Refacció d'Ontologies: creació d'un *plugin* per a Protégé

Jordi Serra i Puig

EI

Felipe Geva i Urbano

Jordi Conesa i Caralt

7/01/2009

Agraïments

A la Núria, a l'Alba i en Bosco, a en Pau i la Laura, a la Clara i en Carles, a l'Aniol, per la paciència i la tolerància demostrades al llarg d'aquests estudis.

Als dos consultors, en Felipe i en Jordi. Més que consultors han estat per a mi uns bons companys de viatge en la descoberta que ha representat el treball.

Als desenvolupadors de programari lliure. Sense ells, aquest treball i molts altres no serien possibles.

Resum

Molta de la informació disponible en l'actual Web ha estat dissenyada per a ser interpretada i consumida per les persones, cosa que dificulta la resolució satisfactòria d'una pregunta com la següent:

> *"Voldria trobar un lloc per anar de vacances amb la meva parella durant aquestes festes de Nadal on puguem gaudir tranquil·lament de la natura i passejar prop del mar, no faci molta fred en aquesta època de l'any i puguem assistir a concerts de música clàssica."*

Els cercadors actuals proporcionen resultats relacionats amb el que es busca, però necessiten que l'usuari extregui, entre un mar d'informació, la que veritablement necessita.

La Web Semàntica pretén donar resposta a qüestions com l'anterior. Per a poder-ho fer, els documents disponibles en la Web han de ser interpretables tan per les persones com per les màquines. Aquest fet serà possible si els continguts es representen de la forma adequada. La Web Semàntica respon a aquest repte fent ús de les ontologies. Aquestes permeten especificar explícitament els conceptes d'un domini, descrivint les seves propietats i les restriccions que els afecten. Si una pàgina web descriu la informació utilitzant ontologies, els agents de *software* poden arribar a resoldre preguntes de tipus semàntic com l'anterior processant el seu contingut.

Les ontologies es poden crear des de zero, o bé aprofitar-ne de creades i ajustar-les a les necessitats d'un domini. En aquest cas, cal refinar-les –incorporant-hi el coneixement no present–, podar-les per tal d'eliminar el que no és rellevant, i realitzar un procés de refacció per tal d'obtenir-ne una que, tot conservant la semàntica de l'original, sigui més reduïda i fàcil d'interpretar.

El projecte centra els seus esforços en la darrera activitat: el *refactoring* d'ontologies. En concret, les expressades en **OWL**, el llenguatge per a ontologies pensat pel **W3C**. El resultat final del projecte ha estat la implementació en **Java** d'un *plugin* per a **Protégé** que permet l'execució d'operacions de *refactoring* sobre ontologies **OWL**. Part principal del *plugin* és un *framework* que ofereix el marc d'execució per a les operacions implementades i permet incorporar noves operacions sense haver-lo de modificar.

Índex de Continguts

Agraïments 1

Resum 2

1.- Introducció. 7

1.1.- Descripció i justificació del projecte. 7

1.2.- Presentació i Objectius. 7

1.2.1- Presentació. 7

1.2.2.- Objectius. 8

1.3.- Enfocament i mètode seguit. 9

1.4.- Planificació Temporal. 11

1.4.1.- Definició de la jornada laboral. Calendari del projecte. 11

1.4.3.- Planificació Temporal. Tasques i precedències. Diagrama de Gannt. 11

1.4.4.- Fites de Control. 15

1.4.5.- Canvis en la planificació inicial. 15

1.5.- Productes obtinguts. 16

1.6.- A partir d'ara. 17

2.- Fonaments. 18

2.1.- La Web Semàntica. Les ontologies en la Web Semàntica. 18

2.1.1.- La Web Semàntica. 18

2.1.2.- El concepte d'ontologia. 19

2.1.3.- Ontologies i Web Semàntica. 20

2.1.4.- Model de capes de la Web Semàntica. 20

2.2.- Llenguatges emprats en la Web Semàntica. 21

2.2.1.- XML i XML-Schema. 21

2.2.2.- RDF i RDF Schema. 25

2.2.2.1.- RDF. *26*

2.2.2.2.- RDF Schema. *28*

2.2.3.- OWL. 29

2.3.- La refacció i la seva aplicació a les ontologies. 34

2.3.1.- El concepte de refacció. 35

2.3.2.- La refacció d'ontologies. Catàleg d'operacions. 36

2.3.3.- El procés de refacció automàtica. 38

2.3.4.- Formulació dels algorismes de refactoring implementats. 39

2.3.4.1.- Implicit Subsets. *40*

2.3.4.2.- Generalize Realization. *41*

2.3.5.- Els algorismes implementats i OWL. 44

2.4.- La plataforma Protégé. 45

2.4.1.- L'API OWL. 45

2.4.2.- Desenvolupament de plugins. 47

3.- Disseny. 49

3.1.- Introducció. 49

3.2.- Disseny del *plugin*. 50

3.2.1.- Processos relacionats amb la càrrega d'operacions. 51

3.2.2.- Execució de les operacions de refactoring. 53

3.2.3.- Notificació a l'usuari dels missatges generats durant l'execució. 60

3.2.4.- Salvaguarda de resultats i models intermedis. 60

3.2.5.- Classes i interfícies auxiliars. 60

3.3.- Disseny de la interfície gràfica. 61
3.3.1.- Finestra principal. 61
3.3.2.- Altres components. 62
4.- Aspectes de la implementació. 63
4.1.- Introducció. 63
4.2.- El *framework* referencial. La classe controladora. 63
4.3.- Els algorismes de refacció. 64
4.3.1.- Implicit Subsets. 65
4.3.2.- Generalize Realization. 65
4.4.- La interfície d'usuari. 67
5.- Manuals d'usuari. 68
5.1.- Instal·lació i execució del *plugin*. 68
5.1.1.- Integració i execució des d'un entorn de desenvolupament. 69
5.2.- Manual d'ús. 70
5.2.1.- Configuració del plugin. 70
5.2.2.- Entorn de treball. Panell principal. 70
5.2.3.- Execució de les operacions. 72
5.2.4.- Emmagatzemar resultats. 73
5.2.5.- Guardar models intermedis. 73
5.2.6.- Emmagatzemar resultats finals i Canvi d'ontologia. 74
6.- Proves de funcionament. 75
6.1.- Prova de l'operació *Implicit Subsets*. 75
6.2.- Prova de l'operació *Generalize Realization*. 78
6.2.- Proves de salvaguarda. 87
7.- Conclusions i comentaris finals. 89
Glossari. 91
Bibliografia. 92

Índex de Figures

FIG. 1 - WEB SEMÀNTICA. ARQUITECTURA. 7
FIG. 2 - DIAGRAMA DE GANTT INICIAL 14
FIG. 3 - RELACIÓ DE TASQUES 14
FIG. 4 - DIAGRAMA DE GANTT MODIFICAT 16
FIG. 5 - UN EXEMPLE D'ONTOLOGIA. 19
FIG. 6 - MODEL DE CAPES DE LA WEB SEMÀNTICA. 21
FIG. 7 - CONCEPTES FONAMENTALS DE RDF. 26
FIG. 8 - RDF. CONCEPTES BÀSICS: REPRESENTACIÓ GRÀFICA. 26
FIG. 9 - RELACIONS DE CLASSE ENTRE OWL I RDF/RDFS. 30
FIG. 10 - OWL. PROPIETATS INVERSES. 32
FIG. 11 - REFACTORING D'ONTOLOGIES. CATÀLEG D'OPERACIONS. 37
FIG. 12 - TIPUS D'ENTITAT I TIPUS DE RELACIÓ. 39
FIG. 13 - OPERACIÓ *IMPLICIT SUBSETS*. 40
FIG. 14 - OPERACIÓ *PULL UP FIELD*. 41
FIG. 15 - OPERACIÓ *GENERALIZE REALIZATION*. 41
FIG. 16 - DIAGRAMA DE CLASSES DE L'API PROTÉGÉ-OWL. 46
FIG. 17 - EL *PLUGIN* I PROTÉGÉ 49
FIG. 18 - CASOS D'ÚS 50
FIG. 19 - OPERACIONS. FITXER DE CONFIGURACIÓ. 51
FIG. 20 - DIAGRAMA DE CLASSES. CLASSE CONTROLADORA I FINESTRA PRINCIPAL. 52
FIG. 21 - DIAGRAMA DE CLASSES. JERARQUIA REFACTORING OPERATION. 54
FIG. 22 - DIAGRAMA DE CLASSES. JERARQUIA REFACTORING OPPORTUNITY. 55
FIG. 23 – CLASSES OPPORTUNITYIMPLICITSUBSETS.JAVA I IMPLICITSUBSETS.JAVA 56
FIG. 24 - CLASSE REALIZATIONOWLPROPERTY 57
FIG. 25 - *GENERALIZE REALIZATION*. ESTRUCTURA DE DADES D'UN ELEMENT CANDIDAT. 57
FIG. 26 - CLASSE OPPORTUNITYGENERALIZEREALIZATION 58
FIG. 27 - CLASSE GENERALIZEREALIZATION 59
FIG. 23 - DIAGRAMA UML DE CLASSES AUXILIARS. 61
FIG. 29 - DISSENY GUI. FINESTRA PRINCIPAL. 61
FIG. 30 - DISSENY GUI. MENÚS CONTEXTUALS. 62
FIG. 31 - DISSENY GUI. FINESTRA SELECCIÓ DE FITXERS DE SALVAGUARDA. 62
FIG. 32 - INTEGRACIÓ DEL PLUGIN EN PROTÉGÉ. 68
FIG. 33 - RESULTAT D'INSTAL·LACIÓ DEL *PLUGIN* 69
FIG. 34 - INTEGRACIÓ AMB UN IDE. LLIBRERIES DE PROTÉGÉ. 69
FIG. 35 - INTEGRACIÓ AMB UN IDE. INDICACIÓ DEL DIRECTORI D'INSTAL·LACIÓ DE PROTÉGÉ. 70
FIG. 36 - OPCIONS BARRA D'EINES. 70
FIG. 37 - INCORPORACIÓ DE NOVES OPERACIONS EMPRANT PROTÉGÉ. 71
FIG. 38 - ENTORN DE TREBALL DEL PLUGIN. 71
FIG. 39 - EXECUCIÓ D'OPERACIONS. 72
FIG. 40 - INICI EMMAGATZEMAR *LOGS*. 73
FIG. 41 - SAVE RESULTS AS... 73
FIG. 42 - SAVE MODEL AS... 74
FIG. 43 - EMMAGATZEMAR RESULTATS FINALS. OBRIR NOU PROJECTE. 74
FIG. 44 - *IMPLICIT SUBSETS*. PROVA 1. 76
FIG. 45 - IMPLICIT SUBSETS. PROVA 2. 77
FIG. 46 - *GENERALIZE REALIZATION*. PROVA 1: ESTAT INICIAL DE L'ONTOLOGIA. 80
FIG. 47 - *GENERALIZE REALIZATION*. PROVA 2: ESTAT FINAL DE L'ONTOLOGIA. 81
FIG. 48 - *GENERALIZE REALIZATION*. PROVA 1: LOG D'EXECUCIÓ. 82
FIG. 49 - *GENERALIZE REALIZATION*. PROVA 2: LOG D'EXECUCIÓ. 84
FIG. 50 - *GENERALIZE REALIZATION*. PROVA 2: ESTAT INICIAL DE L'ONTOLOGIA. 85
FIG. 51 - *GENERALIZE REALIZATION*. PROVA 2: ESTAT FINAL DE L'ONTOLOGIA (1). 86
FIG. 52 - *GENERALIZE REALIZATION*. PROVA 2: ESTAT FINAL DE L'ONTOLOGIA (2). 87
FIG. 53 - PROVA PROCÉS DE SALVAGUARDA. 87
FIG. 54 - RESULTAT PROCÉS DE SALVAGUARDA. 88

Índex de Taules

TAULA 1 - TASQUES I PRECEDÈNCIES 13
TAULA 2 - FITES DE CONTROL 15
TAULA 3 - TASQUES I PRECEDÈNCIES. MODIFICACIONS. 16
TAULA 4 - RESTRICCIONS XML SCHEMA. 25
TAULA 5 - RESUM RDF/XML. 28
TAULA 6 – L'OPERACIÓ *IMPLICIT SUBSETS*. 40
TAULA 7 - *GENERALIZE REALIZATION*. CANVIS EN LES CARDINALITATS DE LES NOVES REALITZACIONS. 42
TAULA 8 - L'OPERACIÓ *GENERALIZE REALIZATION*. 44

1.- Introducció.

1.1.- Descripció i justificació del projecte.

La **refacció** –*refactoring*– és *un procés a través del qual es reescriu un material amb l'objectiu de millorar-ne tant la seva comprensió com l'estructura interna, amb el ferm propòsit de conservar-ne el significat i el comportament*[1].

En l'enginyeria del programari, aquesta tecnologia va aparèixer a principis dels anys noranta aplicada a la reestructuració del codi font i s'ha estès des de llavors a d'altres tipus de fonts: bases de dades, diagrames **UML**,... En el document referenciat en la bibliografia *Pruning and Refactoring Ontologies in the Development of Conceptual Schemas of Information Systems* es presenta un catàleg d'operacions de refacció aplicables a les ontologies i, també, la formalització algorísmica d'algunes d'aquestes operacions. El projecte a desenvolupar pretén implementar en el llenguatge **Java** un *plugin* per a **Protégé** amb l'objectiu de disposar d'un *framework* que ofereixi un marc general per a l'execució d'operacions de refacció sobre ontologies. A més, s'implementaran dues de les operacions indicades en el catàleg.

1.2.- Presentació i Objectius.

1.2.1- Presentació.

La Web Semàntica es pot considerar com una extensió de l'actual Web que pretén que el contingut dels documents disponibles en la xarxa sigui interpretable no només per les persones sinó també per les màquines. Tal com manifesta el creador de la Web, **Tim Berners-Lee**, en l'article *Semantic Web Road Map*[2]: "*La Web es va dissenyar com un espai d'informació, amb l'objectiu de ser útil no només per a la comunicació entre les persones, sinó que fos també possible la participació de les màquines, com a eines d'ajut, en el procés. Un dels obstacles més grans en la consecució d'aquest objectiu ha estat el fet que molta informació en la Web s'ha dissenyat pel consum humà,[...], amb la qual cosa l'estructura de les dades no es fa evident a una màquina que explora la xarxa. La Web Semàntica no pretén el mateix objectiu que la intel·ligència artificial –entrenar màquines per a que actuïn com a persones–, sinó que aposta per desenvolupar llenguatges que permetin expressar la informació de forma que pugui ser processada per una màquina*".

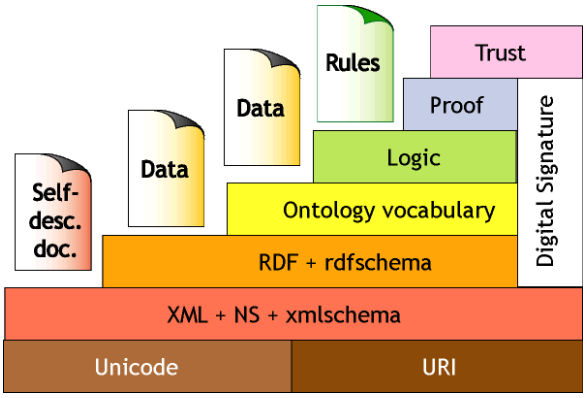


Fig. 1 - Web semàntica. Arquitectura.

Actualment, molts dels materials accessibles des de la xarxa han estat formatats a través de HTML. En ser aquest un llenguatge que només indica com es presenten els documents, no es disposa normalment d'informació referent al seu significat. Per tal d'aconseguir l'objectiu proposat, cal disposar de les eines i tecnologies que permetin afegir als documents la semàntica necessària, de forma que els distints agents puguin oferir als usuaris els resultats que aquests esperen. La infraestructura que suporta la Web Semàntica se sol representar a través d'una arquitectura de capes que va presentar l'any 2000 el mateix **Berners-Lee** en la *XML-Conference*.

En el gràfic de la figura 1, es pot observar l'ús de diferents llenguatges i tecnologies. Entre els distints components podem esmentar com a més interessants respecte al projecte desenvolupat, els següents:

- L'**XML** –*Extensible Markup Language*– com a llenguatge que permet estructurar la informació –dades i metadades– continguda en els documents de forma que pugui ser manipulada per una màquina i intercanviada entre equips heterogenis. No dota, però, de semàntica a les dades transmeses.

[1] Aquesta definició es pot trobar en Vikipèdia –http://ca.wikipedia.org/wiki/Refacció. S'ha cregut la més adequada al cas en no concretar el material sobre el que s'aplica el procés.

[2] Disponible a http://www.w3.org/DesignIssues/Semantic.html.

L'ús d'aquest llenguatge proporciona la base sintàctica sobre la que es recolzen els llenguatges de les capes superiors.

- **RDF** –*Resource Description Framework*– és bàsicament un model de dades que permet escriure sentències senzilles sobre recursos disponibles en la web. No permet, però, definir la semàntica pròpia d'un determinat domini. Per a fer-ho, cal emprar **RDFS** –*RDF Schema*. Aquest permet definir classes i propietats, relacions entre ambdós conceptes, relacions d'herència i realitzar restriccions sobre els distints elements del domini.

- Per sobre de la capa associada a l'**RDF**, se situen els llenguatges propis de les **ontologies**.

  Aquest és un concepte clau en el camp de la Web Semàntica. Una ontologia permet especificar explícita i formalment els conceptes relacionats amb un determinat domini, descrivint les seves propietats i les restriccions que els afecten. En la seva vessant tècnica, l'ontologia és un concepte propi del camp de la Intel·ligència Artificial i ha estat adoptat per la Web Semàntica ja que permet representar el coneixement associat a un domini i inferir-ne de nou gràcies a l'aplicació d'una sèrie de regles d'inferència. Podem dir que si una pàgina web descriu la seva informació utilitzant ontologies, els agents de *software* poden arribar a resoldre preguntes de tipus semàntic processant el seu contingut.

  L'expressivitat del **RDFS** per a descriure ontologies és molt reduïda. Per aquest motiu, el *Web Ontology Working Group* del **W3C** va pensar en d'altres llenguatges que fossin més adequats a aquesta finalitat. Entre aquests hi ha **OWL** –*Web Ontology Language*. Aquest llenguatge, basat en **RDF** i **RDFS**, es presenta en tres variants o dialectes: **OWL-Lite**, **OWL-DL** i **OWL-Full**. El primer és el menys expressiu, mentre que, com es pot suposar, el darrer és el que ho és més. Això el fa, però, inadequat per a realitzar raonaments de tipus automàtic.

  Per aquest motiu, en aquest projecte s'emprarà **OWL-DL**. Està basat en *Description Logics* –d'aquí el nom– i és adient per a l'automatització de la inferència.

Com ja s'ha indicat, una peça clau en l'èxit de la Web Semàntica són les ontologies. Aquestes es poden crear des de zero, o bé aprofitar una ontologia ja creada i ajustar-la al domini propi de l'esquema conceptual que s'està treballant. En aquest darrer cas, cal realitzar tres tasques per a adequar l'ontologia existent a la necessitat concreta:

- *Refinar l'ontologia base*, incorporant-hi el coneixement propi del domini concret que no hi és present.
- *Podar l'ontologia resultant* del pas anterior, eliminant els conceptes que no són rellevants en el domini que es treballa.
- *Efectuar una refacció de l'ontologia obtinguda de la poda*, per tal d'obtenir-ne una més reduïda i comprensible, conservant, però, la semàntica de l'original.

El projecte desenvolupat focalitza els seus esforços en la darrera de les activitats citades. Com s'ha esmentat en el subapartat anterior, en el document *Pruning and Refactoring Ontologies in the Development of Conceptual Schemas of Information Systems* es presenta un catàleg d'operacions de *refactoring* aplicables a les ontologies i es formalitzen els algorismes corresponents a 10 d'aquestes operacions.

El resultat final del projecte és la implementació en **Java** d'un *plugin* per a **Protégé** que permet l'execució d'operacions de refacció sobre ontologies expressades en **OWL**. El nucli central del *plugin* desenvolupat és un *framework* que ofereix un marc general per a l'execució de les operacions de *refactoring*. Dues de les operacions definides en el catàleg esmentat –*Implicit Subsets* i *Generalize Realization*– han estat incorporades al *plugin*. A més, el disseny del *framework* permet incorporar fàcilment noves operacions del catàleg sense haver de modificar el *framework* creat, gràcies a l'ús d'un fitxer de configuració –escrit en el mateix llenguatge **OWL**– i a les possibilitats que ofereix la classe **Class** de **Java** d'instanciar, a partir del nom d'una classe, objectes d'un determinat tipus en temps d'execució.

1.2.2.- Objectius.

El projecte s'encabeix en l'àrea de **PFC** anomenada *XML i Web semàntica*. Tenint en compte aquest fet, l'estudiant, al llarg del desenvolupament del projecte s'ha plantejat aconseguir els següents objectius bàsics:

- Conèixer els principals conceptes de la Web Semàntica i la seva arquitectura.
- Conèixer que és una ontologia, així com una de les eines d'edició i gestió d'ontologies més coneguda: **Protégé**.
- Conèixer els aspectes més importants dels llenguatges emprats en la Web Semàntica, **XML**, **RDF** i **OWL**, així com el perquè i el lloc d'ús de cadascun d'ells.

A més, el projecte concret desenvolupat ha comportat perseguir les següents finalitats:

- Aconseguir conèixer, a nivell conceptual, el conjunt de tècniques –refinament, poda i *refactoring*– que permeten reutilitzar una determinada ontologia per tal d'ajustar-la a les necessitats d'un domini concret.
- Conèixer el concepte de refacció, les operacions associades i la seva aplicació a les ontologies.
- Conèixer el catàleg d'operacions de *refactoring* que es poden aplicar a les ontologies.
- Estudiar amb profunditat dues de les operacions incloses en el catàleg: *Implicit Subsets* i *Generalize Realization*.
- Conèixer, per a cadascuna d'aquestes operacions, els criteris que permeten detectar quan es presenta una oportunitat per a realitzar-les de forma automàtica sobre una determinada ontologia.
- Conèixer com es desenvolupa en **Java** un *plugin* per a **Protégé**.
- Implementar en **Java** un *plugin* per a **Protégé** del tipus **Tab Widget** que permeti executar de forma automàtica operacions de *refactoring* sobre ontologies expressades en **OWL**. Com a component fonamental del *plugin* figura el *framework* que ofereix el marc general d'execució i que fa possible la incorporació en el futur de noves operacions a la base de *plugin* dissenyada.

La implementació del *plugin* exigeix un coneixement prou detallat del llenguatge **OWL** i de l'API de **Protégé-OWL** per a **Java**, per la qual cosa el seu estudi i coneixement, han esdevingut, també, altres objectius del projecte.

1.3.- Enfocament i mètode seguit.

Abans d'indicar quins han estat l'enfocament i el mètode seguits tan a l'hora de planificar el treball com en el moment de desenvolupar l'aplicació que es lliura, cal fer esment de la situació de partida de l'estudiant en l'instant d'iniciar el treball:

- El domini del projecte a desenvolupar era bastant desconegut per l'estudiant en l'inici del treball. Aquest fet dificultava avaluar correctament la grandària i la dificultat de les distintes tasques a realitzar en el transcurs del projecte i, per tant, la planificació realitzada podia ser inadequada, tenint present, a més, que en la **UOC** el projecte ha d'estar enllestit en un període de temps molt concret.
- En el transcurs dels seus estudis, havia cursat matèries –*Bases de Dades I* i *II*, *Enginyeria del Programari I*, *Enginyeria del Programari Orientat a l'Objecte*, *Criptografia* i *Compiladors II*– que li aportaven la formació bàsica suficient tan en el què es refereix a temes de modelització conceptual –sobretot mitjançant diagrames **ER** i **UML**– com al llenguatge **XML**.
- Disposava d'un coneixement prou ampli del llenguatge **Java** en el que havia desenvolupat el **TFC** corresponent a l'**ETIS**, estudis cursats també a la **UOC**.
- Tenia un coneixement bàsic –adquirit en les matèries *Intel·ligència Artificial I* i *II*– del què era una ontologia i dels sistemes de representació d'ontologies basats en marcs –*frames*. En canvi, conceptes com la Web semàntica, el *refactoring*, els llenguatges **RDF** i **OWL**, i la plataforma **Protégé** eren, com ja s'ha esmentat, prou desconeguts.

D'acord aquests antecedents, la primera tasca a realitzar fou la cerca d'informació relacionada amb aquells temes on es presentaven dèficits. Aquest procés de cerca va permetre arribar a les següents conclusions –comentades anteriorment de forma més extensa en el subapartat corresponent a la presentació:

- Un dels objectius de la Web Semàntica és que el contingut dels documents que se situen en Internet pugui ser interpretat per agents no humans distribuïts en la xarxa.
- Per tal d'aconseguir-ho, cal afegir als documents significat semàntic, la qual cosa és possible gràcies a l'ús d'ontologies: si la informació que conté el document és descrita emprant ontologies, un agent de programari podrà processar el seu contingut i, a partir d'ell, inferir coneixement.
- Per a descriure una ontologia cal un llenguatge amb l'expressivitat adequada. En el cas de la Web Semàntica, aquest llenguatge és l'**OWL**.
- Tot i que l'ontologia que descriu un determinat domini conceptual pot ser generada partint des de zero, sovint és millor adaptar-ne una ja existent, la qual cosa s'aconsegueix després d'efectuar seqüencialment tres processos: refinament, poda i refacció. Distintes parts d'aquests processos es poden realitzar automàticament.

Una vegada analitzada la informació obtinguda, tenint presents els objectius plantejats –sobretot que tot s'havia de concretar en un producte de programari–, l'enfocament inicial donat al treball es pot resumir en els següents punts:

- Com ja s'ha esmentat, una característica fonamental del *plugin* és la de permetre oferir un marc d'execució per a les operacions de *refactoring* sobre ontologies. En aquest sentit, es va pensar en crear una interfície o una classe abstracta per a definir els mètodes comuns a les classes que havien de representar les operacions de refacció. Aquests mètodes, en principi, havien de ser dos:
  - Un per a detectar l'oportunitat de l'operació, és a dir, quan l'operació és beneficiosa per a l'ontologia[3].
  - Un per a portar a la pràctica l'operació de *refactoring*.

  Tenint present aquesta línia de treball, cada operació del catàleg estaria representada per una classe que implementaria la interfície o bé derivaria de la classe abstracta, amb la qual cosa cada operació acabaria concretant l'oportunitat de refacció i com aplica aquesta.
- A més, aquest enfocament permetria en el futur la incorporació d'altres operacions definides en el catàleg.

  En aquest sentit, l'ús d'un fitxer de configuració havia de facilitar la declaració de les classes associades a les distintes operacions de refacció implementades, de forma que el *plugin* podia saber en tot moment quines eren les operacions vàlides –les definides en el fitxer– i quina classe implementava cadascuna.

  La idea inicial era escriure aquest fitxer emprant alguns dels llenguatges a treballar al llarg del projecte.

Aquesta idea inicial es va acabar concretant en el transcurs de la fase d'anàlisi[4], de forma que l'enfocament final ha estat el següent:

- Representar el concepte genèric operació de refacció a través d'una classe abstracta que defineix el mètode abstracte que hauran d'implementar les classes hereves encarregades d'especialitzar les distintes operacions de *refactoring* definides en el catàleg. El mètode esmentat, sense paràmetres, centralitza les dues tasques a desenvolupar en qualsevol operació:
  - la detecció de l'oportunitat
  - i l'execució concreta.
- Per a portar a la pràctica l'execució d'una operació concreta, les classes hereves d'aquesta classe abstracta utilitzen els serveis d'una altra classe que disposa d'un mètode amb atributs específics per a cada operació.
- Les oportunitats de *refactoring* es representen mitjançant una interfície que defineix el mètode a especialitzar per les classes que la implementen. Cadascuna d'aquestes classes s'associa a una operació concreta de refacció.
- El *plugin* disposa d'una classe controladora que forma, conjuntament amb la classe abstracta i la interfície anteriorment esmentades, el *framework* que ofereix el marc general d'execució.

  Com ja s'ha indicat, el conjunt d'operacions de *refactoring* disponibles està definit en un fitxer de configuració. En la inicialització del *plugin*, la classe controladora carrega des del fitxer de configuració –una ontologia molt senzilla escrita en **OWL**– les operacions disponibles i, durant l'execució, es creen objectes de les classes pertinents.

  Aquesta forma de procedir permet anar afegint operacions del catàleg al *plugin* sense haver de modificar el *framework* creat.
- També s'hauran de crear les classes d'utilitat necessàries per a poder obtenir la informació continguda en les ontologies que s'analitzen.

[3] Es considera (Conesa, 2008) que una operació és oportuna quan la seva aplicació redueix la grandària de l'ontologia.

[4] A l'hora de concretar l'enfocament definitiu, va ser de gran ajut un prototip escrit en **Java** per a refaccionar ontologies descrites en **UML** que va proporcionar el professor responsable de l'Àrea, en Jordi Conesa.

La metodologia seguida en el desenvolupament ha estat la clàssica: *recollir requeriments*, *analitzar*, *dissenyar*, *implementar* i *provar*, efectuant aquestes tasques de forma seqüencial. La línia de treball seguida es concreta en la planificació que es comenta tot seguit.

1.4.- Planificació Temporal.

Al final d'aquest apartat es mostren els diagrames de **Gantt** corresponents a la planificació del projecte. En lliurar el Pla de Treball es va realitzar una planificació inicial, tenint presents la jornada laboral i el calendari que s'indiquen tot seguit. Posteriorment, aquesta planificació inicial es va modificar lleugerament, mantenint, però, la jornada laboral i el calendari.

1.4.1.- Definició de la jornada laboral. Calendari del projecte.

Inicialment, es va considerar que, com a mitjana al llarg del semestre, es podrien dedicar dues diàries al projecte durant els set dies de la setmana. Aquesta previsió ha resultat certa en la pràctica.

Es va preveure iniciar el projecte l'endemà de la data d'inici del semestre, el dia 19 de setembre del 2008. En la planificació, es va considerar com a data de lliurament la marcada en el Pla Docent: el 7 de gener del 2009. Com a data de tancament es va preveure la corresponent a la finalització del debat virtual, el dia 30 de gener del 2009.

Segons el marc indicat, s'ha disposat de cent sis dies laborables tenint en compte que s'han considerat festives les següents dates:

- 25 de Desembre: Nadal
- 26 de Desembre: Sant Esteve
- 1 de Gener: Cap d'Any
- 6 de Gener: Reis

Per tant, es va planificar considerant que es disposaria de 212 hores per a dedicar al projecte.

1.4.3.- Planificació Temporal. Tasques i precedències. Diagrama de Gannt.

La planificació temporal es va realitzar tenint presents, entre altres qüestions, les proves d'avaluació continuada a lliurar en el transcurs del semestre. Coincidint amb les dates de lliurament es van marcar fites de control (veure Taula 2). Aquesta planificació s'ha complert de forma prou satisfactòria al llarg del semestre.

Gairebé totes les feines s'han desenvolupat una rere l'altra. Només ha existit un cert paral·lelisme en el que es refereix a l'elaboració del Pla de Treball i a la cerca d'informació i a la posterior anàlisi, així com en l'elaboració de la documentació final que ha començat mentre s'implementava el segon algorisme. Per tal de poder absorbir lleugers retards en el desenvolupament del **PFC**, es va preveure poder disposar d'un cert marge de maniobra entre el lliurament final i l'acabament de la darrera tasca a realitzar abans d'aquest, l'*Elaboració de la Presentació*.

La relació de les tasques principals desenvolupades és la següent:

- **Pla de Treball:**

  En aquesta fase s'ha elaborat el document en què s'ha establert l'abast del projecte, els requisits funcionals i tècnics, així com la temporalització. S'ha considerat que la data d'inici era el 27 de setembre –data de la trobada presencial– i la final, el dia de lliurament del pla.

- **Cerca i obtenció d'informació:**

  Es tractava de cercar i recollir informació sobre els distints conceptes i/o eines a utilitzar en el transcurs del treball:

  - Arquitectura i característiques de la Web Semàntica.
  - Concepte d'ontologia i llenguatges associats encara no coneguts: **RDF** i **OWL**.
  - Operacions a realitzar sobre les ontologies.
  - Concepte de *refactoring* i la seva aplicació a les ontologies.
  - **Protégé**: descàrrega i instal·lació de l'eina, documentació sobre l'API **OWL** i el desenvolupament de *plugins*, i la seva integració amb els IDE **Eclipse** i **NetBeans**.
  - Algorismes a implementar.

  Part d'aquesta fase es realitzà en paral·lel amb l'anterior, ja que havia de servir per a escollir i començar a centrar el treball.

- **Estudi i Anàlisi de la informació:**

  Una vegada acabada la fase de cerca i obtenció, calia analitzar la informació recollida per tal d'acabar de definir el treball a realitzar. Per altra banda, calia estudiar-la detingudament ja que hi havia llenguatges i eines que eren noves pel desenvolupador. En aquesta fase, calia tenir cura sobretot dels següents aspectes:

  - L'estudi de **RDF** i **OWL**.
  - L'anàlisi detallat dels algorismes a implementar.
  - L'estudi de **Protégé**, centrant-lo en la gestió de ontologies expressades en **OWL**, l'anàlisi de l'API per a **Java** de **OWL** i el desenvolupament de *plugins*. En aquest darrer cas, es tractava d'implementar un petit *plugin* i provar com s'integrava amb el *framework*.

  Tenint en compte el volum de feina associat a aquesta tasca, es va pensar en una durada de dues setmanes i mitja.

- **Anàlisi i Disseny:**

  Tenint en compte les característiques del projecte i la feina ja realitzada, es va preveure per a aquesta fase una durada de gairebé una setmana, considerant les següents subtasques:

  - L'anàlisi i disseny del programa a nivell de processos.
  - El disseny de la interfície gràfica.

- **Implementació:**

  Aquesta fase tenia una durada de més de 7 setmanes. Es va considerar que calia realitzar les següents activitats:

  - Programar el marc referencial pensat per a acollir l'execució dels algorismes de refacció.
  - Implementar els dos algorismes corresponents a les dues operacions de refacció esmentades anteriorment –*Implicit Subsets* i *Generalize Realization*.
  - La implementació de la interfície gràfica.

  Dins aquesta tasca es va marcar una fita de control coincidint amb el lliurament de la segona prova d'Avaluació continuada. Si tot transcorria d'acord amb el pla previst, era d'esperar que s'haguessin acabat les implementacions del marc referencial i del primer algorisme citat –la qual cosa va esdevenir certa.

- **Proves d'integració:**

  En aquesta fase es pretenia acabar de depurar l'aplicació i preparar-la per al lliurament final. Havia d'estar acabada el 28 de novembre per tal de dedicar les darreres setmanes a l'elaboració de la memòria i la presentació.

- **Elaboració de la memòria:**

  Es va pensar dedicar unes tres setmanes a l'elaboració de la documentació final, solapant-se en la primera setmana amb les proves finals. La idea era que quedés gairebé un mes dedicat plenament a la memòria i a la presentació.

- **Elaboració de la presentació:**

  Tasca que se solapa parcialment amb l'anterior. El dia 7 de gener s'ha de lliurar el producte final juntament amb la memòria i la presentació.

- **Debat:**

  Aquesta fase s'inicia un cop s'ha lliurat el projecte i se li ha assignat la durada prevista en el Pla Docent. És la darrera fase del projecte, prèvia al seu tancament.

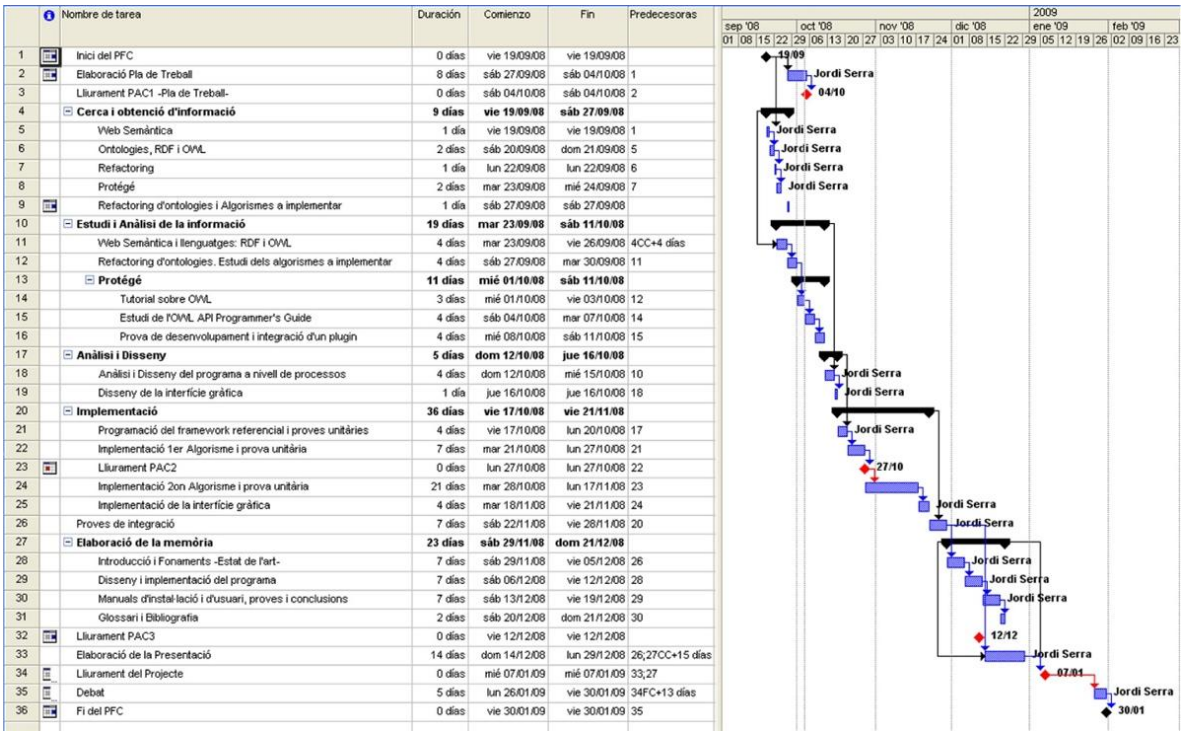
En el primer diagrama de **Gantt** es pot observar com tot just abans de l'inici de la fase *Elaboració de la presentació*, s'ha situat una fita de control lligada a la presentació de la PAC3. La idea era que en aquesta data estigués acabada la implementació i s'hagués avançat en l'elaboració de la memòria.

La taula següent mostra el detall de les distintes tasques a realitzar així com les dependències entre elles.

Codi Activitat	Nom	Precedències
01	**Inici del PFC**	
02	**Elaboració del Pla de Treball**	01
03	**Cerca i obtenció d'informació**	
03.01	Web Semàntica	01
03.02	Ontologies, RDF i OWL	03.01
03.03	*Refactoring*	03.02
03.04	Protégé	03.03
03.05	*Refactoring* d'ontologies i Algorismes a implementar	
04	**Estudi i anàlisi de la informació**	
04.01	Web Semàntica i llenguatges: RDF i OWL	03.02
04.02	*Refactoring* d'ontologies i Algorismes a implementar	04.01
04.03	Protégé	
04.03.01	Tutorial sobre OWL	04.02
04.03.02	Estudi de l'API OWL	04.03.01
04.03.03	Tutorial sobre desenvolupament de *plugins*	04.03.02
05	**Anàlisi i disseny**	
05.01	Processos	04
05.02	Interfície gràfica	05.01
06	**Implementació**	
06.01	Programació del marc referencial i proves unitàries	05
06.02	Implementació del 1er algorisme	06.01
06.03	Implementació del 2on algorisme	06.02
06.04	Implementació de la interfície gràfica	06.03
07	**Proves de integració**	06
08	**Elaboració de la memòria**	
08.01	Introducció i fonaments –estat de l'art–	07
08.02	Dissenys i implementació del programa	08.01
08.03	Manuals d'usuari, proves i conclusions	08.02
08.04	Glossari i bibliografia	08.03
09	**Elaboració de la presentació**	07
10	**Presentació del Projecte i Debat**	08, 09
11	**Fi del Projecte**	10

Taula 1 - Tasques i precedències

La figura 2 mostra la planificació que es va proposar inicialment per al projecte.



	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	Inici del PFC	0 días	vie 19/09/08	vie 19/09/08	
2	Elaboració Pla de Treball	8 días	sáb 27/09/08	sáb 04/10/08	1
3	Lliurament PAC1 -Pla de Treball-	0 días	sáb 04/10/08	sáb 04/10/08	2
4	**Cerca i obtenció d'informació**	**9 días**	**vie 19/09/08**	**sáb 27/09/08**	
5	Web Semàntica	1 día	vie 19/09/08	vie 19/09/08	1
6	Ontologies, RDF i OWL	2 días	sáb 20/09/08	dom 21/09/08	5
7	Refactoring	1 día	lun 22/09/08	lun 22/09/08	6
8	Protégé	2 días	mar 23/09/08	mié 24/09/08	7
9	Refactoring d'ontologies i Algorismes a implementar	1 día	sáb 27/09/08	sáb 27/09/08	
10	**Estudi i Anàlisi de la informació**	**19 días**	**mar 23/09/08**	**sáb 11/10/08**	
11	Web Semàntica i llenguatges: RDF i OWL	4 días	mar 23/09/08	vie 26/09/08	4CC+4 días
12	Refactoring d'ontologies. Estudi dels algorismes a implementar	4 días	sáb 27/09/08	mar 30/09/08	11
13	**Protégé**	**11 días**	**mié 01/10/08**	**sáb 11/10/08**	
14	Tutorial sobre OWL	3 días	mié 01/10/08	vie 03/10/08	12
15	Estudi de l'OWL API Programmer's Guide	4 días	sáb 04/10/08	mar 07/10/08	14
16	Prova de desenvolupament i integració d'un plugin	4 días	mié 08/10/08	sáb 11/10/08	15
17	**Anàlisi i Disseny**	**5 días**	**dom 12/10/08**	**jue 16/10/08**	
18	Anàlisi i Disseny del programa a nivell de processos	4 días	dom 12/10/08	mié 15/10/08	10
19	Disseny de la interfície gràfica	1 día	jue 16/10/08	jue 16/10/08	18
20	**Implementació**	**36 días**	**vie 17/10/08**	**vie 21/11/08**	
21	Programació del framework referencial i proves unitàries	4 días	vie 17/10/08	lun 20/10/08	17
22	Implementació 1er Algorisme i prova unitària	7 días	mar 21/10/08	lun 27/10/08	21
23	Lliurament PAC2	0 días	lun 27/10/08	lun 27/10/08	22
24	Implementació 2on Algorisme i prova unitària	21 días	mar 28/10/08	lun 17/11/08	23
25	Implementació de la interfície gràfica	4 días	mar 18/11/08	vie 21/11/08	24
26	Proves de integració	7 días	sáb 22/11/08	vie 28/11/08	20
27	**Elaboració de la memòria**	**23 días**	**sáb 29/11/08**	**dom 21/12/08**	
28	Introducció i Fonaments -Estat de l'art-	7 días	sáb 29/11/08	vie 05/12/08	26
29	Disseny i implementació del programa	7 días	sáb 06/12/08	vie 12/12/08	28
30	Manuals d'instal·lació i d'usuari, proves i conclusions	7 días	sáb 13/12/08	vie 19/12/08	29
31	Glossari i Bibliografia	2 días	sáb 20/12/08	dom 21/12/08	30
32	Lliurament PAC3	0 días	vie 12/12/08	vie 12/12/08	
33	Elaboració de la Presentació	14 días	dom 14/12/08	lun 29/12/08	26;27CC+15 días
34	Lliurament del Projecte	0 días	mié 07/01/09	mié 07/01/09	33;27
35	Debat	5 días	lun 26/01/09	vie 30/01/09	34FC+13 días
36	Fi del PFC	0 días	vie 30/01/09	vie 30/01/09	35

Fig. 2 - Diagrama de Gantt inicial

En la figura 3 es pot comprovar millor la planificació temporal i l'assignació de recursos.

Diagrama de Gantt

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
1	Inici del PFC	0 días	vie 19/09/08	vie 19/09/08		
2	Elaboració Pla de Treball	8 días	sáb 27/09/08	sáb 04/10/08	1	Jordi Serra
3	Lliurament PAC1 -Pla de Treball-	0 días	sáb 04/10/08	sáb 04/10/08	2	
4	**Cerca i obtenció d'informació**	**9 días**	**vie 19/09/08**	**sáb 27/09/08**		
5	Web Semàntica	1 día	vie 19/09/08	vie 19/09/08	1	Jordi Serra
6	Ontologies, RDF i OWL	2 días	sáb 20/09/08	dom 21/09/08	5	Jordi Serra
7	Refactoring	1 día	lun 22/09/08	lun 22/09/08	6	Jordi Serra
8	Protégé	2 días	mar 23/09/08	mié 24/09/08	7	Jordi Serra
9	Refactoring d'ontologies i Algorismes a implementar	1 día	sáb 27/09/08	sáb 27/09/08		
10	**Estudi i Anàlisi de la informació**	**19 días**	**mar 23/09/08**	**sáb 11/10/08**		**Jordi Serra**
11	Web Semàntica i llenguatges: RDF i OWL	4 días	mar 23/09/08	vie 26/09/08	4CC+4 días	
12	Refactoring d'ontologies. Estudi dels algorismes a implementar	4 días	sáb 27/09/08	mar 30/09/08	11	
13	**Protégé**	**11 días**	**mié 01/10/08**	**sáb 11/10/08**		
14	Tutorial sobre OWL	3 días	mié 01/10/08	vie 03/10/08	12	
15	Estudi de l'OWL API Programmer's Guide	4 días	sáb 04/10/08	mar 07/10/08	14	
16	Prova de desenvolupament i integració d'un plugin	4 días	mié 08/10/08	sáb 11/10/08	15	
17	**Anàlisi i Disseny**	**5 días**	**dom 12/10/08**	**jue 16/10/08**		
18	Anàlisi i Disseny del programa a nivell de processos	4 días	dom 12/10/08	mié 15/10/08	10	Jordi Serra
19	Disseny de la interfície gràfica	1 día	jue 16/10/08	jue 16/10/08	18	Jordi Serra
20	**Implementació**	**36 días**	**vie 17/10/08**	**vie 21/11/08**		
21	Programació del framework referencial i proves unitàries	4 días	vie 17/10/08	lun 20/10/08	17	Jordi Serra
22	Implementació 1er Algorisme i prova unitària	7 días	mar 21/10/08	lun 27/10/08	21	
23	Lliurament PAC2	0 días	lun 27/10/08	lun 27/10/08	22	
24	Implementació 2on Algorisme i prova unitària	21 días	mar 28/10/08	lun 17/11/08	23	
25	Implementació de la interfície gràfica	4 días	mar 18/11/08	vie 21/11/08	24	Jordi Serra
26	Proves de integració	7 días	sáb 22/11/08	vie 28/11/08	20	Jordi Serra
27	**Elaboració de la memòria**	**23 días**	**sáb 29/11/08**	**dom 21/12/08**		
28	Introducció i Fonaments -Estat de l'art-	7 días	sáb 29/11/08	vie 05/12/08	26	Jordi Serra
29	Disseny i implementació del programa	7 días	sáb 06/12/08	vie 12/12/08	28	Jordi Serra
30	Manuals d'instal·lació i d'usuari, proves i conclusions	7 días	sáb 13/12/08	vie 19/12/08	29	Jordi Serra
31	Glossari i Bibliografia	2 días	sáb 20/12/08	dom 21/12/08	30	
32	Lliurament PAC3	0 días	vie 12/12/08	vie 12/12/08		
33	Elaboració de la Presentació	14 días	dom 14/12/08	lun 29/12/08	26;27CC+15 días	Jordi Serra
34	Lliurament del Projecte	0 días	mié 07/01/09	mié 07/01/09	33;27	
35	Debat	5 días	lun 26/01/09	vie 30/01/09	34FC+13 días	Jordi Serra
36	Fi del PFC	0 días	vie 30/01/09	vie 30/01/09	35	

Fig. 3 - Relació de Tasques

1.4.4.- Fites de Control.

Finalment, les fites de control previstes –gairebé totes ja realitzades– es recullen en la taula següent.

Data	Fita de Control	Participants
09-09-2008	Inici del projecte	Consultor i alumne
04-10-2008	Lliurament Pla de Treball – PAC1	Consultor i alumne
27-10-2008	PAC2	Consultor i alumne
12-12-2008	PAC3	Consultor i alumne
27-12-2008	Lliurament pre-memòria	Consultor i alumne
07-01-2009	Lliurament del projecte	Consultor i alumne
30-01-2009	Final del projecte	Consultor i alumne

Taula 2 - Fites de control

1.4.5.- Canvis en la planificació inicial.

Coincidint amb el lliurament de la segona prova d'avaluació continuada es varen realitzar alguns petits canvis en pla de projecte inicialment previst. En concret, es pretenia avançar l'inici de l'elaboració de la memòria. Prevista inicialment per a després de les proves d'integració, es tractava de començar-la en paral·lel a la implementació dels segon algorisme. La taula 3 recull –marcats en color taronja– els canvis realitzats, i tot seguit es mostra el diagrama de Gantt resultat de les modificacions introduïdes.

Referent a les fites de control, cal dir que no hi ha hagut cap canvi.

Codi Activitat	Nom	Precedències
01	**Inici del PFC**	
02	**Elaboració del Pla de Treball**	01
03	**Cerca i obtenció d'informació**	
03.01	Web Semàntica	01
03.02	Ontologies, RDF i OWL	03.01
03.03	*Refactoring*	03.02
03.04	Protégé	03.03
03.05	*Refactoring* d'ontologies i Algorismes a implementar	
04	**Estudi i anàlisi de la informació**	
04.01	Web Semàntica i llenguatges: RDF i OWL	03.02
04.02	*Refactoring* d'ontologies i Algorismes a implementar	04.01
04.03	Protégé	
04.03.01	Tutorial sobre OWL	04.02
04.03.02	Estudi de l'API OWL	04.03.01
04.03.03	Tutorial sobre desenvolupament de *plugins*	04.03.02
05	**Anàlisi i disseny**	
05.01	Processos	04
05.02	Interfície gràfica	05.01
06	**Implementació**	
06.01	Programació del marc referencial i proves unitàries	05
06.02	Implementació del 1er algorisme	06.01
06.03	Implementació del 2on algorisme	06.02
06.04	Implementació de la interfície gràfica	06.03
07	**Proves de integració**	06
08	**Elaboració de la memòria**	

Codi Activitat	Nom	Precedències
08.01	Introducció i fonaments –estat de l'art–	08.03
08.02	Dissenys i implementació del programa	06.02
08.03	Manuals d'usuari, proves i conclusions	08.02
08.04	Glossari i bibliografia	06.02
09	**Elaboració de la presentació**	07
10	**Presentació del Projecte i Debat**	08, 09
11	**Fi del Projecte**	10

Taula 3 - Tasques i precedències. Modificacions.

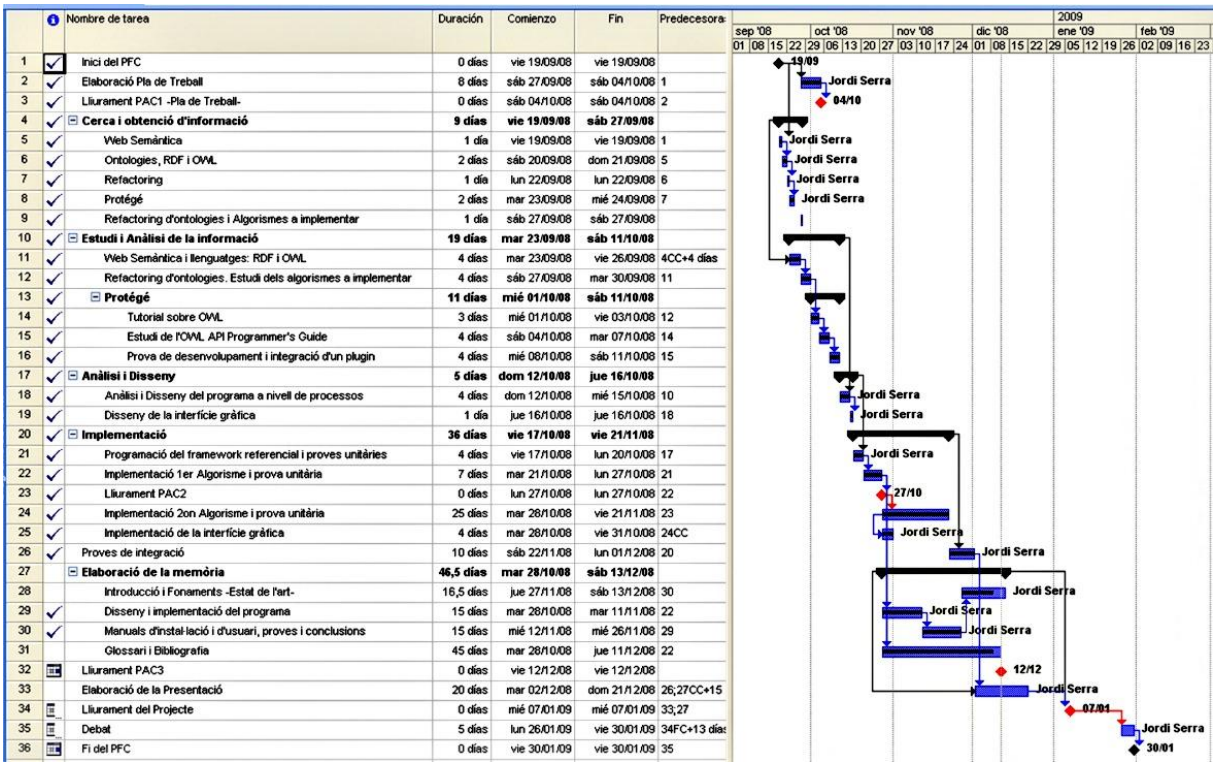


Fig. 4 - Diagrama de Gantt modificat

1.5.- Productes obtinguts.

A més del present document i de la presentació que l'acompanya, el resultat del **PFC** és un producte de programari, concretament un *plugin* per a **Protégé**: ***Refactoring Plugin Protégé***. Com ja s'ha indicat, és un *plugin* que permet realitzar automàticament operacions de refacció sobre ontologies expressades en **OWL**. L'usuari pot executar tot el conjunt d'operacions o escollir quines vol efectuar en un determinat instant. Té la possibilitat, també, d'emmagatzemar els models intermedis que es van obtenint al llarg del procés o els *logs* d'aquest. A més, la interacció amb l'aplicació es realitza de forma gràfica en un típic entorn de finestres.

El producte de programari es troba en el fitxer **RefactoringPluginProtege.zip**. Dins d'aquest es troba la carpeta **edu.uoc.pfc.semanticweb.refactoring** amb el següent contingut:

- El fitxer **RefactoringPluginProtege.jar** que conté les classes compilades conjuntament amb l'arxiu **MANIFEST.MF**.
- L'arxiu de configuració **refactoring-operation.owl** que defineix les operacions de refactoring possibles.
- El fitxer **plugin.properties** que declara el nom i les dependències del *plugin*.

A més, es lliuren les fonts i la documentació de les classes: carpetes **src** i **javadoc**.

1.6.- A partir d'ara.

Els capítols que segueixen constitueixen la raó de ser d'aquest document. El segon capítol tracta dels conceptes que fonamenten el producte desenvolupat. Tot i que amb una extensió reduïda per tal de mantenir el document en els límits demanats, es tracta sobre la Web Semàntica i els llenguatges emprats en ella –en especial **RDF**, **RDFS** i **OWL**–, es parla sobre el *refactoring* i la seva aplicació a la refacció d'ontologies i s'explica què és la plataforma **Protégé** i com es poden desenvolupar *plugins* per a ella.

El capítol 3 parla del disseny de l'aplicació: introdueix quins han estat els principis del disseny, esmenta com s'ha dissenyat el programa a nivell de processos i acaba comentant les bases del disseny de la interfície gràfica. El capítol següent tracta dels aspectes més destacables de la implementació realitzada: com s'han programat els temes relacionats amb el *framework*, com s'han implementat la classe controladora, els algorismes de *refactoring* i, finalment, la interfície d'usuari.

El capítol 5 està dedicat als manuals d'usuari. Explica com instal·lar i executar el *plugin*, i incorpora un petit manual d'ús de l'aplicació. El capítol 6 mostra una sèrie de proves realitzades –a més de les que es poden observar en el manual d'usuari del capítol anterior. Finalment, abans d'un breu glossari i de la bibliografia emprada, el capítol 7 presenta les conclusions i els comentaris finals. El codi implementat es lliura, com a annex, en un document apart.

2.- Fonaments.

2.1.- La Web Semàntica. Les ontologies en la Web Semàntica.

Actualment, els cercadors com Google o Yahoo molt sovint treballen mitjançant l'ús de paraules clau. Aquest fet ocasiona un conjunt de problemes quan una persona cerca determinada informació (Antoniou & Van Harmelen, 2008):

- Es pot arribar a obtenir molta informació però poc precisa: conjuntament amb les pàgines que realment interessen, se'n troben moltes que són irrellevants.
- A vegades no s'obté aquella informació que és realment important en el context del què s'està cercant.
- Els resultats obtinguts són molt sensibles al vocabulari emprat, la qual cosa dificulta trobar el que es busca quan la terminologia del document no coincideix amb les paraules emprades en la cerca.
- Els resultats poden ser pàgines individuals quan la informació que realment interessa es troba repartida en varis documents, per obtenir els quals calen varies cerques.

A més, tot i que la cerca proporcioni resultats relacionats amb el que es busca, cal la intervenció final de l'usuari per tal d'extreure d'ells la informació que realment desitja. Cal dir, també, que els resultats que actualment proporcionen els cercadors són de difícil lectura per part d'altres agents de programari.

Per tal de donar resposta a totes aquestes qüestions, neix la Web Semàntica: es tracta de representar el contingut dels documents de forma que faciliti el seu processament mitjançant agents de *software*. Com s'esmenta en el text referenciat en la bibliografia *A Semantic Web Primer* dels autors citats en el primer paràgraf "*És important entendre que la Web Semàntica no serà una nova via d'informació paral·lela a l'actual Web sinó que aquesta evolucionarà gradualment cap el nou paradigma*".

2.1.1.- La Web Semàntica.

D'acord amb el que s'acaba d'esmentar i tal com s'indica en una altra de les referències bibliogràfiques (Guia breve de Web Semántica, 2008), *la Web Semàntica es una extensió de l'actual Web que pretén dotar-la de més significat –és a dir, de semàntica– de forma que els usuaris, amb l'ajut d'agents de programari, puguin resoldre les cerques d'informació de forma més ràpida, senzilla i, sobretot, més precisa d'acord amb les seves necessitats*.

La introducció de semàntica en la Web permet que els agents de *software* puguin processar el contingut dels documents, raonar a partir d'ell i obtenir els resultats que fan possible atendre millor les demandes d'informació que realitzen els usuaris. Aquests poden desitjar, per exemple, saber quins dentistes els poden atendre prop del seu domicili en un horari determinat.

Per tal d'aconseguir els objectius que es proposa, la Web Semàntica fa ús d'un conjunt de conceptes, llenguatges i tecnologies que es comenten tot seguit de forma breu.

En primer lloc, cal esmentar que actualment els continguts en la Web s'expressen majoritàriament en **HTML**. Aquest és, bàsicament, un llenguatge de formatat que no aporta informació semàntica sobre el contingut dels documents i no permet, per tant, que siguin processats fàcilment en el que afecta al seu significat pels programes.

La idea de la Web Semàntica és canviar l'**HTML** per llenguatges més adients –entre ells, l'**XML**– que expressin, no només quina serà l'aparença final del document, sinó que aportin informació rellevant i estructurada sobre el seu contingut, amb la qual cosa aquest podrà ser interpretat per agents de programari. L'exemple següent[5] il·lustra aquesta qüestió.

```
<h1>Agilitas Physiotherapy Centre</h1>
Welcome to the Agilitas Physiotherapy Centre home page.
Do you feel pain? Have you had an injury? Let our staff
Lisa Davenport, Kelly Townsend (our lovely secretary)
and Steve Matthews take care of your body and soul.
<h2>Consultation hours</h2>
Mon 11am - 7pm<br>
Tue 11am - 7pm<br>
Wed 3pm - 7pm<br>
Thu 11am - 7pm<br>
Fri 11am - 3pm
<p>But note that we do not offer consultation during the weeks of the
<a href=". . .">State Of Origin</a> games.
```

[5] L'exemple ha estat tret del text esmentat anteriorment d'Antoniou i Van Harmelen.

```
<company>
    <treatmentOffered>Physiotherapy</treatmentOffered>
    <companyName>Agilitas Physiotherapy Centre</companyName>
    <staff>
       <therapist>Lisa Davenport</therapist>
       <therapist>Steve Matthews</therapist>
       <secretary>Kelly Townsend</secretary>
    </staff>
</company>
```

Es pot observar com la forma d'expressar un contingut –en ambdós casos apareix el nom de la empresa i els seus membres– facilita el seu processament: en el document expressat en **XML**, un programa pot extreure fàcilment els noms i l'ocupació de cadascuna de les persones que formen part de l'empresa, cosa prou complicada en el primer cas.

Cal dir que l'**XML** és un llenguatge que permet estructurar la informació per tal de fer-la més accessible a les màquines, facilitant entre altres qüestions el seu intercanvi, però que no dota de significat a les dades que expressa. Els documents **XML** poden presentar dues característiques:

- estar ben formats, és a dir, ser sintàcticament correctes;
- i ser vàlids, és a dir que el seu contingut s'ajusta a les especificacions definides en documents **DTD** –*Document Type Definition*– o **XSD** –*XML Schema*.

Aquestes no són, però, de tipus semàntic. Si ens fixem en el rerefons de l'exemple proposat, observem que apareixen conceptes d'un determinat domini: una empresa formada per persones amb distintes qualificacions que ofereix tractaments d'un determinat tipus. L'**XML** no permet expressar la semàntica d'aquest domini. Només amb aquest llenguatge no podríem indicar, per exemple, que els membres de l'*staff* han de ser persones i que n'hi ha d'haver, com a mínim, una que sigui terapeuta i una altra que realitzi feines de secretaria.

En general, en la Web Semàntica es necessita representar la informació de forma que apareguin clars els conceptes relacionats amb un determinat domini, les seves característiques i les seves relacions. El que s'acaba d'esmentar no és res més que el concepte tècnic d'ontologia, és a dir, la Web Semàntica necessita de les ontologies per a poder complir amb els seus objectius. Es necessitaran, però, altres llenguatges per a expressar-les.

2.1.2.- El concepte d'ontologia.

El terme ontologia prové del camp de la filosofia i es considera, en aquest sentit, com *la part de la metafísica que tracta de l'ésser en general i de les seves propietats essencials*[6]. El terme, com d'altres, ha estat adoptat i adaptat pel món de la Informàtica, bàsicament en el camp de la Intel·ligència Artificial. En aquest context, es pot utilitzar la definició proposada per Gruber i Studer: *una ontologia es una especificació explicita i formal d'un model conceptual*.

Rere aquesta definició, s'hi troba el fet que una ontologia descriu els conceptes bàsics que cal considerar en un cert domini –les classes d'objectes–, les seves característiques –propietats o atributs–, les seves relacions –per exemple, les de tipus jeràrquic– i les restriccions que els afecten. El diagrama de la figura 5 mostra, com a exemple, una part d'una ontologia.

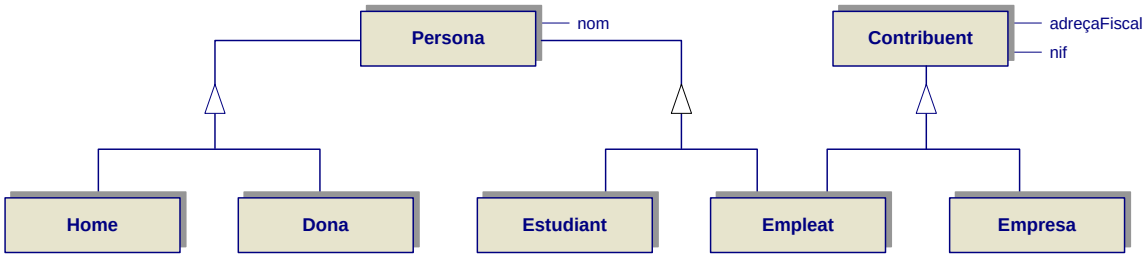


Fig. 5 - Un exemple d'ontologia.

A més de les relacions de tipus jeràrquic entre classes, podem observar les propietats associades als objectes d'una determinada classe: una *Persona* té un `nom`, mentre que un *Contribuent* té una `adreçaFiscal` i se'l pot identificar mitjançant el seu `nif`. Ara bé, una ontologia pot incloure, a més, altres informacions com, per exemple:

[6] La definició ha estat extreta del *Diccionari de la Llengua Catalana* de l'**Institut d'Estudis Catalans**.

- Restriccions de diferents tipus:
  - Quin tipus d'objecte pot tenir associada una determinada propietat –per exemple, només les instàncies de *Contribuent* poden tenir `adreçaFiscal`.
  - Quants valors d'una propietat poden relacionar-se amb una determinada instància d'un objecte –una *Persona* només pot tenir un `nom`.
- Declarar explícitament que dues classes són disjuntes –seria el cas en l'exemple anterior de *Home* i *Dona*.

Per a expressar tota la informació que pot contenir una ontologia cal un llenguatge amb l'expressivitat adequada. En el cas de la Web Semàntica, aquest llenguatge és l'**OWL**. Com ja s'ha indicat en l'apartat 1.2.1, es basa en **RDF** i **RDFS**, que alhora se sustenten en l'**XML**.

2.1.3.- Ontologies i Web Semàntica.

En els paràgrafs anteriors s'ha indicat que, en el context de la informàtica, l'ontologia és un concepte propi del camp de la Intel·ligència Artificial i ha estat adoptat per la Web Semàntica ja que permet representar de forma precisa el coneixement associat a un domini i, un cop expressat, posar-lo a disposició d'altres.

En el camp de la Web Semàntica, disposar d'aquest coneixement compartit té una sèrie de conseqüències:

- Permet superar els problemes causats per la terminologia a l'hora de descriure conceptes. En concret:
  - Fa possible que documents o aplicacions diferents que treballen sobre dominis –o parts de domini– comuns puguin superar les diferències terminològiques.
  - També es poden solucionar els problemes ocasionats per l'ús d'un terme comú aplicat a conceptes diferents.

  En aquests casos, n'hi ha prou en relacionar adequadament les ontologies que usen els documents o aplicacions implicats.
- Millora la precisió dels resultats que proporcionen els cercadors:
  - Aquests, en lloc de cercar pàgines que contenen les paraules clau, poden buscar en pàgines que descriuen de forma precisa els conceptes desitjats emprant ontologies.
  - A més, poden tenir presents les relacions jeràrquiques entre classes que defineix l'ontologia, bé per suggerir a l'usuari noves cerques quan la pàgina demanada ha fallat, o bé per acotar-la quan s'obtenen massa respostes.
- A més, tenir el coneixement d'un domini representat en una ontologia permet obtenir-ne de nou gràcies a l'aplicació d'una sèrie de regles d'inferència.

Podem dir, per tant, que si una pàgina web descriu la seva informació utilitzant ontologies, els agents de *software* poden arribar a resoldre preguntes de tipus semàntic processant el seu contingut, la qual cosa només fa que aportar beneficis i permet aconseguir l'objectiu principal de la web semàntica: ajudar als usuaris en les activitats que realitzen diàriament *online*.

2.1.4.- Model de capes de la Web Semàntica.

Tal com s'ha esmentat en la presentació del **TFC** –apartat 1.2.1–, la infraestructura que suporta la Web Semàntica se sol representar a través d'un model arquitectònic de capes que **Tim Berners-Lee** va presentar en la *XML-Conference* de l'any 2000.

En la figura 6, es torna a reproduir el model de capes esmentat. Per sobre del nivell inferior, ocupat pels identificadors uniformes de recursos –**URI**, *Uniform Resource Identifier*– utilitzats per a identificar els diferents recursos i objectes disponibles en la xarxa, s'hi troba l'**XML**. Com ja s'ha indicat, aquest llenguatge permet estructurar la informació –dades i metadades– mitjançant l'ús d'un vocabulari ajustat a les necessitats del context, de forma que la informació pugui ser fàcilment processada i bescanviada pel distints equips situats en la xarxa.

Com l'**XML** no aportat significat a les dades transmeses, es necessiten altres llenguatges que es troben en la capa superior: **RDF** i **RDFS**. Tot i que existeixen altres representacions sintàctiques d'ambdós llenguatges, en el cas de la Web Semàntica, els dos fan ús de l'**XML** com a base sintàctica.

RDF és un model de dades que permet escriure sentències senzilles sobre els objectes –normalment recursos– disponibles en la web, indicant el seu tipus i les relacions que mantenen uns amb els altres. La seva semàntica, però, és molt limitada i no permet definir de forma clara les característiques d'un determinat domini. Per a poder-ho fer, cal utilitzar **RDFS**. Aquest llenguatge permet organitzar els objectes de la web de forma jeràrquica, ja que

és possible definir classes i propietats, relacions entre ambdós conceptes, relacions d'herència –es poden crear subclasses i subpropietats– i realitzar restriccions sobre els distints elements del domini o del rang d'una determinada propietat.

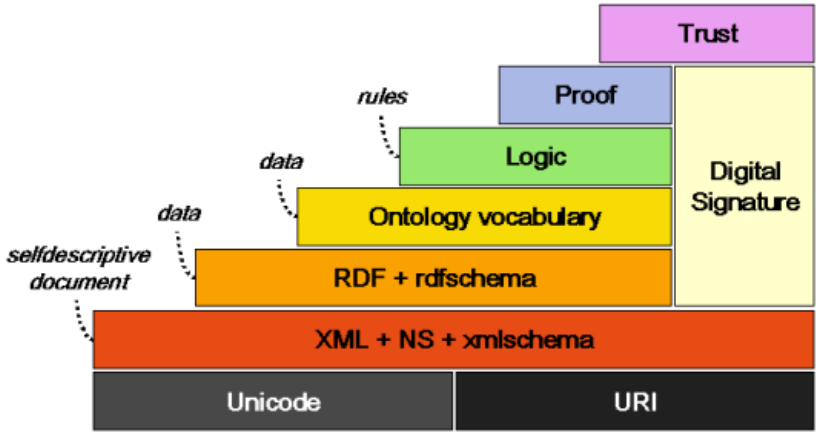


Fig. 6 - Model de capes de la Web Semàntica.

Tot i que **RDFS** es pot considerar com un llenguatge que permet descriure ontologies, la seva expressivitat en aquest camp és molt reduïda, per la qual cosa el *Web Ontology Working Group* del **W3C** va pensar en d'altres llenguatges que fossin més adequats per a expressar la complexitat de les relacions entre els objectes que representen els recursos existents en la web. Entre aquests llenguatges es troba **OWL**. Basat, com ja s'ha indicat en **RDF** i **RDFS**, n'existeixen tres variants: **OWL-Lite**, **OWL-DL** i **OWL-Full** –citades de menys a més expressivitat.

La figura anterior mostra tres capes per sobre de les esmentades fins ara:

- La capa lògica permet realitzar consultes sobre els elements del domini i inferir coneixement a partir de les capacitats que ofereixen els llenguatges emprats per a descriure les ontologies.
- La capa de prova associa el procés deductiu amb la representació de proves en els llenguatges de les capes inferiors i amb la seva validació.
- La capa de confiança permet assignar característiques de seguretat als recursos disponibles en la xarxa mitjançant l'ús de signatures digitals i altres tipus d'estructures de seguretat com poden ser, per exemple, les xarxes d'agents de confiança.

En la construcció del model anterior es van tenir en compte dos principis bàsics (Antoniou & Van Harmelen, 2008):

- El manteniment de la compatibilitat descendent, de forma que un agent plenament integrat en una capa sigui capaç d'interpretar i utilitzar la informació dels nivells inferiors –com a exemple, un agent que entén **OWL** treu avantatge del fet d'usar els llenguatges **RDF** i **RDFS** en els nivells inferiors.
- La comprensió de caire parcial en el plànol ascendent. En aquest cas, un element situat en una determinada capa hauria de ser capaç de treure com a mínim un avantatge parcial de la informació que es troba en les capes superiors. Per exemple, un agent amb coneixements només de **RDF** i **RDFS** pot interpretar parcialment el coneixement escrit en **OWL**, deixant de banda aquells elements de **OWL** que no pertanyen als llenguatges anteriors.

2.2.- Llenguatges emprats en la Web Semàntica.

En aquest apartat es comenten determinats aspectes dels llenguatges citats en els paràgrafs anteriors, amb una atenció més acurada per a **RDF**, **RDF Schema** i **OWL**, ja que l'**XML** era un llenguatge prou conegut per l'estudiant abans de l'inici del projecte.

2.2.1.- XML i XML-Schema.

L'**XML** –*Extensible Markup Language*– és un llenguatge de marques creat amb l'objectiu d'estructurar lògicament els documents, de forma que sigui possible la seva manipulació per les màquines i es puguin emprar per l'intercanvi d'informació, independentment de l'estructura dels sistemes emprats en l'intercanvi.

Un document **XML** té dues parts diferenciades:

- La **capçalera**, on hi apareix la informació associada a la gestió del document –versió, codificació emprada, tipus de document,...–. Com a mínim, la capçalera ha de contenir el número de versió:

```
<?xml version="1.0" encoding="UTF-8"?>
```

- La **instància del document** on es troba el contingut real d'aquest.

Els components bàsics del que hem anomenat la instància són els elements, els atributs i les dades. Es comenten tot seguit.

- **Elements**: constitueixen les unitats bàsiques dels documents, ja que representen els components lògics que el conformen.

  Cada element es descriu a través d'una etiqueta on hi consta el nom del tipus d'element i una llista, opcional, d'atributs que marquen les característiques o propietats de l'element. El contingut de l'element se situa entre una marca d'inici i una final:

```
<project name="RefactoringPluginProtege" default="default" basedir=".">

    <description>
        Builds, tests, and runs the project RefactoringPluginProtege.
    </description>
    <import file="nbproject/build-impl.xml"/>
    <target name="-post-compile">
        <obfuscate>
            <fileset dir="${build.classes.dir}"/>
        </obfuscate>
    </target>

</project>
```

  En l'exemple es pot observar com es declara l'element arrel del document a través de l'etiqueta **`project`** i com se situa el seu contingut entre la marca d'inici **`<project [llista_atributs]>`** i la marca final **`</project>`**.

- **Atributs**: descriuen les propietats dels elements i formen part de l'etiqueta.

  Cada atribut es representa a través d'una parella nom-valor en què aquest últim ha d'anar tancat entre cometes.

  L'exemple mostra com l'element **`project`** té, entre altres, l'atribut **`name`**, el valor del qual és, en aquest cas, el nom d'un *plugin*.

- **Dades**: els elements del document, excepte els buits, contenen, a més dels atributs, les dades. Aquestes es troben entre les marques d'inici i de fi, i poden ser de diferents tipus, entre els quals hi ha d'altres elements.

  En l'exemple anterior es pot observar com l'element arrel conté tres elements, **`description`**, **`import`** i **`target`**. El contingut del primer d'ells és una cadena. El segon només té un atribut, mentre que el tercer conté un altre element.

Els documents **XML** han de complir una sèrie de regles sintàctiques. Si les compleixen es diu que el document és **ben format**. En cas contrari, no es considera un document **XML**. Enunciades breument són les següents:

- L'element arrel ha de ser únic.
- L'estructura d'etiquetes ha d'estar jerarquitzada.
- Els valors dels atributs han d'anar entre cometes –dobles o senzilles.
- **XML** distingeix entre majúscules i minúscules.
- Els caràcters reservats –*<*, *>*, *"*, *&*– s'han de codificar.
- **XML** admet les etiquetes buides –com **`import`** en l'exemple anterior.

Ara bé, aquestes regles no diuen res referent a l'estructura del document. En aquest sentit, es fa necessari definir els elements i atributs que un document pot utilitzar i dins quins altres elements. Un cop definida aquesta estructura es pot dir si un document és **vàlid**: ho és quan està ben format, utilitza informació estructurada i, sobretot, ha estat construït respectant aquesta estructura.

L'estructura d'un document **XML** es pot definir de dues formes distintes:

- Mitjançant **DTD** –*Document Type Definition*. Els components que formen part d'un **DTD** es poden situar dins el propi document **XML** o bé en un fitxer extern, la qual cosa permet aprofitar-los en varis documents.

  Els **DTD** presenten dos problemes:

  - No són documents **XML**.
  - Són poc rics en expressar tipus de dades.

- A través de **XML Schema**, la sintaxi del qual es basa en **XML** i és més ric en el que es refereix a tipus de dades.

  El fet que un document **XML Schema** sigui, alhora, un document **XML** permet que es puguin emprar totes les eines d'edició i anàlisi vàlides per a **XML**, la qual cosa li representa un avantatge respecte **DTD**.

Tot seguit esmentarem algunes característiques de l'**XML Schema** donat el seu ús en la web semàntica. Abans però, comentarem breument un concepte: l'espai de noms.

Un **espai de noms** –*namespace*– defineix una correspondència entre un prefix i un **URI**. Ha estat dissenyat per tal d'evitar ambigüitats en els documents **XML** quan, per exemple, es fusionen documents procedents de fonts distintes que han fet ús de la mateix etiqueta en contexts diferents.

Es declara de la forma següent:

```
xmlns:prefix = "uri_espai_de_noms"
```

i per a definir una etiqueta que pertany a l'espai de noms ho fem mitjançant:

```
prefix_espai:nom_etiqueta
```

la qual cosa permet resoldre les ambigüitats conservant el nom de l'etiqueta i afegint-li el prefix que identifica l'àmbit en què l'etiqueta és vàlida. Tot seguit se'n mostra un exemple que inclou tan la declaració de l'espai de noms com el seu ús:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.owl-pfc.com/2008/OntologyImplicitSubsets.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.owl-pfc.com/2008/OntologyImplicitSubsets.owl">

    <owl:Ontology rdf:about=""/>
```

⇒ L'element arrel d'un document **XML Schema**[7] es declara a través de la següent etiqueta:

```
<xsd:schema
    xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" version="1.0">
```

Hom pot observar com el prefix de l'espai de noms utilitzat és **xsd**.

El contingut més important del document són les definicions dels tipus d'elements i d'atributs. Es realitzen en la forma següent:

- **Tipus d'elements**.

  Es declaren mitjançant l'etiqueta **element** i s'ha d'indicar, a través de l'atribut **name**, el nom de l'element. Altres atributs opcionals són, entre altres, el tipus –**type**– i les restriccions de cardinalitat –**minOccurs** i **maxOccurs**. El valor d'aquests dos darrers atributs pot ser qualsevol número, inclòs el zero, o bé la paraula **unbounded** en el cas de la cardinalitat màxima.

  ```
  <element name="surName" type="string" minOccurs="1" maxOccurs="1"/>
  ```

- **Tipus d'atributs**.

  Es declaren emprant l'etiqueta **attribute** i cal indicar el nom de l'atribut. Opcionalment, el tipus, l'existència –**use** amb tres valors possibles: **optional**, **required** o **prohibited**– i un valor per defecte –**default**.

[7] Una bona explicació sobre **XML**, **DTD** i **XML Schema** es pot trobar en la següent referència **Mateu, C.** (2004) *Desarrollo de Aplicaciones Web.* (1a ed.) FUOC.

```
<attribute name="id" type="ID" use="required"/>
```

- **Tipus de dades**.

  XML Schema té predefinits un ampli ventall de tipus de dades, entre els que podem distingir:

  - Tipus booleà: **boolean**.
  - Tipus numèrics: **integer**, **short**, **byte**, **long**, **float** i **decimal**.
  - Tipus de cadena: **string**, **ID**, **IREF**, **CDATA** i **language**.
  - Tipus de data i hora: **date**, **time**, **gMonth** i **gYear**.

  L'exemple següent mostra –en el context de l'**RDF**– l'ús de dos d'aquests tipus de dades: un per a proporcionar un valor enter, i l'altre un valor de cadena.

```
<owl:Class rdf:ID="MenorEdat">

    <rdfs:subClassOf rdf:resource="#Persona"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#edat"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
          1
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>

</owl:Class>

<owl:FunctionalProperty rdf:ID="nif">

    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Empleat"/>
          <owl:Class rdf:about="#Empresa"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>

</owl:FunctionalProperty>
```

  A més, en **XML Schema** es poden definir tipus de dades nous, que poden ser:

  - Tipus senzills –*simple data types*–: en aquest tipus no es poden emprar ni elements ni atributs.

    L'exemple següent defineix un nou tipus de dada per a representar els dies de la setmana.

```
<xsd:simpleType name="diaSetmana">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Dilluns"/>
    <xsd:enumeration value="Dimarts"/>
    <xsd:enumeration value="Dimecres"/>
    <xsd:enumeration value="Dijous"/>
    <xsd:enumeration value="Divendres"/>
    <xsd:enumeration value="Dissabte"/>
    <xsd:enumeration value="Diumenge"/>
  </xsd:restriction>
</ xsd:simpleType>
```

  - Tipus complexos –*complex data types*–: en aquest cas, es poden utilitzar elements i atributs.

    Es defineixen a partir d'altres tipus de dades emprant una sèrie d'elements predefinits en el llenguatge:

    - **sequence**, per a indicar que els elements que formen part del nou tipus han d'aparèixer en un ordre determinat.
    - **all**, quan es vol que apareguin tots els elements però no importa l'ordre.
    - **choice**, per a indicar un conjunt d'elements dels que cal escollir-ne un.

Tot seguit se'n mostra un exemple en què s'utilitza la primera de les possibilitats comentades.

```
<xsd:element name="student" type="person"/>

  <xsd:complexType name="person">
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="surname" type="xsd:string"/>
    </xsd:sequence>

</xsd:complexType>
```

En un dels exemples anteriors es pot observar com a l'hora de definir el tipus de dades senzill s'ha fet ús de l'etiqueta **restriction**. Com es pot suposar, s'utilitza per a restringir els valors que pot prendre un determinat tipus de dades.

En l'exemple, s'ha declarat que el nou tipus –**diaSetmana**– es crea prenent com a base –d'aquí l'atribut **base**– el tipus de dada **string**, i es restringeixen els valors que pot prendre als diferents valors enumerats.

A més de les que apareixen en l'exemple, el llenguatge n'admet d'altres. Es mostren en la taula que tanca aquest apartat.

Restriccions XML Schema	
Restricció	**Descripció**
enumeration	Especifica la relació de valors vàlids.
fractionDigits	Indica el número màxim de xifres decimals.
length	Especifica la grandària exacta requerida.
maxExclusive	Especifica el límit superior per a valors numèrics –exclou el valor indicat.
maxInclusive	Especifica el límit superior per a valors numèrics –inclou el valor indicat.
maxLength	Especifica la grandària màxima requerida.
minExclusive	Especifica el límit inferior per a valors numèrics –exclou el valor indicat.
minInclusive	Especifica el límit inferior per a valors numèrics –inclou el valor indicat.
minLength	Especifica la grandària mínima requerida.
Pattern	Indica el patró que el valor ha de complir.
totalDigits	Indica el número exacte de dígits.
whiteEspace	Indica com cal tractar els caràcters en blanc.

Taula 4 - Restriccions XML Schema.

2.2.2.- RDF i RDF Schema.

L'**XML** no proporciona per se cap significat a les dades que es representen emprant la seva sintaxi. L'exemple[8] que es detalla tot seguit és clarificador en aquest sentit. El següent fet:

David Billington is a lecturer of Discrete Mathematics

es pot representar en **XML** mitjançant qualsevol de les expressions que es mostren a continuació:

```
<course name="Discrete Mathematics">
     <lecturer>David Billington</lecturer>
</course>

<lecturer name="David Billington">
     <teaches>Discrete Mathematics</teaches>
</lecturer>

<teachingOffering>
     <lecturer>David Billington</lecturer>
     <course>Discrete Mathematics</course>
</teachingOffering>
```

Es pot observar que la imbricació –*nesting*– ha estat capgirada en els dos primers casos, mentre que en el tercer cas el professor i el curs han estat situats al mateix nivell. És clar que en **XML** no existeix una forma estandarditzada d'assignar significat a la imbricació de les etiquetes.

[8] L'exemple ha estat tret del text *A Semantic Web Primer* d'Antoniou i Van Harmelen.

L'**RDF** –*Resource Description Framework*– constitueix el primer pas per a solucionar aquest problema. Com ja s'ha comentat en pàgines precedents, **RDF** és essencialment un model de dades en què el component bàsic, anomenat sentència, està format per l'estructura **`objecte-atribut-valor`** –se sol anomenar triple. Si l'exemple anterior es volgués expressar en **RDF**, tindríem la següent equivalència:

- *David Billington* seria l'**`objecte`**,
- *is a lecturer* seria l'**`atribut`** i
- *Discrete Mathematics* el seu **`valor`**.

Com qualsevol model de dades abstracte, **RDF** necessita una determinada sintaxi per a ser representat. Tot i que s'admeten altres representacions, la sintaxi més utilitzada és la basada en **XML**.

Una altra de les característiques de **RDF** és que corre a càrrec dels usuaris la definició de la terminologia pròpia d'un determinat domini, cosa que es fa emprant l'anomenat *RDF Schema* –**RDFS**. La relació entre **RDF** i **RDFS** no és del mateix tipus que la que existeix entre **XML** i **XML Schema**. Mentre que aquest últim marca l'estructura d'un document XML, **RDFS** defineix el significat dels termes que usen els models de dades **RDF**. En aquest sentit, especifica a quins objectes es poden aplicar determinades propietats, quins valors poden prendre, i quines són les relacions entre objectes.

En els paràgrafs següents, es mostren de forma resumida els conceptes i les característiques claus dels dos llenguatges.

2.2.2.1.- RDF.

Els conceptes bàsics de **RDF** són els recursos, les propietats i les sentències.

- **Recursos**: són els objectes o elements que la sentència descriu. Cada recurs s'identifica mitjançant una **URI**.
- **Propietats**: descriuen les relacions entre recursos. S'identifiquen, també, a través de **URI**.
- **Sentències**: indiquen quines són les propietats que tenen els recursos. Com s'ha esmentat anteriorment, una sentència segueix l'estructura **`objecte-atribut-valor`**, en què l'objecte és un recurs, l'atribut és una propietat i el valor pot ser tan un recurs com una dada literal.

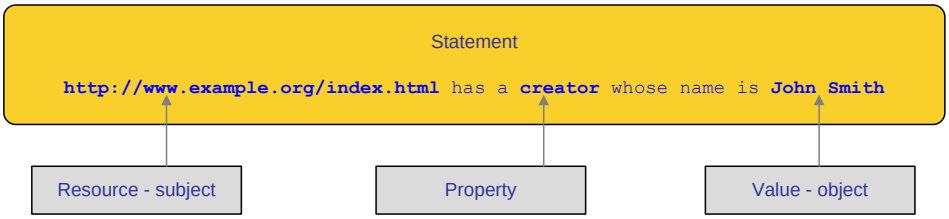


Fig. 7 - Conceptes fonamentals de RDF.

La figura 7 mostra un exemple[9] del que s'acaba d'indicar. Es pot comprovar que una sentència no és res més que un predicat binari en què els diferents elements s'identifiquen per la seva **URI**. En forma de triple, es podria expressar com:

```
(http://www.example.org/index.html, http://purl.org/dc/elements/1.1/creator, #JohnSmith)
```

També es pot expressar gràficament a través d'un graf dirigit en què les arestes van des del recurs –anomenat també *subject*– fins el valor –l'*object* de la sentència. En aquesta línia, hom es refereix a la propietat com a *predicate*.

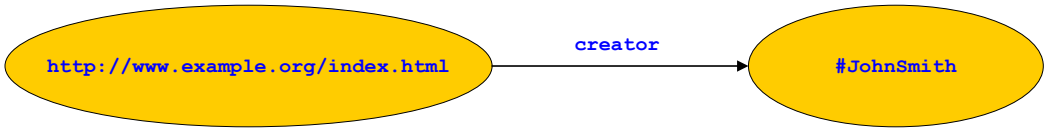


Fig. 8 - RDF. Conceptes bàsics: representació gràfica.

[9] L'exemple ha estat tret de la pàgina web http://www.w3.org/TR/rdf-primer/#basicconcepts.

Emprant **XML**, la sentència anterior s'expressaria en la forma següent:

```
<?xml version="1.0" encoding="UTF-16"?>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">

    <rdf:Description rdf:about="http://www.example.org/index.html">
       <dc:creator rdf:resource="#JohnSmith"/>
    </rdf:Description>

</rdf:RDF>
```

En aquesta representació, podem destacar els següents aspectes:

- Un document **RDF** es representa per l'etiqueta **`rdf:RDF`**.
- El document conté una sèrie d'elements etiquetats pel *tag* **`rdf:Description`**. Cada element dins la descripció expressa una sentència sobre un recurs. Aquest es pot identificar de tres formes:
  - Per un atribut **`about`**.
  - A través de l'atribut **`ID`** que crea un nou recurs.
  - Sense nom, amb la qual cosa es crea un recurs anònim.
- En els elements que formen part de la descripció, la propietat s'usa com a etiqueta i el valor de la propietat l'expressa el contingut.

En l'exemple següent[10] es mostra l'aplicació d'aquests conceptes de forma més ampliada. Referent al que s'esmenta en el darrer punt del paràgraf anterior, es poden observar distints elements propietat, és a dir, parelles

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:uni="http://www.mydomain.org/uni-ns#">

    <rdf:Description rdf:about="http://www.mydomain.org/uni-ns#949352">
       <uni:name>Grigoris Antoniou</uni:name>
       <uni:title>Professor</uni:title>
    </rdf:Description>

    <rdf:Description rdf:ID="949318">
       <uni:name>David Billington</uni:name>
       <uni:title>Associate Professor</uni:title>
       <uni:age rdf:datatype="&xsd;integer">27</uni:age>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.mydomain.org/uni-ns#CIT1111">
       <uni:courseName>Discrete Mathematics</uni:courseName>
       <uni:isTaughtBy rdf:resource="#949318"/>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.mydomain.org/uni-ns#CIT1112">
       <uni:courseName>Concrete Mathematics</uni:courseName>
       <uni:isTaughtBy>Grigoris Antoniou</uni:isTaughtBy>
    </rdf:Description>

</rdf:RDF>
```

propietat-valor corresponents a les sentències **RDF** en què l'element descripció n'és el subjecte. També es pot observar com **RDF** fa ús –a través de l'element **`rdf:datatype`**– dels tipus de dades que defineix **XML Schema**.

La següent taula recull altres etiquetes que s'usen en **RDF**/**XML**, explica on s'apliquen i el seu sentit.

Resum RDF/XML		
Etiqueta	**Tipus**	**Descripció**
`rdf:resource`	**Atribut**	Indica que el valor d'una propietat és un recurs que s'identifica a través del seu **URI**.

[10] L'exemple és una versió simplificada del model que apareix en la pàgina 74 del text *A Semantic Web Primer* d'Antoniou i Van Harmelen.

Resum RDF/XML		
Etiqueta	**Tipus**	**Descripció**

rdf:type	**Element**	Indica que el valor d'una propietat és un recurs que representa una determinada classe d'elements, de forma que el *subject* de la propietat és una instància d'aquesta classe.
		`<rdf:Description rdf:about="CIT1111">`
		`<rdf:type rdf:resource="&uni;course">`
		`<uni:courseName>Discrete Mathematics</uni:courseName>`
		`<uni:isTaughtBy rdf:resource="#949318"/>`
		`</rdf:Description>`

rdf:Bag	**Element**	Permet indicar una col·lecció no ordenada de recursos o atributs.
		`<rdf:Bag>`
		`<rdf:li rdf:resource="CIT1111"/>`
		`<rdf:li rdf:resource="CIT1112"/>`
		`</rdf:Bag>`

rdf:Seq	**Element**	Permet indicar una col·lecció ordenada.
		`<rdf:Seq>`
		`<rdf:li rdf:resource="CIT1111"/>`
		`<rdf:li rdf:resource="CIT1112"/>`
		`</rdf:Seq>`

rdf:Alt	**Element**	Permet indicar una conjunt d'alternatives dins d'una col·lecció.
		`<rdf:Alt>`
		`<rdf:li rdf:resource="CIT1111"/>`
		`<rdf:li rdf:resource="CIT1112"/>`
		`</rdf:Alt>`

rdf:li	**Element**	Indica un membre dins una col·lecció.

rdf:Statement
rdf:subject
rdf:predicate
rdf:object
	Element	Permet descriure una sentència a través del que s'anomena la reificació de la sentència. En aquest procés, cal indicar el subjecte, el predicat i l'objecte.
		Per exemple, si considerem la sentència:
		`<rdf:Description rdf:about="949352">`
		`<uni:name>Grigoris Antoniou</uni:name >`
		`</rdf:Description >`
		La seva reificació seria:
		`<rdf:Statement rdf:about="About949352">`
		`<rdf:subject rdf:resource="949352"/>`
		`<rdf:subject rdf:resource="&uni;name"/>`
		`<rdf:object>Grigoris Antoniou</rdf:object >`
		`</rdf: Statement >`

Taula 5 - Resum RDF/XML.

2.2.2.2.- RDF Schema.

El llenguatge introduït en l'apartat anterior permet descriure els recursos d'un determinat domini emprant una terminologia pròpia d'aquest. No permet, però, definir-ne la semàntica ja que no admet, per exemple, classes ni jerarquies per a aquestes ni per a les propietats. Aquesta tasca cal portar-la a terme utilitzant les prestacions que ofereix l'anomenat *RDF Schema* –**RDFS**. Mitjançant aquest llenguatge, hom pot declarar classes i propietats, relacions entre ambdós conceptes, definir relacions d'herència i realitzar restriccions sobre els distints elements del domini.

Una qüestió clau a considerar és que en **RDFS** les primitives de modelat es defineixen utilitzant recursos i propietats, és a dir, utilitzant l'**RDF**, de forma que es pot dir que un document **RDFS** és, alhora, un document **RDF** i, naturalment, un document **XML** quan s'usa aquesta sintaxi per a expressar el model **RDF**.

Les primitives per a modelar que ofereix **RDFS** es comenten classificades tot seguit.

- **Classes**: les classes que formen el nucli són:
  - **rdfs:Resource**, que representa la classe de tots els recursos.
  - **rdfs:Class**, la classe de totes les classes.
  - **rdfs:Literal**, la classe de totes les cadenes.
  - **rdfs:Property**, que representa la classe de totes les propietats.

- **rdfs:Statement**, la classe de totes les sentències reificades.

- **Propietats**: permeten definir relacions i restringir el domini i el rang de les propietats.
  - **rdf:type**, relaciona un recurs amb la seva classe, la qual cosa permet declarar que el recurs és una instància de la classe.
  - **rdfs:subClassOf**, declara una classe com a subclasse d'una determinada classe. El llenguatge admet que una classe sigui subclasse de més d'una superclasse.
  - **rdfs:subPropertyOf**, declara que una propietat hereta d'una altra.
  - **rdfs:domain**, especifica el domini d'una propietat i indica que qualsevol recurs que té aquesta propietat és una instància de les classes del domini.
  - **rdfs:range**, especifica el rang d'una propietat i, com en l'element anterior, indica que els valors d'aquesta propietat són instàncies de les classes del rang.

```
<rdfs:Class rdf:about="lecturer">
   <rdfs:subClassOf rdf:resource="staffMember"/>
</rdfs:Class>

<rdfs:Property rdf:ID="phone">
   <rdfs:domain rdf:resource="#staffMember"/>
   <rdfs:range rdf:resource="&rdf;Literal"/>
</rdfs:Property>
```

  L'exemple mostra com es declara que tots els *lecturers* són instàncies de *staffMember*, i com qualsevol recurs que té un telèfon és una instància de *staffMember* i el valor de la propietat és una cadena.

- **Contenidors**: a més dels elements esmentats en l'apartat 2.2.2.1 –**rdf:Bag**, **rdf:Seq** i **rdf:Alt**– es defineix:
  - **rdfs:Container**, que representa la classe de tots els contenidors, inclosos els anteriors.

- **Propietats d'utilitat**: com els recursos poden estar declarats en varis llocs, les propietats d'aquest tipus permeten definir enllaços a les adreces que declaren els recursos.
  - **rdf:seeAlso**, relaciona un recurs amb un altre que l'explica.
  - **rdfs:isDefinedBy**, és una subpropietat de l'anterior i relaciona el recurs amb la seva definició.

2.2.3.- OWL.

L'expressivitat de **RDF** i de **RDFS** per a descriure ontologies és molt reduïda. El primer permet modelar mitjançant predicats binaris i el segon amplia aquesta possibilitat incorporant jerarquies de classes i propietats, amb definicions de domini i rang per a aquestes, però no permet, per exemple, indicar que dues classes són disjuntes, combinar classes o declarar restriccions de cardinalitat.

Aquestes raons van portar al *Web Ontology Working Group* del **W3C** a pensar en d'altres llenguatges que fossin més adequats a la descripció d'ontologies en l'àmbit de la Web Semàntica. Entre aquests hi ha **OWL** –*Web Ontology Language*, que és el llenguatge destinat a ser l'estàndard en l'àmbit esmentat.

Un llenguatge ontològic ha de permetre descriure de manera formal i explícita els conceptes d'un determinat domini, així com les seves relacions. Per a aconseguir aquest objectiu, el llenguatge ha de complir una sèrie de característiques:

- Tenir una sintaxi correctament definida.
- Una semàntica ben formalitzada, per tal de poder saber, per exemple:
  - quan un element pertany a una classe,
  - si dues classes són equivalents,
  - o bé, si el model presenta inconsistències.
- Ha d'oferir un bon suport per al raonament –inferència i nou coneixement– que li permeti, entre d'altres coses:
  - comprovar la consistència d'una ontologia,
  - assignar de forma automàtica individus a les classes.

- Tenir la suficient potència expressiva i comoditat en l'expressió.

OWL presenta aquestes característiques. Basat en **RDF** i **RDFS** i construït en part sobre la lògica descriptiva, es presenta en tres variants o dialectes:

- **OWL Full**. És el més expressiu dels tres i utilitza, d'acord amb el nom, totes les primitives del llenguatge.

  El seu avantatge principal és que té compatibilitat ascendent plena amb **RDF** –del qual pot arribar inclús a canviar el significat de les primitives. El desavantatge és que la seva gran expressivitat el fa indecidible i, per tant, inadequat per a realitzar raonaments de tipus automàtic.

- **OWL DL** –**OWL** *Description Logic*. Per tal d'aconseguir eficiència computacional, restringeix la forma en què es poden usar els constructors de **RDF** i **OWL** –s'inhabilita l'aplicació d'un determinat constructor sobre els altres. En fer-ho, es perd la plena compatibilitat amb **RDF**.

- **OWL Lite**. És el menys expressiu. Respecte l'anterior, no admet, entre d'altres qüestions, les classes enumerades i les sentències de disjunció, i té limitades les restriccions de cardinalitat.

Cal tenir presents les següents compatibilitats entre les distintes variacions del llenguatge.

- Una ontologia que és vàlida en **OWL Lite**, ho és també en **OWL DL**.
- Una ontologia **OWL DL** vàlida és una ontologia **OWL Full** vàlida.
- Qualsevol conclusió que és vàlida en **OWL Lite**, ho és també en **OWL DL**.
- Una conclusió **OWL DL** correcta és una conclusió **OWL Full** correcta.

A més, com ja s'ha indicat, **OWL** es construeix sobre **RDF** i **RDFS** i utilitza la seva sintaxi basada en **XML**. També, tal com mostra la figura, determinats constructors **OWL** no són res més que especialitzacions d'elements de **RDFS**.

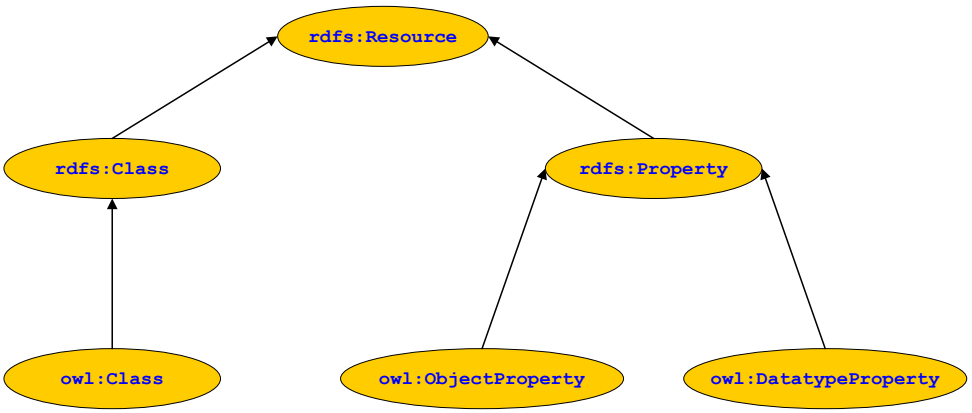


Fig. 9 - Relacions de classe entre OWL i RDF/RDFS.

Per tant, l'element arrel d'una ontologia **OWL** és un element `rdf:RDF`, tal com es mostra a continuació.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.owl-pfc.com/2008/RefactoringOperation.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.owl-pfc.com/2008/RefactoringOperation.owl">

  <owl:Ontology rdf:about="">
    <owl:versionInfo rdf:datatype="&xsd;string">1.0</owl:versionInfo>
    <rdfs:comment xml:lang="ca">
        Fitxer de configuraci&#243; de les operacions de refactoring
    </rdfs:comment>
  </owl:Ontology>
```

Després de la declaració de l'element arrel, poden aparèixer una sèrie de capçaleres sota l'etiqueta `owl:Ontology` que solen incloure comentaris, la versió del document o referències a d'altres ontologies. La resta del document està format per definicions de classes, de propietats, i declaracions sobre els fets relacionats amb les instàncies de les classes. Aquests elements es comenten a continuació.

- **Classes**: les classes proporcionen el mecanisme que permet agrupar els recursos de característiques similars. Tota classe **OWL** està associada a un conjunt de individus –les instàncies– que rep el nom d'extensió de la classe. La classe té un significat intensional de forma que dues classes poden tenir la mateixa extensió però ser diferents.

  Cada classe es defineix a través de la seva descripció conjuntament amb el que anomenem axiomes de classe. La descripció es realitza mitjançant el nom que identifica la classe o bé especificant la seva extensió, la qual cosa defineix una classe anònima.

  En **OWL** es distingeixen sis tipus de descripcions de classe:

  - Un identificador de classe que defineix la classe a través del seu nom.
  - L'enumeració de les seves instàncies –excepte en **OWL Lite**. S'indica a través de l'etiqueta **`owl:oneOf`**.
  - La que es basa en la restricció d'una determinada propietat. S'indica mitjançant l'element **`owl:Restriction`**.
  - La intersecció de dues o més classes. Usa l'element **`owl:intersectionOf`**.
  - La unió de dues o més classes. S'indica a través de l'element **`owl:unionOf`**.
  - El complement de la descripció d'una classe. Usa l'element **`owl:complementOf`**.

  Els cinc darrers tipus descriuen classes anònimes. En tots els casos, l'element que s'utilitza per a definir una classe és **`owl:Class`**. L'exemple següent mostra la definició d'una classe a través del seu nom.

```
<owl:Class rdf:ID="RefactoringOperation"/>
```

  En els altres casos, la descripció de la classes no és res més que un conjunt de triples **RDF** en què el node en blanc representa la classe que es descriu. L'exemple mostra la definició d'una classe com a unió d'altres dues:

```
<owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:ID="CostReport"/>
          <owl:Class rdf:ID="FinancialReport"/>
        </owl:unionOf>
</owl:Class>
```

  Cal dir que en **OWL** existeixen dues classes predefinides, **`owl:Thing`** –que conté totes les classes– i **`owl:Nothing`** que representa la classe buida.

  Relacionats amb les classes hi ha tot un conjunt d'axiomes. Aquests defineixen elements addicionals que permeten establir característiques necessàries i/o suficients que hauran de complir les instàncies d'una classe. N'existeixen tres:

  - **`rdfs:subClassOf`**, que permet especificar que una classe és una subclasse d'una altra.
  - **`owl:EquivalentClass`**, per a definir classes equivalents, és a dir, que dues classes tenen la mateixa extensió.
  - **`owl:disjointWith`**, per a declarar que dues classes són disjuntes.

  L'exemple següent explicita el que s'acaba d'esmentar. Descriu una classe, *Persona*, i dues subclasses d'aquesta, *Home* i *Dona*, de forma que l'especialització és total i disjunta. Es pot observar l'ús de l'axioma **`owl:EquivalentClass`** en combinació amb **`owl:unionOf`** per a indicar que no existeixen instàncies de *Persona* que no siguin o bé instàncies d'*Home*, o bé instàncies de *Dona*.

```
<owl:Class rdf:ID="Dona">
    <owl:disjointWith>
      <owl:Class rdf:ID="Home"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Persona"/>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Home">
    <rdfs:subClassOf rdf:resource="#Persona"/>
    <owl:disjointWith rdf:resource="#Dona"/>
</owl:Class>
```

```
<owl:Class rdf:about="#Persona">
    <owl:equivalentClass>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Home"/>
          <owl:Class rdf:about="#Dona"/>
        </owl:unionOf>
      </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

- **Propietats**.

  En **OWL** es distingeix principalment entre dos tipus de propietats[11]:

  - Les *object properties* que relacionen objectes amb d'altres objectes. Es declaren emprant l'etiqueta **`owl:ObjectProperty`**.
  - Les *datatype properties*[12] que relacionen objectes amb valors de dades com, per exemple, cadenes o enters. En la declaració es fa servir l'etiqueta **`owl:DatatypeProperty`**. **OWL** admet tres tipus de dades:
    - Els especificats en l'**RDF**.
    - Les de la classe **RDFS** **`rdfs:Literal`**.
    - Les de tipus enumeratiu creades a través del constructor **`owl:oneOf`**.

  L'exemple mostra la definició de propietats d'ambdós tipus.

```
<owl:ObjectProperty rdf:about="#studiesReport">
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#FinancialReport"/>
        <owl:Class rdf:about="#CostReport"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:domain rdf:resource="#FinancialAnalyst"/>
  <owl:inverseOf rdf:resource="#hasAnalyst"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="#dateCarriedOut">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#FinancialReport"/>
        <owl:Class rdf:about="#CostReport"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:DatatypeProperty>
```

  També es pot observar com una de les propietats es declara inversa d'una altra que no s'ha explicitat. Per a fer-ho, s'ha emprat un dels axiomes que defineixen característiques addicionals de les propietats: **`owl:inverseProperty`**.

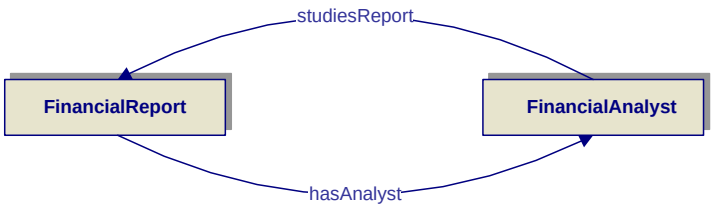


Fig. 10 - OWL. Propietats inverses.

[11] També defineix dos tipus més: *annotation properties* i *ontology properties*. Son necessàries en **OWL DL** per qüestions semàntiques.

[12] En **OWL Full**, les *datatype properties* són subclasses de les *object properties*.

En **OWL**, les propietats són dirigides –sempre des del domini al rang. Per a definir una relació en les dues direccions –en la figura, un informe té un analista i aquest estudia el report–, **OWL** ofereix l'axioma indicat que només es pot aplicar a les *object properties*.

A més d'aquest axioma i dels heretats de l'**RDFS** –`rdfs:subPropertyOf`, `rdfs:domain` i `rdfs:range`–, se'n contemplen d'altres:

- Els que estan associats a restriccions de cardinalitat globals:
  - `owl:FunctionalProperty`, permet indicar que per a cada valor del domini només n'hi ha un de possible en el rang.
  - `owl:InverseFunctionalProperty`. En aquest cas, l'*object* de la propietat en la sentència determina clarament qui és el *subject*, és a dir, només existeix una instància *x* que es relacioni amb una altra instància *y* a través de la propietat *P*. Aquest axioma només és aplicable a propietats de tipus *object properties*.
- Els relacionats amb característiques lògiques:
  - `owl:SymmetricProperty`. En aquest cas, si (*x*,*y*) és una instància de la propietat *P*, (*y*,*x*) també ho és.
  - `owl:TransitiveProperty`. Permet especificar que si *x* es relaciona amb *y* a través de *P* i, alhora, *y* ho fa amb *z* també a través de la *P*, llavors *x* es relaciona amb *z* mitjançant *P*.

Naturalment, aquests axiomes només són aplicables a les propietats del tipus *object properties*.

L'exemple mostra la declaració d'una propietat com a funcional.

```
<owl:FunctionalProperty rdf:ID="className">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#RefactoringOperation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
```

- **Restriccions de propietats**.

S'ha esmentat en paràgrafs anteriors que **OWL** admet que una classe es descrigui a través de la restricció de les característiques d'una determinada propietat, la qual cosa defineix una classe anònima que representa totes les instàncies que satisfan la restricció. Es distingeix entre dos tipus de restriccions: les de valor i les de cardinalitat.

- Restriccions de valor:
  - `owl:allValuesFrom`[13], permet indicar que tots els valors d'una propietat han de ser d'un determinat tipus. Aquest tipus de restricció introdueix un quantificador universal en els valors d'una propietat.
  - `owl:someValuesFrom`[14]. En aquest cas, s'indica que només alguns valors de la propietat han de ser d'un determinat tipus. Introdueix un quantificador existencial en els valors d'una propietat.
  - `owl:hasValue`[15], descriu el conjunt de totes les instàncies en les que com a mínim es presenta aquest valor per a la propietat.
- Restriccions de cardinalitat[16]:
  - `owl:maxCardinality`. Especifica el número màxim de valors amb què la instància d'una classe es pot relacionar a través de la propietat.
  - `owl:minCardinality`. Permet especificar el número mínim de valors amb què la instància d'una classe s'ha de relacionar mitjançant la propietat.
  - `owl:cardinality`. Indica el número concret de valors amb què una instància del domini s'ha de relacionar a través de la propietat.

[13] En **OWL Lite**, el valor ha de ser un objecte.

[14] En **OWL Lite**, el valor ha de ser un objecte.

[15] No està inclosa en **OWL Lite**.

[16] En **OWL Lite**, el valors de les cardinalitats només poden ser 0 o 1.

L'exemple següent mostra com s'apliquen algunes de les restriccions comentades en les paràgrafs anteriors, en concret **owl:allValuesFrom** i **owl:minCardinality**.

```
<owl:Class rdf:about="#FinancialAnalyst">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="studiesReport"/>
        </owl:onProperty>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#studiesReport"/>
        </owl:onProperty>
        <owl:allValuesFrom>
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="#CostReport"/>
              <owl:Class rdf:about="#FinancialReport"/>
            </owl:unionOf>
          </owl:Class>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>
```

- **Instàncies**.

  En **OWL** els individus que pertanyen a una classe es poden definir de dues formes:

  - Especificant la pertinència a determinades classes i els valors de les propietats. Tot seguit se'n mostra un exemple.

```
<RefactoringOperation rdf:ID="GeneralizeRealization">
  <operationName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Generalize Realization
  </operationName>
  <description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Class that implements the refactoring operation named GeneralizeRealization
  </description>
  <className rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    edu.uoc.pfc.semanticweb.refactoring.operation.GeneralizeRealizationRefactoring
  </className>
</RefactoringOperation>
```

  - Especificant la identitat del individu. En aquest cas, **OWL** proporciona tres construccions que només s'esmentaran:

    - **owl:sameAs**. Especifica que dues referències **URI** es refereixen a la mateixa instància.
    - **owl:differentFrom**. Permet especificar que dues **URI** es refereixen a individus diferents.
    - **owl:AllDifferent**. Estipula que una llista d'individus són tots diferents. Fa ús de la propietat **owl:distinctMembers**.

2.3.- La refacció i la seva aplicació a les ontologies.

El *refactoring* va aparèixer lligat al llenguatge **Smalltalk**. En el pròleg del text de referència sobre *refactoring* (Fowler, 1999), Erich Gamma comenta que tothom que treballa amb *frameworks* sap que el codi es rellegeix i es modifica més freqüentment que no pas s'escriu i que la clau per a mantenir-lo llegible i modificable és precisament el procés de refacció.

Esmenta, però, que aquest procés és arriscat, ja que requereix fer canvis que poden introduir, subtilment, errors de difícil percepció, sobretot si el procés es realitza de manera informal. Textualment, indica que "*Per a evitar cavar la pròpia sepultura, el refactoring s'ha de realitzar sistemàticament*". Cal saber quan i en quin lloc intervenir per tal de millorar el programari i, per tant, conèixer els mecanismes del procés de refacció és clau. En aquest sentit, és important saber que per a reduir els riscos inherents al procés, cal realitzar els canvis de forma senzilla, de mica en mica –*one small step at a time*.

2.3.1.- El concepte de refacció.

En el text esmentat anteriorment –*Refactoring. Improving the Design of Existing Code.*–, Martin Fowler defineix en un primer moment la refacció de la forma següent:

> *El refactoring és el procés de canvi d'un sistema de programari de forma que no s'alteri la seva funcionalitat i es millori la seva estructura interna.*

Tal com el mateix autor indica, és una forma disciplinada de netejar el codi que minimitza les oportunitats d'introduir errors[17]. Com s'ha comentat en anteriors paràgrafs, cada pas que es realitza és senzill, inclús simplista. Per exemple, en el camp de la programació orientada a l'objecte, es tracta de moure un atribut d'una classe a una altra, extreure codi d'un mètode per a col·locar-lo dins un altre mètode de nova creació o moure part del codi d'una classe a una altra dins una jerarquia –en sentit ascendent o descendent. La conseqüència de l'acumulació d'aquests petits canvis pot ser una millora radical del disseny del programari.

Posteriorment, en el mateix text, es presenten dues definicions més de la paraula *refactoring* en funció del context en què s'utilitza –bàsicament, si es fa servir com a nom o com a verb. Són les següents:

> *Refactoring –nom–: un canvi fet en l'estructura interna d'un programari per tal de fer-lo més fàcil d'entendre i més barat de modificar sense canviar el seu comportament –funcionalitat– observable.*
>
> *Refactoring –verb–: reestructurar un programari aplicant una sèrie de refaccions sense canviar el seu comportament observable.*

És clar que la primera definició es refereix a una operació de refacció concreta, mentre que la segona descriu el conjunt del procés.

En el text esmentat es formulen, també, una sèrie de recomanacions referents a perquè i quan cal executar aquest procés de *refactoring*. En resum, són les següents:

- Perquè cal realitzar una refacció?
  - La refacció millora el disseny d'un sistema.
  - El fa més senzill d'interpretar.
  - Ajuda a trobar errades.
  - Permet realitzar el desenvolupament de forma més ràpida.
- Quan cal fer-ho?
  - Quan s'afegeixen noves funcionalitats al sistema.
  - Quan es necessita solucionar una errada.
  - Quan es revisa el sistema.

Una de les aportacions més interessants de Martin Fowler ha estat la definició d'un catàleg inicial d'operacions de *refactoring* i la seva concreció i exemplificació en el context de la programació orientada a l'objecte en **Java**. El catàleg actualitzat d'aquestes operacions es troba disponible en http://www.refactoring.com/catalog/index.html. Sobre alguna d'aquestes operacions catalogades s'hi tornarà més endavant ja que es troba en la base d'una de les operacions de refacció d'ontologies implementada.

Finalment, per a acabar aquest apartat, repetirem de nou la definició amb què s'ha començat aquest document. Com ja s'ha esmentat, es pot trobar en la Wikipèdia i té un sentit ampli en no concretar el material sobre el que s'aplica el procés:

> *La refacció o refactoring és un procés a través del qual es reescriu un material amb l'objectiu de millorar-ne tant la seva comprensió com l'estructura interna, amb el ferm propòsit de conservar-ne el significat i el comportament.*

Aquest procés es pot realitzar, en algunes circumstàncies, de forma automàtica, per la qual cosa cal:

- Decidir automàticament quines operacions es poden aplicar, és a dir, identificar el que anomenarem **oportunitats de *refactoring***.
- Executar-les, també, automàticament.

Com comenta en Jordi Conesa en el document *Pruning and Refactoring Ontologies in the Development of Conceptual Schemas of Information Systems*, "*La detecció de les oportunitats de refactoring no és determinista i requereix molta informació semàntica per tal d'identificar la millor operació a aplicar a cada cas. Per aquest motiu, la detecció només és possible en alguns casos, i inclús en aquests casos l'automatització no és trivial*".

[17] Textualment: "*Refactoring changes the programs in small steps. If you make a mistake, it is easy to find the bug.*"

2.3.2.- La refacció d'ontologies. Catàleg d'operacions. [18]

En l'apartat 1.2.1 d'aquest document s'ha comentat que les ontologies es poden crear de bell nou, o bé aprofitar-ne de creades i ajustar-les a les necessitats d'un domini concret. Si aquest és el cas, es realitzen tres tasques abans no s'utilitza l'ontologia:

- S'hi incorpora el coneixement del domini que no hi és present, és a dir, es refina l'ontologia base.
- Es poda l'ontologia obtinguda en el pas anterior, eliminant els conceptes irrellevants en el domini que es conceptualitza.
- S'efectua un procés de *refactoring* de l'ontologia obtinguda de la poda, per tal d'obtenir-ne una més reduïda i comprensible que conservi la semàntica de l'original.

En aquesta línia, hom pot definir l'aplicació a les ontologies del procés de refacció en la forma següent:

> *La refacció o refactoring és el procés de reestructurar una ontologia per tal de millorar la seva llegibilitat o estructura, preservant-ne el coneixement més rellevant.*

Tenint present que les ontologies es dissenyen per a satisfer un conjunt de requeriments, podem dir que un canvi conserva el coneixement rellevant si el que és necessari per a satisfer els requeriments no s'esborra en el transcurs del procés. En el fons, l'objectiu del *refactoring* d'una ontologia és canviar la forma en què els conceptes hi són representats.

En l'execució d'una operació de *refactoring*, una de les tasques més complicades és la modificació d'aquelles parts del sistema que fan referència als elements afectats. Aquest problema esdevé també en la refacció d'ontologies perquè aquestes solen contenir restriccions d'integritat que poden fer referència a qualsevol concepte definit en l'ontologia. Per aquest motiu, després de canviar un concepte, cal modificar totes les restriccions d'integritat que l'afecten per tal de mantenir la consistència del domini que descriu l'ontologia.

Una altra qüestió que cal tenir present en el *refactoring* d'ontologies és el fet de no poder comprovar si ha estat modificat el comportament de l'ontologia, ja que una ontologia –a diferència d'altres elements de programari– no es pot executar. Tanmateix, després de l'execució de cada operació de *refactoring*, cal portar a terme alguna mena de validació per tal de comprovar que no s'hagi perdut cap element significatiu a conseqüència dels canvis realitzats.

La figura 11 mostra el catàleg d'operacions de *refactoring* aplicables a les ontologies tal com apareixen en el text referenciat anteriorment. Aquest catàleg ha estat formalitzat a partir d'operacions de *refactoring* procedents d'altres camps –com, per exemple, la refacció del programari o de bases de dades– que s'ha considerat podien millorar determinats aspectes d'una ontologia mantenint-ne la semàntica. Com comenta l'autor del catàleg, determinades operacions poden semblar molt senzilles però, tal com ja s'ha esmentat, els passos en el procés de refacció han de ser petits i és la seva combinació la que permet aconseguir objectius més ambiciosos.

El catàleg s'ha organitzat seguint el model proposat per Martin Fowler, classificant les operacions en dues categories principals:

- Operacions estructurals: són les que permeten aplicar el *refactoring* als elements estàtics de l'ontologia –conceptes, relacions i instàncies.

  Dins d'aquesta categoria, es consideren tres subcategories:

  - *Moviment de característiques entre conceptes*: inclou les operacions que canvien el context en què es defineixen les propietats d'una ontologia.
  - *Organització de dades*: categoritza les operacions que canvien l'estructura de l'ontologia per a poder treballar més senzillament amb les seves dades.

    A aquest grup pertany una de les operacions implementades: *Implicit Subsets*. Tal com es veurà posteriorment, aquesta operació esborra una generalització redundant.
  - *Relacions de generalització*: inclou les operacions utilitzades per a moure les propietats a través de la taxonomia de l'ontologia o bé fer canvis en les relacions.

    La segona de les operacions implementades –*Generalize Realization*– correspon a aquesta subcategoria. L'operació canvia un participant en una relació pel seu pare.

[18] La informació d'aquest apartat ha estat treballada a partir dels documents *Pruning and Refactoring Ontologies in the Development of Conceptual Schemas of Information Systems* i *Modelització Conceptual de Sistemes d'Informació*.

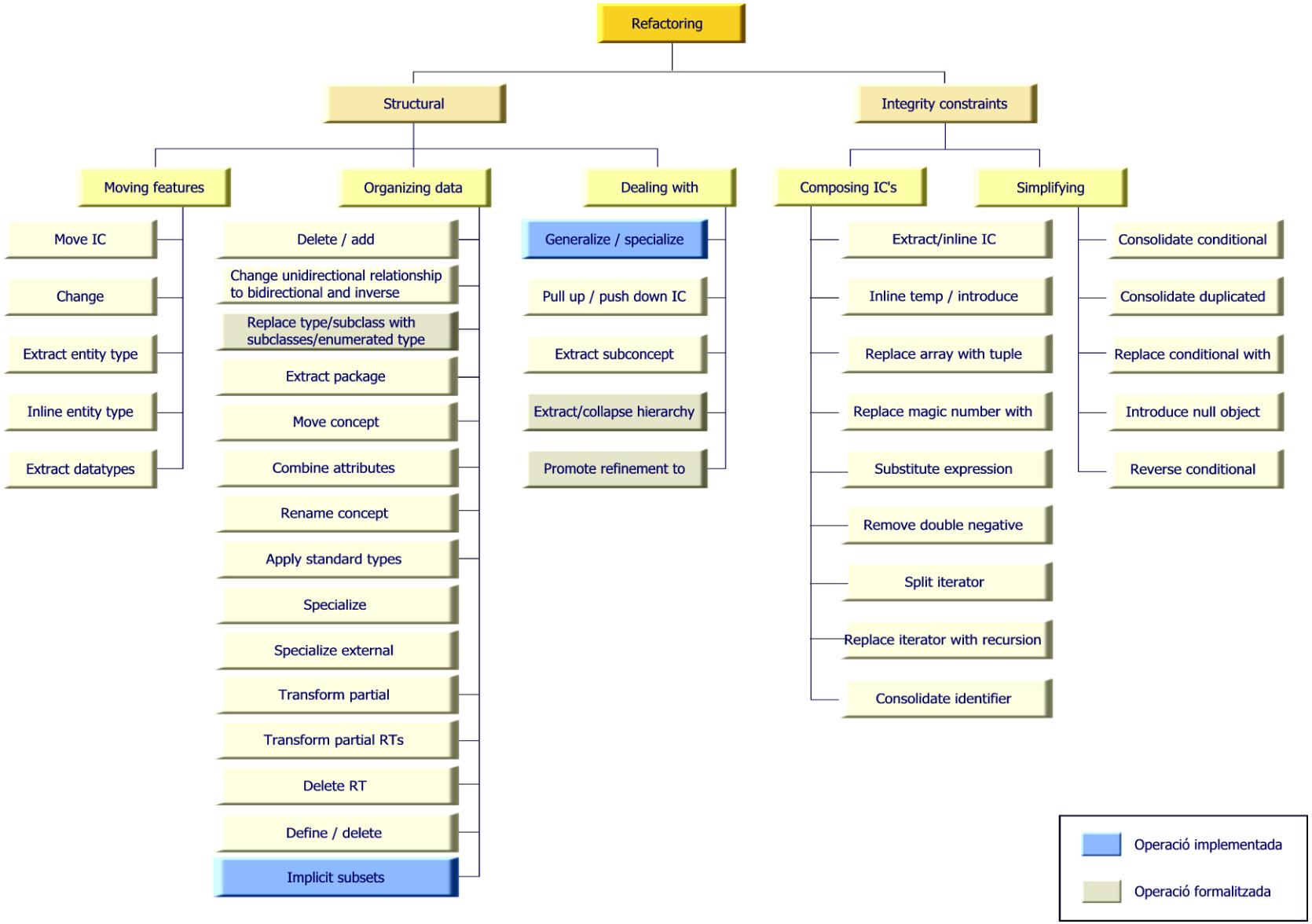


Fig. 11 - Refactoring d'ontologies. Catàleg d'operacions.

- Operacions associades a les restriccions d'integritat: permeten aplicar la refacció als elements dinàmics d'una ontologia i al contingut de les restriccions d'integritat.

  Com a subcategories es consideren:

  - *Composició de restriccions d'integritat*: inclou les operacions que milloren la qualitat de les restriccions d'integritat.
  - *Simplificació d'expressions condicionals*: categoritza les operacions que simplifiquen expressions condicionals.

Les operacions de la primera categoria es poden aplicar a qualsevol ontologia, mentre que les de la segona són només rellevants per a aquelles ontologies que contenen operacions o restriccions d'integritat.

2.3.3.- El procés de refacció automàtica.

El principal problema en l'ús de les tècniques de refacció és com detectar les oportunitats de refacció, és a dir, quina o quines són les condicions que indiquen que l'execució d'una determinada operació millora l'ontologia i a quins elements d'aquesta cal aplicar-la. Algunes d'aquestes oportunitats es poden detectar de forma automàtica, de forma que la proposta del document *Pruning and Refactoring Ontologies in the Development of Conceptual Schemas of Information Systems* és dividir l'activitat de *refactoring* en dues parts:

- Una automàtica, en què un conjunt d'algorismes analitza l'ontologia per tal de detectar les oportunitats de *refactoring* que redueixen la grandària de l'ontologia mantenint la informació rellevant.

  En aquest procés, un cop es detecta una oportunitat, s'executa l'operació associada.

- Una altra manual, en què el sistema ajuda al dissenyador en el procés de reestructuració de l'ontologia.

⇒ En general, l'ontologia a la que se li aplica la refacció estarà formada per el conjunt d'elements que es comenten tot seguit[19].

- **Conceptes**.

  Són les abstraccions que representen les característiques que són comunes a determinades instàncies. Els conceptes poden ser de dos tipus:

  - Tipus d'entitat: són conceptes en què les instàncies –anomenades entitats– són objectes individuals i identificables existents en el domini en un instant determinat.
  - Tipus de relació –que entendrem genèriques.

    Un tipus de relació és un concepte en què les instàncies són conjunts distints formats per entitats, cadascuna d'un determinat tipus, que en un instant determinat estan relacionades en l'ontologia i juguen cadascuna un determinat paper en la relació.

    Un tipus de relació genèric és un tipus de relació normal amb un determinat número de realitzacions i una restricció associada a la realització. Cada realització és un conjunt de tipus d'entitat que participen en una concreció del tipus de relació, essent cadascun d'ells subtipus –directes o indirectes– dels participants que defineixen el tipus de relació.

  En el diagrama **UML** de la figura 12 apareixen, concretats, exemples del que s'acaba de comentar. Es poden observar tipus d'entitat, com *Person* i *Report*, i tipus de relació com *Studies*, aquest últim amb una parell de realitzacions concretes:

  {analyst:FinancialAnalyst, report:CostReport} i *{analyst:FinancialAnalyst, report:FinancialReport}*

[19] No s'entra en excessius detalls ni s'utilitzen excessivament les expressions de tipus formal. En l'apartat 8.2 –*Automatic Refactoring Problem*– del document citat anteriorment es troba perfectament explicat el que aquí només s'apunta.

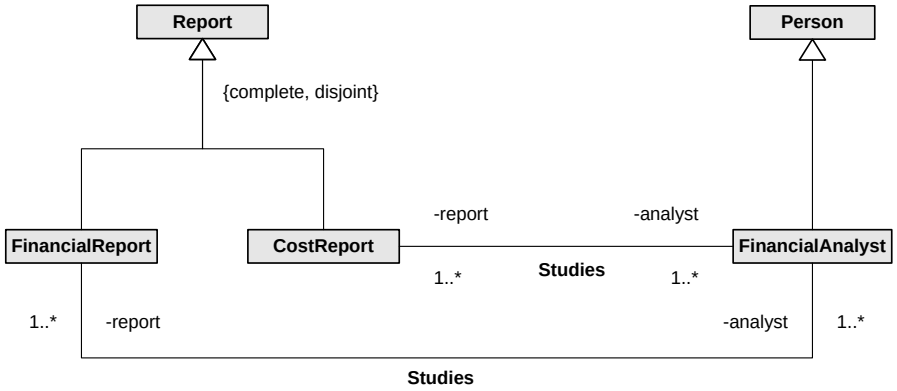


Fig. 12 - Tipus d'entitat i tipus de relació.

- **Relacions de generalització**.

  Com indiquen el nom, són relacions entre conceptes del tipus *IsA(C_1,C_2)* en què *C_1* és el subtipus i *C_1* el tipus pare.

  Per exemple, en el diagrama anterior la relació entre *FinancialAnalyst* i *Person*.

- **Restriccions d'integritat estàtiques**.

  Especifiquen les condicions a satisfer pels elements que formen part de l'ontologia en qualsevol estat d'aquesta.

  Tenint present que es treballarà sobre ontologies expressades en **OWL**, esmentarem, a més de les restriccions generals, les següents:

  - Restriccions de completesa –*covering*. En una relació de generalització, significa que no hi pot haver una instància del pare que no sigui d'algun dels fills –és el cas de la jerarquia *Report* en el diagrama de la figura.
  - Restriccions de disjunció –*disjoint*: entre un determinat conjunt de conceptes no poden existir instàncies que pertanyin a més d'un d'ells –també és el cas de la jerarquia *Report*.
  - Restriccions de cardinalitat. Expressen els nombres màxim i mínim de les instàncies d'un tipus que es poden relacionar amb un determinada instància d'un altre tipus a través d'un tipus de relació.

    En l'exemple, un analista estudiarà un o més informes.
  - Restriccions de realització. Expressen que si unes entitats participen en una instància d'un tipus de relació és que són instàncies dels tipus d'entitat que defineixen el tipus de relació.

Cal esmentar, també, que es defineix la **grandària d'una ontologia** com la suma del número de conceptes i del de restriccions d'integritat estàtiques.

⇒ El problema de refacció d'una ontologia es pot formular en la forma següent:

> *Si es disposa d'un conjunt d'operacions de refactoring –o_1,....,o_n– i d'una ontologia podada –O_P–, l'objectiu del procés de refactoring automàtic és detectar automàticament les oportunitats de refacció de forma tal que l'execució sobre O_P de les operacions de refacció associades redueixi la grandària de l'ontologia, produint com a resultat un esquema conceptual C_S amb les següents característiques:*
>
> - *C_S inclou tots els veritables conceptes d'interès directe de l'ontologia –és a dir, és semànticament equivalent a O_P.*
> - *C_S és sintàcticament correcte.*
> - *C_S és més petit que O_P i és l'esquema mínim possible.*

2.3.4.- Formulació dels algorismes de refactoring implementats.

En aquest **PFC** s'han implementat dues de les operacions de refacció definides en el catàleg de la figura 11. Tot seguit es detallen, en forma resumida, els algorismes implementats.

2.3.4.1.- Implicit Subsets.

Aquesta operació és una adaptació de l'operació del mateix nom definida en el text *Diseño Conceptual de Bases de Datos* (Batini, C; et al., 1994, pág. 169). Segons s'esmenta en aquest text, després d'integrar esquemes conceptuals procedents de distintes fonts, alguns subconjunts es podrien derivar d'altres ja presents en l'esquema final, la qual cosa introdueix redundància en el model definitiu. El propòsit de l'operació, és precisament eliminar una relació de generalització redundant.

Una generalització és redundant quan dos conceptes es relacionen per més d'un camí, bé de forma o de forma indirecta. Formalment, es pot expressar que una relació de generalització *IsA(Child, Parent)* és redundant si existeix un altre concepte *Mid*, tal que:

$$IsA^{+}(Child, Mid) \wedge IsA^{+}(Mid, Parent)$$

En la part esquerra del diagrama de la figura 13 se'n pot observar un exemple: la generalització entre *Analyst* i *Employee* és redundant, ja que el primer concepte ja està relacionat amb *Employee* via *ComputerExpert*.

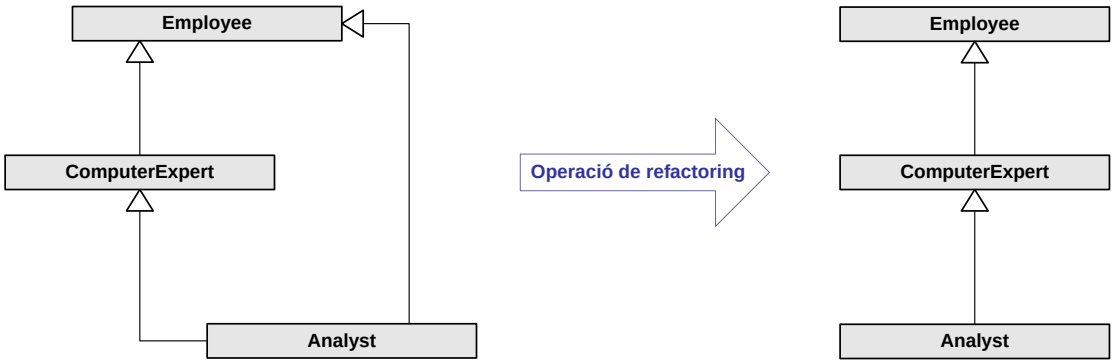


Fig. 13 - Operació *Implicit Subsets*.

Un cop aplicada l'operació s'obté la jerarquia de classes situada en la part dreta del diagrama, en què la redundància ha estat eliminada.

La taula següent conté tota la informació necessària per a l'execució de l'operació.

Operació *Implicit Subsets*	
Signatura	L'operació té dos paràmetres que representen els conceptes relacionats a través de la relació de generalització redundant: `ImplicitSubsets(Child, Parent)`
Precondicions	S'han de satisfer dues precondicions: - Ha d'existir una relació de generalització directe entre els conceptes fill i pare. - Ha d'existir, com a mínim, un camí de generalització –per naturalesa, indirecte– entre els conceptes fill i pare.
Postcondicions	Un vegada aplicada l'operació, es compleix la següent postcondició: - La relació de generalització directe entre els conceptes fill i pare ha estat esborrada.
Efecte en la qualitat de l'ontologia resultant	L'eliminació d'una generalització redundant sempre millora la qualitat de l'ontologia.
Quan cal aplicar-la	Com s'ha esmentat, l'execució d'aquesta operació sempre millora la qualitat de l'ontologia, cosa que s'observa en la reducció de la seva grandària: `nova_grandaria = vella_grandaria - 1` Per tant, aquesta operació cal aplicar-la sempre.

Taula 6 – L'operació *Implicit Subsets*.

2.3.4.2.- Generalize Realization.

Aquesta operació està basada en una de les operacions definides per Martin Fowler: *Pull up field*. La figura 14 mostra, en la part esquerra, dues classes germanes amb un atribut del mateix tipus i amb la mateixa semàntica. En aquest cas, el procés de refacció situa l'atribut en la classe base de la jerarquia, obtenint el diagrama de la part esquerra que conserva la semàntica de l'anterior.

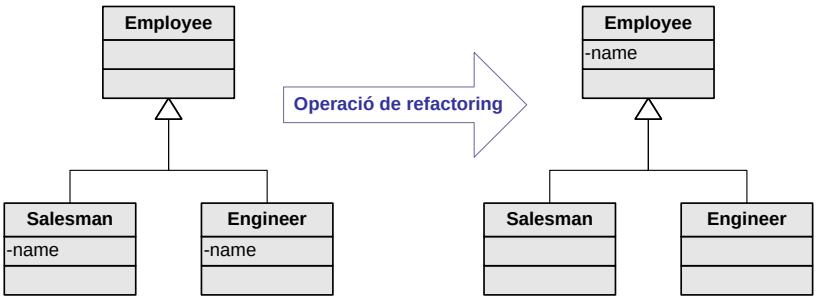


Fig. 14 - Operació *Pull up field.*

En la seva aplicació al *refactoring* d'ontologies, es tracta de substituir un conjunt de realitzacions d'un determinat tipus de relació genèrica definida en tipus d'entitat que són germans per realitzacions del mateix tipus en què intervenen els pares comuns.

El gràfic de la figura 15 mostra l'aplicació d'aquesta operació a un model que ja ha estat comentat en l'apartat 2.3.3. Es pot observar el perquè del nom de l'operació definida per Martin Fowler: la realització ha pujat –*pull up*– des de les classes filles al pare.

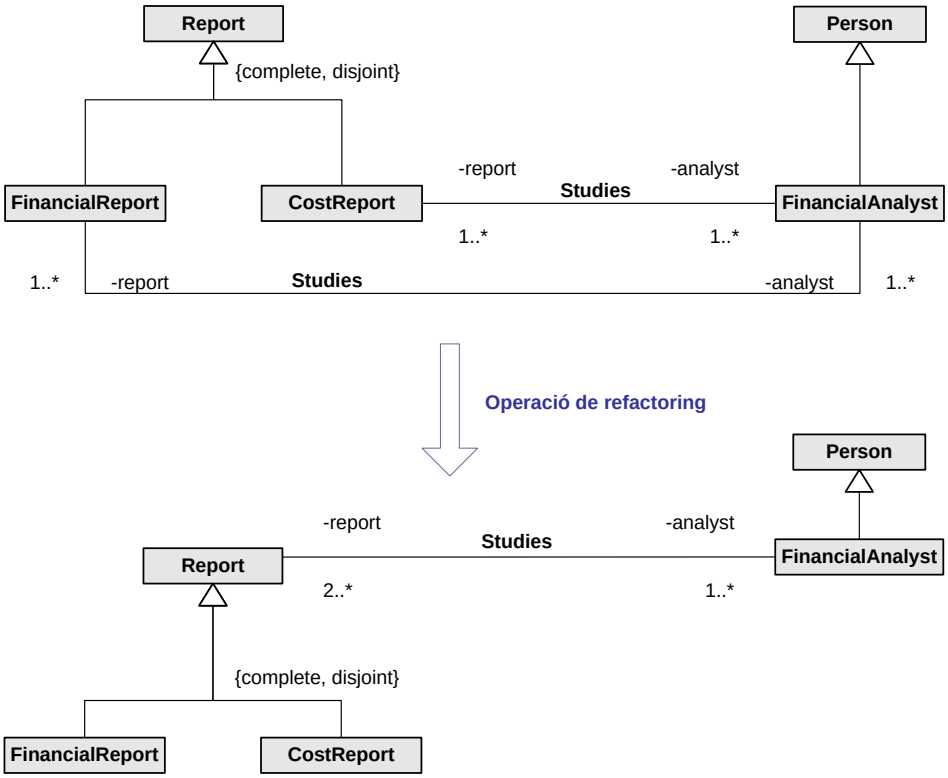


Fig. 15 - Operació *Generalize Realization.*

L'operació es pot formular, formalment, de la manera següent:

- Suposem que R és un tipus de relació genèrica, en què intervenen n tipus d'entitat E_i, cadascun d'ells jugant un determinat paper p_j. Expressarem aquest fet en la forma:

$$R(p_1{:}E_1,\ldots, p_n{:}E_n)$$

- Considerem que disposem d'un conjunt de m realitzacions de R que indicarem mitjançant:

$$r_i = \{p_1{:}E_{1,i},\ldots, p_n{:}E_{n,i}\} \text{ on } 1 \leq i \leq m$$

En la figura anterior, tenim dues realitzacions de *Studies*:

{analyst:FinancialAnalyst,report:CostReport} i *{analyst:FinancialAnalyst,report:FinancialReport}*

- Si es compleix que:
  - els elements que ocupen el lloc *k* en les *m* realitzacions són germans,
  - i els altres elements son iguals en la resta de posicions

  En l'exemple, en *k=2* tenim *CostReport* i *FinancialReport* que són germans, i el primer lloc l'ocupa *FinancialAnalyst* en ambdues realitzacions.
- Llavors, si *s* és el número de pares comuns als elements que ocupen el lloc *k* en les anteriors realitzacions, aquestes es poden substituir per *s* realitzacions definides en la forma següent:
  - Les noves realitzacions tenen el mateix número de participants.
  - En el lloc *k*, els fills se substitueixen pels distints pares comuns.
  - La resta de participants són els mateixos.

  Aquesta conclusió es pot representar a través de l'expressió:

  $$rp_l = \{p_1:E_{1,l},\ldots\ p_k:C_{Pl},\ldots,\ p_n:E_{n,l}\}\ \text{on}\ 1 \leq l \leq s$$

  En la figura 15 es pot observar com les dues realitzacions anteriors de *Studies* han estat substituïdes per la realització:

  {analyst:FinancialAnalyst,report:Report}

L'execució d'aquesta operació de *refactoring* ha de seguir el procés que s'acaba de descriure. Ara bé, per a que les noves realitzacions siguin semànticament equivalents a les anteriors, cal que en el procés de refacció es calculin adequadament les restriccions de cardinalitat que cal associar a les noves realitzacions. La taula següent resumeix els càlculs que s'han de realitzar i n'explica breument els motius.

Canvis en les cardinalitats de les noves realitzacions			
Participant	**Cardinalitat**	**Tipus de generalització**	
		Completa –*covering*–	**No completa –*non-covering*–**
k	**Mínima**	Igual al **valor més baix** de les cardinalitats mínimes del participant número ***k*** en les ***m*** realitzacions. Totes les instàncies de qualsevol pare són, alhora, d'un o més fills, per la qual cosa cal només garantir que se satisfaci la cardinalitat més baixa entre tots els fills.	Igual a **0**. Pot existir una instància d'un pare que no sigui, alhora, instància de cap fill, per la qual cosa cal crear una nova restricció de cardinalitat que indiqui que no totes les instàncies del pare participen en la relació.
	Màxima	Igual al **valor més alt** de les cardinalitats màximes del participant número ***k*** en les ***m*** realitzacions. El motiu coincideix amb l'expressat per a la cardinalitat mínima.	Igual al **valor més alt** de les cardinalitats màximes del participant número ***k*** en les ***m*** realitzacions. A més, caldrà afegir una restricció d'integritat per a indicar que només poden participar en la relació instàncies directes dels fills.
i (i≠k)	**Mínima**	Igual a la **suma de les cardinalitats mínimes** del participant número ***i*** en les ***m*** realitzacions. Les instàncies de cada pare C_p són la unió de les instàncies dels fills que participen en les ***m*** realitzacions. Per tant, ***i*** ha de participar almenys amb un número d'instàncies de C_p igual a la suma de les cardinalitats mínimes dels anteriors participants.	Igual a **0**. Com no totes les instàncies dels pares participen en la relació, la cardinalitat mínima ha de ser nul·la.
	Màxima	Igual a la **suma de les cardinalitats màximes** del participant número ***i*** en les ***m*** realitzacions. El motiu és el mateix que per a la cardinalitat mínima. Ara però, ***i*** participarà com a màxim amb un número d'instàncies de C_p igual a la suma de les cardinalitats màximes dels anteriors participants.	Igual a la **suma de les cardinalitats màximes** del participant número ***i*** en les ***m*** realitzacions. Com a màxim, el participant ***i*** es relacionarà amb un número d'instàncies igual a la suma de les cardinalitats màximes dels anteriors participants.

Taula 7 - *Generalize Realization*. Canvis en les cardinalitats de les noves realitzacions.

Com s'ha indicat, quan la generalització entre els fills i almenys un dels pares no és completa, cal afegir a l'ontologia una restricció d'integritat per tal d'evitar que instàncies directes dels pares participin en el tipus de

relació. Aquesta nova restricció d'integritat –que anomenarem de tipus T1– expressa el fet que el participant *k* en *R* ha de ser una instància directa d'algun dels fills. S'indica en la forma següent:

$$R(x_1,\ldots,x_{k-1},x_k,x_{k+1},\ldots,x_n) \rightarrow E_{k,1}(x_k) \vee \ldots \vee E_{k,m}(x_k)$$

Tenint present que la relació de generalització és completa, l'aplicació d'aquestes regles al cas de la figura 15 portarà a les següents conclusions:

- Per al nou participant *k=2* –*Report*–:
  - cardinalitat mínima: *1*, que és la cardinalitat comuna a *CostReport* i *FinancialReport*.
  - cardinalitat màxima: ∞, cardinalitat també comuna a *CostReport* i *FinancialReport*.
- Per al participant *i=1* –*FinancialAnalyst*–:
  - cardinalitat mínima: *2*, que és la suma de les cardinalitats que *FinancialAnalyst* presenta amb *CostReport* i *FinancialReport*.
  - cardinalitat màxima: ∞.

Com en el cas de l'operació anterior, mostrarem a través d'una taula la informació necessària per a l'execució de l'operació.

Operació *Generalize Realization*	
Signatura	L'operació té dos paràmetres que representen, respectivament, el conjunt de les realitzacions a substituir –**`OldRealizations`**– i el conjunt que conté les realitzacions a crear –**`NewRealizations`**–: **`GeneralizeRealization(OldRealizations, NewRealizations)`**
Precondicions	S'han de satisfer les següents precondicions: - Cap de les noves realitzacions ha d'existir en l'ontologia. - Totes les velles realitzacions han de ser del mateix tipus de relació. - Fóra del participant que ocupa la posició, ***k***, la resta de participants en les noves i velles realitzacions han de ser els mateixos en cadascuna de les altres posicions. - Els participants que ocupen el lloc ***k*** en els noves realitzacions són el pares comuns directes dels que ocupen el mateix lloc en les velles realitzacions.
Postcondicions	Un vegada aplicada l'operació –d'acord amb el procés descrit en els paràgrafs anteriors–, es compleixen les següents postcondicions: - Les noves realitzacions han estat creades en l'ontologia. - Les velles han estat esborrades. - S'han tornat a calcular les relacions de cardinalitat d'acord amb les indicacions recollides en la taula 6. - Si la generalització entre algun pare i els fills no és completa, s'han incorporat a l'ontologia les restriccions d'integritat de tipus T1 adients.
Efecte en la qualitat de l'ontologia resultant	Aquesta operació pot millorar la llegibilitat i el manteniment d'una ontologia. Normalment, si només existeix un pare, resulta una ontologia més petita, precisa i manejable. Ara bé, quan la generalització entre pares i fills no és completa o si el número de pares és superior al de fills, l'ontologia es torna més gran, complexa i poc manejable.

Operació *Generalize Realization*	
Quan cal aplicar-la	Aquesta operació s'hauria d'aplicar en les següents condicions: - El nombre de fills és superior al de pares. - Existeix un subconjunt de fills pels quals la generalització amb cada pare del conjunt de pares és completa. - Les restriccions de cardinalitat del participant ***k*** en les velles realitzacions són iguals. En aquestes condicions, la grandària de l'ontologia es pot calcular a través de la següent expressió: **`nova_grandaria = vella_grandaria + 2n(s-m) + w + v`** on: - **`n`** és el grau del tipus de relació. - **`s`** és el número de pares. - **`m`**, el de realitzacions. - **`v`**, el número de participants ***i*** en les velles realitzacions amb una cardinalitat mínima superior a zero. - **`w`**, el número de participants ***i*** en les velles realitzacions amb una cardinalitat màxima inferior a infinit. És fàcil deduir que: **`nova_grandaria ≤ vella_grandaria`** si **`2n(s-m)≤w + v`** Per tant, cal aplicar aquesta operació, és a dir, existeix l'oportunitat de *refactoring*, quan es compleix: **`2n(s-m)≤w + v`**

Taula 8 - L'operació *Generalize Realization*.

Si s'apliquen aquests criteris al cas que hem estat considerant com a exemple, tenim:

- El número de fills –*2*– és superior al de pares –*1*.
- Entre *CostReport*, *FinancialReport* i *Report*, la generalització és completa.
- La cardinalitat de *CostReport* i *FinancialReport* és la mateixa.
- *n = 2*, *s = 1*, *m = 2*, *v = 1* i *w = 0*, amb la qual cosa *2n(s-m) = -4* i *w+v = 1*, complint-se la condició que fa més petita l'ontologia.

Per tant, es podria aplicar de forma automàtica l'operació al cas plantejat. A més, no caldria afegir la restricció d'integritat de tipus T1.

2.3.5.- Els algorismes implementats i OWL.

A l'hora d'adequar els algorismes esmentats en l'apartat anterior a una ontologia expressada en **OWL** cal tenir presents una sèrie d'aspectes que afecten, sobretot, a la segona de les operacions. Les qüestions a considerar són les següents:

- **OWL** admet les jerarquies de classes, per la qual cosa no hi ha cap problema per a modelar les relacions de generalització.
- En **OWL**, les relacions s'expressen mitjançant les propietats. Aquestes són relacions binàries dirigides.
- Una associació **UML** navegable en el dos sentits es transforma en **OWL** en dues propietats de tipus *object property*, de forma que una és la inversa de l'altra.
- Un atribut de classe no lligat a cap objecte del domini –per exemple, una dada de tipus enter, una cadena o una data–, l'expressarem en **OWL** a través d'una propietat del tipus *data type property*. Aquest tipus de propietats són binàries unidireccionals i, per tant, no tenen inversa.

A més, aquest tipus de propietats tindran sempre cardinalitats 0...*.[20]

- Referent a l'operació *Generalize Realization*, com en **OWL** totes les relacions són binàries, l'element ***k*** a substituir només pot ser el primer o el segon, la qual cosa simplifica la implementació de l'algorisme.
- En **OWL**, una especialització completa –és a dir, de tipus *covering*– s'expressa indicant que el pare és equivalent a la unió dels fills.
- Quan, en executar l'operació *Generalize Realization*, cal incloure una restricció de tipus T1, ho expressarem en **OWL** introduint una restricció de tipus `allValues`.

2.4.- La plataforma Protégé.

Finalment, per a acabar aquest capítol en què s'ha estat tractant sobre l'estat de l'art, comentarem de forma breu alguns aspectes de la plataforma sobre la que s'executa el *plugin* desenvolupat.

Protégé és un *framework* que va ser desenvolupat inicialment per un grup liderat per Mark Munsen en el centre d'Informàtica Mèdica de la Universitat de Stanford –*Stanford Medical Informatics*. Es presenta com una plataforma flexible i configurable que permet el desenvolupament d'aplicacions i components basats en models, i disposa d'una arquitectura oberta que possibilita afegir-hi fàcilment noves extensions –*plugins*.

El nucli central de l'aplicació –el que es coneix com a **Protégé-Frames**– permet treballar amb sistemes basats en marcs –*frames*–, en què els conceptes que són objectes es representen mitjançant classes; les propietats de les classes –com, per exemple, un nom o edat– pel que anomenem *slots* –camps; i les restriccions sobre les propietats o relacions entre classes i/o *slots* pel que rep el nom de *slot faces* –característiques o facetes del camp. **Protégé** disposa també d'un editor **OWL** que proporciona les utilitats necessàries per a editar, visualitzar i analitzar ontologies **OWL**.

2.4.1.- L'API OWL.

Protégé disposa d'un conjunt d'eines de desenvolupament –anomenat *Protégé Programming Development Kit* (PDK)– que descriu com desenvolupar i instal·lar *plugins* que estenen la plataforma i com treballar amb les API que aquesta incorpora.

La plataforma disposa d'una API central –l'anomenada *core* API– emprada per a accedir a les funcionalitats bàsiques de l'eina i a les bases de coneixement basades en marcs, bases que han estat creades a través de **Protégé-Frames**.

A més, disposa de l'API per a **OWL** que estén el nucli central de l'eina per tal de proporcionar accés a ontologies **OWL** creades mitjançant el plugin **Protégé-OWL**. No cal emprar, però, **Protégé** per a accedir a les dues API, ja que es poden utilitzar des de qualsevol aplicació autònoma –com, per exemple, aplicacions **Swing**, *servlets*,...– i crear o accedir a través d'elles als models de coneixement **Protégé**. Aquesta API presenta les següents característiques:

- És una llibreria de codi obert desenvolupada en **Java**.
- Ha estat pensada per a treballar amb ontologies descrites emprant els llenguatges **OWL**, **RDF** i **RDFS**.
- Proporciona tot un conjunt d'interfícies i classes amb els corresponents mètodes que permeten:
  - Carregar, crear i salvaguardar models conceptuals expressats en **OWL**.
  - Manipular ontologies **OWL** i realitzar consultes sobre elles.
  - Realitzar raonaments emprant motors *Description Logic*.
- Ha estat optimitzada per la implementació d'interfícies gràfiques d'usuari.

La figura 16 mostra el diagrama de classes d'aquesta API. La guia per a utilitzar-la i exemples detallats sobre el seu ús es poden consultar en format web en l'adreça http://protege.stanford.edu/plugins/owl/api/guide.html.

[20] Per exemple, una propietat de nom `date` associada a un ***Report*** serà del tipus `xsd:date`. En general, un determinat valor d'aquest tipus pot no estar lligat a cap instància de ***Report***, o bé estar-ho a molts.

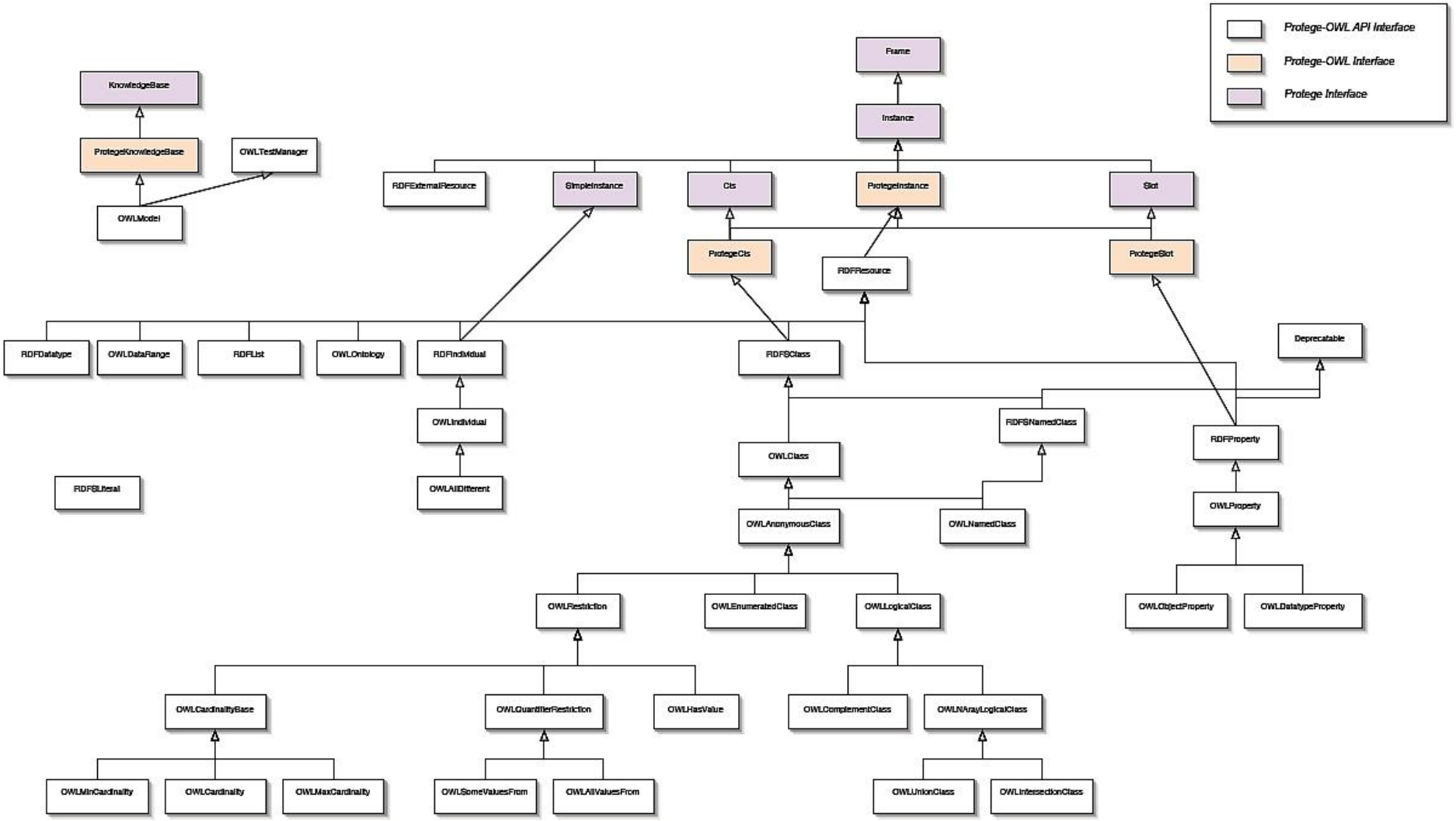


Fig. 16 - Diagrama de classes de l'API Protégé-OWL.

2.4.2.- Desenvolupament de plugins.

Com en d'altres contexts, un *plugin* no és res més que una extensió de **Protégé**. La plataforma ofereix sis tipus bàsics de *plugins*. Es comenten tot seguit.

- **Tab widget**: en aquest tipus –el més senzill de tots–, la interacció de l'usuari amb el *plugin* té lloc en una pestanya associada al *plugin* que apareix en la finestra principal de **Protégé**. És el tipus de *plugin* emprat en aquest projecte.
- **Slot widget**: apareix en un formulari i s'usa per adquirir i visualitzar les dades associades a un *slot* en una instància.
- **Back end**: permet especificar el mecanisme que **Protégé** utilitzarà per a emmagatzemar la informació –sigui com a fitxer de text o en una base de dades.
- **Create project**: permet llegir un fitxer en un determinat format generat des d'una altra aplicació i crear, a partir d'ell, un model de coneixement **Protégé**.
- **Export**: proporciona el mecanismes per a exportar els models **Protégé** en distints formats.
- **Project**: permeten la manipulació de projectes **Protégé**.

A més, l'API **Protégé-OWL** permet desenvolupar els següents tipus addicionals:

- **OWL model action**: possibilita la incorporació de noves opcions al menú i a la barra d'eines principal.
- **Resource action**: permet incorporar noves funcionalitats als menús contextuals dels diferents recursos –classes, propietats, ...
- **Ontology test**: per a incorporar noves funcionalitats als test d'ontologies.
- **Resource display**: per a poder afegir components nous components als formularis.

El procediment per a crear un *plugin* per a **Protégé** és relativament senzill. Només cal:

- Crear una classe que hereti de la classe que representa el tipus de *plugin* a crear –en el cas del **PFC**, cal crear una subclasse de **AbstractTabWidget**.
- Implementar els mètodes que es requereixen de la classe –en el cas del projecte, el mètode `initialize()`.
- Generar un fitxer de manifest –de nom **MANIFEST.MF**– que contingui la informació que **Protégé** necessita per a que el *plugin* desenvolupat s'integri adequadament en l'eina.

  El manifest ha de començar amb la següent línia de declaració:

  ```
  Manifest-Version: 1.0
  ```

  i, després d'una línia en blanc, ha de contenir una secció que descriu el *plugin* a través d'una estructura de línies que segueixen el format *clau-valor*, cadascuna separada de les altres mitjançant una línia en blanc. A més, cal que hi hagi també una nova línia en blanc al final del fitxer. En el cas d'un *plugin* **tab widget**, la informació necessària és la següent:

  - El nom de la classe principal del *plugin* com a valor de la clau `Name`:

    ```
    Name: edu/uoc/pfc/semanticweb/refactoring/widget/RefactoringWidget.class
    ```

  - El tipus de *plugin*. Per a aquest atribut, el nom de la clau és precisament el tipus de plugin:

    ```
    Tab-Widget: True
    ```

- Finalment, és normal empaquetar el *plugin* en un fitxer JAR i situar-lo en el directori de *plugins* de **Protégé**. Posteriorment, des de l'entorn de treball de la plataforma s'activa el *plugin*.
- Si el *plugin* depèn d'altres, com és el cas del *plugin* desenvolupat, cal declarar aquestes dependències a través d'un fitxer de *properties* anomenat **plugin.properties** i situar aquest fitxer en el mateix directori del *plugin*.

  En aquest fitxer de dependències s'indica la relació de directoris on se situen els *plugins* dels que es depèn. En aquest cas, es depèn només de **Protégé-OWL *Plugin*** i ho indicarem:

  ```
  plugin.dependency.count=1
  plugin.dependency.0=edu.stanford.smi.protegex.owl
  ```

La primera propietat –*count*– expressa el nombre de dependències, cadascuna de les quals s'indica mitjançant el valor assignat a la clau de nom `plugin.dependency.<num_id>` , on `num_id` és el número correlatiu que identifica l'entrada.

3.- Disseny.

3.1.- Introducció.

L'aplicació que es descriu és un *plugin* per a **Protégé** desenvolupat en **Java** que utilitza l'API **OWL** de l'esmentada plataforma. Aquesta és una llibreria de codi obert pensada per a treballar amb elements relacionats amb els llenguatges **OWL**, **RDF** i **RDFS**, proporcionant classes i mètodes que permeten manipular models conceptuals expressats en **OWL**.

Un cop el *plugin* ha estat incorporat a **Protégé**[21], quan l'usuari executa aquest programa i obre un determinat projecte, li apareix, a més de les normals, una nova pestanya que li permet realitzar les operacions de *refactoring* definides en el fitxer de configuració del *plugin*.

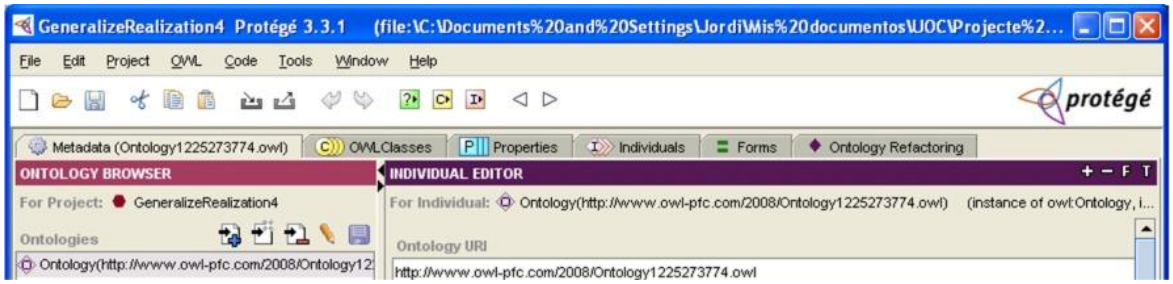


Fig. 17 - El *plugin* i Protégé

Tenint presents els objectius i l'enfocament indicats en l'apartat d'introducció d'aquest document, el disseny de l'aplicació s'ha realitzat d'acord amb les següents pautes:

- El *plugin* a implementar és del tipus **Tab Widget**, la qual cosa obliga a derivar una classe de la classe **AbstractTabWidget** del paquet **edu.stanford.smi.protegex.owl.ui.widget** i a implementar el mètode `initialize()` que aquesta hereta de la classe **Widget**.
- Una característica fonamental del *plugin* és que ha de permetre afegir noves operacions a les ja implementades sense haver de modificar el codi elaborat. Per tal d'aconseguir aquesta finalitat, s'ha treballat en la línia següent:
  - Totes les operacions de *refactoring* deriven d'una classe abstracta que defineix un mètode abstracte, `execute()`, a implementar per les subclasses hereves. Aquest serà l'encarregat de detectar l'oportunitat de *refactoring* i d'executar l'operació, la qual cosa farà utilitzant els serveis d'altres classes auxiliars específiques per a cada operació.
  - Les oportunitats de *refactoring* es representen a través d'una interfície que defineix el mètode a especialitzar per cada classe que la implementi, classe que serà específica per a cada operació concreta.
  - El conjunt d'operacions disponibles està definit en un fitxer de configuració que es carrega en la inicialització del *plugin*.
  - Durant l'execució es creen els objectes de les classes que corresponen a les distintes operacions definides en el fitxer.
- La interacció de l'usuari amb el *plugin* es realitza a través d'una interfície gràfica d'acord amb les característiques presents en **Protégé**. A més de les prestacions que ja incorpora aquesta eina, el *plugin* en concret ha de permetre:
  - Executar el conjunt d'operacions de *refactoring* definides en el fitxer de configuració, de forma conjunta o seleccionant el subconjunt d'operacions a executar.
  - Conèixer els resultats de l'execució, de forma que se sàpiga si s'han o no detectat oportunitats de *refactoring* i quines han estat executades.
  - Poder emmagatzemar aquests resultats –mitjançant *logs*– i els models intermedis obtinguts.

Arquitectònicament, l'aplicació s'ha desenvolupat seguint en part el model MVC, conegut també com a Model 2. En aquest model, els objectes que formen part de l'aplicació es divideixen en tres tipus –**Model**, **Vista** i

[21] El procés d'instal·lació es descriu en l'apartat 5.1 d'aquest document.

Control– i s'associen respectivament a tres capes, cadascuna de les quals presenta un comportament perfectament definit.

La capa de presentació engloba les vistes –només una en aquest cas–, que s'encarreguen de mostrar a l'usuari la part del domini que requereix en un instant determinat. El model ens permet representar els objectes que pertanyen al domini del problema. En l'aplicació que ens ocupa, es pot considerar que el model està format per l'ontologia sobre la que s'apliquen les operacions de *refactoring*.

Finalment, l'arquitectura MVC contempla un tercer tipus d'objecte: els objectes de control. Aquests estan relacionats amb una capa, anomenada d'aplicació, i implementen la lògica d'aquesta proporcionant-li la funcionalitat desitjada. Formaran part d'aquesta capa totes les classes i interfícies relacionades amb les operacions de refacció.

A més, en el *plugin* desenvolupat, les peticions que genera l'usuari mitjançant la seva interacció amb la pestanya del *plugin* –la vista–, son recollides per una classe controladora –**RefactoringController**– que podrà accedir, directament o a través de les classes que representen les operacions o d'altres classes d'utilitat, als objectes del domini –el model– o crear-ne de nous. Aquesta informació generada pel sistema, la posarem de nou a disposició de l'usuari final mitjançant un canvi en la vista.

De cara al desenvolupament i al manteniment de l'aplicació, aquesta divisió de responsabilitats permet separar la lògica de l'aplicació –el control– de la lògica de la presentació –la vista–, la qual cosa permet reutilitzar la lògica de l'aplicació i evita que els canvis en la presentació influeixin en la lògica de l'aplicació.

Val a dir que per a propagar a la vista el que està succeint en el rerefons, s'utilitzarà el mecanisme que ofereix **Java** a través de la interfície **PropertyChangeListener** i la classe **PropertyChangeSupport**.

Finalment, indicar que la classe controladora **RefactoringController** constitueix, conjuntament amb la classe abstracta i la interfície esmentades anteriorment, la classe **RefactoringWidget** i el fitxer de configuració, el *framework* que forma el nucli central del *plugin* i ofereix el marc general d'execució estable on s'hi poden encabir noves operacions de *refactoring* sense haver de modificar l'estructura creada.

En els paràgrafs següents, s'explica com s'han concretat aquestes pautes de disseny amb l'ajut de diagrames **UML**.

3.2.- Disseny del *plugin*.

La figura 18 mostra el diagrama de casos d'ús d'acord amb les pautes especificades en l'apartat anterior.

Observant el diagrama, es pot concloure que es poden classificar els casos d'ús tenint en compte quina és la funcionalitat principal que realitzen. En aquest sentit, establirem la següent classificació que ens servirà de base en els apartats que segueixen:

- Processos relacionats amb la càrrega d'operacions.
- Processos relacionats amb l'execució de les operacions de *refactoring*.
- Processos relacionats amb la salvaguarda de resultats i models intermedis.

Aquesta classificació no és estanca i s'utilitza només amb criteris d'explicació.

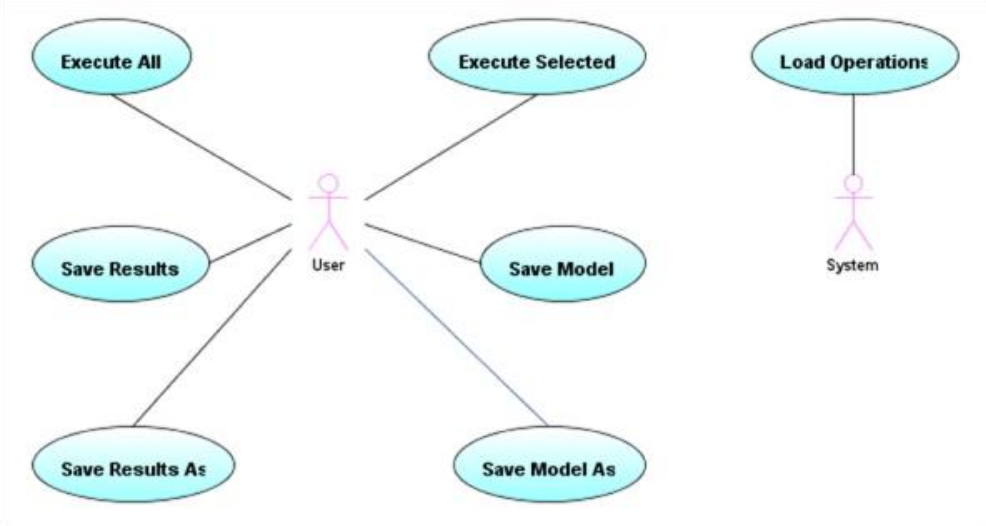


Fig. 18 - Casos d'ús

3.2.1.- Processos relacionats amb la càrrega d'operacions.

Per tal d'explicar quina ha estat la base del disseny d'aquests processos, se seguirà el fil d'un possible procés d'execució de **Protégé** un cop s'ha configurat adequadament la plataforma per a que carregui el *plugin*. La seqüència podria ser la següent:

- En iniciar-se l'aplicació, es carrega el *plugin*.
- En carregar-se un determinat projecte, s'executa el mètode `initialize()` del *plugin*, amb la qual cosa es tracta de realitzar les següents accions:
  - Carregar el model **OWL** associat al projecte.
  - Obtenir una instància de la classe controladora –si la instància no existeix, es crea.
  - Llegir el fitxer de configuració per tal d'obtenir les operacions de refacció que s'hi troben definides.
  - Crear la interfície d'usuari associada al *plugin* –objecte de la classe **JPanelPrincipal**.

El diagrama de la figura 20 mostra les classes implicades en aquestes operacions. Una altra classe que apareix en el diagrama és **OWLHelper**. Aquesta és una classe d'utilitat que disposa del mètode que proporciona la relació d'operacions: `getOperations()`.

Ja s'ha comentat anteriorment que aquestes es troben definides en un fitxer de configuració. Aquest fitxer, de nom **refactoring-operation.owl** presenta l'estructura que mostra la figura 19.

refactoring-operation.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.owl-pfc.com/2008/RefactoringOperation.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.owl-pfc.com/2008/RefactoringOperation.owl">

  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="RefactoringOperation"/>
  <owl:FunctionalProperty rdf:ID="className">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#RefactoringOperation"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="description">
    <rdfs:domain rdf:resource="#RefactoringOperation"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="operationName">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#RefactoringOperation"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
  <RefactoringOperation rdf:ID="ImplicitSubsets">
    <operationName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Implicit Subsets
    </operationName>
    <description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Class that implements the refactoring operation named ImplicitSubsets
    </description>
    <className rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        edu.uoc.pfc.semanticweb.refactoring.operation.ImplicitSubsetsRefactoring
    </className>
  </RefactoringOperation>
  <RefactoringOperation rdf:ID="GeneralizeRealization">
    <operationName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Generalize Realization
    </operationName>
    <description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Class that implements the refactoring operation named GeneralizeRealization
    </description>
    <className rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        edu.uoc.pfc.semanticweb.refactoring.operation.GeneralizeRealizationRefactoring
    </className>
  </RefactoringOperation>
</rdf:RDF>
```

Fig. 19 - Operacions. Fitxer de configuració.

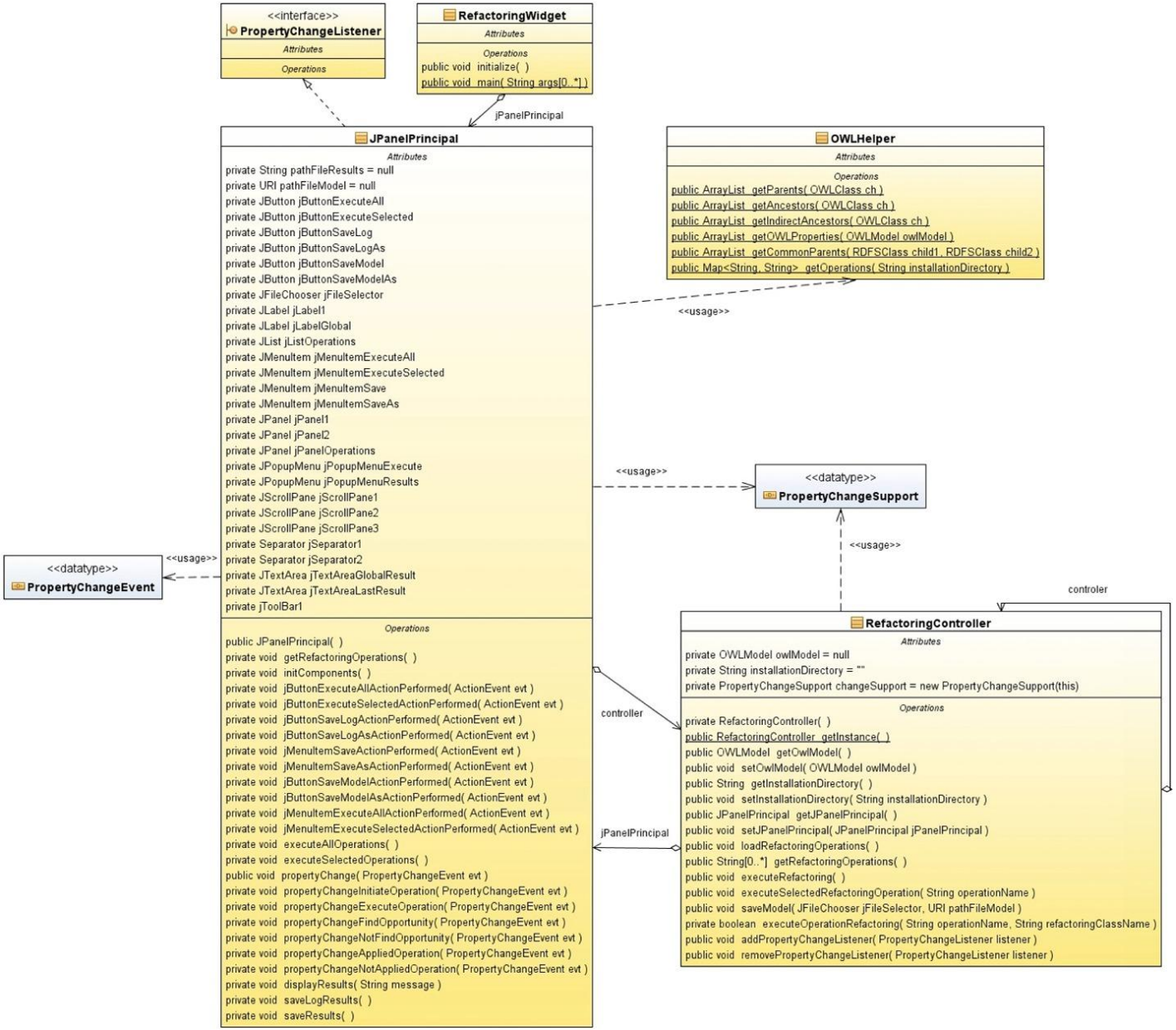


Fig. 20 - Diagrama de classes. Classe controladora i finestra principal.

Com es pot comprovar, **refactoring-operation.owl** no és res més que un fitxer **OWL** que conté una ontologia molt senzilla. Tenint en compte la poca dificultat, es va dubtar entre l'esmentat i l'**XML** a l'hora d'escollir el llenguatge a emprar. El fet de crear el *plugin* per a **Protégé** i les facilitats que ofereix la seva API per a obtenir la informació que conté una ontologia expressada en **OWL** varen fer decantar cap a l'ús d'aquest llenguatge.

Aquesta ontologia defineix els següents elements:

- Una classe, *RefactoringOperation*, per a modelar una operació de refacció.
- Tres propietats per a definir les característiques que interessen de les instàncies d'aquesta classe:
  - el nom de l'operació: `operationName`,
  - el nom complet de la classe que implementa l'operació: `className`,
  - i una descripció de l'operació: `description`.

Tal com reflecteix l'estat actual del fitxer de configuració, aquesta forma d'actuar permet anar afegint operacions del catàleg al *plugin* sense haver de modificar el *framework* creat. Tot i que es tornarà a aquest tema en acabar el següent subapartat, és clar que un dels passos a realitzar és la modificació del fitxer de configuració per a incloure les noves operacions a afegir al *plugin*.

3.2.2.- Execució de les operacions de refactoring.

Els diagrames **UML** de les figures 21 i 22 mostren les classes i interfícies més relacionades amb les operacions de refacció. Quan l'usuari escull executar algunes de les operacions implementades, el procés dissenyat realitza les següents accions:

- El sistema obté la llista d'operacions a executar i crida per a cadascuna el mètode `executeOperationRefactoring()` de la classe **RefactoringController**. Un dels paràmetres d'aquest mètode és el nom de la classe que implementa l'operació a executar.
- Amb aquest nom, la classe controladora crearà l'objecte adequat gràcies a les possibilitats d'instanciar objectes a partir del nom de la classe que ofereix la classe **Class** de **Java**.
- Un cop ha estat creat l'objecte, la classe controladora crida el mètode `execute()` que, com indica el seu nom, executa l'operació concreta de *refactoring*.

Aquesta forma d'actuar és possible gràcies al disseny de classes realitzat que es comenta tot seguit:

- Tal com mostra el primer dels diagrames, la classe abstracta **RefactoringOperation** representa el concepte genèric operació de refacció. Defineix el mètode abstracte `execute()` que hauran d'implementar les classes hereves encarregades d'especialitzar les distintes operacions de *refactoring* definides en el catàleg. En aquestes classes, el mètode abstracte esmentat centralitza les dues tasques a desenvolupar en qualsevol operació:
  - la detecció de l'oportunitat
  - i l'execució concreta.

  Com es pot observar, el mètode esmentat no conté cap paràmetre, amb la qual cosa és d'aplicació general per a qualsevol operació de refacció, fent possible el que s'ha indicat anteriorment: per a executar les distintes operacions, la classe controladora només ha de cridar aquest mètode per a cadascun dels objectes del tipus **RefactoringOperation** que va creant.
- **ImplicitSubsetsRefactoring** i **GeneralizeRealizationRefactoring** són les dues classes hereves corresponents a les dues operacions de *refactoring* implementades en aquest projecte. En el diagrama s'observa que són classes lleugeres que actuen com a façana, representant cara a l'exterior a les classes concretes que realitzen les operacions de *refactoring*, oferint un punt d'entrada genèric per a l'execució d'aquestes.
- Per a portar a la pràctica l'execució concreta, cadascuna de les classes hereves de **RefactoringOperation** utilitza els serveis d'una altra classe. Per a les operacions implementades, aquestes classes són **ImplicitSubsets** i **GeneralizeRealization**. Podem observar com, a més d'altres, aquestes classes disposen del mètode `execute()` amb atributs específics per a cada operació.
- En la figura 21 s'observa també com les oportunitats de *refactoring* estan representades per la interfície **RefactoringOpportunity**. Aquesta defineix el mètode que hauran d'especialitzar les classes que la implementin: `getElementsWhereOperationCanBeApplied()`. Cadascuna d'aquestes classes està associada a una operació concreta de refacció.

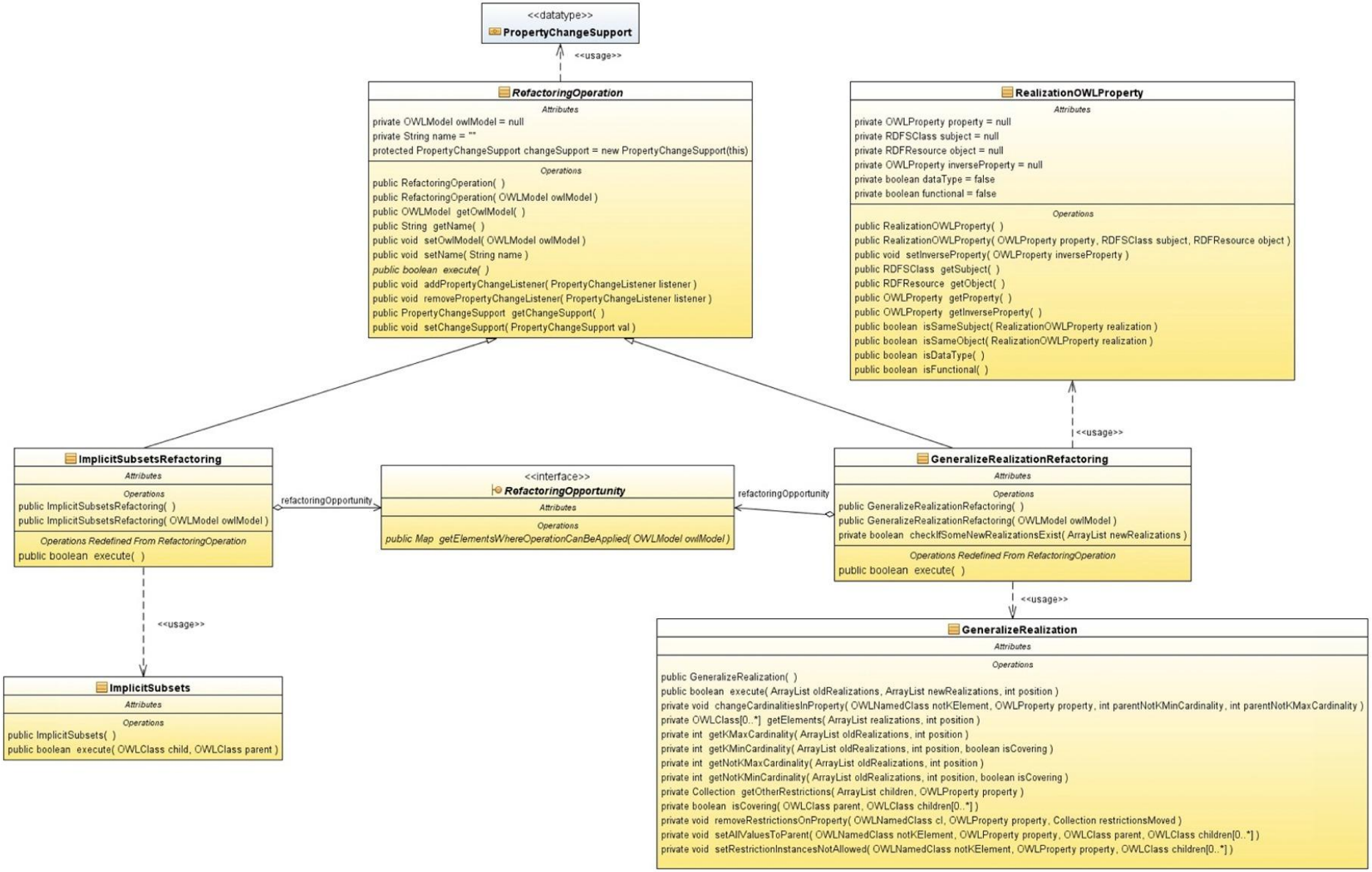


Fig. 21 - Diagrama de classes. Jerarquia Refactoring Operation.

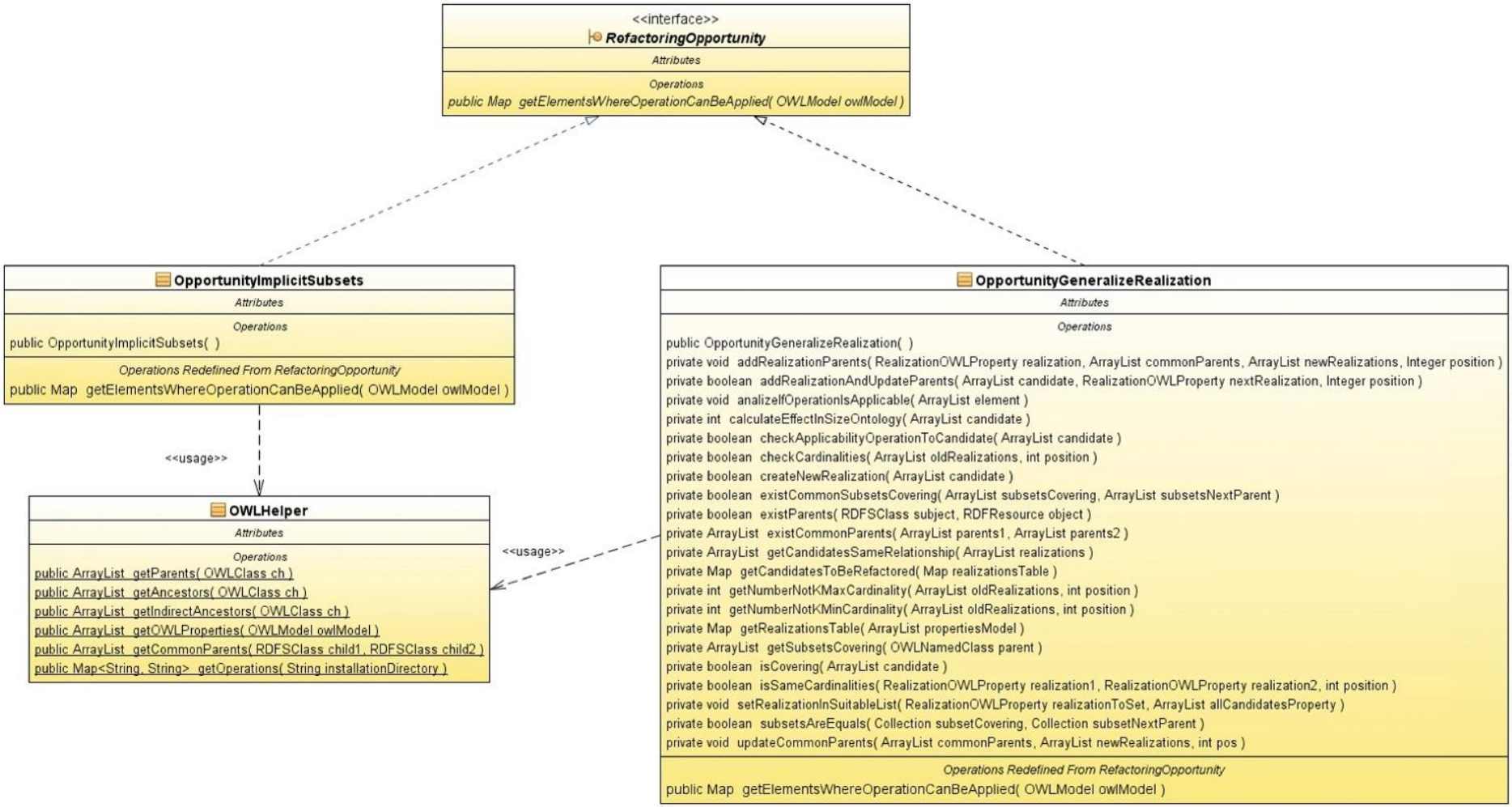


Fig. 22 - Diagrama de classes. Jerarquia Refactoring Opportunity.

Quan es produeix una crida al mètode `execute()` d'un objecte del tipus **RefactoringOperation** que representa una operació concreta, el sistema realitza essencialment les següents accions:

- Obté, gràcies als serveis d'un objecte del tipus **RefactoringOpportunity** associat a l'operació, la relació d'elements del model als que es pot aplicar l'operació.
- En cas que aquesta relació no sigui buida, executa per a cadascun d'ells l'operació, fent ús del mètode `execute()` de la classe auxiliar lligada a l'operació.

⇒ La figura 23 mostra amb més detall l'estructura de les classes utilitzades en l'execució de la primera de les operacions de refactoring implementades: *Implicit Subsets*.

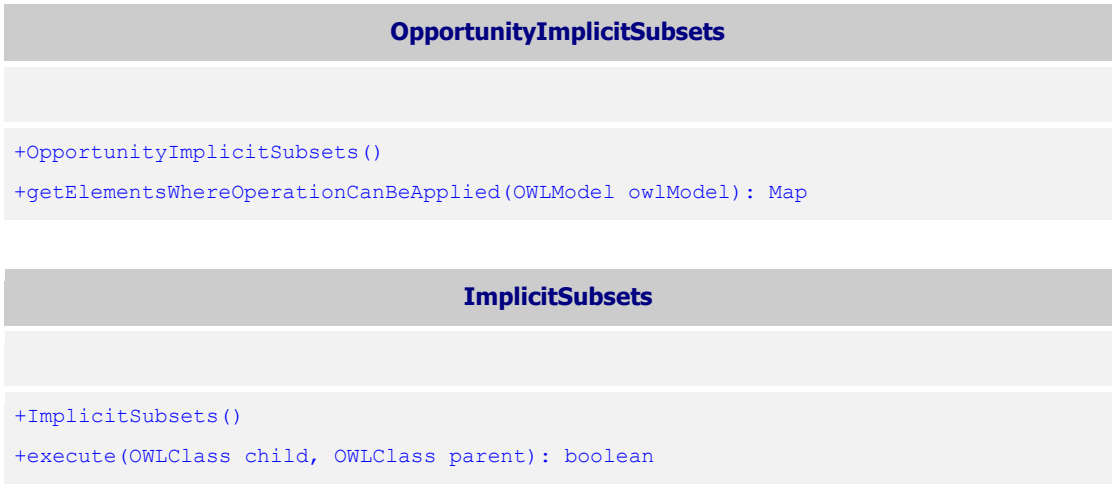


Fig. 23 – Classes OpportunityImplicitSubsets.java i ImplicitSubsets.java

La primera permet obtenir un objecte **Map** que conté els elements del model als què es pot aplicar l'operació de forma automàtica. Cada element del **Map** està format per una subclasse i per la relació de superclasses que són, alhora, ascendents directes i indirectes.

La segona classe permet, gràcies al mètode `execute()`, anular l'associació directa entre les dues classes que rep com a paràmetre.

⇒ En el cas de la segona de les operacions implementades, *Generalize Realization*, les classes implicades són, també, dues: **OpportunityGeneralizeRealization** i **GeneralizeRealization**. Ambdues usen una classe, **RealizationOWLProperty**, per a representar la realització d'una determinada propietat en una ontologia expressada en **OWL**. Aquesta classe, que es podria considerar que pertany a la capa del model, es mostra en detall en la figura 24. Tot seguit es comenten les raons que han ocasionat el seu disseny definitiu.

Tal com s'ha comentat en l'apartat 2.2.3 del present document, en **OWL** les relacions entre classes s'expressen a través de les propietats. Com ja hem vist, n'hi ha de varis tipus. Les que ens interessen respecte a l'operació a implementar són les *object properties* i les *data type properties*. Cal tenir present, també, que degut a l'estructura del llenguatge, en **OWL** totes les relacions són binàries, per la qual cosa l'element ***k*** a substituir –és a dir, l'element sobre el qual s'aplica la generalització– només pot ser el primer o el segon. Aquest fet simplifica, naturalment, el disseny i la implementació de l'algorisme.

Una altre qüestió a considerar és com s'expressen en **OWL** les relacions que en **UML** són associacions binàries bidireccionals entre classes. Aquesta qüestió també ha estat comentada anteriorment i s'ha indicat que una relació d'aquest tipus es transforma en **OWL** en dues propietats, de forma que una és la inversa de l'altra. A més, relacions d'aquesta mena només poden expressar-se mitjançant propietats del tipus *object property*.

La classe **RealizationOWLProperty** pretén donar resposta a totes aquestes qüestions. Es tracta de modelar la realització d'una determinada propietat a través d'una classe amb els següents atributs:

- `property`, per a representar la propietat.
- `subject` i `object`, per a representar els elements que es relacionen.
- `inverseProperty`, per a poder-hi relacionar la propietat inversa.
- `dataType`, element booleà que indica si ens trobem davant d'una propietat del tipus *object property* o *data type property*.
- `functional`, per a poder saber si la propietat és funcional.

RealizationOWLProperty
- property : OWLProperty - subject : RDFSClass - object : RDFResource - inverseProperty : OWLProperty - dataType : boolean - functional : boolean
+ RealizationOWLProperty() + RealizationOWLProperty(OWLProperty property,RDFSClass subject,RDFResource object) + setInverseProperty(OWLProperty inverseProperty) + getSubject() : RDFSClass + getObject() : RDFResource + getProperty() : OWLProperty + getInverseProperty() : OWLProperty + isSameSubject(RealizationOWLProperty realization) : boolean + isSameObject(RealizationOWLProperty realization) : boolean + isDataType() : boolean + isFunctional() : boolean

Fig. 24 - Classe RealizationOWLProperty

Aquesta classe ens permet disposar –o poder accedir fàcilment– a tota la informació necessària en el transcurs del procés. Així, per exemple, es poden carregar a l'hora les dades associades a una determinada propietat i, si existeix, a la seva inversa, amb la qual cosa, si cal executar l'operació, n'hi haurà prou en sincronitzar al final del procés fent ús de mètodes que l'API **OWL** de **Protégé** proporciona. També, saber que una propietat és funcional facilita el procés de càlcul de les cardinalitats ja que aquestes es manifesten, en aquest cas, de forma implícita.

Com s'ha indicat en paràgrafs anteriors, el mètode `execute()` d'un objecte **RefactoringOperation** utilitza els serveis d'un objecte **RefactoringOpportunity** per a obtenir la relació d'elements del model als que es pot aplicar l'operació fent una crida al mètode `getElementsWhereOperationCanBeApplied()`. En el cas de l'operació *Generalize Realization*, l'objecte **Map** obtingut presenta l'estructura que es comenta tot seguit.

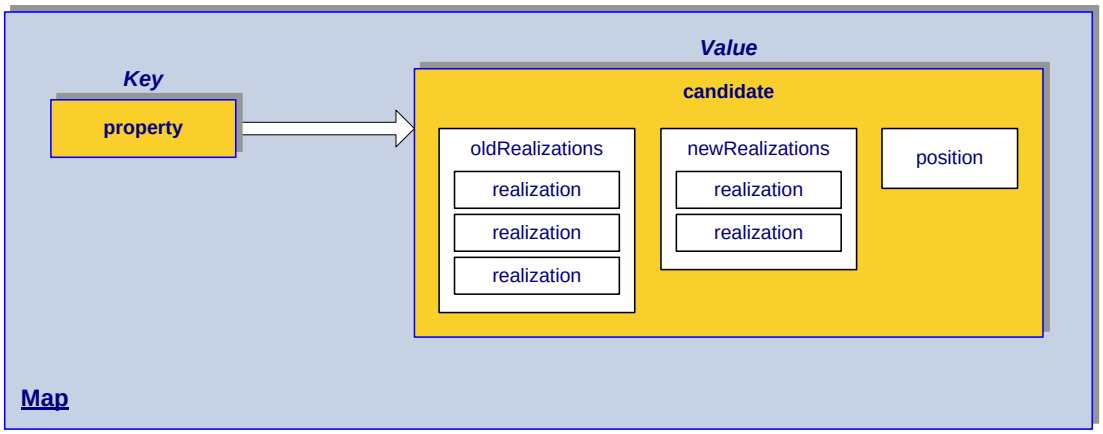


Fig. 25 - *Generalize Realization*. Estructura de dades d'un element candidat.

Com es pot observar en la figura 25, la clau del **Map** es la propietat a generalitzar, mentre que el valor és un objecte **ArrayList** que, a partir d'ara, anomenarem candidat –*candidate*– perquè conté la informació referent a les realitzacions de la propietat que són candidates a la refacció. Aquest **ArrayList**, alhora, està format per tres elements:

- un objecte **ArrayList** que conté el conjunt de realitzacions actuals de la propietat, és a dir, les realitzacions a generalitzar. L'anomenarem *oldRealizations*.

- Un altre **ArrayList** amb la llista de les noves realitzacions a crear. L'indicarem per *newRealizations*.
- Un objecte **Integer** que indica la posició que ocupa l'element a generalitzar.

Per tal de poder crear l'objecte **Map** que retornarà, `getElementsWhereOperationCanBeApplied()` obté en primer lloc les propietats del model definides pel creador de l'ontologia. Posteriorment, fa ús de tres mètodes privats de la classe **OpportunityGeneralizeRealization**:

OpportunityGeneralizeRealization

```
+OpportunityGeneralizeRealization()

+getElementsWhereOperationCanBeApplied(OWLModel owlModel): Map

-addRealizationParents(RealizationOWLProperty realization,
                       ArrayList commonParents, ArrayList newRealizations,
                       Integer position)

-addRealizationParents(RealizationOWLProperty realization,
                       ArrayList commonParents, ArrayList newRealizations,
                       Integer position)

-addRealizationAndUpdateParents(ArrayList candidate,
                                RealizationOWLProperty nextRealization,
                                Integer position) : boolean

-analizeIfOperationIsApplicable(ArrayList element)

-calculateEffectInSizeOntology(ArrayList candidate) : int

-checkApplicabilityOperationToCandidate(ArrayList candidate) : boolean

-checkCardinalities(ArrayList oldRealizations, int position) : boolean

-createNewRealization(ArrayList candidate) : boolean

-existCommonSubsetsCovering(ArrayList subsetsCovering,
                            ArrayList subsetsNextParent) : boolean

-existParents(RDFSClass subject, RDFResource object) : boolean

-existCommonParents(ArrayList parents1,
                    ArrayList parents2) : ArrayList

-getCandidatesSameRelationship(ArrayList realizations): ArrayList

-getCandidatesToBeRefactored(Map realizationsTable) : Map

-getNumberNotKMaxCardinality(ArrayList oldRealizations,
                             int position) : int

-getNumberNotKMinCardinality(ArrayList oldRealizations,
                             int position) : int

-getRealizationsTable(ArrayList propertiesModel) : Map

-getSubsetsCovering(OWLNamedClass parent) : ArrayList

-isCovering(ArrayList candidate) : boolean

-isSameCardinalities(RealizationOWLProperty realization1,
                     RealizationOWLProperty realization2, int position) : boolean
-setRealizationInSuitableList(RealizationOWLProperty realizationToSet,
                              ArrayList allCandidatesProperty)

-subsetsAreEquals(Collection subsetCovering,
                  Collection subsetNextParent) : boolean

-updateCommonParents(ArrayList commonParents,
                     ArrayList newRealizations, int pos)
```

Fig. 26 - Classe OpportunityGeneralizeRealization

- `getRealizationsTable()` proporciona un objecte **Map** amb totes les realitzacions de propietats contingudes en l'ontologia.
- `getCandidatesToBeRefactored()` analitza la informació que rep del mètode anterior comprovant si existeixen pares i germans per als distints components que formen part d'una

determinada realització, amb la qual cosa podrà crear i retornar un objecte **Map** amb la relació de candidats.

- Finalment, es fa una crida al mètode `analizeIfOperationIsApplicable()` per a cadascun dels elements del **Map** retornat pel mètode anterior. Com indica el seu nom, aquest mètode permet analitzar si l'operació és aplicable a les distintes realitzacions d'una determinada propietat –és a dir, a un candidat concret. Si comprova que no ho és, esborra el candidat del **Map**.

Com s'intueix del diagrama **UML** de la figura 26, aquests tres mètodes fan ús d'altres mètodes privats que són els que acaben implementant les funcionalitats específiques que permeten detectar que es compleixen les condicions[22] necessàries per a l'execució automàtica de l'operació de refactoring.

⇒ Un cop el mètode `execute()` de la classe **GeneralizeRealizationRefactoring** disposa dels elements del model als que es pot aplicar l'operació, inicia la fase final d'execució d'aquesta. Després de comprovar que no existeixi ja en l'ontologia la realització d'una propietat que es crearia fruit del procés, es crida per a cada element del **Map** obtingut anteriorment el mètode `execute()` de la classe **GeneralizeRealization**.

GeneralizeRealization

```
+GeneralizeRealization()

+execute(ArrayList oldRealizations,ArrayList newRealizations,int position): boolean

-changeCardinalitiesInProperty(OWLNamedClass notKElement,
                               OWLProperty property,
                               int parentNotKMinCardinality,
                               int parentNotKMaxCardinality)

-getElements(ArrayList realizations, int position) : ArrayList<OWLClass>

-getKMaxCardinality(ArrayList oldRealizations, int position) : int

-getKMinCardinality(ArrayList oldRealizations,int position,boolean isCovering) : int

-getNotKMaxCardinality(ArrayList oldRealizations, int position) : int

-getNotKMinCardinality(ArrayList oldRealizations,
                       int position,boolean isCovering) : int

-isCovering(OWLClass parent, ArrayList<OWLClass> children) : boolean

-removeRestrictionsOnProperty(OWLNamedClass cl, OWLProperty property)

-setAllValuesToParent(OWLNamedClass notKElement, OWLProperty property,
                      OWLClass parent, ArrayList<OWLClass> children)

-setRestrictionInstancesNotAllowed(OWLNamedClass notKElement, OWLProperty property,
                                   ArrayList<OWLClass> children)
```

Fig. 27 - Classe GeneralizeRealization

Com es pot comprovar, aquest mètode rep tres paràmetres:

- Una llista que conté les realitzacions a generalitzar.
- Una altra llista amb les realitzacions a crear.
- Un enter que indica quin és el participant de les antigues realitzacions que s'ha de substituir.

El procés d'execució fa ús de mètodes privats de la classe i passa per les següents etapes:

- En primer lloc obté, mitjançant el mètode `getElements()`, la relació de les classes a substituir i de les superclasses que les substituiran
- Tot seguit, s'obté la propietat, les seves característiques –si és *data type property* o funcional– i, cas d'existir, la seva inversa.
- A continuació, se substitueixen en el domini o en el rang de la propietat les subclasses per les superclasses i es calculen i es creen les noves cardinalitats.

[22] Aquestes condicions són les expressades en l'apartat 2.3.4.2 d'aquest document.

- A més, si la generalització no és total –*is not-covering*– s'afegeix, o es respecte si ja existeix, una restricció de tipus T1. Aquesta, com ja s'ha indicat anteriorment, s'expressa en **OWL** mitjançant una restricció de tipus *allValues*.
- Finalment, el procés acaba esborrant en les classes que han estat generalitzades les restriccions associades a les cardinalitats i sincronitzant la propietat implicada en la realització objecte del *refactoring* amb la seva inversa, cas que aquesta existeixi.

3.2.3.- Notificació a l'usuari dels missatges generats durant l'execució.

L'usuari interacciona amb el *plugin* a través d'una interfície gràfica. Com es comentarà en l'apartat següent, part d'aquesta interfície són dues zones que el sistema utilitza per a adreçar-li missatges relacionats amb el que succeeix en el transcurs del procés de refactoring. Aquests missatges són semblants als que s'indiquen tot seguit:

> Detected opportunity refactoring in Implicit Subsets...
>
> Operation Implicit Subsets: delete relationship between Analyst and Employee.
>
> The operation Implicit Subsets has been executed.

S'ha comentat en la introducció a aquest capítol que el mecanisme utilitzat per a comunicar aquests esdeveniments està basat en l'ús de la interfície **PropertyChangeListener** i de les classes **PropertyChangeEvent** i **PropertyChangeSupport**.

Qualsevol classe que necessita comunicar quelcom –per exemple, la classe controladora– disposa d'un atribut del tipus **PropertyChangeSupport**. Aquesta classe d'utilitat del paquet `java.beans` forma part del model de delegació d'esdeveniments de **Java** que permet a les classes publicar esdeveniments i, també, enregistrar-se com a subscriptores dels esdeveniments que es produeixen en d'altres. L'esmentada classe s'utilitza en l'aplicació sempre que es desitja notificar a d'altres –per exemple, la classe corresponent a la interfície d'usuari– determinats canvis.

Per altra banda, qualsevol classe –en aquest cas, la classe que representa la interfície d'usuari– que vol rebre noticia de determinats esdeveniments que es produeixen en una altra ha de realitzar dues accions: implementar la interfície **PropertyChangeListener** –i, naturalment, els mètodes que aquesta defineix– i enregistrar-se com a *listener* dels canvis.

Quan una classe vol notificar un esdeveniment fa una crida al mètode `firePropertyChange()` associat al seu objecte **PropertyChangeSupport**. Aquesta crida ocasiona que la notificació de l'esdeveniment es propagui als *listeners* enregistrats, els quals rebran un objecte **PropertyChangeEvent** que conté la informació de l'esdeveniment.

3.2.4.- Salvaguarda de resultats i models intermedis.

S'ha comentat anteriorment que el *plugin* ofereix la possibilitat d'emmagatzemar els resultats dels processos de *refactoring* –mitjançant *logs*– i els models intermedis obtinguts en el transcurs de l'execució.

La salvaguarda dels *logs* es gestiona des de la interfície d'usuari ja que la classe associada a la pestanya del *plugin* disposa de la informació a emmagatzemar en el disc. En canvi, els models intermedis, expressats com a ontologies **OWL**, es gestionen des de la classe controladora i s'executen fent una crida al mètode `saveModel()`. Aquest utilitza els serveis de classes de l'API **OWL** de **Protégé** per a crear el fitxer **.owl** que emmagatzemarà el model a conservar.

En ambdós casos, el *plugin* ofereix les dues possibilitats típiques de salvaguarda: Save i Save As.

3.2.5.- Classes i interfícies auxiliars.

A més de les classes i interfícies que s'ha estat tractant en els apartats anteriors, el disseny realitzat n'incorpora d'altres que es pot considerar efectuen tasques auxiliars. El diagrama **UML** de la pàgina següent les mostra.

Una d'elles –**OWLHelper**– ja ha estat vista anteriorment. És una classe d'utilitat que proporciona mètodes que permeten accedir a la informació continguda en les ontologies que s'analitzen. La interfície **Constants** permet definir una sèrie de constants pròpies del *plugin*, mentre que les altres tres classes especialitzen la classe **Exception** i representen les possibles excepcions que pot generar el *plugin*.

Fig. 23 - Diagrama UML de classes auxiliars.

3.3.- Disseny de la interfície gràfica.

Protégé presenta una interfície gràfica. Per tant, un dels requeriments del *plugin* és que l'usuari hi ha d'interaccionar mitjançant una interfície d'aquest tipus. Aquesta s'ha dissenyat com s'indica tot seguit.

3.3.1.- Finestra principal.

En escollir la pestanya corresponent al *plugin*, l'usuari visualitzarà un panell com el que es representa en la finestra de la figura 29. Com es pot comprovar, conté una barra d'eines amb les distintes opcions per a poder executar les operacions, emmagatzemar resultats o salvaguardar els models intermedis obtinguts al llarg del procés.

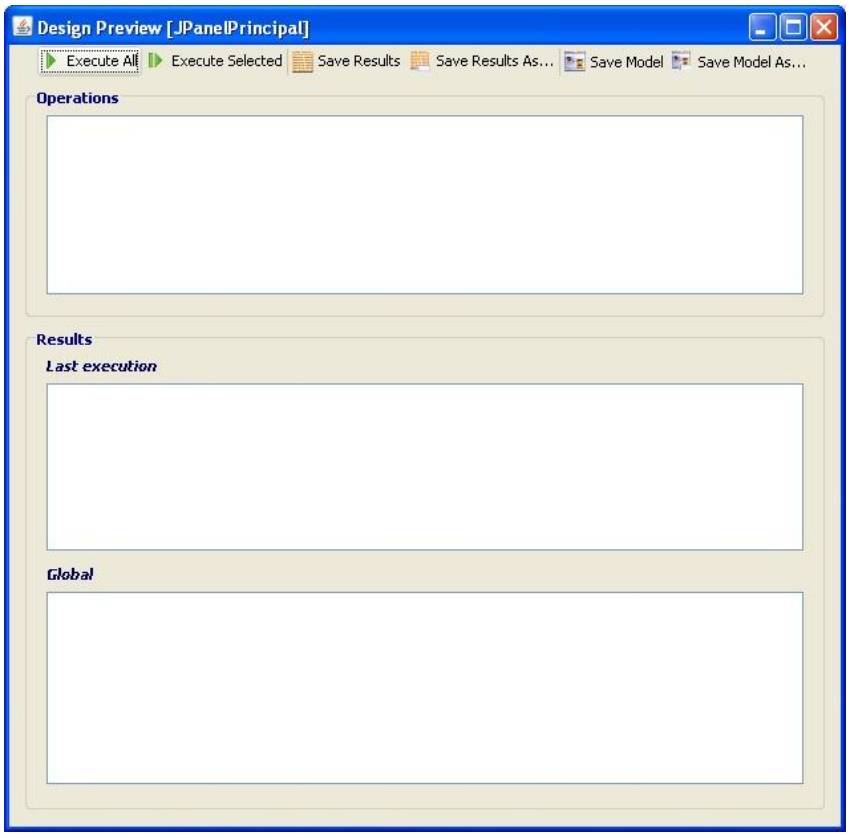


Fig. 29 - Disseny GUI. Finestra Principal.

Gairebé tota la finestra està ocupada per tres objectes: un **JList** i dos **JTextArea** del paquet **Swing** de **Java**. El primer contindrà, en temps d'execució, la llista d'operacions. Els altres dos components s'usaran per a notificar a l'usuari els resultats de l'execució de les operacions de *refactoring*.

3.3.2.- Altres components.

A més de la possibilitat que li ofereixen els botons de la barra d'eines, l'usuari disposarà també de la possibilitat d'ordenar accions mitjançant menús contextuals. N'hi haurà dos: un, amb les distintes possibilitats d'execució, estarà associat a la llista que conté les operacions, i un altre que permet emmagatzemar els resultats obtinguts estarà lligat a la zona de la finestra que mostra aquests.

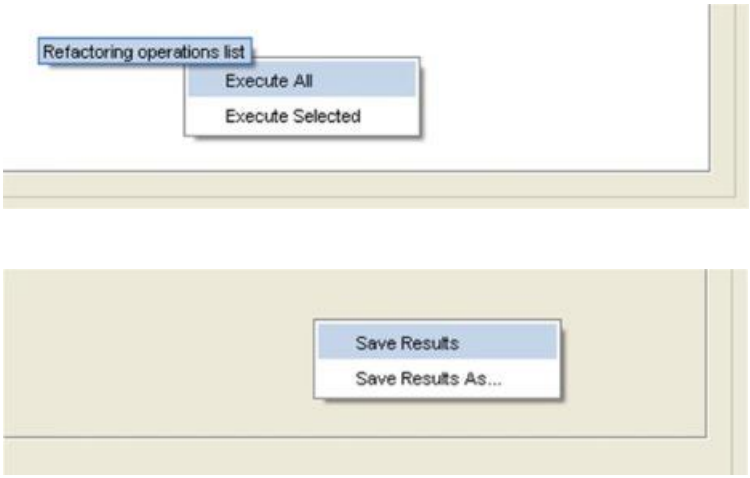


Fig. 30 - Disseny GUI. Menús contextuals.

Les opcions de salvaguarda disponibles en el *plugin* tenen associat un objecte **JFileChooser** que permet que l'usuari indiqui el fitxer que emmagatzemarà la informació a salvaguardar. Una imatge del disseny previ d'aquest objecte obtinguda a través de l'IDE utilitzat en el desenvolupament del projecte es mostra en la figura 31.

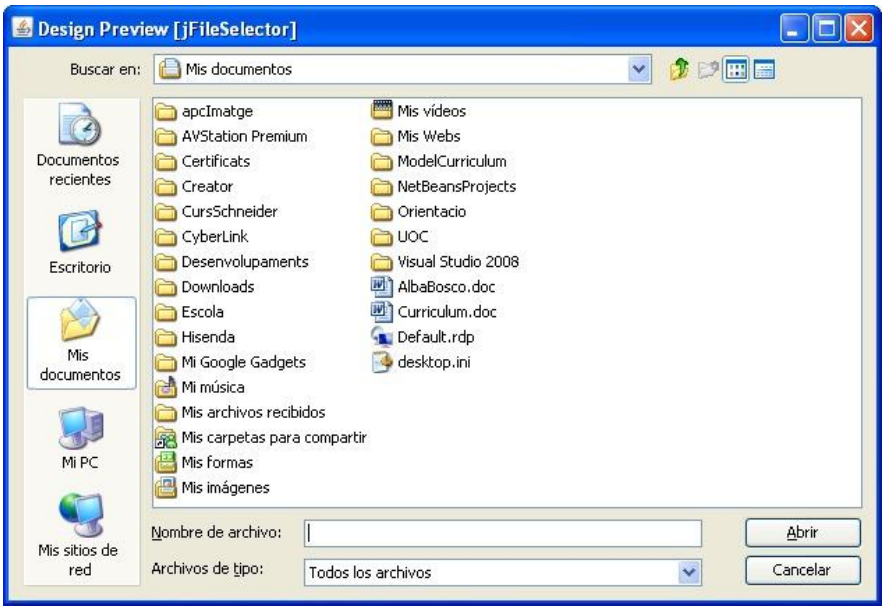


Fig. 31 - Disseny GUI. Finestra selecció de fitxers de salvaguarda.

4.- Aspectes de la implementació.

4.1.- Introducció.

Aquest projecte s'ha desenvolupat en el llenguatge **Java** emprant la plataforma **Protégé**. Les eines utilitzades en el desenvolupament han estat les següents:

- **Llenguatges i entorn de desenvolupament:**
  - **Java 2 SE** versió **1.6.0**.
  - **NetBeans** versió **6.1**[23].
  - **OWL 1.0**.
- **D'altres eines i/o llibreries:**
  - **Protégé 3.3.1 full** –darrera versió estable de la plataforma que inclou el nucli de **Protégé** i l'editor **OWL**.
  - **API** per a **Java** de **Protégé**.

Les classes de l'aplicació estan organitzades en un conjunt de paquets. Són els següents:

- `edu.uoc.pfc.semanticweb.refactoring.control` conté la classe controladora del *plugin*: **RefactoringController**.
- `edu.uoc.pfc.semanticweb.refactoring.widget` conté la classe **RefactoringWidget**.
- `edu.uoc.pfc.semanticweb.refactoring.model` amb la classe que representa les realitzacions de les propietats de les ontologies: **RealizationOWLProperty**.
- `edu.uoc.pfc.semanticweb.refactoring.operation` amb les classes associades a l'execució de les operacions de refacció.
- `edu.uoc.pfc.semanticweb.refactoring.opportunity` conté la interfície i classes relacionades amb la detecció de les oportunitats de *refactoring*.
- `edu.uoc.pfc.semanticweb.refactoring.form` conté la classe **JPanelPrincipal**.
- `edu.uoc.pfc.semanticweb.refactoring.utility` conté les classes i interfícies d'utilitat.
- `edu.uoc.pfc.semanticweb.refactoring.exception` amb les classes que representen les excepcions.

A més, existeix una carpeta que conté les icones utilitzades en la interfície d'usuari. Aquesta carpeta, emprant la nomenclatura pròpia dels paquets de Java, s'anomena `edu.uoc.pfc.semanticweb.refactoring.icons`.

En aquest capítol es descriuen –en algun cas, amb més detall– aquelles qüestions que es creuen destacables en la implementació de l'aplicació ***RefactoringPluginProtégé***, seguint un criteri fixat per la seva relació amb els següents temes:

- El *framework* referencial. La classe controladora.
- Els algorismes de refacció.
- La interfície d'usuari.

4.2.- El *framework* referencial. La classe controladora.

Una de les classes que podem considerar forma part del *framework* és **RefactoringWidget**. Com s'ha indicat en el capítol anterior, estén la classe **AbstractTabWidget** i redefineix el mètode `initialize()` d'aquesta. Les tasques principals que realitza aquest mètode són les següents:

- Obté una referència al model **OWL** corresponent al projecte actualment carregat en **Protégé**.
- Obté una instància de la classe controladora.

[23] En el pla de treball presentat a l'inici del semestre s'expressava un altre IDE: **Eclipse**. Es varen fer proves d'integració d'aquest IDE amb **Protégé** i, també, amb un altre entorn de treball: **NetBeans**. Com ha estat possible integrar aquest últim amb **Protégé**, finalment s'ha decidit emprar com a IDE el **NetBeans 6.1**. L'autor del **PFC** es troba més còmode treballant amb aquesta eina, sobretot quan es tracta de crear les interfícies gràfiques d'usuari, en què l'entorn gràfic que incorpora l'IDE facilita molt la feina.

- Fent ús del mètode estàtic `getInstallationDirectory()` de la classe **PluginUtilities** del paquet **`edu.stanford.smi.protege.plugin`**, obté el directori d'instal·lació del *plugin*, amb la qual cosa la classe controladora podrà llegir des del fitxer de configuració la relació d'operacions.
- Crea el panell de la interfície d'usuari, el situa en la pestanya associada al *plugin* i li passa una referència a la classe controladora.

Cal esmentar, també, que en aquesta classe s'ha definit el mètode `main()` que només té sentit a efectes de depuració.

⇒ La classe **RefactoringController** és la classe de control de l'aplicació. En aquest sentit, disposa, entre d'altres, dels atributs que li permeten conèixer el model a analitzar i el conjunt d'operacions de *refactoring* possibles. Aquest conjunt es guarda en un objecte **Map** en què la clau és el nom de l'operació i el valor el nom complet de la classe que l'implementa. Com a classe controladora, rep les sol·licituds d'altres classes per a:

- Carregar des del fitxer de configuració les operacions de *refactoring* –qüestió que acaba delegant en la classe d'utilitat **OWLHelper**.
- Obtenir la relació d'operacions carregades.
- Executar les operacions de *refactoring*, bé de forma global o bé aquella que ha estat seleccionada, a través, respectivament, dels mètodes `executeRefactoring()` i `executeSelectedRefactoringOperation()`.
- Emmagatzemar els models intermedis obtinguts en el transcurs del procés de refacció.

Els dos mètodes esmentats fan ús del mètode privat `executeOperationRefactoring()`. Aquest rep dos paràmetres, el nom de l'operació i el nom de la classe que la representa, i actua en la forma següent:

- A partir del nom de la classe, n'obté una instància fent ús del mètode estàtic `forName()` de la classe **Class** de **Java**.
- Li passa a l'objecte **RefactoringOperation** obtingut la referència del model **OWL** que conté l'ontologia.
- Crida al mètode `execute()` d'aquest objecte per a executar l'operació de *refactoring*. Prèviament, ha donat d'alta el panell principal com a *listener* per a que l'usuari pugui rebre informació del que succeeix en el transcurs de l'execució.

La classe controladora, com d'altres en l'aplicació, disposa d'un atribut del tipus **PropertyChangeSupport**. Com ja s'ha indicat, aquesta classe del paquet **`java.beans`** és una classe d'utilitat que forma part del model de delegació d'esdeveniments de **Java**.

En el cas de la classe controladora, es notifiquen els esdeveniments associats a l'inici de l'operació de *refactoring*, al resultat final i, cas de produir-se, l'excepció associada a una operació no existent. La notificació es fa mitjançant una crida al mètode `firePropertyChange()` associat a l'objecte **PropertyChangeSupport**. L'objecte que rep la notificació dels canvis és el panell principal del *plugin*.

⇒ S'ha indicat en el capítol anterior que també es consideraven part del *framework* la classe abstracta **RefactoringOperation** i la interfície **RefactoringOpportunity**. Aquesta última es limita a definir la signatura del mètode que hauran d'implementar les classes que la implementin.

Referent a la classe **RefactoringOperation**, no hi ha gairebé res més a destacar apart del que ja s'ha comentat en el capítol dedicat al disseny:

- Defineix atributs comuns a les classes que l'estendran: referència al model **OWL** a analitzar i nom de l'operació.
- Defineix els mètodes accessors per a aquests atributs.
- Defineix un atribut protegit del tipus **PropertyChangeSupport** i els mètodes per a afegir-hi i eliminar-hi *listeners*, de forma que les subclasses puguin notificar a aquests els canvis que es produeixen en el transcurs de l'execució de les operacions.

4.3.- Els algorismes de refacció.

Tenint present que el codi ve comentat i que cal mantenir el present document dins els límits marcats, tot seguit es comenten breument, sobretot en el cas de la segona operació, les línies principals que s'han seguit en la implementació dels algorismes de *refactoring*.

4.3.1.- Implicit Subsets.

Les classes associades a aquesta operació són **ImplicitSubsetsRefactoring**, **OpportunityImplicitSubsets** i **ImplicitSubsets**.

La primera disposa d'un atribut del tipus que representa la segona classe per a obtenir els elements del model als que l'operació es pot aplicar de forma automàtica. El mètode principal de **ImplicitSubsetsRefactoring** és `execute()`. Aquest mètode realitza principalment les següents accions:

- Crear l'objecte **ImplicitSubsets** que acabarà executant la microperació que esborrarà la generalització redundant.
- Obtenir, de l'objecte **OpportunityImplicitSubsets** associat, el **Map** que conté la relació dels elements del model als que cal aplicar l'operació.
- Si aquesta relació no és buida, extreure per a cada subclasse de l'objecte **Map** les superclasses que són ascendents indirectes.
- Demanar al mètode `execute()` de l'objecte **ImplicitSubsets** que executi el *refactoring*, passant-li com a paràmetres els objectes **OWLClass** que representen el fill i el pare.

El mètode principal d'**OpportunityImplicitSubsets** és `getElementsWhereOperationCanBeApplied()`. Aquest rep com a paràmetre el model **OWL** i executa les següents accions:

- Obté del model la llista de classes no anònimes definides per l'usuari fent una crida al mètode `getUserDefinedOWLNamedClasses()` de la classe que representa el model.
- Per a cadascuna de les classes obtingudes, cerca la relació d'ascendents directes i indirectes emprant el mètodes `getParents()` i `getIndirectAncestors()` de la classe d'utilitat **OWLHelper**.
- Reté en la llista de pares directes aquells que també ho són indirectes i, si la relació final no és buida, la incorpora al **Map**, agregant una entrada en què l'objecte fill és la clau i la relació d'ascendents que són directes i indirectes alhora és el valor.

La classe **ImplicitSubsets** és molt lleugera i el mètode principal –`execute()`– és molt senzill: es limita a comprovar si l'objecte **OWLClass** rebut com a fill ho és del que ha rebut com a pare. En cas afirmatiu, esborra la relació. Es considera que és responsabilitat de qui ha trucat al mètode haver comprovat anteriorment que la relació no és directe.

4.3.2.- Generalize Realization.

De forma anàloga a l'operació anterior, es disposa de tres classes associades a l'operació: **GeneralizeRealizationRefactoring**, **OpportunityGeneralizeRealization** i **GeneralizeRealization**.

Es podria asseverar que en el disseny de les dues últimes s'ha seguit una de les propostes que Martin Fowler realitzà sobre refacció del codi en el seu text de referència sobre *refactoring*, en concret, *Extract Method*: quan el contingut d'un mètode es preveu voluminós, es millor agrupar el codi lligat a tasques molt relacionades en mètodes especialitzats de dimensió més reduïda, la qual cosa fa el sistema més fàcil d'entendre i manipular. Aquest principi s'ha aplicat en aquest cas degut a la complexitat de l'algorisme.

La segona classe esmentada centralitza totes les tasques associades a la detecció de l'oportunitat de *refactoring*, mentre que la tercera s'encarrega de les execucions concretes de l'operació. En els dos casos, cal tenir presents molts aspectes i s'ha pensat en què dos mètodes públics – `getElementsWhereOperationCanBeApplied()` en el primer cas i `execute()` en el segon– centralitzessin la tasca principal i deleguessin en mètodes privats les tasques concretes.

⇒ Començarem comentant les principals accions del mètode `execute()` de la primera de les classes indicades anteriorment, **GeneralizeRealizationRefactoring**. Són les següents:

- Crear l'objecte **GeneralizeRealization** que acabarà executant la operació per a una parella concreta de velles i noves realitzacions de la propietat **OWL** a generalitzar.
- Obtenir, de l'objecte **OpportunityGeneralizeRealization** associat, el **Map** que conté la relació de realitzacions a les que es pot aplicar l'operació de forma automàtica.
- Si el **Map** no és buit, s'extreuen per a cada propietat les llistes de candidats.
- Com s'ha comentat en el capítol anterior, cada objecte candidat està format per dues llistes –la de les velles realitzacions de la propietat i la de les noves a crear– i un enter que indica la posició de l'element a generalitzar.

- Després de comprovar que les noves realitzacions no existeixin –a través del mètode `checkIfSomeNewRealizationsExist()`–, es fa una crida al mètode `execute()` de l'objecte **GeneralizeRealization** per a que executi el *refactoring*.

⇒ Deixant de banda el constructor, l'únic mètode públic de la classe **OpportunityGeneralizeRealization** és `getElementsWhereOperationCanBeApplied()`. Tenint en compte la complexitat de la classe, el mètode és relativament senzill, rep com a paràmetre el model **OWL**, i executa les següents accions:

- Obté la llista de propietats incloses en el model fent ús del mètode `getOWLProperties()` de la classe d'utilitat **OWLHelper**.
- Fa una crida a dos mètodes privats de la pròpia classe:
  - `getRealizationsTable()`. Aquest mètode, fent ús dels mètodes de l'API **OWL** de **Protégé**, va creant els objectes **RealizationOWLProperty** que representen les realitzacions de les distintes propietats contingudes en el model, i acaba lliurant un objecte **Map** que les conté.
  - `getCandidatesToBeRefactored()`, forma a partir del **Map** anterior un nou **Map** que conté una primera relació d'objectes als que es podria aplicar l'operació.

    Per a fer-ho, utilitza el mètode `getCandidatesSameRelationship()` que analitza l'existència de pares i germans i va creant la taula que conté les velles –les obtingudes anteriorment– i les possibles noves realitzacions associades a una determinada propietat.
- Un cop es disposa de la relació de candidats agrupats per propietats, s'analitza a partir de la propietat si l'operació es pot aplicar de forma automàtica. L'encarregat de l'anàlisi és el mètode `analizeIfOperationIsApplicable()`. Aquest mètode rep una llista que conté tots els objectes candidat associats a una determinada propietat i delega en un altre mètode, `checkApplicabilityOperationToCandidate()`, l'anàlisi de cadascun d'ells.

  Aquest darrer comprova si es compleixen les condicions especificades en l'algorisme:
  - Analitza si el nombre de fills és més gran que el de pares.
  - Comprova si la relació és completa, la qual cosa fa amb l'ajut del mètode `isCovering()`.
  - Mira, fent és de `checkCardinalities()`, si tots els fills tenen la mateixa cardinalitat en la posició a generalitzar.
  - I, finalment, comprova si es redueix la grandària de l'ontologia mitjançant el mètode `calculateEffectInSizeOntology()`.

  Si les condicions no es compleixen, s'esborra l'element de la llista de candidats.
- Un cop aplicat el mètode `analizeIfOperationIsApplicable()`, si la llista de candidats associada a una determinada propietat és buida, s'esborra aquesta entrada en el **Map**, de forma que `getElementsWhereOperationCanBeApplied()` pugui retornar només aquells elements als que efectivament l'operació sigui aplicable.

⇒ Finalment, comentarem la forma en què s'executa l'operació. Aquesta corre a càrrec del mètode `execute()` de la classe **GeneralizeRealization**. Aquest mètode rep com a paràmetres dues llistes, una amb les velles realitzacions de la propietat i l'altra amb les noves, i un enter que indica la posició que ocupa el participant que cal moure. L'operació s'executa mitjançant els passos següents:

- S'obtenen, fent ús del mètode privat `getElements()`, les relacions de pares i fills.
- S'obté la propietat i, si existeix, la seva inversa. S'analitza, també, si la propietat és del tipus *data type property* i si és funcional.
- Tot seguit s'inicia el procés pròpiament dit. En primer lloc, en funció de la posició que ocupa el participant a moure, se situen els pares en lloc dels fills en el domini o en el rang de la propietat, la qual cosa es fa emprant mètodes propis de l'API **OWL** de **Protégé** .
- A continuació, per a cadascun dels pares inclosos en la llista de noves realitzacions:
  - Es comprova si la relació amb els fills és completa mitjançant el mètode privat `isCovering()`.
  - Es calculen les cardinalitats, emprant, també, mètodes privats definits en la classe:

        - `getKMinCardinality()`, que proporciona el valor de la mínima cardinalitat que cal assignar al pare que ocuparà el lloc del participant ***k***.
        - `getKMaxCardinality()`, que fa el mateix per a la cardinalitat màxima.
        - `getNotKMinCardinality()`, permet calcular la cardinalitat mínima a assignar al participant del lloc ***not-k***.
        - `getNotKMaxCardinality()`, permet fer el mateix càlcul per a la cardinalitat màxima.
    - Fent ús de mètodes de l'API **OWL** de **Protégé** es creen les restriccions associades a l'element ***k*** i, mitjançant el mètode `changeCardinalitiesInProperty()`, s'actualitzen les restriccions de cardinalitat de l'element ***not-k***.
    - Tot seguit, es realitzen dues accions:
        - si la generalització és completa, es canvia la restricció del tipus **`owl:allValues`** en la propietat indicant que tots els valors en la posició ***k*** han del tipus del pare, cosa que es fa a través del mètode `setAllValuesToParent()`.
        - Si no és completa, s'afegeix la restricció de tipus T1. Aquesta tasca l'efectua el mètode `setRestrictionInstancesNotAllowed()`.
- Finalment, s'esborren les restriccions no necessàries encara relacionades amb els fills participants que han estat moguts i, en cas d'existir la propietat inversa, se sincronitza amb la que ha estat retocada per tal de deixar el model consistent.

  La primera feina la porta a la pràctica el mètode `removeRestrictionsOnProperty()`, mentre que la segona l'executa un dels mètodes propis de l'API **OWL** de **Protégé**, `synchronizeDomainAndRangeOfInverse()`.

És clar que tan d'aquesta darrera classe com de l'anterior han quedat aspectes a tractar en el que es refereix a la implementació. Ja s'ha esmentat a l'inici del subapartat que el document tenia una extensió limitada i que el codi font venia comentat, la qual cosa s'espera cobreixi les mancances que, sens dubte, les explicacions anteriors tenen.

4.4.- La interfície d'usuari.

Les classes de la interfície d'usuari es troben en el paquet **`edu.uoc.pfc.semanticweb.refactoring.form`**. Es disposa d'una única classe que correspon al panell que s'encabeix en la pestanya del plugin.

En la implementació s'han aprofitat les prestacions de disseny gràfic que ofereix l'IDE emprat –**NetBeans**–, i es podrien destacar els següents aspectes:

- La única classe del paquet és peça bàsica en el que, en el model MVC, anomenem la vista. Qualsevol canvi en el model –en aquest cas, la notificació del que esdevé en el rerefons– cal reflectir-lo en la vista.
- La forma de fer-ho ja s'ha emprat en d'altres parts de l'aplicació: el panell associat a la GUI es constitueix com a *listener* dels canvis que es produeixen en els objectes que interessen –classe controladora i classes associades a les operacions de *refactoring*–; implementa la interfície **PropertyChangeListener** i defineix el mètode `propertyChange()`, amb la qual cosa obté informació dels canvis a través de l'objecte **PropertyChangeEvent** que rep en executar el mètode.
- La classe disposa dels menús –alguns contextuals– que permeten a l'usuari escollir l'acció que vol que realitzi l'aplicació.

5.- Manuals d'usuari.

5.1.- Instal·lació i execució del *plugin*.

Com s'ha esmentat en l'apartat 1.5, el producte de programari resultat del **PFC** es troba en el fitxer **RefactoringPluginProtege.zip**. En aquest arxiu, es troba la carpeta **edu.uoc.pfc.semanticweb.refactoring** amb el següent contingut:

- El fitxer **RefactoringPluginProtege.jar** que conté les classes compilades conjuntament amb l'arxiu **MANIFEST.MF**.
- L'arxiu de configuració **refactoring-operation.owl** que defineix les operacions de refactoring possibles.
- El fitxer **plugin.properties** que declara el nom i les dependències del *plugin*.

Per a integrar el plugin desenvolupat amb la plataforma **Protégé** n'hi ha prou en realitzar les següents accions:

- Copiar el contingut d'aquesta carpeta en el directori *plugins* del directori d'instal·lació de **Protégé**.
- Executar **Protégé** i activar el *plugin* escollint l'opció Configure... dins el menú Project i, tal com mostra la figura 32, seleccionar el *tab* corresponent al *plugin* desenvolupat.

  Després de validar aquesta acció, la pestanya corresponent al plugin –Ontology Refactoring– apareixerà en l'entorn de treball de **Protegé**.

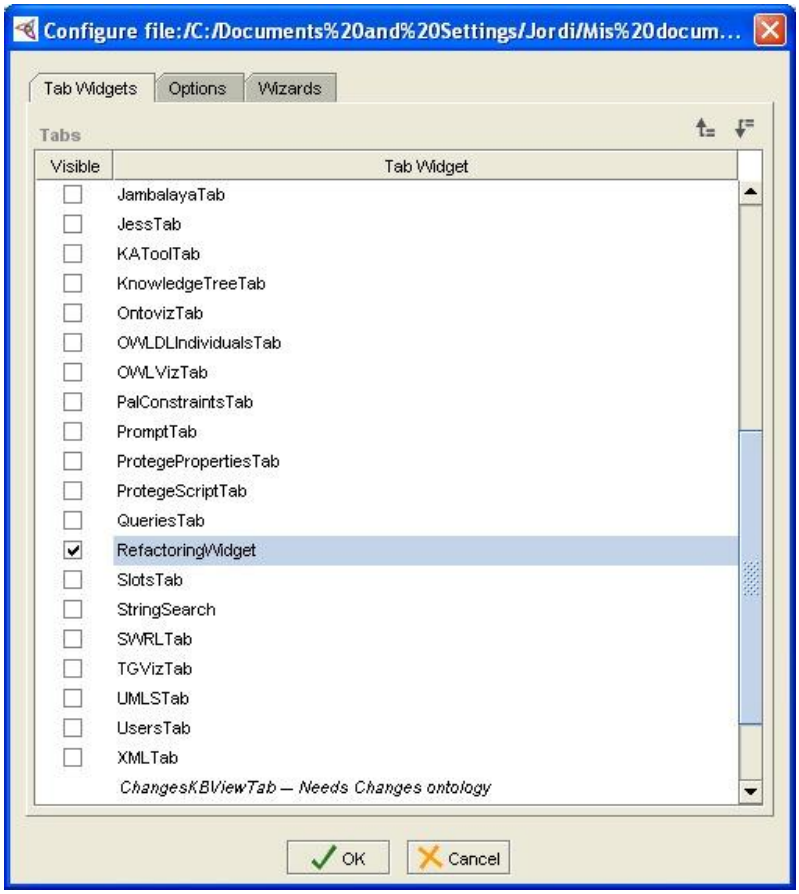


Fig. 32 - Integració del plugin en Protégé.

La figura següent mostra aquest fet.

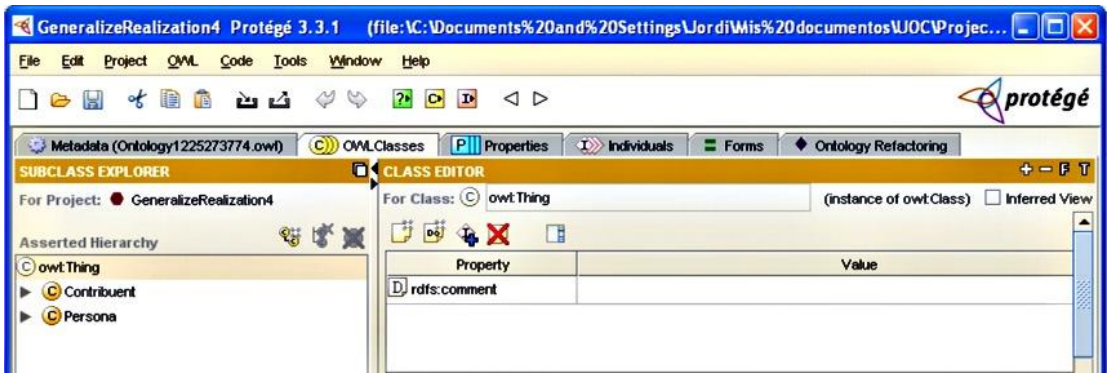


Fig. 33 - Resultat d'instal·lació del *plugin*

Per tal d'executar el *plugin* només cal activar la pestanya i fer ús de les opcions de menú que ofereix la barra d'eines situada dins el panell del *plugin*.

5.1.1.- Integració i execució des d'un entorn de desenvolupament.

Hom pot desitjar executar el *plugin* des d'un entorn de desenvolupament com **NetBeans** o **Eclipse**, bé perquè s'han afegit noves operacions de *refactoring*, o bé amb l'objectiu de depurar el codi o observar el que succeeix en el rerefons. Els passos a seguir en aquest cas serien els següents –prenent com a exemple el primer dels IDE esmentats–:

- Si no es disposa d'un projecte, crear-lo i importar les fonts del *plugin* –es troben disponibles dins la carpeta **src**.
- Incorporar al projecte les llibreries bàsiques de **Protégé** que estan incloses en el directori d'instal·lació de la plataforma i, a més, les que es troben en el directori del *plugin* **OWL** situat en la carpeta **plugins** del directori d'instal·lació de **Protégé**.

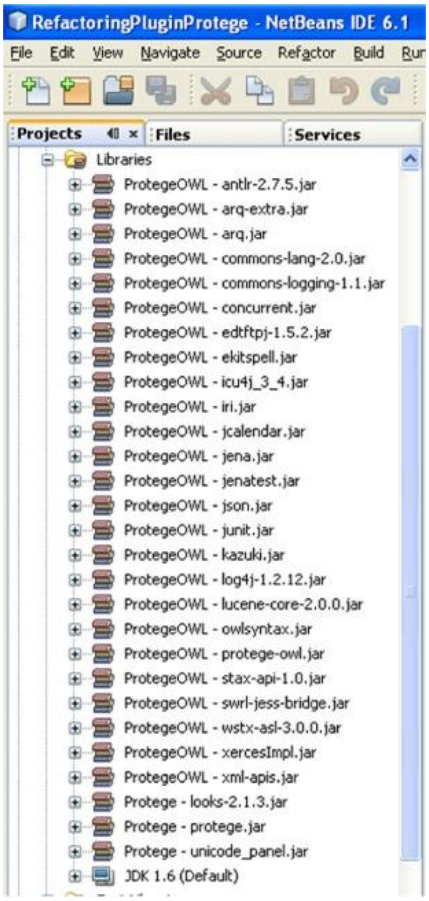


Fig. 34 - Integració amb un IDE. Llibreries de Protégé.

- En el procés d'execució, l'IDE necessita conèixer quin és el directori d'instal·lació de **Protégé**. Aquesta indicació se li proporciona, tal com mostra la següent figura, a través de l'opció *Dprotege.dir*.

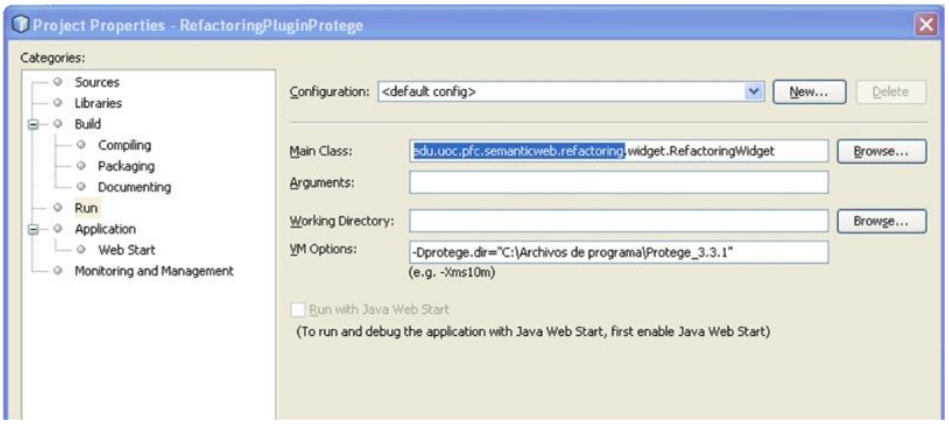


Fig. 35 - Integració amb un IDE. Indicació del directori d'instal·lació de Protégé.

5.2.- Manual d'ús.

5.2.1.- Configuració del plugin.

Ja s'ha esmentat anteriorment que el *plugin* es pot configurar en el sentit que es poden afegir noves operacions a les ja implementades sense haver de modificar el *framework* creat. Per a fer-ho, cal actuar en la forma següent:

- Obtenir el codi font del *plugin*.
- Implementar les classes associades a les operacions que es pensen incorporar al *plugin*. En aquest sentit, s'ha de tenir present que cal:
  - Crear una classe lligada a la nova operació que especialitzi la classe abstracta **RefactoringOperation**, així com la classe auxiliar encarregada de l'operació concreta.
  - Crear una classe, també relacionada amb la nova operació, que implementi la interfície **RefactoringOpportunity**.
  - Si és necessari, estendre la classe d'utilitat **OWLHelper**.
- Modificar el fitxer de configuració descrit en l'apartat 3.2.1 per a incloure la nova operació.

Aquest fitxer es pot modificar des del mateix **Protégé**. N'hi ha prou en obrir-lo des de l'eina, escollir la pestanya Individuals i crear una instància nova de la classe **OWL** *RefactoringOperation* indicant el valor de les distintes propietats. La primera figura –37– de la pàgina següent mostra el procés.

5.2.2.- Entorn de treball. Panell principal.

L'entorn de treball del *plugin* es mostra en la figura 38. La barra d'eines, que s'observa en detall en la figura següent, conté les següents opcions: Execute All, Execute Selected, Save Results, Save Results As..., Save Model i Save Model As....

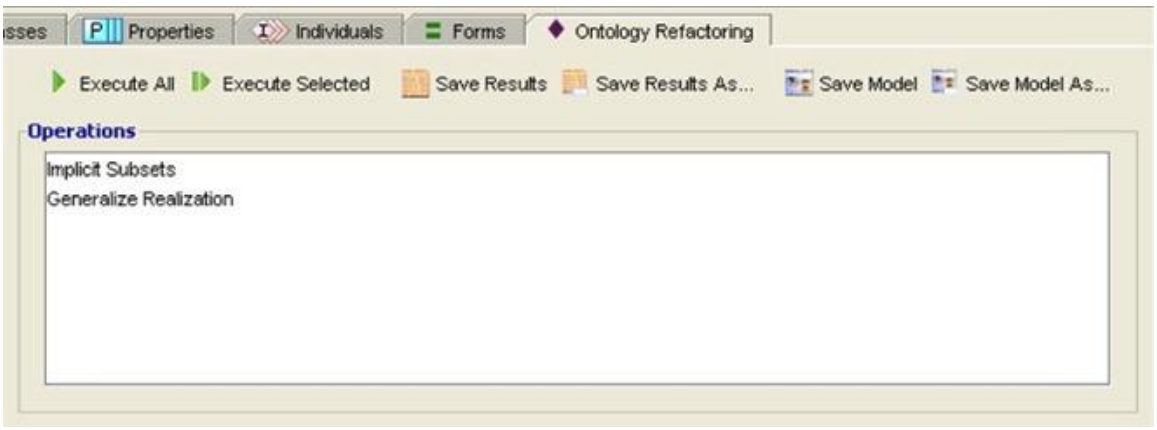


Fig. 36 - Opcions barra d'eines.

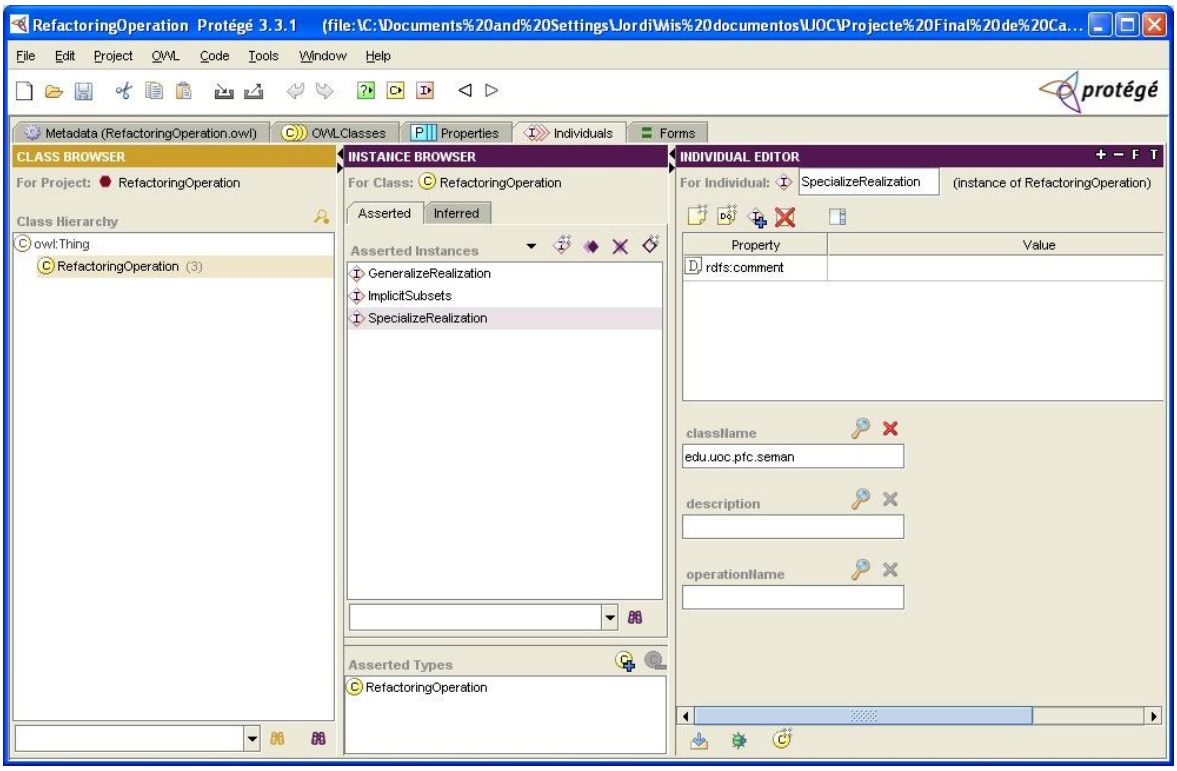


Fig. 37 - Incorporació de noves operacions emprant Protégé.

Aquestes opcions permeten executar totes o un conjunt d'operacions seleccionades, emmagatzemar els *logs* resultat de l'execució i enregistrar en el disc models intermedis –com a fitxers **.owl**– que es puguin anar generant en el transcurs del procés.

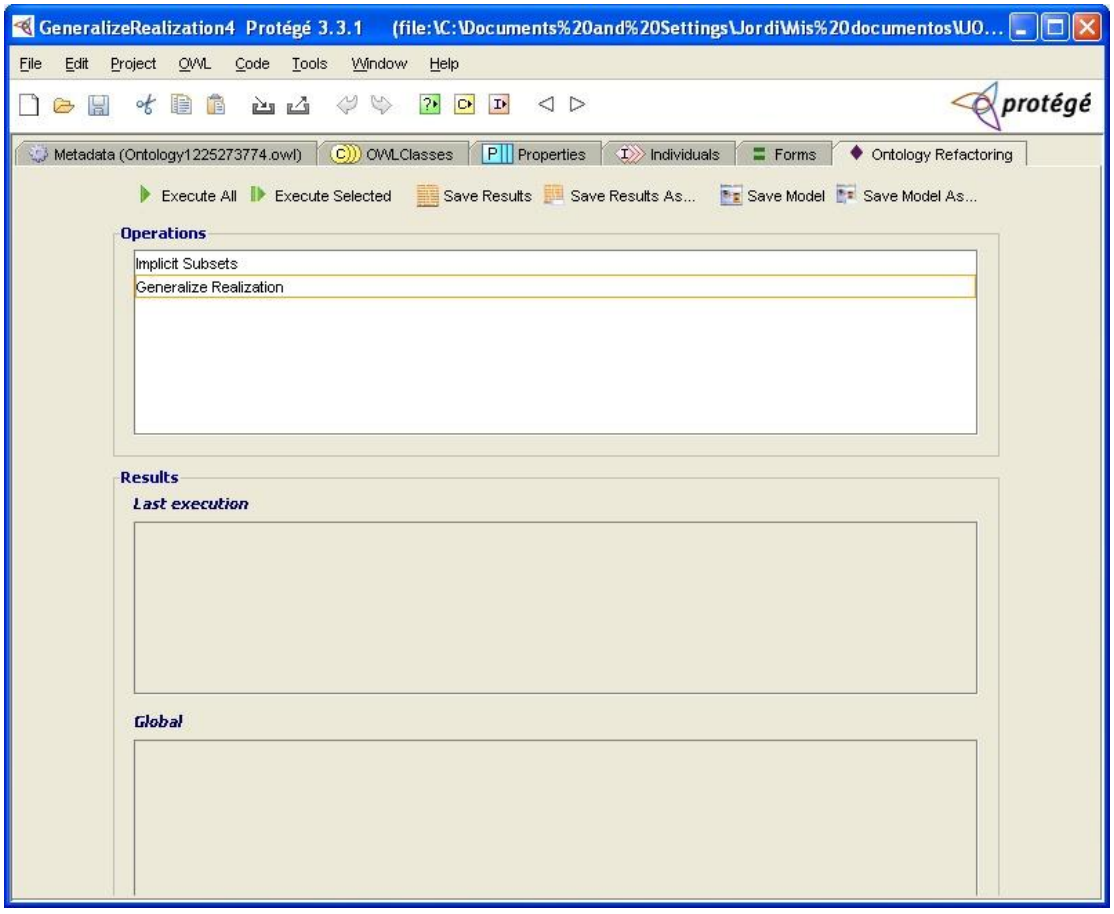


Fig. 38 - Entorn de treball del plugin.

5.2.3.- Execució de les operacions.

Per a executar les operacions n'hi ha prou en escollir el mode d'execució –total o selectiu–, bé a través del botó de la barra d'eines o mitjançant el menú contextual associat a la llista d'operacions. La figura mostra l'inici del procés i l'estat final de la pantalla després de l'execució.

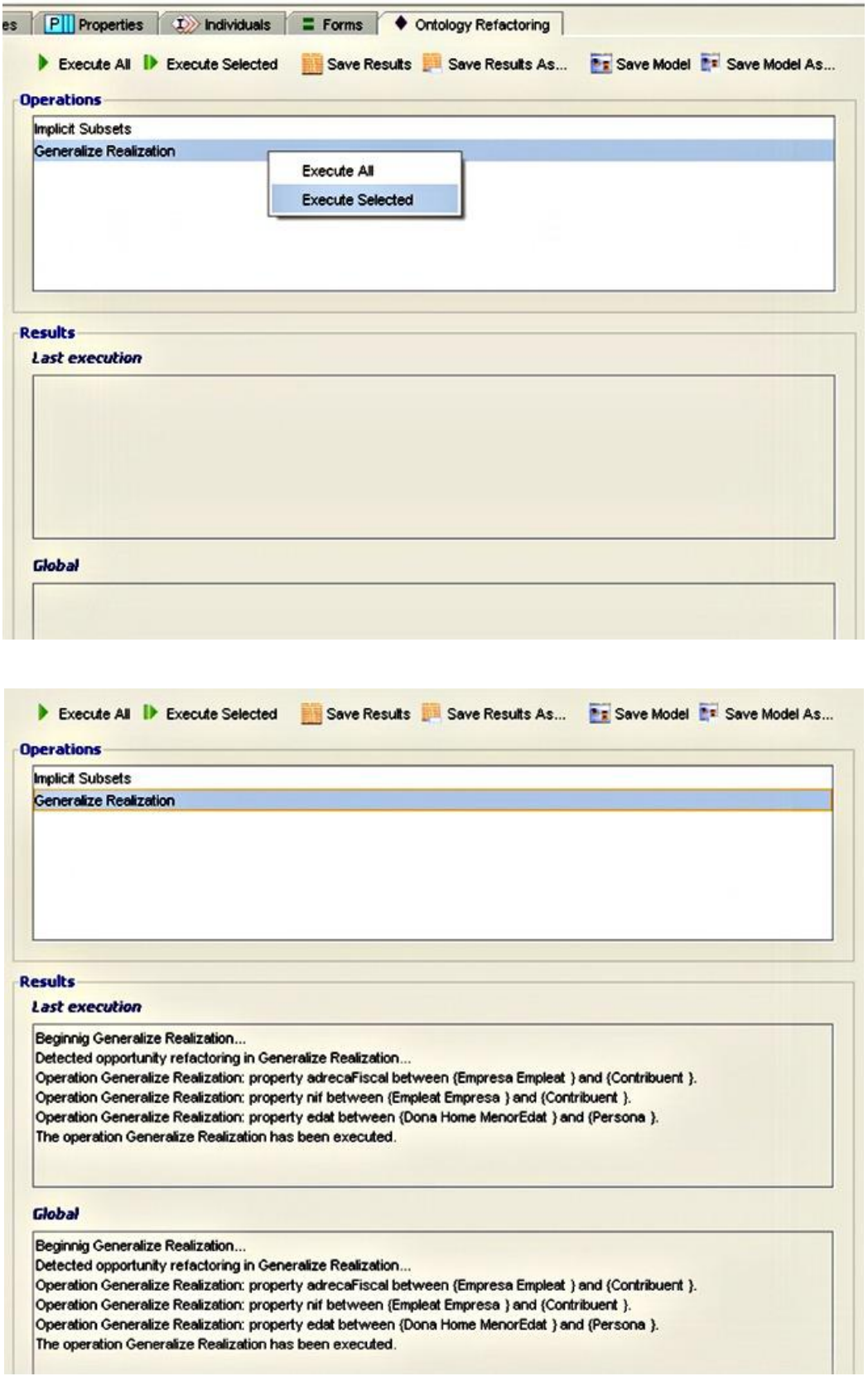


Fig. 39 - Execució d'operacions.

5.2.4.- Emmagatzemar resultats.

Si es volen emmagatzemar els *logs* enregistrats al llarg del procés s'escull l'opció Save Results o Save Results As... des de la barra d'eines o des del menú contextual associat a les àrees de resultats.

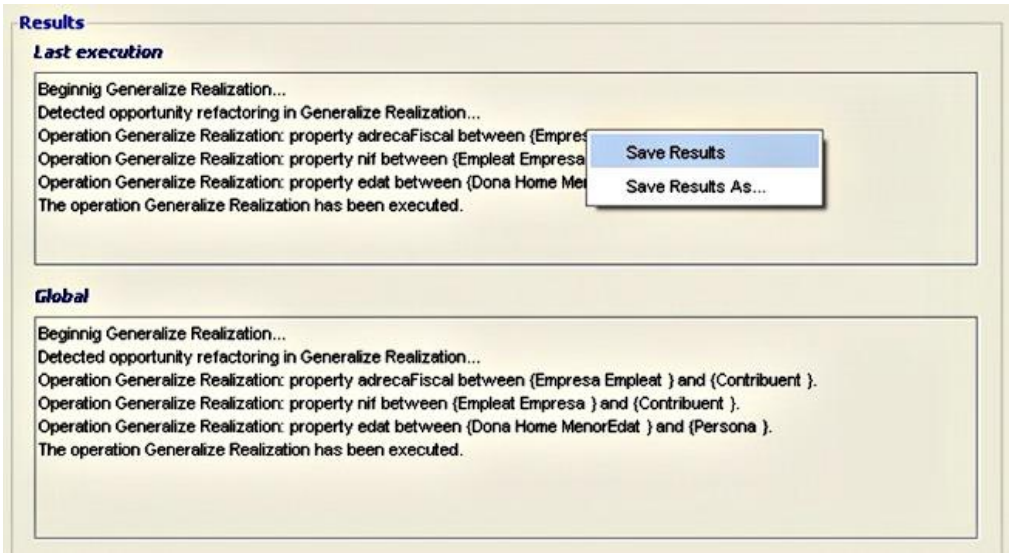


Fig. 40 - Inici emmagatzemar *logs*.

Quan s'executa la segona de les opcions o bé la primera per primera vegada, el sistema demana el nom i el destí del fitxer que guardarà els resultats a través de la finestra de la figura adjunta. Només cal indicar la ruta i el nom del fitxer i s'emmagatzemaran les dades en el disc.

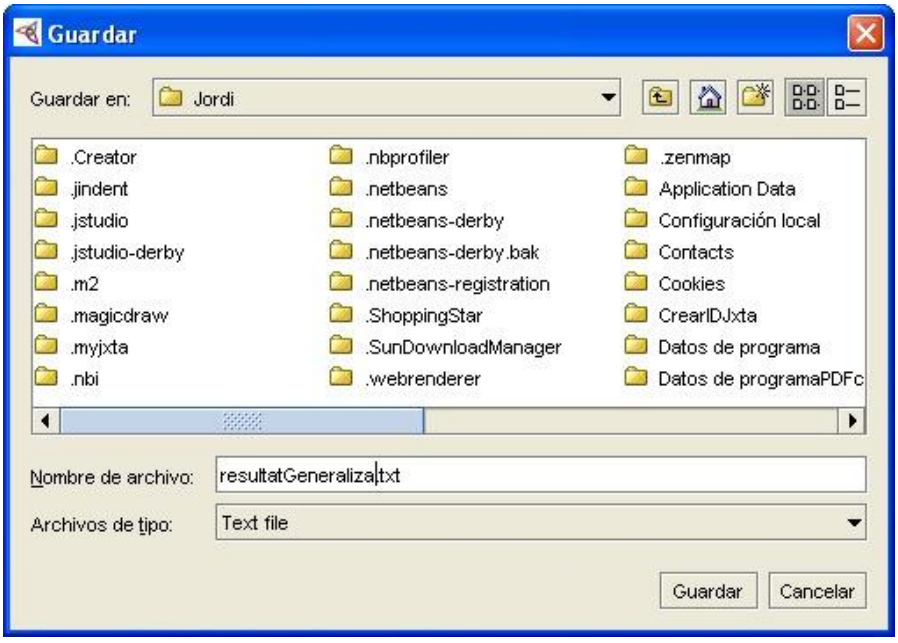


Fig. 41 - Save Results As...

5.2.5.- Guardar models intermedis.

Aquest procés és gairebé idèntic a l'anterior. S'inicia escollint alguna de les opcions que ofereix la barra d'eines, Save Model o Save Model As.... Si cal –perquè s'ha elegit la segona opció o perquè s'executa per primer cop la primera–, el *plugin* demanarà les dades del fitxer i crearà o substituirà aquest, mitjançant el quadre de diàleg que es pot observar en la primera figura de la pàgina següent.

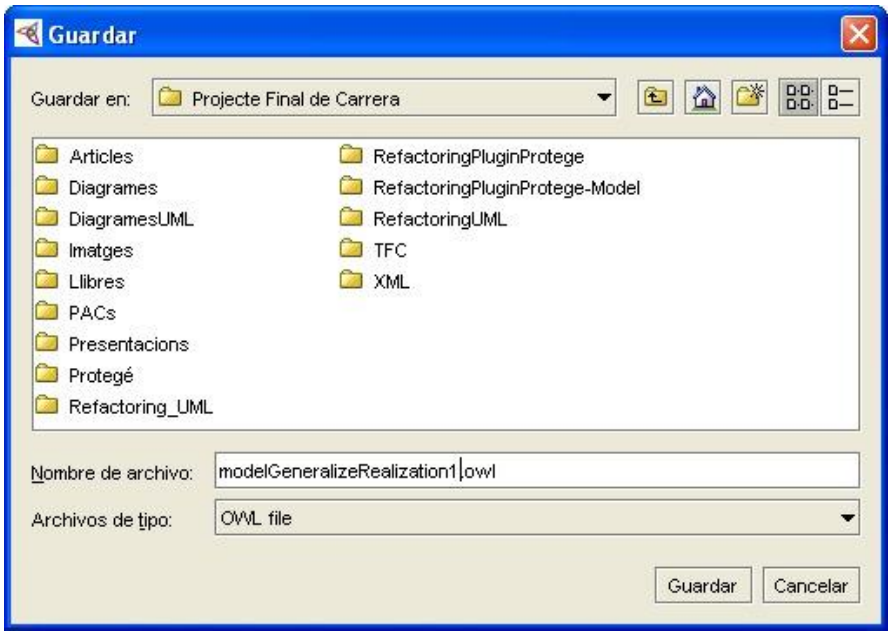


Fig. 42 - Save Model As...

5.2.6.- Emmagatzemar resultats finals i Canvi d'ontologia.

Un cop acabat el procés de *refactoring*, si es consideren definitius els resultats i es desitja emmagatzemar-los en disc, només cal escollir les opcions que ofereix **Protégé** des del menú File: Save i Save As.... El mateix cal fer si es vol treballar en una altra ontologia: escollir en el menú File alguna de les opcions Open o Open Recent.

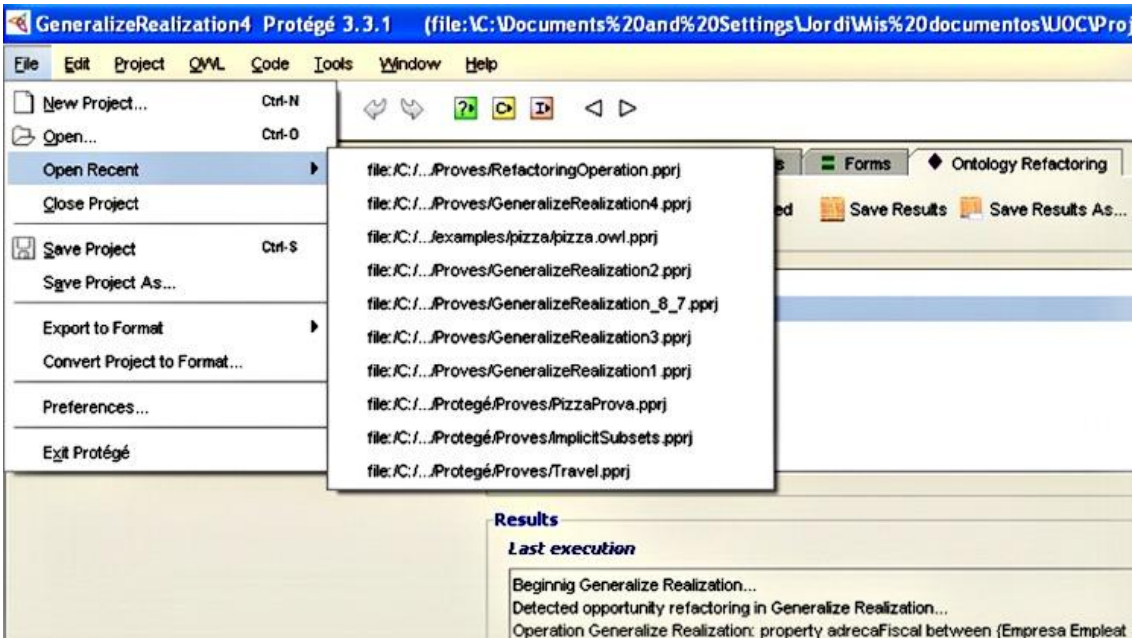


Fig. 43 - Emmagatzemar resultats finals. Obrir nou projecte.

6.- Proves de funcionament.

Els aspectes desenvolupats han estat tots provats i s'ha comprovat que funcionen correctament. L'apartat anterior proporciona algunes mostres del funcionament de l'aplicació, per la qual cosa en aquest ens limitarem a indicar quines han estat les pràctiques de comprovació realitzades, així com a mostrar d'altres resultats obtinguts.

Les proves efectuades han estat les següents:

- S'ha comprovat que el *plugin* s'integra en l'entorn de **Protégé** i que el fitxer de configuració funciona correctament, de forma que el *plugin* carrega les distintes operacions definides en el fitxer.
- S'ha comprovat que els dos algorismes implementats funcionen adequadament, executant de manera correcta les dues operacions de *refactoring*.
- També s'ha comprovat que s'emmagatzemen correctament tan els resultats de l'execució com els models intermedis obtinguts.

Tot seguit es detallen les proves realitzades i els resultats que s'han donat per a cadascun dels algorismes.

6.1.- Prova de l'operació *Implicit Subsets*.

En les proves s'han utilitzat tres ontologies. La primera prova s'ha realitzat en la següent ontologia que és una petita extensió d'un dels models que apareix com a exemple en el document *Pruning and Refactoring Ontologies in the Development of Conceptual Schemas of Information Systems*:

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://www.owl-pfc.com/2008/OntologyImplicitSubsets.owl#"
     xml:base="http://www.owl-pfc.com/2008/OntologyImplicitSubsets.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <owl:Ontology rdf:about=""/>
    <owl:Class rdf:ID="Analyst">

        <rdfs:subClassOf rdf:resource="#Employee"/>
        <rdfs:subClassOf rdf:resource="#ComputerExpert"/>
    </owl:Class>

    <owl:Class rdf:ID="ComputerExpert">
        <rdfs:subClassOf rdf:resource="#Employee"/>
    </owl:Class>

    <owl:Class rdf:ID="Employee"/>

    <owl:Class rdf:ID="SystemAnalist">
        <rdfs:subClassOf rdf:resource="#Analyst"/>
        <rdfs:subClassOf rdf:resource="#ComputerExpert"/>
    </owl:Class>

</rdf:RDF>
```

La figura 44 mostra el resultat de l'execució del *plugin* conjuntament amb l'estat de l'ontologia abans i després de l'execució.

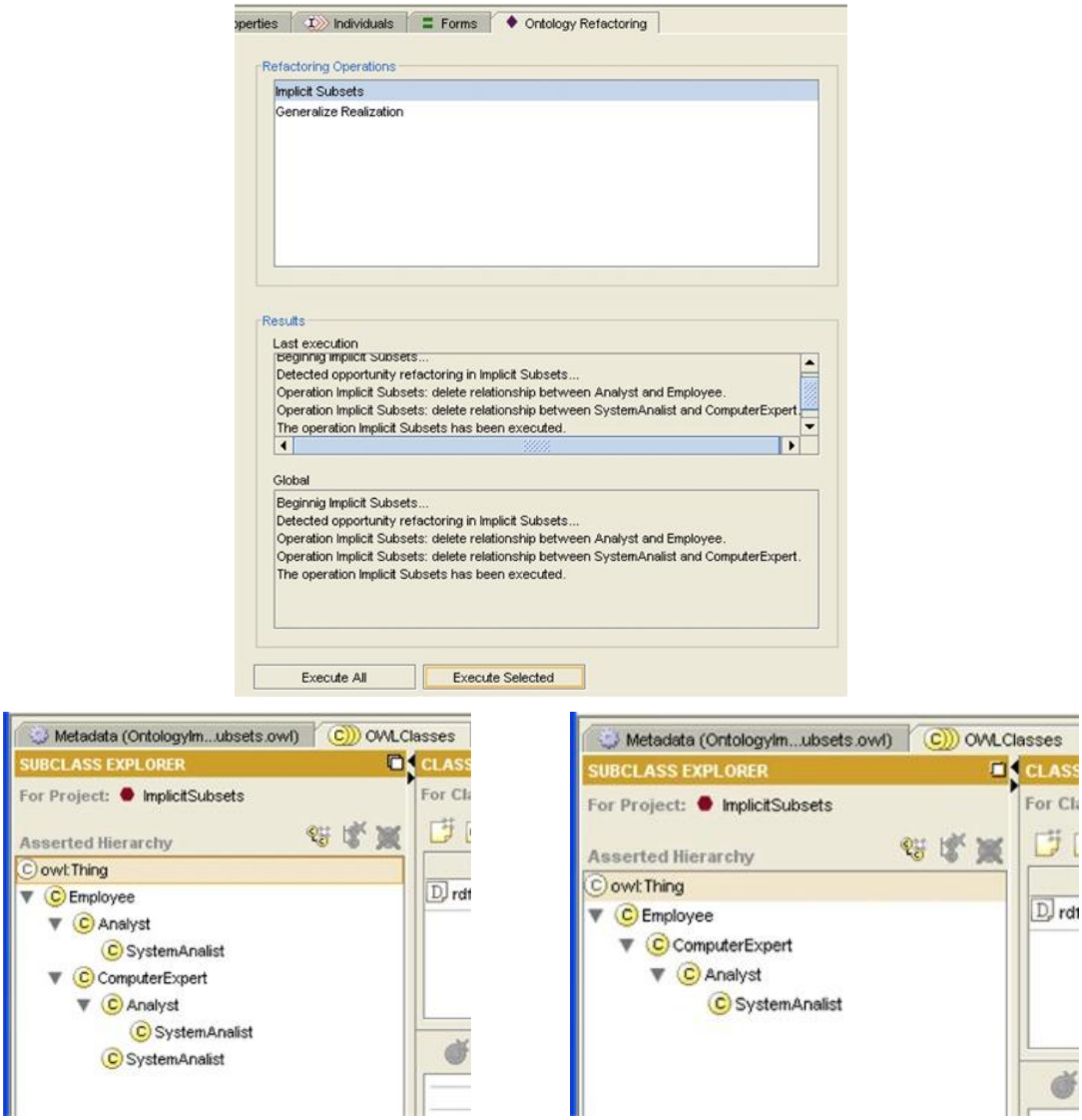


Fig. 44 - *Implicit Subsets*. Prova 1.

La segona prova s'ha realitzat amb una ontologia que complica el model anterior. Es mostra tot seguit, conjuntament amb els resultats obtinguts.

```
<?xml version="1.0"?>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.owl-pfc.com/2008/OntologyImplicitSubsets.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-pfc.com/2008/OntologyImplicitSubsets.owl">

  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="SubClassAnalyst">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Employee"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Analyst"/>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="SystemAnalist">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="ComputerExpert"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Analyst"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
    <owl:Class rdf:about="#Analyst">
      <rdfs:subClassOf>
        <owl:Class rdf:about="#ComputerExpert"/>
      </rdfs:subClassOf>
      <rdfs:subClassOf rdf:resource="#Employee"/>
    </owl:Class>

    <owl:Class rdf:about="#ComputerExpert">
      <rdfs:subClassOf rdf:resource="#Employee"/>
    </owl:Class>

    <owl:Class rdf:ID="SubClassSystemAnalist">
      <rdfs:subClassOf rdf:resource="#Employee"/>
      <rdfs:subClassOf rdf:resource="#ComputerExpert"/>
      <rdfs:subClassOf rdf:resource="#Analyst"/>
      <rdfs:subClassOf rdf:resource="#SystemAnalist"/>
    </owl:Class>

  </rdf:RDF>
```

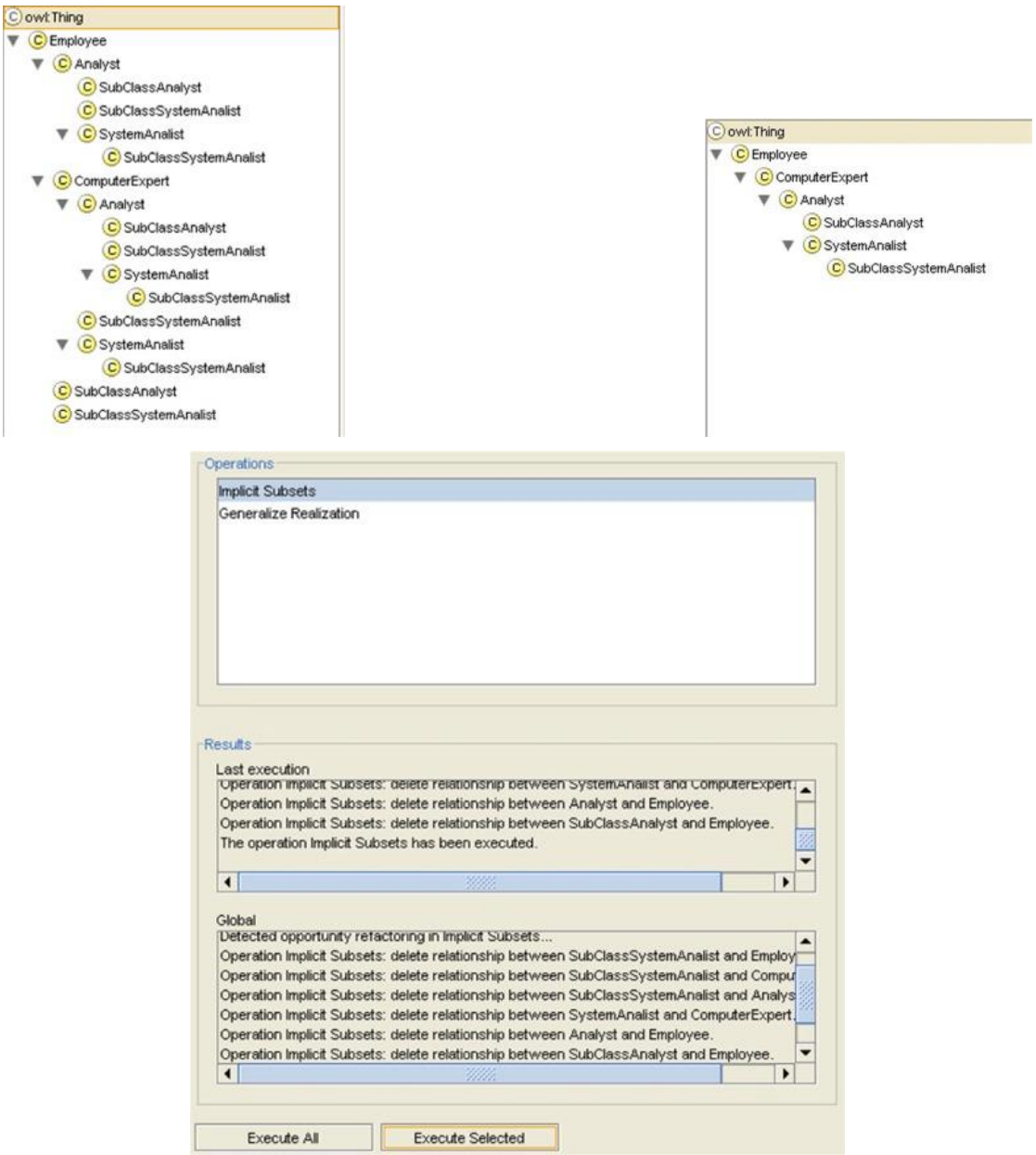


Fig. 45 - Implicit Subsets. Prova 2.

S'ha aplicat també el *plugin* a l'ontologia sobre viatges que es pot descarregar o bé importar des de la següent adreça:

http://smi-protege.stanford.edu/repos/protege/owl/trunk/examples/travel.owl

El resultat obtingut ha estat que l'ontologia no presenta oportunitats de refacció associades a subconjunts implícits.

6.2.- Prova de l'operació *Generalize Realization*.

Per a provar aquest algorisme s'han emprat dues ontologies. La primera prova s'ha realitzat en una ontologia que és una petita variació de l'exemple sobre *Reports* i *Analistes* que apareix també com a exemple en el document esmentat anteriorment:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.owl-pfc.com/2008/Ontology1225273774.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-pfc.com/2008/Ontology1225273774.owl">

  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="Report">
    <owl:equivalentClass>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:ID="CostReport"/>
          <owl:Class rdf:ID="FinancialReport"/>
        </owl:unionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>

  <owl:Class rdf:about="#FinancialReport">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="dateCarriedOut"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="FinancialAnalyst"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasAnalyst"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Report"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasAnalyst"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#CostReport"/>
    </owl:disjointWith>
  </owl:Class>

  <owl:Class rdf:about="#CostReport">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="#dateCarriedOut"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:about="#FinancialAnalyst"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasAnalyst"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
```

```
    <rdfs:subClassOf rdf:resource="#Report"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasAnalyst"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#FinancialReport"/>
  </owl:Class>

  <owl:Class rdf:ID="Person"/>

  <owl:Class rdf:about="#FinancialAnalyst">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="studiesReport"/>
        </owl:onProperty>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#studiesReport"/>
        </owl:onProperty>
        <owl:allValuesFrom>
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="#CostReport"/>
              <owl:Class rdf:about="#FinancialReport"/>
            </owl:unionOf>
          </owl:Class>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>

  <owl:ObjectProperty rdf:about="#hasAnalyst">
    <rdfs:range rdf:resource="#FinancialAnalyst"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#studiesReport"/>
    </owl:inverseOf>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#FinancialReport"/>
          <owl:Class rdf:about="#CostReport"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#studiesReport">
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#FinancialReport"/>
          <owl:Class rdf:about="#CostReport"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
    <rdfs:domain rdf:resource="#FinancialAnalyst"/>
    <owl:inverseOf rdf:resource="#hasAnalyst"/>
  </owl:ObjectProperty>

  <owl:DatatypeProperty rdf:about="#dateCarriedOut">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#FinancialReport"/>
          <owl:Class rdf:about="#CostReport"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:DatatypeProperty>

</rdf:RDF>
```

Les figures següents mostren el resultat de l'execució del *plugin* conjuntament amb l'estat de l'ontologia abans i després de l'execució.

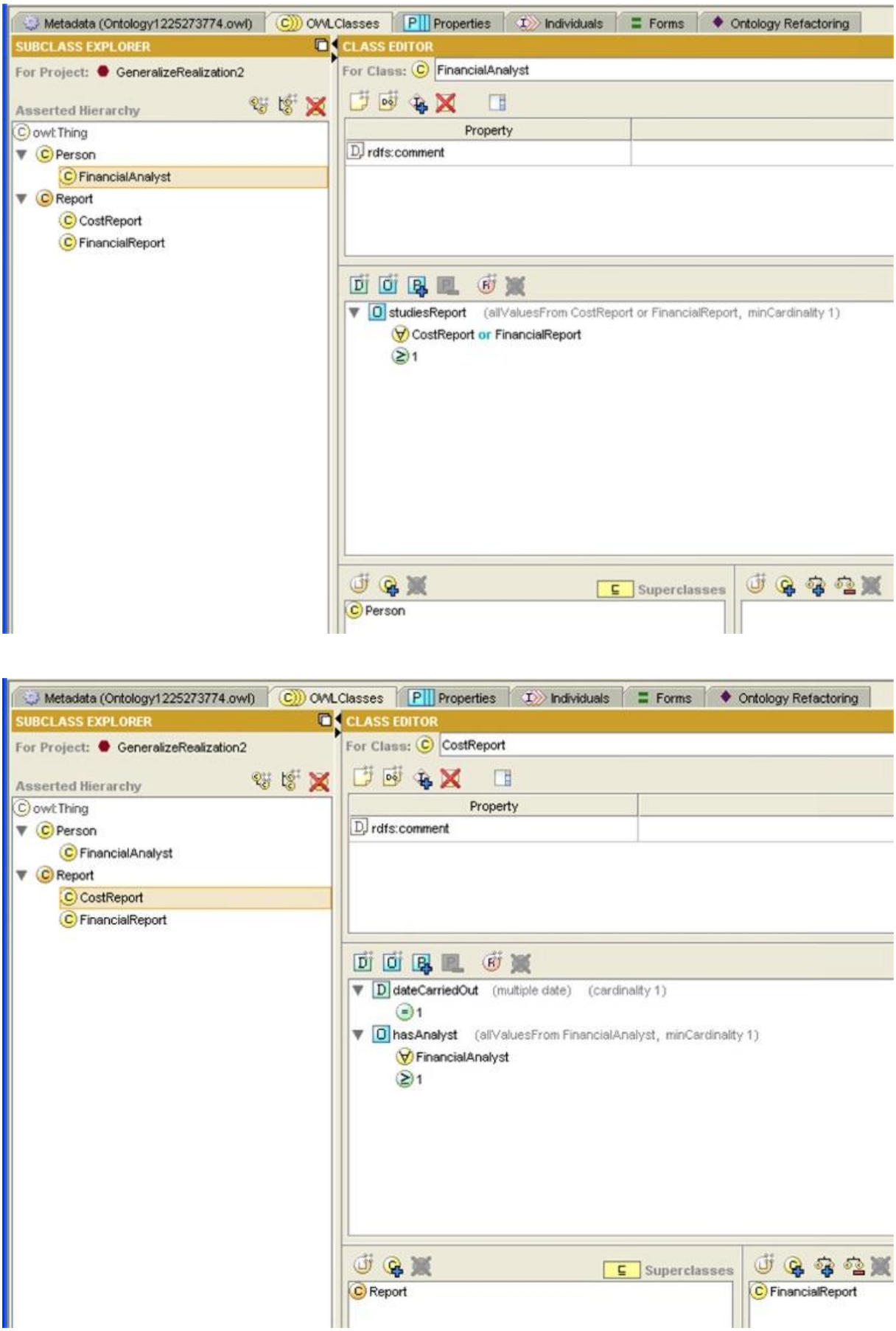


Fig. 46 - *Generalize Realization*. Prova 1: estat inicial de l'ontologia.

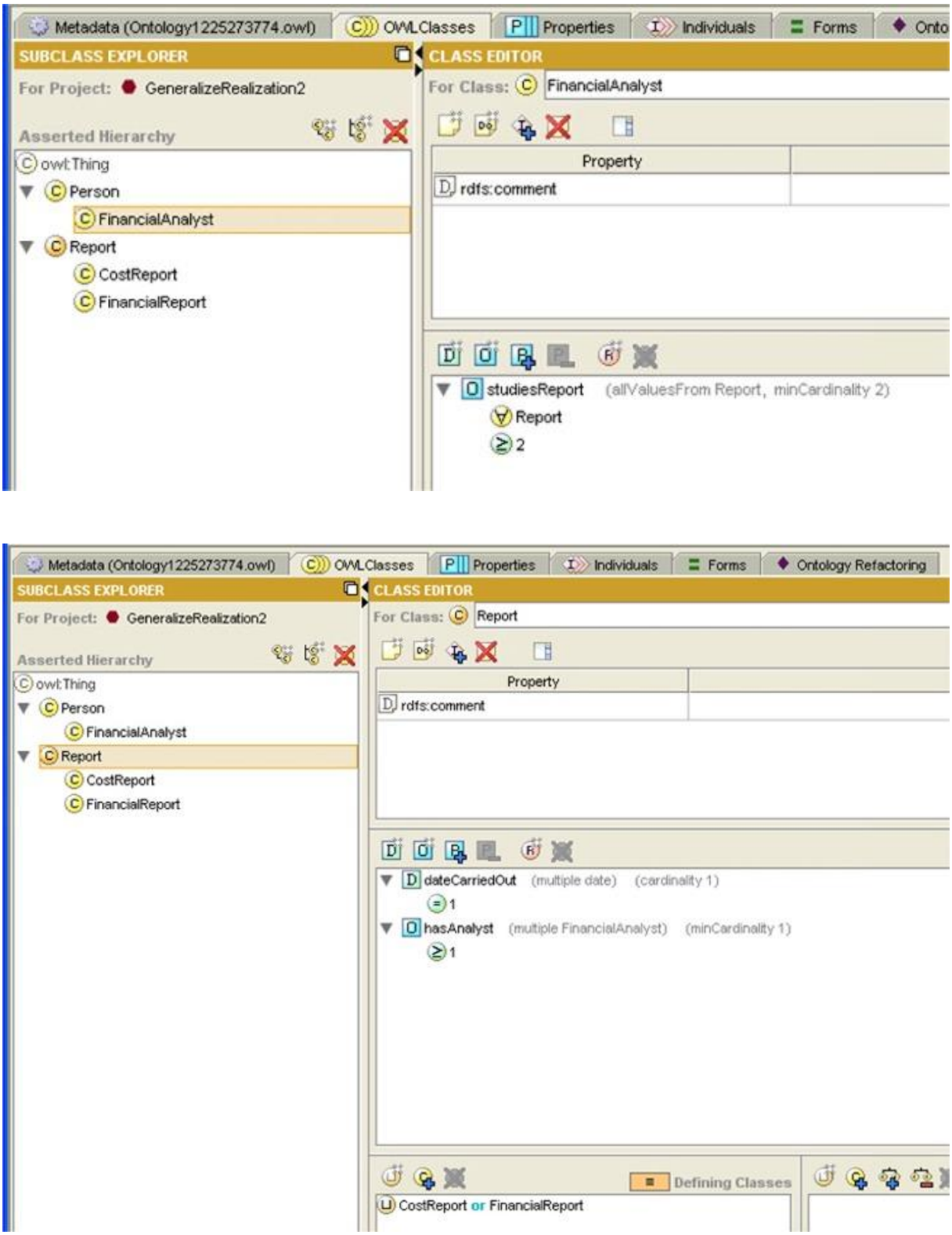


Fig. 47 - *Generalize Realization*. Prova 2: estat final de l'ontologia.

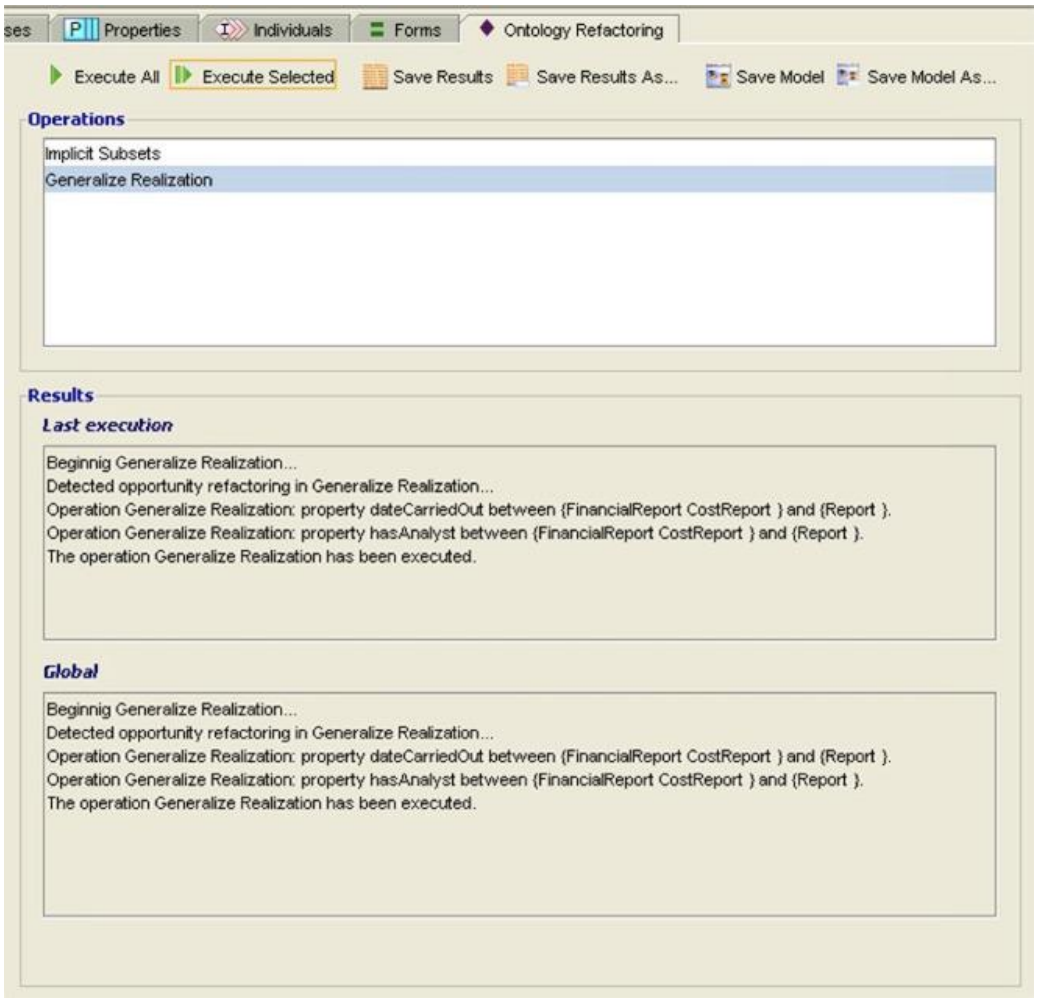


Fig. 48 - *Generalize Realization*. Prova 1: Log d'execució.

La segona prova s'ha realitzat amb una ontologia basada en un model conceptual que apareix en una de les referències bibliogràfiques (Olivé, 2001). Es mostra tot seguit, conjuntament amb els resultats obtinguts.

```
<?xml version="1.0"?>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.owl-pfc.com/2008/Ontology1225273774.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-pfc.com/2008/Ontology1225273774.owl">

  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="Dona">
    <owl:disjointWith>
      <owl:Class rdf:ID="Home"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Persona"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="edat"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="Estudiant">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Persona"/>
```

```
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Empleat">
  <owl:disjointWith>
    <owl:Class rdf:ID="Empresa"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Contribuent"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Persona"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="adrecaFiscal"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Persona">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Home"/>
        <owl:Class rdf:about="#Dona"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:about="#Contribuent">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Empleat"/>
        <owl:Class rdf:about="#Empresa"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:ID="MenorEdat">
  <rdfs:subClassOf rdf:resource="#Persona"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#edat"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Empresa">
  <owl:disjointWith rdf:resource="#Empleat"/>
  <rdfs:subClassOf rdf:resource="#Contribuent"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#adrecaFiscal"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Home">
  <rdfs:subClassOf rdf:resource="#Persona"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="#edat"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Dona"/>
</owl:Class>

<owl:DatatypeProperty rdf:about="#adrecaFiscal">
```

```
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Empresa"/>
          <owl:Class rdf:about="#Empleat"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#edat">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Dona"/>
          <owl:Class rdf:about="#Home"/>
          <owl:Class rdf:about="#MenorEdat"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>

  <owl:FunctionalProperty rdf:ID="nif">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Empleat"/>
          <owl:Class rdf:about="#Empresa"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>

</rdf:RDF>
```

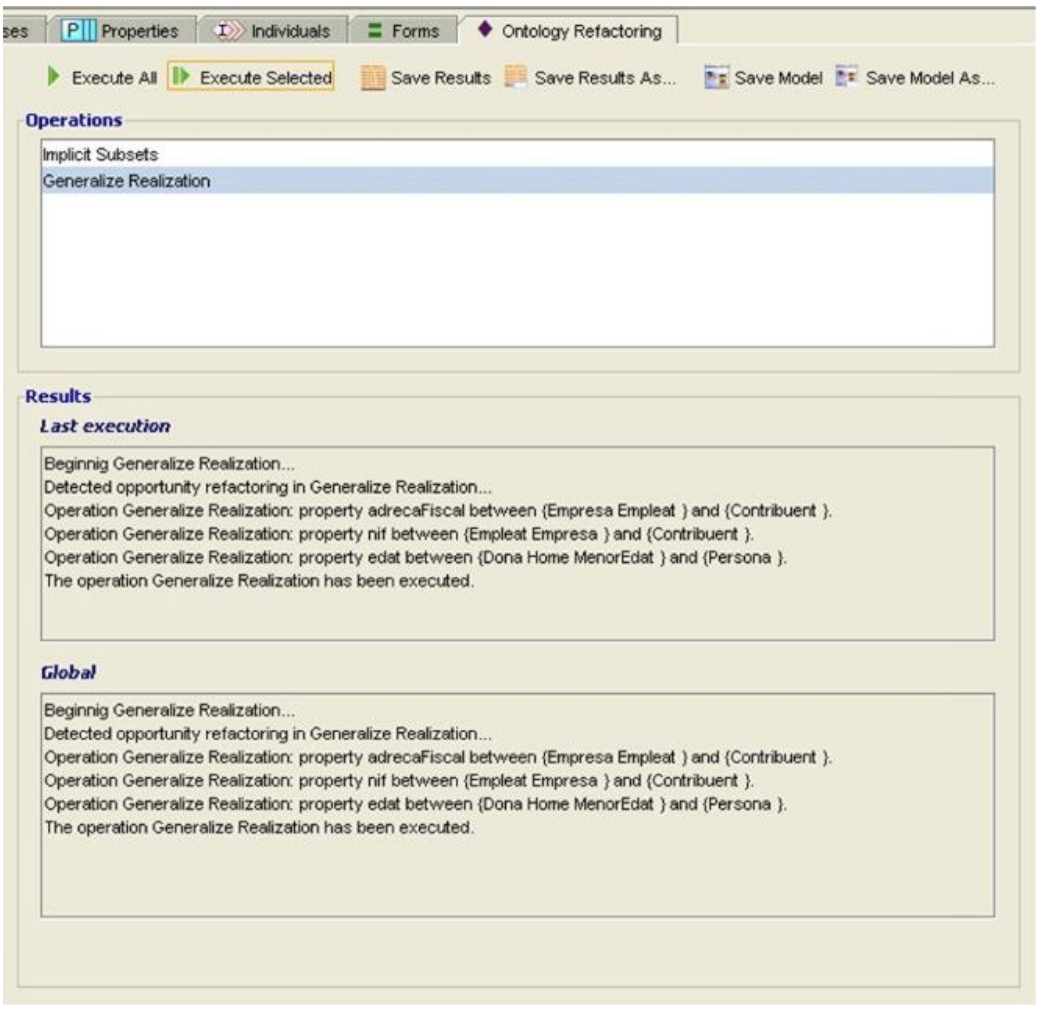


Fig. 49 - *Generalize Realization*. Prova 2: Log d'execució.

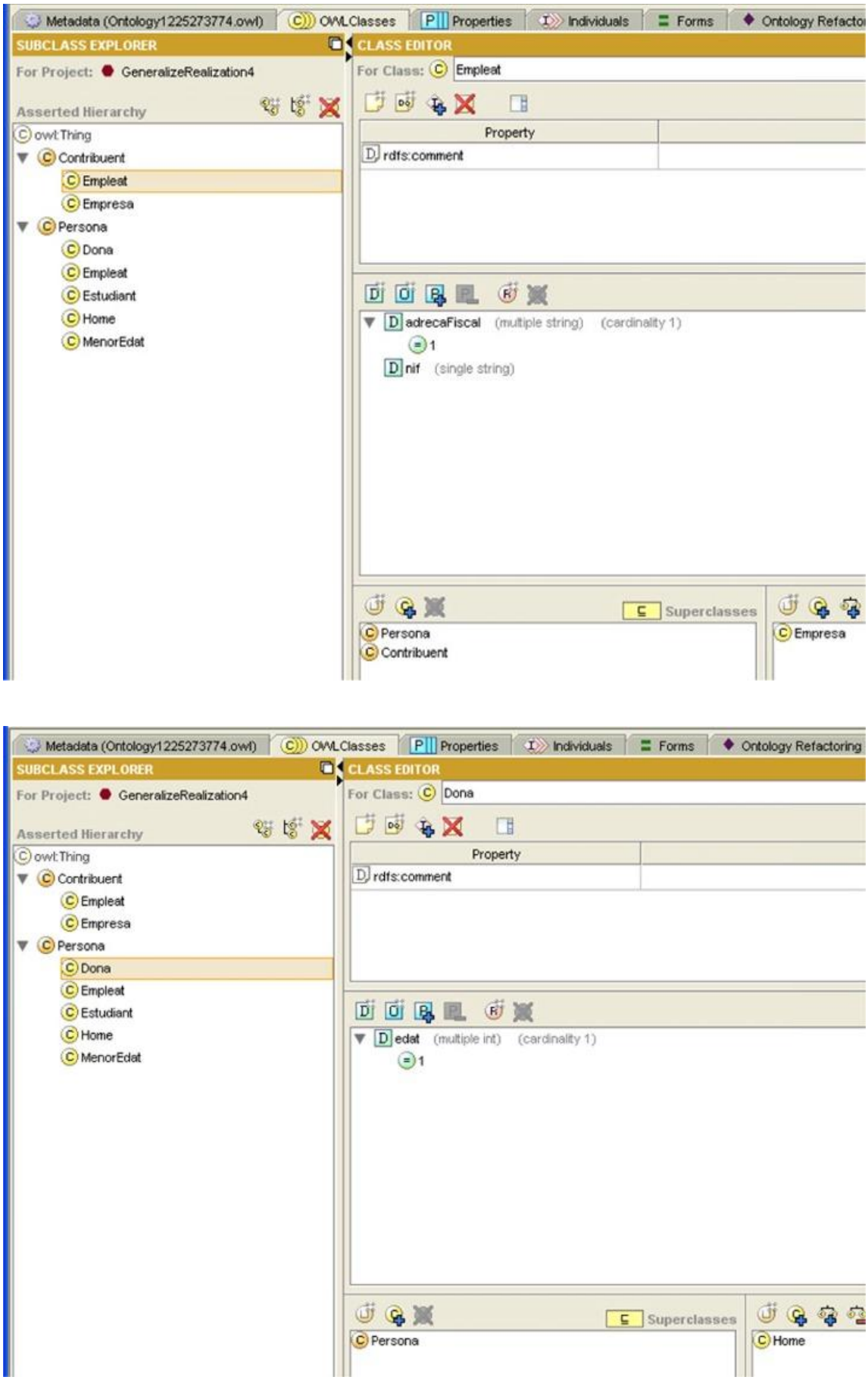


Fig. 50 - *Generalize Realization*. Prova 2: estat inicial de l'ontologia.

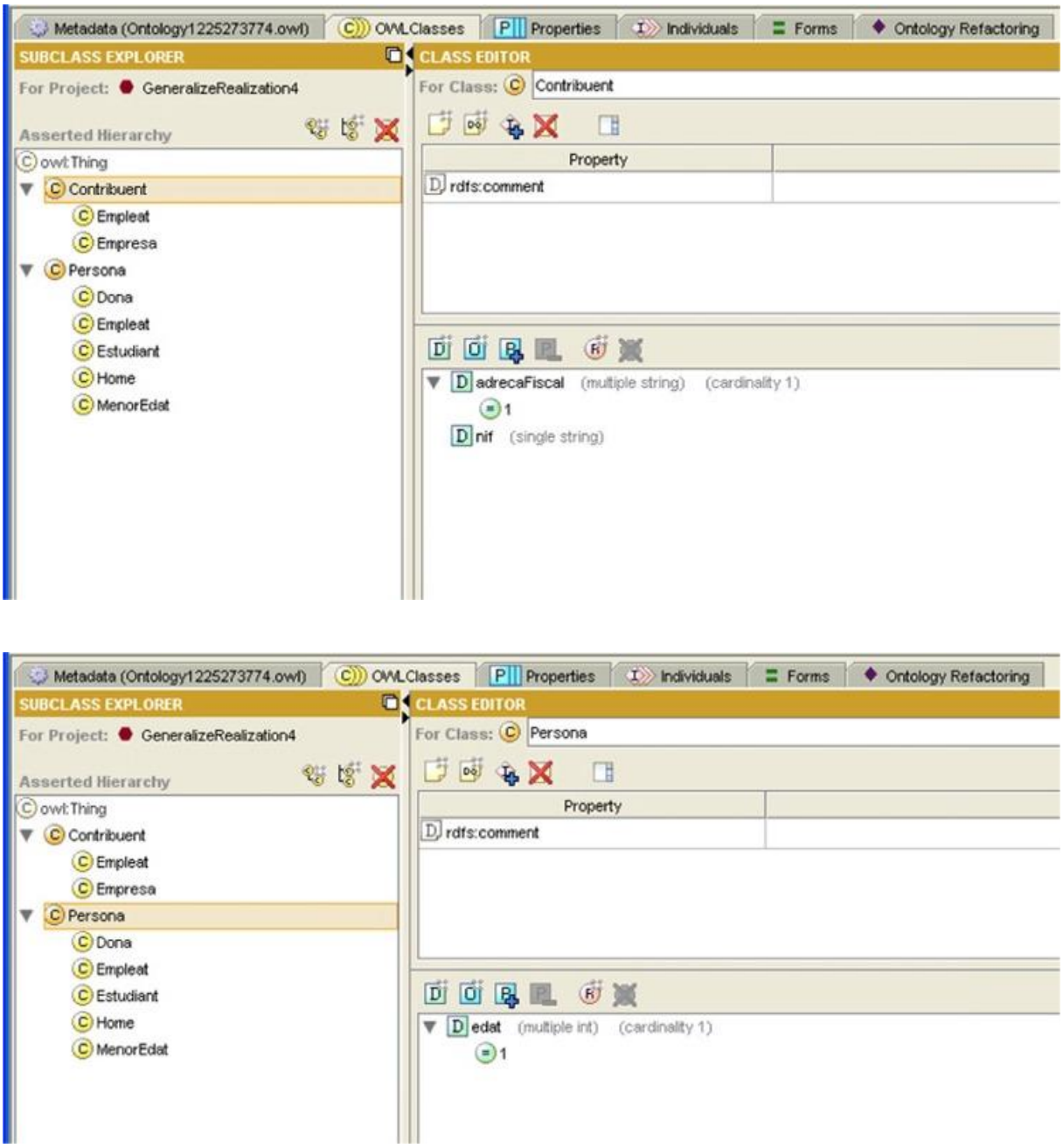


Fig. 51 - *Generalize Realization*. Prova 2: estat final de l'ontologia (1).

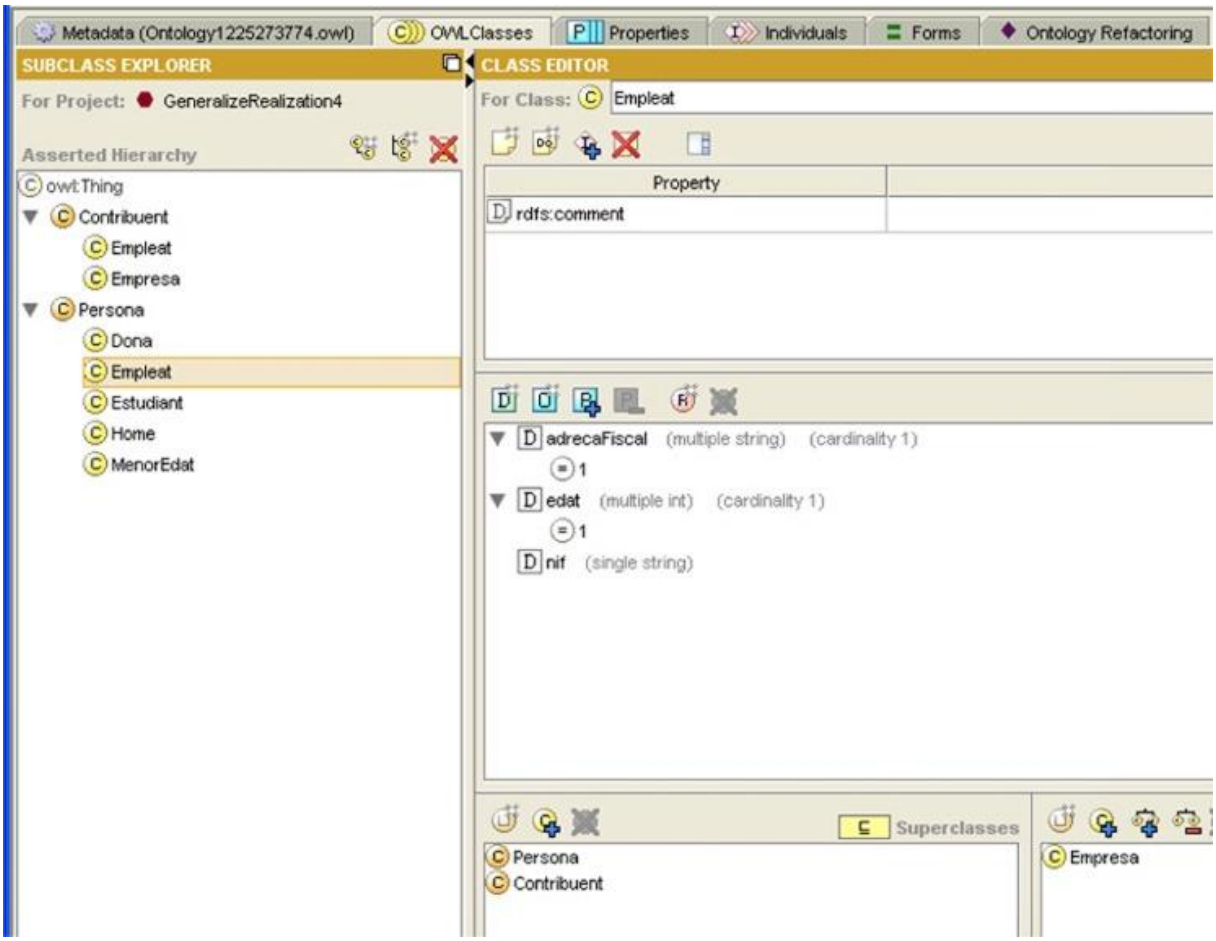


Fig. 52 - *Generalize Realization*. Prova 2: estat final de l'ontologia (2).

6.2.- Proves de salvaguarda.

La figura següent mostra l'execució d'un procés de salvaguarda d'un model intermedi i una part del fitxer final obtingut visualitzat en un navegador.

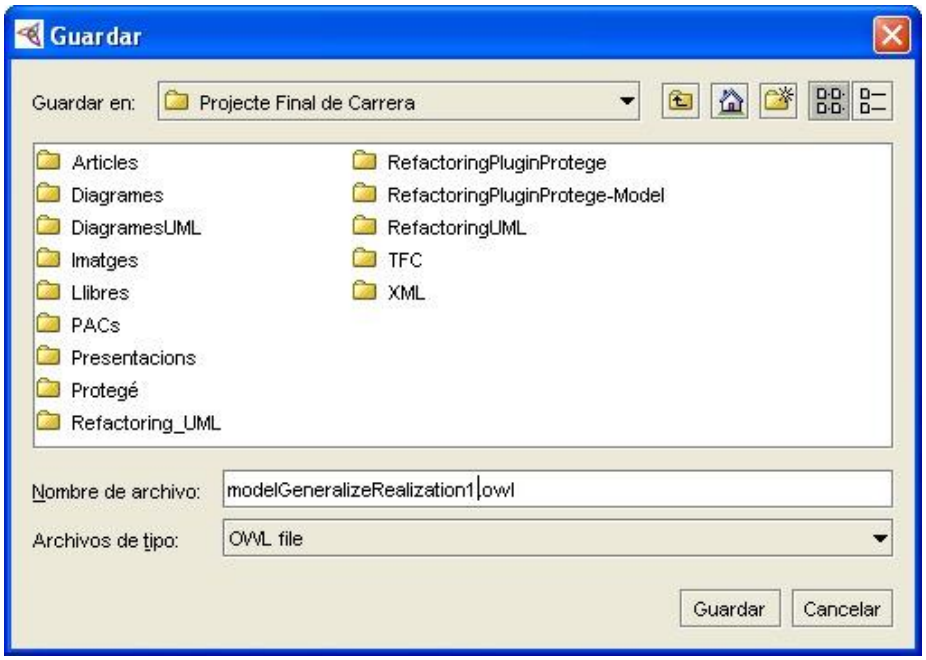


Fig. 53 - Prova procés de salvaguarda.

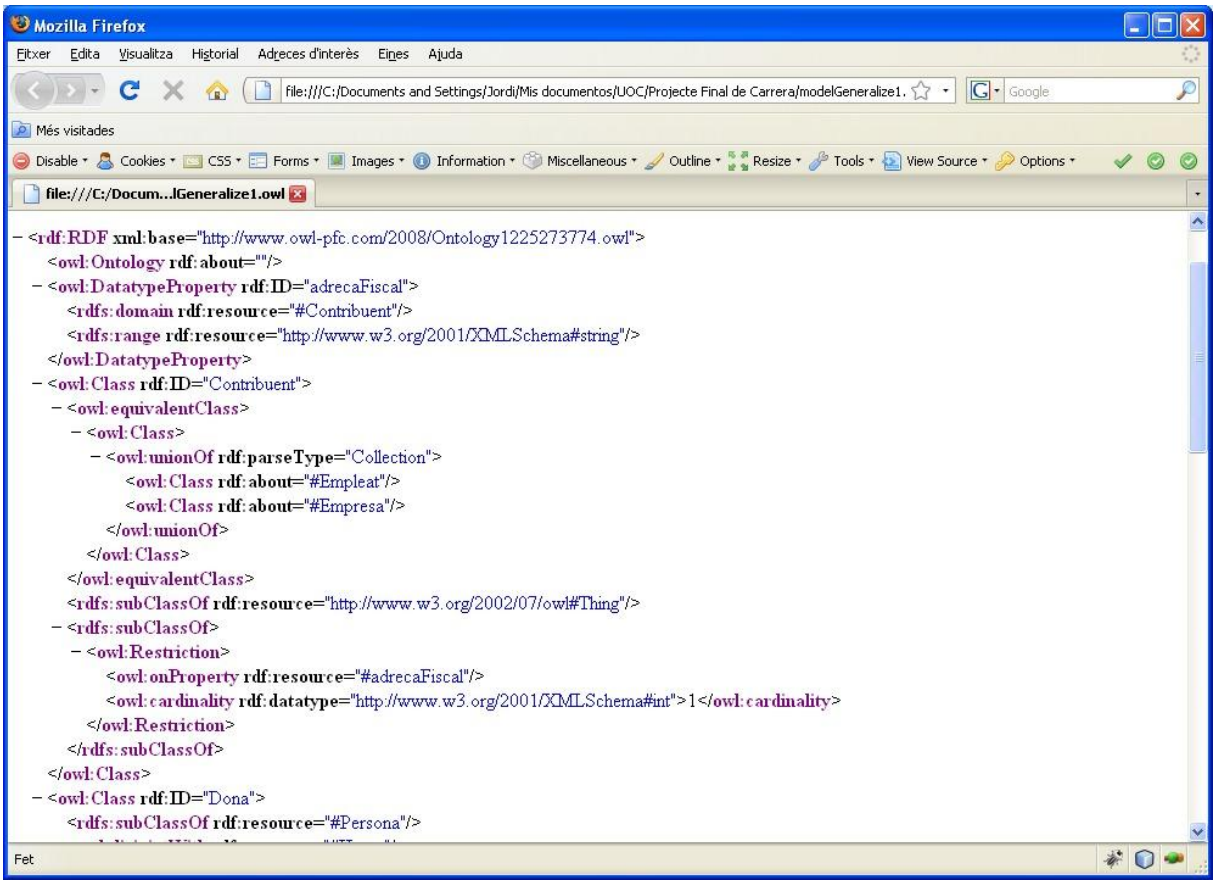


Fig. 54 - Resultat procés de salvaguarda.

7.- Conclusions i comentaris finals.

Com a conclusió final podem indicar que gairebé tots els objectius que es van plantejar en iniciar el PFC –recollits en el Pla de treball– s'han pogut complir. El producte obtingut satisfà les funcionalitats previstes i l'estudiant ha avançat en el coneixement del que representa la Web Semàntica, en la importància que hi tenen les ontologies i els llenguatges emprats per a descriure-les.

També s'ha avançat en tot el que es refereix al *refactoring*: es coneix el seu concepte i, sobretot, la seva aplicació al camp de les ontologies. En aquest sentit, hi ha hagut la possibilitat d'analitzar el catàleg de les operacions que s'hi poden aplicar i es disposa d'un coneixement prou profund de les dues operacions implementades. La descoberta de **Protégé** i de la seva API per a **OWL** també ha estat un dels punts interessants del projecte. En aquest sentit, hagués estat bo poder disposar de més temps per a poder provar la plataforma en el que es refereix a la comprovació de la consistència de les ontologies i a les seves possibilitats per al raonament.

En el transcurs del projecte, un cop esbossades i provades les línies mestres dels algorismes implementats, l'estudiant havia manifestat als consultors l'interès en valorar i/o tractar determinades qüestions, entre d'altres:

- Referent al primer dels algorismes implementats –*Implicit Subsets*–, s'havien plantejat dubtes sobre com considerar les *no-named class* de l'**OWL**.
- Referent a l'operació *Generalize Realization*, un dels problemes era com tractar el tema de les *superproperties* en **OWL**.

En ambdós casos, els consultors van aconsellar obviar aquestes qüestions donat el temps disponible, ja que, si es consideraven, les operacions de *refactoring* es complicaven –en el segon dels casos sembla ser que molt.

Una altra qüestió que es va plantejar en el transcurs del projecte fou la següent –transcriu fidelment, tan la consulta realitzada com la resposta rebuda–:

> *A l'hora d'aplicar l'operació [Generalize Realization] a l'OWL, a més de les restriccions de cardinalitat se'n poden trobar d'altres associades a una determinada propietat en les subclasses afectades pel procés de generalització. En posaré un exemple. Suposem que en la traducció a OWL de l'exemple de la figura 8.6 b [Fig. 12 en aquest document], a l'hora de definir una propietat hasAnalyst per als reports s'afegeix a cada subclasse la següent restricció:*
>
> *onProperty hasAnalyst allValuesFrom FinancialAnalyst*
>
> *Com actuem en el procés de refacció? És a dir, què fem amb les restriccions de les subclasses afectades pel procés en el cas de les restriccions que no són del tipus cardinalitat?*
>
> *El que m'ha semblat lògic de fer és el següent: moure als pares totes aquelles restriccions comunes als fills. M'agradaria saber si ho trobeu correcte o no. També estava valorant aquests darrers dies si no seria més correcte moure-hi aquelles que, o bé són comunes a tots als fills, o bé ho són a aquells pels quals la generalització és completa.*
>
> Les restriccions d'integritat (que no siguin de cardinalitat) no és poden moure amb aquesta operació.
>
> T'explicaré el perquè amb un exemple:
>
> Suposa que tens:
>
> Agent Gens Persona, Negoci
>
> Persona i Negoci tenen un atribut anomenat nom.
>
> Persona i Negoci tenen una restricció d'integritat que diu que no poden haver dues instàncies de la classe amb el mateix nom. Per tant, no hi poden haver dues persones amb el mateix nom, ni dos negocis amb el mateix nom.
>
> En cas de que fem l'operació de refactoring i moguem la restricció cap a dalt, tindríem que no hi poden haver dos Agents amb el mateix nom. És això equivalent al que teníem abans? La resposta és no. Abans de l'operació de *refactoring* podíem tenir un negoci i una persona amb el mateix nom, després de fer el *refactoring* això no.
>
> Sí que és cert que hi ha algunes restriccions que es podrien moure cap a dalt, però només algunes d'elles i sota condicions extres. Les operacions de *refactoring* han de ser molt simples, i més tard composar-les per crear-ne de més complexes. Seguint aquesta filosofia, moure restriccions d'integritat als pares és feina d'una altra operació de refactoring: *Pull Up Integrity constraint*.
>
> El tema de les restriccions és més complex del que sembla.

Hi ha una qüestió a destacar en aquesta resposta que en el seu moment havia passat en part desapercebuda per a l'estudiant i que, en redactar aquest document li ha vingut de nou a la memòria perquè coincideix amb una de les característiques de les operacions de *refactoring* que comenta Martin Fowler: el fet que aquestes operacions han de ser molt simples i és la seva composició la que acaba ocasionant els efectes perseguits.

Relacionant aquesta darrera resposta amb els consells proporcionats pels consultors al llarg del projecte, es fa palesa la possibilitat de millorar el producte obtingut treballant en els següents sentits:

- Acabant de polir les operacions implementades, considerant els aspectes que s'han obviat.
- Estudiant la forma d'afegir noves operacions del catàleg a les ja implementades.
- Finalment, recordar que en parlar de l'activitat de *refactoring* en el camp de les ontologies s'ha fet esment a una etapa en què el sistema podria ajudar al desenvolupador en el procés de reestructuració de l'ontologia.

  Per a poder realitzar aquesta tasca, caldria millorar la comunicació amb l'usuari, de forma que, cas de no poder efectuar el procés de manera automàtica, s'informés amb més detall dels motius que han portat a la decisió.

Glossari.

Terme	Descripció
DTD	Acrònim de *Document Type Definition*. Descripció d'un conjunt de regles **XML** per a determinar com s'interpreten els documents **XML** d'un determinat tipus.
Generalize Realization	Operació de *refactoring* d'ontologies que substitueix un conjunt de realitzacions d'un determinat tipus de relació genèrica definida en tipus d'entitat que són germans per realitzacions del mateix tipus en què intervenen els pares comuns.
Implicit Subsets	Operació de *refactoring* d'ontologies que elimina una relació de generalització redundant, és a dir, una generalització en què dos conceptes es relacionen per més d'un camí, de forma o de forma indirecta.
Ontologia	Vocabulari que permet especificar explícita i formalment els conceptes relacionats amb un determinat domini, descrivint les seves propietats i les restriccions que els afecten.
Oportunitat de refacció	Condició que indica quan l'execució d'una determinada operació millora l'ontologia i a quins elements cal aplicar-la.
OWL	Llenguatge definit pel *Web Ontology Working Group* del **W3C** per a descriure ontologies en la Web Semàntica. Es construeix sobre **RDF** i **RDFS** i utilitza la seva sintaxi basada en **XML**. Es presenta en tres variants o dialectes: **OWL-Lite**, **OWL-DL** i **OWL-Full**. El primer és el menys expressiu, mentre que el darrer és el que ho és més, la qual cosa el fa inadequat per a realitzar raonaments de tipus automàtic.
Protégé	*Framework* de codi obert que permet editar ontologies i gestionar la base de coneixement associada. Desenvolupat en **Java** per la *Universitat de Stanford* amb col·laboració amb la de *Manchester*, ofereix la possibilitat de modelar les ontologies en **OWL** i permet ser estès a través del mecanisme ofert pels *plugins*.
Refacció	Procés a través del qual es reescriu un material amb l'objectiu de millorar-ne tant la seva comprensió com l'estructura interna, amb el ferm propòsit de conservar-ne el significat i el comportament.
Refactoring	Veure refacció.
Refactoring opportunity	Veure oportunitat de refacció.
RDF	Acrònim de *Resource Description Framework*. Llenguatge emprat per a definir els models de dades que permeten escriure sentències senzilles sobre els recursos disponibles en la web.
RDFS –RDF Schema–	Llenguatge o vocabulari basat en **RDF** que permet descriure els recursos mitjançant una orientació a l'objecte, definint classes i propietats, relacions entre ambdós conceptes, relacions d'herència i realitzant restriccions sobre els distints elements del domini o del rang de les propietats.
URI	Acrònim de *Uniform Resource Identifier*. Conjunt de normes que permet especificar de forma inequívoca una referència a qualsevol recurs de la xarxa.
Web semàntica	Extensió de l'actual Web que pretén dotar-la de més significat, de forma que els usuaris, amb l'ajut d'agents de *software*, puguin resoldre les cerques d'informació de forma més ràpida, senzilla i, sobretot, més precisa per tal de poder satisfer les seves necessitats.
XML	Acrònim de *eXtensible Mark-up Language*. És una especificació que defineix una sintaxi i unes regles sobre l'ús d'etiquetes per a estructurar la informació.
XML Schema	Estàndard que utilitza la notació **XML** per a definir les regles que han de complir els documents **XML** per a ser considerats vàlids. Suporta distints tipus de dades i és extensible, permetent la definició de nous tipus.

Bibliografia.

Llibres:

Akif, M.; [et al.] (2001) *Java y XML.* (1a ed.) Wrox Press.

Allemang, D.; Hendler, J. (2008) *Semantic Web for the Working Ontologist. Modeling in RDF, RDFS and OWL.* (1a ed.) Morgan Kaufmann Publishers.

Antoniou, G.; Van Harmelem, F. (2008) *A Semantic Web Primer.* (2a ed.) MIT Press.

Batini, C.; [et al.] (1994) *Diseño conceptual de bases de datos.* (1a ed.) Addison-Wesley.

Fowler, M; [et al.] (1999) *Refactoring: Improving the Design of Existing Code.* (1a ed.) Addison-Wesley.

Harold, E.R. (2004) *XML 1.1 Bible.* (3a ed.) Wiley Publishing.

Horstmann, C.S.; Cornell, G. (2006) *Core Java 2 Volumen I - Fundamentos.* (7a ed.) Prentice Hall.

Horstmann, C.S.; Cornell, G. (2006) *Core Java 2 Volume II – Advanced Features.* (4a ed.) Prentice Hall.

Olivé, A. (2001) *Modelització Conceptual de Sistemes d'Informació. L'estructura.* (1a ed.) Edicions UPC.

Articles i altres documents:

Conesa, J. (2008) *Pruning and Refactoring Ontologies in the Development of Conceptual Schemas of Information System.*

Conesa, J. (2004) *Ontology-Driven Information Systems: Pruning and Refactoring of Ontologies .* Disponible a http://www-ctp.di.fct.unl.pt/UML2004/DocSym/JordiConesaUML2004DocSym.pdf

Horridge, M.; [et al.] (2004) *A Practical Guide to Building OWL Ontologies Using the Protégé-OWL Pluguin and CO-ODE Tools.* Disponible a http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf

Horridge, M. *Protégé OWL API Interfaces.* Disponible a http://protege.stanford.edu/plugins/owl/api/ProtegeOWLModelWithProtege.pdf

Knublauch, H.; [et al.] (2005) *The Protégé OWL plugin: An Open Development Environment for Semantic Web Applications.* Disponible a http://protege.stanford.edu/plugins/owl/publications/ISWC2004-protege-owl.pdf

Noy, N.F.; McGuinness, D.L. *Ontology Development 101: A Guide to Creating Your First Ontology* . Disponible a http://protege.stanford.edu/publications/ontology_development/ontology101.pdf

Pàgines web:

Protégé-OWL API Programmer's Guide. http://protege.stanford.edu/plugins/owl/api/guide.html

Protégé Developer Documentation. http://protege.stanford.edu/doc/dev.html#plugins

Semantic Web. http://semanticweb.org

W3C Extensible Markup Language. http://www.w3.org/XML/

W3C Guía Breve de Web Semántica. http://www.w3c.es/divulgacion/guiasbreves/WebSemantica

W3C Resource Description Framework. http://www.w3.org/RDF/

W3C Semantic Web. http://www.w3.org/2001/sw/

W3C Web Ontology Language. http://www.w3.org/2004/OWL/