Estudi d’eines CASE que suporten OCL per a la generació automàtica de codi Java

Antoni Garriga Rovira
ETIG

Anna Queralt Calafat

Dilluns, 14 de gener de 2008

Pàgina en blanc expressament

Dedicatòria i agraïments

Dedico el treball a tots els que com jo pensen que "*No tot és bufar i fer ampolles*" però que "*De mica en mica, s'omple la pica*" de la mateixa manera que "*Roma no es va fer en un dia*".

Vull agrair...

... a la UOC pel seu sistema i mitjans, tan materials com humans, que m'han permès poder compaginar feina i estudis, i després de set anys de molta dedicació als estudis, poder presentar aquest treball.

... a l'Anna Queralt, la meva consultora d'aquest treball, al guiatge continu, els bons consells, i els ànims que no han faltat mai i que m'han motivat a seguir i poder fer aquest treball.

... a la família, per la paciència que han tingut per no poder estar més per ells degut al cost d'oportunitat que representen els estudis.

Resum del treball

Amb aquest treball ens introduïm al coneixement de l'OCL (Object Constraint Model), que és un llenguatge textual que ens permet especificar d'una manera no ambigua totes les restriccions necessàries per dotar els diagrames UML d'una semàntica completa. Amb l'OCL podem expressar tot allò que no es pot especificar gràficament en un diagrama UML, des de les restriccions del model fins a consultes d'aquest.

Com que cada vegada hi ha més eines CASE que generen automàticament el codi en alguns llenguatges de programació a partir de la representació gràfica en UML, i com que l'OCL està prenent cada vegada més importància, he realitzat una recerca d'eines CASE que actualment suporten OCL en la generació automàtica de codi Java per estudiar-les i analitzar-les a través d'un model de proves consistent en un diagrama de classes del model estàtic de l'UML i una mostra variada d'instruccions OCL, amb l'objectiu de detectar les seves mancances, analitzant el codi obtingut i determinar si controla o no cada tipus de restricció, i si s'han implementat bé en el codi.

Per a cada una de les eines seleccionades per analitzar n'he fet una explicació prèvia i no massa extensa en la qual es descriuen les principals característiques, s'hi indica a on i com podem aconseguir el programari, en dono una guia de com instal·lar-lo i com s'utilitza, i posteriorment l'analitzo a través del model de proves, i finalment en faig un resum concloent de l'eina.

I per acabar trobem unes taules comparatives entre les diferents eines analitzades i les conclusions finals d'aquest treball.

Paraules clau

CASE
Computer-Aided Software Engineering
Generació de codi
Java
Object Constraint Language
OCL
UML
Unified Modeling Language

Àrea del TFC

Generació automàtica de programari

Índex de continguts

Índex de figures

Memòria del treball

1. Introducció

L'OCL (Object Constraint Model) és un llenguatge formal per descriure expressions en els models UML i que ens permet especificar d'una manera no ambigua totes les restriccions necessàries per dotar els diagrames del model UML d'una semàntica completa. Amb l'OCL podem expressar tot allò que no es pot especificar gràficament en un diagrama UML, des de les restriccions del model fins a consultes d'aquest.

Si bé podem trobar moltes eines CASE (Computer Aided Software Engineering) que ens poden ajudar per la generació automàtica de programari a partir de la seva especificació, per exemple podem trobar eines CASE que a partir d'un diagrama de classes UML generin el codi SQL d'una base de dades o el codi d'un llenguatge de programació orientat a objectes com pot ser el Java, trobem que no totes les eines suporten OCL (Object Constraint Language) per poder representar les restriccions que volem tenir al codi generat i així obtenir un codi el més complert possible.

En aquest TFC ens centrarem en l'anàlisi d'eines CASE que suporten OCL per l generació automàtica de codi Java, per analitzar fins a quin punt aconsegueixen generar el codi el més complert possible, segons les restriccions prèviament expressades.

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

L'OCL està prenent cada vegada més importància, i és per aquest motiu que actualment existeixen diverses eines que ja utilitzen les restriccions expressades en aquest llenguatge per tal de generar un codi més complert.

Partint d'un desconeixement inicial de què és l'OCL i de quines eines el suporten o utilitzen per a la generació automàtica de codi, a través del treball anem aprenent els nous conceptes d'aquest llenguatge, tals com la sintaxi i les possibilitats d'aquest llenguatge formal que complementa a l'UML. Després sobre algunes eines CASE que generen codi Java a partir de models UML i que suporten OCL, ens documentem sobre aquestes, en seleccionem algunes i les obtenim i instal·lem. Aplicant un model de proves, les analitzem amb detall per veure si realment en el codi Java generat hi ha mancances o no segons les instruccions OCL aplicades.

1.2. Objectius del TFC

L'objectiu general del TFC és l'anàlisi de diferents eines CASE que suporten l'OCL per a la generació automàtica de codi Java. I els objectius específics són aprendre el llenguatge formal OCL, i realitzar un treball de recerca d'eines actuals CASE, analitzant el seu comportament en la generació automàtica de codi Java, detectant el seu compliment o bé les mancances.

1.3. Enfocament i mètode seguit

Inicialment vaig preveure i avaluar totes les tasques a fer i en vaig fer 4 grups principals per facilitar el treball i així, i en determinats casos, poder treballar en paral·lel amb tasques de diferents grups. Aquests grups són :

- OCL. Aprenentatge del llenguatge per especificar les restriccions.

- Eines CASE. Recerca, selecció i aprenentatge de les mateixes.
- Anàlisi de les eines CASE treballant amb les restriccions en OCL.
- Documentació a elaborar.

Cada grup de tasques l'he dividit en subtasques. El conjunt de tasques i subtasques previst ha estat el següent:

1. OCL (Object Constraint language)

- **1.1.** Recerca d'informació sobre OCL.
- **1.2.** Aprenentatge d'OCL per entendre'l i utilitzar-lo.
- **1.3.** Model o joc de proves. Model UML i instruccions OCL sobre el mateix.

2. Eines CASE

- **2.1** Recerca d'eines actuals CASE que suportin OCL, anotant característiques.
- **2.2** Classificar i documentar les eines trobades.
- **2.3** Elecció d'eines per analitzar.
- **2.4** Adquisició i instal·lació de les eines per analitzar.
- **2.5** Aprenentatge bàsic de les eines.

3 Anàlisis de les eines CASE amb OCL (Per a cada eina a estudiar)

- **3.1** Aplicació del test de proves.
- **3.2** Revisions del codi generat.
- **3.2** Informe de resultats.

4 Documentació

- **4.1** Documentar les diferents parts de les tasques que es van fent.
- **4.2** Elaborar el document de la fita PAC2.
- **4.3** Elaborar el document de la fita PAC3.
- **4.4** Elaborar la memòria del TFC.
- **4.5** Elaborar la presentació del TFC.

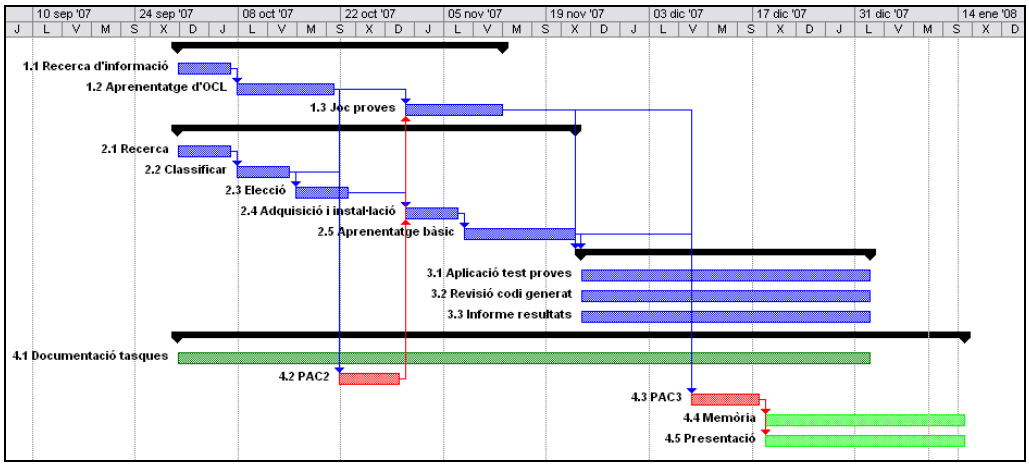
1.4. Planificació del projecte

Les fites de les PACs, vistes com unes entregues parcials del treball, i l'entrega final de la memòria i presentació, concretades en dies determinats i prefixats, en marquen unes dates concretes en el temps que cal complir. També hi ha tasques que es poden concretar en el temps, ja siguin parts prèvies o posteriors a altres, però n'hi una d'important que s'ha d'anar fent mentre dura el treball, que és la documentació de les diferents parts (4.1).

Al final de les tasques dels grups 1 i 2, que és la part de l'aprenentatge OCL i de les eines CASE, arribem a l'anàlisi de les eines (grup 3) on tindrem que les subtasques s'executaran repetidament per a cada una de les eines, serà com un cicle repetitiu a fer a cada una de les eines. Passats uns dies es preveu ja l'elaboració dels documents a entregar del TFC com són la memòria i la presentació, tot i que continuarem en paral·lel la feina d'anàlisis fins a la primera setmana de l'any 2008.

Reflectint-ho amb dates concretes i amb l'ajuda d'un programari, podem mostrar-ho amb el diagrama de Gantt següent:

	Nombre de tarea	Duración	Comienzo	Fin	Predec
1	**- 1. OCL (Object Constraint lang**	**45 días**	**sáb 29/09/07**	**lun 12/11/07**	
2	1.1 Recerca d'informació	8 días	sáb 29/09/07	sáb 06/10/07	
3	1.2 Aprenentatge d'OCL	14 días	dom 07/10/07	sáb 20/10/07	2
4	1.3 Joc proves	14 días	mar 30/10/07	lun 12/11/07	3;17
5	**- 2. Eines CASE**	**55 días**	**sáb 29/09/07**	**jue 22/11/07**	
6	2.1 Recerca	8 días	sáb 29/09/07	sáb 06/10/07	
7	2.2 Classificar	8 días	dom 07/10/07	dom 14/10/07	6
8	2.3 Elecció	8 días	lun 15/10/07	lun 22/10/07	7
9	2.4 Adquisició i instal·lació	8 días	mar 30/10/07	mar 06/11/07	8;17
10	2.5 Aprenentatge bàsic	16 días	mié 07/11/07	jue 22/11/07	9
11	**- 3. Anàlisis**	**40 días**	**vie 23/11/07**	**mar 01/01/08**	**4;10**
12	3.1 Aplicació test proves	40 días	vie 23/11/07	mar 01/01/08	
13	3.2 Revisió codi generat	40 días	vie 23/11/07	mar 01/01/08	
14	3.3 Informe resultats	40 días	vie 23/11/07	mar 01/01/08	
15	**- 4. Documentació**	**108 días**	**sáb 29/09/07**	**lun 14/01/08**	
16	4.1 Documentació tasques	94,44 días	sáb 29/09/07	mar 01/01/08	
17	4.2 PAC2	9 días	dom 21/10/07	lun 29/10/07	3;7
18	4.3 PAC3	10 días	sáb 08/12/07	lun 17/12/07	4;10
19	4.4 Memòria	28 días	mar 18/12/07	lun 14/01/08	18
20	4.5 Presentació	28 días	mar 18/12/07	lun 14/01/08	18



Il·lustració 1: Diagrama Gantt de la planificació del treball

La planificació, en el conjunt de tasques i subtasques previst, s'ha mantingut durant el projecte, tot i que sí que s'ha hagut d'ajustar alguns terminis segons ha anat evolucionant, i segons les complicacions trobades i la disponibilitat de temps dedicat. Cal comentar que l'aprenentatge d'OCL no s'ha fet només durant la tasca 1.2, sinó que s'ha anat complementant durant tot el treball.

1.5. Productes obtinguts

La temàtica d'aquest TFC es basa en l'aprenentatge d'OCL i l'anàlisi i estudi d'eines del mercat que el suporten per generar codi Java, i no inclou cap desenvolupament de programari i per tant aquesta memòria del treball és l'únic producte resultant i peça central del TFC.

1.6. Apartats de la memòria

La memòria s'ha dividit en varis capítols que seguiran l'ordre temàtic següent:

- Descripció i exemples d'ús de l'OCL.
- Eines CASE que suporten OCL.
- Creació del model de proves per l'estudi de les eines.
- Anàlisi i estudi de les eines seleccionades.
- Comparativa de les eines estudiades.
- Conclusions.

2. OCL un complement a l'UML

2.1. Què és l'OCL?

Inicialment podem veure l'OCL com un llenguatge de consulta estàndard, que és part de l'UML (Unified Modeling Language) fixat per l'OMG (Object Management Group) com un estàndard per als anàlisis i dissenys orientats a objectes. [1]

La darrera especificació d'OCL, és la versió 2.0 i que l'OMG ha publicat en el document "Object Constraint language. OMG Available Specification. Version 2.0" en el document "formal/06-05-01" de data de maig de 2006.[2][3]

Segons l'especificació oficial del llenguatge que fa l'OMG, l'Object Constraint Language és un llenguatge formal per descriure expressions en els models UML. Les seves expressions especifiquen condicions invariants que s'han de mantenir en el sistema que s'està modelant, o especifiquen consultes sobre els objectes descrits en el model. S'ha de fer notar que quan les expressions són avaluades, aquestes no produeixen efectes interns, o sigui, les seves avaluacions no poden alterar l'estat del corresponent sistema en execució.

Les expressions OCL poden ser utilitzades per especificar operacions o accions que, un cop executades, poden modificar l'estat del sistema. Els que fan models UML poden utilitzar l'OCL per especificar restriccions en els seus models, també poden utilitzar l'OCL per especificar consultes del model UML, que són programades per un llenguatge totalment independent.

En general l'OCL pot ser utilitzat per aquests motius :

- Com a llenguatge de consulta.
- Per especificar invariants en classes i tipus en el model de classes.
- Per especificar tipus d'invariants en estereotips.
- Per descriure pre i post condicions en operacions i mètodes.
- Per descriure guardes (Guards). Restriccions que han de ser certes perquè una transició a un estat es pugui fer.
- Per especificar objectiu (actituds) per missatges i accions.
- Per especificar restriccions en operacions.
- Per especificar els valors inicials o les regles de derivació per a atributs o associacions finals d'un model UML.
- Per especificar el cos de les operacions.

Hi ha quatre tipus de restriccions que es poden especificar :

- Invariant. És una obligació. L'expressió que la defineix s'ha d'avaluar sempre a cert.
- Pre condició. És una restricció o condició en una operació que s'ha de complir abans d'iniciar-se l'operació.
- Post condició. És una restricció o condició en una operació que s'ha de complir just en el moment que acaba l'operació.
- Guarda. És una obligació que ha de ser certa perquè es pugui fer una transició a un estat.

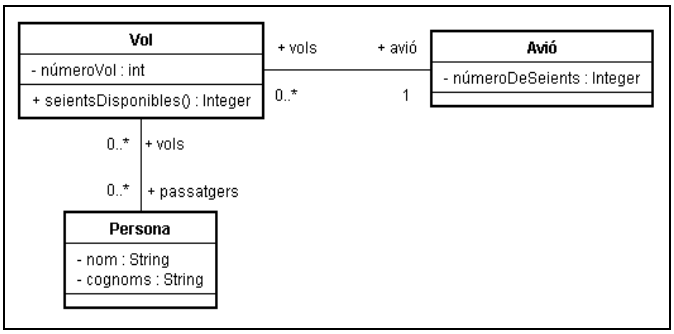
2.2. Com complementa l'OCL a l'UML?

L'UML és un llenguatge gràfic i flexible de modelat que permet descriure models que representen sistemes tan de programari com del món real basant-se amb els conceptes de l'orientació a objectes. L'UML proporciona diferents diagrames que permeten cobrir els diferents punts de vista dels sistemes.

Però si volem que els models siguin el més complets possibles i precisos és necessari utilitzar un llenguatge formal que ens permeti expressar tot allò que no podem especificar mitjançant els diagrames UML. Per exemple, si volem que un atribut d'una classe del model estàtic de l'UML identifiqui els objectes necessitem especificar-ho, i aquest és l'objectiu del llenguatge OCL. Un cas pràctic pot ser que tenint definida la classe Persona i volem identificar-les per un atribut únic com podria ser el NIF, no ho podem fer només amb l'UML, necessitem especificar-ho per evitar tenir dues instàncies amb l'atribut amb el mateix valor. L'OCL ens permet fer-ho, i d'aquesta manera complementem a l'UML.

2.3. Exemples pràctics en OCL.

La millor manera de veure com s'especifica l'OCL, és amb un exemple. Partint del següent diagrama UML hi expressem algunes restriccions, i hi fem algunes consultes a través d'OCL versió 2, que és la versió en que ens basarem en aquest treball :



Il·lustració 2: Model UML pels exemples pràctics en OCL (JUDE 5.1)

Especifiquem una invariant per obligar a que es compleixi sempre que el nombre de passatgers no pot ser mai superior al nombre de seients que té l'avió d'un vol:

```
context Vol
inv: passatgers->size() <= avió.númeroDeSeients
```

Especifiquem un valor inicial en l'associació final 'passatgers'. En aquest cas inicialitzem amb un conjunt buit, un 'Set' buit:

```
context Vol::passatgers : Set(Persona)
init: Set{}
```

Declarem un nou atribut a la classe Avió per saber els vols previstos:

```
context Avió
def: nombreDeVols : Integer = vols->size()
```

Declarem una regla de derivació per definir un suposat atribut compost del nom i cognoms d'una Persona:

```
context Persona::nomComplet
derive: nom.concat(' ').concat(cognoms)
```

Definim el cos de l'operació seientsDisponibles() de la classe Vol:

```
context Vol::seientsDisponibles() : Integer
body: avió.númeroDeSeients - passatgers->size()
```

Definim les pre i post condicions de l'operació de contractar el vol per part d'una persona:

```
context Vol::contractarVol( p : Persona )
pre: self.seientsDisponibles() > 0 and
     not passatgers->includes(p)
post: passatgers = passatgers@pre->including(p)
```

3. Eines CASE que suporten OCL

3.1. Recerca d'eines. Criteris, sistema i comentaris.

Entre les diverses eines CASE que podem trobar actualment per a la generació automàtica de codi Java a partir de la seva especificació, m'he centrat en la recerca d'eines basades en models UML i que suportin el llenguatge OCL.

Vaig realitzar la recerca a través d'internet, centrant-me en pàgines web en anglès i ajudant-me de cercadors com el Google, emprant paraules i frases tals com "CASE", "tool", "OCL", "object constraint language", "OCL support", "code generation", ... i utilitzant diverses combinacions de les mateixes. Trobant des de pàgines web especifiques de productes, com pàgines més generals amb llistes d'eines.

Vaig analitzar les informacions trobades de cada una de les eines, però sense entrar-hi en gaire profunditat, realitzant una llista de les que em van semblar més interessants, o sigui, de les que semblava que complien els requeriments inicials com la utilització de l'OCL i la generació de codi Java.

En la llista en forma de taula hi vaig anotar les característiques més importants de cada una, per posteriorment poder fer la selecció per analitzar en aquest treball en detall. Concretament, les característiques que vaig considerar més importants són:

- Nom / versió
- URL
- Fabricant
- Versió UML/OCL suportada
- Codi que pot generar
- Tipus llicència programari (Cost)
- Sistema operatiu en què es pot executar
- Requeriments d'altres programaris
- Requeriments de maquinari
- Altres comentaris importants

3.2. Eines pre-seleccionades.

Inicialment ja vaig descartar eines que fa molt de temps que no s'actualitzen o les versions han quedat aturades en el temps com "ROCASE 1.0 (2001)", i aquestes ja no les vaig posar a la llista de pre-seleccionades, però per altre banda hi vaig anotar eines que en la seva documentació inicial no especificaven si suportaven les instruccions OCL en la generació de codi, tot i que feien referències a l'ús de les restriccions de l'OCL. Aquestes eines més dubtoses de la llista són Altova, Poseidon for UML, StarUML, Visual Paradigm, i Visual UML.

Per tant, la llista inicial d'eines que semblava que complien o podien complir els requeriments preestablerts eren :

- Altova Umodel 2008 (2007.09.12)
- ArgoUML 0.24 (2007.02.12)
- BluePrint Software Modeler Version 1.4.0 (2007.07.08)
- Dresden OCL2 Toolkit 1.2 (2007.03.27)
- MagicDraw UML 14.0 (2007.09.24)
- Moflon 1.1.1 (2007.07.12)
- OCLE 2.0.4 (2005.07.01)
- Octopus 2.2.0 (2006.03.12)
- Poseidon for UML Community Edition 6.0.1

- Rational Software Architect 7.0
- StarUML 5.0 (2005.12.30)
- Together 2007, Borland (2007.10.12)
- Visual paradigm for UML 6.1 Standard Edition
- Visual UML version 5.3 Developer Edition (2007.10.11)

3.3. Eines descartades, motius.

Investigant amb més profunditat algunes de les eines, les vaig provar i vaig descartar les següents ja que no s'ajustaven als requeriments per fer l'estudi :

- MagicDraw 14.0.
  Les edicions que suporten la generació de codi són "Enterprise" i "Professional", però actualment la única que suporta l'avaluació de les restriccions OCL és l'edició "Enterprise" d'un elevat cost.

- Moflon 1.1.1
  Admet la inclusió de les restriccions OCL, però no en genera bé el codi en Java. Sembla ser que el novembre de 2007 havia de sortir la versió 1.2, la qual havia d'incloure l'avaluació de les instruccions OCL i el metamodel MOF + OCL, però a dates de desembre la nova versió no havia aparegut.

- Poseidon for UML Community Edition 6.0.1
  L'edició "Community" no permet ni gravar ni exportar dades entrades, investigant trobo documentació que amb el programari base d'edicions superiors i amb cost important, només utilitza l'OCL com a comentaris o informació del model, no per generar el codi.

3.4. Eines seleccionades per a analitzar en aquest treball.

De la resta d'eines de la llista, vaig seleccionar i provar les que per llicència, requeriments de maquinari/programari i altres característiques podien ser més assequibles, i amb més possibilitats de complir les condicions necessàries per poder analitzar en aquest treball. Després d'aconseguir els programaris i veure que complien els requeriments, han estat les estudiades en aquest treball. Les eines seleccionades són :

- Java Code Generator GUI de Dresden OCL2 Toolkit 1.2
- OCLE 2.0.4
- Octopus 2.2.0

Són eines diferents, el que ens permetrà veure diferents tipus de programaris.

4. Model de proves

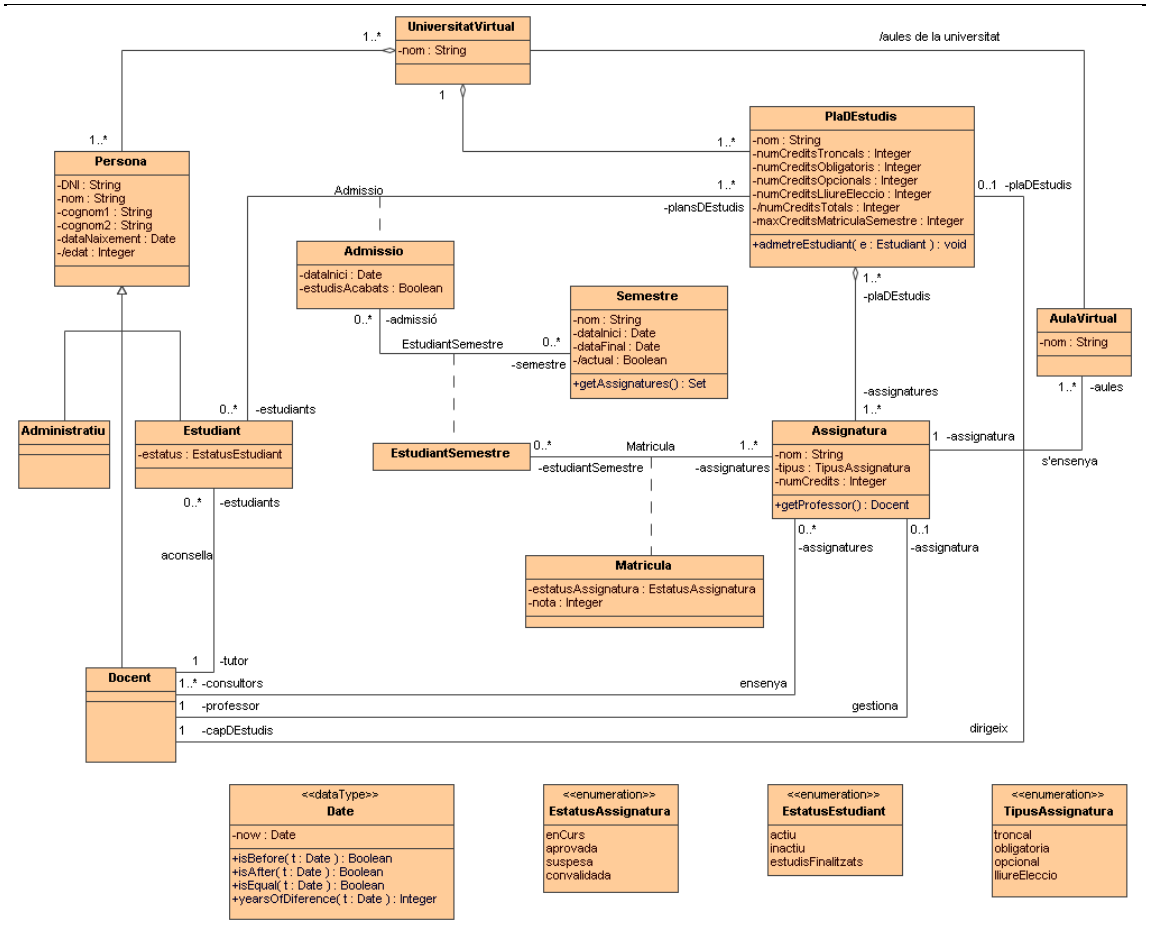
Per poder analitzar i provar d'una manera el més realista possible les eines CASE que suporten OCL per generar el codi Java, he creat un model UML que em servirà per realitzar les proves.

El model representa una part del que és i com funciona actualment la UOC, una universitat virtual. Concretament m'he centrat sobretot en l'estudiant, els docents, el pla d'estudis, i les assignatures a estudiar en un semestre per part de l'estudiant aplicant els coneixements actuals que en tinc com a estudiant de la UOC.

A partir del model UML preparat, he confeccionat una sèrie d'instruccions OCL per poder aplicar al mateix model, el què ens ajudarà a analitzar les diferents eines CASE en generen el codi.

Vegem tot seguit, i en detall, el model UML i les instruccions OCL preparades, amb alguns comentaris sobre els mateixos.

4.1. Model UML



Il·lustració 3: Model UML per a les proves. (MagicDraw 12.0)

La representació gràfica del diagrama de classes del model estàtic de l'UML l'he realitzada amb el programari MagicDraw 12.0 del qual la UOC ens facilita el programa i la clau amb els materials docents. El programari pot exportar la informació generant un fitxer en format XMI, que és un format estàndard i que moltes eines CASE utilitzen.

En el diagrama hi veiem la classe principal *UniversitatVirtual* que bàsicament conté persones (classe *Persona*) i plans d'estudis (classe *PlaDEstudis*), i de manera derivada està relacionada amb les aules virtuals (classe *AulaVirtual*), que és on s'ensenyen les assignatures (classe *Assignatura*), i les assignatures formen els plans d'estudis.

La classe *Persona* és la generalització dels diferents tipus de persones que formen part de la universitat virtual. Així tenim que com a classes més específiques de *Persona* hi tenim les classes *Administratiu*, *Docent* i *Estudiant*. La classe *Docent* tindrà diferents comportaments segons sigui la seva associació amb altres classes, i segons aquestes veiem que hi ha tutors, consultors, professor i cap d'estudis.

Un estudiant es admès a un pla d'estudis determinat una sola vegada, però un estudiant pot estar admès a diferents plans d'estudis. Representem l'admissió a un pla d'estudis amb la classe associativa *Admissio* entre les classes *Estudiant* i *PlaDEstudis*.

Un cop l'estudiant està admès a uns estudis, es pot matricular uns semestre de diferents assignatures. Hauríem de tenir la classe *Matricula* que fos una classe associativa entre les classes *Admissio*, *Assignatura* i *Semestre*, però com que amb la majoria d'eines CASE ens dóna problemes a l'hora d'introduir aquesta representació al model, ho representem amb classes associatives de la següent manera:

- Una classe associativa entre *Admissio* i *Semestre* anomenada *EstudiantSemestre*, que ens informarà quin estudiant es matricularà a un semestre determinat.

- I la classe *Matricula*, que serà una classe associativa entre *EstudiantSemestre* i *Assignatura*, que ens informarà de quines assignatures s'ha matriculat l'estudiant.

Utilitzem algunes classes auxiliars com la classe de tipus data anomenada Data, i altres classes per representar les enumeracions de determinats valors que podran prendre alguns atributs, concretament les classes *EstatusAssignatura*, *EstatusEstudiant* i *TipusAssignatura*.

4.2. Instruccions OCL

Tot seguit adjunto la llista d'instruccions en OCL que aplicaré a l'estudi, i alguns comentaris sobre aquestes.

```
-- Valors inicials
-- ---------------

context UniversitatVirtual::nom  -- String
init:   'UOC'

context Admissio::semestre       -- Set(Semestre)
init:   Set{}


-- Atributs derivats
-- -----------------

context Persona::edat:Integer
derive: self.dataNaixement.yearsOfDiference(Date::now).abs()

context Semestre::actual:Boolean
derive: not self.dataFinal.isBefore(Date::now)  and
        not self.dataInici.isAfter(Date::now)

context PlaDEstudis::numCreditsTotals:Integer
derive: self.numCreditsTroncals + self.numCreditsObligatoris +
        self.numCreditsOpcionals + self.numCreditsLliureEleccio
```

```
-- Invariants
-- ----------

context Persona
inv:    Persona.allInstances()->isUnique(DNI)

context Persona
inv:    self.dataNaixement.isBefore(Date::now)

context Estudiant
inv:    edat >= 18

context Estudiant
inv:    self.plansDEstudis.assignatures->
          includesAll(self.admissio.estudiantSemestre.assignatures)

context Docent
inv:    edat >= 18 and edat <= 75

context  EstudiantSemestre
inv:     self.assignatures.numCredits->sum() <=
          self.admissio.plansDEstudis.maxCreditsMatriculaSemestre

context Assignatura
inv:    self.consultors->excludes(self.professor)

context Admissio
inv:    let credTroncals : Integer = self.estudiantSemestre.matricula->
            select((estatusAssignatura = EstatusAssignatura::aprovada or
                    estatusAssignatura = EstatusAssignatura::convalidada) and
                    assignatures.tipus =
              TipusAssignatura::troncal).assignatures.numCredits->sum()
        let credObligatoris : Integer = self.estudiantSemestre.matricula->
            select((estatusAssignatura = EstatusAssignatura::aprovada or
                    estatusAssignatura = EstatusAssignatura::convalidada) and
                    assignatures.tipus =
              TipusAssignatura::obligatoria).assignatures.numCredits->sum()
        let credOpcionals : Integer = self.estudiantSemestre.matricula->
            select((estatusAssignatura = EstatusAssignatura::aprovada or
                    estatusAssignatura = EstatusAssignatura::convalidada) and
                    assignatures.tipus =
              TipusAssignatura::opcional).assignatures.numCredits->sum()
        let credLliureElec : Integer = self.estudiantSemestre.matricula->
            select((estatusAssignatura = EstatusAssignatura::aprovada or
                    estatusAssignatura = EstatusAssignatura::convalidada) and
                    assignatures.tipus =
              TipusAssignatura::lliureEleccio).assignatures.numCredits->sum()
        estudisAcabats = true implies
           credTroncals >= self.plansDEstudis.numCreditsTroncals          and
           credObligatoris >= self.plansDEstudis.numCreditsObligatoris    and
           credOpcionals >= self.plansDEstudis.numCreditsOpcionals        and
           credLliureElec >= self.plansDEstudis.numCreditsLliureEleccio


-- Definició de nous atributs
-- --------------------------

context PlaDEstudis
def:    nombreEstudiants : Integer = estudiants->size()


-- Definició del Cos d'operacions
-- ------------------------------

context  Semestre::getAssignatures() : Set(Assignatura)
body:    self.estudiantSemestre.assignatures->asSet()


-- Pre i post condicions
-- ---------------------
```

```
context Assignatura::getProfessor() : Docent
pre:    -- none
post:   result = self.professor


context PlaDEstudis::admetreEstudiant(e : Estudiant)
pre:    not estudiants->includes(e)
post:   estudiants = estudiants@pre->including(e) and
        Admissio.allInstances()->one(Estudiant = e and PlaDEstudis = self) and
       (let admi: Admissio =
            Admissio.allInstances()->any(Estudiant = e and PlaDEstudis = self)
        in  admi.oclIsNew() and
            admi.dataInici.isEqual(Date::now) and
            admi.estudisAcabats = false
       )
```

Il·lustració 4: Instruccions OCL per a les proves.

Per aconseguir un estudi el més complert possible de les eines CASE a analitzar, he mirat de seleccionar expressions OCL variades i evitant de posar-ne de semblants. No ens cal fer un model on especifiquem totes les restriccions possibles, sinó que volem veure com les eines CASE ens generen el codi dels diferents tipus d'expressions OCL.

Us cas concret a comentar, i seguint els criteris inicials de provar expressions variades, és que en operacions semblants de '*getters*' faig servir dues maneres de fer-ho, per exemple podem veure en el context *Assignatura* l'operació '*getProfessor()*' on utilitzo '*post: result*', i en el context *Semestre* amb l'operació '*getAssignatures()*' utilitzo '*body*'.

5. Anàlisi de l'eina: Java Code Generator GUI de Dresden OCL2 Toolkit 1.2

5.1. Informació general de l'eina

Desdren OCL2 Toolkit 1.2 [4] és un conjunt d'eines que podem trobar per internet i que han generat estudiants i científics que treballen pel 'Software Technology Group at Technische Universität Dresden', que és qui coordina el projecte sota llicència GNU Library or Lesser General Public License (LGPL). Les eines es poden integrar en altres programaris com ArgoUML, Borland Together, Fujaba4Eclipse, MetaBoss, Poseidon, Rational Rose

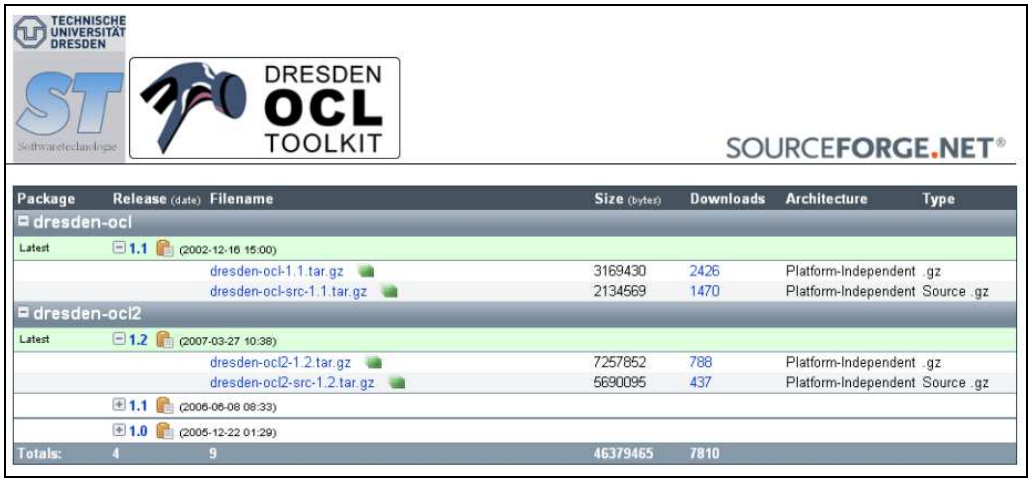
Entre les diferents eines que composen el conjunt, hi trobem l'eina "Java Code Generator GUI" o JCGG, que és una extensió d'una altra eina també del conjunt anomenada "OCL2 Workbench" per a la generació de codi Java que aporta més funcionalitats.

Aquest programari va ser escrit per en Ronny Brandt per al projecte de recerca del seu estudiant Grober Beleg. I amb aquest programari es poden carregar models UML des de fitxers XMI i restriccions OCL per generar codi Java per comprovar aquestes restriccions en temps d'execució. Amb el codi generat es pot injectar en fitxer existents de Java i també es pot fer enginyeria inversa.

5.2. Instal·lació del programari

Obtenció del programari

El programari el podem baixar sense cap requeriment previ des de la pàgina de descàrrega del lloc web. Veiem que es pot baixar de dues maneres, en binaris executables o en codi ("*Source code*"). Per fer l'anàlisi que volem fer, triem el binari executable, i la versió 1.2, la més nova de 'dresden-ocl2':



Il·lustració 5: JCGG de Dresden Toolkit. Logotips i detall descàrrega del programari.

Requeriments

El programari és multi plataforma perquè s'executa a través de Java, i com a requeriments per executar-lo en format binari només indiquen els següents:

- Java JRE 1.5
- Tenir ben definida la variable *$JAVA_HOME* de l'entorn del sistema

- I tenir el fitxer "*tools.jar*" localitzat a "*$Java_Home/lib/tools.jar*".

En quan a maquinari es recomana 256 MB o més de memòria RAM i almenys 400 MHz de CPU.

Comentaris instal·lació

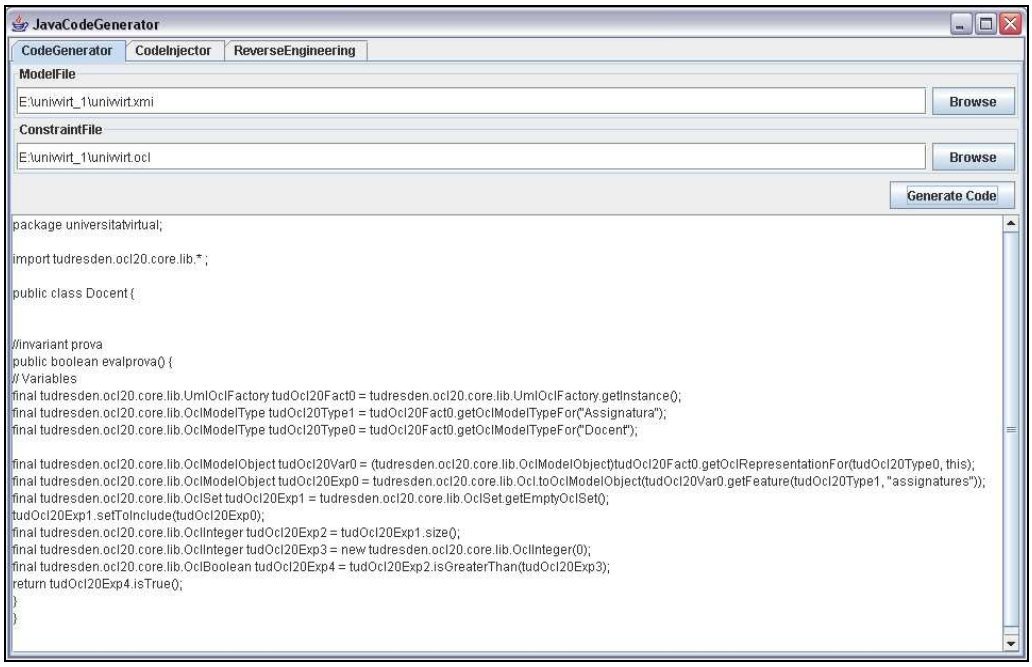
El programari descarregat ve en un fitxer, en aquest cas '*dresden-ocl2-1.2.tar.gz*', i es troba en format comprimit, dins del qual hi ha totes les eines del conjunt. Només cal desempaquetar-lo en un directori.

El que ens interessa per fer l'anàlisi és el fitxer '*ocl20codegenGUI-java.jar*', i els exemples que podem consultar a "*./resources/javacodegenexample/*". Un cop desempaquetat el fitxer descarregat, i si complim amb el requeriments, el programari ja és operatiu i només s'ha d'invocar executant el fitxer '*ocl20codegenGUI-java.jar*'.

5.3. Utilització del programari

Funcionament de l'eina

El funcionament de l'eina és simple i gens complicat. En la interfície inicial trobem les tres possibilitats que ofereix: generació de codi, injecció de codi, i enginyeria inversa.



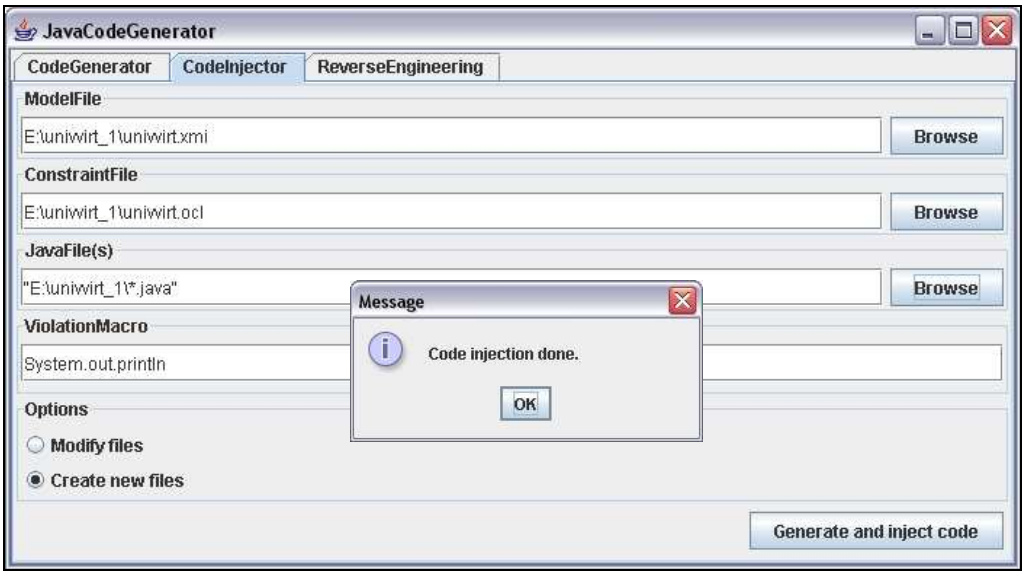
Il·lustració 6: JCGG de Dresden Toolkit. Interfície gràfica de CodeGenerator

En aquest treball bàsicament analitzarem la generació de codi, ja que d'injecció de codi és la gravació del codi a unes classes Java prèviament generades, ja sigui per una aplicació o programari extern, o bé manualment per l'usuari.

Però vegem-n'hi un exemple. Usant el model UML d'aquest treball, i una instrucció simple com la següent :

```
context Docent
inv prova: self.assignatures->size() > 0
```

Amb el "*CodeGenerator*" obtenim el resultat per pantalla (Il·lustració anterior). Però utilitzant el "CodeInjector", a part d'especificar els mateixos fitxers de model, i instruccions OCL, especifiquem també on tenim el codi Java generat prèviament :



Il·lustració 7: JCGG de Dresden Toolkit. Interfície gràfica de CodeInjector

Així, si al fitxer "*Docent.java*" inicialment era :

```
package univvirt;

/** Class ...
 */
public class Docent {

	/** Default constructor for Persona
	 */
	public Docent() {
	}

}
```

Un cop executada la injecció de codi tenim :

```
package univvirt;

/** Class ...
 */
public class Docent {

	/** Default constructor for Persona
	 */
	public Docent(tudresden.ocl20.injection.lib.WrapperDummy wrappedbyocl)
{
	}

/**
A wrapper for injection. Generated automatically, DO NOT CHANGE!
@author ocl_injector
@see #Docent(tudresden.ocl20.injection.lib.WrapperDummy)
*/public Docent(){
this((tudresden.ocl20.injection.lib.WrapperDummy)null);
try{tudresden.ocl20.injection.ocl20.lib.Invariant.checking_flag=true;
tudresden.ocl20.injection.ocl20.lib.Invariant.checkVacantInvariants();
}finally{tudresden.ocl20.injection.ocl20.lib.Invariant.checking_flag=false;}
}/**
Checks object features, whether they have changed. Generated automatically, DO
NOT CHANGE!
@author ocl_injector
```

```
*/private void checkForChangedFeatures(){
}/**
Checks object features, whether they have changed. Generated automatically, DO
NOT CHANGE!
@author ocl_injector
*/private static void checkForChangedStaticFeatures(){
}/**
An object representing ocl invariant prova on this object. Generated
automatically, DO NOT CHANGE!
@author ocl_injector
*/private final tudresden.ocl20.injection.ocl20.lib.Invariant
zzzCheckOclInvariantObject812374_prova=new
tudresden.ocl20.injection.ocl20.lib.Invariant("prova", this);/**
Checks ocl invariant prova on this object. Generated automatically, DO NOT
CHANGE!
@author ocl_injector
*/public final void zzzCheckOclInvariantMethod812374_prova(){
tudresden.ocl20.core.lib.UmlModelObject.setFeatureListener(zzzCheckOclInvarian
tObject812374_prova);
// Variables
final tudresden.ocl20.core.lib.UmlOclFactory tudOcl20Fact0 =
tudresden.ocl20.core.lib.UmlOclFactory.getInstance();
final tudresden.ocl20.core.lib.OclModelType tudOcl20Type1 =
tudOcl20Fact0.getOclModelTypeFor("Assignatura");
final tudresden.ocl20.core.lib.OclModelType tudOcl20Type0 =
tudOcl20Fact0.getOclModelTypeFor("Docent");
// Invariant
final tudresden.ocl20.core.lib.OclModelObject tudOcl20Var0 =
(tudresden.ocl20.core.lib.OclModelObject)tudOcl20Fact0.getOclRepresentationFor
(tudOcl20Type0, this);
final tudresden.ocl20.core.lib.OclModelObject tudOcl20Exp0 =
tudresden.ocl20.core.lib.Ocl.toOclModelObject(tudOcl20Var0.getFeature(tudOcl20
Type1, "assignatures"));
final tudresden.ocl20.core.lib.OclSet tudOcl20Exp1 =
tudresden.ocl20.core.lib.OclSet.getEmptyOclSet();
tudOcl20Exp1.setToInclude(tudOcl20Exp0);
final tudresden.ocl20.core.lib.OclInteger tudOcl20Exp2 = tudOcl20Exp1.size();
final tudresden.ocl20.core.lib.OclInteger tudOcl20Exp3 = new
tudresden.ocl20.core.lib.OclInteger(0);
final tudresden.ocl20.core.lib.OclBoolean tudOcl20Exp4 =
tudOcl20Exp2.isGreaterThan(tudOcl20Exp3);
tudresden.ocl20.core.lib.UmlModelObject.clearFeatureListener();
if(!tudOcl20Exp4.isTrue())System.out.println("violated ocl invariant 'prova'
on object '"+this+"'.");
}}
```

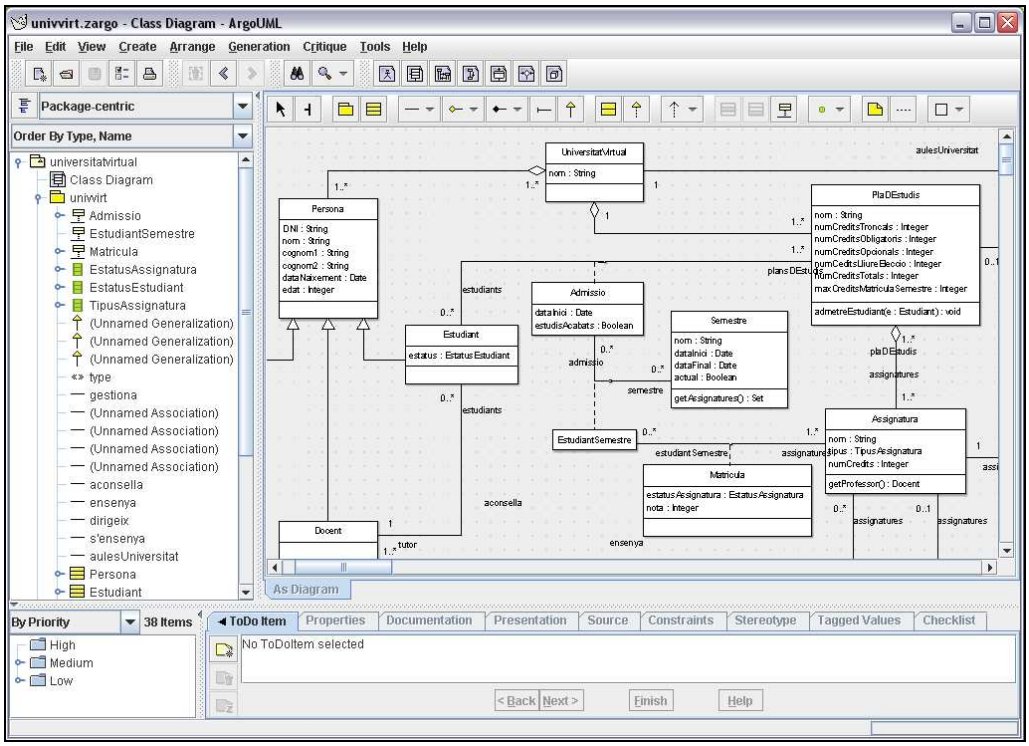
Veiem que és un codi Java difícil de llegir afegit al final del fitxer que no presenta entrades o marges (en anglès '*indentations*'), i el codi de la injecció és molt semblant al de la generació. Com que el codi de la generació és més simplificat que el de la injecció en l'anàlisi ens basarem en el resultat d'aquesta opció.

Creació del model UML

L'eina no disposa d'un entorn gràfic per crear el model UML, i necessita importar el model creat per un altre programari en format XMI compatible. En el fitxer "GETTING-STARTED" del programari comenten que els format XMI provats són els creats per les eines UML "ArgoUML" i "Poseidon For UML 3.0".

Després de fer vàries proves amb fitxers XMI del model UML generats per eines tals com MagicDraw, OCLE, i altres, i veient la no obtenció de resultats positius, he realitzat el model a través d'ArgoUML 0.24 i en proves posteriors del programari m'ha funcionat correctament.

Amb l'eina ArgoUML també podem generar el codi en Java del model UML, i utilitzar-lo després en la injecció del codi de les restriccions que ens proporciona l'eina JCGG.



Il·lustració 8: ArgoUML. Construcció del model UML per a JCGG

Entrada instruccions OCL

Per especificar les instruccions OCL l'eina es basa en la utilització d'un únic fitxer de text on hi ha d'haver totes les restriccions o instruccions en OCL. L'eina JCGG no proporciona cap editor propi per crear/modificar o accedir al fitxer, però es pot utilitzar qualsevol altre editor de text pla.

El fitxer ha de tenir ben anotat el nom de paquet de programari ('package') al quan correspon el model UML que s'utilitza, ja que si són diferents no es tractaran les instruccions, i la eina no retornarà cap error:

```
package univvirt

-- instruccions

endpackage
```

Depuració d'errors

Segons les moltes proves fetes, he observat que l'eina no retorna els errors detectats en el seu tractament, l'eina simplement o genera el codi o si troba errors no el genera. Les instruccions OCL que no reconeix, no les tracta, i si la instrucció no reconeguda o errònia i es troba en el mig del fitxer, a partir d'aquesta s'atura el programa i ja no en genera cap més. També he observat que si és fa la injecció amb un codi de model complicat, l'eina pot generar la injecció del codi només a una part dels fitxers Java, però no indica per quin problema s'ha deixat d'executar en els altres fitxers.

Per evitar alguns possibles errors, he eliminat l'ús de paraules amb accents tan en el model UML creat amb l'eina ArgoUML, exportat a XMI i utilitzat amb l'eina JCGG, com en el fitxer d'instruccions OCL.

Les mancances de l'eina en l'avís i depuració d'errors m'han obligat a fer l'anàlisi instrucció per instrucció, comentant totes les instruccions del fitxer OCL, i anant descomentant la que anava provant, per posteriorment un cop estudiada, tornar-la a comentar.

5.4. Anàlisi del codi generat

Valors inicials ('init')

Les instruccions '**init**' no són interpretades per l'eina. Aplicant les instruccions preparades no obtenim cap resultat per pantalla, i si en el fitxer d'instruccions OCL n'hi havia que abans anaven bé, aquestes deixen de funcionar quan hi afegim instruccions 'init'.

Derivacions ('derive')

Les instruccions '**derive**' tampoc són interpretades per l'eina. He provat de transformar les instruccions a invariants, i tampoc m'ha acceptat les instruccions del model de proves com a invariants :

```
context Persona
inv: edat = self.dataNaixement.yearsOfDiference(Date::now).abs()

context Semestre
inv:    if not self.dataFinal.isBefore(Date::now) and
           not self.dataInici.isAfter(Date::now)
        then actual = true
        else actual = false
        endif

context PlaDEstudis
inv:    numCreditsTotals =
           self.numCreditsTroncals+self.numCreditsObligatoris+
           self.numCreditsOpcionals+self.numCreditsLliureEleccio
```

Invariants ('inv')

L'eina accepta algunes invariants tal i com hem vist en la prova sobre el model en l'apartat "Funcionament de l'eina", i com podem consultar en els exemples que tenim en el fitxer "*Testszenario.ocl*" que podem trobar en la instal·lació i localitzem en el directori "*./dresden-ocl2-1.2/resources/ javacodegenexample/*", per exemple:

```
context Person
inv notHomeless: self.accomodation->size() > 0

context Person
inv registeredResidence: (self.accomodation->
       select(x:Accomodation|x.category=1))->size() = 1

context Person
inv iter: Set{1,2,3}->
    iterate( i: Integer; sum: Integer = 0 | sum + i ) = 6

context Examination
inv exPassed: let examPassed : Boolean = self.passed
in examPassed = true implies self.mark <= 4.0
```

Però un cop provades totes les invariants del model de proves preparat, veiem que no ens n'ha tractat cap. He fet diferents modificacions, com afegir el nom a les invariants, afegir parèntesis a grups d'instruccions, ... sense èxit. Inclús he provat de crear diferents sub-models del model UML, però sense resultats positius.

Definicions ('def')

Davant la instrucció que tenim en el model de proves, l'eina genera un codi incomplert sense afegir-hi la definició, i per tant no podem donar-ho per bo:

```
    package univvirt;

    import tudresden.ocl20.core.lib.* ;

    public class PlaDEstudis {
    }
```

Cos d'operacions ('body')

L'eina no accepta la instrucció 'body' ja que no ens genera cap tipus de codi i/o informació.

Pre i post condicions ('pre', 'post')

En el model de proves tenim dues instruccions d'aquest tipus. Passem a veure-les per separat:

En el primer cas, ho provem amb la pre condició buida amb un comentari i no obtenim codi, eliminant la pre condició obtenim el codi generat, i veiem que satisfà la post condició especificada :

```
    context Assignatura::getProfessor() : Docent
--  pre:    -- none
    post:   result = self.professor
```

```
package univvirt;

import tudresden.ocl20.core.lib.* ;

public class Assignatura {


//postcondition
public boolean eval() {


// Variables
final tudresden.ocl20.core.lib.UmlOclFactory tudOcl20Fact0 =
tudresden.ocl20.core.lib.UmlOclFactory.getInstance();
final tudresden.ocl20.core.lib.OclModelType tudOcl20Type1 =
tudOcl20Fact0.getOclModelTypeFor("univvirt::Assignatura");
final tudresden.ocl20.core.lib.OclModelType tudOcl20Type0 =
tudOcl20Fact0.getOclModelTypeFor("univvirt::Docent");

final tudresden.ocl20.core.lib.OclModelObject tudOcl20Var0 =
(tudresden.ocl20.core.lib.OclModelObject)tudOcl20Fact0.getOclRepresentationFor
(
tudOcl20Type0, result);
final tudresden.ocl20.core.lib.OclModelObject tudOcl20Var1 =
(tudresden.ocl20.core.lib.OclModelObject)tudOcl20Fact0.getOclRepresentationFor
(
tudOcl20Type1, this);
```

```
final tudresden.ocl20.core.lib.OclModelObject tudOcl20Exp0 =
tudresden.ocl20.core.lib.Ocl.toOclModelObject(tudOcl20Var1.getFeature(
tudOcl20Type0, "professor"));
final tudresden.ocl20.core.lib.OclBoolean tudOcl20Exp1 =
tudOcl20Var0.isEqualTo(tudOcl20Exp0);
return tudOcl20Exp1.isTrue();
}
}
```

En el segon cas tenim:

```
    context PlaDEstudis::admetreEstudiant(e : Estudiant)
    pre:    not estudiants->includes(e)
    post:   estudiants = estudiants@pre->including(e) and
            Admissio.allInstances()->
              one(Estudiant = e and PlaDEstudis = self) and
           (let admi: Admissio =
                Admissio.allInstances()->
                  any(Estudiant = e and PlaDEstudis = self)
            in  admi.oclIsNew() and
                admi.dataInici.isEqual(Date::now) and
                admi.estudisAcabats = false
           )
```

I veiem que l'eina no ens retorna codi, no interpreta la instrucció.

Com a les invariants, veiem que en el fitxer d'exemples "*Testszenario.ocl*" hi ha una amplia gamma de pre i post condicions:

```
    context Student::writeExamination(ex:Examination):Boolean
    pre courseIsVisited: self.visitedCourses->includes(ex.course)
    post passed: result = ex.passed

    context Student::visitCourse(course:Course)
    post courseVisited: self.visitedCourses->includes(course)
    post addedC: self.visitedCourses->size() =
        (self.visitedCourses@pre->size() + 1)

    context Insurance::calculateOffer(insuranceSum:Real):Real
    post static: result = insuranceSum / self.averageYears

    context City::getSomeString(like:String):String
    pre testCity: like <> ''

    context Accomodation::setCategory(category:Integer):Boolean
    post returnValue: result=(category=self.category@pre)
```

però com hem comprovat l'eina no interpreta totes les instruccions.

5.5. Conclusions de l'eina

L'eina Java Code Generator GUI (JCGG) de Dresden OCL2 Toolkit 1.2 pot interpretar models UML i instruccions OCL sobre el model. És un programari multi plataforma que s'executa amb Java i és de llicència pública. És fàcil d'utilitzar, ja que no te gaires opcions ni complicacions, tot i que he trobat algun problema depenent de la màquina on l'executava.

No té entorn propi per generar els models UML a interpretar, i aquests han d'estar en fitxers en format XMI i s'han de generar per eines externes, tot i que no tots els formats XMI són reconeguts. Les instruccions OCL sobre el model UML han d'estar

en un únic fitxer, el que pot provocar que una instrucció errònia perjudiqui a altres instruccions correctes.

Pot generar el codi de les instruccions a través de la pantalla on s'executa, o bé injectant-lo en fitxers Java que representen les classes del model UML interpretat. Aquests fitxers Java s'han de crear prèviament per altres aplicacions o manualment per l'usuari. El codi generat és difícil de llegir, ja que no hi presenta marges o entrades.

El programa no informa dels errors trobats ni a on és localitzen, només interpreta el model i les instruccions, i si ho troba correcte genera el codi, però sinó, o genera informació parcial com pot ser algunes de les restriccions OCL interpretades, o no genera informació de cap tipus.

Les instruccions bàsiques d'OCL que tracta són invariants, pre i post condicions.

Personalment penso que és una eina molt bàsica i limitada però que el seu ús ens pot servir per provar determinades instruccions, i afegir el codi executable de les restriccions OCL al codi Java prèviament creat.

6. Anàlisi de l'eina: OCLE 2.0.4

6.1. Informació general de l'eina

OCLE (Object Constraint Language Environment) 2.0.4 és una eina CASE UML que trobem a la universitat de Romania "'BABES-BOLYAI' University Computer Science Research Laboratory" [5].

OCLE és una eina CASE UML que ofereix ple suport a l'OCL, tant al model superior UML com al model pla o de l'usuari. Entre les característiques principals indicades tenim:

- Suporta OCL 2.0 i UML 1.5
- Pot intercanviar models UML utilitzant XMI 1.0 i 1.1.
- El model UML es comprova amb les regles definides al model superior UML o metamodel (regles ben formades, regles del perfil, regles metodològiques).
- Validació semàntica de documents XML descrits per mitjà de DTDs (Document Type Definition).
- Generació de codi Java, tant de l'arquitectura del model UML com de les especificacions en OCL.

6.2. Instal·lació del programari

Obtenció del programari

El programari és lliure i es pot obtenir donant una sèrie de dades com són: el nom, l'empresa, i una adreça de correu electrònic. Un cop registrats rebem un correu que ens permet descarregar-lo, juntament amb el manual i uns exemples.

Il·lustració 9: OCLE. Logotip i detall descàrrega del programari

Requeriments

El programari és multi plataforma perquè s'executa a través de Java, i com a requeriments per executar-lo només indiquen els següents:

- Java "Sun JRE 1.4" o superior.
- PII-300 i 128 MB de memòria (recomanat PII-500 amb 256 MB) o altres ordinadors amb les mateixes prestacions.

Comentaris instal·lació

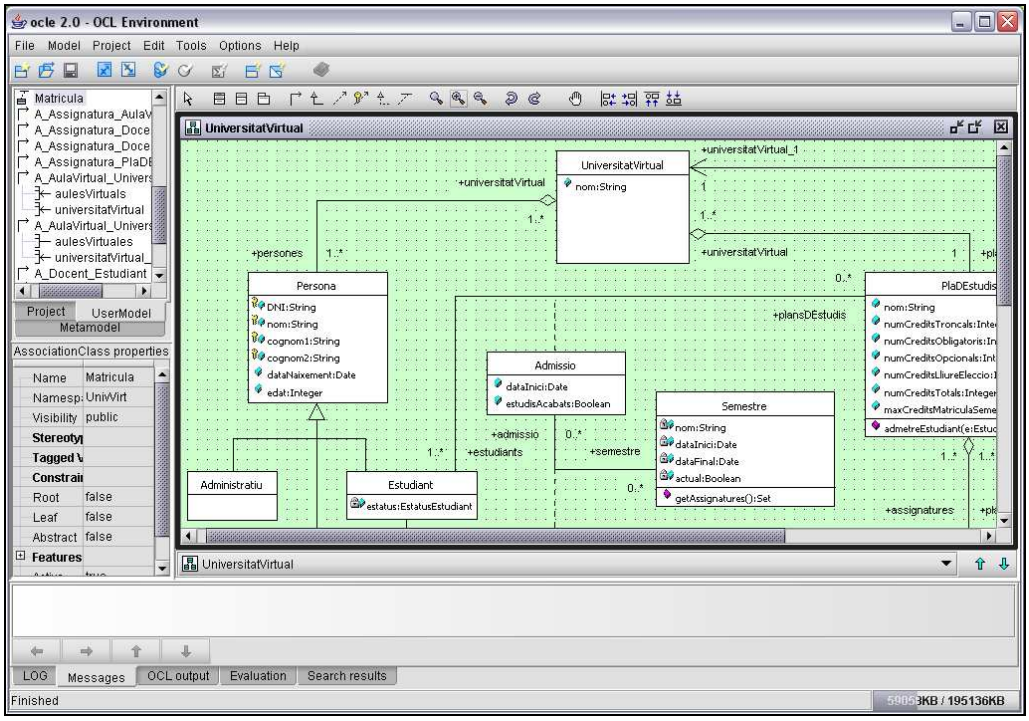
El programari ve en un fitxer en format 'zip', en aquesta versió el nom del fitxer és '*ocle-2.0.4.zip*'. Només cal desempaquetar el fitxer mantenint l'estructura interna de directoris.

En el directori on l'hem descarregat hi trobarem un fitxer de nom '*run_windows.bat*', executant el fitxer engegarem el programari en màquines d'entorn Windows, que és el nostre cas.

6.3. Utilització del programari

Funcionament de l'eina i creació del model UML

L'eina disposa d'un entorn gràfic propi per crear el model UML. És un entorn fàcil d'utilitzar, amb opcions intuïtives, i sense complexitats. Es complementa amb el manual de l'usuari, on es detalla el funcionament.



Il·lustració 10: OCLE. Interfície gràfica

He creat el model de proves amb l'eina. Inicialment utilitzava accents en determinats atributs o nom de classes, el que provocava errors posteriors interns, per tant cal evitar-los així com caràcters especials.

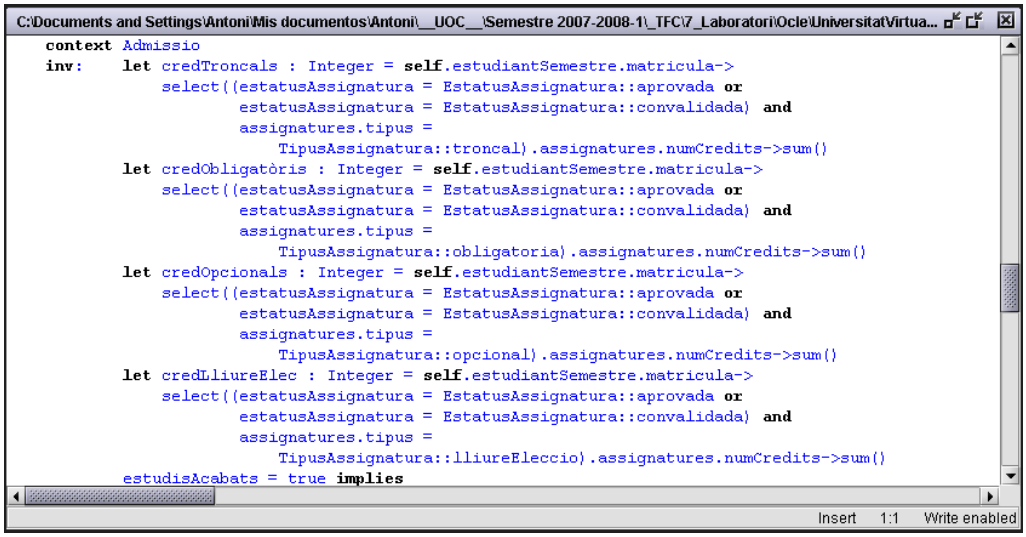
Cal indicar que treballant amb el model UML, de tant en tant apareix un petit '*bug*' o problema de programari consistent en un canvi dels noms i multiplicitats en les associacions en la visualització gràfica del diagrama, posant els 2 noms en un extrem i les 2 multiplicitats en l'altre. Sembla que només té efectes visuals ja que consultant els objectes es mantenen ben definits.

Amb aquesta eina és important també definir les visibilitats correctes dels atributs, ja que si una instrucció OCL ha de fer referència o ha d'utilitzar un atribut d'una classe que no és la del context, segons la visibilitat d'aquest, no hi podrà accedir.

Entrada instruccions OCL

Per especificar les instruccions OCL l'eina es basa en la utilització de dos fitxers de text pla, un per a les instruccions OCL del metamodel o model de l'UML, i l'altre per a les instruccions del model de l'usuari. El fitxer amb les instruccions del metamodel s'anomena amb extensió '*.ocl*', i el del model de l'usuari amb l'extensió '*.bcr*'. Es poden afegir instruccions OCL en els dos fitxers, però com que el metamodel ja te les regles internes bàsiques definides, i ens interessa analitzar el model d'usuari, només utilitzarem el fitxer '*.bcr*'.

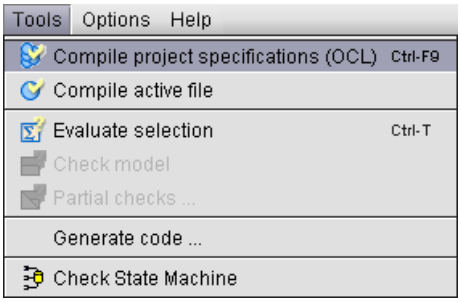
L'eina proporciona un editor propi integrat a la mateixa eina, tot i que es pot utilitzar qualsevol editor de text per crear-lo, modificar-lo o visualitzar-lo.

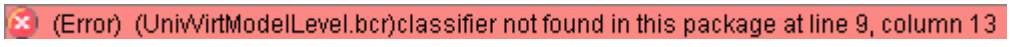


Il·lustració 11: OCLE. Editor propi per editar instruccions OCL

Depuració d'errors

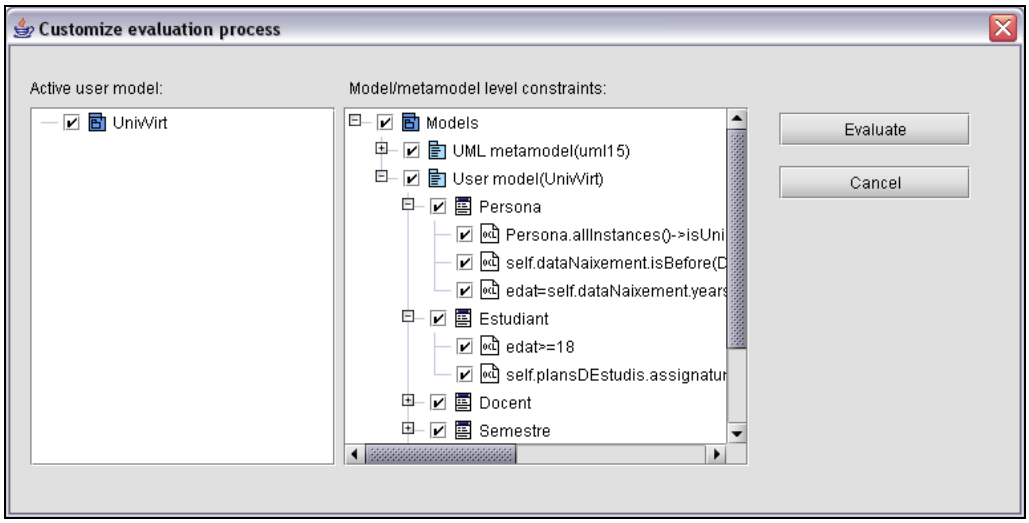
Un cop disposem del model UML i les instruccions OCL del model podem compilar-lo i l'eina ens detectarà els errors sintàctics, indicant el tipus i la localització dins el fitxer de les instruccions.





Il·lustració 12: OCLE. Error de compilació

I un cop tenim les instruccions entrades i compilades, podem veure les restriccions i fer avaluacions parcials o totals del model a través del menú de l'eina (Tools + Partial checks...), detectant possibles errors semàntics de l'estil que hi hagi una expressió impossible de satisfer per algun motiu.

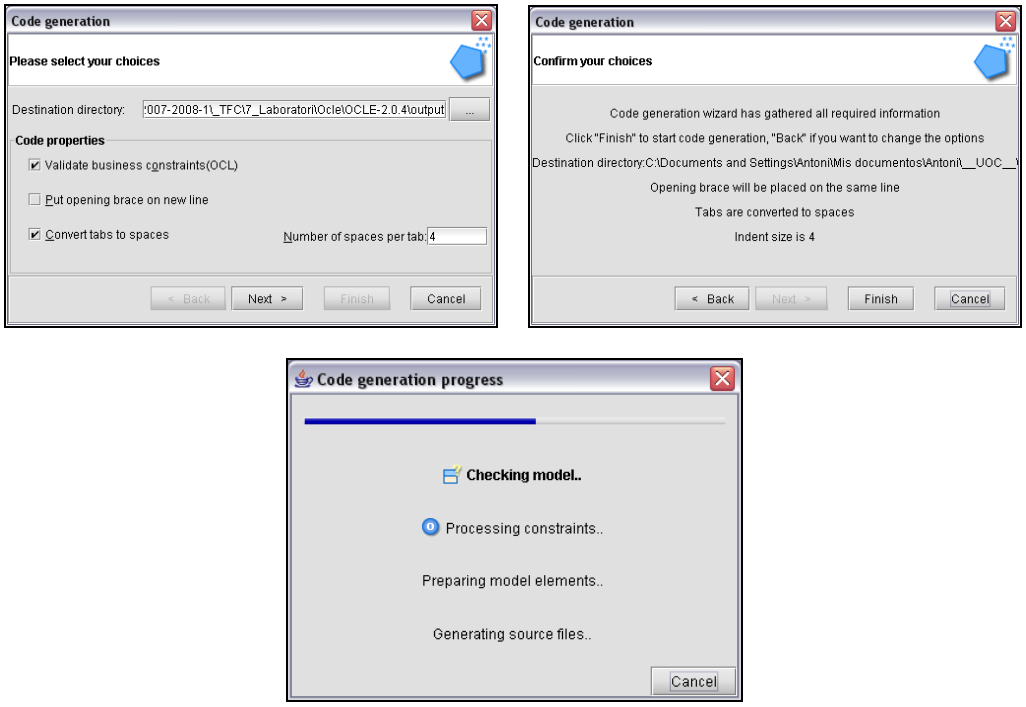


Il·lustració 13: OCLE. Avaluacions parcials de les restriccions

Generació del codi Java

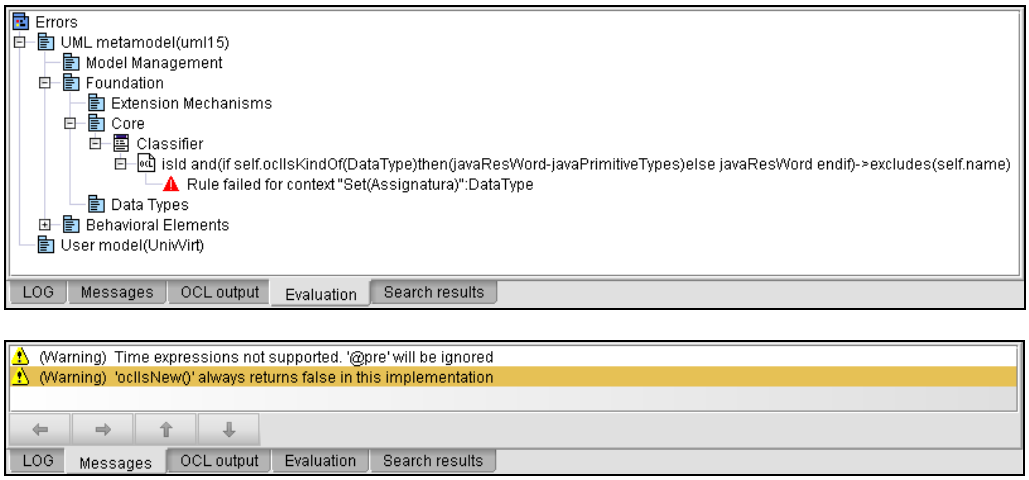
Un cop hem compilat el projecte amb el model UML i les instruccions OCL, podem generar el codi Java. (Tools + Generate code ...).

Ens deixa seleccionar algunes opcions. I un cop confirmades es veu com la barra del procés de generació. Si en la generació troba un error, ens dirà en quina fase s'ha trobat i quin error és, però si va bé, o tot i trobar errors volem confirmar la generació del codi, en el directori que hem especificat hi trobarem el codi Java generat.



Il·lustració 14: OCLE. Finestres de generació de codi

Un cop demanem a l'eina que ens generi el codi segons el model UML i les instruccions OCL del model de proves introduït, obtenim un error en el model (tot i haver passat les validacions prèvies), i dues alertes (warnings) en unes instruccions.



Il·lustració 15: OCLE. Errors generant el codi del model

Concretament tenim:

- Un error al comprovar el model. A l'apartat '*Evaluation*':
  - Errors -> UML metamodel(uml15) -> Foundation -> Core -> Classifier -> isId and(if self.oclIsKindOf(DataType)then(javaResWord-javaPrimitiveTypes) else javaResWord endif)->excludes(self.name)
  -> Rule failed for context "Set(Assignatura)":DataType

- Una alerta diu:
  "*Time expressions not supported. '@pre' will be ignored.*"

- I l'altra alerta diu:
  "*oclIsNew()' always returns false in this implementation.*"

L'error ve donat al definir la funció '*getAssignatures()*' de la classe *Semestre* amb la instrucció OCL '*body*', ja que amb l'eina no hem pogut definir correctament el valor que ha de retornar la funció, això és un '*Set(Assignatura)*'. He fet moltes proves, definit diferent tipus de dades amb l'eina tals com *Set*, *Set(Assignatura)*, ... però sense resultat satisfactori.

I segons trobem al manual de l'eina, sembla que les alertes venen donades per utilitzar les instruccions '*@pre*' i '*oclIsnew*', ja que no estan implementades en l'eina i la generació del codi :

> *"some postcondition specific constructs, such as@pre and the oclIsNew operation in OclAny are not properly mapped in the generated source code. If the code generator encounters such constructs, it issues adequate warning messages."*

Haurem de tenir-ho en compte quan analitzem el codi.

6.4. Anàlisi del codi generat

Valors inicials ('init')

La instrucció '**init**' no està prevista a l'eina. A l'aplicar les instruccions amb 'init', l'eina no les ha reconegut donant error, i les he hagut de treure del test de proves.

Derivacions ('derive')

La instrucció '**derive**' no està prevista a l'eina. A l'aplicar les instruccions, l'eina no les ha reconegut. He convertit les instruccions a invariants, i aquestes s'han acceptat.

```
--* context Persona::edat:Integer
--* derive: self.dataNaixement.yearsOfDiference(Date::now).abs()
--* ******* - Podem convertir en 'invariant'
    context Persona
    inv:    edat =
self.dataNaixement.yearsOfDiference(Date::now).abs()

--* context Semestre::actual:Boolean
--* derive: not self.dataFinal.isBefore(Date::now)  and
--*         not self.dataInici.isAfter(Date::now)
--* ******* - Podem convertir en 'invariant'
    context Semestre
    inv:    if not self.dataFinal.isBefore(Date::now) and
               not self.dataInici.isAfter(Date::now)
            then actual = true
            else actual = false
            endif

--* context PlaDEstudis::numCreditsTotals:Integer
--* derive: self.numCreditsTroncals + self.númCreditsObligatoris +
--*         self.numCreditsOpcionals + self.númCreditsLliureEleccio
--* ******* - Podem convertir en 'invariant'
    context PlaDEstudis
    inv:    numCreditsTotals =
               self.numCreditsTroncals+self.numCreditsObligatoris+
               self.numCreditsOpcionals+self.numCreditsLliureEleccio
```

Invariants ('inv')

L'eina ha acceptat totes les invariants del test de proves. Analitzem-ho amb detall en el codi que ha generat en la classe *Persona* on hi apliquem 3 invariants (les dues inicials del model de proves i una instrucció '*derive*' passada a invariant) :

```
    context Persona
    inv:    Persona.allInstances()->isUnique(DNI)

    context Persona
    inv:    self.dataNaixement.isBefore(Date::now)

    context Persona
    inv: edat = self.dataNaixement.yearsOfDiference(Date::now).abs()
```

Observem el codi generat del fitxer Persona.java, on hi hem marcat amb negreta les funcions que ens validen les invariants :

```
/*
 * @(#)Persona.java
 *
 * Generated by <a href="http://lci.cs.ubbcluj.ro/ocle/>OCLE 2.0</a>
 * using <a href="http://jakarta.apache.org/velocity/">
 * Velocity Template Engine 1.3rc1</a>
 */
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.Set;
import ro.ubbcluj.lci.codegen.framework.ocl.BasicConstraintChecker;
import ro.ubbcluj.lci.codegen.framework.ocl.BasicUtilities;
import ro.ubbcluj.lci.codegen.framework.ocl.CollectionUtilities;
import ro.ubbcluj.lci.codegen.framework.ocl.Ocl;
```

```
import ro.ubbcluj.lci.codegen.framework.ocl.OclType;

/**
 *
 * @author unascribed
 */
protected class Persona {

    public final Set getUniversitatVirtual() {

        if (universitatVirtual == null) {
            return java.util.Collections.EMPTY_SET;
        }
        return java.util.Collections.unmodifiableSet(universitatVirtual);
    }

    public final void addUniversitatVirtual(UniversitatVirtual arg) {

        if (arg != null) {
            if (universitatVirtual == null) {
                universitatVirtual = new LinkedHashSet();
            }
            universitatVirtual.add(arg);
        }
    }

    public final void removeUniversitatVirtual(UniversitatVirtual arg) {

        if (universitatVirtual != null && arg != null) {
            universitatVirtual.remove(arg);
        }
    }

    public Persona() {

    }

    public class ConstraintChecker extends BasicConstraintChecker {

        public void checkConstraints() {

            super.checkConstraints();
            check_Persona_invariant();
            check_Persona_invariant0();
            check_Persona_invariant1();
        }

        public void check_Persona_invariant() {

        Set setAllInstances = Ocl.getType(new
Class[]{Persona.class}).allInstances();
            //evaluate 'isUnique(DNI)':
            Set uniquenessValidator = CollectionUtilities.newSet();
            boolean bIsUnique = true;
            final Iterator iter = setAllInstances.iterator();
            while (bIsUnique && iter.hasNext()) {
                final Persona iter1 = (Persona)iter.next();
                String strDNI = iter1.DNI;

                bIsUnique = uniquenessValidator.add(strDNI);
            }
            if (!bIsUnique) {
                System.err.println("invariant 'invariant' failed for object
"+Persona.this);
            }
        }

        public void check_Persona_invariant0() {

            Date dateDataNaixement = Persona.this.dataNaixement;
            boolean bIsBefore = dateDataNaixement.isBefore(Date.now);
            if (!bIsBefore) {
                System.err.println("invariant 'invariant0' failed for object
```

```
"+Persona.this);
            }
        }

        public void check_Persona_invariant1() {

            int nEdat = Persona.this.edat;
            Date dateDataNaixement = Persona.this.dataNaixement;
            int nYearsOfDiference =
dateDataNaixement.yearsOfDiference(Date.now);
            float fAbs = BasicUtilities.abs(nYearsOfDiference);
            boolean bEquals = nEdat == fAbs;
            if (!bEquals) {
                System.err.println("invariant 'invariant1' failed for object
"+Persona.this);
            }
        }
    }

    public Date dataNaixement;
    public int edat;
    public Set universitatVirtual;
    protected String DNI;
    protected String nom;
    protected String cognom1;
    protected String cognom2;
    {

        OclType.registerInstance(this, Persona.class);

    }

}
```

Veiem que dins la classe principal ens genera la classe *ConstraintChecker* i com a funcions d'aquesta hi trobem una funció principal per fer totes les validacions anomenada '*checkConstraints*', i una funció per a cada una de les invariants.

Un cop vist el sistema que utilitza en la generació del codi, veiem que les altres invariants les construeix de manera semblant.

Definicions ('def')

Per acceptar les definicions, l'eina necessita que hi hagi la clàusula 'let', ja que altrament retorna un error. Per tant la sentència s'ha de corregir i afegir el 'let'. Així, i com a exemple, tenim la següent instrucció 'def' :

```
    context PlaDEstudis
    def:    let nombreEstudiants : Integer = estudiants->size()
```

En el codi generat veiem que ha construït una funció principal dins de la classe que calcularà el valor que hem definit :

```
    public int nombreEstudiants() {

        Set setEstudiants = PlaDEstudis.this.getEstudiants();
        int nSize = CollectionUtilities.size(setEstudiants);
        return nSize;

    }
```

No crea l'atribut a la classe, ni el '*setter*' per donar el valor, sinó que crea només la funció per consultar el valor.

Cos d'operacions ('body')

En el model de proves he definit la instrucció OCL següent per provar el funcionament de la clàusula '**body**' :

```
    context Semestre::getAssignatures() : Set(Assignatura)
    body:   self.estudiantSemestre.assignatures->asSet()
```

En l'editor de l'eina d'OCL, les paraules conegudes les mostra en negreta, però en aquest cas la paraula 'body' no. Tot i això sembla que reconeix la paraula.

Com comentava en l'apartat "Generació del codi Java", en aquest instrucció dóna un error al comprovar el metamodel UML15, ja que en la definició del model, no s'ha pogut definir correctament el valor a retornar '*Set(Assignatura*', tot i passar les validacions inicials tals com la validació de la instrucció OCL. Tot i l'error en la generació del codi, ens permet generar el codi Java. Observem-lo:

```
    public Set(Assignatura) getAssignatures() {

        class ConstraintChecker {

            Set(Assignatura) result;

            public void checkPreconditions() {

            }

            public void checkPostconditions() {
                check_postcondition();
            }

            public void check_postcondition() {

                Set setEstudiantSemestre =
Semestre.this.getEstudiantSemestreAdmissio();
                //evaluate 'collect(assignatures)':
                List bagCollect = CollectionUtilities.newBag();
                final Iterator iter = setEstudiantSemestre.iterator();
                while (iter.hasNext()) {
                    final EstudiantSemestre decl =
(EstudiantSemestre)iter.next();
                    Set setAssignatures = decl.getAssignatures();

                    bagCollect.add(setAssignatures);
                }
                bagCollect = CollectionUtilities.flatten(bagCollect);

                Set setAsSet = CollectionUtilities.asSet(bagCollect);
                if (!setAsSet) {
                    System.err.println("postcondition 'postcondition' failed
for object "+Semestre.this);
                }
            }
        }

        ConstraintChecker checker = new ConstraintChecker();
        checker.checkPreconditions();
        checker.result = internal_getAssignatures();

        checker.checkPostconditions();
        return checker.result;
    }

    public final Set getEstudiantSemestreAdmissio() {

        if (admissio == null) {
```

```
            return java.util.Collections.EMPTY_SET;
        }
        return java.util.Collections.unmodifiableSet(admissio);
    }

// ... saltem línies de codi ...

    private Set(Assignatura) internal_getAssignatures() {
        return null;
    }
```

Revisant-ho, veiem que ho ha generat com una 'post condició' dins el codi, i '*Set(Assignatura*' ho agafa com un tipus de dada, quan el tipus de dada hauria de ser 'Set'. Però si en el model UML definim que el valor de la funció és del tipus 'Set' ens retorna un error en la validació de la instrucció OCL, ja que no corresponen els valors de la instrucció que ha de ser '*Set(Assignatura)*' amb el valor del model UML que és 'Set'.

També veiem que dins de la funció '*getAssignatures()*' es defineix la classe '*ConstraintChecker*' amb l'atribut '*result*' que hauria de ser el valor retornat per la funció, però en aquest atribut mai s'arriba a posar-hi un valor; caldria que agafés el valor de la variable '*setAsSet*' del tipus 'Set'.

Degut a aquests errors, s'ha de tenir en compte que caldrà solucionar-ho, modificant i adaptant manualment el codi Java generat.

Pre i post condicions ('pre', 'post')

En el model de proves hi tenim 2 casos, un de simple i un de més complex. Passem a veure'ls per separat.

En el primer cas veiem que l'eina no ens accepta una pre condició buida o només amb comentari, i per tant l'he eliminada convertint-la en comentari :

```
    context Assignatura::getProfessor() : Docent
--* pre:    -- none
    post:   result = self.professor
```

```
    public Docent getProfessor() {

        class ConstraintChecker {

            Docent result;

            public void checkPreconditions() {
            }

            public void checkPostconditions() {
                check_postcondition();
            }

            public void check_postcondition() {

                Docent docentProfessor = Assignatura.this.getProfessor0();
                boolean bEquals = result.equals(docentProfessor);
                if (!bEquals) {
                    System.err.println("postcondition 'postcondition' failed
for object "+Assignatura.this);
                }
            }
        }
        ConstraintChecker checker = new ConstraintChecker();
        checker.checkPreconditions();
        checker.result = internal_getProfessor();
```

```
        checker.checkPostconditions();
        return checker.result;
    }

// ... saltem línies de codi ...

    public final Docent getProfessor0() {
        return professor;
    }

// ... saltem línies de codi ...

    private Docent internal_getProfessor() {
        return null;
    }

// ... saltem línies de codi ...

    public Docent professor;
    public Set consultors;
}
```

Però el codi generat sembla que no és correcte del tot, ja que en la variable '*result*' de la classe '*ConstraintChecker*' no s'hi posa el valor correctament. Si ens hi fixem, un cop ha executat le pre condició que no n'hi ha, executa aquesta línia "*checker.result = internal_getProfessor();*", carregarem el valor '*null*' a '*result*' a través de la funció cridada, i en la post condició, al comparar el valor amb la instrucció "*boolean bEquals = result.equals(docentProfessor);*", tindrem que no serà correcte ja que sempre estarem comparant amb el valor '*null*'.

En el segon cas a analitzar, "*context PlaDEstudis::admetreEstudiant(e : Estudiant)*", l'eina ens ha acceptat la instrucció OCL. Però recordem que ha donat unes alertes en la generació del codi ja que les instruccions '@pre' i 'oclIsNew' ja que no estan implementades correctament (veure apartat "*Generació del codi Java*"). Tot i això observem la instrucció i el codi generat:

```
    public void admetreEstudiant(Estudiant e) {

        class ConstraintChecker {

            public void checkPreconditions(Estudiant e) {
                check_precondition(e);
            }

            public void checkPostconditions(Estudiant e) {
                check_postcondition(e);
            }

            public void check_precondition(Estudiant e) {
                Set setEstudiants = PlaDEstudis.this.getEstudiants();
                boolean bIncludes =
CollectionUtilities.includes(setEstudiants, e);
                boolean bNot = !bIncludes;
                if (!bNot) {
                    System.err.println("precondition 'precondition' failed for
object "+PlaDEstudis.this);
                }
            }

            public void check_postcondition(Estudiant e) {
                Set setEstudiants = PlaDEstudis.this.getEstudiants();
                Set setEstudiants0 = PlaDEstudis.this.getEstudiants();
                Set setIncluding =
CollectionUtilities.including(setEstudiants0, e);
                boolean bEquals = setEstudiants.equals(setIncluding);
            Set setAllInstances = Ocl.getType(new
Class[]{Admissio.class}).allInstances();
                //evaluate 'one(Estudiant=e and PlaDEstudis=self)':
```

```
                int counter = 0;
                final Iterator iter = setAllInstances.iterator();
                while (counter<2 && iter.hasNext()) {
                    final Admissio iter1 = (Admissio)iter.next();
                boolean bEquals0 = Ocl.getType(new
Class[]{Estudiant.class}).equals(e);
                boolean bEquals1 = Ocl.getType(new
Class[]{PlaDEstudis.class}).equals(PlaDEstudis.this);
                    boolean bAnd1 = bEquals0 && bEquals1;

                    if (bAnd1) counter++;
                }
                boolean bOne = counter == 1;
                boolean bAnd0 = bEquals && bOne;
            Set setAllInstances0 = Ocl.getType(new
Class[]{Admissio.class}).allInstances();
                //evaluate 'any(Estudiant=e and PlaDEstudis=self)':
                Object temp = null;
                final Iterator iter0 = setAllInstances0.iterator();
                while (temp == null && iter0.hasNext()) {
                    Object temp0 = iter0.next();
                    Admissio iter1 = (Admissio)temp0;
                boolean bEquals2 = Ocl.getType(new
Class[]{Estudiant.class}).equals(e);
                boolean bEquals3 = Ocl.getType(new
Class[]{PlaDEstudis.class}).equals(PlaDEstudis.this);
                    boolean bAnd2 = bEquals2 && bEquals3;

                    if (bAnd2) temp = temp0;
                }
                Admissio admissioAny;
                if (temp == null) admissioAny = null;
                else admissioAny = (Admissio)temp;

                Admissio admi = admissioAny;
                boolean bOclIsNew = Ocl.isNew(admi);
                Date dateDataInici = admi.dataInici;
                boolean bIsEqual = dateDataInici.isEqual(Date.now);
                boolean bAnd4 = bOclIsNew && bIsEqual;
                boolean bEstudisAcabats = admi.estudisAcabats;
                boolean bEquals4 = bEstudisAcabats == false;
                boolean bAnd3 = bAnd4 && bEquals4;
                boolean bAnd = bAnd0 && bAnd3;
                if (!bAnd) {
                    System.err.println("postcondition 'postcondition' failed
for object "+PlaDEstudis.this);
                }
            }
        }
        ConstraintChecker checker = new ConstraintChecker();
        checker.checkPreconditions(e);
        checker.result = internal_admetreEstudiant(e);

        checker.checkPostconditions(e);
        return checker.result;
    }

// ... saltem línies de codi ...

    private void internal_admetreEstudiant(Estudiant e) {

    }
```

Inicialment veiem que sembla correcte, ja que totes les instruccions hi són reflectides, però segurament trobarem que algunes funcions de llibreries pròpies del programari com "*Ocl.isNew()*" deuen estar sense implementar o parcialment implementades segons ens han alertat les alertes en la generació del programari.

6.5. Conclusions de l'eina

L'eina OCLE 2.0.4 ens permet generar codi Java a través d'un model estàtic UML, i unes instruccions en OCL del mateix model.

L'eina s'instal·la fàcilment i els requeriments són mínims. El funcionament és molt intuïtiu a través d'un entorn gràfic. Amb el programari es pot obtenir també el manual d'usuari, on s'explica detalladament les principals operacions amb varis exemples, i les instruccions OCL de què disposa el compilador de l'eina.

Amb la utilització del programari he trobar alguns errors o '*bugs*', tals com el problema d'utilitzar noms de classes amb accents, i algun canvi imprevist en la representació gràfica de les associacions, ja que a vegades apareixien canviats de lloc el nom dels extrems de les associacions amb les seves multiplicitats, apareixent els noms en un extrem i les multiplicitats en l'altre. No són errors molt importants pel funcionament, però si que dificulten el treballar amb l'eina i distorsionen el model.

En quan al tractament de l'OCL en les especificacions de l'eina s'hi indica que suporta la versió 2.0 de OCL, però hi trobem unes mancances ja que no s'hi poden utilitzar totes les instruccions. Concretament :

- No es poden inicialitzar valors ('init').
- No es poden fer derivacions ('derive'), i la única solució és convertir-les en invariants.
- No genera correctament els cossos d'operacions ('body') i ho agafa com a post condicions dins el codi.

Les definicions ('def') són aplicables, però hi cal afegir el literal 'let'. Els atributs els genera com a funcions dins de la classe.

El tema més ben resolt són les invariants ('inv'), i les pre i post condicions ('pre' i 'post'), tot i que l'eina té pendent d'acabar d'implementar algunes instruccions com '@pre' i'oclIsNew'.

Un problema en l'anàlisi que he fet, i que no he sabut resoldre, ha estat la definició de tipus de dades 'Set' de determinades classes, concretament quan he definit el cos de la funció que havia de retornar dades del tipus '*Set(Assignatura)*'.

El codi generat conté les funcions necessàries per validar el model mitjançant les instruccions OCL implementades, però per executar les funcions caldrà implementar-les en les classes que utilitzaran les classes bàsiques generades pel model.

7. Anàlisi de l'eina: Octopus 2.2.0

7.1. Informació general de l'eina

Octopus (OCL Tool for Precise UML Specifications) 2.2.0 és un programari que s'integra a la plataforma IDE Eclipse [6], és un plugin d'Eclipse. Un cop instal·lat el programari a l'Eclipse funciona com una eina CASE UML que ofereix ple suport a l'OCL. Actualment el programari és un projecte Sourceforge [7], i els administradors d'aquest són Anneke Kleppe i Jos Warmer, els mateixos autors del llibre de referència sobre OCL de títol "*The Object Constraint Language Second Edition*" [8].

Entre les característiques principals indicades tenim:

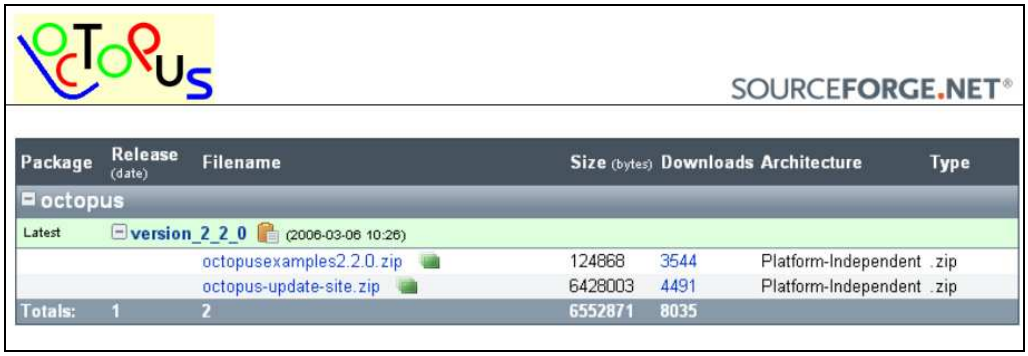
- És capaç de comprovar expressions OCL estàticament. En comprova la sintaxi, els tipus d'expressions, i l'ús correcte dels elements del model com associacions i atributs.
- És capaç de transformar el model UML i les expressions OCL en codi Java.
- Pot importar models UML utilitzant fitxers en format XMI generats per altres eines.
- Conforma plenament la versió 2.0 d'OCL. Tots els nous constructors, com regles de derivació i especificacions de valors inicials, estan completament suportats.
- Ofereix la possibilitat de veure les expressions en una sintaxi semblant a l'SQL.

7.2. Instal·lació del programari

Obtenció del programari

El programari el podem trobar a través de la mateixa web de Sourceforge i el podem baixar sense cap requeriment previ, ja que és distribuït sota llicència publica BSD,

Un cop accedim a la pàgina de descàrrega, veiem que també podem baixar uns exemples. Entre els quals hi ha tot el model "*The Royal and Loyal model*" desenvolupat en el llibre dels autors del programari [8].



Il·lustració 16: Octopus. Logotip i detall descàrrega del programari

Requeriments

El programari també és multi plataforma perquè s'executa a través de Java i Eclipse, i com a requeriments ens indiquen:

- Eclipse 3.1 i Java 1.5 (No funcionarà amb versions anteriors).

Jo ho he realitzat l'estudi amb Eclipse 3.2.2 i Java 1.5.0_12 sota Windows XP amb resultat satisfactori, tot i que els exemples que hi ha a l'ajuda (Help) de l'Eclipse un cop instal·lat el programari, difereixen en alguns menús i opcions disponibles.

Comentaris instal·lació

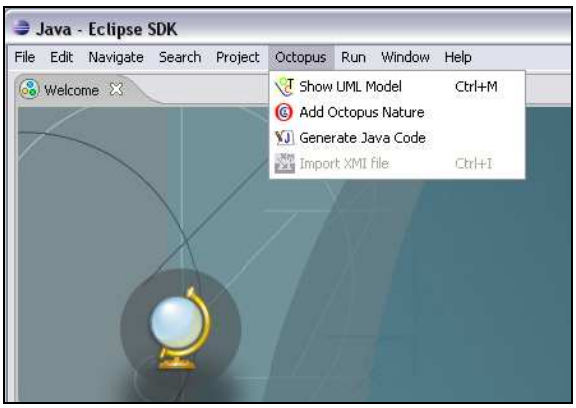
La instal·lació del programari a l'Eclipse es fa utilitzant els seus menús. Però hi ha dues opcions, instal·lar-lo a través d'Internet o a través del fitxer baixat. Jo ho he fet amb aquesta darrera opció.

El programari descarregat ve en un fitxer en format 'zip', en aquesta versió el nom del fitxer és "*octopus-update-site.zip*". S'ha de desempaquetar el fitxer mantenint l'estructura interna de directoris. La millor opció és descarregar-lo en un directori on hi hagi altres plugins d'Eclipse i mantenint el nom superior de l'estructura de directoris amb el nom " *octopus-update-site* ".

A través del menú de l'Eclipse, l'instal·lem:

Help → Software Updates → Find and install → Search for news features to install → New local site

Tot seguit cerquem el directori "*octopus-update-site*" del fitxer desempaquetat prèviament i el seleccionem, i continuem amb la instal·lació acceptant la pantalla de la llicència. Segurament surti una alerta (warning) referent a una verificació de signatura del programari, i si és el cas, l'acceptem. El programari s'instal·larà i al final ens farà reiniciar l'Eclipse per carregar la nova configuració d'aquest, i ja amb l'Octopus formant-ne part.

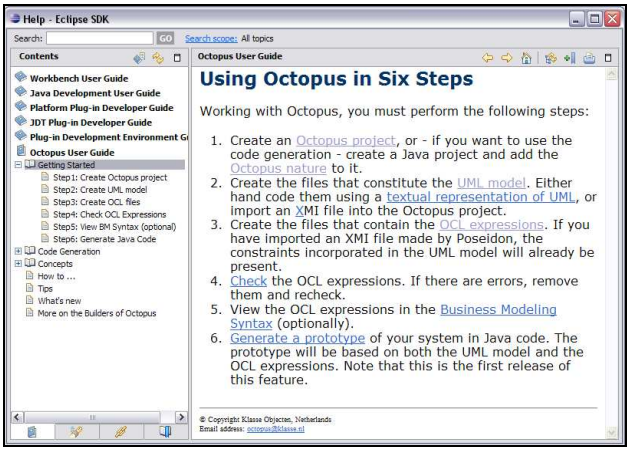


Il·lustració 17: Octopus. Programari formant part d'Eclipse

7.3. Utilització del programari

Seguidament adjunto unes explicacions generals i bàsiques del funcionament del programari, però la millor ajuda és la guia de l'usuari que s'ha instal·lat a l'Eclipse amb la instal·lació del programari, i que hi podem accedir a través del menú:

Help → Help Contents → "Octopus User Guide"



Il·lustració 18: Octopus. Guia de l'usuari del programari

Creació del projecte a Eclipse

Primerament cal crear un projecte a Eclipse. Per fer-ho a través del menú fem:

File → New → Project...

S'obrirà una finestra amb els diferents tipus de projectes que podem obrir:

Seleccionem "Plug-in project".

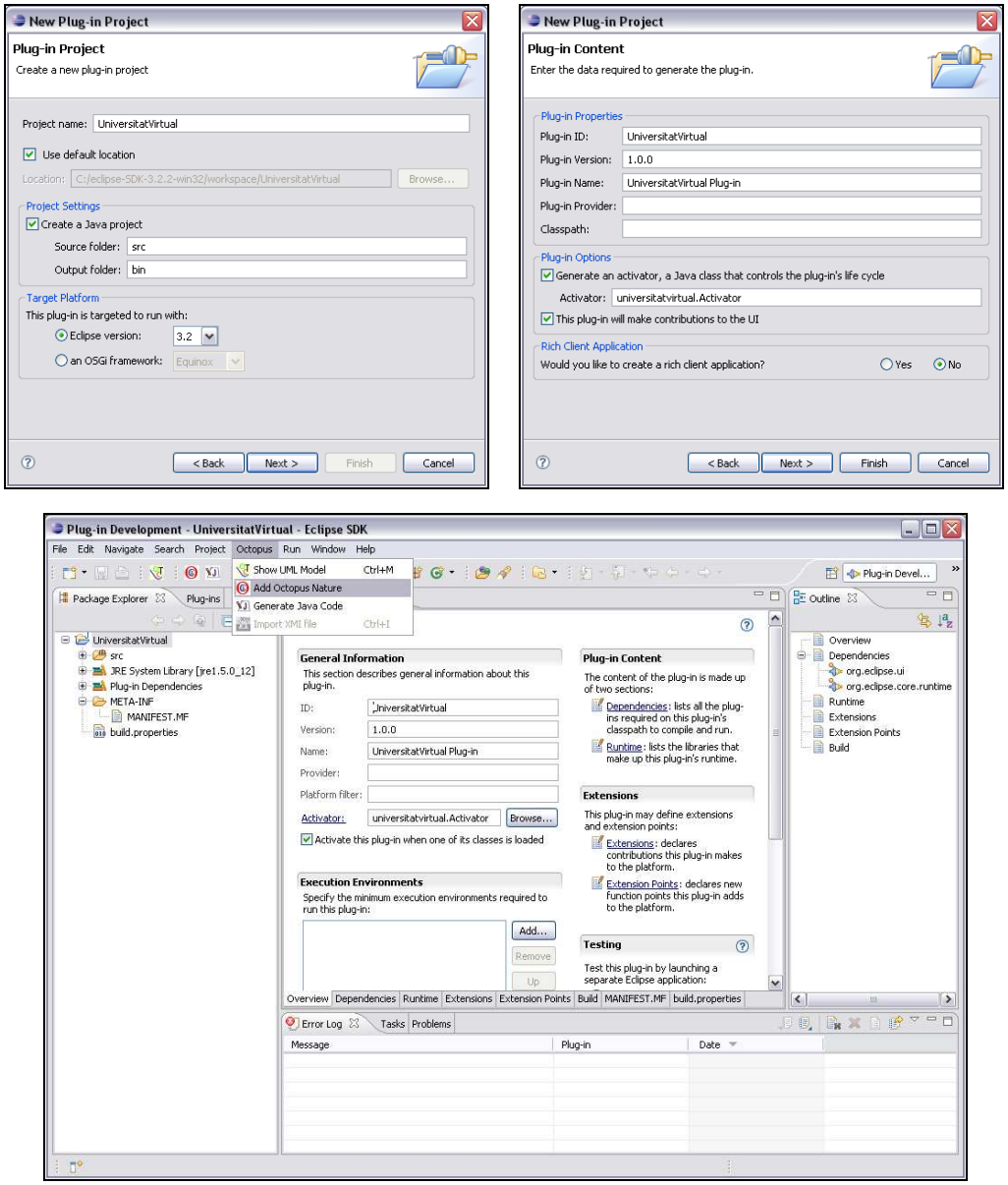
Tot seguit s'obrirà una finestra per entrar el nom al projecte, ho fem i premem "*Next*". A la propera finestra podem seleccionar "*Finish*" si no volem canviar altres opcions, en principi no cal.

Si surt una finestra demanat obrir la perspectiva associada al projecte, acceptem.

Ara només cal afegir el tipus de projecte Octopus, afegint el "*Octopus Nature*", per fer-ho seleccionem el projecte, i a través del menú fem:

Octopus → "Add Octopus Nature"

I això ens afegirà les carpetes 'model' i 'expressions' al projecte.



Il·lustració 19: Octopus. Creació del projecte a Eclipse

Creació del model UML

L'eina no disposa d'un entorn gràfic per crear el model UML, però té 2 possibilitats:

- Importar el model creat per un altre programari en format XMI compatible.
- Generar el model UML a través d'un fitxer de text i utilitzant un format propi per representar l'UML de manera textual. Dins l'ajuda del programari trobem explicat i detallat com fer-ho.

Al provar la primera opció amb diferents fitxers XMI, i generats amb diferents programaris externs, m'he trobat sempre amb el mateix error, com si faltés algun modul a l'Eclipse:

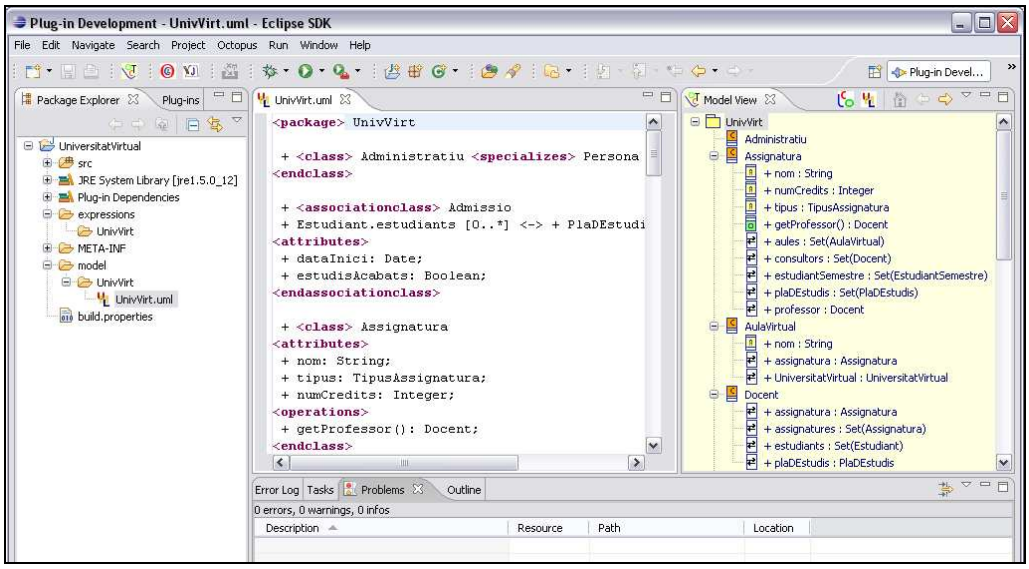
/XSLT_TEMP is not present

Veient les dificultats amb la primera opció, he passat a la segona, i he generat el model sense problemes.

Per generar el model amb el format propi per representar l'UML de manera textual, primerament cal crear una carpeta amb el nom del paquet (*package*) dins de la carpeta 'model' i dins la carpeta 'expressions'. El fitxer que conté el model té extensió '*.uml*' i s'ha de situar dins de la carpeta *'/model/${nom_paquet}/*'.

Un cop creat el model, el podem veure ("*Model view*")en una representació gràfica a pròpia a través del menú:

Octopus → Show UML model



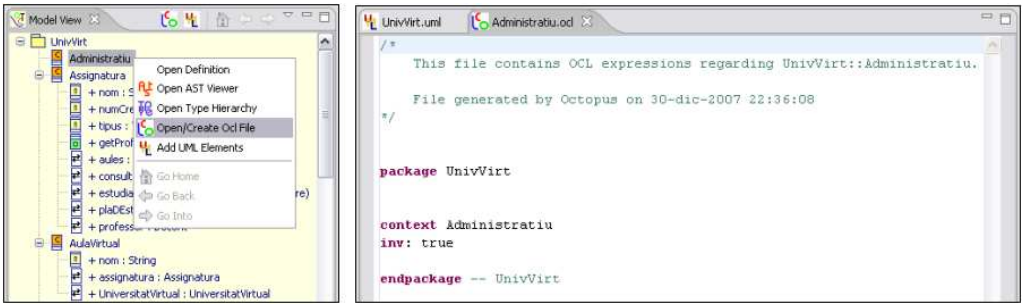
Il·lustració 20: Octopus. Interfície gràfica

Entrada instruccions OCL

Per especificar les instruccions OCL l'eina es basa en la utilització d'un fitxer de text per a cada classe o context que en tingui. L'eina proporciona facilitats, tal com la generació automàtica del fitxer a través de la vista del model.

Els fitxers es generaran a la carpeta *'/expressions/${nom_paquet}/*' amb l'extensió "*.ocl*" .

També tenim l'opció de generar-los, i/o de manipular-los, amb un altre editor de text i guardar-los amb el nom del context o classe en el mateix directori.

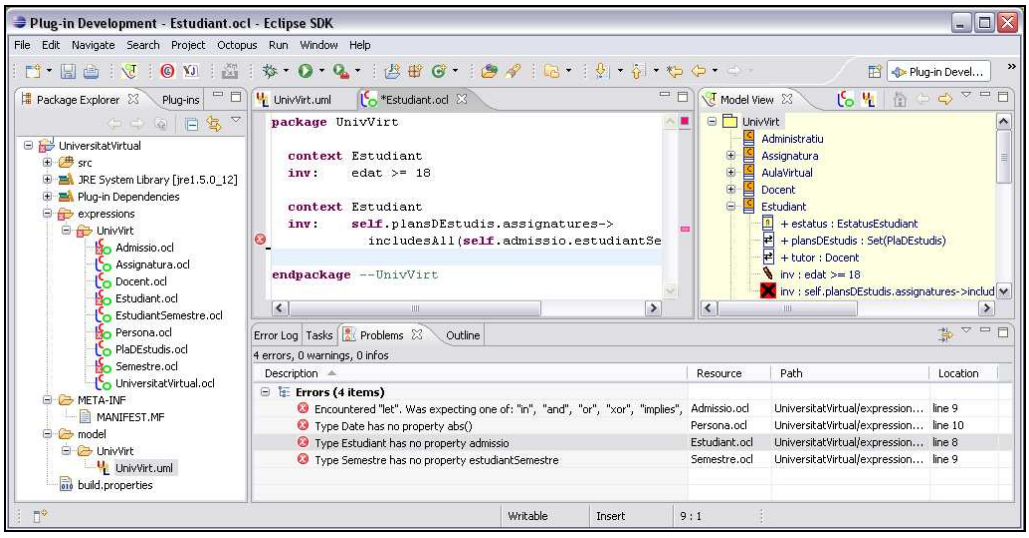


Il·lustració 21: Octopus. Entrada d'instruccions OCL a través de la vista del model

Un cop entrades les instruccions OCL també les podrem veure en la interpretació gràfica del model.

Depuració d'errors

Un cop disposem del model UML i les instruccions OCL del model, l'eina ens detecta els errors de sintaxis i de coherència amb el model UML entrat indicant el tipus i la localització dins el fitxer de les instruccions.



Il·lustració 22: Octopus. Depuració d'errors

Un error que he trobat en dues instruccions OCL era la no navegabilitat entre dues classes associatives consecutives dependent de la situació, per exemple i seguint el nostre model UML "*UniversitatVirtual*", tenim una classe associativa '*Admissio*' entre '*Estudiant*' i '*PlaDEstudis*', i una altra classe associativa '*EstudiantSemestre*' entre '*Admissio*' i '*Assignatura*'. Ens trobem que si bé podem navegar entre '*Estudiant*' i '*Admissio*', no podem fer-ho de '*Estudiant*' a '*EstudiantSemestre*' passant per '*Admissio*'. Però si que podem fer-ho des de la classe associativa més profunda, o sigui podem navegar des de '*EstudiantSemetre*' fins a '*Estudiant*' i passant per '*Admissio*'. Sembla ser que és la manera en què l'eina construeix les associacions entre les classes a partir de les classes associatives. En l'anàlisi d'aquesta eina he redefinint les instruccions canviant el context, Concretament:

```
--  context Estudiant
--  inv:    self.plansDEstudis.assignatures->
--          includesAll(self.admissio.estudiantSemestre.assignatures)
```

```
    context EstudiantSemestre
    inv:    self.admissio.plansDEstudis.assignatures->
            includesAll(self.assignatures->asSet())
```

```
--  context Admissio
--  inv:    let credTroncals : Integer = self.estudiantSemestre.matricula->
--          select((estatusAssignatura = EstatusAssignatura::aprovada or
--                  estatusAssignatura = EstatusAssignatura::convalidada) and
--                  assignatures.tipus =
--            TipusAssignatura::troncal).assignatures.numCredits->sum()

-- ...salto codi, no cal exposar-lo tot en aquest exemple, és similar ...

--         estudisAcabats = true implies
--         credTroncals >= self.plansDEstudis.numCreditsTroncals          and
--         credObligatoris >= self.plansDEstudis.numCreditsObligatoris    and
--         credOpcionals >= self.plansDEstudis.numCreditsOpcionals        and
--         credLliureElec >= self.plansDEstudis.numCreditsLliureEleccio
```

```
  context Matricula
  inv:    let credTroncals : Integer = self->
            select((estatusAssignatura = EstatusAssignatura::aprovada or
                    estatusAssignatura = EstatusAssignatura::convalidada) and
                    assignatures.tipus =
              TipusAssignatura::troncal).assignatures.numCredits->sum(),

-- ...salto codi, no cal exposar-lo tot en aquest exemple, és similar ...

  in self.estudiantSemestre.admissio.estudisAcabats = true implies
       credTroncals >=
         self.estudiantSemestre.admissio.plansDEstudis.numCreditsTroncals
   and credObligatoris >=
         self.estudiantSemestre.admissio.plansDEstudis.numCreditsObligatoris
   and credOpcionals >=
         self.estudiantSemestre.admissio.plansDEstudis.numCreditsOpcionals
   and credLliureElec >=
         self.estudiantSemestre.admissio.plansDEstudis.numCreditsLliureEleccio
```

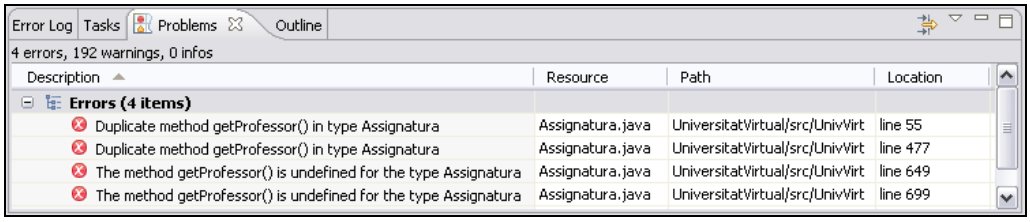
Altres errors trobats en l'aplicació del codi OCL del model que provem ja són més propis de les instruccions i els detallaré en l'anàlisi del codi generat.

Generació del codi Java

Un cop tenim el model UML creat i les instruccions OCL preparades, es pot procedir a generar el codi Java. Primer hem de seleccionar el projecte de l'Eclipse, i tot seguit l'opció del menú :

Octopus → Generate Java Code

I directament ens intentarà generar el codi en la carpeta "*/src/${nom_paquet}*", però si troba errors ens els indicarà :



Il·lustració 23: Octopus. Errors en la generació del codi Java

En el nostre exemple, i degut a que l'eina sempre genera tots els '*getters*' i '*setters*', ens troba com a duplicada l'operació '*getProfessor()*' que també hem definit al model. Per solucionar-ho anomenem diferent a l'operació del model en aquest anàlisi, i utilitzem '*getProfessorAssignatura()*' en el fitxer '*.uml*' on hi ha la definició del model, i en el fitxer '*Assignatura.ocl*' on hi ha les instruccions OCL.

7.4. Anàlisi del codi generat

Valors inicials ('init')

La instrucció '**init**' és interpretada correctament i el seu codi s'inclou en les funcions dels constructors de la classe corresponent al context. Vegem el 2 casos que hi tenim. En aquest primer cas el constructor amb paràmetre li sobraria la primera definició del nom "*this.setNom(nom)*", però per altra banda la seva presència ens pot servir en algunes eines Java per evitar una alerta o warning per no utilitzar el paràmetre:

```
    /** Default constructor for UniversitatVirtual
     */
    public UniversitatVirtual() {
        this.setNom( "UOC" );
        if ( usesAllInstances ) {
            allInstances.add(this);
        }
    }

    /** Constructor for UniversitatVirtual
     *
     * @param nom
     */
    public UniversitatVirtual(String nom) {
        super();
        this.setNom(nom);
        this.setNom( "UOC" );
        if ( usesAllInstances ) {
            allInstances.add(this);
        }
    }
```

En el següent cas tot sembla correcte:

```
    /** Constructor for Admissio. Always use this constructor, do NOT use the
one without parameters.
     *
     * @param a
     * @param b
     */
    public Admissio(Estudiant a, PlaDEstudis b) {
        if ( a != null && b != null ) {
            this.f_estudiants = a;
            a.z_internalAddToAdmissio(this);
            this.f_plansDEstudis = b;
            b.z_internalAddToAdmissio(this);
        }
        this.setSemestre( collectionLiteral1() );
        if ( usesAllInstances ) {
            allInstances.add(this);
        }
    }

    //...

    /** Implements the setter for '+ semestre : Set(Semestre)'
     *
     * @param par
     */
    public void setSemestre(Set par) {
        // make copy to avoid a ConcurrentModificationException;
        List xx = new ArrayList(this.f_estudiantSemestre);
        Iterator it = xx.iterator();
        while ( it.hasNext() ) {
            EstudiantSemestre elem = (EstudiantSemestre) it.next();
            elem.clean();
        }
        if ( par != null ) {
```

```
            it = par.iterator();
            while ( it.hasNext() ) {
                Semestre elem = (Semestre) it.next();
                this.addToSemestre(elem);
            }
        }
    }

    //...

    /** Implements Set{}
     */
    private Set collectionLiteral1() {
        Set /*(OclVoid)*/ myList = new HashSet( /*OclVoid*/);
        return myList;
    }
```

Derivacions ('derive')

La instrucció '**derive**' també funciona bé. Primer defineix la variable com a atribut, i després la funció '*getter*' de la variable, i en aquesta funció hi codifica el codi de la instrucció OCL. Vegem un dels casos que hi tenim, ja que els altres són semblants:

```
    private int f_edat = 0;

    // ...

    /** Implements the getter for feature '+ edat : Integer'
     */
    public int getEdat() {
        return
Math.abs(this.getDataNaixement().yearsOfDiference(Date.getNow()));
    }
```

La variable definida com a atribut de la classe no s'utilitza en el codi, però en la funció generada hi veiem les instruccions OCL codificades correctament.

Invariants ('inv')

L'eina ha acceptat totes les invariants del test de proves. Analitzem-ho amb detall el codi que ha generat en la classe Persona on hi tenim aquestes 2 invariants :

```
    /** Implements ->isUnique( i_Persona : Persona | i_Persona.DNI )
     */
    private boolean isUnique1() {
        Iterator it = Persona.allInstances().iterator();
        List _valuesInSource = new ArrayList();
        while ( it.hasNext() ) {
            Persona i_Persona = (Persona) it.next();
            if ( _valuesInSource.contains(i_Persona.getDNI()) ) {
                return false;
            }
            _valuesInSource.add(i_Persona.getDNI());
        }
        return true;
    }

    /** Implements Persona.allInstances()->isUnique( i_Persona : Persona |
i_Persona.DNI )
     */
    public void invariant_Persona1() throws InvariantException {
        boolean result = false;
        try {
            result = isUnique1();
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
        if ( ! result ) {
            String message = "invariant Persona.allInstances()->isUnique(
i_Persona : Persona | i_Pe... ";
            message = message + "is broken in object '";
            message = message + this.getIdString();
            message = message + "' of type '" + this.getClass().getName() +
"'";
            throw new InvariantException(this, message);
        }
    }

    /** Implements self.dataNaixement.isBefore(Date::now)
     */
    public void invariant_Persona2() throws InvariantException {
        boolean result = false;
        try {
            result = this.getDataNaixement().isBefore(Date.getNow());
        } catch (Exception e) {
            e.printStackTrace();
        }
        if ( ! result ) {
            String message = "invariant self.dataNaixement.isBefore(Date::now)
";
            message = message + "is broken in object '";
            message = message + this.getIdString();
            message = message + "' of type '" + this.getClass().getName() +
"'";
            throw new InvariantException(this, message);
        }
    }

    /** Checks all invariants of this object and returns a list of messages
about broken invariants
     */
    public List checkAllInvariants() {
        List /* InvariantError */ result = new ArrayList /* InvariantError
*/();
        try {
            invariant_Persona1();
        } catch (InvariantException e) {
            result.add(new InvariantError(e.getInstance(), e.getMessage()));
        }
        try {
            invariant_Persona2();
        } catch (InvariantException e) {
            result.add(new InvariantError(e.getInstance(), e.getMessage()));
        }
        return result;
    }
```

Observem també el detall de les invariants de la classe *EstudiantSemestre* on hi ha una invariant interessant com és la que utilitza la instrucció '*includesAll*'. En l'altra, on validem que el nombre de crèdits matriculat en un semestre per un alumne no és superior al permès al pla d'estudis, l'hem hagut d'adaptar ja que ens donava un error en l'eina :

```
    context EstudiantSemestre
--  inv:    self.assignatures.numCredits->sum() <=
--  ******* Error al generar el codi Java:
--  ******* "The method getNumCredits() is undefined for the type
--  ******* Set", corregim amb 'collect'. Ho hem canviat per:
    inv:    self.assignatures->collect( numCredits )->sum() <=
              self.admissio.plansDEstudis.maxCreditsMatriculaSemestre
```

```
    /** Implements ->collect( i_Assignatura : Assignatura |
i_Assignatura.numCredits )
     */
    private List collect1() {
```

```
        List /*(Integer)*/ result = new ArrayList( /*Integer*/);
        Iterator it = Stdlib.objectAsSet(this.getAssignatures()).iterator();
        while ( it.hasNext() ) {
            Assignatura i_Assignatura = (Assignatura) it.next();
            Object bodyExpResult = new Integer(i_Assignatura.getNumCredits());
            if ( bodyExpResult != null ) result.add( bodyExpResult );
        }
        return result;
    }

    /** Implements .sum() on ->collect( i_Assignatura : Assignatura |
i_Assignatura.numCredits )
     */
    private int sum2() {
        int result = 0;
        Iterator it = collect1().iterator();
        while ( it.hasNext() ) {
            Integer elem = (Integer) it.next();
            result = result + elem.intValue();
        }
        return result;
    }

    /** Implements (self.assignatures.asSet()->collect( i_Assignatura :
Assignatura | i_Assignatura.numCredits )->sum()) <=
self.admissio.plansDEstudis. maxCreditsMatriculaSemestre
     */
    public void invariant_EstudiantSemestre1() throws InvariantException {
        boolean result = false;
        try {
            result = (sum2() <=
this.getAdmissio().getPlansDEstudis().getMaxCreditsMatriculaSemestre());
        } catch (Exception e) {
            e.printStackTrace();
        }
        if ( ! result ) {
            String message = "invariant (self.assignatures.asSet()->collect(
i_Assignatura : Assigna... ";
            message = message + "is broken in object '";
            message = message + this.getIdString();
            message = message + "' of type '" + this.getClass().getName() +
"'";
            throw new InvariantException(this, message);
        }
    }

    /** Implements self.admissio.plansDEstudis.assignatures->
includesAll(self.assignatures.asSet()->asSet())
     */
    public void invariant_EstudiantSemestre2() throws InvariantException {
        boolean result = false;
        try {
            result = this.getAdmissio().getPlansDEstudis().getAssignatures().
containsAll(Stdlib.collectionAsSet(Stdlib.objectAsSet(this.getAssignatures()))
);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if ( ! result ) {
            String message = "invariant
self.admissio.plansDEstudis.assignatures->includesAll(self.a... ";
            message = message + "is broken in object '";
            message = message + this.getIdString();
            message = message + "' of type '" + this.getClass().getName() +
"'";
            throw new InvariantException(this, message);
        }
    }

    /** Checks all invariants of this object and returns a list of messages
about broken invariants
     */
    public List checkAllInvariants() {
```

```
        List /* InvariantError */ result = new ArrayList /* InvariantError
*/();
        try {
            invariant_EstudiantSemestre1();
        } catch (InvariantException e) {
            result.add(new InvariantError(e.getInstance(), e.getMessage()));
        }
        try {
            invariant_EstudiantSemestre2();
        } catch (InvariantException e) {
            result.add(new InvariantError(e.getInstance(), e.getMessage()));
        }
        return result;
    }
```

Un cop vist el sistema que utilitza en la generació del codi, veiem que les altres invariants les construeix de manera semblant.

Veiem que dins la classe principal (*Persona* o *EstudiantSemestre*) ens genera la funció '*checkAllInvariants()*' per comprovar totes les invariants i retornar el resultat en una llista. Aquesta funció no es executada dins la classe pel que s'haurà d'executar quan es faci la implementació d'aquesta. En el codi adjunt hi he ressaltat en negreta les funcions indicades i les cridades a partir d'aquesta funció principal del la classe.

Un detall a comentar és que a les classes on no hi ha invariants, hi genera també la funció '*checkAllInvariants()*', per retornar un '*List*' buit:

```
    /** Checks all invariants of this object and returns a list of messages
about broken invariants
     */
    public List checkAllInvariants() {
        List /* InvariantError */ result = new ArrayList /* InvariantError
*/();
        return result;
    }
```

Definicions ('def')

Davant la següent instrucció '**def**', l'eina ens ha afegit un atribut nou a la classe i una funció del tipus '*getter*' on hi codifica el contingut de la instrucció :

```
    private int f_nombreEstudiants = 0;

// ...

    /** Implements the getter for feature '+ nombreEstudiants : Integer'
     */
    public int getNombreEstudiants() {
        return this.getEstudiants().size();
    }
```

No es fa un ús intern de l'atribut, ni crea el setter per donar el valor, només crea la funció per consultar el valor a través de l'operació.

Cos d'operacions ('body')

La instrucció que tenim al model de proves ens ha generat el següent codi :

```
    /** Implements the user defined operation '+ getAssignatures() :
Set(Assignatura)'
     */
    public Set getAssignatures() {
```

```
        return Stdlib.collectionAsSet(collect1());
    }

// ...

    /** Implements ->collect( i_EstudiantSemestre : EstudiantSemestre |
i_EstudiantSemestre.assignatures )
     */
    private List collect1() {
    List /*(Assignatura)*/ result = new ArrayList( /*Assignatura*/);
        Iterator it = this.getEstudiantSemestre().iterator();
        while ( it.hasNext() ) {
            EstudiantSemestre i_EstudiantSemestre = (EstudiantSemestre)
it.next();
            Object bodyExpResult = i_EstudiantSemestre.getAssignatures();
            if ( bodyExpResult != null ) result.add( bodyExpResult );
        }
        return result;
    }
```

On podem comprovar que ho ha fet correctament, definint el cos de l'operació indicat amb l'OCL.

Pre i post condicions ('pre', 'post')

Dels dos casos que tenim en el model de proves, tenim que en el primer, i tal com hem comentat en l'apartat "Generació del codi Java", hem canviat de nom la funció *'getProfessor()*' del context Assignatura pel nom '*getProfessorAssignatura()*' degut a tenir una associació entre les classes Assignatura i Docent amb el nom de '*professor*', ja que per defecte l'eina ens genera sempre totes les funcions 'getters' i 'setters', i ja ens ha generat una funció amb el nom indicat. Ara un cop generat el codi :

```
    /** Implements the user defined operation '+ getProfessorAssignatura() :
Docent'
     */
    public Docent getProfessorAssignatura() {
        Docent result = null;
        return result;
    }
```

trobem que no és correcte del tot, ja que la funció sempre ens retornarà el valor '*null*', quan hauria de retornar el *Docent* corresponent.

En l'altre cas, trobem que només ens comprova la pre condició, però no la post condició.

```
    /** Implements the user defined operation '+ admetreEstudiant( e:
Estudiant )'
     *
     * @param e
     */
    public void admetreEstudiant(Estudiant e) {
        assert !this.getEstudiants().contains(e) :
           "precondition: 'not self.estudiants->includes(e)' has failed.";
    }
```

7.5. Conclusions de l'eina

L'eina Octopus 2.2.0 ens permet generar codi Java a través d'un model estàtic UML i unes instruccions en OCL del mateix model.

Octopus és un plugin de la plataforma Eclipse, i es requereix la plataforma Eclipse 3.1 i Java 1.5 (o superiors) per poder funcionar. La mateixa plataforma aporta facilitats per poder-lo instal·lar des dels seus menús. Un cop instal·lat, trobem la guia de l'usuari una eina útil i ben desenvolupada.

No té un entorn gràfic per poder crear els models UML, i es basa en la importació de fitxers en format XMI. En la guia de l'usuari aconsellen determinades eines que podrien generar fitxers compatibles com Poseidon. Però a la pràctica m'he trobat que en varis fitxers XMI creats amb diferents eines externes no són compatibles i no es poden importar. Com a alternativa als fitxers XMI per crear el models, l'eina té un format propi a través de fitxers de text, que és el sistema que he utilitzat sense dificultats. Un cop és té un model UML entrat, l'eina el pot interpretar gràficament i mostrar-lo.

Per entrar les instruccions OCL del model, es basa en la generació d'un fitxer de text per a cada classe o context del model. L'eina proporciona facilitats, com accedir o crear el fitxer d'instruccions a través de la vista del model interpretat gràficament i/o mostrar les instruccions en un editor propi.

Amb la utilització del programari he trobar algun problema com la no construcció de totes les associacions entre dues o més classes associatives consecutives, el que provoca la no navegabilitat en determinats sentits entre les classes.

En la generació del codi Java, cal destacar que genera un codi clar i ben estructurat, fa la creació de totes les funcions '*setters*' i '*getters*' dels atributs i associacions, i documenta unes descripcions clares a les funcions generades.

En quan al tractament de l'OCL del test que hi he aplicat, cal remarcar que gairebé totes les instruccions són ben interpretades i en genera el codi correctament, tot i que he hagut d'adaptar alguna instrucció puntualment, com el cas d'afegir una instrucció '*collect*' a una invariant, o l'adaptació de les instruccions al problema citat entre classes associatives on he hagut de canviar el context perquè la instrucció OCL fos ben interpretada. Només he trobat un punt feble i important, que és en la generació del codi de les post condicions, on ho no en genera el codi correctament, o simplement no el genera.

El codi generat conté les funcions necessàries per validar el model mitjançant les instruccions OCL implementades. Algunes funcions s'integren amb el codi i s'executaran implícitament, com les instruccions d'inicialitzar valors '*init*' que s'integren en els constructors, però moltes altres com les invariants '*inv*', caldrà implementar-les explícitament cridant a les funcions en les classes que utilitzaran les classes bàsiques generades pel model.

8. Resum comparatiu de les eines analitzades

Especificacions segons fabricant	**JCGG Dresden OCL2 Toolkit 1.2**	**OCLE 2.0.4**	**Octopus 2.2.0**
Plataforma	Independent	Independent	Independent
Llicència	GNU o LGPL	No especificat, deixen baixar còpia	BSD License
UML suportat	1.5	1.5	No especificat
OCL suportat	2.0	2.0	2.0
XMI suportat / ús	Ha d'importar models XMI d'unes determinades eines	Intercanvi amb altres eines en versions 1.0 i 1.1	Pot importar determinats fitxers
Codi que pot generar	Java	Java	Java
Requeriments programari	Java SDK 1.5	Java JRE 1.4 o superior	Java 1.5 i Eclipse 3.1 o superiors
Requeriments maquinari	Mínim 400 MHz de CPU i 256 MB de RAM	Mínim PII-300 i 128 MB RAM (Recomanat PII-500 i 256 MB RAM)	No especificat

Suport a instruccions OCL segons anàlisi treball	**JCGG Dresden OCL2 Toolkit 1.2**	**OCLE 2.0.4**	**Octopus 2.2.0**
Valors inicials '**init**'	No	No	Sí
Derivacions '**derive**'	No	No	Sí
Invariants '**inv**'	Sí, però amb moltes limitacions	Sí	Sí
Definicions '**def**'	No	Sí, però s'hi ha d'afegir el 'let'	Sí
Cos d'operacions '**body**'	No	Sí, però ho tracta com una 'post'	Sí
Pre i post condicions '**pre**' i '**post**'	Sí, però amb moltes limitacions	Sí, però amb algunes limitacions	Sí, però alguna 'post' no es genera

Altres comentaris	**JCGG Dresden OCL2 Toolkit 1.2**	**OCLE 2.0.4**	**Octopus 2.2.0**
Codi generat	No genera tot el codi del model, només afegeix les restriccions OCL al final de les classes preexistents a través de la injecció. És un codi sense marges i difícil de llegir i mantenir.	Genera el codi estructurat del model i el de les restriccions, però posa els atributs al final de la classe, genera els constructors buits, i no genera tots els 'getters'.	Genera el codi amb tots els 'getters' i constructors amb el seu codi. És un codi ben estructurat i fàcil de llegir i mantenir. Afegeix comentaris a les funcions OCL que implementa sobre el que fan.

9. Conclusions del treball

Amb aquest treball he aprés el llenguatge OCL que desconeixia, i he investigat moltes eines que en les seves característiques indicaven que l'implementaven per a la generació automàtica de codi, concretament m'he centrat e les eines de generació automàtica de codi Java. He seleccionat una representació de les mateixes per analitzar-les i estudiar-les amb detall, però després de veure que moltes no implementaven l'OCL d'una manera real, sinó només textual i com a comentaris. Altres eines les he hagut de descartar també perquè tenien un cost prohibitiu per a un estudiant, i sí que de determinades eines amb versions o edicions més obertes, més assequibles, però estan molt limitades en les seves capacitats.

L'OCL és un complement important que ja forma part de l'UML actual, i tal i com diuen Jos Warmer i Anneke Kleppe en el seu llibre, de referència en aquest món de l'OCL, "*The Object Constraint Language Second Edition*" [8], anem avançant cap a un nou sistema més automàtic de generació dels programes a través dels models, i l'OCL és una peça clau que ajudarà a avançar en aquest sentit.

Actualment, a part de l'estàndard de l'OMG [3] i el llibre abans citat [8], no hi ha gaire més documentació completa sobre l'OCL, però aquests són suficients per aprendre'n, tot i que estic segur que ens els propers anys la cosa ha de canviar.

Al principi del treball em pensava que trobaria moltes eines, però actualment d'eines tampoc n'hi ha gaires que implementin i donin suport a l'OCL per generar automàticament codi Java. N'he seleccionat tres, que són curiosament de tres maneres diferents de fer :

- Java Code Generator GUI de Dresden OCL2 Toolkit 1.2 és una eina que necessita el model UML en format XMI i el codi Java prèviament generat, per injectar-li les restriccions OCL.
- OCLE 2.0.4 és una aplicació amb el seu propi entorn gràfic per generar el model UML i amb editor d'instruccions OCL.
- Octopus 2.2.0 és un plugin d'Eclipse, amb un llenguatge textual propi per generar el model UML.

A partir del model o joc de proves que vaig confeccionar inicialment, amb una mostra variada d'expressions OCL, he anat estudiant les diferents eines i el seu codi generat per validar el grau de compliment segons les instruccions. En aquest treball s'adjunten i expliquen detalladament els anàlisis fets, així com un resum comparatiu de les mateixes, on podem observar diferents graus d'assoliment.

Podem comprovar com l'eina Octopus 2.2.0 és la millor de les analitzades, tant pel compliment amb l'OCL, com pel tipus de codi generat.

Estic segur que cada eina és un món, i que noves versions i noves eines ens poden aportar noves millores i noves funcionalitats en aquest tipus de programari. Espero que d'igual manera que aquest treball que m'ha introduït al món de la generació automàtica del programari, pugui ser útil a altres persones que amb el mateix descobreixin l'OCL i les eines que l'implementen per treure'n un bon rendiment.

Glossari

CASE
"Computer Aided Software Engineering", o Enginyeria de programari assistida per ordinador. Aplicacions informàtiques destinades a augmentar la productivitat en el desenvolupament de programari.

Metamodel
El metamodel en UML és una descripció d'UML en UML. Descriu els objectes, atributs, i relacions necessaris per representar els conceptes d'UML en aplicacions de programari. [9]

OCL
L'"Object Constraint Language" és un llenguatge formal especificat per l'OMG per descriure expressions en els models UML. Les seves expressions especifiquen condicions invariants que s'han de mantenir en el sistema que s'està modelant, o especifiquen consultes sobre els objectes descrits en el model. [3]

OMG
L'Object Management Group (OMG), creat el 1989, és una organització no lucrativa en què participen més de vuit-centes grans empreses de programari, de maquinari, usuàries i consultores, i té la finalitat de fomentar l'ús de la tecnologia d'objectes i impulsar la introducció de programari orientat a objectes que ofereixi reusabilitat, portabilitat i interoperabilitat en entorns distribuïts heterogenis. [10]

Plugin
Un connector o plugin (o plug-in -en anglès "endollar"-, també conegut com a extensió, addin, add-in, addon o add-on) és una aplicació informàtica que interactua amb una altra aplicació per a aportar-li una funció o utilitat addicional, generalment molt específica. Aquesta aplicació addicional és executada per part de l'aplicació principal. [11]

UML
L'Unified Modeling Language (UML) és un model per a la construcció de programari orientat a objectes que ha estat proposat com a estàndard d'ISO per l'OMG. Consta d'un conjunt de tipus de diagrames interrelacionats, dins els quals s'empren elements del model, que serveixen per a descriure diversos aspectes de l'estructura i la dinàmica del programari. [10]

TFC
Treball final de carrera.

XMI
"XML Metadata Interchange" és un estàndard d'OMG per intercanviar metadades o informacions descriptives en XML. L'ús més comú és com el format d'intercanvi de models UML entre aplicacions. [12]

Bibliografia i referències

1. Warmer, Jos, i Kleppe, Anneke. *The professional site of Jos Warmer and Anneke Kleppe*. En línia. Internet. 20/desembre/2005. Accessible a http://www.klasse.nl/ocl/ocl-introduction.html (Consulta 12/octubre/2007).

2. OMG. *Catalog of OMG Modeling and Metadata Specifications.* En línia. Internet. 20/setembre/2007. Accessible a http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL (Consulta 02/octubre/2007).

3. OMG. *Object Constraint Language. OMG Available Specification. Version 2.0.*(Maig/2006) En línia. http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf. (Consulta 22/setembre/2007).

4. *Dresden OCL Toolkit*. En línia. Internet. 01/octubre/2007. Accessible a http://dresden-ocl.sourceforge.net/ (Consulta 28/novembre/2007).

5. *OCLE Object Constraint Language Environment version 2.* En línia. Internet. Accessible a http://lci.cs.ubbcluj.ro/ocle/overview.htm (Consulta 27/octubre/2007).

6. *Eclipse - an open development platform.* En línia. Internet. Accessible a http://www.eclipse.org/ (Consulta 18/juny/2007).

7. *Octopus: OCL Tool for Precise Uml Specifications.* En línia. Internet. 23/febrer/2006. Accessible a http://octopus.sourceforge.net/ (Consulta 30/novembre/2007).

8. Warmer, Jos, i Kleppe, Anneke. *The Object Constraint language second Edition: Getting Your Models Ready for MDA.* Boston : Addison-Wesley, cop., 2003.

9. Wikipedia. *UML class metamodel*. En línia. Internet. 02/octubre/2007. Accessible a http://en.wikipedia.org/wiki/UML_class_metamodel (Consulta 13/gener/2008).

10. Campderrich Falgueras, Benet. "Introducció a l'enginyeria del programari OO" *Enginyeria del programari*. Fundació per a la Universitat Oberta de Catalunya. Barcelona. 2004.

11. Viquipèdia. *Connector*. En línia. Internet. 04/gener/2008. Accessible a http://ca.wikipedia.org/wiki/Connector (Consulta 13/gener/2008).

12. Viquipèdia. *Extensible Markup Language*. En línia. Internet. 09/gener/2008. Accessible a http://ca.wikipedia.org/wiki/Extensible_Markup_Language (Consulta 13/gener/2008).