classes més importants:

- *Classe*: Classe que representa una classe UML.
- *TClasse*: Classe per manejar la col·lecció de Classe de l'aplicació.
- *Associacio*: Classe que representa una associació UML.
- *TAssociacio*: Classe per manejar la col·lecció d'Associacio de l'aplicació.
- *Generalitzacio:* Classe que representa una generalització, una relació d'herència.
- *TGeneralitzacio*: Classe per manejar la col·lecció de Generalitzacio de l'aplicació.
- *Participant*: Classe que representa un participant situat en un extrem d'una associació UML .
- *Cardinalitat*: Classe que representa una cardinalitat d'un extrem d'una associació UML.
- *Atribut*: Classe que representa un atribut d'una classe UML.
- *ValorEtiquetat*: Classe que representa un valor etiquetat UML.
- *Estereotip*: Classe que representa un estereotip UML .

5.2.1.3. Diagrama de classes del model de taules

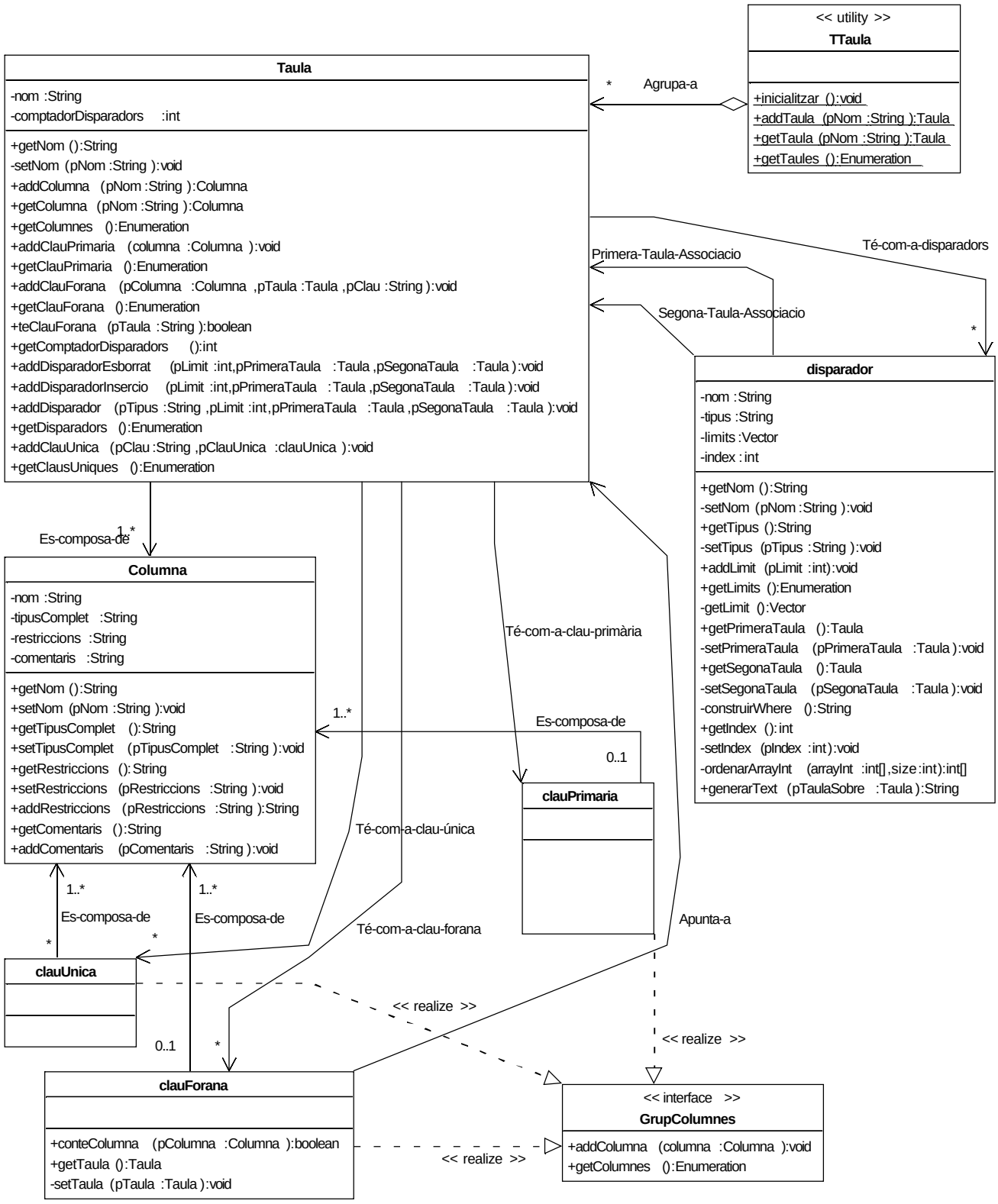


Fig. 11: Diagrama de classes de disseny del model de taules

Observem la presència de la classe TTaula, la qual controla la col·lecció de Taula per cohesionar més el model i evitar atapeir encara més la classe Logic.

Aquesta és la descripció concreta de les classes més importants:

- *Taula*: Classe que representa una taula.
- *TTaula*: Classe per manejar la col·lecció de Taula de l'aplicació.
- *Disparador*: Classe que representa un disparador d'una taula.
- *Columna*: Classe que representa una columna d'una taula.
- *GrupColumnes*: Interfície amb les operacions bàsiques dels grups de columnes.
- *ClauForana*: Classe que representa una clau forana d'una taula que va contra una altra taula.
- *ClauPrimaria*: Classe que representa una clau primària d'una taula.
- *ClauUnica*: Classe que representa una clau única d'una taula.

5.2.2. Diagrames de seqüència

S'han inclòs només els diagrames de seqüència que poden tenir un cert interès. Així, no s'ha inclòs inicialitzar() perquè només inicialitza trivialment diverses estructures de dades. Tampoc s'ha inclòs res relatiu a obtenirModel(), ja que senzillament es passa la informació de l'API de Poseidon al model de classes; la manera de fer-ho ja s'ha vist a l'apartat *L'API de Poseidon: l'accés al model* de la subsecció *3.1.Accedir al model plasmat en diagrames*. De la transformarModel() s'ha inclòs tot, ja que es considera la clau d'aquest PFC. De generarModel() no s'ha inclòs res, ja que a partir del model de taules la generació de codi és trivial (la informació està preparada per passar-se a text sense dificultats); la única part que té un cert interès és generarText() de la classe Disparador, però no s'inclou ja que es resol el problema internament sense pràcticament interacció amb altres classes.

5.2.2.1. Diagrama de sequència principal

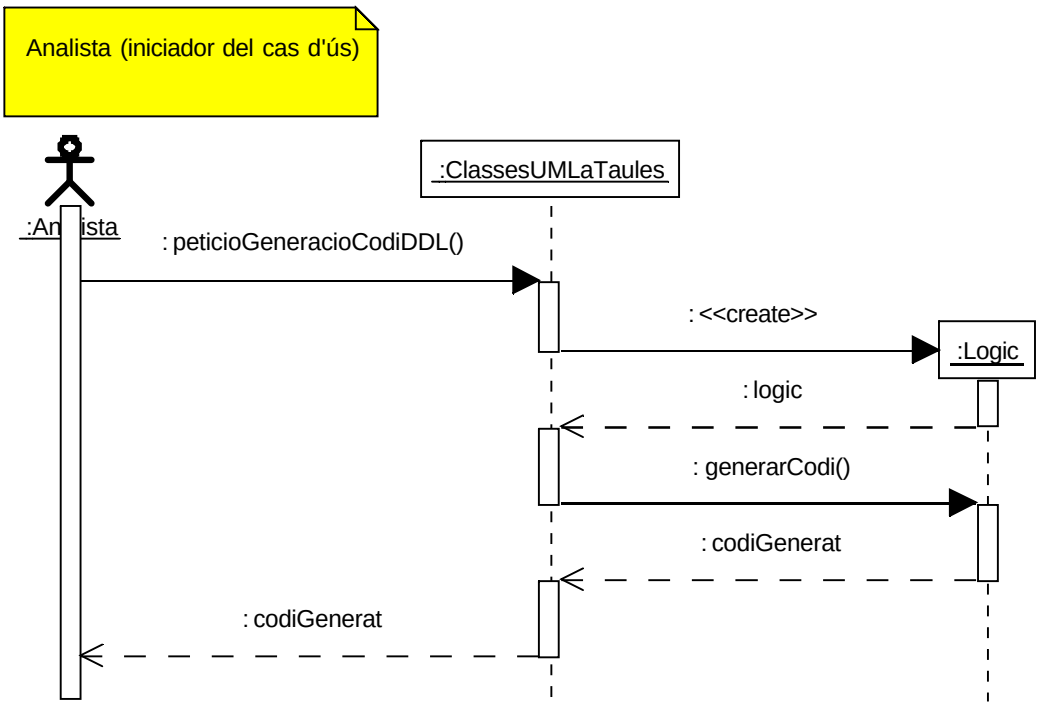


Fig. 12: Diagrama de seqüència principal

5.2.2.2. Diagrama de sequència de generarCodi()

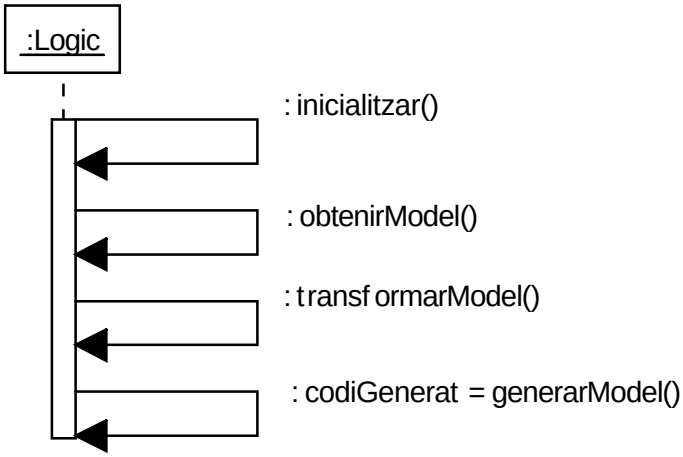


Fig. 13: Diagrama de seqüència de generarCodi()

5.2.2.3. Diagrama de sequència de transformarModel()

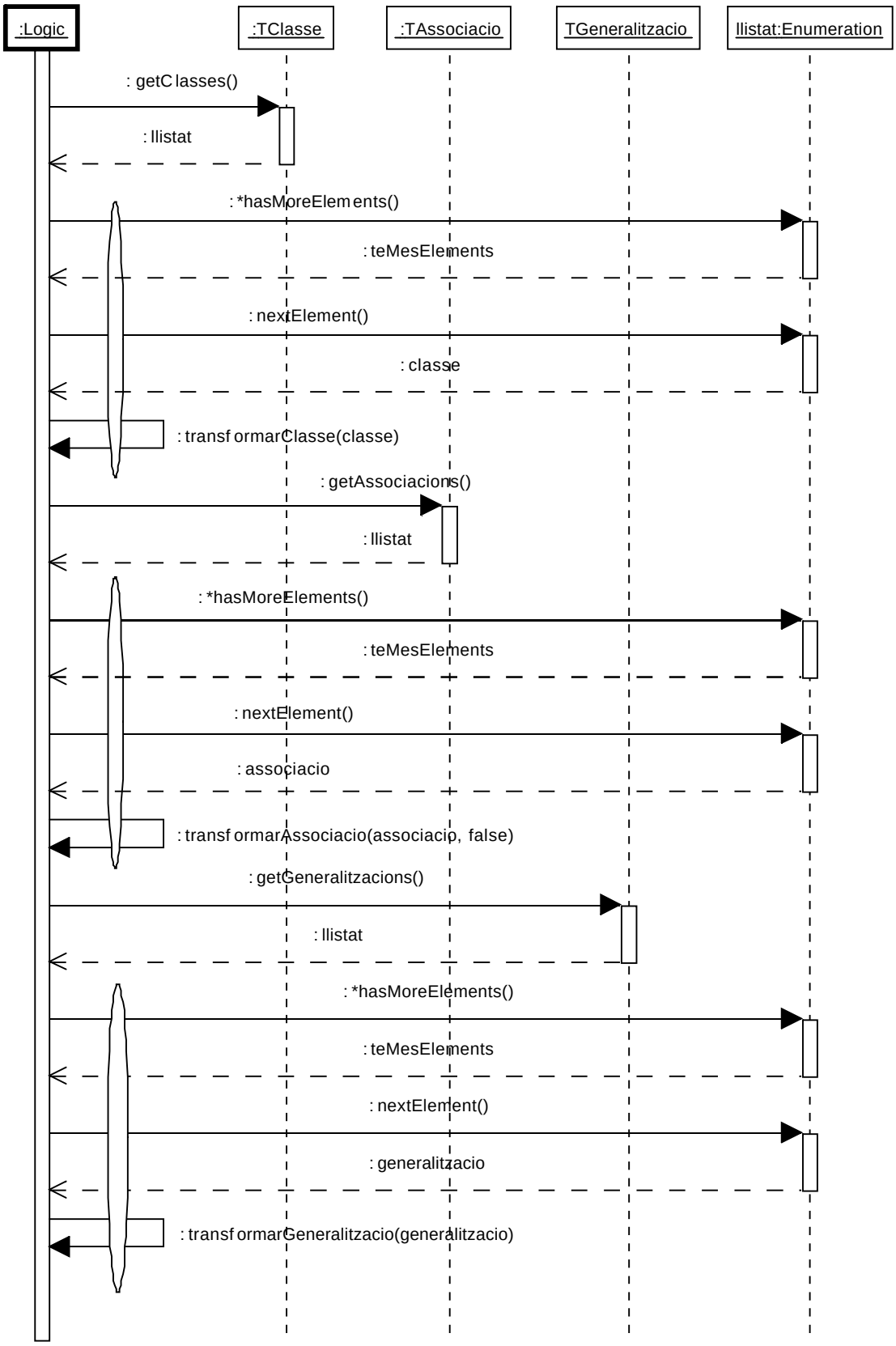


Fig. 14: Diagrama de seqüència de transformarModel()

5.2.2.4. Diagrama de sequència de transformarClasse()

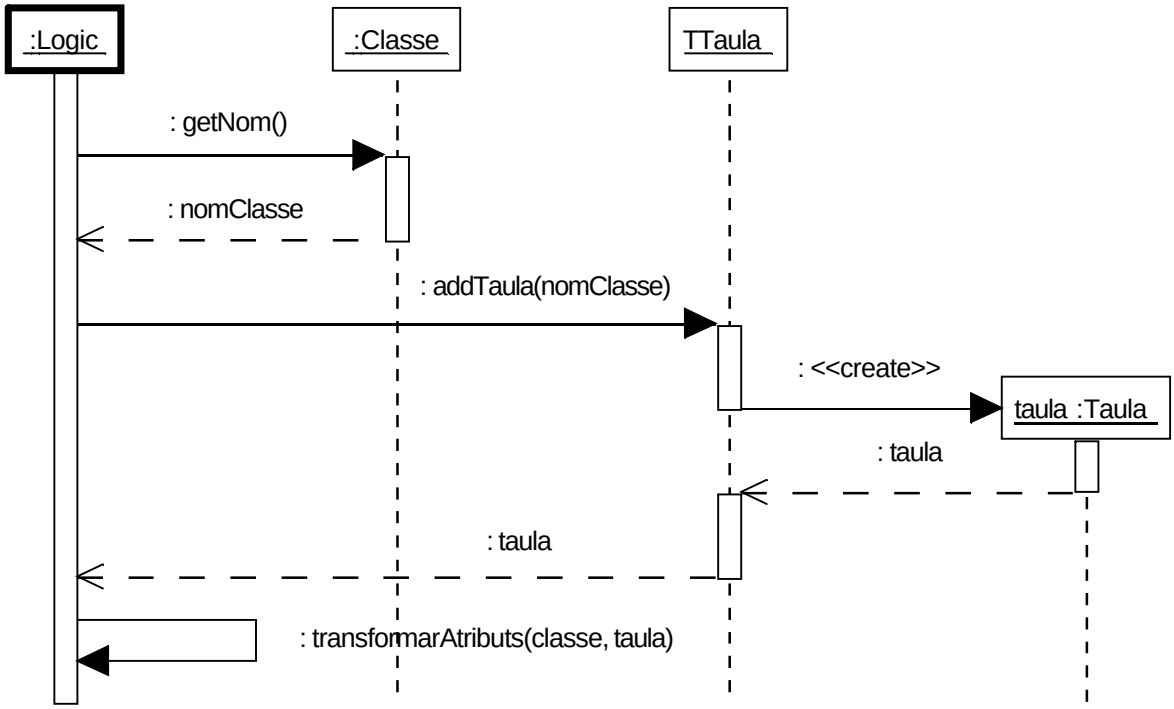


Fig. 15: Diagrama de seqüència de transformarClasse()

5.2.2.5. Diagrama de sequència de transformarAtributs()

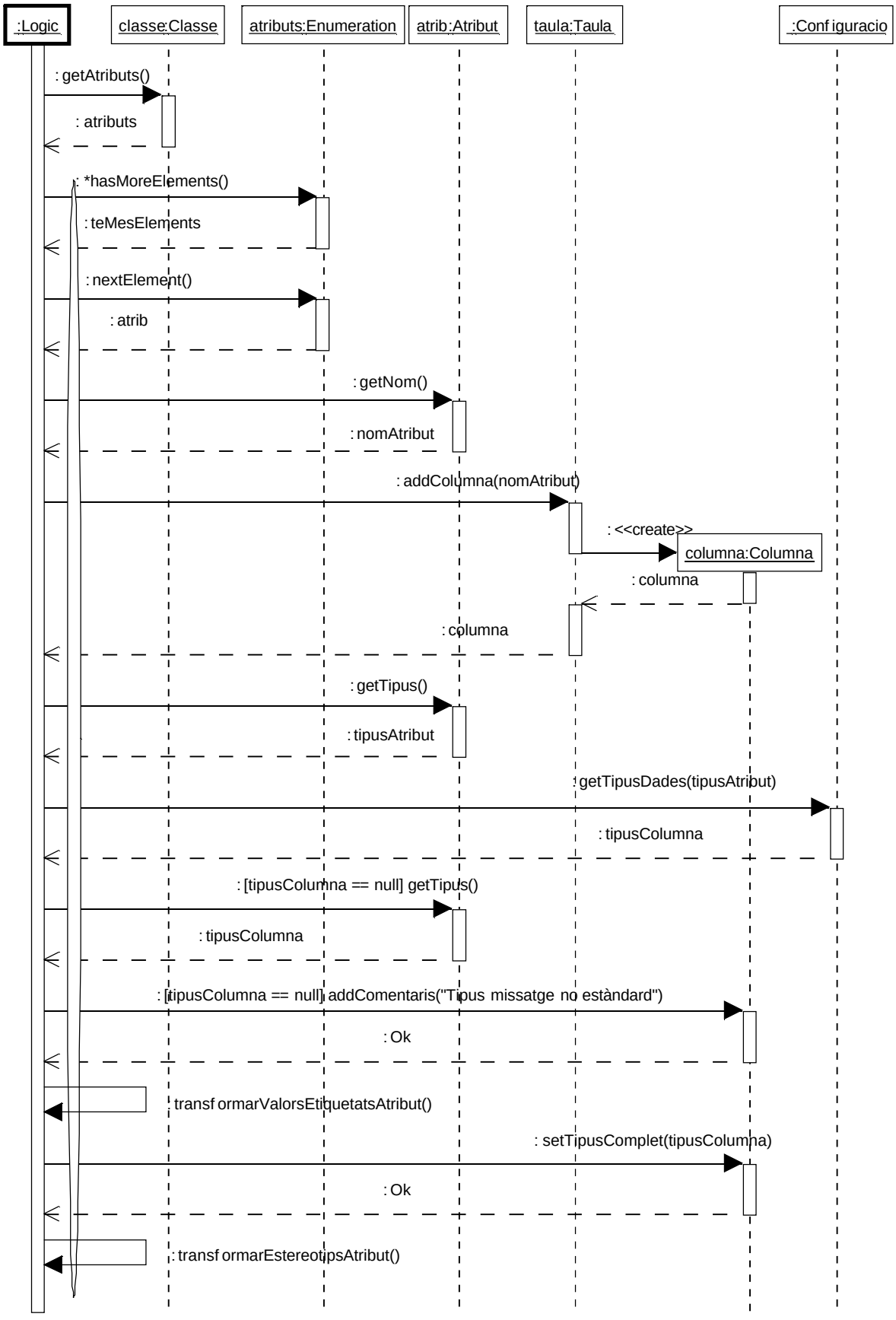


Fig. 16: Diagrama de seqüència de transformarAtribut()

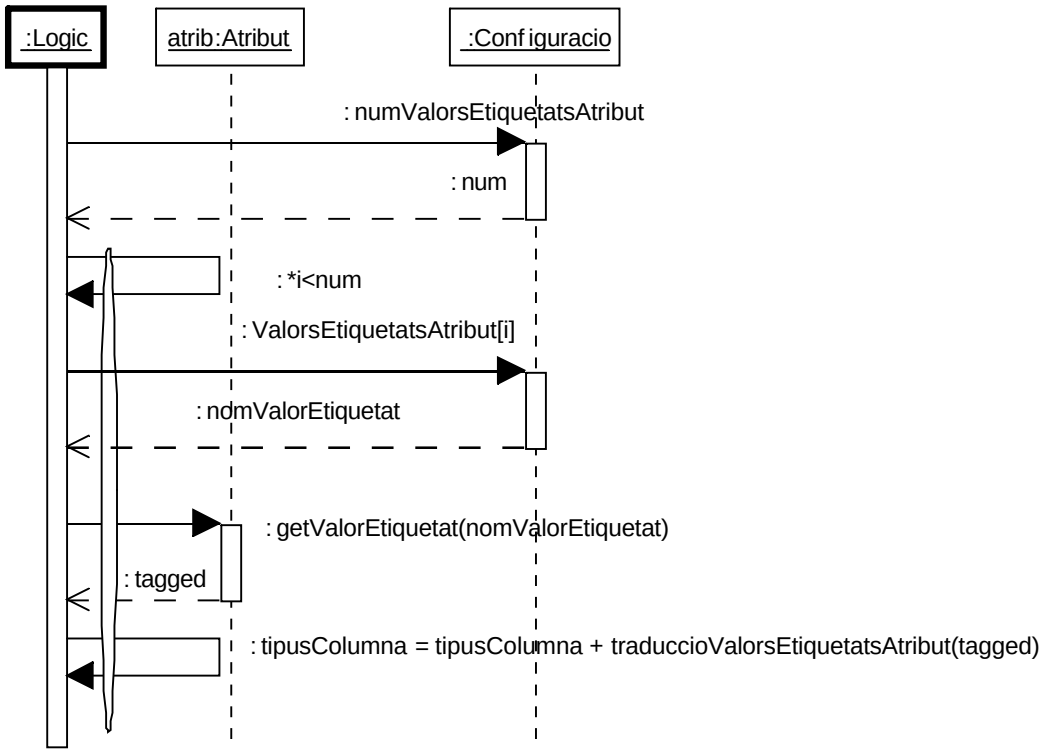


Fig. 17: Diagrama de seqüència de transformarValorsEtiquetatsAtribut()

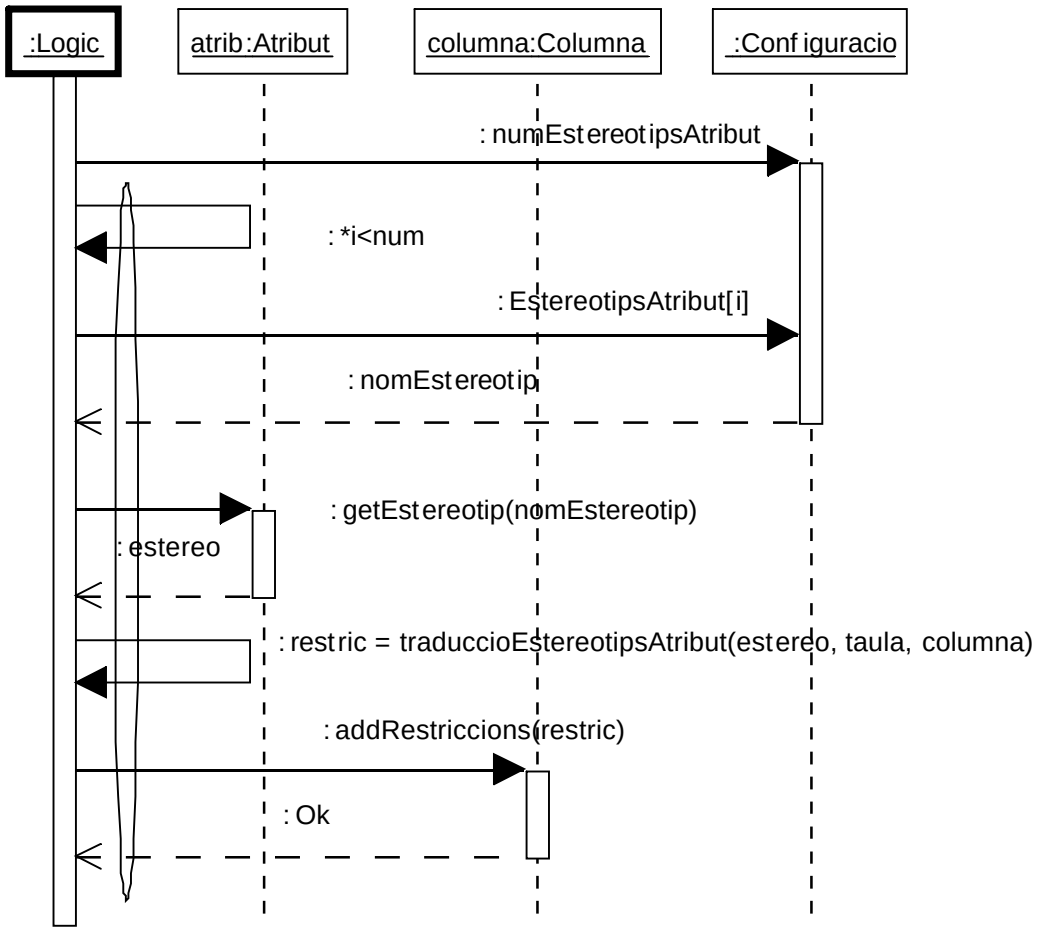


Fig. 18: Diagrama de seqüència de transformarEstereotipsAtribut()

5.2.2.6. Diagrama de sequència de transformarAssociacio()

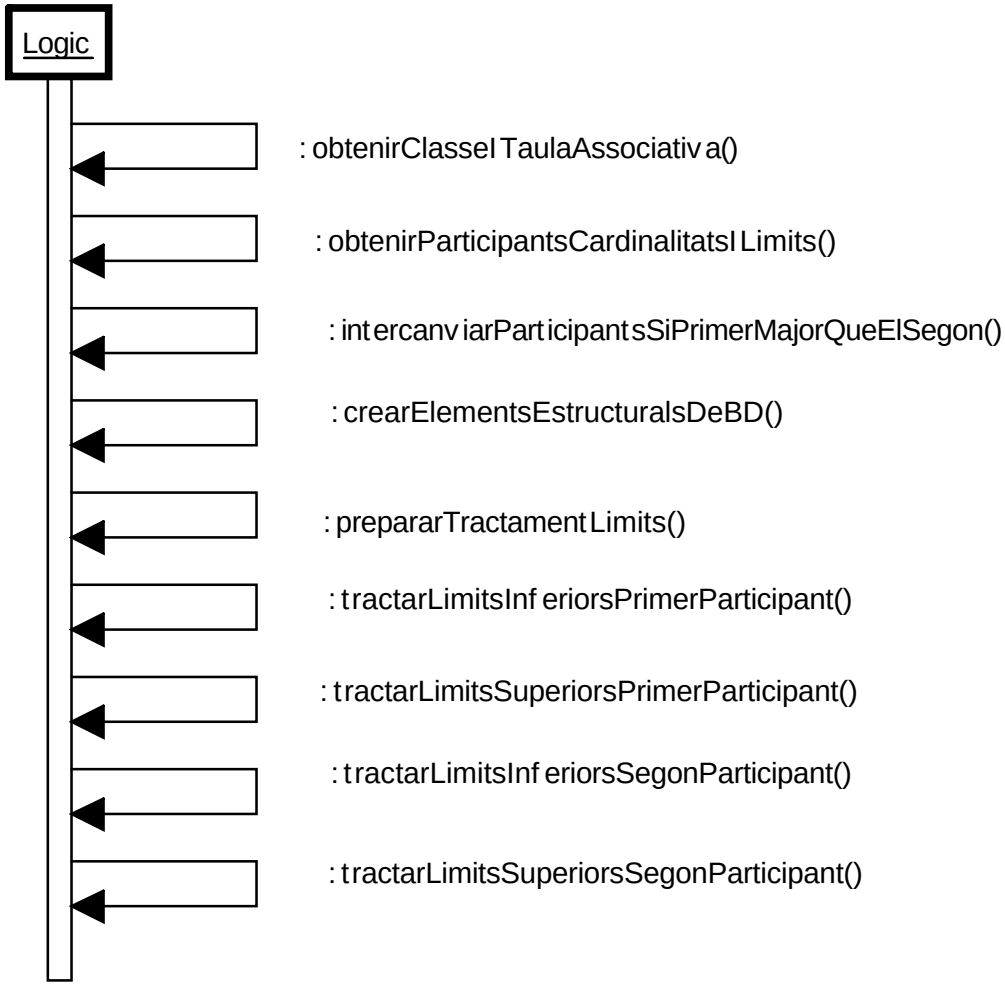


Fig. 19: Diagrama de seqüència de transformarAssociacio()

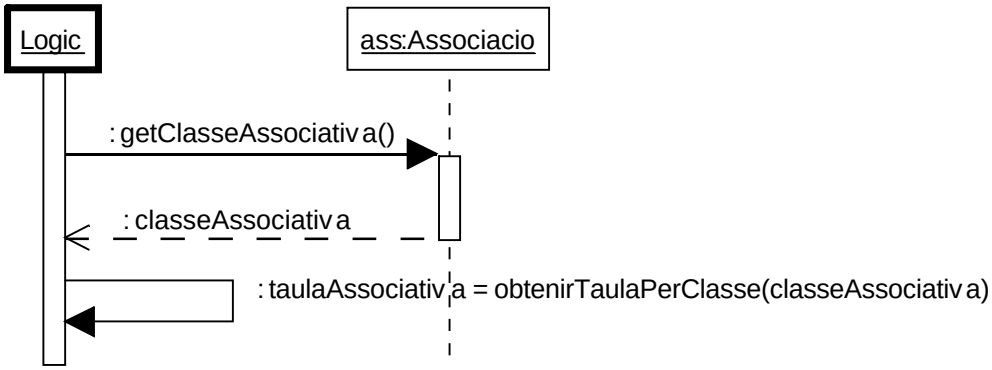


Fig. 20: Diagrama de seqüència d'obtenirClasseITaulaAssociativa()

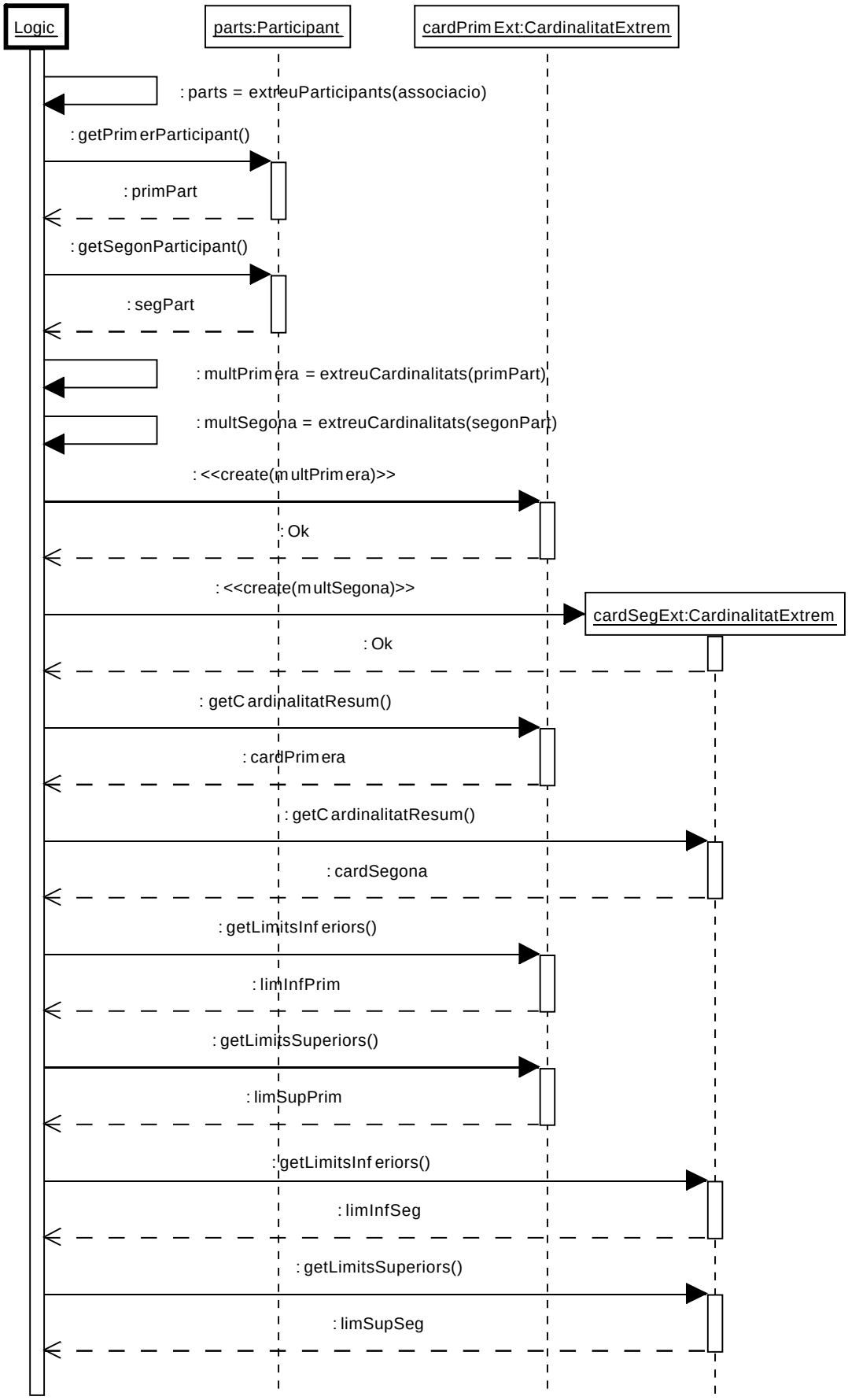


Fig. 21: Diagrama de seqüència d'obtenirParticipantsCardinalitatsILimits()

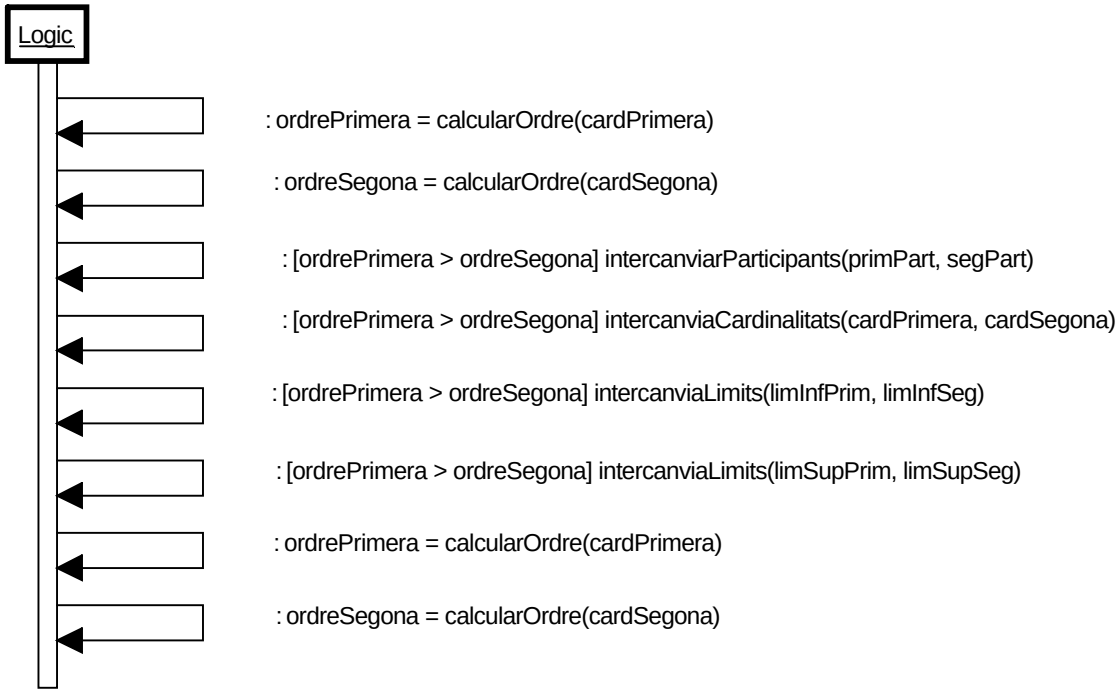


Fig. 22: Diagrama de seqüència d'intercanviarParticipantsSiPrimerMajorQueElSegon()

Fig. 23: Diagrama de seqüència de crearElementsEstructuralsDeBD()

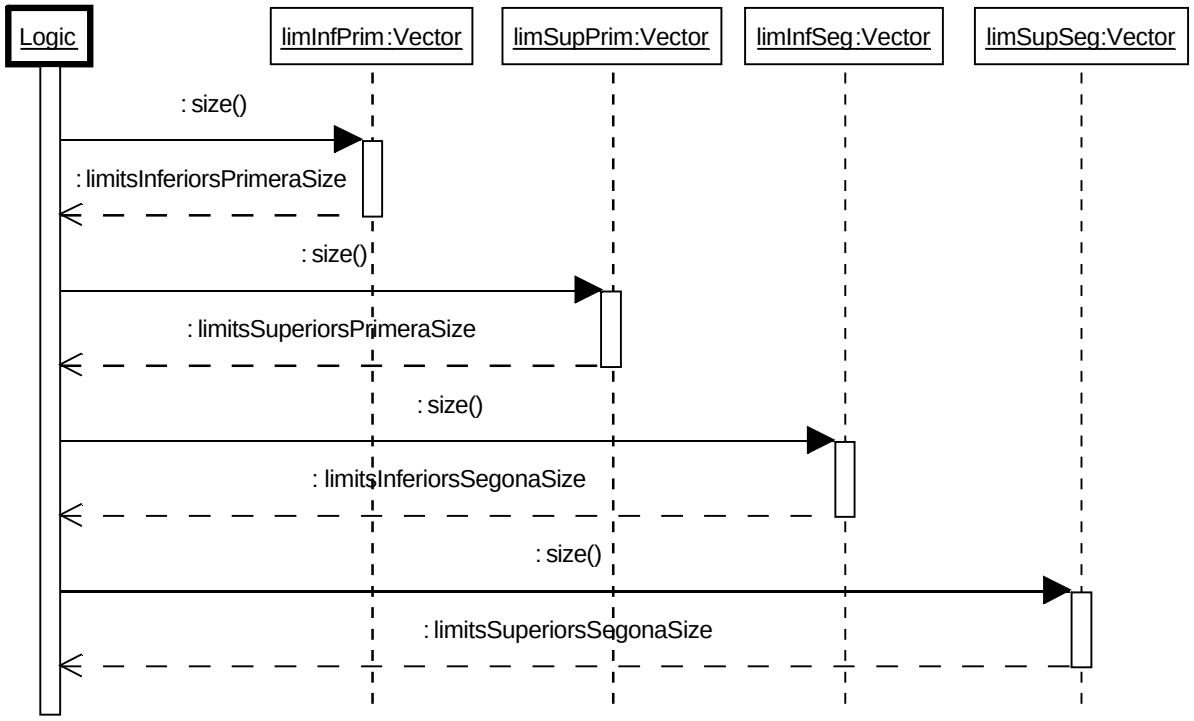


Fig. 24: Diagrama de seqüència de prepararTractamentLimits()

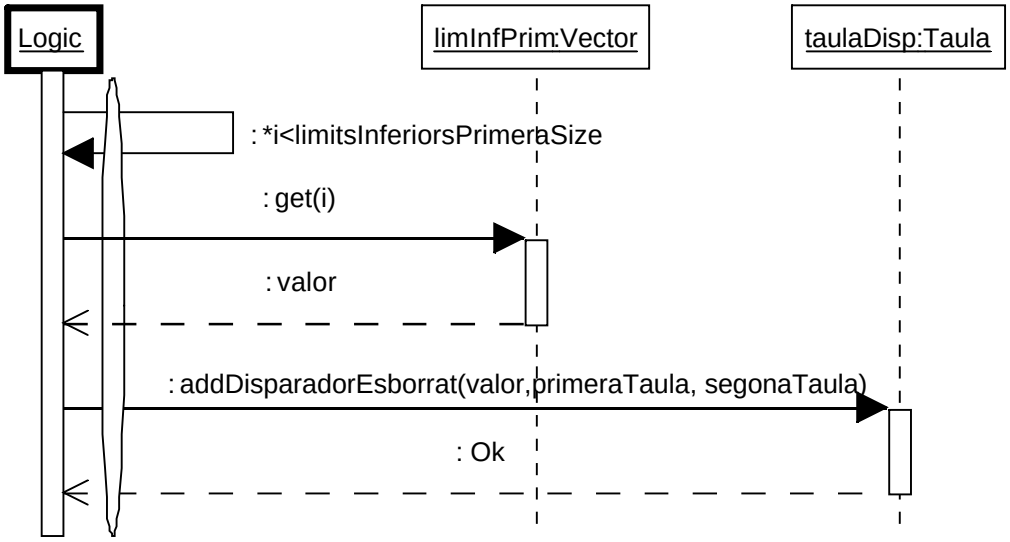


Fig. 25: Diagrama de seqüència de tractarLimitsInferiorsPrimerParticipant()

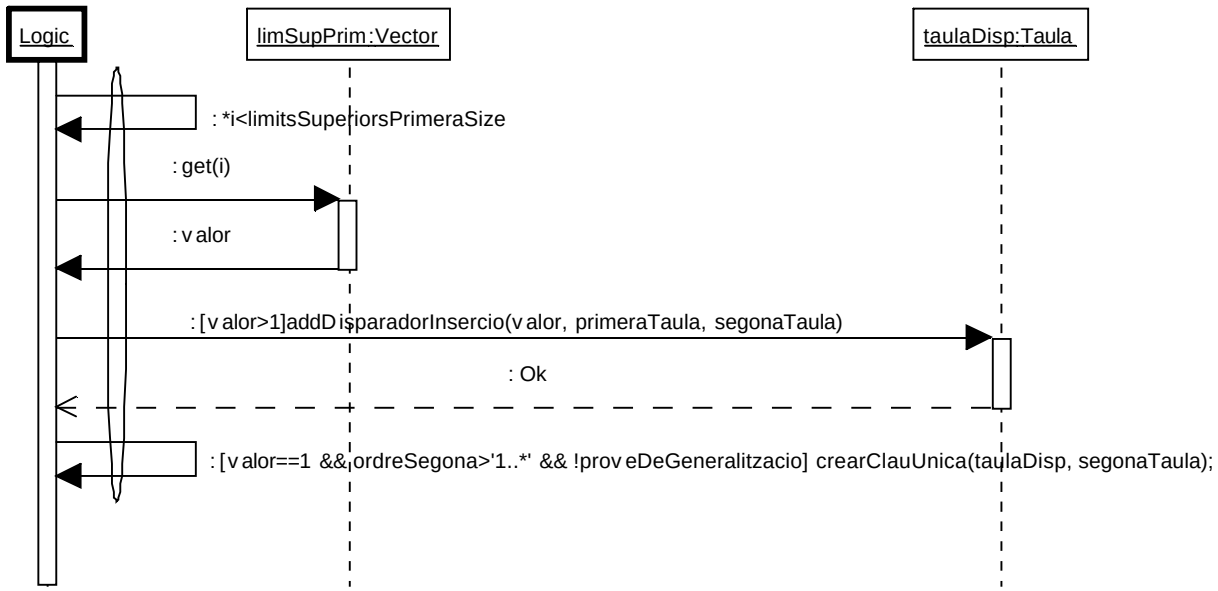


Fig. 26: Diagrama de seqüència de tractarLimitsSuperiorsPrimerParticipant()

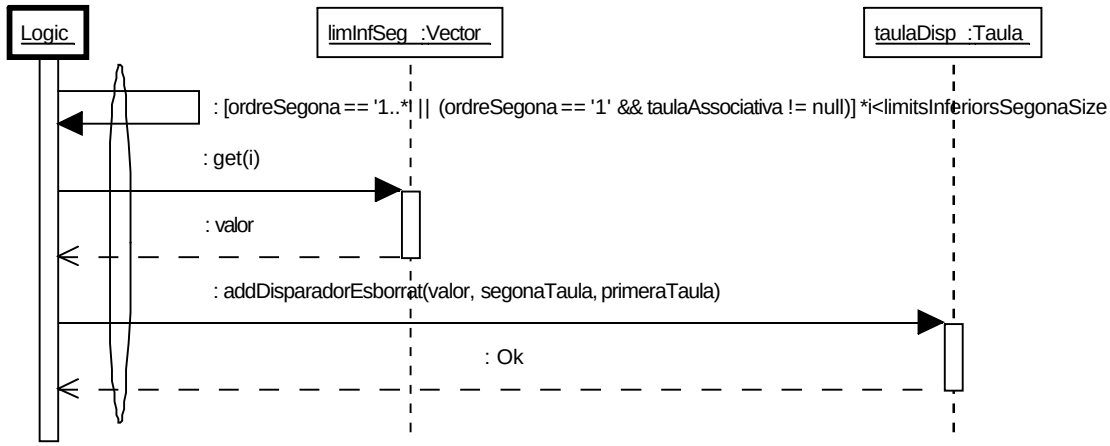


Fig. 27: Diagrama de seqüència de tractarLimitsInferiorsSegonParticipant()

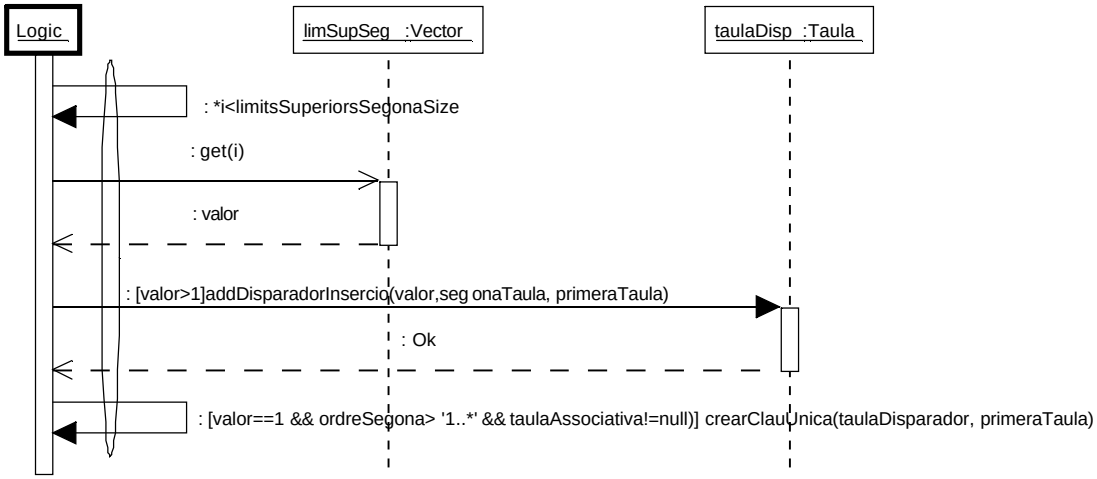


Fig. 28: Diagrama de seqüència de tractarLimitsSuperiorsSegonParticipant()

5.2.2.7. Diagrama de sequència de transformarGeneralitzacio()

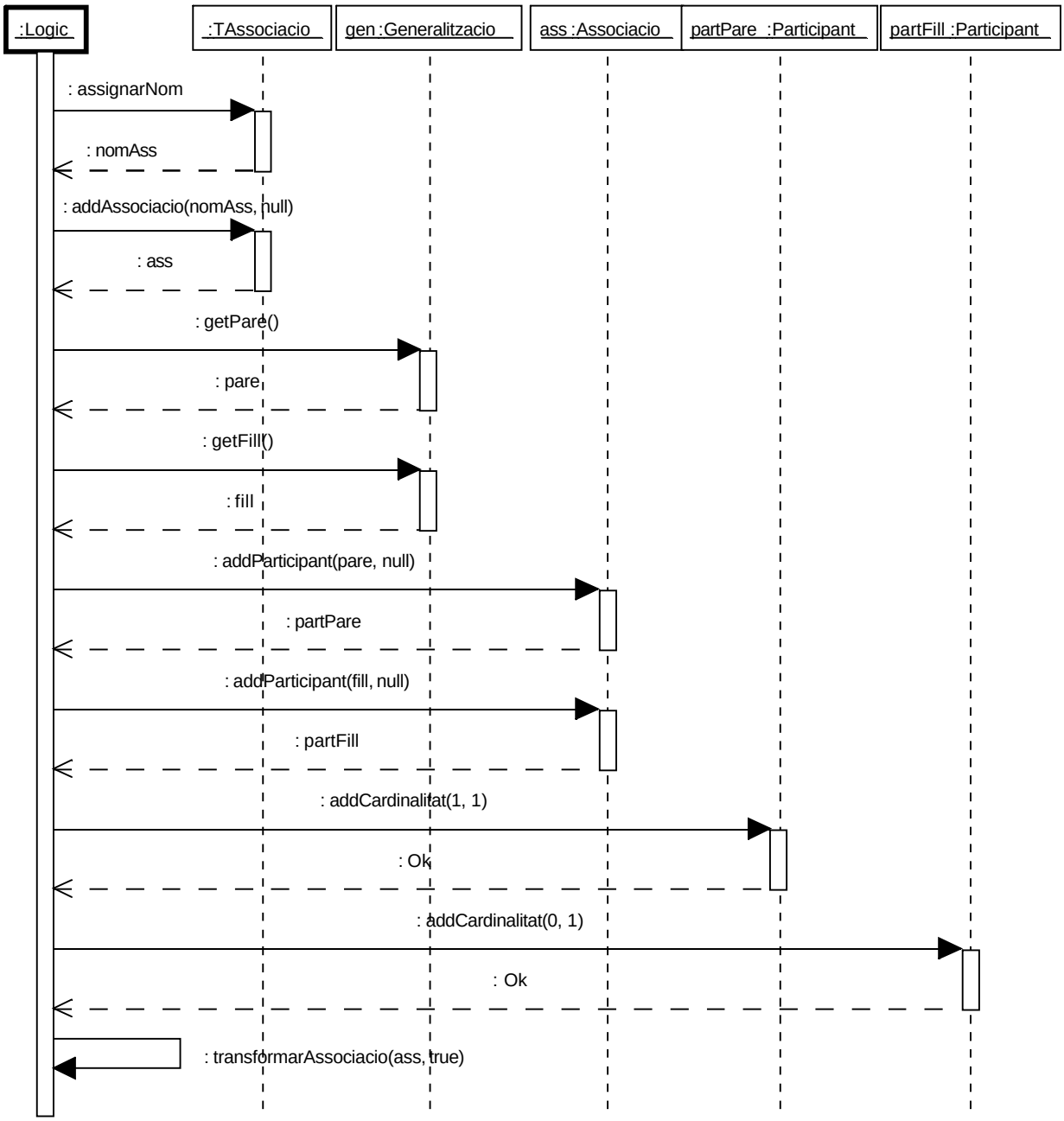


Fig. 29: Diagrama de seqüència de transformarGeneralitzacio()

5.3. Construcció

La construcció del programari està comentada en altres seccions, així com el producte resultant.

S'adjunta ara una captura de pantalla per poder observar l'aspecte visual del plug-in funcionant:

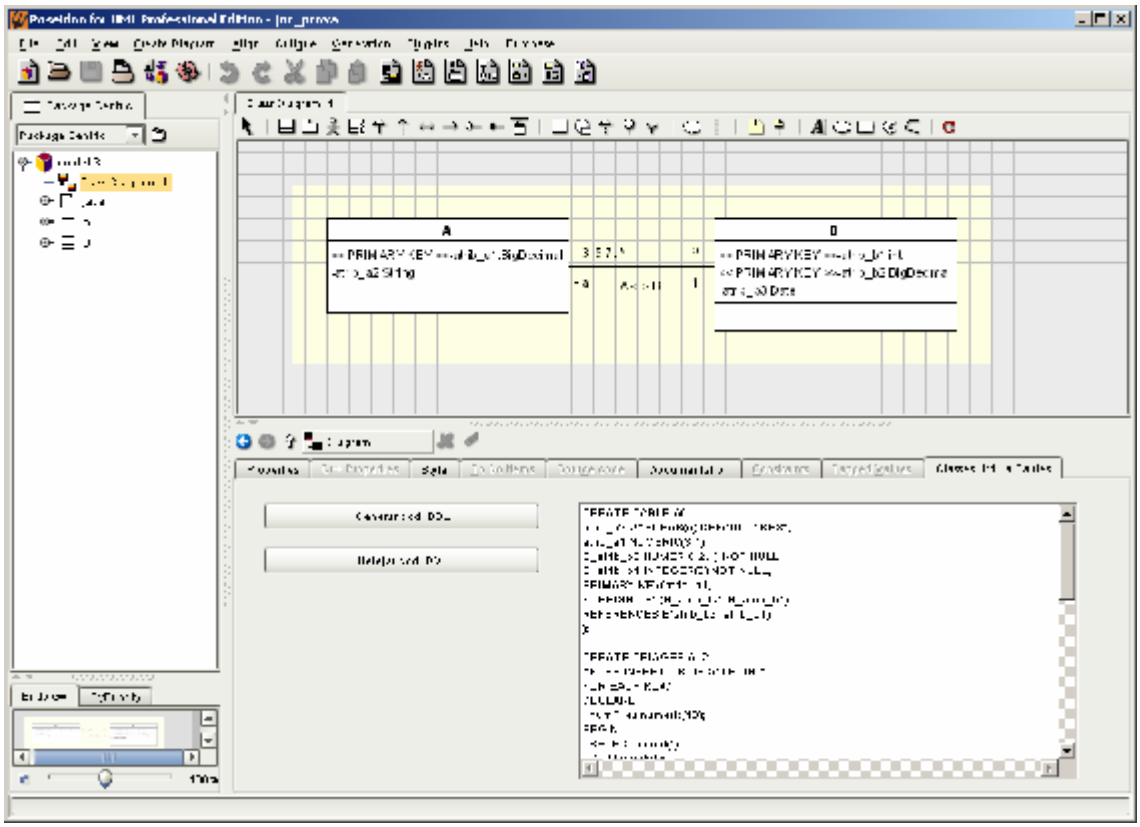


Fig. 30: Pantalla del plug-in en funcionament

Podem observar que ocupa la pestanya de més a la dreta de la part inferior de la pantalla. Es veuen també a la part de baix els dos botons d'acció (generació de codi i neteja del codi) i l'àrea de text on es genera el codi, que es pot copiar i enganxar on es vulgui.

5.4. Jocs de prova

Els resultats que genera el plug-in són sempre els esperats, incloem alguns dels casos de prova a tall d'exemple.

Partirem del següent diagrama:

A
<< PRIMARY KEY >>-atrib_a1 :BigDecimal -atrib_a2 :String

B
<< PRIMARY KEY >>-atrib_b1 :int << PRIMARY KEY >>-atrib_b2 :BigDecimal -atrib_b3 :Date

Els atributs atrib_a1, atrib_b1 i atrib_b2 tenen l'estereotip PRIMARY KEY, tal com es veu al diagrama.

L'atribut atrib_a1 té el valor etiquetat PRECISION amb el valor 3.
L'atribut atrib_a1 té el valor etiquetat SCALE amb el valor 1.
L'atribut atrib_a2 té el valor etiquetat DEFAULT amb el valor "RES".
L'atribut atrib_a2 té el valor etiquetat LENGTH amb el valor 5.

L'atribut atrib_b1 té el valor etiquetat PRECISION amb el valor 3.
L'atribut atrib_b2 té el valor etiquetat PRECISION amb el valor 2.
L'atribut atrib_b2 té el valor etiquetat SCALE amb el valor 1.

Afegirem tot seguit una associació entre les classes A i B i anirem canviant les seves cardinalitats als extrems.

Les taules que es generen si no hi ha cap associació són les següents:

```
CREATE TABLE A(
atrib_a2 VARCHAR(5) DEFAULT "RES",
atrib_a1 NUMERIC(3,1),
PRIMARY KEY(atrib_a1)
);

CREATE TABLE B(
atrib_b3 DATE,
atrib_b2 NUMERIC(2,1),
atrib_b1 INTEGER(3),
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

Quan es digui que hi ha una classe associativa, cal suposar que l'associació és una classe associativa amb l'atribut atrib_c1 DATE, el qual no té valors etiquetats ni estereotips. Per tant, la taula C representarà una classe associativa amb un atribut

```
CREATE TABLE C(
atrib_c1 DATE
);
```

Aquí s'inclouen alguns resultats:

0..* a 0..* sense classe associativa

```
CREATE TABLE A(
atrib_a2 VARCHAR(5) DEFAULT "RES",
atrib_a1 NUMERIC(3,1),
PRIMARY KEY(atrib_a1)
);

CREATE TABLE A_B(
A_atrib_a1 NUMERIC(3,1),
B_atrib_b2 NUMERIC(2,1),
B_atrib_b1 INTEGER(3),
PRIMARY KEY(A_atrib_a1, B_atrib_b2, B_atrib_b1),
FOREIGN KEY(A_atrib_a1)
REFERENCES A(atrib_a1),
FOREIGN KEY(B_atrib_b2, B_atrib_b1)
REFERENCES B(atrib_b2, atrib_b1)
);

CREATE TABLE B(
atrib_b3 DATE,
atrib_b2 NUMERIC(2,1),
atrib_b1 INTEGER(3),
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

0..1 a 0..1 sense classe associativa

```
CREATE TABLE A(
atrib_a2 VARCHAR(5) DEFAULT "RES",
atrib_a1 NUMERIC(3,1),
B_atrib_b2 NUMERIC(2,1),
B_atrib_b1 INTEGER(3),
PRIMARY KEY(atrib_a1),
FOREIGN KEY(B_atrib_b2, B_atrib_b1)
REFERENCES B(atrib_b2, atrib_b1),
UNIQUE(B_atrib_b2, B_atrib_b1)
);

CREATE TABLE B(
atrib_b3 DATE,
atrib_b2 NUMERIC(2,1),
atrib_b1 INTEGER(3),
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

1..* a 1..* amb classe associativa

```
CREATE TABLE A(
atrib_a2 VARCHAR(5) DEFAULT "RES",
atrib_a1 NUMERIC(3,1),
PRIMARY KEY(atrib_a1)
);

CREATE TABLE C(
A_atrib_a1 NUMERIC(3,1),
B_atrib_b2 NUMERIC(2,1),
B_atrib_b1 INTEGER(3),
atrib_c1 DATE,
PRIMARY KEY(A_atrib_a1, B_atrib_b2, B_atrib_b1),
FOREIGN KEY(A_atrib_a1)
REFERENCES A(atrib_a1),
FOREIGN KEY(B_atrib_b2, B_atrib_b1)
REFERENCES B(atrib_b2, atrib_b1)
);

CREATE TRIGGER C_2
AFTER DELETE ON C
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM C
  WHERE A_atrib_a1 = :old.A_atrib_a1;
  IF (numFiles < 1) THEN
    RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
  END IF;
END;

CREATE TRIGGER C_1
AFTER DELETE ON C
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM C
  WHERE B_atrib_b2 = :old.B_atrib_b2
     AND B_atrib_b1 = :old.B_atrib_b1;
  IF (numFiles < 1) THEN
    RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
  END IF;
END;

CREATE TABLE B(
atrib_b3 DATE,
atrib_b2 NUMERIC(2,1),
atrib_b1 INTEGER(3),
```

```
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

1 a 1 amb classe associativa

```
CREATE TABLE A(
atrib_a2 VARCHAR(5) DEFAULT "RES",
atrib_a1 NUMERIC(3,1),
PRIMARY KEY(atrib_a1)
);

CREATE TABLE C(
A_atrib_a1 NUMERIC(3,1),
B_atrib_b2 NUMERIC(2,1),
B_atrib_b1 INTEGER(3),
atrib_c1 DATE,
PRIMARY KEY(A_atrib_a1, B_atrib_b2, B_atrib_b1),
FOREIGN KEY(A_atrib_a1)
REFERENCES A(atrib_a1),
FOREIGN KEY(B_atrib_b2, B_atrib_b1)
REFERENCES B(atrib_b2, atrib_b1),
UNIQUE(B_atrib_b2, B_atrib_b1) ,
UNIQUE(A_atrib_a1)
);

CREATE TRIGGER C_2
AFTER DELETE ON C
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM C
  WHERE A_atrib_a1 = :old.A_atrib_a1;
  IF (numFiles < 1) THEN
    RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
  END IF;
END;

CREATE TRIGGER C_1
AFTER DELETE ON C
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM C
  WHERE B_atrib_b2 = :old.B_atrib_b2
     AND B_atrib_b1 = :old.B_atrib_b1;
  IF (numFiles < 1) THEN
    RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
  END IF;
END;

CREATE TABLE B(
atrib_b3 DATE,
atrib_b2 NUMERIC(2,1),
atrib_b1 INTEGER(3),
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

3..5 a 7..9 sense classe associativa

```
CREATE TABLE A(
atrib_a2 VARCHAR(5) DEFAULT "RES",
atrib_a1 NUMERIC(3,1),
PRIMARY KEY(atrib_a1)
);

CREATE TABLE A_B(
A_atrib_a1 NUMERIC(3,1),
B_atrib_b2 NUMERIC(2,1),
B_atrib_b1 INTEGER(3),
```

```
PRIMARY KEY(A_atrib_a1, B_atrib_b2, B_atrib_b1),
FOREIGN KEY(A_atrib_a1)
REFERENCES A(atrib_a1),
FOREIGN KEY(B_atrib_b2, B_atrib_b1)
REFERENCES B(atrib_b2, atrib_b1)
);

CREATE TRIGGER A_B_3
AFTER DELETE ON A_B
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM A_B
 WHERE A_atrib_a1 = :old.A_atrib_a1;
 IF (numFiles < 7 OR numFiles > 9) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = A_B, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TRIGGER A_B_2
AFTER INSERT OR UPDATE ON A_B
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM A_B
 WHERE B_atrib_b2 = :new.B_atrib_b2
    AND B_atrib_b1 = :new.B_atrib_b1;
 IF (numFiles < 3 OR numFiles > 5) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = A_B, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TRIGGER A_B_4
AFTER INSERT OR UPDATE ON A_B
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM A_B
 WHERE A_atrib_a1 = :new.A_atrib_a1;
 IF (numFiles < 7 OR numFiles > 9) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = A_B, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TRIGGER A_B_1
AFTER DELETE ON A_B
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM A_B
 WHERE B_atrib_b2 = :old.B_atrib_b2
    AND B_atrib_b1 = :old.B_atrib_b1;
 IF (numFiles < 3 OR numFiles > 5) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = A_B, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TABLE B(
atrib_b3 DATE,
atrib_b2 NUMERIC(2,1),
atrib_b1 INTEGER(3),
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

(0..1, 1..*) a (3..5, 7..9) amb classe associativa

```
CREATE TABLE A(
atrib_a2 VARCHAR(5) DEFAULT "RES",
atrib_a1 NUMERIC(3,1),
PRIMARY KEY(atrib_a1)
);

CREATE TABLE C(
A_atrib_a1 NUMERIC(3,1),
B_atrib_b2 NUMERIC(2,1),
B_atrib_b1 INTEGER(3),
atrib_c1 DATE,
PRIMARY KEY(A_atrib_a1, B_atrib_b2, B_atrib_b1),
FOREIGN KEY(A_atrib_a1)
REFERENCES A(atrib_a1),
FOREIGN KEY(B_atrib_b2, B_atrib_b1)
REFERENCES B(atrib_b2, atrib_b1)
);

CREATE TRIGGER C_1
AFTER DELETE ON C
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM C
  WHERE A_atrib_a1 = :old.A_atrib_a1;
  IF (numFiles < 3 OR numFiles > 5 AND numFiles < 7 OR numFiles > 9) THEN
    RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
  END IF;
END;

CREATE TRIGGER C_2
AFTER INSERT OR UPDATE ON C
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM C
  WHERE A_atrib_a1 = :new.A_atrib_a1;
  IF (numFiles < 3 OR numFiles > 5 AND numFiles < 7 OR numFiles > 9) THEN
    RAISE_APPLICATION_ERROR(-20000,'taula = C, files = ' || to_char(numFiles));
  END IF;
END;

CREATE TABLE B(
atrib_b3 DATE,
atrib_b2 NUMERIC(2,1),
atrib_b1 INTEGER(3),
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

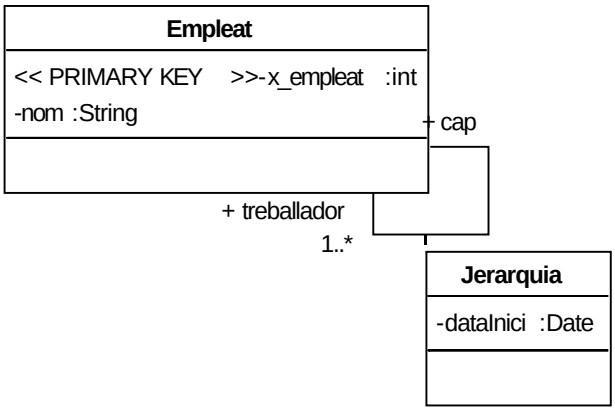
(3..5, 7..*) a 1 sense classe associativa

```
CREATE TABLE A(
atrib_a2 VARCHAR(5) DEFAULT "RES",
atrib_a1 NUMERIC(3,1),
B_atrib_b2 NUMERIC(2,1) NOT NULL,
B_atrib_b1 INTEGER(3) NOT NULL,
PRIMARY KEY(atrib_a1),
FOREIGN KEY(B_atrib_b2, B_atrib_b1)
REFERENCES B(atrib_b2, atrib_b1)
);

CREATE TRIGGER A_2
AFTER INSERT OR UPDATE ON A
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
```

```
 INTO numFiles
 FROM A
 WHERE B_atrib_b2 = :new.B_atrib_b2
    AND B_atrib_b1 = :new.B_atrib_b1;
 IF (numFiles < 3 OR numFiles > 5 AND numFiles < 7) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = A, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TRIGGER A_1
AFTER DELETE ON A
FOR EACH ROW
DECLARE
 numFiles numeric(10);
BEGIN
 SELECT count(*)
 INTO numFiles
 FROM A
 WHERE B_atrib_b2 = :old.B_atrib_b2
    AND B_atrib_b1 = :old.B_atrib_b1;
 IF (numFiles < 3 OR numFiles > 5 AND numFiles < 7) THEN
   RAISE_APPLICATION_ERROR(-20000,'taula = A, files = ' || to_char(numFiles));
 END IF;
END;

CREATE TABLE B(
atrib_b3 DATE,
atrib_b2 NUMERIC(2,1),
atrib_b1 INTEGER(3),
PRIMARY KEY(atrib_b2, atrib_b1)
);
```

Reflexivitat (1..* a 1) amb classe associativa

Partirem del següent diagrama:



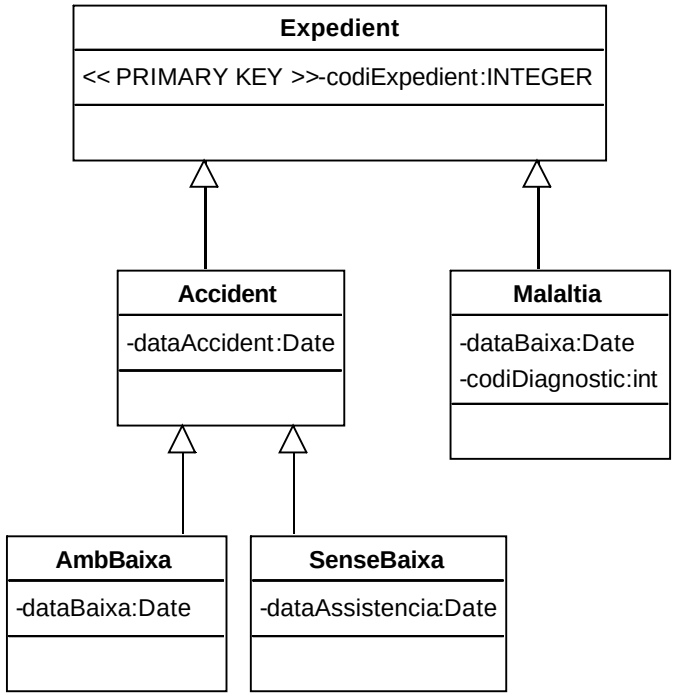
L'atribut x_empleat té l'estereotip PRIMARY KEY, tal com es veu al diagrama.
L'atribut x_empleat té el valor etiquetat PRECISION amb el valor 9.
L'atribut nom té el valor etiquetat LENGTH amb el valor 60.

```
CREATE TABLE Empleat(
nom VARCHAR(60),
x_empleat INTEGER(9),
PRIMARY KEY(x_empleat)
);

CREATE TABLE Jerarquia(
dataInici DATE,
Empleat_x_empleat_1 INTEGER(9),
Empleat_x_empleat INTEGER(9),
PRIMARY KEY(Empleat_x_empleat_1, Empleat_x_empleat),
FOREIGN KEY(Empleat_x_empleat_1)
```

```
REFERENCES Empleat(x_empleat),
FOREIGN KEY(Empleat_x_empleat)
REFERENCES Empleat(x_empleat),
UNIQUE(Empleat_x_empleat)
);

CREATE TRIGGER Jerarquia_2
AFTER DELETE ON Jerarquia
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM Jerarquia
  WHERE Empleat_x_empleat_1 = :old.Empleat_x_empleat_1;
  IF (numFiles < 1) THEN
    RAISE_APPLICATION_ERROR(-20000,'taula = Jerarquia, files = ' || to_char(numFiles));
  END IF;
END;

CREATE TRIGGER Jerarquia_1
AFTER DELETE ON Jerarquia
FOR EACH ROW
DECLARE
  numFiles numeric(10);
BEGIN
  SELECT count(*)
  INTO numFiles
  FROM Jerarquia
  WHERE Empleat_x_empleat = :old.Empleat_x_empleat;
  IF (numFiles < 1) THEN
    RAISE_APPLICATION_ERROR(-20000,'taula = Jerarquia, files = ' || to_char(numFiles));
  END IF;
END;
```

Generalització

Partirem del següent diagrama:



L'atribut codiExpedient té l'estereotip PRIMARY KEY, tal com es veu al diagrama.
L'atribut codiExpedient té el valor etiquetat PRECISION amb el valor 7.
L'atribut codiDiagnostic té el valor etiquetat PRECISION amb el valor 9.

```
CREATE TABLE Accident(
dataAccident DATE,
Expedient_codiExpedient INTEGER(7),
PRIMARY KEY(Expedient_codiExpedient),
FOREIGN KEY(Expedient_codiExpedient)
REFERENCES Expedient(codiExpedient)
);

CREATE TABLE Malaltia(
Expedient_codiExpedient INTEGER(7),
codiDiagnostic INTEGER(9),
dataBaixa DATE,
PRIMARY KEY(Expedient_codiExpedient),
FOREIGN KEY(Expedient_codiExpedient)
REFERENCES Expedient(codiExpedient)
);

CREATE TABLE AmbBaixa(
Accident_Expedient_codiExpedient INTEGER(7),
dataBaixa DATE,
PRIMARY KEY(Accident_Expedient_codiExpedient),
FOREIGN KEY(Accident_Expedient_codiExpedient)
REFERENCES Accident(Expedient_codiExpedient)
);

CREATE TABLE Expedient(
codiExpedient INTEGER(7),
PRIMARY KEY(codiExpedient)
);

CREATE TABLE SenseBaixa(
dataAssistencia DATE,
Accident_Expedient_codiExpedient INTEGER(7),
PRIMARY KEY(Accident_Expedient_codiExpedient),
FOREIGN KEY(Accident_Expedient_codiExpedient)
REFERENCES Accident(Expedient_codiExpedient)
);
```

6. Valoració econòmica

El PFC s'ha realitzat en poc més de 300 hores segons la distribució de fases adjuntada. El cost total és una mica superior als 4.000 euros.

Fase	Rol	Preu/hora	Hores	Total
Estudi teòric	Analista	15	65	975
Estudi tecnologia + Anàlisi	Analista	15	33	495
Estudi extensió de l'eina	Analista	15	25	375
Disseny	Dissenyador	13	38	494
Construcció	Programador	10	99	990
Jocs de prova	Dissenyador	13	20	260
Documentació	Dissenyador	13	12	156
Redacció memòria	Analista	15	12	180
Construcció presentació virtual	Analista	15	15	225
			319	**4.150**

Fig. 31: Desglossament per fases i rols de la valoració econòmica

7. Conclusions

Tot plegat, ha estat un treball força interessant en la part teòrica. La implementació ha estat relativament senzilla (exceptuant la part de la transformació de les associacions que és complexa i llarga) un cop aclarits els dubtes teòrics. Considerem que s'han assolit els objectius del projecte: s'ha fet un estudi complet de la transformació UML a SQL i el plug-in resultant és operatiu i funcional.

Aquest PFC pot ser la primera versió d'un treball més ambiciós. Es podria continuar més endavant amb coses com el tractament de la persistència, de les restriccions de l'herència, de les relacions n-àries amb n>2, la possibilitat d'afegir altres tipus de restriccions com checks i claus úniques definides per l'usuari i en general qualsevol element que pugui aparèixer en una base de dades relacional (fins i tot, paràmetres físics com tablespaces). Caldria trobar els estereotips o valors etiquetats adequats orientats a la generació de codi SQL per tal de fer més poderosa la capacitat expressiva dels diagrames de classes UML.

Seria interessant aconseguir que es pogués construir el plug-in amb menors dificultats tècniques derivades de la dependència tecnològica de l'empresa Gentleware (necessitat d'una clau nova si es canvia el nom o la versió del plug-in) i del fet que està embolcallat per l'estructura de l'aplicació *apidemo*.

La pròpia estructura del plug-in permet que, modificant només la classe instal·ladora i els procediments de recollida del model de l'eina, es pugui implementar sobre eines diferents de Poseidon que permetin la instal·lació de plug-ins escrits en Java (en general, al plug-in li valdrà qualsevol entrada que s'adapti al model de classes que recull el model dels diagrames). Es podria fer una classe que recollís el model d'on fos i el passés al model de classes que recull el model del diagrames; per a cada eina o entorn es podria escriure una classe "recollidora" pròpia. És una possibilitat que es podria estudiar, ja que ampliaria molt l'abast d'aquest plug-in.

Seguidament s'inclouen les diferències entre el plug-in construït i el que va incloure Gentleware a l'eina Poseidon. El plug-in del PFC està completament orientat a la generació de codi, de manera que inclou pocs elements que no siguin estrictament necessaris per aquesta finalitat (per exemple, NOT NULL). També s'inclou DEFAULT, que no inclou Gentleware, que en canvi inclou UNIQUE com a estereotip d'atribut. Aquest darrer fet no és gaire correcte si es té en compte que una columna pot pertànyer a més d'una clau única dins de la mateixa taula. Gentleware abusa de NOT NULL, posant-lo a totes les columnes que són PRIMARY KEY.

El tractament que fa Gentleware de les associacions és molt pobre. Les relacions molts a molts les resol amb una clau forana a cada extrem de la relació, en comptes de crear una taula-relació. També resol així les relacions u a u. Gentleware no controla els límits de les cardinalitats als extrems amb disparadors o claus úniques. Tampoc presta atenció a l'herència.

Element	Tipus	Estereotip	Valor etiquetat(nom;valor)	PFC	Gentl.
PRIMARY KEY	Restricció de columnes	PRIMARY KEY		Sí	Sí
UNIQUE	Restricció de columnes	UNIQUE		No	**Malament**
NOT NULL	Restricció de columna	NOT NULL		Sí	Sí
CHECK	Restricció de columna		CHECK; Condicio_sobre_columna	No	No
PRIMARY KEY	Restricció de taula		PRIMARY KEY; Llista columnes	No	No
UNIQUE	Restricció de taula		UNIQUE; Llista columnes	No	No
CHECK	Restricció de taula		CHECK; Condicio_sobre_columnes	No	No
DEFAULT	Modificador tipus		DEFAULT; Valor_per_defecte	**Sí**	No
LENGTH	Modificador tipus		LENGHT; Valor_enter	Sí	Sí
PRECISION	Modificador tipus		PRECISION; Valor_enter	Sí	Sí
SCALE	Modificador tipus		SCALE; Valor_enter	Sí	Sí
FOREIGN KEY	Resultat de traducció d'associació			Sí	Sí
Disparador	Resultat de traducció d'associació			**Sí**	No
UNIQUE	Resultat de traducció d'associació			**Sí**	No
Taula-relació	Resultat de traducció d'associació			**Sí**	No
Generalització				**Sí**	No
Persistència				No	No

Fig. 32: Diferències entre el plug-in construït i el de Gentleware

Glossari

Agregació
És una relació estructural que especifica que un element és "part" d'un altre (agregat). Si els continguts no tenen sentit sense el contenidor i no poden formar part de més d'un contenidor, parlem de *composició*; altrament parlem d'*agregació simple*.

Associació
És una relació estructural que especifica que els objectes d'un element es connecten als objectes d'un altre.

Atribut
Propietat d'una classe identificada amb un nom, que descriu un rang de valors que poden prendre les instàncies de la propietat.

Atribut derivat
Atribut el valor del qual ve determinat pel valor d'altres atributs.

Base de dades
Col·lecció d'informació.

Cardinalitat
Nombre d'elements d'un conjunt.

Cas d'ús
Flux complet d'accions iniciades per un actor que el sistema executa, les quals proporcionen un resultat a l'actor.

Classe
Descripció d'un conjunt d'objectes que comparteixen els mateixos atributs, operacions, relacions i semàntica.

Classe-associació
Classe que incorpora les propietats d'una associació.

Classe de control
Classe activa que controla el comportament d'altres classes.

Classe de frontera
Classe que representa la interfície de l'usuari amb la pantalla, o sigui la interfície d'usuari entre un cas d'ús i un actor.

Clau única
Conjunt de columnes dins d'una taula que no poden repetirse com a conjunt en l'esmentada taula.

Clau forana
Conjunt de columnes dins d'una taula que mapegen la clau primària d'una altra taula.

Clau primària
Conjunt de columnes dins d'una taula que formen la clau candidata que es tria per identificar files en una taula.

DDL
Data Definition Language (llenguatge de definició de dades).

Diagrama de casos d'ús
Diagrama que mostra els casos d'ús i les seves relacions amb actors i altres casos d'ús. El model de casos d'ús és un model de les funcions desitjades del sistema i el seu entorn, les quals suporten els processos de negoci. Aquest model serveix com un contracte entre el client i els desenvolupadors.

Diagrama de classes
Diagrama que mostra classes, les seves relacions internes i amb d'altres elements del model. Pretèn explicitar l'estructura bàsica del sistema.

Diagrama d'interacció
Poden ser diagrames de seqüència o diagrames de col·laboració, els quals mostren les interaccions dels objectes dins del sistema.

Diagrama de seqüència
Diagrama on es mostra la col·laboració d'objectes i els missatges que s'envien entre ells en ordre temporal. Expliciten com es realitzen els casos d'ús.

Disparador
Procediment guardat a la base de dades dissenyat per executar-se quan es modifiquen els valors d'una taula. En anglès, *trigger*.

Domini
Conjnt de valors vàlids d'un atribut o una columna.

Entitat
Objecte de negoci o de sistema.

Esquema
Descripció de l'estructura d'una base de dades.

Estereotip
Extensió del vocabulari d'UML per poder crear nous blocs de construcció derivats a partir dels existents però específics d'un problema concret. N'hi ha que formen part de l'estàndard d'UML.

Generalització
Una relació entre elements del model que indica que un element (subclasse) és un "tipus de" un altre element (superclasse).

Herència
Mecanisme pel qual elements més especifícs incorporen l'estructura i el comportament d'elements més generals.

Instància
Realització concreta d'una abstracció; entitat a la que se li poden aplicar un conjunt d'operacions i que té un estat que guarda l'efecte de les operacions.

Interfície
Col·lecció d'operacions que s'usa per especificar un servei d'una classe o un component.

Integritat referencial
Regla que estableix que si existeix una clau forana en una taula, el seu valor ha de ser null o bé ha d'aparèixer en un valor de la clau primària de la taula relacionada.

Mètode
Implementació d'una operació.

Multiplicitat
Especificació del rang de cardinalitats admissible que pot admetre un extrem d'una associació.

Operació
Implementació d'un servei que es pot demanar a qualsevol objecte de la classe per a que mostri un comportament.

Postcondició
Possibles estats del sistema després de l'execució d'un cas d'ús.

Precondició
Estat del sistema requerit, o condicions que han de ser certes, per tal que un cas d'ús es pugui executar.

Relació
Associació entre taules.

Requeriment
Frase que descriu una capacitat desitjada del sistema.

Restricció
Regla que limita els valors o les accions sobre un determinat camp de dades.

Rol
Conjunt de comportaments d'un element específic per interactuar amb d'altres elements en una situació donada.

SGBD
Sistema gestor de base de dades.

SQL
Structured Query Language (Llenguatge estructurat de consultes).

Transacció
Operació o sèrie d'operacions contra una base de dades, que podran ser o no confirmades.

Tipus
Estereotip de classe que s'usa per especificar un domini d'objectes, juntament amb les operacions (no mètodes) aplicables a aquests objectes.

Tipus de dades
Tipus els valors dels quals no tenen identitat. Inclouen tipus primitius predefinits (com nombres i cadenes de caràcters), i també tipus enumerats (com els booleans).

UML
Unified Modeling Language (Llenguatge unificat de modelatge).

Valor etiquetat
Permet afegir informació addicional a un element estàndard d'UML per completar la seva especificació.

Bibliografia

- **Jacobson, I, Booch, G. i Rumbaugh, J**.: El Lenguaje unificado de modelado . Guía del usuario. Madrid: Addison-Wesley, 1999.
- **Larman, C..** UML y Patrones. Introducción al análisis y diseño orientado a objetos (1ª. ed.). Prentice Hall, 1999
- **Naiburg, E. J. i Maksimchuk, R. A.:** *UML for Database Design*. Addison-Wesley, 2001. Versió electrònica de la Biblioteca Digital de la UOC.
- **Muller, P.**: Modelado de objetos con UML. Barcelona: Gestión 2000, 1997.
- **Lopez, N. , Migueis J. i Pichon E.**: Integrar UML en los proyectos. Barcelona: Gestión 2000, 1998.
- **Jacobson, I., Booch, G. i Rumbaugh, J.**: El Proceso Unificado de Desarrollo . Madrid: Addison-Wesley, 2000.
- **Groff, J.R., Weinberg, P.N.**: Aplique SQL. Madrid: Mc-Graw-Hill, 1996.
- Apunts d'Enginyeria del Programari III de la UOC, mòdul 1.
- Apunts d'Enginyeria del Programari I de la UOC.

ANNEX A: Documentació del plug-in

La següent documentació s'ha obtingut amb l'eina Javadoc i el doclet RTFDoclet.

public class **com.gentleware.apidemo.Associacio**

Classe que representa una associació UML

Constructors

public Associacio(
String pNom)
Constructor només amb nom
Paràmetres
pNom - nom d'Associacio

public Associacio(
String pNom,
Classe pClasseAssociativa)
Constructor amb nom i classe associativa
Paràmetres
pNom - nom d'Associacio
pClasseAssociativa - Classe associativa d'Associacio

Mètodes

public java.lang.String getNom()
Obtenir el nom de l'Associacio
Retorna
nom de l'Associacio

public com.gentleware.apidemo.Classe getClasseAssociativa()
Obtenir la Classe associativa de l'Associacio
Retorna
Classe associativa de l'Associacio

public void setClasseAssociativa(
Classe pClasseAssociativa)
Establir la Classe associativa de l'Associacio
Paràmetres
pClasseAssociativa - Classe associativa d'Associacio

public com.gentleware.apidemo.Participant addParticipant(
Classe pClasse,
String pRol)
Crea un Participant i l'afegeix a la col·lecció de l'Associacio
Paràmetres
pClasse - Classe participant
pRol - rol de la Classe participant
Retorna
nou Participant creat

public java.util.Enumeration getParticipants()
Torna una Enumeration amb tots els Participant de l'Associacio
Retorna
tots els Participant de l'Associacio

public class **com.gentleware.apidemo.Atribut**

Classe que representa un atribut d'una classe UML

Constructors

public Atribut(
String pNom,
String pTipus)
Constructor
Paràmetres

pNom - nom de l'Atribut
pTipus - tipus de l'Atribut

Mètodes

public java.lang.String getNom()
Obtenir el nom de l'Atribut
Retorna
nom de l'Atribut

public java.lang.String getTipus()
Obtenir el tipus de l'Atribut
Retorna
tipus de l'Atribut

public void setTipus(
String pTipus)
Establir el tipus de l'Atribut
Paràmetres
pTipus - tipus de l'Atribut

public com.gentleware.apidemo.ValorEtiquetat addValorEtiquetat(
String pNom,
String pValor)
Crea un ValorEtiquetat i l'afegeix a la col·lecció de ValorEtiquetat de l'Atribut
Paràmetres
pNom - nom del ValorEtiquetat
pValor - valor del ValorEtiquetat
Retorna
nou ValorEtiquetat creat

public com.gentleware.apidemo.ValorEtiquetat getValorEtiquetat(
String pNom)
Obtenir el ValorEtiquetat especificat de l'Atribut
Retorna
ValorEtiquetat especificat de l'Atribut

public com.gentleware.apidemo.Estereotip addEstereotip(
String pNom)
Crea un Estereotip i l'afegeix a la col·lecció d'Estereotip de l'Atribut
Paràmetres
pNom - nom de l'Estereotip
Retorna
nou Estereotip creat

public com.gentleware.apidemo.Estereotip getEstereotip(
String pNom)
Obtenir l'Estereotip especificat de l'Atribut
Retorna
Estereotip especificat de l'Atribut

public class **com.gentleware.apidemo.Cardinalitat**

Classe que representa una cardinalitat d'un extrem d'una associació UML

Constructors

public Cardinalitat(
int pCardInf,
int pCardSup)
Constructor
Paràmetres
pCardInf - la multiplicitat inferior de la Cardinalitat
pCardSup - la multiplicitat superior de la Cardinalitat

Mètodes

public int getCardInf()
Obtenir la multiplicitat inferior de la Cardinalitat
Retorna

multiplicitat inferior de la Cardinalitat

public int getCardSup()
Obtenir la multiplicitat superior de la Cardinalitat
Retorna
multiplicitat superior de la Cardinalitat

public class **com.gentleware.apidemo.Classe**

Classe que representa una classe UML

Constructors

public Classe(
String pNom)
Constructor
Paràmetres
pNom - nom de la Classe

Mètodes

public java.lang.String getNom()
Obtenir el nom de la Classe
Retorna
nom de la Classe

public com.gentleware.apidemo.Atribut addAtribut(
String pNom,
String pTipus)
Crea un Atribut i l'afegeix a la col·lecció d'Atribut de la Classe
Paràmetres
pNom - nom de l'Atribut
pTipus - tipus de l'Atribut
Retorna
nou Atribut creat

public com.gentleware.apidemo.Atribut getAtribut(
String pNom)
Obtenir un Atribut de nom especificat de la Classe
Retorna
atribut de nom especificat de la Classe

public java.util.Enumeration getAtributs()
Torna una Enumeration amb tots els Atribut de la Classe
Retorna
tots els Atribut de la Classe

public class **com.gentleware.apidemo.ClassesUMLaTaulesPane** extends TabSpawnable

Classe de presentació per generar codi DDL/SQL a partir de diagrames de classes UML de l'eina Poseidon

Constructors

public ClassesUMLaTaulesPane()
Constructor: es crea la interfície d'usuari, amb botons, àrea de text i es connecten el listeners als botons. S'encarrega de demanar el codi DDL/SQL i el mostra per pantalla.

public class **com.gentleware.apidemo.ClauForana** implements com.gentleware.apidemo.GrupColumnes

Classe que representa una clau forana d'una taula que va contra una altra taula

Constructors **public ClauForana(**
Columna pColumna,
Taula pTaula)
Constructor
Paràmetres
pColumna - primera Columna a afegir a la ClauForana
pTaula - Taula contra la que va la ClauForana

Mètodes **public void addColumna(**
Columna pColumna)
Afegeix una Columna a la col·lecció de Columna de la ClauForana
Paràmetres
pColumna - Columna a afegir a la ClauForana

public java.util.Enumeration getColumnes()
Torna una Enumeration amb totes les Columna de la ClauForana
Retorna
totes les Columna de la ClauForana

public boolean conteColumna(
Columna pColumna)
Retorna si la Columna de nom especificat pertany a la ClauForana
Paràmetres
pColumna - Nom de la columna que es valida si pertany a la ClauForana
Retorna
si la Columna de nom especificat pertany a la ClauForana

public com.gentleware.apidemo.Taula getTaula()
Obtenir la Taula contra la que va la ClauForana
Retorna
taula contra la que va la ClauForana

public class **com.gentleware.apidemo.ClauPrimaria** implements com.gentleware.apidemo.GrupColumnes

Classe que representa una clau primària d'una taula

Constructors **public ClauPrimaria()**
Constructor

Mètodes **public void addColumna(**
Columna pColumna)
Afegeix una Columna a la col·lecció de Columna de la ClauPrimaria
Paràmetres
pColumna - Columna a afegir a la ClauPrimaria

public java.util.Enumeration getColumnes()
Torna una Enumeration amb totes les Columna de la ClauPrimaria
Retorna
totes les Columna de la ClauPrimaria

public boolean conteColumna(
String pColumna)
Retorna si la Columna de nom especificat pertany a la ClauPrimaria
Paràmetres
pColumna - Nom de la columna que es valida si pertany a la ClauPrimaria
Retorna
si la Columna de nom especificat pertany a la ClauPrimaria

public class **com.gentleware.apidemo.ClauUnica** implements com.gentleware.apidemo.GrupColumnes

Classe que representa una clau única d'una taula

Constructors

public ClauUnica()
Constructor

Mètodes

public void addColumna(
Columna pColumna)
Afegeix una Columna a la col·lecció de Columna de la ClauUnica
Paràmetres
pColumna - Columna a afegir a la ClauUnica

public java.util.Enumeration getColumnes()
Torna una Enumeration amb totes les Columna de la ClauUnica
Retorna
totes les Columna de la ClauUnica

public class **com.gentleware.apidemo.Columna**

Classe que representa una columna d'una taula

Constructors

public Columna(
String pNom,
String pTipusComplet,
String pRestriccions,
String pComentaris)
Constructor
Paràmetres
pNom - nom de la Columna
pTipusComplet - tipus complet de la Columna
pRestriccions - restriccions de la Columna inicials
pComentaris - els comentaris de la Columna inicials

public Columna(
String pNom)
Constructor només amb el nom
Paràmetres
pNom - nom de la Columna

Mètodes

public java.lang.String getNom()
Obtenir el nom de la Columna
Retorna
nom de la Columna

public void setNom(
String pNom)
Establir el nom de la Columna
Paràmetres
pNom - nom de la Columna

public java.lang.String getTipusComplet()
Obtenir el tipus complet de la Columna
Retorna
tipus complet de la Columna

public void setTipusComplet(
String pTipusComplet)
Establir el tipus complet de la Columna
Paràmetres
pTipusComplet - tipus complet de la Columna

public java.lang.String getRestriccions()
Obtenir les restriccions de la Columna
Retorna

restriccions de la Columna

public void setRestriccions(
String pRestriccions)
Establir les restriccions de la Columna
Paràmetres
pRestriccions - restriccions de la Columna

public java.lang.String addRestriccions(
String pRestriccions)
Afegir restriccions a una Columna. S'afegeixen si no existeixen, en cas contrari s'ignoren.
Paràmetres
pRestriccions - noves restriccions de la Columna
Retorna
restriccions de la Columna després de l'operació

public java.lang.String getComentaris()
Obtenir els comentaris de la Columna
Retorna
comentaris de la Columna

public void addComentaris(
String pComentaris)
Afegir els comentaris a una Columna
Paràmetres
pComentaris - els comentaris de la Columna per afegir

public abstract class **com.gentleware.apidemo.Configuracio**

Classe d'utilitat amb informació bàsica per l'aplicació, missatges, etc. Permet fer canvis ràpidament sobre característiques bàsiques de l'aplicació, canviar els textos dels missatges, etc.

Constructors

public Configuracio()

Mètodes

public static void inicialitzarTipusDades()
Crea la correspondència entre tipus UML i tipus de dades SQL

public static java.lang.String getTipusDades(
String pTipus)
Retorna el tipus de dades SQL que correspon al tipus UML especificat
Paràmetres
pTipus - tipus UML
Retorna
tipus de dades SQL que correspon al tipus UML especificat

public static byte ordreCardinalitatParticipant(
int pLower,
int pUpper)
Retorna l'ordre d'una Cardinalitat del Participant segons les seves cardinalitats màxima i mínima
Paràmetres
pLower - cardinalitat mínima del Participant
pUpper - cardinalitat màxima del Participant
Retorna
ordre de la Cardinalitat concreta del Participant

Camps

public static final CARD_ZERO_ENA

public static final CARD_U_ENA

public static final CARD_ZERO_U

public static final CARD_U

public static final MISS_TIPUS_DADES_NO_ESTANDARD

public static final MISS_CREAR_TAULA

public static final MISS_CLAU_PRIMARIA

public static final MISS_CLAU_FORANA

public static final MISS_REFERENCIES

public static final MISS_CLAU_UNICA

public static final MISS_CREAR_DISPARADOR

public static final MISS_QUAN

public static final MISS_NIVELL

public static final MISS_TEXT_INICIAL_DISPARADOR

public static final MISS_TEXT_CENTRAL_DISPARADOR

public static final MISS_TEXT_INIFINAL_DISPARADOR

public static final MISS_TEXT_FIFINAL_DISPARADOR

public static final TIPUS_DISPARADOR_ESBORRAT

public static final TIPUS_DISPARADOR_INSERCIO

public static final ValorsEtiquetatsAtribut
Modificadors de columnes posats en l'ordre en que es vulgui que siguin tractats. Provenen de valors etiquetats d'atributs. Els que no s'especifiquin en aquesta llista seran ignorats. Modificadors del tipus: LENGTH, PRECISION, SCALE. Modificadors del valor: DEFAULT.

public static final numValorsEtiquetatsAtribut

public static final EstereotipsAtribut
Restriccions de columnes posades en l'ordre en que es vulgui que siguin tractades. Provenen d'estereotips d'atributs. Les que no s'especifiquin en aquesta llista seran ignorades. Restriccions: NOT NULL, PRIMARY KEY.

public static final numEstereotipsAtribut

public class **com.gentleware.apidemo.Disparador**

Classe que representa un disparador d'una taula

Constructors

public Disparador(
String pNom,
String pTipus,
int pLimit,
Taula pPrimeraTaula,
Taula pSegonaTaula,
int pIndex)

Constructor

Paràmetres

pNom - nom del Disparador
pTipus - tipus del Disparador
pLimit - limit inicial per afegir al vector de límits
pPrimeraTaula - Taula del primer extrem de l'Associacio generadora del Disparador
pSegonaTaula - Taula del segon extrem de l'Associacio generadora del Disparador
pIndex - nombre de disparadors del mateix tipus si pPrimeraTaula = pSegonaTaula

Mètodes

public java.lang.String getNom()
Obtenir el nom del Disparador
Retorna
nom del Disparador

public java.lang.String getTipus()
Obtenir el tipus del Disparador
Retorna
tipus del Disparador

public void addLimit(
int pLimit)
Afegeix un limit al vector de límits si aquest límit no existeix
Paràmetres
pLimit - limit del vector de límits

public java.util.Enumeration getLimits()
Torna una Enumeration amb tots els límits del Disparador
Retorna
tots els límits del Disparador

public com.gentleware.apidemo.Taula getPrimeraTaula()
Obtenir la Taula del primer extrem de l'Associacio que ha generat el Disparador
Retorna
primera Taula del Disparador

public com.gentleware.apidemo.Taula getSegonaTaula()
Obtenir la Taula del segon extrem de l'Associacio que ha generat el Disparador
Retorna
segona Taula del Disparador

public int getIndex()
Obtenir l'índex del Disparador del mateix tipus contra les dues mateixes taules
Retorna
índex del Disparador del mateix tipus contra les dues mateixes taules

public java.lang.String generarText(
Taula pTaulaSobre)
Genera el text del Disparador
Paràmetres
pTaulaSobre - Taula sobre la que s'aplica el Disparador
Retorna
text del Disparador

public class **com.gentleware.apidemo.Estereotip**

Classe que representa un estereotip UML

Constructors

public Estereotip(
String pNom)
Constructor
Paràmetres
pNom - nom de l'Estereotip

Mètodes

public java.lang.String getNom()
Obtenir el nom de l'Estereotip
Retorna
nom de l'Estereotip

public class **com.gentleware.apidemo.Generalitzacio**

Classe que representa una generalització, una relació d'herència

Constructors

public Generalitzacio(
Classe pPare,
Classe pFill)
Constructor
Paràmetres
pPare - Classe pare a la Generalitzacio
pFill - Classe fill a la Generalitzacio

Mètodes

public com.gentleware.apidemo.Classe getPare()
Obtenir el pare de la Generalitzacio
Retorna
pare de la Generalitzacio

public com.gentleware.apidemo.Classe getFill()
Obtenir el fill de la Generalitzacio
Retorna
fill de la Generalitzacio

public interface **com.gentleware.apidemo.GrupColumnes**

Interfície amb les operacions bàsiques dels grups de columnes

Mètodes

public void addColumna(
Columna pColumna)
Afegeix una Columna a la col·lecció de Columna
Paràmetres
pColumna - Columna a afegir

public java.util.Enumeration getColumnes()
Torna una Enumeration amb totes les Columna de la Col·lecció
Retorna
totes les Columna de la Col·lecció

public class **com.gentleware.apidemo.Logic**

Classe que conté la lògica de l'aplicació. Genera codi DDL/SQL a partir de diagrames de classes UML

Mètodes

public static com.gentleware.apidemo.Logic crearLogic()
Constructor que només permet una instància de la classe

public java.lang.String generarCodi()
Retorna el codi generat corresponent al model del projecte actual. Recorre el model, l'obté, el transforma i torna el codi DDL/SQL generat

public class **com.gentleware.apidemo.Participant**

Classe que representa un participant situat en un extrem d'una associació UML

Constructors

public Participant(
Classe pClasse,
String pRol)
Constructor només amb classe i rol
Paràmetres
pClasse - Classe del Participant
pRol - rol del Participant

public Participant(
Classe pClasse,

String pRol,
Vector pCardinalitats)
Constructor
Paràmetres
pClasse - Classe del Participant
pRol - rol del Participant
pCardinalitats - totes les Cardinalitat del Participant a l'Associacio

Mètodes

public com.gentleware.apidemo.Classe getClasse()
Obtenir la Classe del Participant
Retorna
Classe del Participant

public java.lang.String getRol()
Obtenir el rol del Participant
Retorna
rol del Participant

public com.gentleware.apidemo.Cardinalitat addCardinalitat(
int pCardInf,
int pCardSup)
Crea una Cardinalitat i l'afegeix a la col·lecció de Cardinalitat
Paràmetres
pCardInf - cardinalitat inferior
pCardSup - cardinalitat superior
Retorna
nova Cardinalitat creada

public java.util.Enumeration getCardinalitats()
Torna una Enumeration amb totes les Cardinalitat del Participant
Retorna
totes les Cardinalitat del Participant

public java.util.Vector getCardinalitat()
Torna el Vector amb totes les Cardinalitat del Participant
Retorna
Vector amb totes les Cardinalitat del Participant

public class **com.gentleware.apidemo.TAssociacio**

Classe per manejar la col·lecció d'Associacio de l'aplicació

Constructors

public TAssociacio()
Constructor

Mètodes

public static void inicialitzar()
Inicialitza la col·lecció d'Associacio

public static java.lang.String assignarNom()
Dóna un nom consecutiu a les associacions

public static com.gentleware.apidemo.Associacio addAssociacio(
String pNom)
Crea una Associacio sense classe associativa i l'afegeix a la col·lecció
Paràmetres
pNom - nom d'Associacio
Retorna
nova Associacio creada

public static com.gentleware.apidemo.Associacio addAssociacio(
String pNom,
Classe pClasseAssociativa)
Crea una Associacio amb classe associativa i l'afegeix a la col·lecció
Paràmetres

pNom - nom d'Associacio
pClasseAssociativa - classe associativa de l'Associacio

Retorna

nova Associacio creada

public static com.gentleware.apidemo.Associacio getAssociacio(
String pNom)

Retorna l'Associacio de nom especificat

Paràmetres

pNom - Nom de l'Associacio

Retorna

associacio de nom pNom

public static java.util.Enumeration getAssociacions()

Torna una Enumeration amb totes les Associacio

Retorna

totes les Associacio

public class **com.gentleware.apidemo.Taula**

Classe que representa una taula

Constructors

public Taula(
String pNom)

Constructor

Paràmetres

pNom - nom de la Taula

Mètodes

public java.lang.String getNom()

Obtenir el nom de la Taula

Retorna

nom de la Taula

public com.gentleware.apidemo.Columna addColumna(
String pNom)

Crea una Columna i l'afegeix a la col·lecció de Columna de la Taula. Si la Columna ja existeix la retorna, si no retorna la nova Columna creada

Paràmetres

pNom - nom de la Columna

Retorna

nova Columna creada

public com.gentleware.apidemo.Columna getColumna(
String pColumna)

Obtenir una Columna de nom especificat de la Taula

Retorna

Columna de nom especificat de la Taula

public java.util.Enumeration getColumnes()

Torna una Enumeration amb totes les Columna de la Taula

Retorna

totes les Columna de la Taula

public void addClauPrimaria(
Columna columna)

Afegeix una Columna a la col·lecció de Columna de la ClauPrimaria. Si la Columna ja existeix no fa res

Paràmetres

columna - Columna a afegir a la col·lecció de Columna de la ClauPrimaria

public java.util.Enumeration getClauPrimaria()

Torna una Enumeration amb totes les Columna de la ClauPrimaria de la Taula

Retorna

totes les Columna de la ClauPrimaria de la Taula

public void addClauForana(
Columna pColumna,
Taula pTaula,
String clau)
Afegeix una Columna a una ClauForana de la Taula
Paràmetres
pColumna - Columna que es vol afegir a la ClauForana
pTaula - Taula contra la que va la ClauForana
pClau - String per indexar la Hashtable, normalment el nom de la Taula

public java.util.Enumeration getClauForana()
Torna una Enumeration amb totes les Columna de la ClauForana de la Taula
Retorna
totes les Columna de la ClauForana de la Taula

public boolean teClauForana(
String pTaula)
Retorna si existeix una ClauForana contra la Taula pTaula
Retorna
existeix una ClauForana contra la Taula pTaula

public int getComptadorDisparadors()
Incrementa el comptadorDisparadors de la Taula i el retorna
Retorna
comptadorDisparadors de la Taula incrementat prèviament

public com.gentleware.apidemo.Disparador addDisparadorEsborrat(
int pLimit,
Taula pPrimeraTaula,
Taula pSegonaTaula)
Afegeix o modifica un Disparador d'esborrat a la Taula amb el valor pLimit especificat. Si el disparador no existeix pel parell de Taula, s'afegeix amb el nom correlatiu. Si el disparador existeix pel parell de Taula, se li inclou el nou límit si no hi era
Paràmetres
pLimit - valor límit del Disparador
pPrimeraTaula - primera Taula per construir la join del codi del Disparador
pSegonaTaula - segona Taula per construir la join del codi del Disparador
Retorna
el Disparador creat o modificat

public com.gentleware.apidemo.Disparador addDisparadorInsercio(
int pLimit,
Taula pPrimeraTaula,
Taula pSegonaTaula)
Afegeix o modifica un Disparador d'inserció a la Taula amb el valor pLimit especificat. Si el disparador no existeix pel parell de Taula, s'afegeix amb el nom correlatiu. Si el disparador existeix pel parell de Taula, se li inclou el nou límit si no hi era
Paràmetres
pLimit - valor límit del Disparador
pPrimeraTaula - primera Taula per construir la join del codi del Disparador
pSegonaTaula - segona Taula per construir la join del codi del Disparador
Retorna
el Disparador creat o modificat

public com.gentleware.apidemo.Disparador addDisparador(
String pTipus,
int pLimit,
Taula pPrimeraTaula,
Taula pSegonaTaula)
Afegeix o modifica un Disparador del tipus especificat a la Taula amb el valor pLimit especificat. Si el disparador no existeix pel parell de Taula, s'afegeix amb el nom correlatiu.Si el disparador existeix pel parell de Taula, se li inclou el nou límit si no hi era
Paràmetres
pTipus - tipus del Disparador (esborrat o inserció)
pLimit - valor límit del Disparador
pPrimeraTaula - primera Taula per construir la join del codi del Disparador

pSegonaTaula - segona Taula per construir la join del codi del Disparador
Retorna
el Disparador creat o modificat

public java.util.Enumeration getDisparadors()
Torna una Enumeration amb tots els Disparadors de la Taula
Retorna
tots els Disparadors de la Taula

public void addClauUnica(
String pClau,
ClauUnica pClauUnica)
Afegeix una ClauUnica a la Taula
Paràmetres
pClau - clau de la ClauUnica
pClauUnica - ClauUnica que es vol afegir a la Taula

public java.util.Enumeration getClausUniques()
Torna una Enumeration amb totes les ClauUnica de la Taula
Retorna
totes les ClauUnica de la Taula

public class **com.gentleware.apidemo.TClasse**

Classe per manejar la col·lecció de Classe de l'aplicació

Constructors

public TClasse()
Constructor

Mètodes

public static void inicialitzar()
Inicialitza la col·lecció de Classe

public static com.gentleware.apidemo.Classe addClasse(
String pNom)
Crea una Classe i l'afegeix a la col·lecció
Paràmetres
pNom - nom de la Classe
Retorna
nova Classe creada

public static com.gentleware.apidemo.Classe getClasse(
String pNom)
Torna una Classe a partir del seu nom
Paràmetres
pNom - nom de la Classe
Retorna
Classe de nom especificat

public static java.util.Enumeration getClasses()
Torna una Enumeration amb totes les Classe
Retorna
totes les Classe

public class **com.gentleware.apidemo.TGeneralitzacio**

Classe per manejar la col·lecció de Generalitzacio de l'aplicació

Constructors

public TGeneralitzacio()
Constructor

Mètodes

public static void inicialitzar()
Inicialitza la col·lecció de Generalitzacio

public static com.gentleware.apidemo.Generalitzacio addGeneralitzacio(
Classe pPare,
Classe pFill)
Crea una Generalitzacio i l'afegeix a la col·lecció
Paràmetres
pPare - Classe pare a la Generalitzacio
pFill - Classe fill a la Generalitzacio
Retorna
nova Generalitzacio creada

public static java.util.Enumeration getGeneralitzacions()
Torna una Enumeration amb totes les Generalitzacio
Retorna
totes les Generalitzacio

public class **com.gentleware.apidemo.TTaula**

Classe per manejar la col·lecció de Taula de l'aplicació

Constructors

public TTaula()
Constructor

Mètodes

public static void inicialitzar()
Inicialitza la col·lecció de Taula

public static com.gentleware.apidemo.Taula addTaula(
String pNom)
Crea una Taula amb el nom especificat i l'afegeix a la col·lecció
Paràmetres
pNom - nom de la Taula

public static com.gentleware.apidemo.Taula getTaula(
String pNom)
Torna una Taula a partir del seu nom
Paràmetres
pNom - nom de la Taula
Retorna
Taula de nom especificat

public static java.util.Enumeration getTaules()
Torna una Enumeration amb totes les Taula
Retorna
totes les Taula

public class **com.gentleware.apidemo.ValorEtiquetat**

Classe que representa un valor etiquetat UML

Constructors

public ValorEtiquetat(
String pNom,
String pValor)
Constructor
Paràmetres
pNom - nom del ValorEtiquetat
pValor - valor del ValorEtiquetat

Mètodes

public java.lang.String getNom()
Obtenir el nom del ValorEtiquetat
Retorna
nom del ValorEtiquetat

public java.lang.String getValor()
Obtenir el valor del ValorEtiquetat
Retorna
valor del ValorEtiquetat

public void setValor(
String pValor)
Establir el valor del ValorEtiquetat
Paràmetres
pValor – valor del ValorEtiquetat

ANNEX B: Com instal·lar el plug-in

Abans de tot, dir que aquest plug-in s'ha desenvolupat sobre Windows XP. No s'ha provat que la seva compilació i funcionament sobre un altre entorn siguin correctes.

Primerament, cal descomprimir el fitxer ClassesUMLaTaules.zip en una ubicació, per exemple c:\. Aquest fitxer comprimit, a més dels fonts, ja conté el l'executable ClassesUMLaTaules\lib\classesumlataules.jar, que es pot copiar al subdirectori *\lib* del directori on s'ha instal·lat Poseidon (típicament *c:\Archivos de programa\Poseidon For UML PE 2.5.1*). Un cop fet això, se segueixen les instruccions d'instal·lació de plug-ins donades a l'apartat *Instal·lar* de la secció *4. Investigació i explicació de l'extensió de l'eina* per instal·lar-ho. Cal fer servir la següent *key* per instal·lar el plug-in:

MmqVk9yxZ7+jHm1UYgJVJG5jkAkDiv17pUpRxh7TbfXMUMCq8lKa5DVMO8QjLzf0
+R9wlTDHEvTEn3FhD23JMH/zCgjIwePBCp3czCNrYjCQXTot93ukKQhWD+72YBSv
FJ07cNtCNn7LAd00d0a53jtndAytmV7xdRRrJsYqWwY

Si es vol tornar a compilar el plug-in i es disposa de l'eina *ant*, es pot fer seguint les instruccions de l'apartat *Compilar: l'eina ant* de la secció *4. Investigació i explicació de l'extensió de l'eina*. El fitxer *build.xml* està preparat per l'ubicació de l'eina Poseidon comentada anteriorment; caldrà modificar-lo si no és aquí on està instal·lada l'eina Poseidon. Per compilar, simplement cal escriure *ant package* a la línia de comandes des de l'ubicació c:\ClassesUMLaTaules.