Comparació de Diagrames UML

Alumne

Josep Ribot i Costa
Enginyeria en Informàtica
jribotc@uoc.edu

Consultor

Jordi Cabot i Sagrera
jcabot@uoc.edu

19 juny 2006

Dedicatòria

A la Joana.

Als que no han arribat a veure-ho.

Agraïments

Son molts els agraïments que haig de fer i que m'han permès arribar fins aquí, des de la gent de la feina, que m'ha donat, des del primer moment, totes les facilitats del mon per a portar a terme els meus estudis, passant pels companys i amics que ens hem anat recolzant per fer això que no és fàcil de dir, però difícil de fer, com es estudiar quan a més tens altres obligacions familiars i professionals.

Però és evident que el meu agraïment més fort ha de ser per la família, a la qual he robat moments que els pertanyien, però que generosament han sacrificat.

No voldria acabar sense agrair a Jordi Cabot la seva tutoria en la realització del projecte, no només pel guiatge que ha fet, sinó també per la rapidesa en contestar als correus, doncs són moments en els que necessites que el recolzament a part de ser de qualitat també sigui ràpid.

Índex general

Índex de figures

Resum

El nucli del projecte consisteix en desenvolupar una aplicació permeti determinar si dos models UML[1] són iguals i , en cas de no ser-ho, determinar quines són les diferències entre ells. Donada la gran varietat de models UML, el projecte es centra en els diagrames de classes.

Així, ha estat l'objectiu d'aquest projecte el disseny i implementació d'un algoritme que permeti la comparació de dos models de diagrames de classes en format XMI[2] , que és un tipus especial d'XML[3], que usen per defecte moltes de les eines actuals de disseny de diagrames. De fet, tot i ser un estàndard, cadascú l'implementa com vol, i per tant el projecte es concentrar en utilitzar l'XMI generat per una eina concreta, que en aquest cas ha estat la generada per Poseidon.

L'algoritme s'ha dissenyat perquè sigui prou flexible, de tal manera que consideri iguals dos models que tinguin diferències que no afectin a la seva igualtat semàntica, com pot ser el fet d'utilitzar o no paquets o ser flexible a la nomenclatura dels elements dels models, incorporant-hi la possibilitat d'utilitzar sinònims en els elements subjectes a comparació.

Donat que estem en una universitat, cal centrar-nos en aplicacions de l'algoritme que ens afectin directament, per tant, el projecte es completa amb la implementació de l'algoritme obtingut en una aplicació web, que permeti als professors emmagatzemar enunciats i solucions de problemes, així com permetre als estudiants seleccionar un dels problemes emmagatzemats, adjuntar-hi la seva solució i rebre la notificació de si és una solució correcte i , en cas de no ser-ho, obtenir quins són els errors.

[1] Unified Modeling Language
[2] XML Metadata Interchange
[3] eXtensible Markup Language

Paraules clau

UML

Comparació diagrames UML

Comparació diagrames de classes

XMI

Aplicació web

Àrea del projecte

Generació automàtica de programari.

Capítol 1
Introducció

1.1 Justificació del projecte

La comparació de dos diagrames UML no té en si mateix gaire sentit, si es mira com un fet aïllat, i no es tenen en compte totes les utilitats que es poden derivar del fet de poder comparar dos models obtinguts prèviament. Òbviament, la pregunta que segueix és : I això perquè serveix?.

En entorns reals en facilitarà la integració d'esquemes, que és un dels problemes que es troba una empresa quan ha d'integrar el sistema d'informació d'una altre empresa que ha comprat o amb la que s'ha fusionat. Dins d'una mateixa empresa també pot servir per comprovar/integrar diferents especificacions d'una aplicació fetes per diferents treballadors. Tanmateix, pot servir per comprovar que la implementació d'una especificació es correspon amb les especificacions inicials, fent reenginyeria inversa de la implementació i comparant el model resultant amb l'original.

De tota manera, com a estudiant de la UOC, m'interessa enfocar el projecte cap a aspectes que puguin ser d'utilitat per la mateixa UOC, per tal d'ajudar a millorar en l'aprenentatge dels meus companys de l'assignatura d'Enginyeria del Programari de la UOC. De tots es sabut, que hi ha conceptes que són especialment difícils d'aprendre a distància, com és l'habilitat d'aprendre a modelar, és a dir, donat un enunciat textual saber definir el conjunt de classes, associacions, ... , que representen l'enunciat donat i ens costa molt identificar relacions ternàries, classes associatives, relacions recursives , entre d'altres.

Es tracta, doncs, d'acostar una mica més a l'estudiant a un aprenentatge interactiu de l'assignatura, cosa difícil degut a l'estructura de la UOC.

1.2 Objectius

L'objectiu bàsic d'aquest projecte és obtenir una aplicació que sigui capaç de comparar dos models UML de diagrames de classes i obtenir-ne les diferències, si es que aquestes existeixen. L'aplicació s'haurà d'implementar de tal manera que sigui flexible a possibles variacions semàntiques que no afecten a l'estructura del disseny.

L'aplicació per sí sola, no té més sentit que un exercici pràctic, per tant el seu disseny ha de permetre que aquesta aplicació sigui reutilitzable des de altres aplicacions de tal manera que es pugui utilitzar com una llibreria en altres aplicacions que els sigui necessari la comparació de models UML. En entorns reals, aquesta eina ha de poder tenir multitud d'aplicacions, com per exemple:

- Facilitar la integració d'esquemes.
- Comparar diferents especificacions d'una aplicació fetes per diferents treballadors de la mateixa empresa.
- Comprovar que la implementació d'una especificació és correspon amb l'especificació inicial, mitjançant la reenginyeria inversa la implementació, comparant el model resultant amb el model original.

Cal, doncs, que el projecte es plantegi un segon objectiu que es demostrar la possibilitat d'incorporar l'aplicació obtinguda en un altre aplicació, que en aquest cas serà una aplicació web dissenyada des de l'òptica de la UOC. Per tant, enfocarem el projecte cap a una aplicació que ens permeti augmentar les capacitats d'aprenentatge per un concepte que es difícil d'assolir a distància com és aprendre a modelar. Considerarem la situació en que el professor prepari diversos problemes, on cada problema està format per un enunciat (textual) i el diagrama de classes que es considera correcte. A través de la web l'estudiant veu els diferents problemes i a cada

problema hi adjunta la seva solució. La nostre aplicació ens ha d'indicar si l'estudiant ha resolt correctament el problema o no, amb la possibilitat de conèixer les diferències.

1.3 Enfocament i mètode

He enfocat el projecte des de la mateixa perspectiva en que s'enfoquen les assignatures en l'entorn educatiu de la UOC, és a dir, la obtenció de fites temporals que s'han anat lliurant a traves de les PAC's, que han permès a l'hora efectuar una planificació del projecte.

Pel que respecta al mètode utilitzat per aconseguir els objectius marcats en el projecte, ha estat l'estructura tradicional per a projectes informàtics. En primer lloc vaig recopilar la informació relativa al programari necessari per a l'execució del projecte que va consistir en obtenir a través de la web una còpia del següent programari, amb les següents tasques:

- Descàrrega i instal·lació del programari de control de versions *TortoiseCVS versió 1.8.25.*
- Descàrrega i compilació de les llibreries *DresdenOCl* per a la manipulació d'esquemes UML, del paquet *dresden-ocl2*.
- Descàrrega i instal·lació *d'Eclipse versió 3.1.2*.
- Descàrrega i instal·lació dels programes de tractament de models UML: *Poseidon versió 4.0 Comunity Edition i ArgoUML versió 0.20*.
- Obtenció i lectura de la documentació corresponent a l'especificació UML 1.5, que és la que s'ha utilitzat en aquest projecte per a especificar els models tractats.
- Obtenció i instal·lació del SDK[4] de la plataforma Java versió 1.5.0_06-b05
- Obtenció i instal·lació del servidor d'aplicacions Tomcat versió 5.5.17.

[4] Standrad edition Development Kit

Per a cadascun dels dos objectius principals, ja mencionats en l'apartat 1.2 d'aquesta memòria, he seguit, de manera iterativa per a cadascun dels productes obtinguts en el present projecte, el següent mètode:

- Obtenció de requeriments.
- Anàlisi i disseny de l'aplicació, mitjançant la confecció dels casos d'us, diagrama estàtic i altres diagrames que s'han considerat oportuns en funció de l'anàlisi
- Implementació de l'especificació obtinguda en l'etapa d'anàlisi.
- Proves i perfeccionament el resultats obtinguts.

1.4 Planificació

Podem veure el projecte des de dos perspectives diferents pel que fa referència a la seva planificació. Així per una banda tenim que el projecte s'ha dividit en diferents etapes conceptuals, en cadascuna de les quals s'han obtinguts diferents resultats, que són dependents funcionals, i des d'un altre òptica diferent poden veure el projecte com uns terminis de lliuraments temporals que han permès una organització temporal per portar a terme el projecte dins dels terminis establerts.

1.4.1 Planificació per etapes

El projecte a estat planificat en tres etapes, on en cadascuna d'elles s'han obtingut fites incrementals i complementàries de la globalitat del projecte. Les tres etapes són les següents:

- Etapa I: Creació i implementació de l'algoritme que comprova si dos diagrames son iguals, obtenint una llista de diferències en el cas que aquestes existeixin.

- Etapa II: Modificació de l'algoritme obtingut en l'etapa anterior perquè tingui en compte possibles diferències entre els dos esquemes, però que no afecten

al seu disseny, com poden ser els noms dels paquets utilitzats o la possibilitat d'incorporar sinònims als nom dels elements.

- Etapa III: Creació i implementació d'una aplicació web que utilitzi l'aplicació resultant de l'etapa anterior, que permeti la gestió de problemes per part del professorat i la resolució d'aquests problemes per part dels alumnes, obtenint una correcció on-line amb la indicació dels errors si s'escau.

1.4.2 Planificació per terminis de lliuraments

L'altre punt de vista de la planificació la trobem en el planificació temporal, tal com va quedar establert en el pla de treball, que en realitat va ser el primer document d'aquest projecte, en el qual es van establir unes determinades fites per tal de poder dosificar el treball a realitzar. Així les fites establertes han estat les següents:

- 03.03.2006: Lliurament del Pla de treball.
- 18.04.2006: Lliurament de la PAC2 que conté el disseny i implementació de l'algoritme que ens comprova si dos diagrames són iguals, tal com s'ha especificat en l'etapa I de l'apartat 1.4.1.
- 22.05.2006: Lliurament de la PAC3 en la qual es modifica el disseny de l'aplicació perquè permeti variacions que no afectin a l'estructura del disseny del model, i una aplicació pràctica de la implementació per a un entorn web, que corresponen a les etapes II i III, anteriorment ressenyats en l'apartat 1.4.1 .
- 19.06.2006: Lliurament del projecte complet.

1.5 Productes generats

Com a resultat de les tasques generades per a l'execució del present projecte, s'ha obtingut els següents productes:

- Una aplicació de lliure distribució, juntament amb el codi font, que permet la comparació de models UML, pel que fa referència a diagrames de classes, en llenguatge Java d'acord amb les versions indicades en l'apartat 1.3, que pot ser utilitzada per altres aplicacions com a llibreria interna pel tractament i comparació de diagrames de classes.

- Una aplicació web que demostra la utilitat de l'aplicació anteriorment obtinguda, on també s'inclou tot el codi font necessari per a la seva lliure distribució, executable sota l'entorn del servidor d'aplicacions Tomcat.

- La pròpia memòria que conté tota la informació relativa a la confecció del projecte.

- La presentació del projecte.

1.6 Estructura de la memòria

El cos del projecte els formen els capítols 3 a 6 , estant la memòria del projecte estructura de la forma que mes detalladament s'especifica a continuació:

Capítol 2: Primeres passes. Obtenció i instal·lació del programari.

En aquest capítol es descriu tot el programari que s'ha utilitzat per a portar a terme el projecte, així com la manera d'obtenir-los i instal·lar-los.

Capítol 3: UML. Diagrama estàtic

Descripció general de l'UML posant especial atenció al diagrama estàtic, que és l'objecte de principal atenció de l'especificació en el projecte.

Capítol 4: Comparació de diagrames. Disseny i implementació.

Requeriments, especificacions, diagrames, implementació i proves de l'aplicació de comparació de diagrames.

Capítol 5: Modificació i millores del disseny

Incorporació a l'aplicació obtinguda en el capítol 4 de les modificacions necessàries per permetre a l'aplicació que sigui capaç de tenir en compte quines són les diferències que no afecten a la seva igualtat semàntica.

Capítol 6: Aplicació pràctica: Utilització en entorn web

Requeriments, especificacions, diagrames, implementació i proves d'una aplicació web que demostri la possibilitat d'utilitzar el codi anteriorment obtingut.

Conclusions

Glossari

Referències

Capítol 2

Primeres passes. Obtenció i instal·lació del programari

En aquest capítol s'explicarà com he obtingut el programari necessari per a portar a terme el projecte, amb indicació de les versions i passes seguides per a la instal·lació del programari obtingut.

2.1 Relació de programari

La relació de programari utilitzada en aquesta pràctica ha estat la següent, amb la indicació de les versions utilitzades:

TortoiseCVS versió 1.8.25
Eclipse versió 3.1.2
Paquet *dresden-ocl2* de DresdenOcl
Poseidon versió 4.1 Comunity Edition
ArgoUML versió 0.20
Java versió 1.5.0_06-b05
Tomcat versió 5.5.17

A continuació passarem a descriure les passes a seguir per a obtenir i instal·lar cadascun d'aquests programes.

2.2 TortoiseCVS

TortoiseCVS és un programari de control de versions que es necessari per tal d'obtenir el codi font del paquet *dresden-ocl2* , que després de ser compilat, ens proporcionarà les llibreries necessàries per al tractament i manipulació d'esquemes UML i restriccions OCL.

L'adreça per efectuar la descàrrega del software és la següent:

http://prdownloads.sourceforge.net/tortoisecvs/TortoiseCVS-1.8.25.exe

Les instal·lació del programari queda detallat en les figures següents:

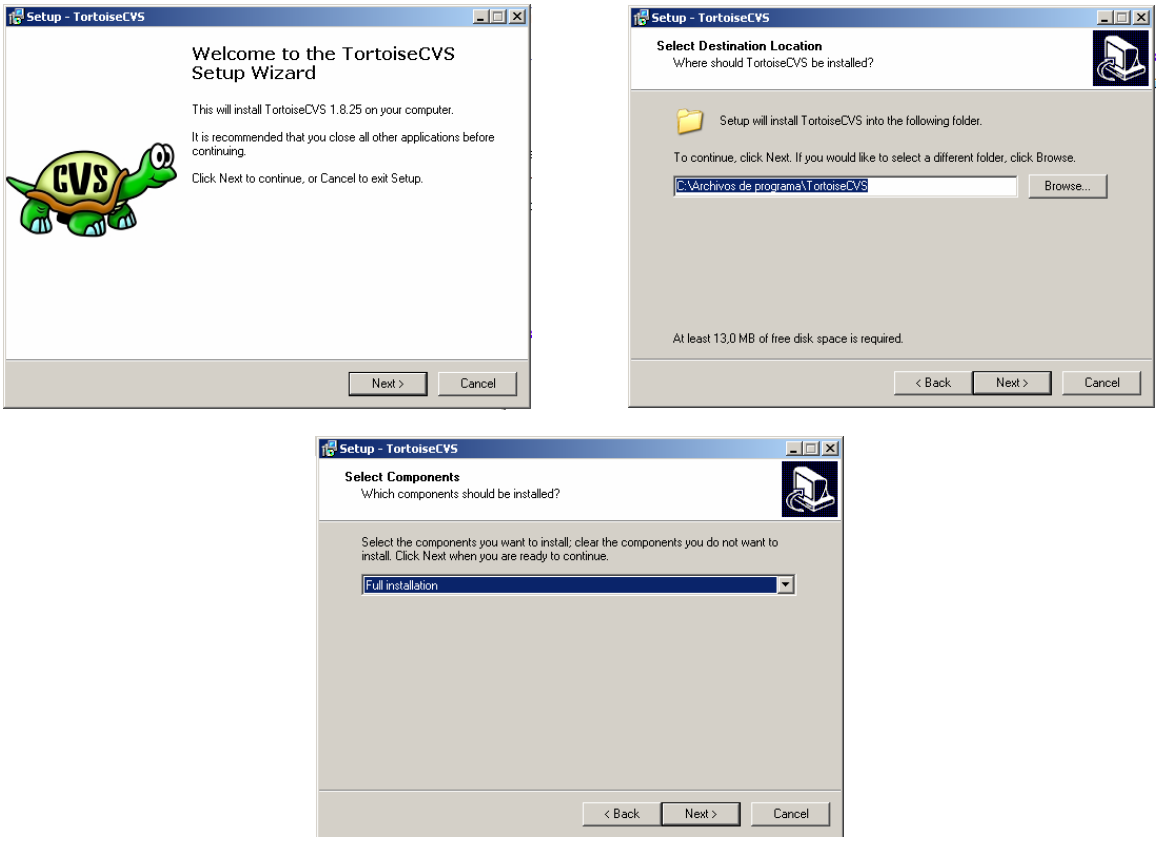


Figura 1. Pantalles d'instal·lació TortoiseCVS

El programari ha estat instal·lat amb totes les opcions per defecte. Una vegada instal·lat el programari es accessible des de l'explorador d'arxius tam com es mostra en la figura següent:

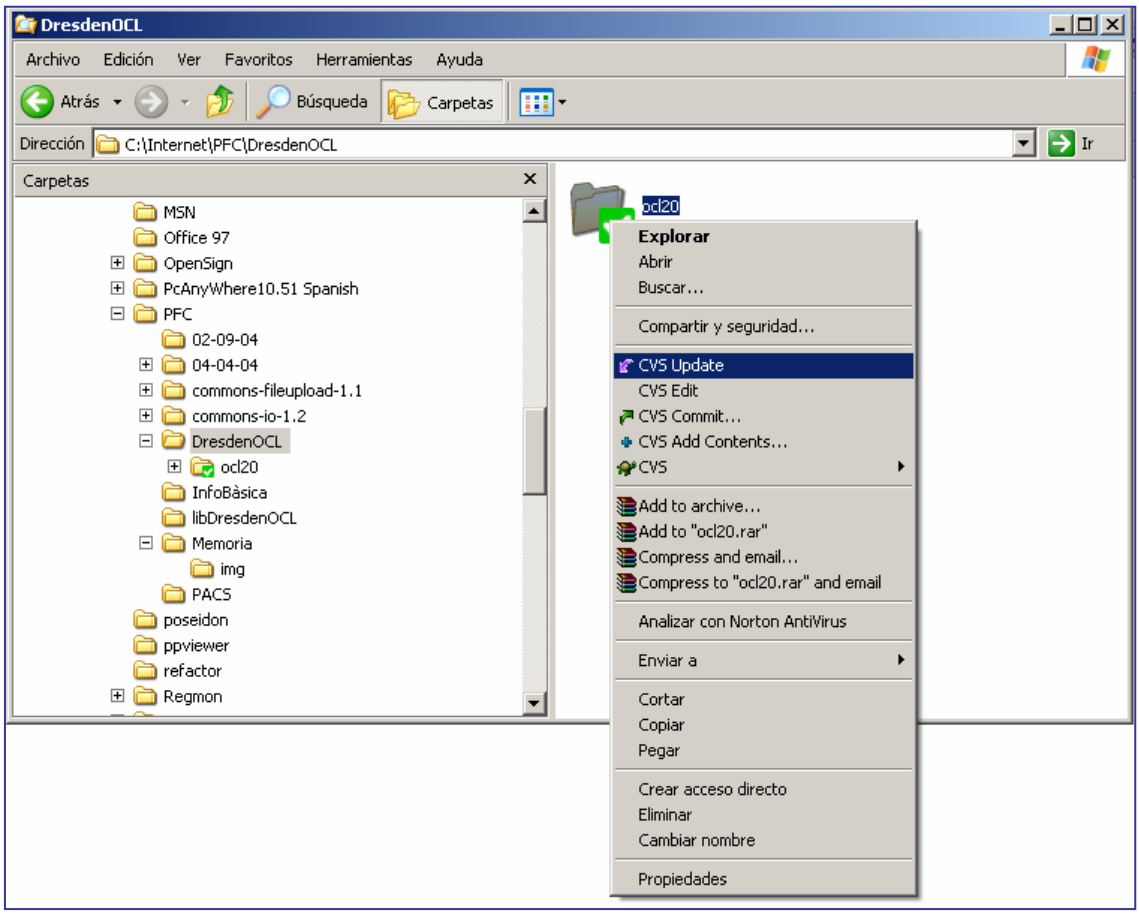


Figura 2. Exemple d'utilització de TortoiseCVS des de l'explorador d'arxius

2.3 Eclipse

Eclipse és l'IDE per a desenvolupament de programes Java, triat per a portar a terme les aplicacions desenvolupades en aquest projecte. La web del projecte la trobarem a http://www.eclipse.org , on podrem efectuar les descàrregues dels mòduls que ens interessin.

El mòduls utilitzats en el projecte han estat els següents:

- Eclipse SDK 3.1.2
- WebTools

La descàrrega s'ha efectuat en dos passos diferents. En el primer s'ha procedit a la descàrrega a disc del fitxer eclipse-SDK-3.1.2-win32.zip. La descompressió d'aquest fitxer a generat directament l'estructura de directoris d'Eclipse.

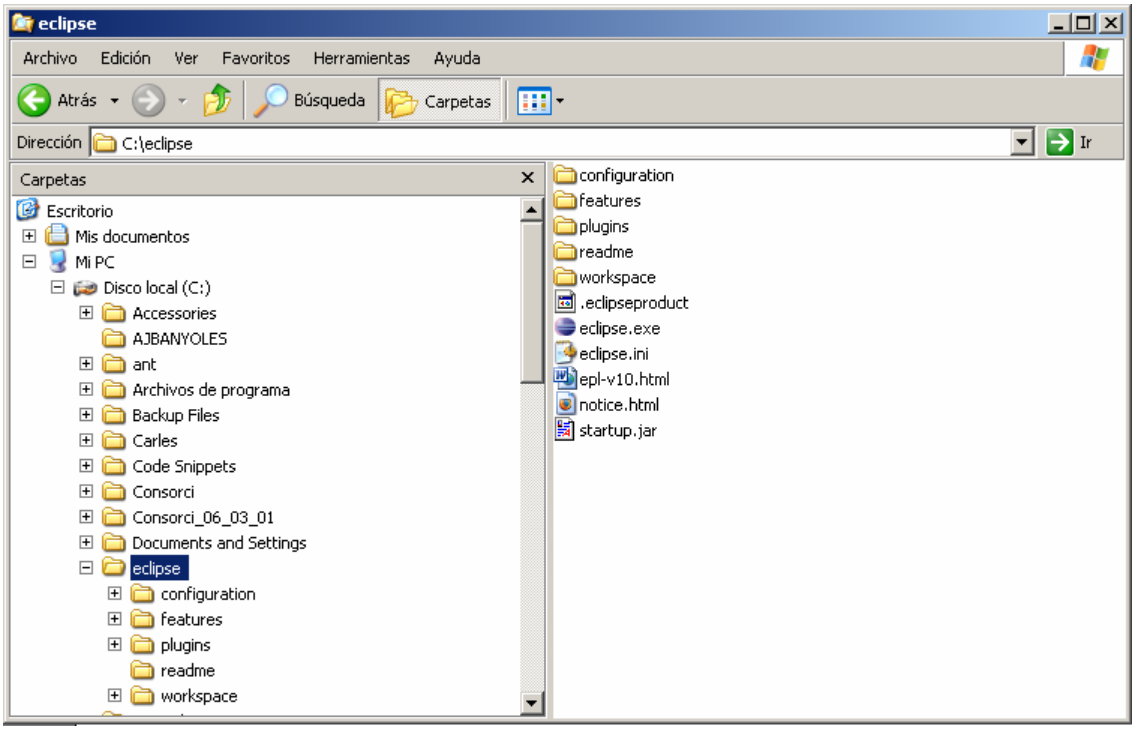


Figura 3. Estructura de directori d'Eclipse

Una vegada instal·lat l'Eclipse, per a obtenir els següents mòduls accedirem a l'opció de *Software Updates* del menú *Help*, on hi trobarem l'opció *Find and Install*, que com el seu nom indica ens permetrà buscar mòduls addicionals.

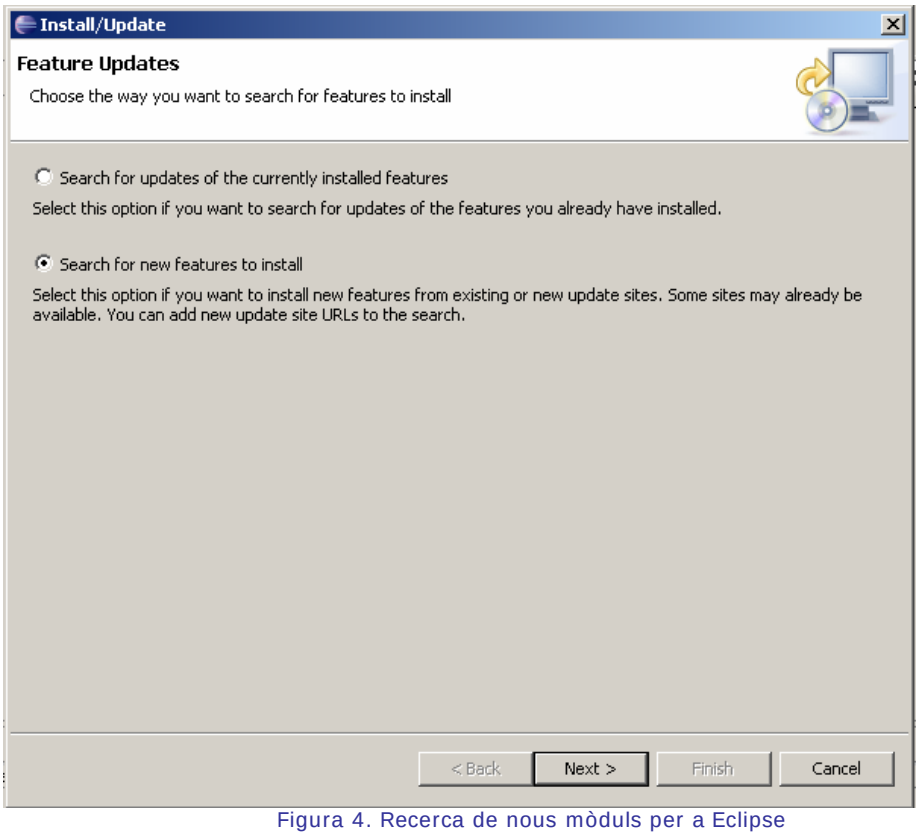


Figura 4. Recerca de nous mòduls per a Eclipse

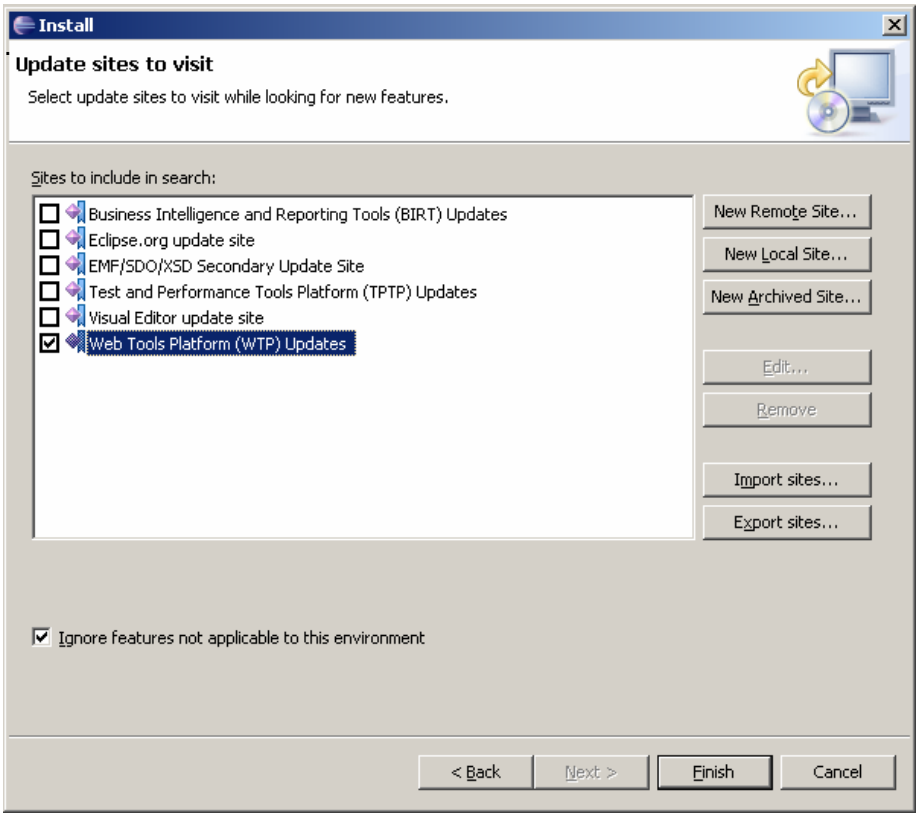


Figura 5. Selecció de la web per a obtenir els mòdul

Tal com s'indica a la figura 5, caldrà triar una web per a efectuar la descàrrega dels mòduls disponibles, per que amb posterioritat es pugui efectuar la instal·lació del mòdul tal com es pot observar en la figura 6.

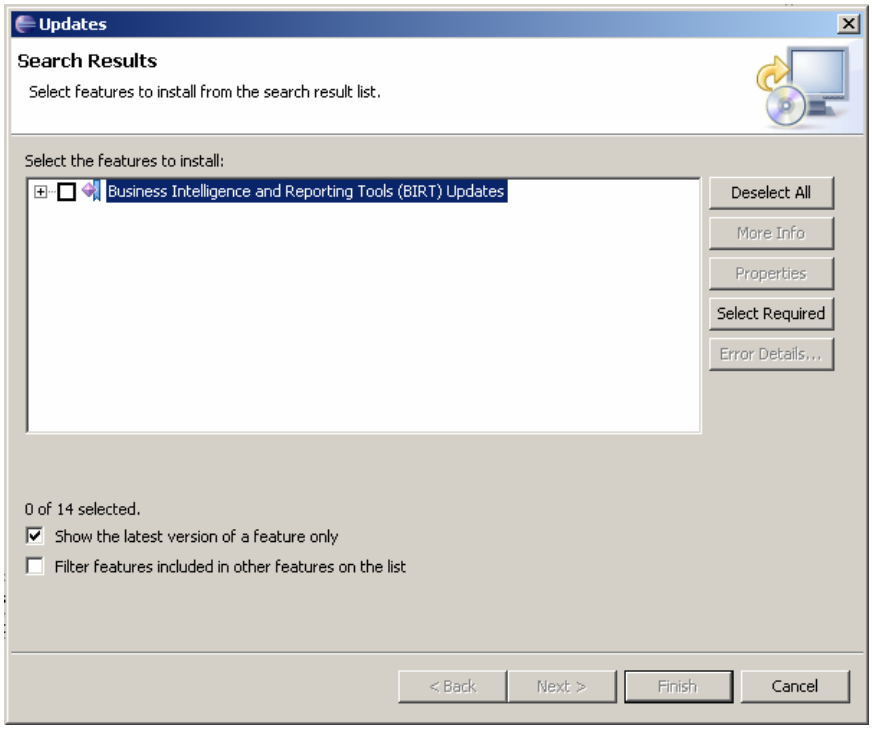


Figura 6. Instal·lació de mòduls

2.4 DresdenOcl

Tal com ja ha quedat dit, DresdenOcl és una eina per la manipulació d'esquemes UML i tractament de restriccions OCL, poden accedir a la web del projecte des de l'adreça http://dresden-ocl.sourceforge.net/ .

2.4.1 Compilació de l'eina

Per poder utilitzar aquestes llibreries primer descarregar-se el codi font i a continuació, compilar-lo. Per baixar-se el codi font el millor és instal·lar-se i utilitzar un controlador de versions, tal com el TortoiseCVS, detallat en l'apartat 2.2 .

Un cop situats a la carpeta on voleu descarregar el parser d'OCL, veureu que prement el botò dret sobre la carpeta us apareixen noves opcions en el menú contextual, entre elles el CVS checkout.

Aquesta opció us permet accedir a un CVS per tal de descarregar el seu contingut. La finestra que us apareixerà serà com la de la Figura 3.

Com a paràmetre CVSRoot heu d'indicar (veure la plana web de DresdenOCL): d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/dresden-ocl.

Com a nom del mòdul heu de posar: OCL20 (DresdenOCL té algun mòdul més, com per exemple, la versió antiga del Parser però no ens interessa). Els noms dels mòduls coincideixen amb els noms de les carpetes del CVS que es poden veure via web des de la plana de Sourceforge.

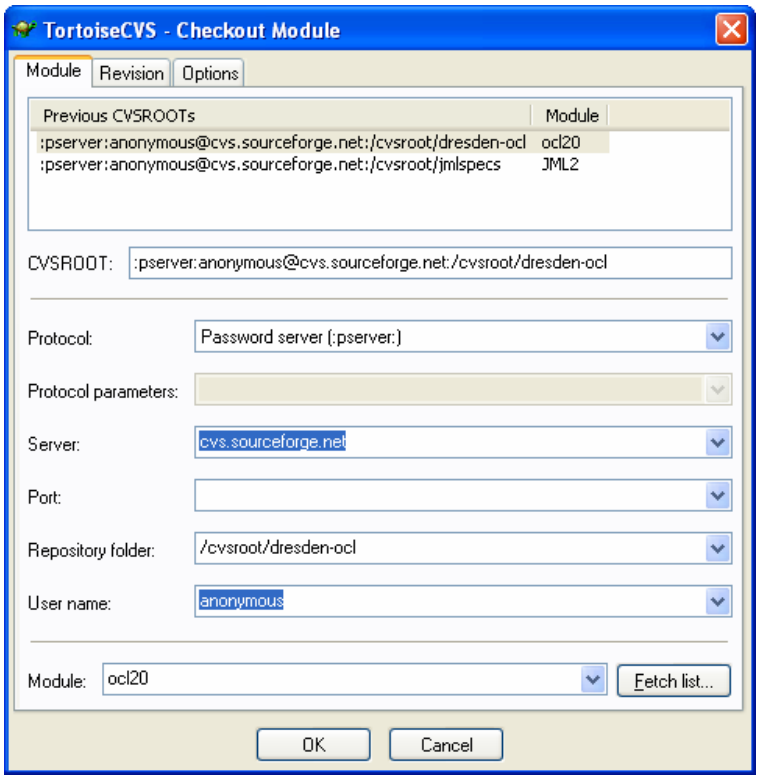


Figura 7. Utilitzacio TortoiseCVS per a a recuperar projectes

A continuació apareixerà la pantalla següent:

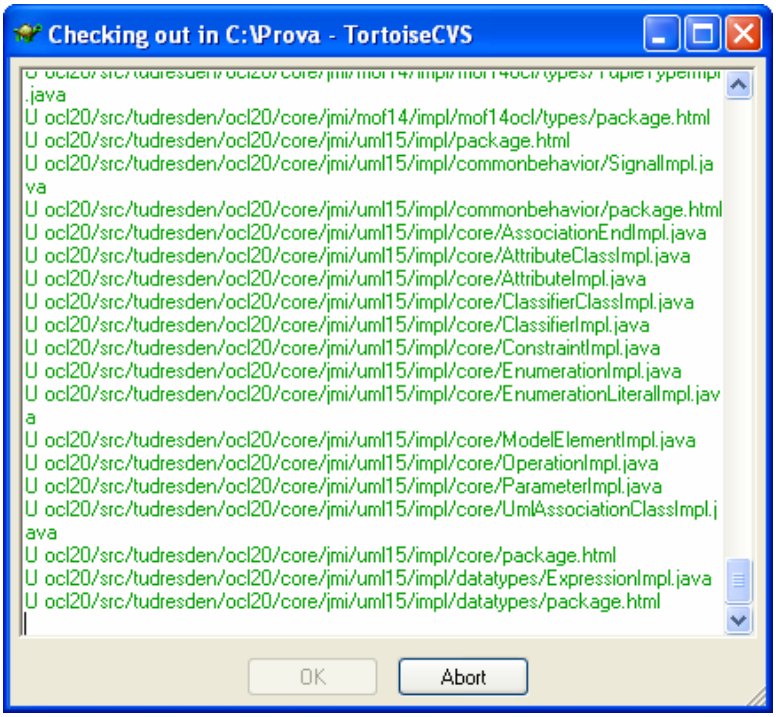


Figura 8. Missatges durant la recuperació de codi font

Un cop baixats els arxius veureu que a la carpeta anterior se us ha creat la subcarpeta OCL20 amb tots els subdirectoris corresponents.

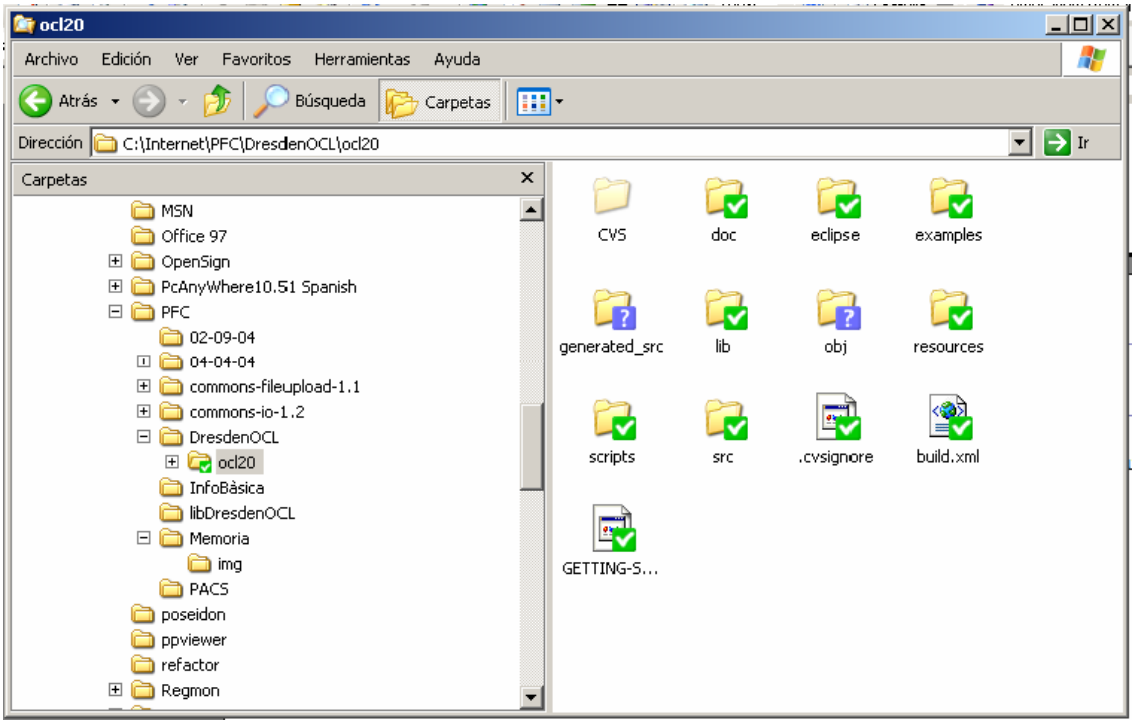


Figura 9- Directori sota el control de versions TortoiseCVS

A continuació, per compilar-la cal usar l'eina ant. Aquesta eina l'únic que fa és utilitzar un fitxer .xml (normalment build.xml) on hi ha tota la informació necessària per compilar un projecte complex (llibreries necessàries, aplicacions depenents...) i executar les instruccions corresponents.

Us podeu descarregar aquesta eina de http://ant.apache.org/. Per instal·lar aquesta eina cal configurar les següents variables del sistema:

set ANT_HOME=c:\ant
set PATH=%PATH%;%ANT_HOME%\bin
i si no la teniu ja configurada (degut a l'instal·lació del jdk) :
set JAVA_HOME=c:\jdk1.2.2 (jdk1.2.2 seria el directori on teniu el jdk)

A continuació ja només heu de compilar les dues aplicacions incloses dins el DresdenOCL toolkit fent:

ant run.workbench
i després
ant run.parser-gui

2.4.2 Utilització de les llibreries

El següent pas és crear una aplicació Java des d'un entorn de desenvolupament (es recomana, i molt, l'entorn Eclipse) que utilitzi les llibreries compilades a l'apartat anterior.

Cal fer els següents passos:

1.- Crear un nou projecte amb l'eclipse

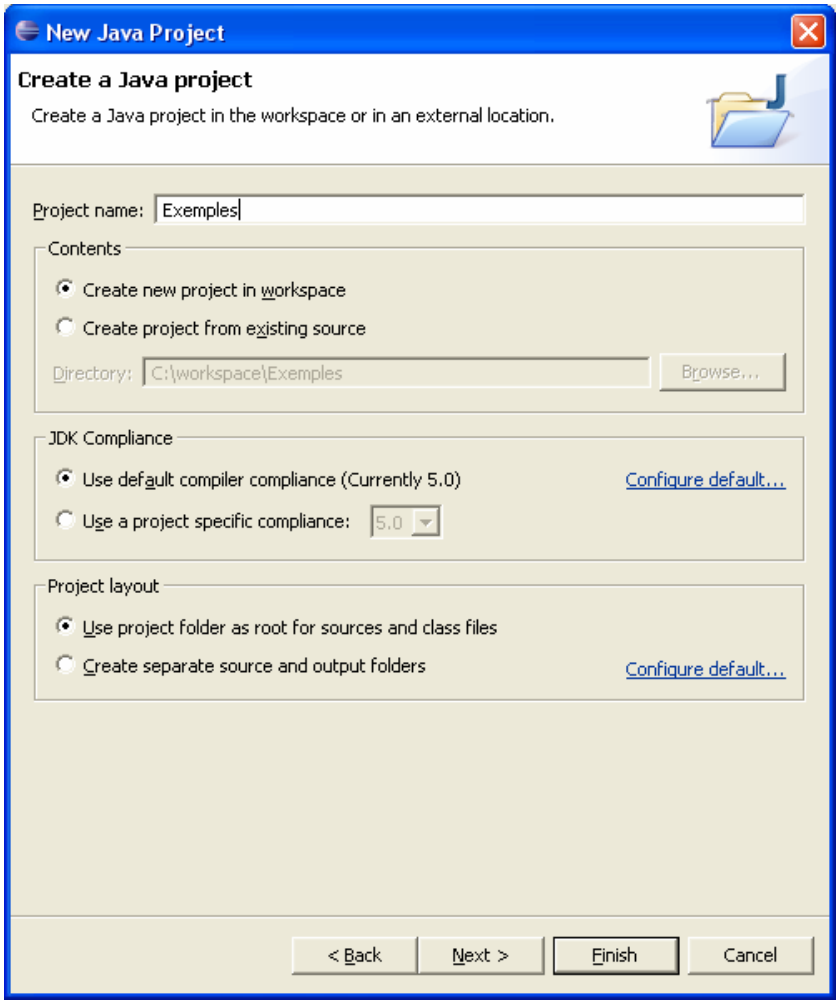


Figura 10- Creació d'un nou projecte Java a Eclipse

Afegir al nostre projecte les llibreries que de l'eina de dresden (veure el directori lib). Dins l'eclipse per fer això n'hi ha prou anant a les propietats del projecte i indicant que volem afegir com a llibreries externes les anteriorment esmentades. Una opció còmode és copiar tots els .jar que trobem en algun dels directoris de l'eina dins una nova carpeta i indicar des de l'eclipse que volem usar com a llibreries externes tot el contingut de la carpeta. També és necessari descarregar-se la llibreria jdom.jar (la podeu trobar dins el jdom-1.0.zip descarregable des de http://www.jdom.org/dist/binary/)

Per afegir les llibreries al projecte eclipse, botó dret sobre el projecte ->build path ->configure build path. Després dins l'apartat libraries seleccionar add external jars. El resultat hauria de ser semblant a:

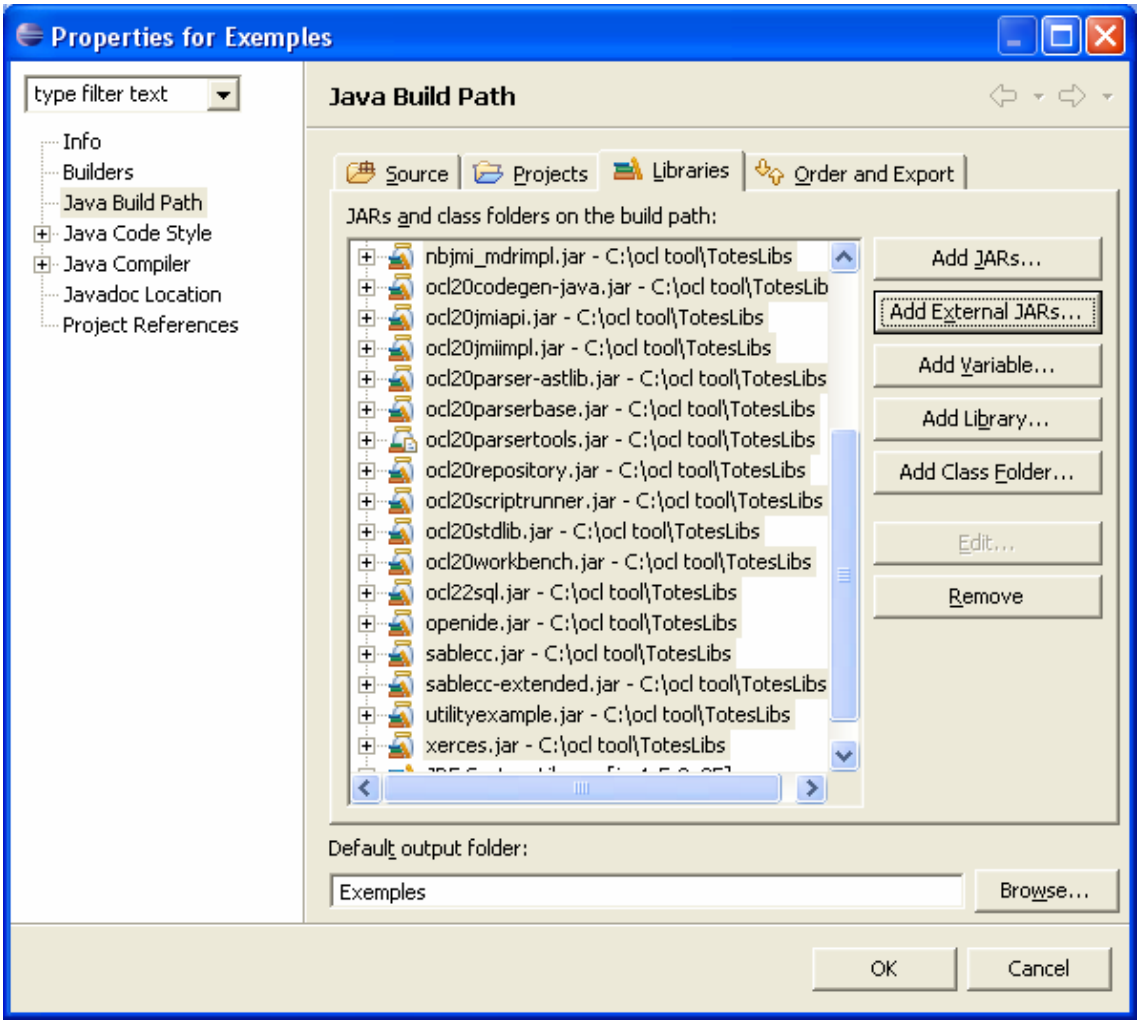


Figura 11. Propietats d'un projecte Java

Es copia al directori de treball del nostre projecte el fitxer UML15_plus_OCLMetamodel.xml que es troba en el directori /resources del DresdenOCLtoolkit.

En el mateix directori de treball s'hi creen les carpetes "repository" i "MetamodelsWithOcl" sense cap contingut.

Aquests dos últims passos tenen a veure amb el respository que creen les llibreries de DresdenOCL per tal de poder guardar la informació dels esquemes UML i OCL amb els que estem treballant (DresdenOCL utilitza les llibreries de NetBeans MDR per

implementar els metamodels de l'UML i de l'OCL, i són aquestes llibreries les que utilizen el repository).

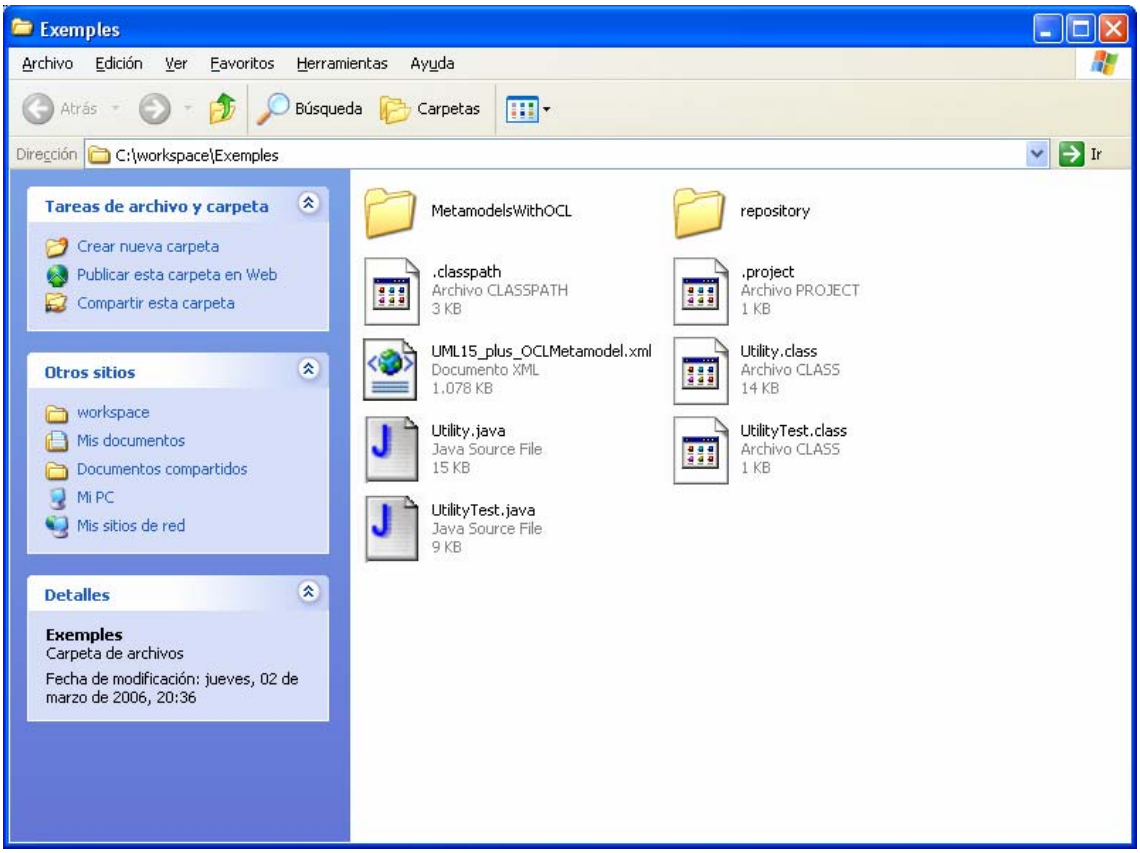


Figura 12. Repository d'un projecte Java a Eclipse

2.5 Poseidon

Poseidon és una eina per a generar diagrames UML, que es capaç d'exportar els diagrames a format estàndard XMI, que és un cas especial d'XML, i donat que encara que sigui un estàndard, cadascú l'implementa amb les seves pròpies personalitzacions, s'ha triat Poseidon com a generador d'XMI per la seva facilitat d'entendres amb les llibreries de Dresden Ocl2. La web de l'eina la podem trobar a http://www.gentleware.com .

Si anem a l'apartat de downloads, podrem observar que disposem de moltes versions per a descarregar, però pel que nosaltres ens interessa, es suficient amb descarregar l'edició Comunity Edition, doncs aquesta es completa i gratuïta per a ús no comercial, i en el nostre cas hem utiitzat la versió 4.1.1 .

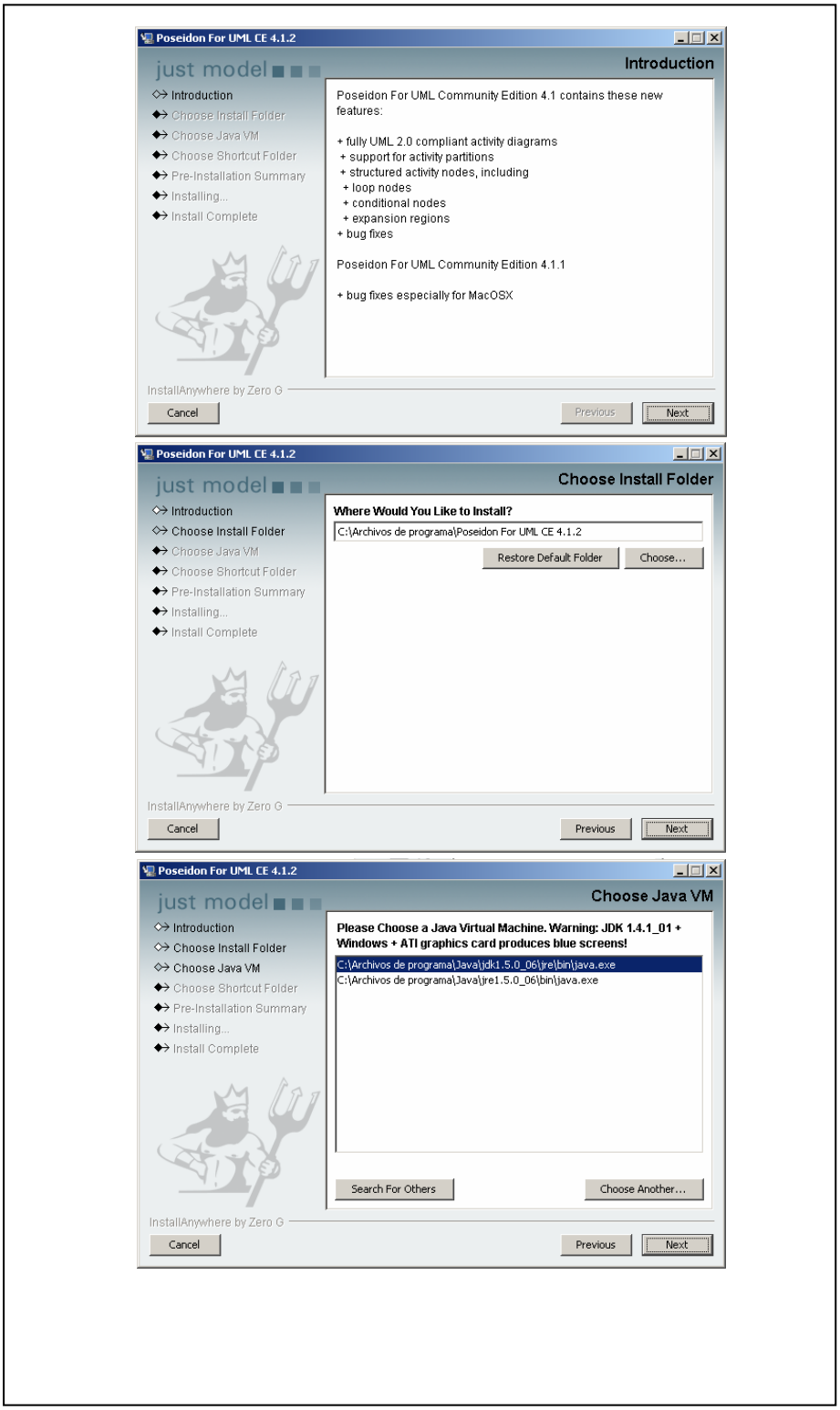


Figura 13. Instal·lacio Poseidon (part I)

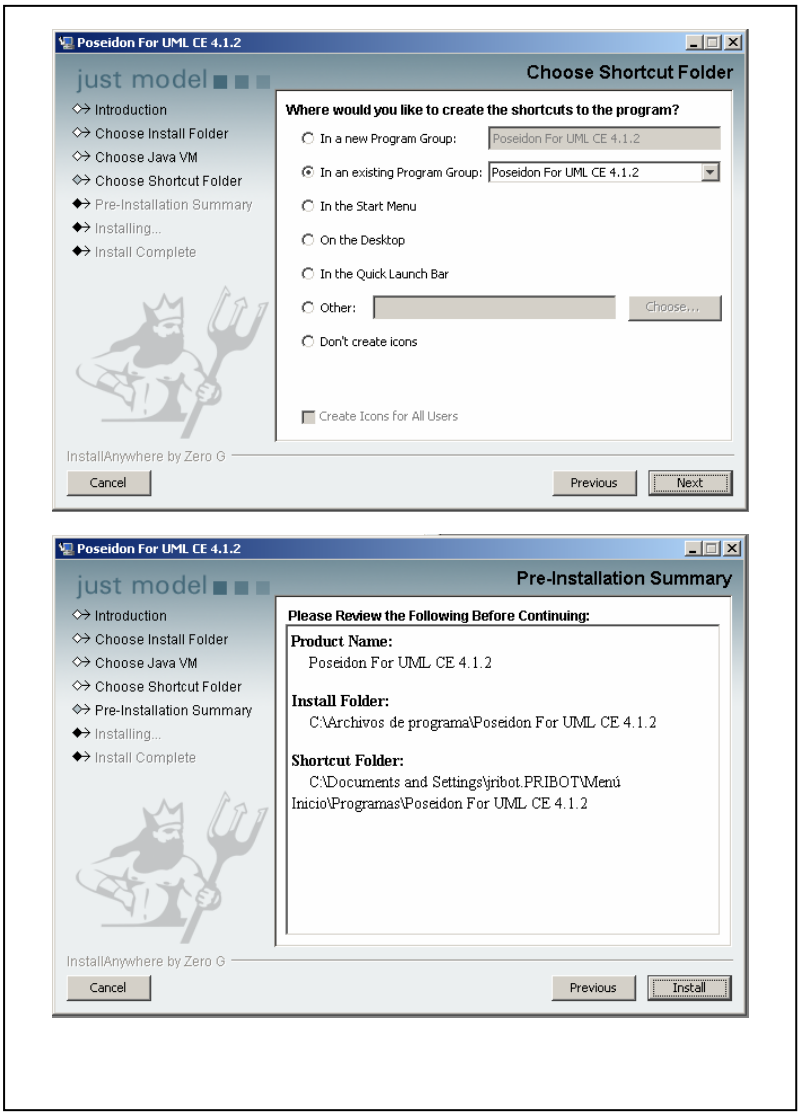


Figura 14. Instal·lacio Poseidon (part II)

2.6 ArgoUML

Durant el projecte vaig poder saber que hi havia programari de codi lliure que també tenia la possibilitat de realitzar exportacions de diagrames UML al format XMI, que és el format que reconeix les llibreries de DresdenOcl. El programari es equivalent a Poseidon i la seva web es http://argouml.tigris.org/ des del qual es pot descarregar una versió del programa. La versió utilitzada en el projecte a estat la 0.20.

Per a la seva instal·lació n'hi ha prou en descomprimir el fitxer descarregat en la carpeta d'instal·lació que es determini.

2.7 JAVA

El projecte ha utilitzat la versió de java 1.5.0_06-b05 obtinguda des la web http://java.sun.com/j2se/1.5.0/download.jsp .

Una vegada descarregat l'arxiu corresponent, caldrà executar el programa descarregat, acceptant tots els defectes que ens proposi el programa d'instal·lació.

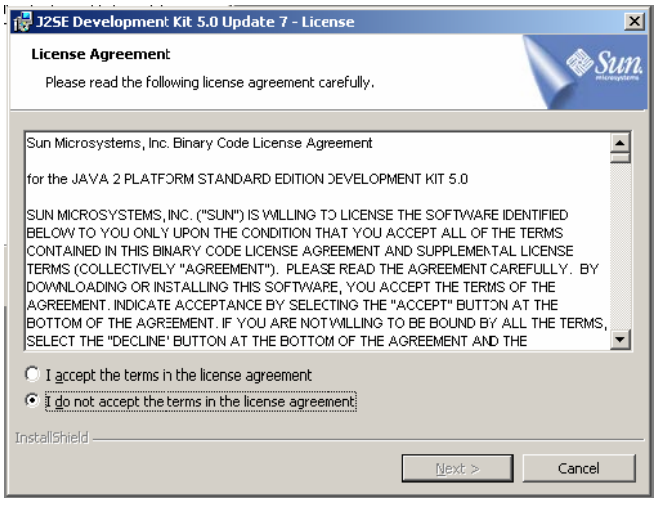


Figura 15. Instal·lacio Java (part I)

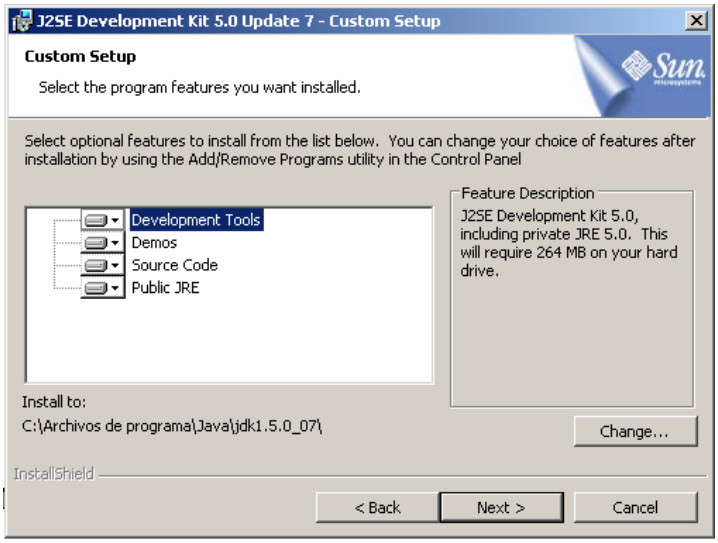


Figura 16. Instal·lacio Java (part II)

Per comprovar que la instal·lació s'ha efectuat de manera correcte, n'hi haurà prou en obri una finestra MS-DOS, i executar la següent comanda:

Java –version

Si la instal·lació s'ha efectuat correctament haurem de veure una pantalla semblant a la següent:

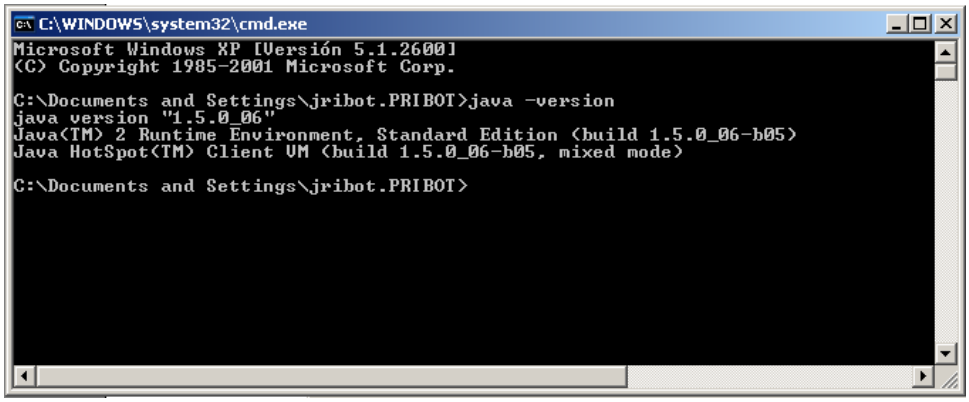


Figura 17. Comprovació instal·lació Java

2.8 Tomcat

Tomcat és un servidor d'aplicacions de programari lliure desenvolupat per Apache, que s'ha utilitzat per a allotjar-hi l'aplicació web que s'ha creat per tal de donar una aplicació a l'aplicació de comparació de diagrames.

La web on es pot trobar informació del programari es http://tomcat.apache.org des d'on ens podrem descarregar una versió del servidor d'aplicacions, que en aquest projecte ha estat la 5.5.17.

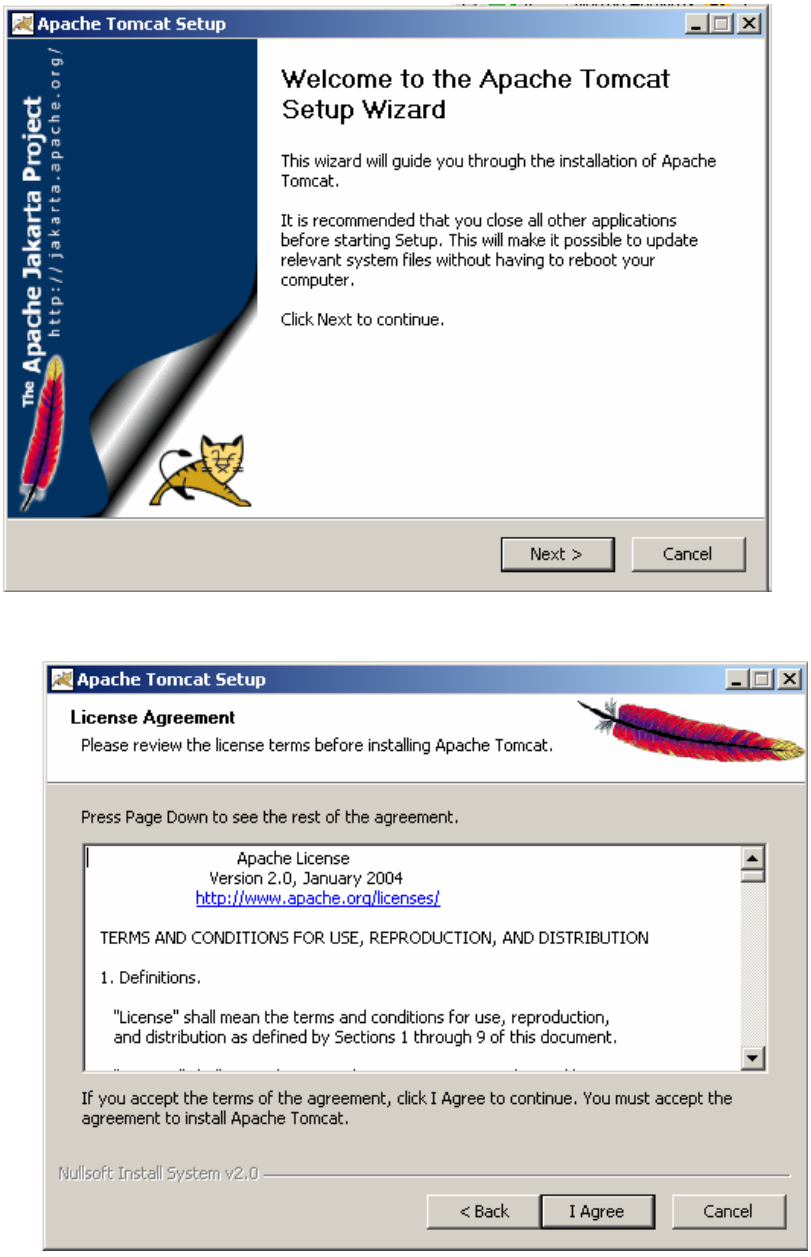
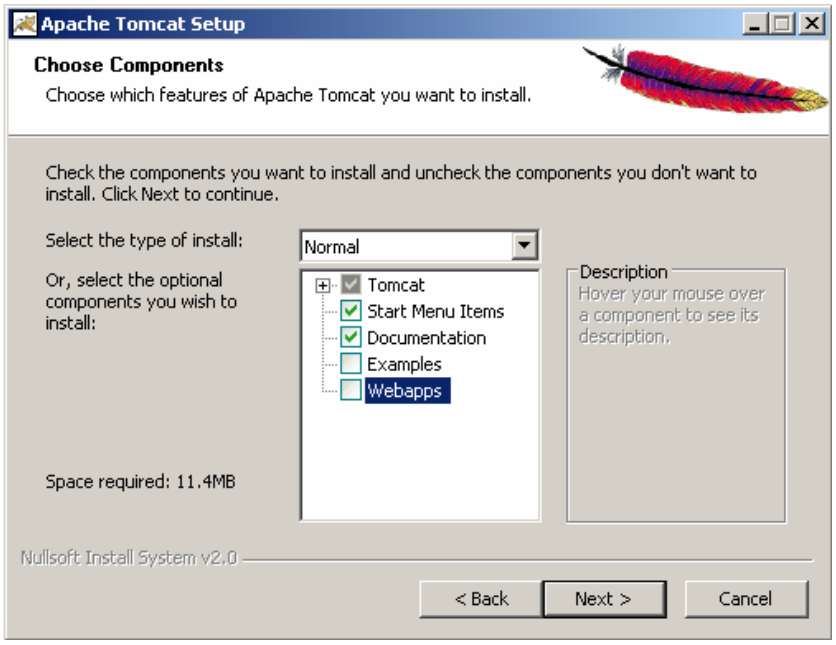


Figura 18. Pantalles d'instal·lació de Tomcat (part I)



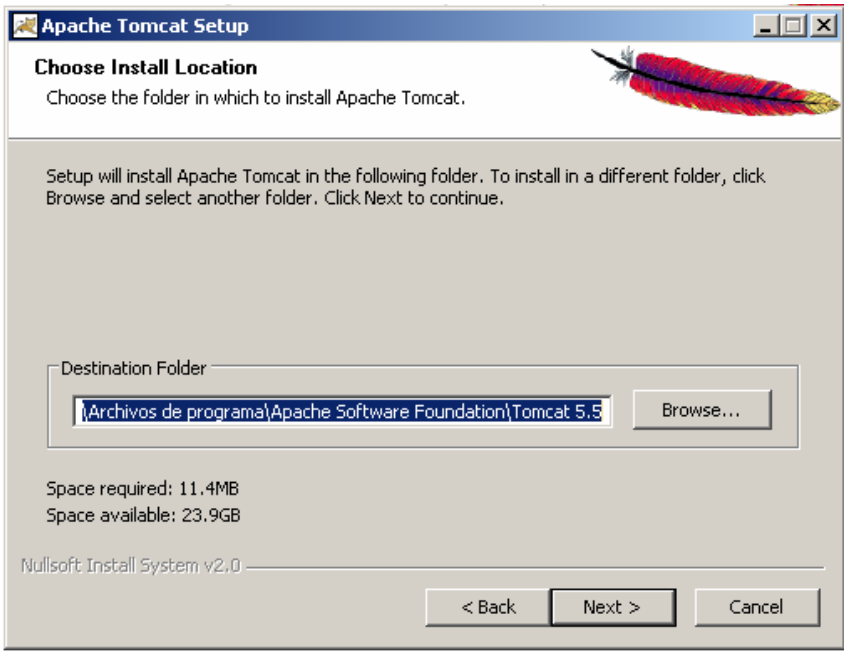


Figura 19. Pantalles d'instal·lació de Tomcat (part II)

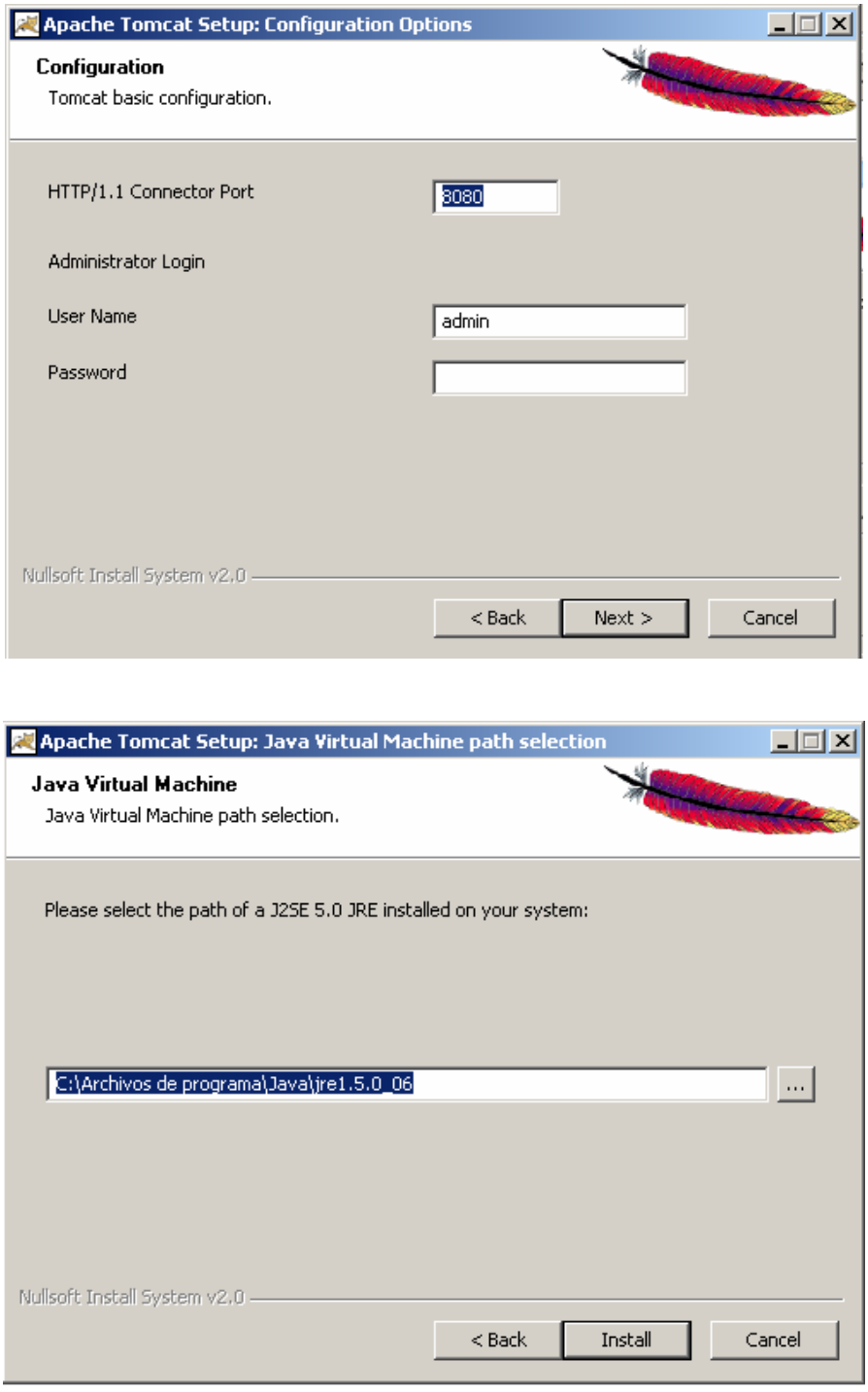


Figura 20. Pantalles d'instal·lació de Tomcat (part III)

Per arrancar el servidor d'aplicacions cal anar a la carpeta bin dins del directori que s'ha instal·lat el Tomcat, i executar el programa Tomcat5.exe, que inicia el servidor d'aplicacions com un servei.

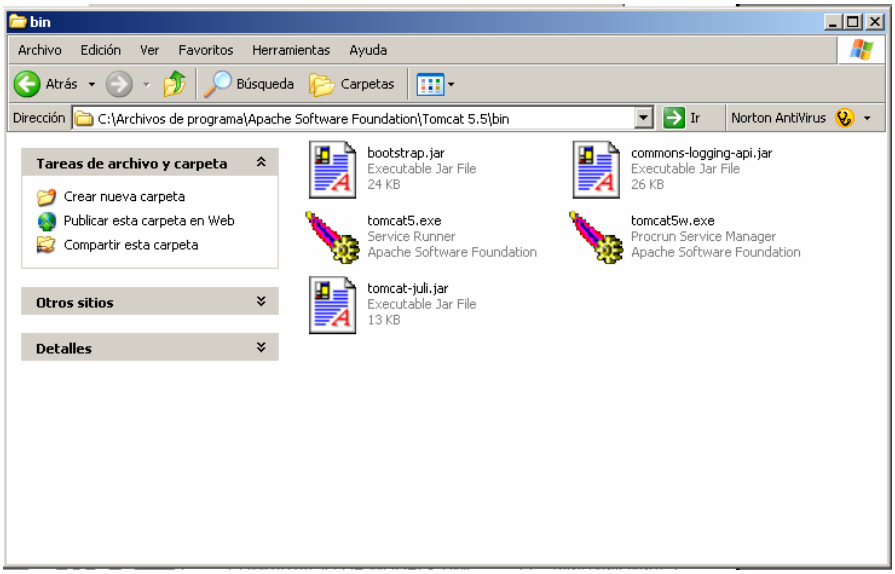


Figura 21. Directori de programes de Tomcat

Per comprovar que la instal·lació ha funcionat correctament, obrirem un navegador i escriurem la següent url http://localhost:8080, amb lo que hauriem d'obtenir la següent pantalla:

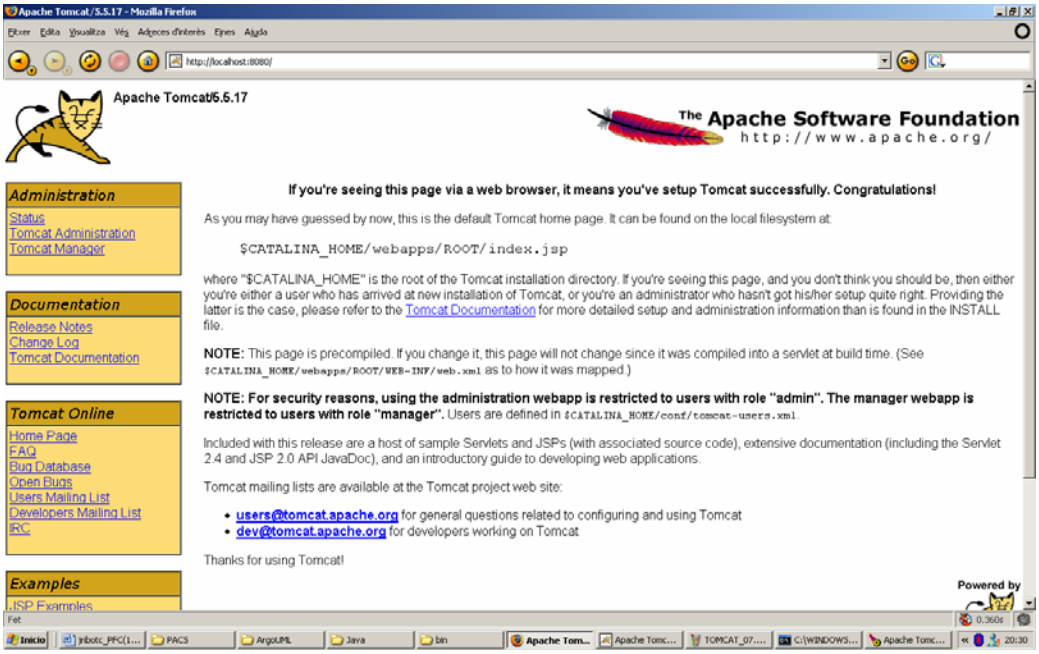


Figura 22. Pantalla inicial de Tomcat

Capítol 3

UML. Diagrama estàtic.

Les primeres coses que vaig tenir que fer per executar el projecte, va ser la comprensió del model que era objecte del tractament. Així a part d'una descripció general d'UML, vaig incidir en la comprensió dels elements que conformaven el diagrama estàtic, així com examinar el format d'intercanvi XMI. Això és el que veurem en aquest capítol.

3.1 Introducció a Unified Modeling Language

Unified Modeling Language (UML) és un llenguatge gràfic per especificar, visualitzar, construir i documentar els elements dels sistemes de software, igual que serveix per a modelar negocis i altres sistemes que no tenen res a veure amb el software.

UML neix entre el finals del 80 i el principi dels 90, quan Grady Boch, James Rumbaugh i Ivar Jacobson treballen per separat en el desenvolupament de notacions per a l'anàlisi i disseny de sistemes orientats a objectes. A mitjans dels noranta es van començar a intercanviar documents i a treballar en conjunt produint grans avanços en el modelatge de sistemes orientats a objectes. L'any 1994 l'empresa Rational, on hi treballava Booch, contracte Rumbaugh i un any més tard fa el mateix amb Jacobson, reunin un equip que l'any 1997 crea la versió 1.0 d'UML. Tot el treball d'aquest projecte està recolzat en la versió 1.5 de març del 2003, encara que a data d'avui ja existeix la versió 2.0 .

Tornant a la definició d'UML podem entendre la especificació com un conjunt d'eines. Si ens imaginem UML com una caixa d'eines on hi podríem trobar el martell, el tornavís, les alicates, etc. , en la caixa d'eines d'UML hi trobarem:

- Diagrama de casos d'us
- Diagrama de classes
- Diagrama d'estats
- Diagrama d'activitat
- Diagrama de seqüència
- Diagrama de col·laboració
- Diagrama de components
- Diagrama de distribució

Seguint amb l'analogia de la caixa d'eines, si volem penjar un quadre, segurament no utilitzarem totes les eines que tenim a la caixa, i potser agafant el martell en tindrem prou per clavar un clau on penjar-hi el quadre.

Exactament el mateix passa amb UML, així només utilitzarem aquells diagrames que ens permetin definir correctament la solució al problema plantejat, i no sempre els utilitzarem tots.

Que no és UML? UML no és un mètode de desenvolupament de software, i so serveix per passar de l'anàlisi al disseny i d'aquest al codi.

3.2 Objectius d'UML

Els objectius que es segueixen al redactar l'especificació els podem resumir en els següents apartats:

- Subministrar als usuaris un llenguatge gràfic per a modelar els seus desenvolupament amb la possibilitat d'intercanviar-los.

- Dotar de mecanismes d'extensibilitat i especialització que permetin ampliar els conceptes principals.
- Suportar especificacions que son independents del llenguatge de programació i dels processos de desenvolupament.
- Proporcionar unes formes bàsiques per entendre el llenguatge amb que es modela.
- Promoure el creixement en el mercat de les eines de modelatge.
- Suportar conceptes de desenvolupament d'alt nivell tals com components, col·laboracions, frameworks i patrons.
- Integrar les millors pràctiques.

3.3 Abast d'UML

En primer lloc, UML refon els mètodes de Boock, Rumbaugh i Jacobson . El resultat és un senzill, comú i àmpliament utilitzable llenguatge per a modelar per als usuaris d'aquests i altres mètodes.

En segon lloc, UML descriu el que es pot fer amb els mètodes existents. Per exemple els autor d'UML van apuntar modelar els sistemes concurrents, distribuïts per assegurar que UML tracta correctament aquests dominis.

En tercer lloc, UML es centra en estandarditzar un llenguatge de modelat, no en estandarditzar processos. Encara que UML pot ser aplicat en el context d'un procés, l'experiència diu que diferents organitzacions i dominis de problemes, requereixen diferents processos.

UML especifica un llenguatge de modelat que incorpora els conceptes d'orientació a objectes com a part principal dels conceptes de modelat. Es permeten que les desviacions siguin expressades en els seus propis mecanismes d'extensió.

En definitiva UML ens proporciona el següent:

- Semàntica i notació per expressar una àmplia varietat de situacions actuals d'una manera directa i econòmica.

- Semàntica per expressar certes situacions futures, específicament de components tecnològics, computació distribuïda i frameworks .
- Mecanismes d'extensió per a projectes individuals que poden estendre el metamodel per a les seves aplicacions a baix cost. No s'espera que els usuaris canviïn directament el metamodel d'UML.
- Semàntica per a facilitar l'intercanvi de models a través de diverses eines

3.4 Característiques d'UML

Donat que els autors d'UML de fet el que dissenyen es un llenguatge (un alfabet gràfic), van posar cura en trobar un equilibri entre el minimalisme, on tot són caixes i textos, i una sobre-enginyeria, on es disposaria d'una icona per a cada element concebible. Al final, els autors van tenir molta cura en afegir nous conceptes per tal de no crear un UML innecessariament complex.

Alguns dels nous conceptes inclosos en l'UML, són:

- Mecanismes d'extensibilitat (estereotips, valors etiquetats i restriccions)
- Threads i processos
- Distribució i concurrència
- Interfícies i components
- Llenguatge de restriccions

Moltes d'aquestes idees eren present en diversos mètodes individuals i teories, però va ser UML que les va conjuntar d'una manera coherent. Els desenvolupadors d'UML no es van inventar moltes d'aquestes idees, senzillament van seleccionar i integrar les millors idees del modelat d'objectes i les pràctiques de la ciència informàtica. Així per exemple:

- Els diagrames de casos d'us son similars en aparença als mateixos diagrames del mètode OOSE[5].
- El diagrames de classe provenen d'OMT[6] .

[5] Object Oriented Software Engineering
[6] Object Modeling Technique

- Els diagrames d'estat estan substancialment basats en els diagrames d'estat de David Harel amb petites modificacions. A aquest i altres diagrames se'ls va afegir el concepte de estereotips, restriccions i valors tabulats.
- Els diagrames de seqüència van refondre varies mètodes amb noms com interacció, traça de missatges o traça d'events.
- Els diagrames de col·laboració van ser adaptats a partir de conceptes desenvolupats per Booch (diagrama d'objectes) i Fusion (gràfic d'interacció d'objectes).

3.5 Concepte de diagrama estàtic.

Tal com hem vist en els anteriors apartats d'aquest capítol, UML està format per un bon nombre de diagrames, que donen la possibilitat a l'analista a expressar el seu disseny des de diferents punts de vista, i permetre així, abrasar la totalitat del projecte.

No és missió d'aquest projecte un estudi sobre UML, per tant ens centrarem en aquells aspectes de l'especificació que són d'interès pel projecte, que en aquest cas és el diagrama estàtic.

El diagrama estàtic, tal com el seu nom indica, mostra l'estructura estàtica del model, en particular, els elements que existeixen, com classes i tipus, la seva estructura interna i les seves relacions amb altres elements. No trobarem en el diagrama estàtic cap tipus d'informació temporal, encara que poden contenir ocurrències referides a elements que tenen o de elements que descriuen comportaments temporals.

És habitual parlar del diagrama estàtic com el diagrama de classes, de fet durant el projecte ens referirem amb ambdós noms, de manera indistinta. Retrobant la definició del diagrama de classes, aquest serà la representació gràfica del les classes amb la següent informació:

- Components de les classes: atributs i operacions
- Relacions entre les classes: associacions i generalitzacions.

3.6 Components del diagrama estàtic.

Per entendre el diagrama estàtic passarem a enumerar, adjuntant-hi una breu explicació, de cadascun dels elements que conformen el diagrama estàtic. Tots aquest components els podrem veure en els apartats següents

3.6.1 CLASSIFIER.

És la superclasse de CLASSE, TIPUS_DE_DADES i INTERFÍCIE. Totes elles tenen una sintaxi similar i per tant han de ser dibuixats d'una manera semblant, que és utilitzant un rectangle utilitzant paraules clau quan sigui necessari. Donat que les classes són molt comunes en els diagrames, un rectangle sense cap paraula clau representa una CLASSE, i la resta de subclasses de CLASSIFIER són indicades amb paraules clau.

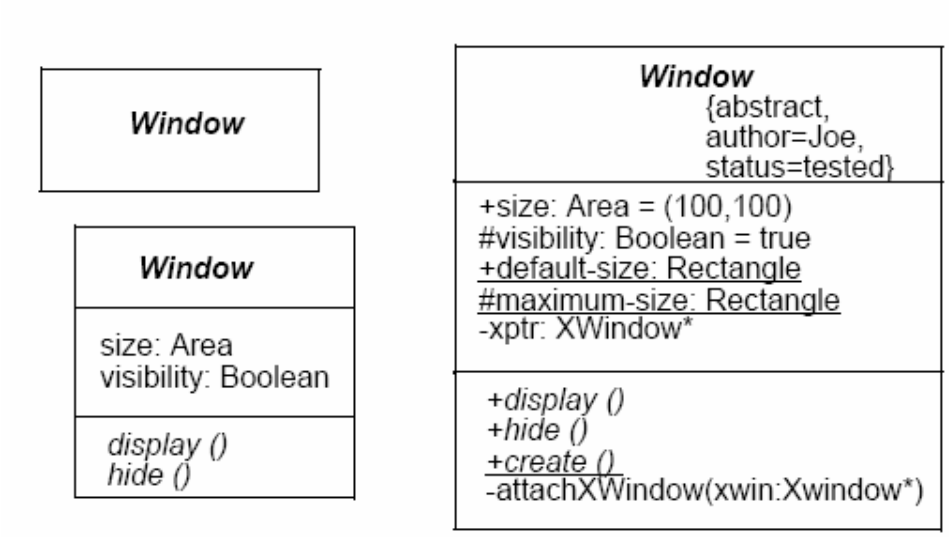
3.6.2 CLASSE.

Una CLASSE es el descriptor per a un conjunt d'objectes amb similar estructura, comportament i relacions. Així una classe representa un concepte dins del sistema que s'està modelant, contenint en elles l'estructura de dades, comportament i relacions amb altres elements.

El nom de la classe de un abast dins del paquet en que es declara i el nom de la classe ha de ser únic dins del paquet que conté la classe.

La seva representació gràfica dins del diagrama és amb un rectangle amb una línia de traç seguit, amb tres compartiments separats per línies horitzontals. En el compartiment superior hi posarem el nom de la classe i altres propietats generals de la classe, en el compartiment intermedi hi trobarem una llista dels atributs, i en el compartiment inferior s'hi col·locaran la llista d'operacions de la classe, que determinen el seu comportament.

Alguns exemples de representació de classes les podem veure en el gràfic següent:



Figura

Figura 23. Exemples de classes

3.6.3 ATRIBUT.

Per a representar un atribut utilitzarem el compartiment intermedi, utilitzant *strings* per a representar els atributs a la classe. Per representar un atribut utilitzarem la següent sintàxi:

visibilitat nom : tipus [multiplicitat ordenació] = valor_inicial {valors possibles}

els valors que pot prendre *visibilitat* són:

+ atribut públic

atribut protegit

- atribut privat

~ atribut visible en el paquet

La visibilitat també pot ser indicada mitjançant paraules clau: public, protected, private i package.

nom és un identificador que representa el nom de l'atribut

multiplicitat mostra la multiplicitat de l'atribut. El terme pot ser omès, i en aquest cas la multiplicitat es 1..1 (exactament 1).

ordenació és una propietat que es significativa només si la multiplicitat es més gran que 1 i pot tenir els següents valors:

- (absent) : els valors estan desordenats
- unordered : els valors estan desordenats
- ordered : els valors estan ordenats

tipus indica el tipus de dada de l'atribut

valor_inicial és el valor que pren inicialment l'atribut, essent opcional la seva existència

valors_possibles són els valors que pot prendre l'atribut.

3.6.4 OPERACIÓ.

Una OPERACIÓ és un servei que una instància d'una classe pot sol·licitar que s'executi. Cada operació té un nom i una llista d'arguments, estan representades gràficament en el compartiment inferior de les CLASSES. Per representar una operació utilitzarem la següent sintaxi:

Visibilitat nom (llista-de-paràmetres) : tipus-de-retorn {valors possibles}

els valors que pot prendre *visibilitat* són:

+ atribut públic
# atribut protegit
- atribut privat
~ atribut visible en el paquet

La visibilitat també pot ser indicada mitjançant paraules clau: public, protected, private i package.

nom és un identificador que representa el nom de l'operació.

tipus-de-retorn és el tipus de dada que l'operació retornarà després d'executar-se.

llista-de-paràmetres és una llista de paràmetres separada per comes, on cada paràmetre està especificat segons la següent sintaxi:

tipus nom : tipus-de-dada = valor-de-defecte

tipus ens dirà si el paràmetre és d'entrada (in), de sortida (out) o d'entrada-sortida (inout). En cas que no aparegui aquest qualificador, el paràmetre serà considerat de només entrada.
nom és un identificador que representa el nom del paràmetre.
tipus-de-dada ens indica el tipus de paràmetre.
valor-de-defecte és el valor de defecte que prendrà el paràmetre.

3.6.5 INTERFÍCIE.

Una INTERFÍCIE és una subclasse CLASSIFIER i es representa com una CLASSE però indicant la paraula clau Interfície i recollint totes les operacions en el mateix compartiment que les classes. El compartiment corresponent als atributs por ser omès, doncs una interfície només pot tenir declarades les operacions i no pot tenir cap atribut.

La sintaxi ha seguir és exactament la mateixa que la especificada per al cas de la CLASSE.

3.6.6 ASSOCIACIÓ.

Una associació és una relació entre dos elements que derivin de CLASSIFIER. Gràficament, una associació es representa amb una línia que uneix ambdós símbols i les línies poden anar acompanyades d'altres elements que mostrin les propietats de les associacions.

3.6.7 ASSOCIACIÓ BINARIA.

Una associació binaria és una associació entre exactament dos CLASSIFIER's, incloent la possibilitat d'una associació entre un CLASSIFIER i ell mateix.

L'associació binària és representa amb una línia sòlida entre dos símbols CLASSIFIER. El final d'una associació a la que està connectada un CLASSIFIER s'anomena ASSOCIATION END, on es troba molta de la informació relativa a les associacions.

Una associació pot estar complementada amb els següents elements:

- El nom de l'associació, que pot anar acompanyat per un triangle negre que indiqui la direcció en que s'ha de llegir l'associació.
- Un símbol de CLASSE que qualifica l'associació, on s'hi poden determinar propietats tals com atributs, operacions i altres associacions.

La figura següent ens mostra exemple d'especificació d'associacions.

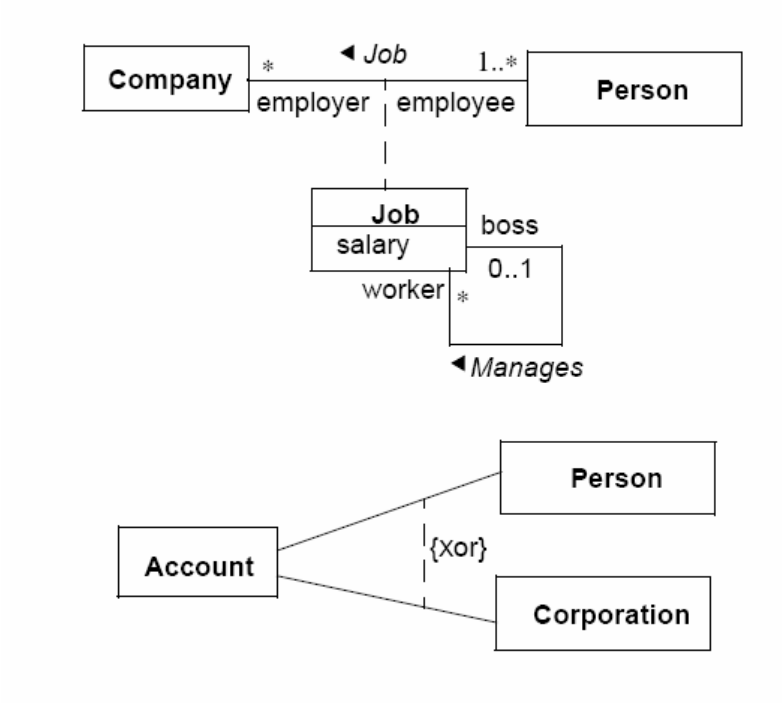


Figura 24. Exemples d'associacions

3.6.8 ASSOCIATION END.

Un ASSOCIATION END és simplement l'extrem d'una associació, on és connecta amb un CLASSIFIER. Cal destacar que aquest element és part de l'associació i no del CLASSIFIER. Cada ASSOCIACIÓ té dos o més finals, on es guarda part de la informació de l'ASSOCIACIÓ i no es pot entendre com un element separable de l'associació.

Una associació pot ser qualificada pels següents elements:

- Mulitplicitat: S'especifica únicament com a text.
- Ordenació: SI la mulitplicitat és més gran que 1 els elements poden estar ordenats o desordenats. Si no es fa cap indicació respecte a la ordenació, aplicarem el valor per defecte que és desordenada, per a que estigui ordenada s'ha d'especificar explícitament.
- Navegabilitat: Per a indicar-ho col·locarem una fletxa al final de la línia, encara que els autors recomanen utilitzar només aquesta indicació només en casos excepcionals.
- Agregació: Col·locarem un romb per a indicar agregació. Si el romb està pintat de negre en indicarà un cas especial d'associació com és composició.
- Rolename: És un string a prop del final de la línia, que indica el paper que juga la classe.
- Visibilitat: Segueix els criteris expressats en altres descripcions ja exposades, o sigui, que pot prendre els valors de públic, privat o protegit.

En la figura següent podem observar gràficament les opcions anteriorment exposades:

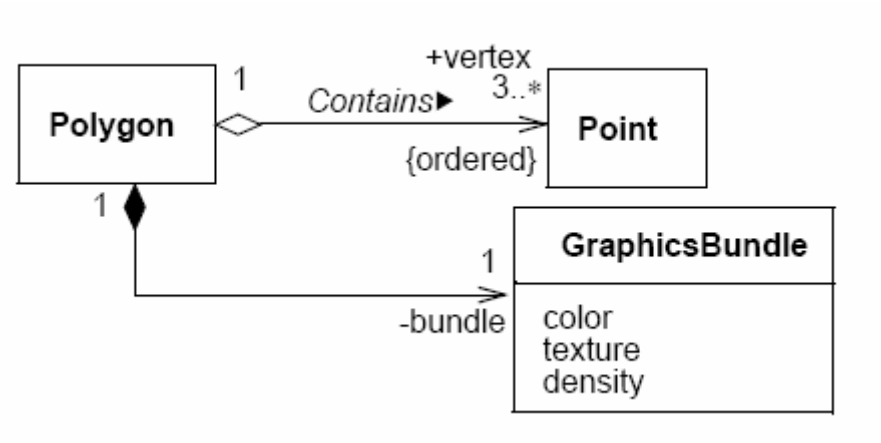


Figura 25. Exemples d'association end

3.6.9 AASSOCIATION CLASS.

Una ASSOCIATION CLASS és una associació que conté propietats de CLASSE. La seva representació gràfica es fa mitjançant un rectangle (símbol de CLASSE) unit a una línia d'associació mitjançant una línia discontinua. Donat que és una associació i una classe es comportarà com a tals.

En la figura següent podem veure la seca representació:

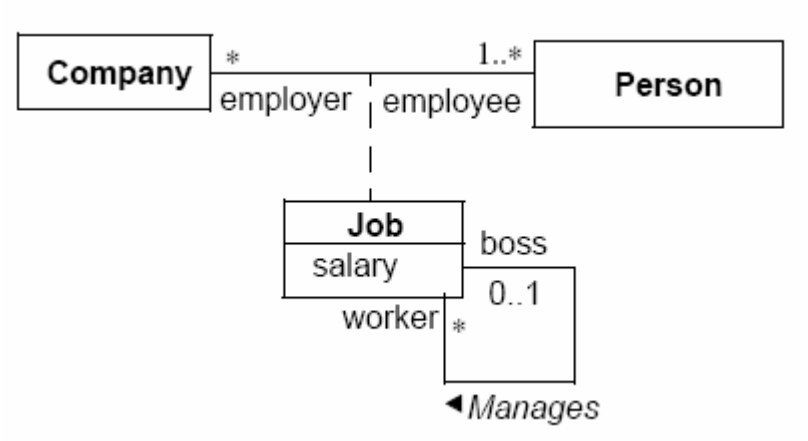


Figura 26. Exemples d'association class

3.7 Model d'intercanvi.

EL model d'intercanvi a UML està basat en la especificació 1.3 de MOF[7]. El elements UML son equivalents a un conjunt de paquets MOF anomenats UML Interchange Metamodel. El paquet està disponible com un document XML anomenat UML_1.4_Interchange_Metamodel.xml, que està basat en MOF 1.3 i XML Metadata Interchange (XMI).

Els models UML poden ser intercanviats entre eines de software com a fitxers amb un format XML estàndard. Existeix un fitxer anomenat UML_1.4_XMI.dtd que ha estat generat desde UML Interchange Metamodel seguint les regles del XML Metadata Interchange.

Sabent que l'explicació no és senzilla, el millor serà veure un exemple, on hi podem veure el seu grafisme i la seva representació corresponent:

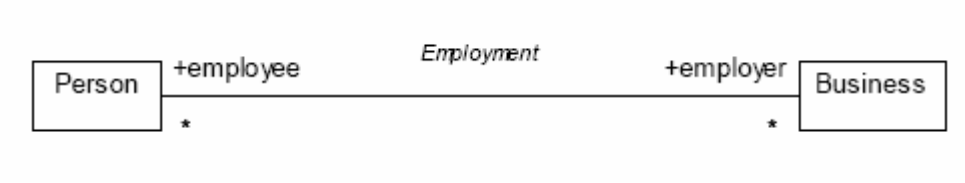


Figura 27. Exemple de diagrama de classes

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE XMI SYSTEM 'UML_1.4_XMI_1.1.dtd'>
<XMI xmi.version='1.2' xmlns:UML='omg.org/UML/1.4'>
<XMI.header>
    <XMI.metamodel xmi.name='UML' xmi.version='1.4'/>
</XMI.header>
<XMI.content>
    <UML:Model xmi.id='S.1' name='Employment Model' visibility='public'
    isSpecification='false' isRoot='false' isLeaf='false' isAbstract='false'>
    <UML:Namespace.ownedElement>
    <UML:Class xmi.id='S.2' name='Person' visibility='public' isSpecification='false'
    namespace='S.1' isRoot='true' isLeaf='true' isAbstract='false' isActive='false'/>
    <UML:Class xmi.id='S.3' name='Business' visibility='public' isSpecification='false'
    namespace='S.1' isRoot='true' isLeaf='true' isAbstract='false' isActive='false'/>
    <UML:Association xmi.id='G.1' name='Employment' visibility='public'
    isSpecification='false' isRoot='false' isLeaf='false' isAbstract='false'>
    <UML:Association.connection>
    <UML:AssociationEnd name='employer' visibility='public' isSpecification='false'
    isNavigable='true' ordering='unordered' aggregation='none' targetScope='instance'
```

[7] Metaobject facility

```
      changeability='changeable' participant='S.3' association='G.1'>
      <UML:AssociationEnd.multiplicity>
      <UML:Multiplicity>
      <UML:Multiplicity.range>
      <UML:MultiplicityRange lower='0' upper='-1'/>
      </UML:Multiplicity.range>
      </UML:Multiplicity>
      </UML:AssociationEnd.multiplicity>
      </UML:AssociationEnd>
      <UML:AssociationEnd name='employee' visibility='public' isSpecification='false'
      isNavigable='true' ordering='unordered' aggregation='none' targetScope='instance'
      changeability='changeable' participant='S.2' association='G.1'>
      <UML:AssociationEnd.multiplicity>
      <UML:Multiplicity>
      <UML:Multiplicity.range>
        <UML:MultiplicityRange lower='0' upper='-1'/>
      </UML:Multiplicity.range>
      </UML:Multiplicity>
      </UML:AssociationEnd.multiplicity>
      </UML:AssociationEnd>
      </UML:Association.connection>
      </UML:Association>
      </UML:Namespace.ownedElement>
      </UML:Model>
</XMI.content>
</XMI>
```

Capítol 4

Comparació de diagrames.

Disseny i implementació.

En aquest capítol hi trobarem l'anàlisi del disseny i implementació de l'aplicació que ens ha de determinar quan dos diagrames UML, una vegada comparats els podem considerar exactament iguals.

4.1 Introducció.

El nucli del projecte el trobarem en aquest capítol, doncs es tracta que donats dos models UML, siguem capaços de determinar si aquests dos diagrames són igual, i si no ho son determinar quines són les diferències. Per simplificar el projecte, aquest es centra únicament en els diagrames de classes.

D'acord amb la teoria de conjunts, direm que dos conjunts A i B són iguals , si i nomes si, del conjunt A és un subconjunt de B i al mateix temps, B és un subconjunt d'A.

Aplicat al nostre cas, direm que dos diagrames A i B són iguals si i només si, tots els elements del diagrama A (classes, atributs, operacions, associacions, cardinalitats, ...) apareixen en B i a l'hora tots els elements de B apareixien en A.

Els diagrames se'ns proporcionen en forma de fitxer XMI i utilitzarem les llibreries DresdenOcl per a obtenir-ne informació vàlida per a la transformació del model que volem efectuar.

Una vegada haguem estat capaços de transformar la informació a la nostre estructura de dades, caldrà fer la comparació i obtenir una llista dels errors detectats, perquè aquests puguin ser tractats per l'aplicació que ens ha sol·licitat la comparació.

4.2 Casos d'us.

4.2.1 Cas d'us: general.

El primer anàlisi de l'aplicació per poder comparar diagrames de classes, ens porta a una aproximació molt simple del que ha de ser el nostre sistema.

A grans trets el que li demanarem al nostre sistema és:

- Carregar els models a comparar. En el nostre cas tindrem un especialització per a carregar el diagrama de classes.
- Una vegada carregats els models, podem procedir a efectuar la comparació, que ens ha de comprovar la igualtat entre els dos models.
- Ha manera d'utilitat, hem de poder imprimir els diagrames per comprovar que el contingut és el correcte, respecte del fitxer XMI.

El gràfic corresponent a aquest cas d¡ús és el que es mostra a continuació:

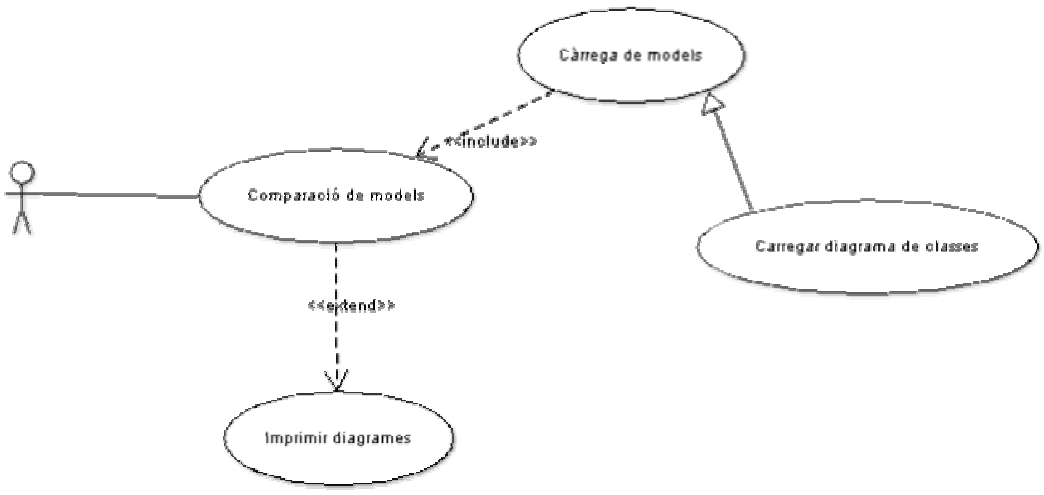


Figura 28. Cas d'us general

Val a dir que s'ha dibuixat un actor que encara no sabem qui serà, doncs estem creant un sistema que a partir de dos models, obté un col·lecció de diferències, però que en principi el nostre sistema es queda en aquest punt.

A partir de la primera aproximació feta en el cas d'us general, ens cal especificar amb més detall els casos d'us corresponents a "Càrrega del diagrama de classes" i "Comparació de models".

4.2.2 Cas d'us: carregar diagrama de classes.

EL gràfic de la càrrega del diagrama de classes es mostra en la figura 29, i és correspon al cas d'us "Carregar Diagrames de Classes". Per poder entendre'l millor, anem a fer una explicació de cadascun dels casos d'us inclosos.

CAS D'US	DESCRIPCIÓ
Obtenir Ocl Model	A partir de l'estructura d'informació de DresdenOcl, proporciona objectes per a cadascun dels elements que conformen el diagrama de classes
Llegir fitxer XMI	És llegeix un fitxer del model proporcionat en format XMI, transformant-lo en l'estructura d'objectes de Dresden Ocl
Carregar classes	Obté la informació de les classes en objectes Dresden Ocl i els transforma a objectes de classe per a poder fer la comparació
Carregar Associacions	Obté la informació de les associacions en objectes Dresden Ocl i els transforma a objectes d'associacions per a poder fer la comparació
Carregar Association Class	Inclou les funcionalitats dels casos d'us "Carregar classes" i "Carregar associacions"
Carregar atributs	Obté la informació dels atributs en objectes Dresden Ocl i els transforma a objectes d'atributs per a poder fer la comparació
Carregar operacions	Obté la informació de les operacions en objectes Dresden Ocl i els transforma a objectes d'operacions per a poder fer la comparació

CAS D'US	DESCRIPCIÓ
Carregar paràmetres operacions	Obté la informació dels paràmetres de les operacions en objectes Dresden Ocl i els transforma a objectes de paràmetres d'operacions per a poder fer la comparació
Carregar Association Ends	Obté la informació de les Association Ends en objectes Dresden Ocl i els transforma a objectes d'association Ends per a poder fer la comparació

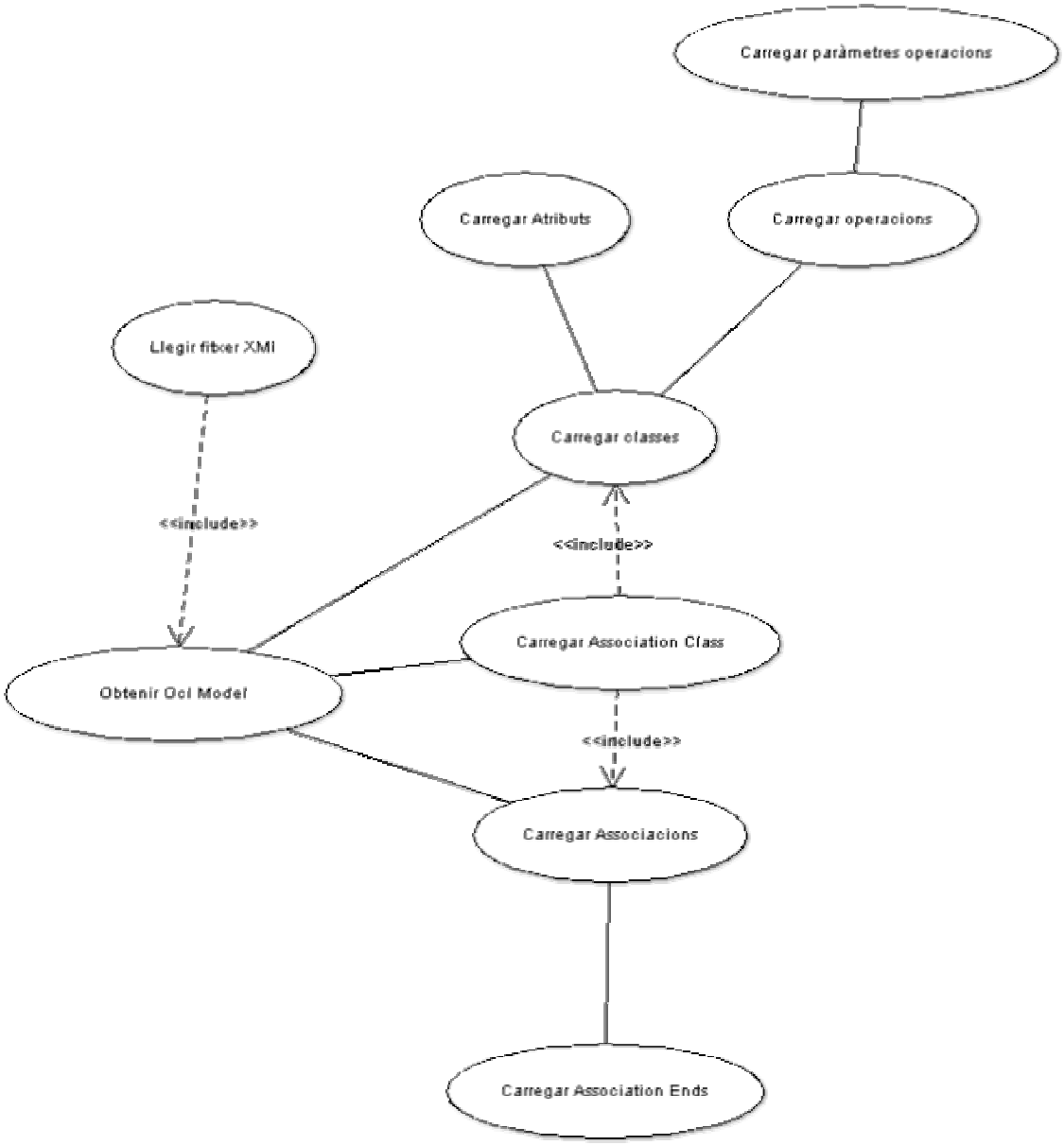


Figura 29. Cas d'ús : carregar diagrama de classes

4.2.3 Cas d'us: comparació de models.

Tal com s'ha dit en la introducció del capítol, s'han de considerar iguals dos diagrames quan cadascun sigui subconjunt de l'altre. Per això haurem de comparar element a element per determinar si aquesta igualtat es certa.

Determinat la funcionalitat de cadascun dels casos d'us que conformen aquest apartat.

CAS D'US	DESCRIPCIÓ
Comparar Models	Recollirà la llista de tots els errors produïts durant la comparació.
Comparar Classes	A partir de dos llistes de classes, una per a cadascun dels diagrames, recorrerà la llista de classes del primer i buscarà cadascuna de les classes en la llista del model a comparar. És considerarà que dos classes són iguals si són iguals el nom, l'espai de noms, els atributs i les operacions.
Comparar Associacions	A partir de dos llistes d'associacions, una per a cadascun dels diagrames, recorrerà la llista d'associacions del primer i buscarà la mateixa associació en la llista d'associacions del model a comparar. És considerarà que dos associacions són iguals si són iguals el nom, l'espai de noms i els AssociationEnds
Comparar Association Class	Contindrà les funcionalitats tant de les classes com les associacions
Comparar atributs	A partir de dos llistes d'atributs, una per a cadascuna de les classes a comparar, recorrerà la llista d'atributs de la primera i buscarà el mateix atribut en la llista a comparar.Dos atributs seran iguals si també ho son el nom, l'espai de noms, el tipus, la visibilitat i la multiplicitat.

CAS D'US	DESCRIPCIÓ
Comparar operacions	A partir de dos llistes d'operacions, una per a cadascuna de les classes a comparar, recorrerà la llista d'operacions de la primera i buscarà la mateixa operació en la llista a comparar. Dos operacions seran iguals si també ho son el nom, l'espai de noms, i els paràmetres de l'operació
Comparar paràmetres operacions	A partir de dos llistes de paràmetres, una per cadascuna de les operacions a comparar, recorrerà la llista de paràmetres de la primera i buscarà el mateix paràmetre en la llista a comparar.Dos paràmetres seran iguals si també ho son el seu nom, l'espai de noms, el sentit, el tipus i el valor per defecte
Comparar Association Ends	A partir de dos llistes d'association Ends, una per a cadascuna de les associacions a comparar, recorrerà la llista d'association Ends de la primera i buscarà la mateixa Association Ends en la llista a comparar. Dos associaction Ends seran iguals si també ho son el seu nom, l'espai de noms i la seva multiplicitat.

El gràfic corresponent és el que es mostra en la figura següent:

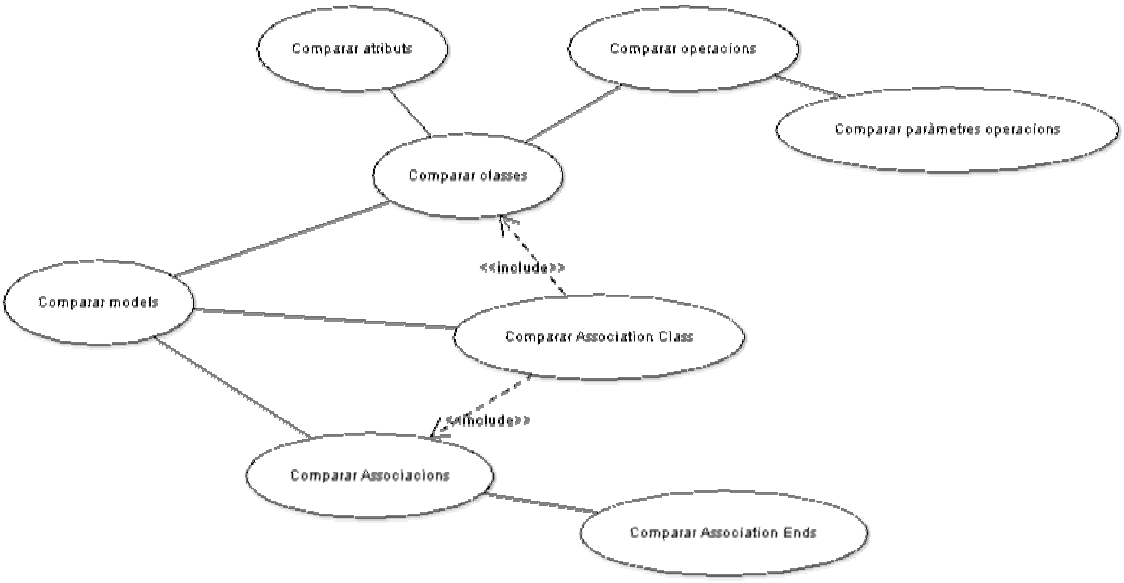


Figura 30. Cas d'ús : comparació de models

La comparació de models, tal com ha quedat en aquest mateix apartat, cal fer-la en els dos sentits, per poder assegurar que tots dos són iguals. Si ho féssim només en 1 sentit, podria ser que el primer fos un subconjunt de l'altre i el segon tingués més elements.

4.3 Diagrama de classes.

4.3.1 Visió general.

Com a resultat de l'estudi dels casos d'us anteriorment exposats, n'extraurem el diagrama de classes. Aquest diagrama és la transformació de la informació continguda en el fitxer XMI, en classes de les quals podrem efectuar una comparació de models.

En el primer diagrama, que es mostra en la figura 31, únicament veurem les classes amb les seves herències i relacions, deixant per un altre apartat posterior la descripció completa de les classes, amb tots els mètodes i atributs corresponents.

Descrivint el model, podem veure que tindrem una classe UmlComparation que contindrà dos instàncies de la classe UmlDiagram, que correspondran als dos models que volem comparar. Cada diagrama, tindrà coma mínim una instància de la classe

UmlClass, poden tenir o no associacions i associacions qualificades, que queden reflectides per les classes UmlAssociation i UmlAssociationClass respectivament. A l'hora, cada instància d'una classe, tindrà un nombre indeterminat d'instàncies del tipus UmlAttribute, per als atributs, passant el mateix amb el nombre d'operacions, que en el cas que ens ocupa seran instàncies de la classe UmlOperation. Les instàncies d'operacions, tindran un nombre indeterminats de paràmetres, però com a mínim en tindran un, que correspondrà al paràmetre de retorn de l'operació, estan representades les operacions per la classe UmlOperationParameter. Cada instància que representi una associació tindrà relacionades com a mínim 2 instàncies de la classe UmlAssociationEnds, que representaran els extrems de les associacions. Ja per acabar, remarcar que les instàncies de la classe UmlAssociationClass, herederan directament de les classes UmlClass i UmlAssociation, doncs estem parlant d'una associació qualificada que a part de tenir els extrems de les associacions, tindrà també els atributs i operacions intrínseques de la relació.

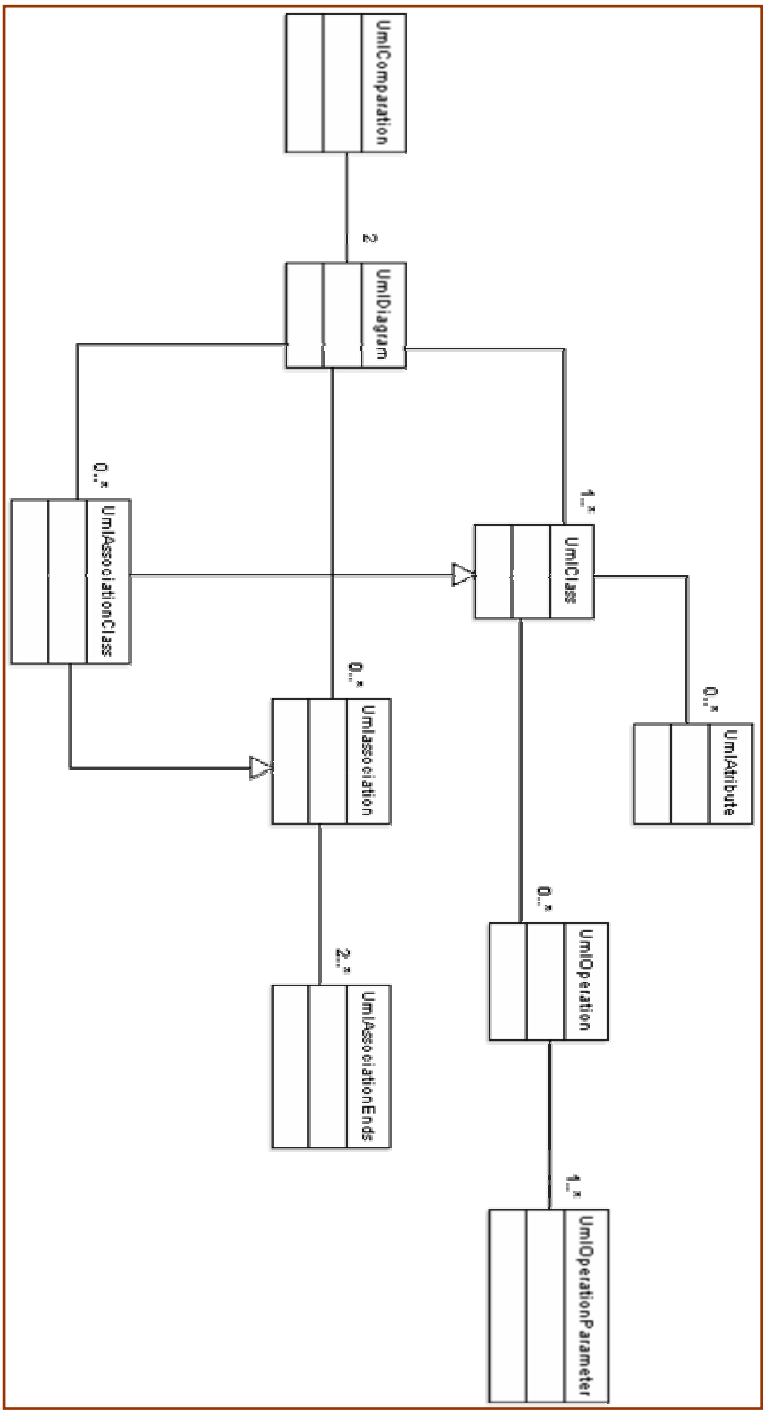


Figura 31. Diagrama de classes de la comparació de models

4.3.2 Descripció detallada de les classes.

En l'apartat anterior, per una part, ens ha interessat veure el diagrama des d'un punt de vista conceptual, que ens permet abstreure'ns dels atributs i mètodes. Per un altre part, presentar el diagrama complert amb atributs i mètodes per a totes les classes que conformen el diagrama, és físicament impossible per problemes d'espai.

Així doncs el que es pretén en aquest apartat, es veure cadascuna de les classes de manera individual i complerta, amb indicació de tots els atributs i mètodes, així com un petita explicació per a cadascuna de les classes.

4.3.2.1 UmlComparation.

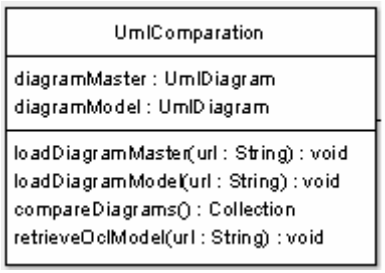


Figura 32. Classe UmlComparation

La classe UmlComparation és la part visible del nostre framework, i serà la única classe que hauran d'instanciar aquelles aplicacions que vulguin utilitzar aquesta aplicació per a fer comparacions de diagrames de classes.

Veurem que la classe conté com atributs els dos diagrames del quals haurem d'efectuar la comparació i així mateix té dos operacions per a carregar cadascun dels diagrames. La comparació dels diagrames la farem cridant el mètode compareDiagrams, el qual ens donarà com a resultat una col·lecció d'errors, que en el supòsit que aquesta col·lecció estigui buida ens indicarà que els dos diagrames són exactament iguals.

També podem veure un mètode print(), que només té utilitat en funcions de depuració del programa.

4.3.2.2 UmlDiagram.

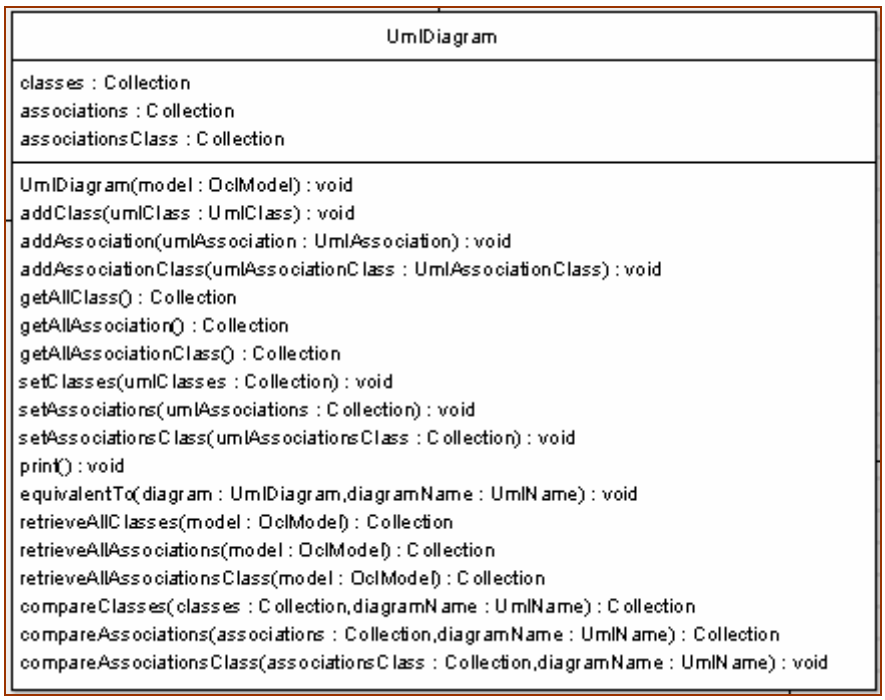


Figura 33. Classe UmlDiagram

La classe UmlDiagram representa el diagrama vist com una totalitat, on hi trobarem en el seus atributs tres col·leccions d'objectes que representen a les classes, associacions i associationsClass.

Les funcions que comencen per *retrieve* , són mètodes privats de la classe, que tenen com a missió transformar els objectes obtinguts des del framework de DresdenOCl i els transformen en objectes propis de la nostra aplicació.

A la classe a part dels "getter's" i "setter's" corresponents a cadascuns dels atributs s'hi ha afegit un altre mètode que a permet afegir un element puntual a la col·lecció, perquè el framework està pensat per a que hi hagi la possibilitat de carregar el diagrama no tant sols a partir de fitxes XMI, sinó que aquest pugui ser construït a partir de zero de manera interactiva. No s'hi ha afegits els mètodes per eliminar elements de les

col·leccions pel propi caràcter de la col·lecció, on hi manquen els índexs. La funció es pot suplir amb els mètodes "get" i "set" que si que existeixen a la classe.

El mètode *equivalentTo* és l'encarregat de comparar dos diagrames, que es recolza amb les funcions que s'inicien per *compare* , que són mètodes privats encarregats de comparar cadascuna de les col·leccions que pertanyen a la classe.

4.3.2.3 UmlClass.

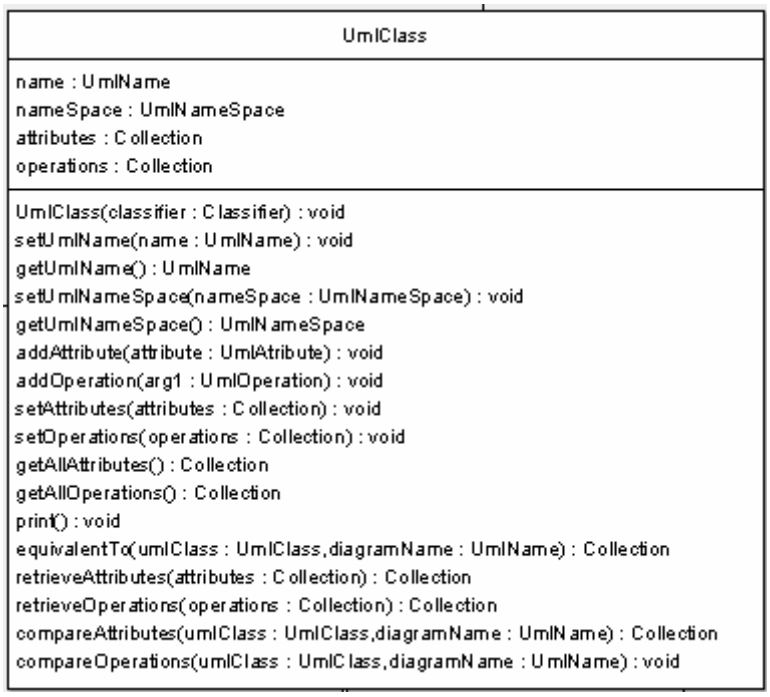


Figura 34. Classe UmlClass

Les classes del diagrama estan representades per UmlClass on en els atributs hi trobem una constants que apareixeran en quasi totes les classes del model que són el nom de l'element, en cas de la classe, i el seu espai de noms corresponents.

En el cas de la classe cal especificar-hi altres dos atributes que en acaben de conformar els atributs de la classe com són la col·lecció d'atributs de la classe del model a tractar, que rep el nom d'*attributes* , i la col·lecció d'operacions de la mateixa classe, reflectits a *operations*.

Pel que respecte a les operacions, valen les mateixes reflexions fetes per a la classe *UmlDiagram* , on apareixen els "getter's" i "setter's" corresponent, així com el mètodes per obtenir informació d'objectes de DresdenOcl i el mètode de comparació, *equivalentTo*, que es complemente amb mètodes *compare*.

4.3.2.4 UmlAssociation.

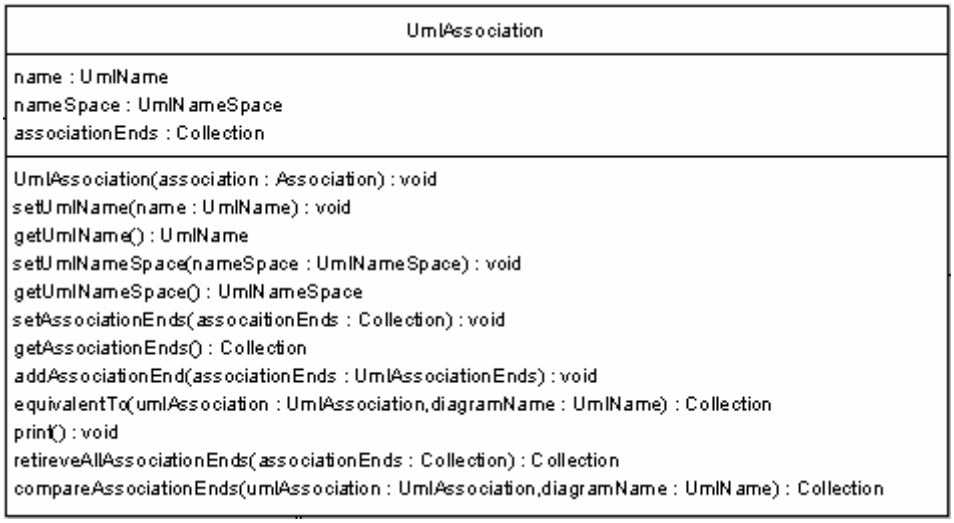


Figura 35. Classe UmlAssociation

Aquesta classe representa la relació que hi ha entre dos o més classes del model, però a part del nom i de l'espai del nom no aporten gaire més informació relativa a les associacions.

Fer notar que els AssociationsEnds, que són els extrems de l'associació que és el lloc on hi trobarem la informació que complementa a la pròpia associació.

Pel que fa als mètodes, els comentaris es repeteixen respecte a les classes anteriorment mostrades, i en tot cas només recordar l'existència del mètode *equivalentTo* que utilitzarem per a efectuar comparacions.

4.3.2.5 UmlAssociationClass.

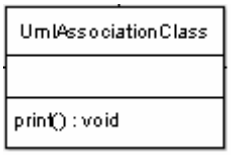


Figura 36. Classe UmlAssociationClass

La classe és una classe pràcticament buida, doncs tot el seu contingut el podrem trobar a la classe UmlClass de la qual és una generalització.

Poden observar que l'únic mètode sobreescrit és *print* però la justificació de la seva existència la trobarem en desenvolupaments futurs.

4.3.2.6 UmlAttribute

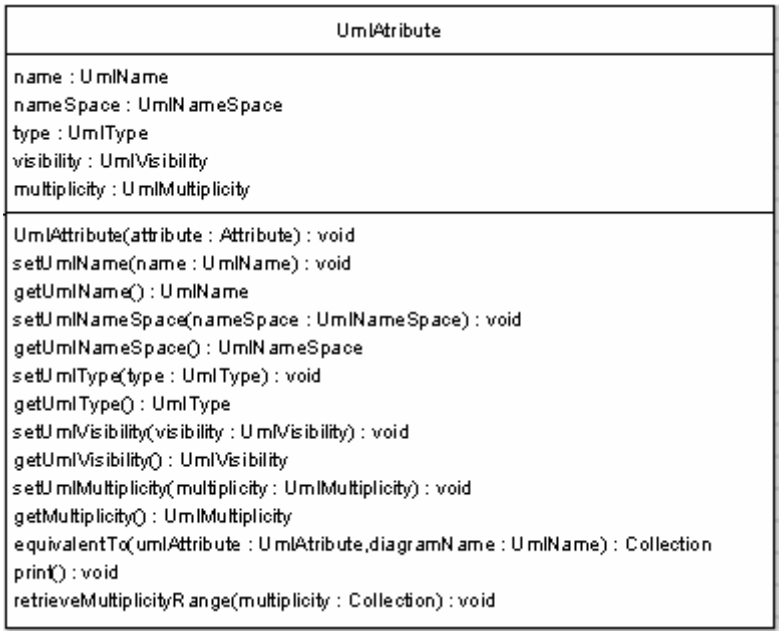


Figura 37. Classe UmlAtribute

És la representació dels atributs on hi podem trobar les característiques de cadascun dels atributes, on podem hi tornem a trobar el nom i l'espai de noms, ja coneguts d'altres classes, però a més trobem que la definició de les característiques de

l'atribut es completa amb el tipus de dades, la visibilitat de l'atribut i la seva multiplicitat, que estan representades per *type, visibility i multiplicity*, respectivament.

El comentari relatiu als mètodes, és pràcticament calcat al que ja s'ha dit respecte als mètodes en les altres classes del diagrama.

4.3.2.7 UmlOperation.

Les operacions del model estan representades per la classe UmlOperation, destacant dels seus atributs l'aparició de la visibilitat que està representada per l'atribut *visibility*. Cal remarcar també l'existència d'una col·lecció que representa els paràmetres que disposa l'operació i que mereixen un tractament igual a les altres col·leccions aparegudes en d'altres classes del diagrama estàtic.

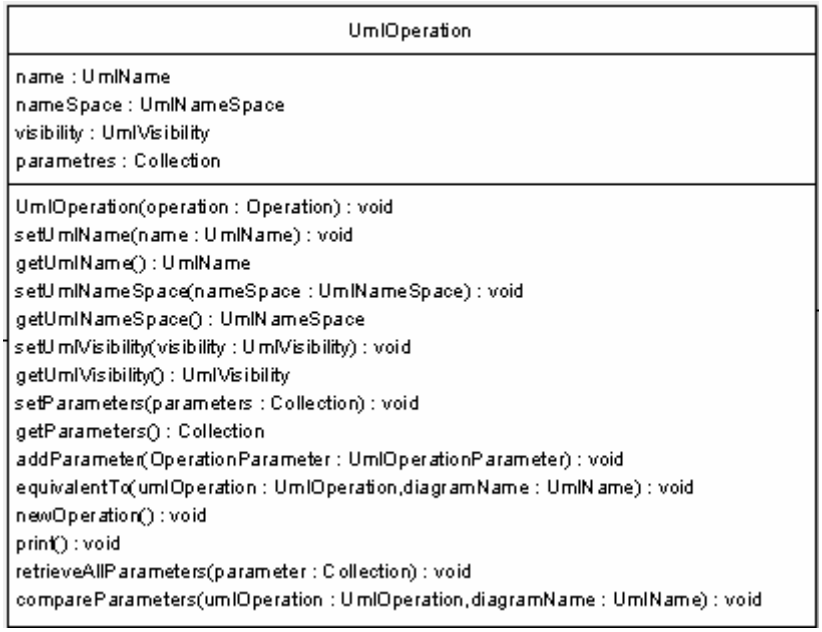


Figura 38. Classe UmlOperation

Els mètodes no tenen cap particularitat que no haig estat esmentada, recordant que la comparació l'efectuarem a partir del mètode *equivalentTo*.

4.3.2.8 UmlOperationParameter.

Els atributs de la classe que representa els paràmetres de les operacions tenen els atributs corresponents al nom i l'espai de noms com quasi la resta de classes, però s'utilitza per qualificar el paràmetre amb les característiques d'abast, tipus de dades i valor per defecte, en els atributs de la classe *kind, type i defaultValue*, respectivament.

El mètode equivalentTo, és el que s'utilitza per a poder efectuar la comparació amb altres paràmetres.

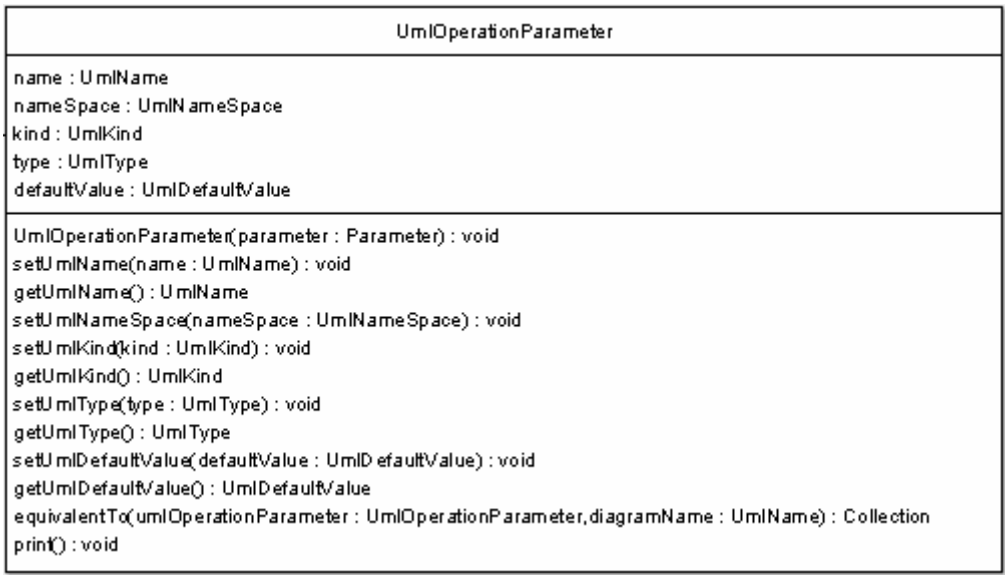


Figura 39. Classe UmlOperationParameter

4.3.2.9 UmlAssociationEnd.

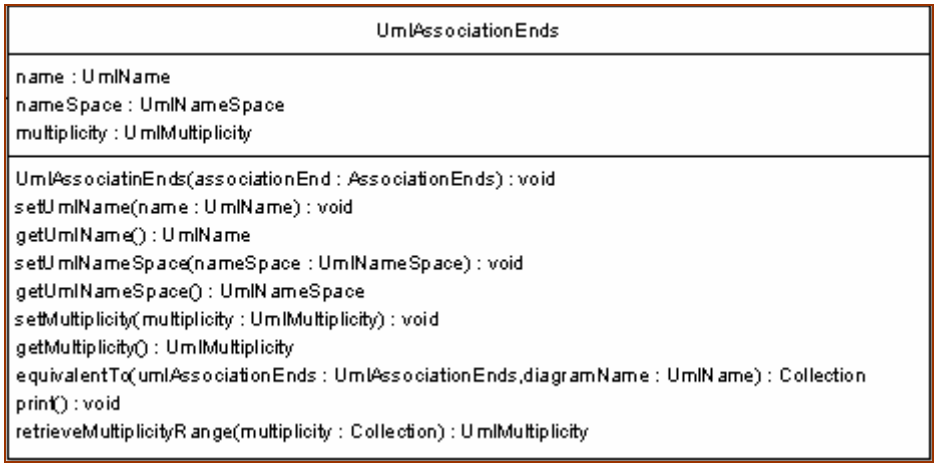


Figura 40. Classe UmlAssociationEnd

Ja per acabar la descripció de les classes, s'exposa la classe que representa els extrems de les associacions, on tornem a trobar un altre com el nom i l'espai de noms ocupat, juntament amb la multiplicitat de l'associació que es representa per l'atribut *multiplicity*.

El mètode *equivalentTo* torna a ser el que s'encarrega de determinar l'igualtat de la instància de la classe amb la instància d'un altre classe.

4.3.2.10 Altres classes auxiliars.

Al llarg de la descripció de les classes del diagrama estàtic, han anat apareixen classes que no han estat descrites com són :

- UmlDefaultValue
- UmlKind
- UmlMultiplicity
- UmlName
- UmlNamespace
- UmlType
- UmlVisibility

Aquestes classes es corresponent als tipus de dades personalitzats en el diagrama estàtic i que podrien ser substituïdes per un string. El motiu que no s'hagi fet així i en canvi s'hagi triat una classe per a representar la dada, està fet pensant que en el futur aquests classes poden ser completades i obtenir d'elles altres utilitats que no seria possible si s'hagués limitat la informació a un String.

Això es pot veure en la classe *UmlMultiplicity*, que permet tractar la multiplicitat com un dada unitària, però en canvi es poden efectuar operacions que van més enllà del que podríem fer si haguéssim descrit la variable com un String, o en aquest cas com un parella d'enters.

Aquestes classes les podem veure agrupades en la figura 41, doncs no tenen més interès que el fet de conèixer la seva existència, i l'estructuració aplicada, sabent que hi ha la possibilitat d'estendre la seva funcionalitat de manera senzilla.

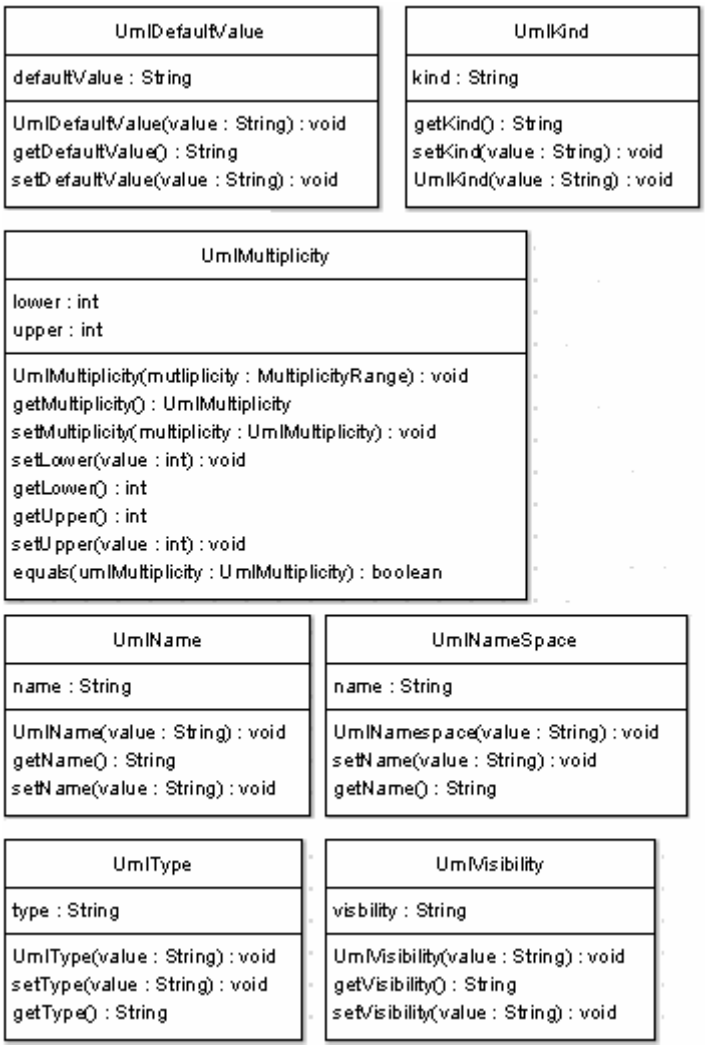


Figura 41. Altres classes

4.4 Diagrama d'estats.

Fins aquest moment em pogut veure l'estructura estàtica, però igual que en un edifici es pot veure la construcció des de diferents punts de vista, també cal que mirem l'estructura des d'altres punts de vista.

Un punt important és veure els estats pels quals poden passar els objectes que conformen el sistema. Així un dels aspectes importants es saber en que es pot trobar una persona que utilitzi el framework des d'altres aplicacions.

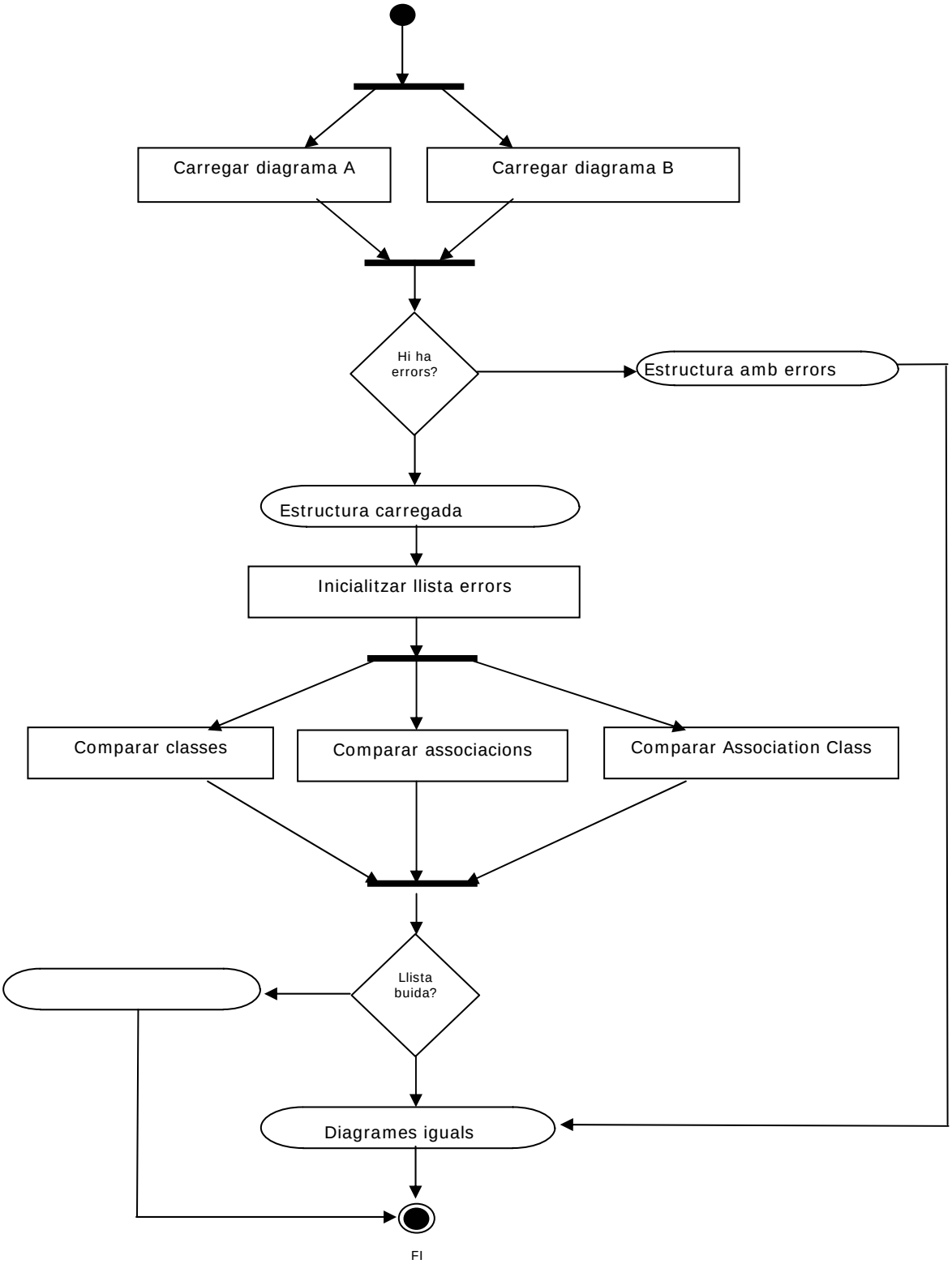


Figura 42. Diagrama d'estats

La figura 42, ens mostra el diagrama d'estats de l'objecte UmlComparation, que pot estar bàsicament en tres estats, a part de l'estat inicial, que no es té en consideració. Així els estats possibles són:

- Estructura amb errors: En aquest cas s'ha intentat carregar els fitxers XMI que contenien l'estructura dels models a comparar, però no s'han pogut arribar a carregar. En aquest supòsit no queda més remei que passar a l'estat final sense procedir a la comparació.
- Diagrames diferents: Una vegada s'han pogut carregar els fitxers XMI, amb una estructura de dades d'acord amb les nostres especificacions, es procedeix a realitzar la comparació i es troben errors que determinen que els models no son iguals.
- Diagrames iguals: Si després d'efectuar la comparació la llista d'errors està buida, significa que els diagrames són exactament iguals.

4.5 Diagrama de seqüència.

Un altra punt de vista de l'estructura ens la dona el diagrama de seqüència que ens permet veure com es relacionen les classes incloses en el diagrama estàtic.

M'ha semblat oportú, separar la càrrega de fitxers de la comparació de models, en primer lloc perquè són els dos casos d'us que hem vist al inici de l'anàlisi i en sóc lloc, un altre cop més, per manca d'espai físic que faci comprensible el diagrama.

4.5.1 Diagrama de seqüència: Càrrega de fitxers.

La seqüència comença obtenint una instància de la classe OclModel per a carregar-hi el contingut del fitxer XMI. Una vegada obtinguda aquesta instància s'utilitza per construir la nostre estructura d'objectes, a partir de subproductes de la mateixa llibreria DresdenOcl, que permeten instanciar tot el conjunt.

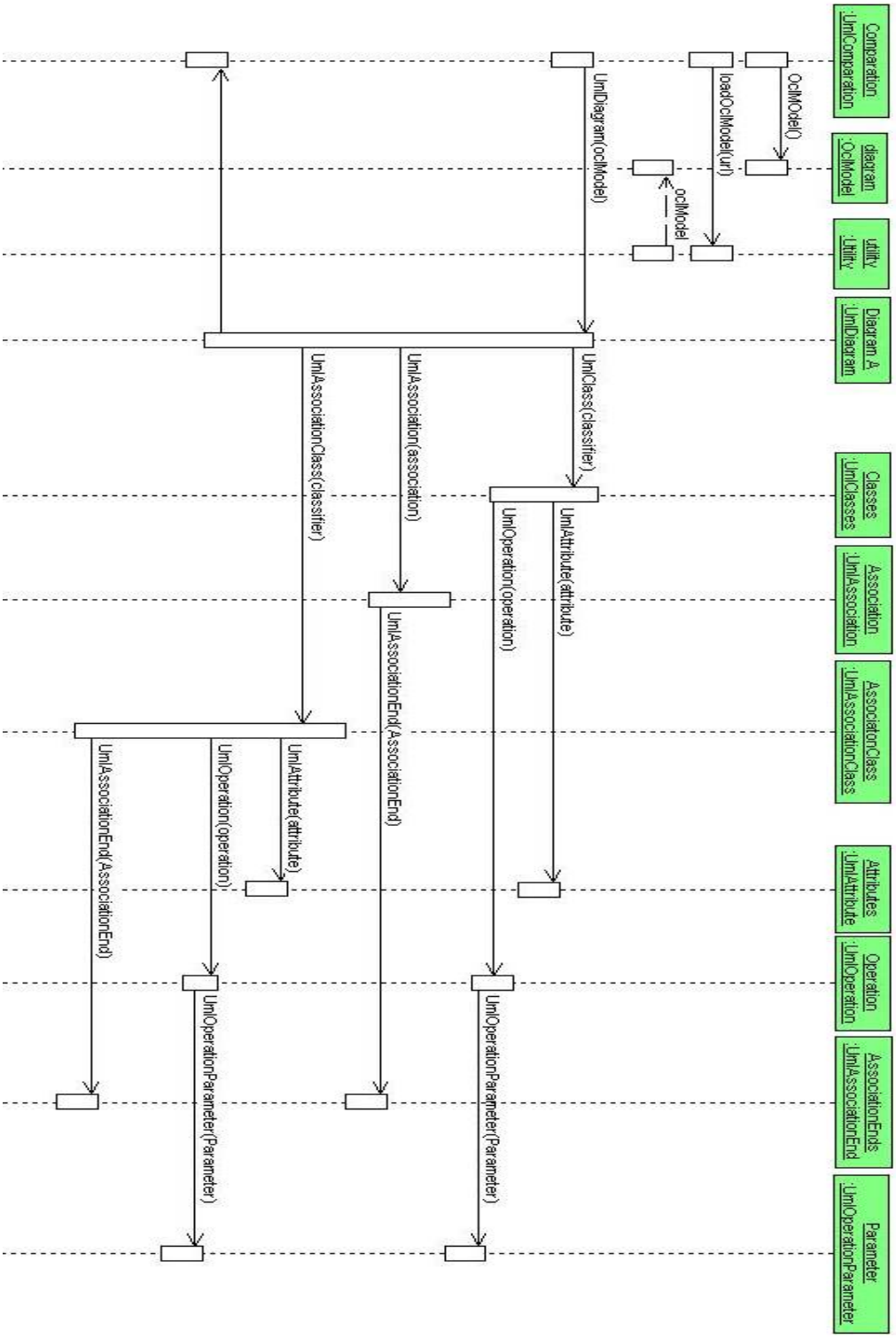


Figura 43. Diagrama de seqüència de càrrega de fitxers

Òbviament, la seqüència correspon a la càrrega d'un fitxer XMI, i aquesta es repetiria per al segon model de la comparació.

4.5.2 Diagrama de seqüència: Comparació de models.

En la seqüència de comparació de models, apareix un element nou, que fins al moment no s'havia mencionat, com és l'objecte d'error que recull les diferències entre els dos models.

Aquesta classe té la següent composició:

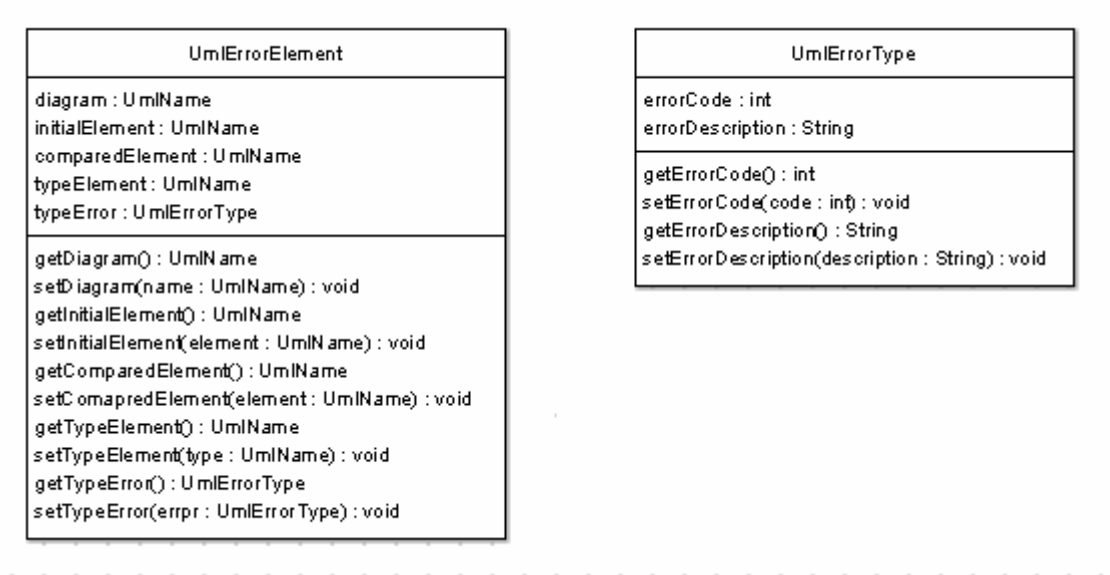


Figura 44. Classes UmlErrorElement i UmlErrorType

Potser hauríem pogut passar amb un codi d'error i un fitxer de descripcions, però en un mon orientat a objectes i pensant en la possibilitat de posteriors evolucions del projecte, sembla una solució amb més possibilitat d'adaptacions.

El diagrama s'inicia amb la instanciació d'un vector de errors, en principi buit. La comparació s'efectua passant al diagrama A el diagrama B , per tal que sigui aquest objecte l'encarregat d'efectuar la comparació. Així és el diagrama A el que s'encarrega de comparar tots els seus elements amb els del diagrama B, fins a recollir tots els errors

en la mateixa col·lecció. Si al final de la comparació la col·lecció és buida significarà que els diagrames són iguals, pel contrari si aquesta col·lecció conté elements, representaran les diferències entre els dos diagrames comparats.

El gràfic es presenta a continuació, i només queda representada una instància per a cadascun dels elements de les col·leccions, però es obvi que caldrà recorre cadascuna de les col·leccions de classes, associacions i associationEnds per a comparar-los un a un.

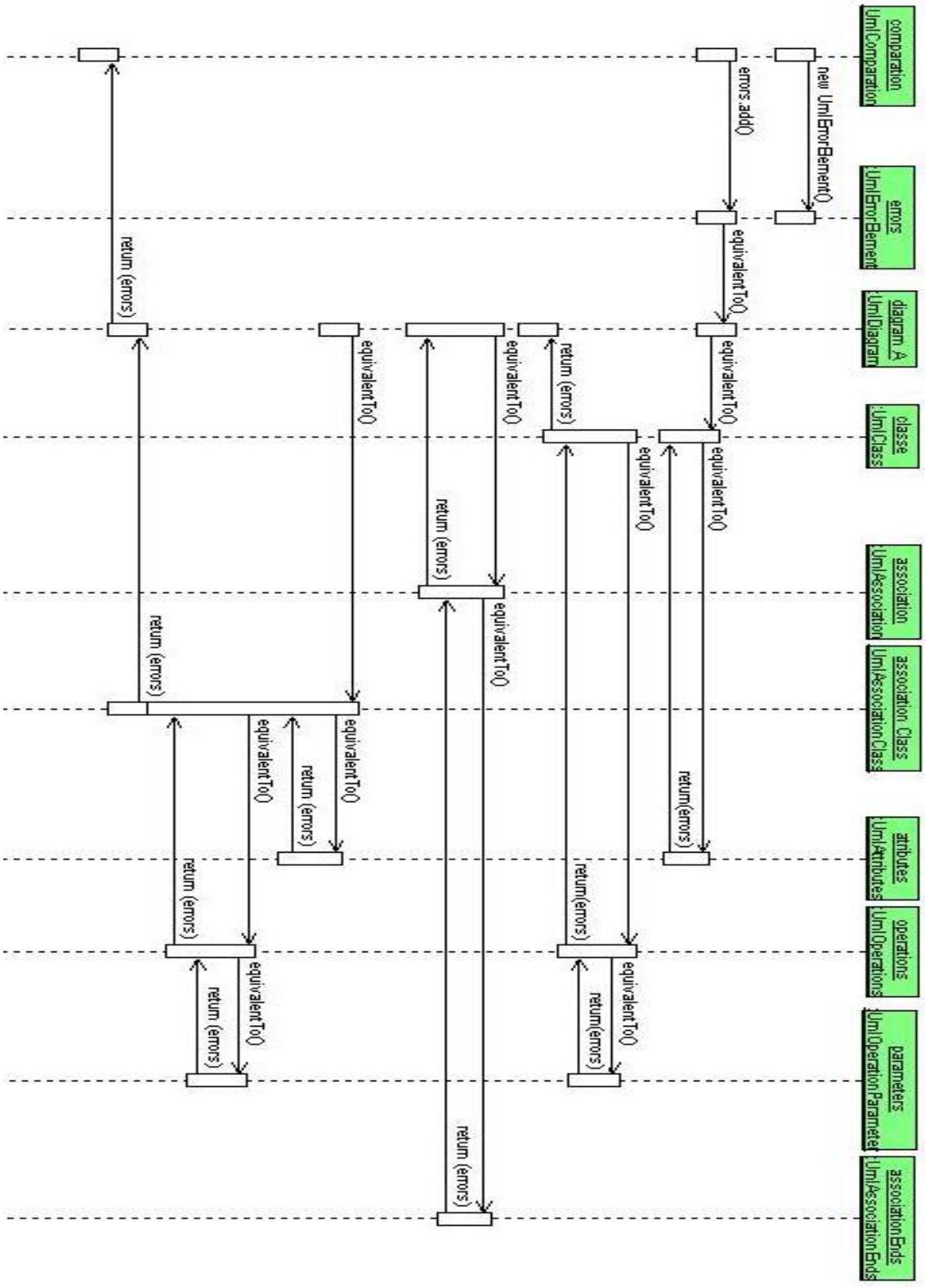


Figura 45. Diagrama de seqüència de comparació de models

4.6 Implementació.

La implantació s'ha efectuat amb la codificació de l'anàlisi en llenguatge Java, i s'han utilitzat interfícies per tal d'evitar el màxim d'acoblament entre el codi.

4.6.1 Distribució.

La distribució en paquets ha quedat tal com segueix:

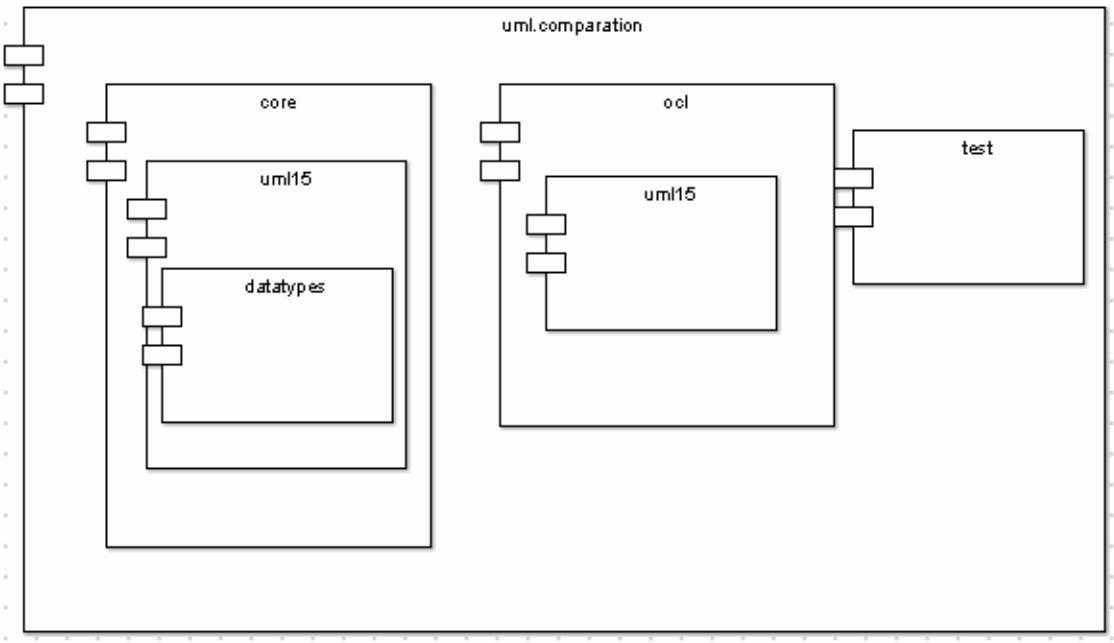


Figura 46. Diagrama de distribució

S'han fet dos paquets diferenciats, més un de test, que té l'única funcionalitat de provar el codi generat. Tots els paquets es trobaran dins de *uml.comparation* i hi ha un paquet *uml.comparation.ocl* on hi trobaren totes aquelles utilitats relatives a la transformació de fitxers XMI a objectes DresdenOcl, que posteriorment en serviran per a generar la nostre estructura d'objectes.

La manipulació de l'estructura i la comparació pròpiament dita, la trobarem al paquet *uml.comparation.core* . S'ha creat un paquet que indica que la versió d'uml que

s'està tractant es la 1.5, així tant el paquet *ocl* com *core* disposen d'un paquet addicional que ens indica aquesta versió. L'aplicació futura en altres versions d'uml es poden anar incorporant als paquets respectius.

També destacar que s'han posat en un paquet apart els tipus de dades generats per a la manipulació d'informació pròpia de la nostre estructura de dades.

4.6.2 Test.

Per provar l'aplicació s'ha generat la següent classe de test:

```
package uoc.umlcomparation.test;

import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import uoc.umlcomparation.core.uml15.UmlComparation;
import uoc.umlcomparation.core.uml15.UmlErrorShow;
import uoc.umlcomparation.core.uml15.impl.UmlComparationImpl;
import uoc.umlcomparation.core.uml15.impl.UmlErrorShowImpl;

public class ComparationTest {

	/**
	 * @param args
	 * @throws MalformedURLException
	 * @throws UnsupportedEncodingException
	 */
	public static void main(String[] args) throws ç
			MalformedURLException, UnsupportedEncodingException {


		UmlComparation comparation = new UmlComparationImpl();

		try {
		    comparation.loadDiagramMaster(
		        "C:\\eclipse\\workspace\\Exemples\\ExampleUoc_Master.xmi");
		} catch (Exception e) {
		    System.out.println(
		        "IMPOSSIBLE TO LOAD THE SPECIFICATION OF THE MASTER DIAGRAM");
		     e.printStackTrace();
		}

		try {
		    comparation.loadDiagramModel(
		        "C:\\eclipse\\workspace\\Exemples\\ExampleUoc_Model.xmi");
		} catch (Exception e) {
		    System.out.println(
		         "IMPOSSIBLE TO LOAD THE SPECIFICATION OF THE SOLUTION DIAGRAM");
		    e.printStackTrace();
		}

		UmlErrorShow showErrors = new UmlErrorShowImpl();
		showErrors.ToConsole(comparation.compareDiagrams());
	}
```

```
}
```

Com es pot veure la classe de test utilitza el framework creat en aquest projecte per a comparar dos models guardats en format XMI, i utilitza una classe que no havíem vist fins al moment. Aquesta classe únicament té com a missió imprimir per la consola els errors detectats durant la comparació dels models. El codi de la classe UmlErrorShowImpl és el següent:

```
package uoc.umlcomparation.core.uml15.impl;

import java.util.Collection;
import java.util.Iterator;
import uoc.umlcomparation.core.uml15.UmlErrorElement;
import uoc.umlcomparation.core.uml15.UmlErrorShow;

public class UmlErrorShowImpl implements UmlErrorShow {

    public void ToConsole(Collection<UmlErrorElement> errors) {
        Iterator ite = errors.iterator();

        while (ite.hasNext()) {
          UmlErrorElement err = (UmlErrorElement) ite.next();
          System.out.println("Diagram          = " + err.getDiagram().getName());
          System.out.println("Type element      = " + err.getTypeElement().getName());
          System.out.println("Initial element   = " + err.getInitialElement().getName());
          if (err.getCompareElement()!= null) {
                System.out.println("Compared element  = " + err.getCompareElement().getName());
          }
          System.out.println("Type error       = " + err.getTypeError().getErrorDescription());
          System.out.println("");

        }

    }

}
```

Capítol 5

Modificació i millores del disseny.

Fins al moment hem creat una aplicació, sense preocupar-nos de les optimitzacions, com tampoc s'han introduït cap dels aspectes que s'han mencionat en l'exposició del projecte. En aquest capítol en ocuparem de refinar el disseny i introduir millores que facin més flexible la comparació de models de diagrames de classes.

5.1 Introducció.

En el diagrama de classes, amb un senzill cop d'ull es pot observar que hi ha elements que es repeteixen al llarg de totes les classes, i que ha més no tant sols comparteixen el nom, sinó també el sentit. Per tant, la lògica ens ha de portar a extreure aquests parts comunes a tots els elements i fer que hi hagi un model base sobre el que s'estenen la resta de classes.

En la introducció de la memòria s'ha parlat de que hi cal fer una comparació de models flexible, de tal manera que sigui capaç de suportar variacions semàntiques que no afectin a la seva estructura.

Aquestes adaptacions de la comparació és centraran en dos aspectes:

- Flexibilitat davant dels paquets: El fet que hi hagi una divisió en paquets diferent a la proposada en el model solució, no ha de suposar que l'estructura de classes no s'adapti a la solució, doncs la distribució de paquets és fa sota un criteri molt personal.

- Possibilitat d'incorporar sinònims als noms del elements: La nomenclatura que s'adopta a l'hora d'anomenar a les classes, es potser una de les variacions més factibles i a l'hora més comuns i que menys afecten a la pròpia estructura.

Així doncs, dedicaré aquest capítol a les modificacions pertinents de l'aplicació, per tal que sigui capaç d'adaptar-se a les circumstàncies que aquí s'han enumerat.

5.2 Racionalització del diagrama de classes.

És pot observar clarament en la definició de les classes del model que hi ha dos elements que es repeteixen absolutament a totes les classes, excepte la classe *UmlComparation*. Aquests dos elements són els atributs *name* i *nameSpace*.

El fet que aquests dos elements no apareguin a *UmlComparation* ve donat pel fet que conceptualment aquesta classe no forma part de la descripció del diagrama de classes, sinó que és la classe del framework que s'instancia per a poder portar a terme la comparació.

Per racionalitzar aquesta situació ens cal fer aparèixer una nova classe en el diagrama estàtic, de la qual hauran d'heretar tota la resta de classes del diagrama, amb l'excepció ja expressada de la classe *UmlComparation* i que contindrà els atributs comuns i amb els seus corresponents "getter's" i "setter's".

Una vegada modificat el diagrama estàtic, aquests queda de la manera que es mostra a la figura 47. Tot i que veurem molts de creuaments de línies, crec que el diagrama es prou explícit per sí mateix.

Tal com ha passat amb el diagrama de classes del capítol anterior, no s'han inclòs en el gràfic la descripció dels atributs i operacions, doncs en aquest cas si que faria impossible la lectura correcte del diagrama.

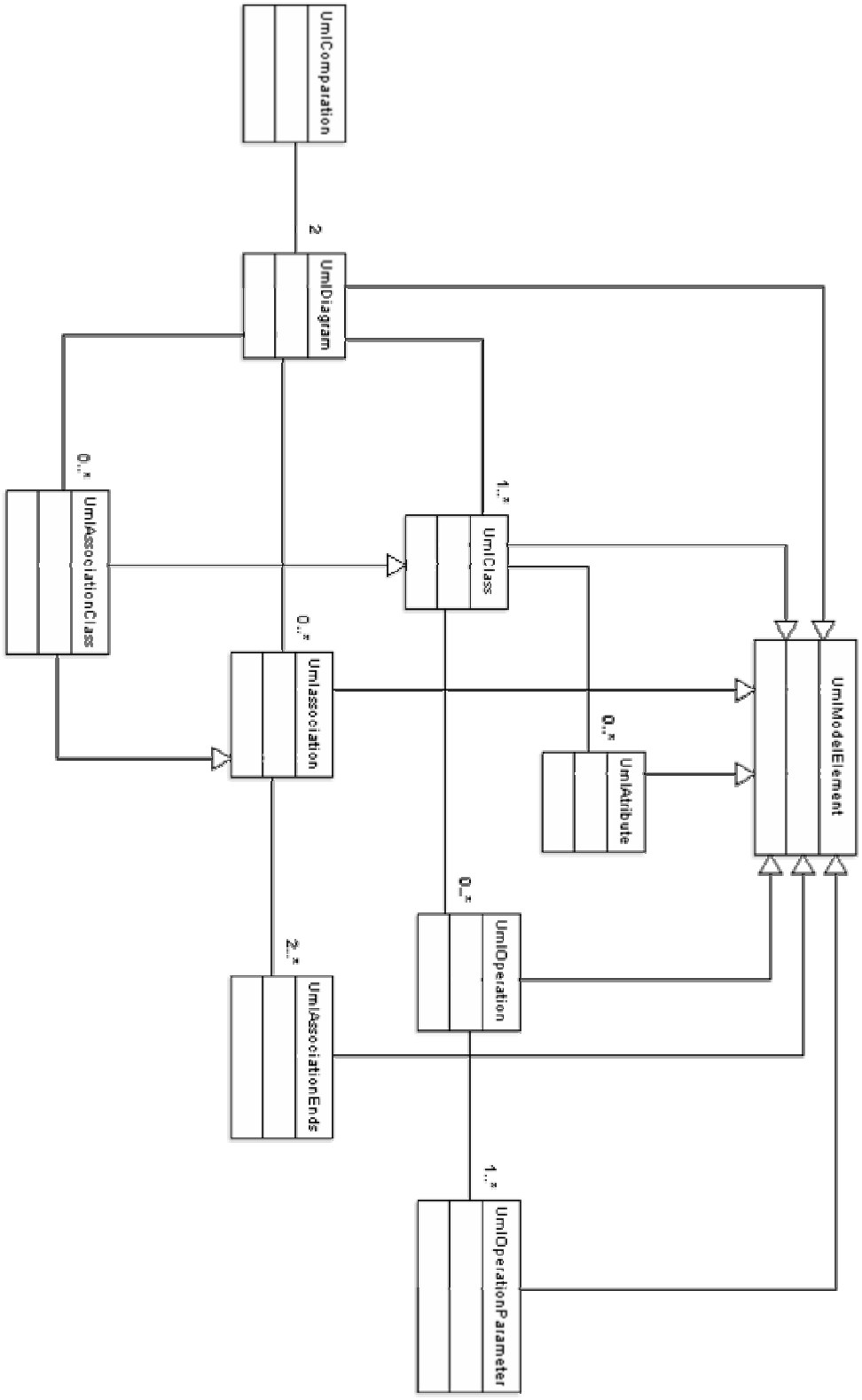


Figura 47. Diagrama de classes de comparació de models modificat

La descripció detallada de la classe *UmlModelElement* la podem veure a continuació:

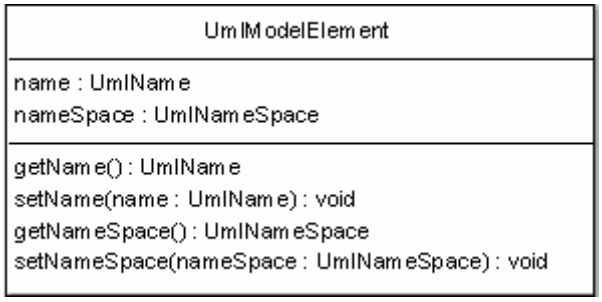


Figura 48. Classe UmlModelElement

Com es pot observar aquesta classe recull els atributs comuns, juntament amb les seves operacions, i per tant les classes que abans tenien com a propis aquests elements, a partir d'aquest moment passaran a disposar d'ells com a elements heretats.

5.3 Flexibilitat semàntica.

S'ha triat aplicar al model dos variacions que permetin aplicar aquesta flexibilitat semàntica: la relativa a l'aplicació de paquets i la relativa a l'aplicació de sinònims.

5.3.1 Relativa als paquets.

No es pot negar que el fet de distribuir les classes a través dels paquets, és un apartat important, que ens serveix per a "endreçar" les aplicacions i fa que quan la quantitat de codi és important, tinguem una ajuda a l'hora de saber on s'han d'anar a buscar els elements. De fet, podem veure la seva importància, quan en la definició d'UML es dedica un diagrama a la distribució de paquets.

Però no és menys veritat, que aquest és un aspecte altament susceptible de ser personalitzat en funció de l'autor del programari. Per tant, quan el que volem és l'anàlisi del diagrama de classes, com és el nostre cas, la distribució de paquets pren una importància relativa, donat que no es pot donar com a incorrecte una solució que

aportant totes les classes sol·licitades, aquestes segueixin una distribució diferent a la aportada com a solució correcte.

És en punts com aquests, que les decisions de disseny prenen importància, així podem destacar la importància de la classe *UmlErrorElement*. Si recordem la definició de la classe de la figura 44, podem veure com el fet donar l'error com una classe i no limitant-nos a un simple codi, ens aporta solucions que d'altre manera serien molt més complexes d'adoptar.

Amb aquesta classe, el programador, sense necessitat de tocar en absolut el model, pot decidir que fer davant de dos classes que són semànticament iguals, a excepció de l'espai de noms:

- Pot donar-ho com error: Tenim dos classes amb una diferència, l'espai de noms, i es reporta la comparació com un error.
- Pot no donar cap error: Les dos classes són equivalents, amb excepció de l'espai de noms, per tant no figurarà com a diferència en la comparació dels models.
- Pot qualificar l'error: El programador pot decidir que tot i que hi ha una diferència entre les classes, aquestes són semànticament iguals, però cal fer notar que no són idèntics, i per tant decideix llençar un error lleu.

Per tant, aquí podem veure com la comparació de models pot variar la seva flexibilitat en funció de les circumstàncies sense necessitat de variar el diagrama estàtic.

En el nostre cas, s'ha decidit prendre com a solució, no donar com a error la diferència de classes, donat que és disposa d'un temps determinat per a l'execució del projecte i s'ha considerat més important incidir en altres aspectes, però queda com una tasca addicional el fet de possibilitar que sigui l'usuari del framework, el que decideixi el nivell de flexibilitat a aplicar a la comparació de models. Així per exemple es podrien fixar uns nivells de flexibilitat i determinar per a cada nivell de flexibilitat que i com s'efectua la comparació.

5.3.2 Relativa a l'aplicació de sinònims.

La possibilitat d'anomenar diferent una mateixa cosa, és una possibilitat que ens proporciona el llenguatge, que ha vegades en jugar a favor i d'altres ens juga en contra. En el cas que ens ocupa, aqueta possibilitat és per a nosaltres un entrebanc, doncs en pot fer donar com a diferents, diagrames que conceptuals són iguals.

Anem a posar un exemple. Suposem que de la lectura d'un enunciat és dedueix que en el nostre diagrama hem de disposar una classe que identifiqui un vehicle i li donem a la classe el nom d'automòbil. De la mateixa lectura, una segona persona en dedueix la mateixa classe però li posa com a nom cotxe. Podem deduir que es tracta de la mateixa classe encara que tinguin un nom diferent? Jo crec que si .

Si determinem que el programa ha de ser flexible a aquestes incidències, també em de determinar a quins dels elements del diagrama apliquem aquesta propietat. En principi donat que el que estem tractant són sinònims de denominacions, aquests sinònims haurien de ser aplicables a tots aquells elements que són susceptibles de rebre denominacions, que en el nostre cas són tots el que conformen el diagrama de classes, amb excepció de la classe *UmlComparation*, que ja em vist que no formava part, pròpiament dit, de la definició del diagrama.

Si afecta a tot els elements del diagrama, em de recordar que en aquest mateix capítol s'ha efectuat una modificació de l'estructura per agrupar tot el que era comú a totes les classes en una classe de la qual es deriven totes, que és *UmlModelElement*. Donat que disposem d'aquesta classe, el més lògic és que apliquem la modificació a aquesta classe.

Per aplicar sinònims relacionarem amb la classe UmlModelElement una classe que ens digui quins són els possibles sinònims de la classe.

Per tant, tornem a modificar el diagrama de classes i deixem-ho tal com segueix.

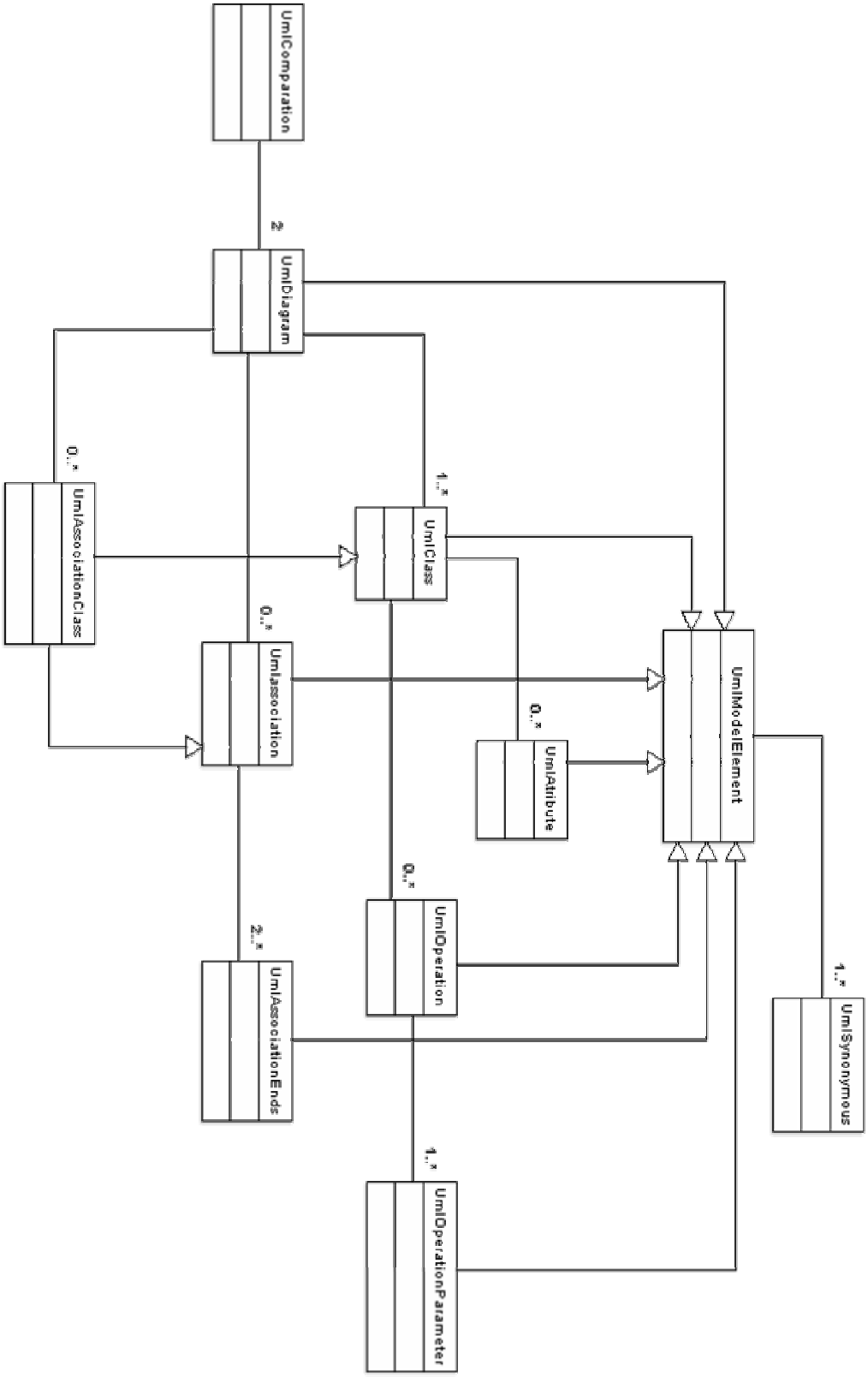


Figura 49. Diagrama de classes de comparació de models definitiu

Anem a veure amb una mica més de detall aquesta nova classe i com això modifica la classe *UmlModelElement*.

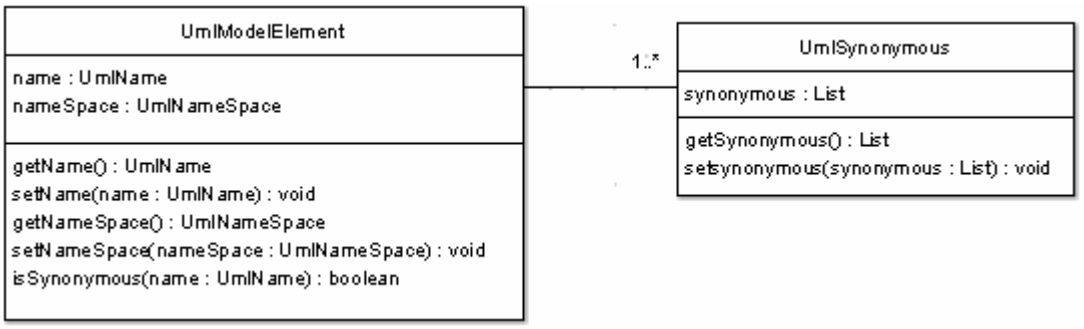


Figura 50. Classes UmlModelElement i UmlSynonymous

La nova classe, *UmlSynonymous*, únicament tindrà un atribut que serà una llista *d'UmlName's* que seran els sinònims possibles per a la classe. La llista de sinònims sempre tindrà com a mínim un element que serà el nom del propi element, així per saber si el nom proposat és un sinònim de la classe, en tindrem prou comprovant si aquest nom es torba dins de la llista de sinònims.

El programa no disposarà d'infinits sinònims, sinó que únicament atendrà a aquells que explícitament hagin estat marcats com a tals.

Capítol 6

Aplicació pràctica: Utilització en entorn web.

El projecte no quedaria complert si no s'hagués buscat alguna implementació del framework obtingut. Tenint en compte que el projecte està desenvolupat des de la Universitat Oberta de Catalunya (UOC) , s'ha triat una aplicació que pugui contribuït a millorar el nivell d'ensenyament, en alguns aspectes on la dificultat de l'ensenyament a distància queda més palès que en altres ocasions.

6.1 Definició de l'aplicació.

L'aplicació que s'ha decidit implementar té a veure amb el sistema d'ensenyament de la UOC, i vol fer incidència en la possibilitat de millorar el mètode d'ensenyament en el modelat d'aplicacions.

Aquest és un dels conceptes que presenta dificultat d'aprenentatge en l'ensenyament a distància, entès com la interpretació d'un enunciat textual per a convertir-lo en una definició de classes, associacions, ... que representin l'enunciat donat. S'ha detectat que hi ha una especial dificultat en identificar relacions ternàries, classes associatives, relacions recursives... arribant a la conclusió que l'única manera de poder assolir aquests conceptes és mitjançant la pràctica.

La idea que es planteja és estendre l'aplicació de comparació de models per oferir la funcionalitat següent:

- El professor prepara diversos problemes, entenent com a problema el conjunt format per un enunciat textual i el diagrama de classes que es considera com una solució correcte.
- A través de la web, l'estudiant veu els diferents problemes que ha preparat el professor i després de llegir l'enunciat adjunta el diagrama amb la seva solució.
- L'aplicació ha d'indicar a l'estudiant si ha resolt correctament el problema i en cas de no ser així donar la relació de diferències entre els models, fruit de la seva comparació.

Aquesta aplicació no és el cos principal del projecte, sinó que vol ser una aplicació del cos principal del projecte, que sí és la comparació de models, per tant el seu desenvolupament serà en el punt d'ampliació.

6.2 Casos d'us.

El cas d'us general de l'aplicació és el que es mostra en la figura 51 , on s'hi poden observar dos actors (professor i alumne) que corresponen als perfils que s'han detectat en l'apartat 6.1 de definició de l'aplicació.

Aquest dos actors tenen uns perfils en els que no comparteixen més tasques que la d'obtenir una llista dels problemes, per la resta cadascú té les seves tasques encara que aquestes es complementen, doncs l'estudiant no pot obtenir problemes fins que el professor a l'ha proposat i per tant tampoc en pot aportar una solució.

En la figura següent podem veure els casos d'us determinats per a l'aplicació que ens ocupa:

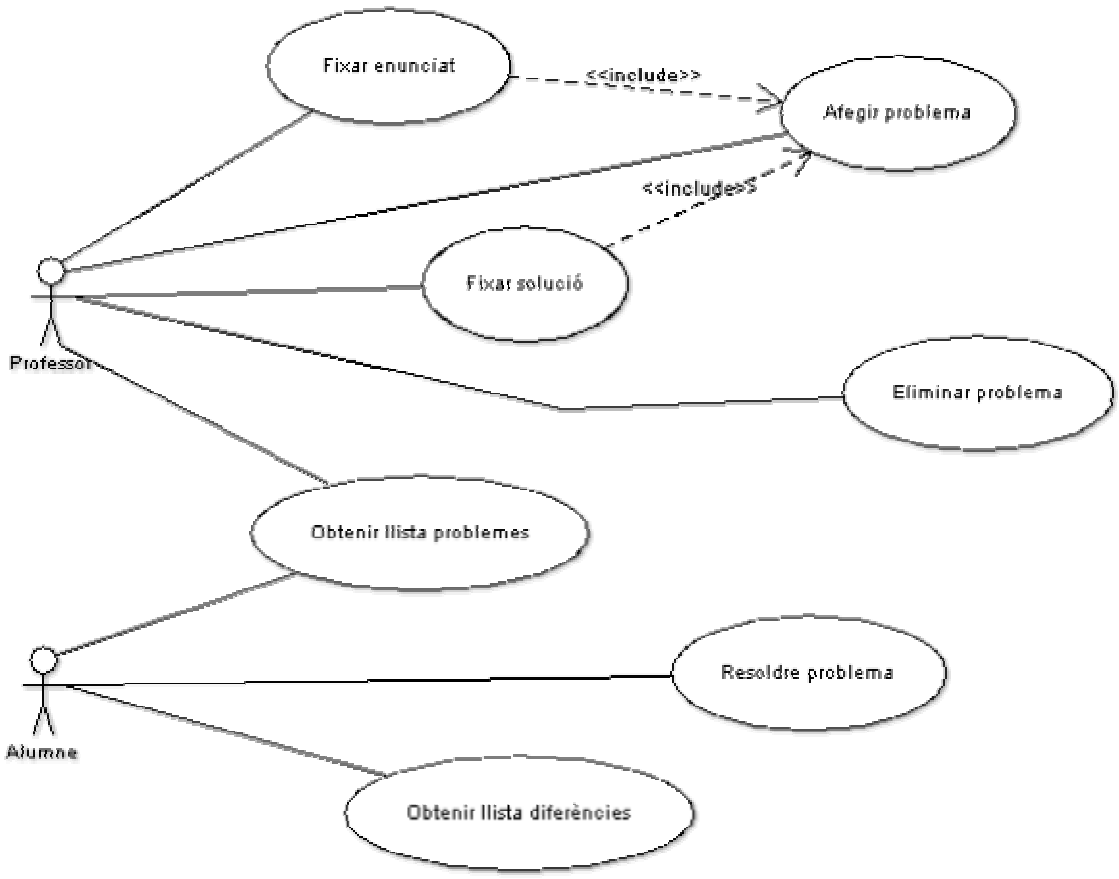


Figura 51. Cas d'us de l'aplicació web

D'aquest gràfic ens cal fer una descripció una mica més detallada de cadascun dels elements que el conformen. Comencem primer per la definció dels actors que seran els dos perfils que ens trobarem a l'hora de desenvolupar l'aplicació.

ACTOR	DESCRIPCIÓ
Professor	És la persona que s'encarrega de mantenir ela llista de problemes així com de fixar quins són els enunciats i quines són les solucions per a cadascun dels problemes.
Alumne	És el que aporta resolucions als problemes plantejats, perquè aquestes siguin validades contra les solucions que han fixat els professors.

Passem a continuació a descriure cadascun dels casos que es detallen en el gràfic, i que ens ajudaran a acabar de comprendre els requeriments i especificacions del sistema.

CAS D'US	DESCRIPCIÓ
Obtenir llista de problemes	És la llista de tots els problemes que tenen disponible un enunciat i la seva solució corresponent. La llista de problemes serà utilitzada tant pel professor com per l'alumne.
Fixar enunciat	És l'enunciat del problema que s'ha de resoldre. L'acció consisteix en pujar al servidor un fitxer de text, que estigui disponible per a ser visualitzat com a enunciat d'un dels problemes de la llista de problemes.
Fixar solució	És la solució a un problema determinat. Caldrà pujar al servidor un fitxer en format XMI, com a solució d'un dels problemes de la llista de problemes.
Afegir problema	Utilitzarem aquest cas d'ús quan calgui afegir un problema a la llista de problemes. Òbviament, al afegir problemes caldrà a l'hora fixar un enunciat i una solució per al problema que s'acaba d'afegir.
Eliminar problema	Utilitzarem aquest cas d'ús quan calgui treure un problema de la llista de problemes.
Resoldre problema	Després que l'alumne hagi llegit l'enunciat, crearà la solució en el seu ordinador, obtenint-ne una transformació de la mateixa en format XMI. Aquest fitxer serà pujat al servidor per a ser comparat amb la solució fixada pel professor.
Obtenir llista de diferències	Una vegada l'alumne hagi pujat la solució al servidor, aquesta serà comparada amb la solució del professor, i s'obtindran la llista de diferències entre ambdós

CAS D'US	DESCRIPCIÓ
	models, si és que n'hi ha, en cas contrari s'obtindrà la indicació que la solució és correcte.

6.3 Diagrama de classes.

El diagrama de classes resultant del casos d'us és extremadament senzill, donat que únicament resulta una classe que ens representa el problema, que té la representació que podem veure en la figura 51.

A comentar que disposa de tres atributs per guardar el títol del problema, el path de l'enunciat i el path de la solució. Òbviament disposa dels "getter's" i "setter's" corresponents, però també disposa d'un mètode *toXML*, que mereix la pena fer-ne un petit comentari.

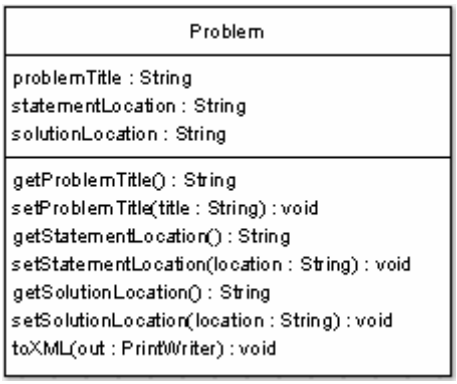


Figura 52. Classe Problem

Aquest mètode apareix per poder serialitzar l'objecte, doncs l'aplicació per la seves característiques no justifica l'esforç d'aplicar-hi un sistema de bases de dades, per tant he pensat que el millor seria serialitzar la informació en un fitxer XML, per la qual cosa es imprescindible disposar d'un mètode que transformi l'objecte a un format XML.

Com a conseqüència d'aquesta decisió ens caldrà disposar de classes que siguin capaces de gestionar la serialització, dit en altres paraules, el nostre sistema gesto de

dades. Per això crearem una classe que estigui encarregada d'aquestes tasques, tal com es veu en la figura 53.

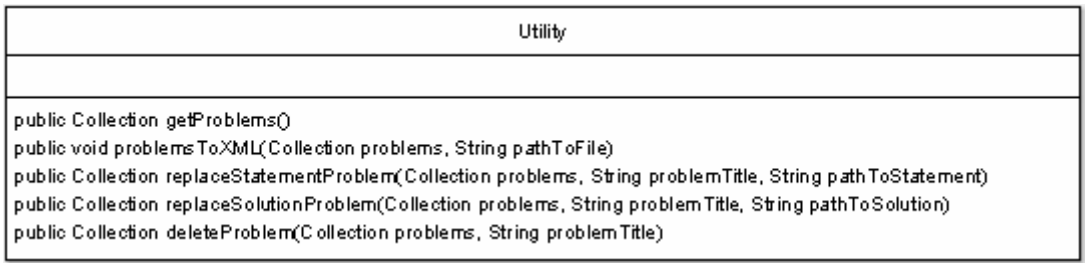


Figura 53. Classe Utility

6.4 Arquitectura de la web.

Donat que l'aplicació desenvolupada per posar en pràctica el framework elaborat és una aplicació web, és fa necessari explicar quin tipus d'arquitectura s'ha triat per portar a terme l'aplicació.

El model triat ha estat el Model-View-Controller (MVC) que es basa en que totes les peticions s'efectuen a un controlador que té com a missió cridar a les classes del model de negoci, necessàries per a portar a terme les accions sol·licitades i just abans d'acabar passa el control a la part de vistes que s'encarreguen de portar el resultat al peticionari de l'accó.

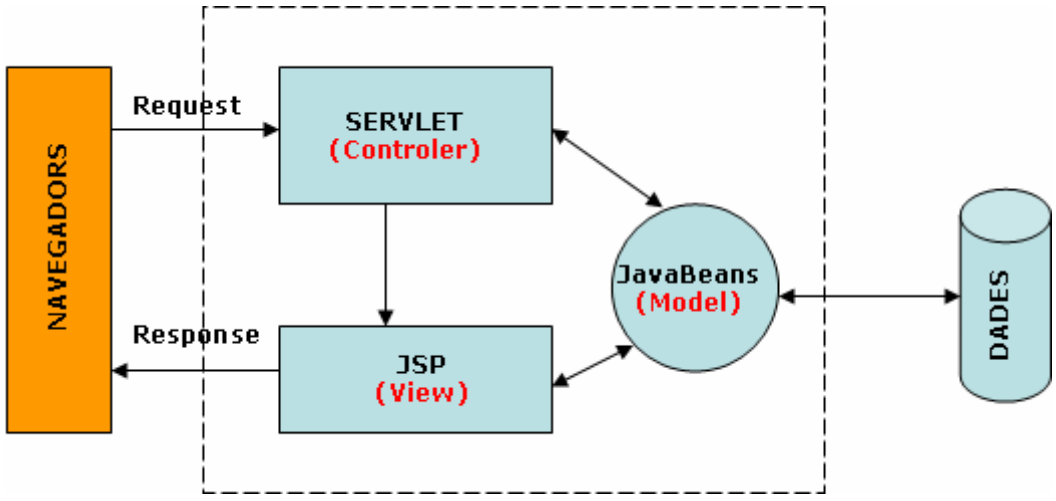


Figura 54. Diagrama Model-View-Controller

En el model Java en que s'ha desenvolupat el projecte són els servlet's els que s'encarreguen de realitzar les tasques de controlador, mentre que la tasca del model de negoci es encomanat als JavaBeans i el control de vistes s'efectua mitjançant planes JSP.

Tal com ja s'ha explicat, un servlet és una classe Java que estén javax.servlet.http.HttpServlet, i que rep les peticions efectuades pel navegador i és acaba enviar el control a un recurs de l'aplicació que en el nostre cas serà una plana JSP. Durant la seva execució pot incloure en la resposta tots aquells paràmetres necessaris per a portar variar el comportament del les planes JSP en temps d'execució, de tal manera que, com es veurà en el figura 55 una mateixa plana pot ser el resultat de diverses accions i presentar diferents parts en funció del contingut de la resposta.

La figura 55 mostra els servlets i les planes JSP, així com la manera en que es desenvolupa el flux de la web.

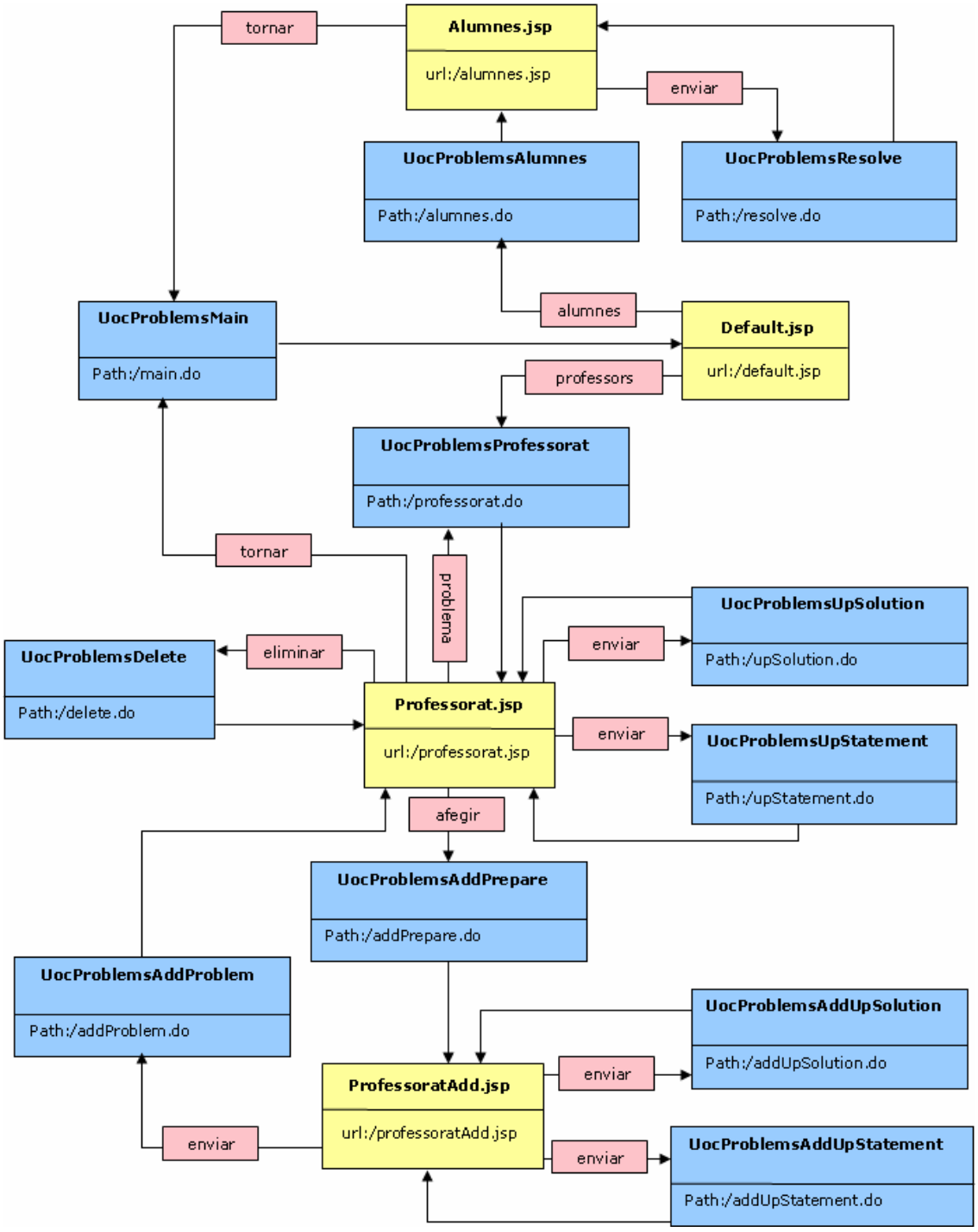


Figura 55. Diagrama de seqüència aplicació web

Per entendre una mica millor la figura precedent, direm que els quadres de color blau representen els servlets o accions que es poden sol·licitar a l'aplicació. Els cuadres grocs, representen el les planes JSP que s'enviaran cap al navegador, mentre que els quadres de color vermell, representen la comanda que s'ha d'executar per iniciar l'acció.

6.5 Implementació.

6.5.1 Distribució de paquets.

La distribució de paquets s'ha fet d'acord amb la distribució efectuada per l'aplicació de comparació de models, tal com s'ha explicat en el capítol 4 d'aquesta memòria.

Així la distribució de paquets ha quedat de la següent manera:

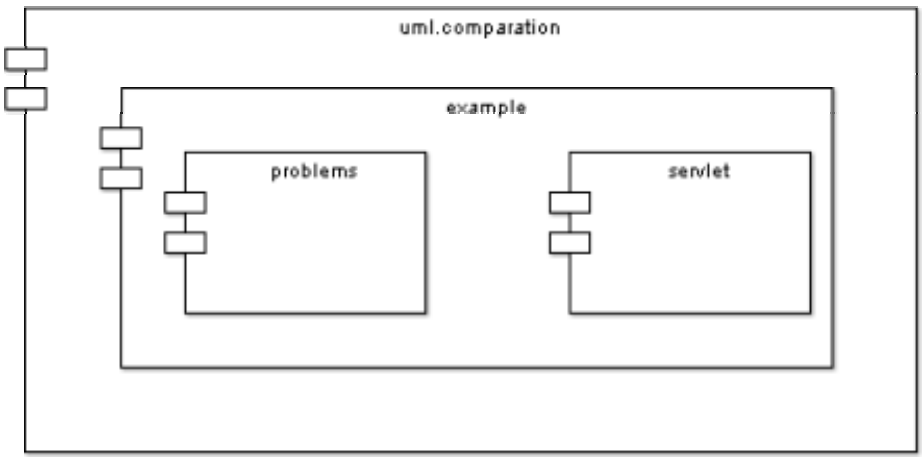


Figura 56. Distribució de paquets de l'aplicació web

En el paquet *problems* hi posarem les classes que fan referència al model de negoci, mentre que al paquet *servlet* hi possarem tots els servlets que responen a totes les accions possibles

.

6.5.2 Consideracions generals.

La implementació de l'aplicació s'ha efectuat sobre un servidor Tomcat 5.5, tal com ha quedat esmentat en el capítol 1 d'aquesta memòria. El primer que es va tenir que fer va ser configurar el servidor amb les llibreries necessàries per a que funcionés l'aplicació és a dir les llibreries proporcionades per DresdenOcl i les que s'acabaven de generar en la primera fase del projecte.

Per a fer això ens calen posar les llibreries en el directori <TOMCAT_INSTALLATION>\common\lib , doncs mentre que fins al moment havíem estat treballant en local, a partir d'aquest moment les peticions es faran a un servidor i serà aquest l'encarregat de resoldre les accions.

L'estructura de directoris de l'aplicació ha quedat tal com es veu a la figura, que és una captura de pantalla de l'aplicació ja desenvolupada

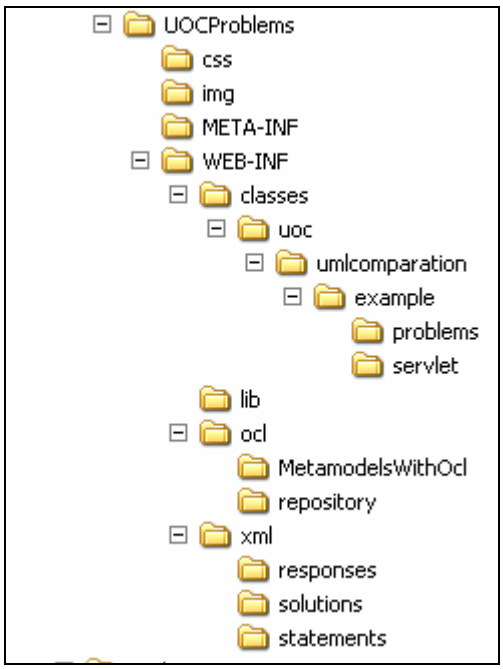


Figura 57. Distribució de carpetes aplicació web

Respecte de l'estructura de directoris, crec que és important fer les següents consideracions:

- La carpeta *css* conté la fulla d'estils, que no es altre que la que aplica en aquests moments la UOC per a la seva web.
- La carpeta *img* conté les imatges que s'utilitzen en la web, que també son extretes de les imatges que utilitza la UOC.
- A la carpeta *classes* hi trobarem totes les classes compilades amb els seus respectius paquets.
- La carpeta *ocl* és una carpeta necessària per a la comparació de diagrames, utilitzada com a temporal per la llibreria *DresdenOcl*.
- La carpeta *xml* conté el repositori de dades, fet així, tal com s'ha explicat en anteriors apartats d'aquests capítol per la poca importància de les dades en l'aplicació desenvolupada.

6.5.3 Problemàtica durant la implementació.

He volgut destacar aquest apartat perquè durant la implementació de l'aplicació es va presentar un problema força important, si més no amb el temps emprat en resoldre'l, que va ser la utilització de les llibreries DresdenOcl.

El problema era el següent. Mentre que la utilització d'aquestes llibreries en l'aplicació de comparació de models, desenvolupada en un entorn local no presentava cap problema, quan aquesta aplicació era invocada en un entorn web, es generava un error per malformació de la url de les carpetes temporals utilitzades per les llibreries DresdenOcl durant la transformació del fitxer XMI a objectes Ocl.

La investigació de l'error em va portar, donat que DresdenOcl són llibreries de codi lliure, a fer una recerca de l'error a través del codi d'aquestes llibreries i vaig detectar que l'error és produïa a la classe *tudresden.ocl20.core.MetaModelConst.java* en el mètode *getIntegratedXmiPath()* que estava definit de la següent manera:

```
public URL getIntegratedXmiPath(){
  try{
     URL outputDir = ClassLoader.getSystemClassLoader().getResource(
                          MetaModelConst.METAMODELSWITHOCLDIR+java.io.File.separatorChar);
      return new URL(outputDir, name+"_plus_OCLMetamodel.xml");
     } catch (Exception e){
```

```
        e.printStackTrace();
        return null;
      }
}
```

Per a poder solventar aquest entrebanc vaid tenir que redifinir aquest mètode de la següent manera:

```
public URL getIntegratedXmiPath(){
 try{
     String resource = metamodelsLoc+java.io.File.separatorChar+
                       MetaModelConst.METAMODELSWITHOCLDIR+java.io.File.separatorChar;
     String cami = resource+name+"_plus_OCLMetamodel.xml";
     return new File(cami).toURL();
    } catch (Exception e){
       e.printStackTrace();
       return null;
     }
}
```

Aquesta modificació va permetre que l'aplicació trobés correctament el path als directoris temporals i l'aplicació va funcionar correctament a partir d'aquest moment. Una vegada efectuada aquesta modificació va ser necessari recompilar les llibrerires DresdenOcl i substituir-les a la carpeta *common/lib* del servidor d'aplicacions.

6.6 Demostració.

Una vegada efectuat tant l'anàlisi com el desevolupament de l'aplicació, per acabar aquest capítol ens cal fer una petita demostració, mitjançant captures de pantalles, per poder veure quin ha estat el resultat final de tota l'apliació.

6.6.1 Pantalla d'inici.

La primera pantalla l'utilitzem com a punt comú d'inici on hi trobem els perfils que s'han descrit durant l'anàlisi, que són el del professor i el del alumne.

Queda fora de l'abast del present projecte l'estudi i implantació de seguretat a les aplicacions, per tant aquí es presenten els perfils accessibles amb un sol enllaç. Per una

aplicació en un entorn real caldria, abans que rés, aplicar una política de seguretat, i per tant demanar l'usuari amb la seva contrasenya.

Fetes aquestes consideracions, podem passar a veure la pantalla inicial que queda tal com es mostra en la figura 58, on hi podem observar que els textos "*Professors*" i "*Alumnes*" són enllaços a les planes d'operativa dels respectius perfils.

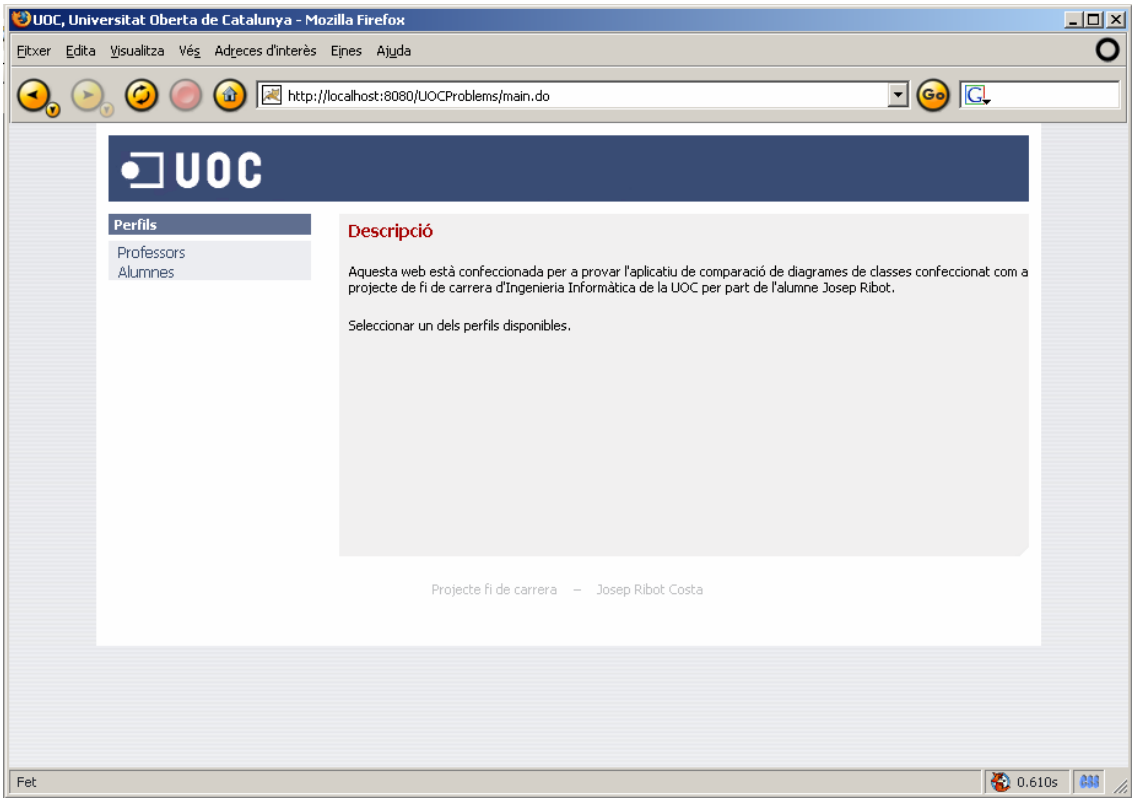


Figura 58. Pantalla d'inici de l'aplicació web

6.6.2 Perfil de professor.

Quan accedim al perfil trobem a l'esquerra una llista dels problemes que estan publicats, i a sota d'aquets les opcions que es poden prendre a partir d'aquesta pantalla que són :

- Afegir: Afegir un altre problema
- Eliminar: Borrar un problema ja publicat
- Tornar: Retorna al menu d'inici

A la part dreta hi trobarem l'enunciat del problema seleccionat, el primer en el supòsit que no hagim seleccionat cap problema, i en aquest apartat hi trobem dos opecions:

- Canviar l'enunciat : canviar el texte per un altre.
- Canviar la solució: canviar el fitxer XMI per un altre.

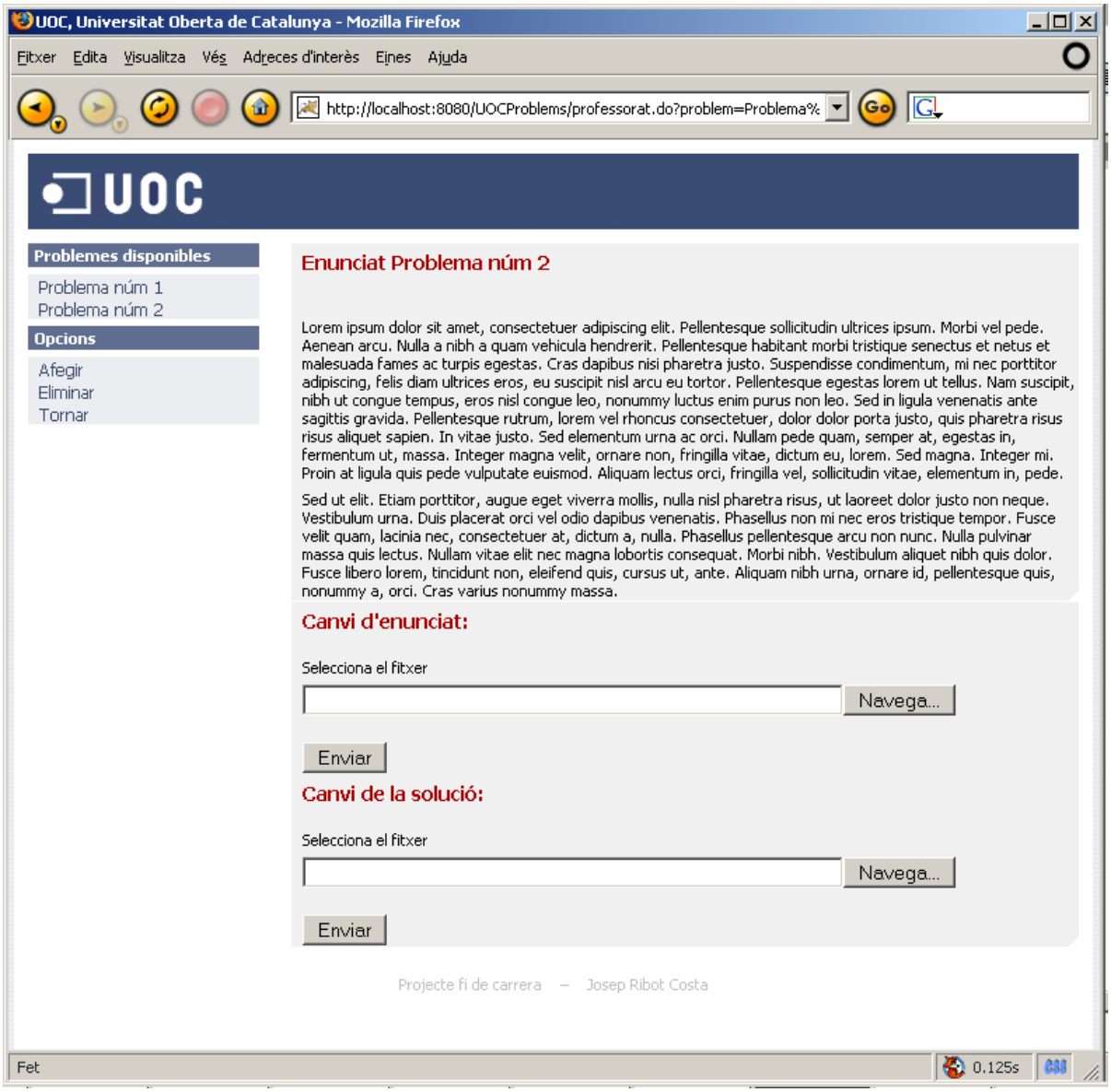


Figura 59. Perfil de professorat

Si premem el link per afegir un nou problema, ens apareixerà la pantalla que es pot observar en la figura 60 , on haurem de seguir la següent operativa:

- 1r. Carregar el fitxer que conté l'enunciat del problema
- 2n. Carregar el fitxer XMI que conté a solució del problema
- 3r. Donar un títol al problema que és el que apareixerà en la part esquerra de la pantalla com a problema disponible.
- Premer el butó Afegir.

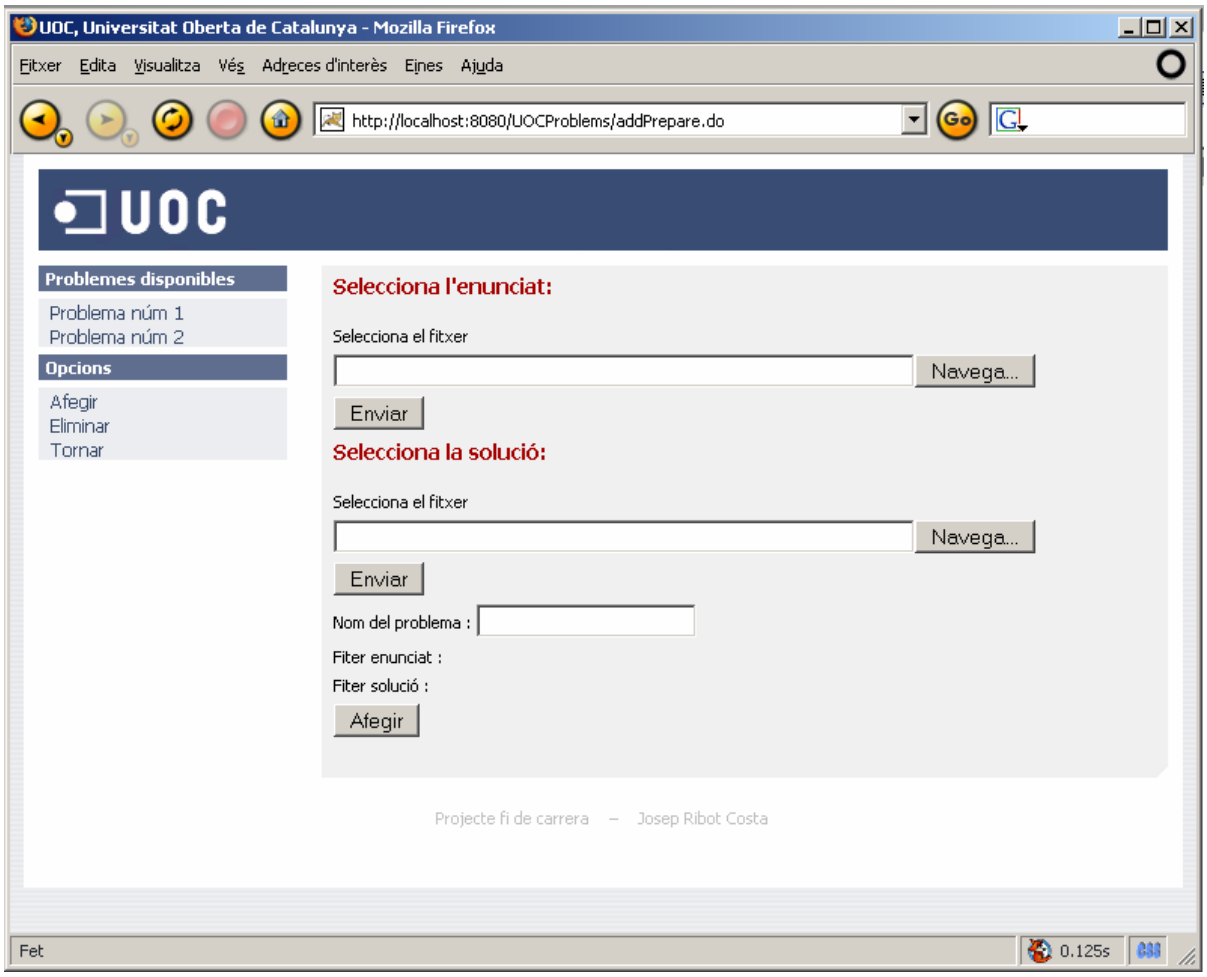


Figura 60. Perfil de professorat. Afegir problemes

6.6.3 Perfil de l'alumne.

La visió que té alumne de l'aplicació és semblant a la que té el professor però amb menys opcions. A l'igual que el professor el que veu l'alumne a la part esquerra és una llista dels problemes que hi ha publicats i a sota d'aquests les opcions disponibles que en el cas dels alumnes és únicament la de tornar a la pantalla d'inici.

A la part dreta hi trobem l'enunciat del problema seleccionat, que serà l'enunciat del primer problema en el supòsit que no n'haguem seleccionat cap. Ha sota de l'enunciat hi trobem la possibilitat de pujar al servidor una solució al problema perquè efectuï una comparació amb la solució que ha deixat el professor per a cada problema.

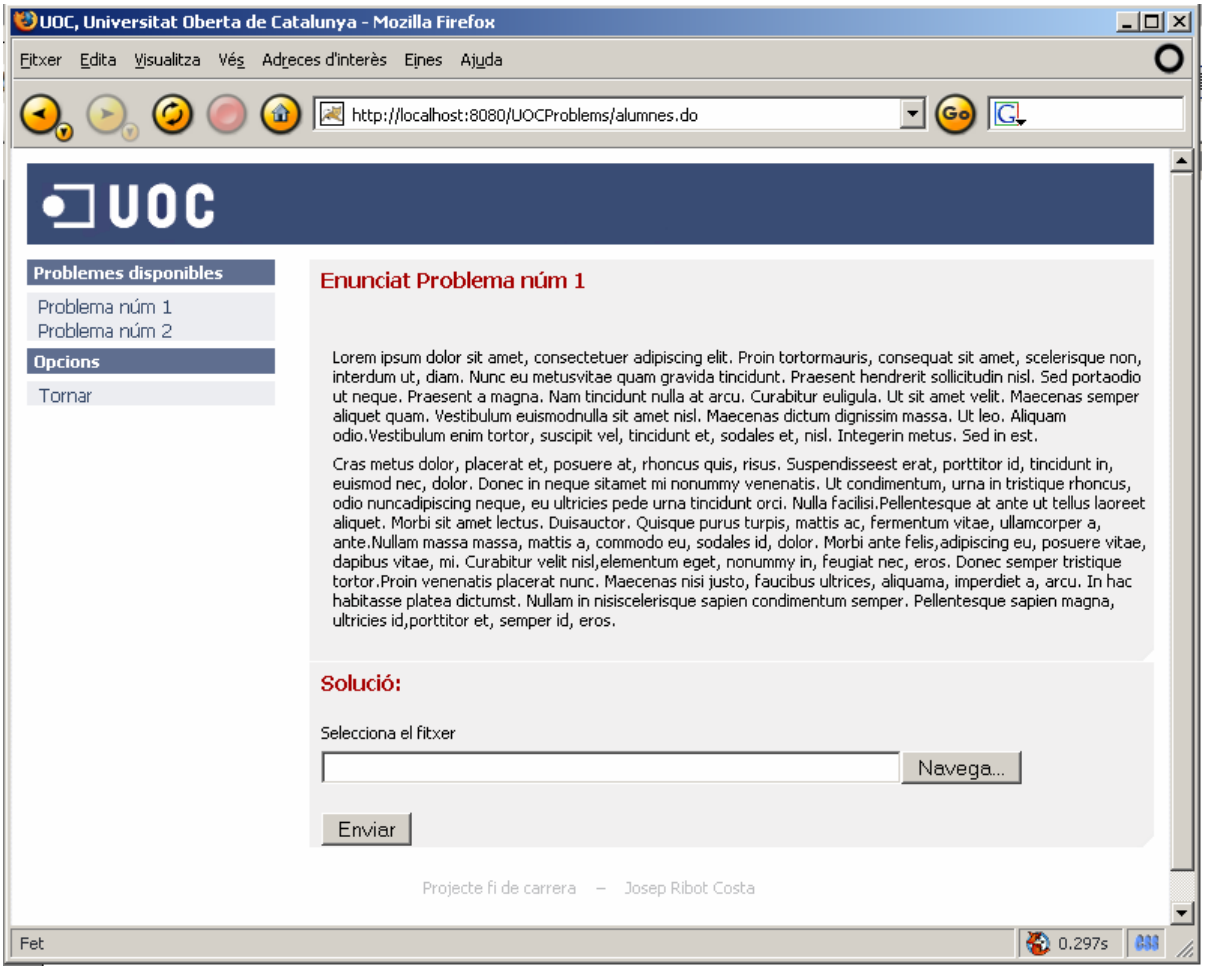


Figura 61. Perfil de l'alumnat

Per iniciar la comparació en tenim suficient en localitzar el fitxer de resolució i prèmer “Enviar”.

L’aplicatiu s’encarrega de comparar el fitxerde resolució amb la solució oficial i en el cas de que hi hagin diferència entre ambdós solucions, mostrarà quines són aprofitant la classe d’error *UmlErrorElement*, que ens permetrà detectar clarament on s’han produït les diferències.

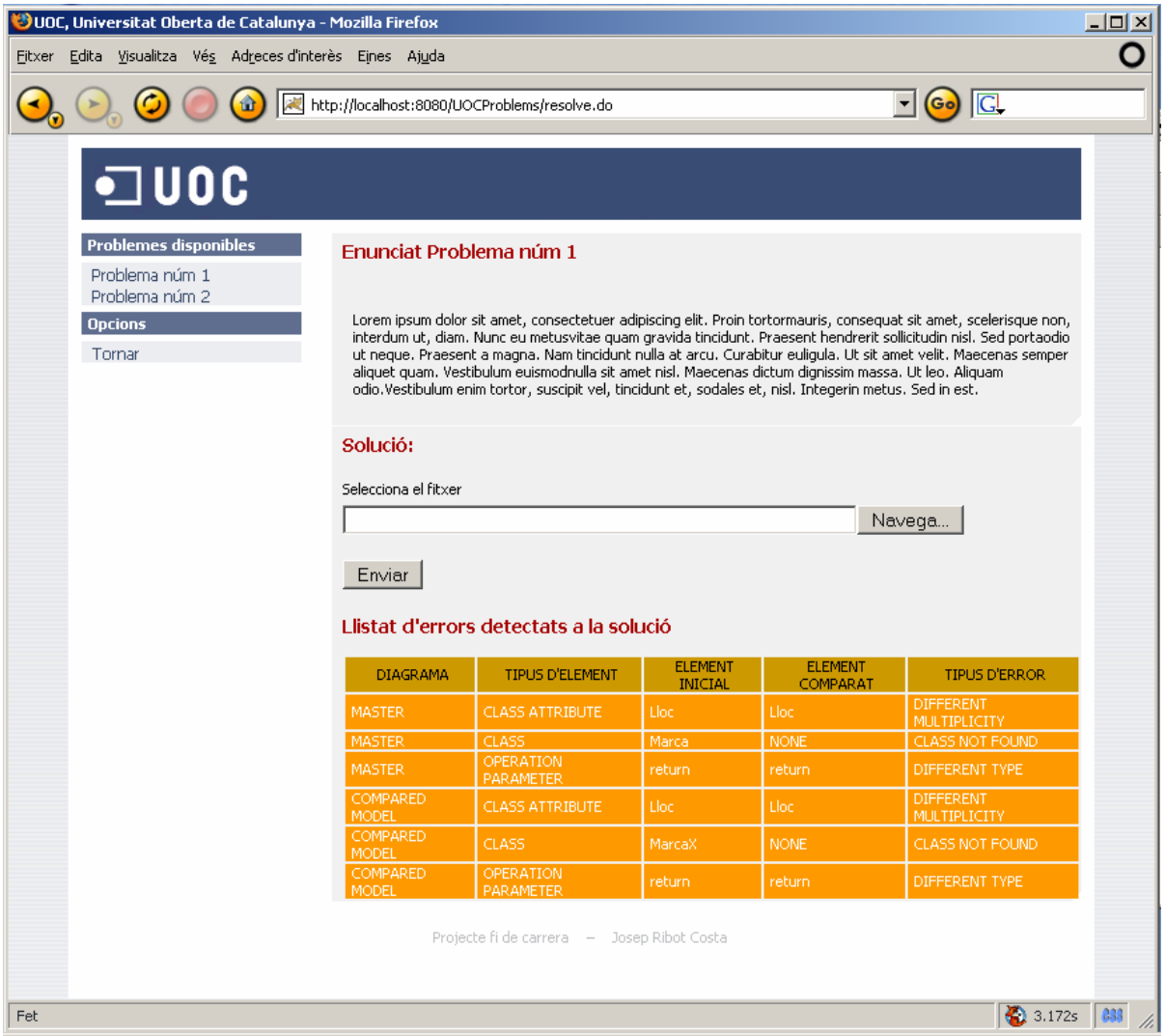


Figura 62. Perfil de l’alumnat. Resolució de problemes

Conclusions

Ha estat una temporada dura de treball, com cap semestre a la UOC, però al final estic content perquè he fet coses de les quals he pogut aprendre, en definitiva m'he enriquit professionalment.

Però també crec que he fet un treball que si algú el reprèn, no haurà de tornar a partir de zero, doncs el projecte contribueix a desenvolupar un framework per a fer comparacions de models, que al menys des del punt de vista educatiu segur que pot tenir una sortida al mon real. El projecte demostra que és pot acostar la interactivitat en l'aprenentatge a distància, millorant conceptes educatius de difícil resolució a traves d'explicacions en paper.

Evidentment que una vegada immers en el projecte, observes que encara es poden fer moltes coses més que ampliarien i millorarien el projecte, però el temps es finit, al menys per a l'execució d'aquest projecte.

Glossari

Algoritme : És un conjunt finit de passos que serveixen per executar una tasca o resoldre un problema.

Aplicació de lliure distribució: Codi generat que es permet que sigui modificat i redistribuït per tercers diferents a l'autor.

Aplicació web: Programa que s'executa en un servidor d'aplicacions, accessible a través de navegadors.

Argo UML: Programa de lliure distribució per al desenvolupament d'anàlisi d'acord amb especificacions UML.

Associació: Relació existent entre dos o més classes en un diagrama estàtic.

Associació binària: Relació entre exactament dos classes d'un diagrama estàtic.

Association Class: Classe que qualifica una associació.

Association End: Representen els extres de les associacions.

Atribut: Element d'una classe que representa un tipus de dada relacionat amb la classe.

Cas d'us: Diagrama UML per a explicar funcionalment una aplicació

Classe: Representació d'un objecte.

Classifier: Superclasse de l'especificació UML

Comunity Edition: Versió d'un programa que no es pot utilitzar per a usos comercials.

Diagrama de distribució: Representa com es distribueixen les classes d'una aplicació.

Diagrama d'estat: Representa els diferents estats en que es pot trobar un objecte.

DresdenOcl: Projecte de codi lliure per al tractament d'OCL.

Eclipse: IDE de programació per a diferents plataformes, entre elles Java.

Espai de noms: veure package.

Framework: Conjunt de programes que actuen com una unitat, amb funcionalitats accessibles sense necessitat de conèixer el seu contingut.

Getter: Mètodes de les classes que s'utilitzen per a obtenir els valors dels atributs.

Instància: Objecte creat en base a una classe definida.

Java: Llenguatge de programació

Llibreria: Conjunt de programes agrupats que serveixen a una finalitat conjunta.

Metadata: Informació descriptiva associada a un recurs.

Mètode: Operació d'una classe.

Multiplicitat: Nombre d'ocurrències d'un recurs.

OCL: acrònim de Object Constraints Language, que és una part d'UML

OMT: Acrònim d'Object Modeling Language.

On-line: en directe, al moment.

OOSE: Acrònim de Object Oriented Software Engineering

Operació: veure mètode.

PAC: Prova d'Avaluació Continuada a la UOC.

Package: directori amb els que es distribueixen les classes.

Paquet: veure package.

Patrons: Estructures de programació establertes que resolen problemes comuns.

Pla de treball: Planificació de l'execució del projecte.

Poseidon: Programa comercial per a la creació i manipulació de diagrames UML.

Setter: Mètodes de les classes que s'utilitzen per a donar valors als atributs.

String: Classe estàndard de Java que proporciona tractament sobre cadenes de caràcters.

Tomcat: Servidor d'aplicacions de lliure distribució.

TotoiseCVS: Programari de control de versions.

UML: Unified Modeling Language. Llenguatge per a l'anàlisi i definició d'aplicacions.

UOC: Universitat Oberta de Catalunya.

url: Acrònim d'Uniform Resource Locator. Una URL és una sequència de caràcters que identifica de forma unívoca un recurs a la web. També es coneix com 'adreça web' o 'identificador de recursos web'.

War: Acrònim de Web Application aRchive. Es tracta d'un fitxer comprimit que conté una aplicació web, instal·lable a qualsevol servidor d'aplicacions Java.

Web: veure world wide web.

World Wide Web: La World Wide Web és un sistema d'hipertext que funciona sobre la xarxa física Internet i que possibilita l'accés a infinitat de recursos mitjançant identificadors específics.

XMI: Acrónim de XML Metadata Language.

XML: Acrònim d'eXtensible Markup Language.

Referències

OMG Unified Modeling Language Specifications. March 2003 . Version 1.5

Java 2 Platform Standard Editon 5.0. API Specification.

http://www.tortoisecvs.org/index.shtml

http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops/R-3.1-200506271435/org.eclipse.jdt.doc.user.3.1.pdf.zip

http://dresden-ocl.sourceforge.net/usageexample.html

The Parser Subsystem of the Dresden OCL2 Toolkit. Ansgar Konermann

A Metamodel-Based OCL-Compiler for UML and MOF. Sten Loecher, Stefan Ocke

http://argouml.tigris.org/documentation/defaulthtml/manual/

http://tomcat.apache.org/tomcat-5.5-doc/index.html