UNIVERSITAT OBERTA DE CATALUNYA

Enginyeria Tècnica Informàtica de Sistemes

GENERACIO AUTOMATICA DE LES CONSTRAINTS A PARTIR DE UN DIAGRAMA DE CLASSES UML AMB RESTRICCIONS ESPECIFICADES EN OCL

Alumne/a: Sergi Aleixandre Díaz
Dirigit per: Jordi Cabot Sagrera
Co-dirigit per:

CURS 2003-04 (Setembre/Gener)

UNIVERSITAT OBERTA DE CATALUNYA

GENERACIO AUTOMATICA DE LES CONSTRAINTS A PARTIR DE UN DIAGRAMA DE CLASSES UML AMB RESTRICCIONS ESPECIFICADES EN OCL

L'UML és el llenguatge de modelització més emprat a la indústria per a l'especificació d'aplicacions informàtiques.

Aquest treball té com a punt de partida l'especificació en UML de la part estàtica d'un sistema d'informació.

El model estàtic especifica les classes amb els seus atributs i operacions permeses, les relacions i associacions entre aquestes i també les restriccions especificades en OCL (Object Constraint Language) que han de complir.

A parir del model estàtic, diagrama de classes, certes eines CASE poden generar només les taules de la base de dades, aquestes eines són propietàries, són dependents de cada eina, i en qualsevol cas el seu abast es restringeix al camp de les bases de dades relacionals.

L'àmbit del present treball és la generació automàtica de les restriccions d'integritat (claus primàries, alternatives i checks) tant per les bases de dades relacionals com per a les object-relational. Per a dur a terme aquest propòsit partirem de les especificacions de les restriccions en OCL especificades en fitxers XMI (XML Metadata Interchange, part del XML definida per al intercanvi de dades, al nostre cas suporta l'intercanvi de models UML) per a poder automatitzar mitjançant una aplicació programada en Java la seva implantació a les taules ja creades. Les restriccions que pretenem crear són les que es poden crear directament als gestor actuals de bases de dades, en principi claus primàries, claus alternatives i checks, d'altres restriccions com són el triggers , les assertions i les claus foranes són més fortament dependents del SGBD i tenen l'ajut d'eines pròpies de cada gestor o eines gràfiques per a les claus foranes, tot i així farem un apropament a l'Implementació d'aquestes via el mateix programa.

Aquest treball es pretén integrar amb d'altres que generaran l'estructura de taules tant per a bases de dades relacionals com per a les object-relational.

ÍNDEX

GLOSSARI

UML Unified Modeling Language, Llenguatge definit i desenvolupat per OMG (Open Management Group) per especificar i especificar software, especialment indicat per a models orientats a objectes.

XML Extensible Markup Language. Llenguatge de marques ampliable és una forma flexible de crear formats d'informació i compartir el format i les dades a la World Wide Web, intranets y altres xarxes OMG.

CASE Computer-Aided Software Engineering, eines amb ajut automàtic o semiautomàtic per al desenvolupament ràpid d'aplicacions, integren a la mateixa aplicació el màxim de passos del cicle de vida del software.

SAX Simple API for XML. Va començar essent un API de Java per tractar documents XML, actualment suporta altres llenguatges

DOM Document Object Model. API feta amb llenguatge neutre amb implementacions amb Java, C++... És més pràctica que SAX per a tractaments complexos del XML, també requereix més recursos.

XMI Part de XML per a intercanvi de dades és aplicable a una gran varietat d'objectes: anàlisi (UML), software (Java, C++),components (EJB, IDL, Corba Component Model), i bases de dades (CWM).

CUP Constructor of Useful Parsers. Eina en certa forma successora de yacc per a la definició de gramàtiques.

OCL Object Constraint Language. Llenguatge de modelització textual que amplia les possibilitats de UML, no és només una eina de definició de restriccions, permet la navegació, definició de registres, definició de pre-condicions i post-condicions, definició de tipus de dades...

PREFACI

L'objectiu del present treball és el de completar els automatismes amb els que les eines CASE actuals són capces de generar l'estructura de bases de dades de l'aplicació dissenyada.

Actualment les eines de disseny es descomponen en dues branques, les que són utilitzades únicament com a eines de disseny amb diagrames UML i les que poden afegir la generació de la conseqüent estructura de taules al gestor de bases de dades corresponent. Fins i tot aquestes últimes passen per alt la generació automàtica de les restriccions que es defineixen utilitzant el llenguatge OCL.

Aquest treball no es restringeix al camp de les bases de dades relacionals ni al de les object-realtionals ja que podem considerar que és una part comuna a ambdós tipus de gestors.

El punt de partida escollit és un nexe d'unió prou gran per que aquest treball pugui ser introduït i ampliat si s'escau a d'altres generadors d'estructures de bases de dades fets amb eines CASE. Aquest punt és XMI, segons aquest estàndard diferents eines CASE a parir del mateix diagrama UML haurien de generar el mateix fitxer XMI per a intercanvi de dades, en aquest cas contindria l'estructura UML i les restriccions OCL.
A partir d'aquest s'ha de poder generar l'estructura de bases de dades i les seves restriccions.

Aquest treball s'ha intentat fer de la manera més generalista possible amb la intenció que la reutilització de tot o alguna de les seves parts pugui ser factible, bàsicament seguint els models de lectura de fitxers XMI o parsers que ja s'han desenvolupat com són SAX i DOM.

Com anirem veient el desenvolupament d' una estructura amb voluntat generalista a un mon propietari com és el de les bases de dades te un límit actualment.
El camp al que estem està justament a la vora d'aquest límit, així per exemple és relativament fàcil abordar aquest projecte fins a la generació de claus primàries i alternatives, que conceptualment des de l'esquema extern poden ser idèntiques i també per a la generació de checks simples però tot i així cada gestor accepta sentències amb una mateixa base gramàtica però amb diferent format.

El següent pas és ara per ara molt més difícil de completar, no pas per a la seva dificultat d'implementació sinó pel fet de que en realitat estaríem partint una eina generalista per a adaptar-la a un gestor, com a exemple citarem la diferència en l'elaboració d'un trigger amb INFORMIX, que pot acceptar certes assercions, o amb ORACLE que te un procés de creació de triggers força diferent dintre del que hauríem d'incloure les assercions, ja que no les accepta directament.

D'altres gestors de dades poden necessitar fins i tot l'ajut de eines externes com programes que controlin les insercions, actualitzacions i esborrat de registres ja que no suporten aquest tipus de restriccions.

Per últim altre tipus de restriccions que no tractarem de la mateixa forma que els checks i les claus són les claus foranes, el motiu és que són fàcilment representades amb l'ajut d'eines gràfiques i normalment no calen representar-les amb OCL.

Tot i així tractarem les claus foranes amb una possible definició amb XMI que ha acceptat el nostre entorn de proves i deixarem el camp obert per poder afegir diferents tractaments específics a la nostra estructura de forma més senzilla possible.

Aquest darrer punt cal tenir-ho en compte, tot i que OCL sigui un llenguatge formal basat en text i sense ambigüitats el suport de les eines CASE acostuma a veure's limitat i per tant les possibles representacions correctes en OCL poden ser acceptades per una eina i no pas per un'altre.

L'estudi de casos del nostre projecte per tant es veu limitat per l'eina escollida per a fer les proves ja que aquest treball és un treball pràctic.

INTRODUCCIÓ

Aquest treball parteix dels fitxers XMI i d'altres treball que ens ajuden a recórrer aquests fitxers estructurats per a trobar l'informació de les restriccions especificades amb l'ajut d'eines avançades.
L'entorn escollit per al desenvolupament a estat la plataforma Java ja que disposava de projectes concurrents per la lectura d'aquests fitxers, parsers, i a la pròpia estructura de classes de Java.

Basant-nos en aquesta estructura hem de ser capaços de crear no només una aplicació que ens permeti assolir la creació de les restriccions àmbit d'aquest treball sinó que hem de poder facilitar una eina ampliable que pugui ser revisada i utilitzada total o parcialment per d'altres que tinguin com a objectiu el traspàs d'informació continguda de forma textual a fitxers XMI al gestor de bases de dades que ens vulguem connectar.

Primerament hem de localitzar dins el fitxer XMI la part d'informació que volem automatitzar per implementar-la al gestor escollit.

En segon terme hem d'analitzar sintàcticament la correcció d'aquesta informació i traslladar-la a SQL, en aquest punt trobarem que no tots els gestors admeten la mateixa sintaxi pel que hem de planificar un extracció de la informació necessària i passar-la a la classe que la particularitza per a cada gestor.

En tercer terme hem d'aconseguir que l'eina ens faciliti la distinció i elecció de el que volem passar al gestor de bases de dades al cas que aquesta no sigui prou clara a XMI, com a exemple citarem el possible enviament de claus primàries sobre una mateixa taula o la distinció entre aquestes i les claus alternatives ja que es poden definir de forma idèntica al fitxer XMI.

Finalment hem de fer una connexió al gestor per a enviar-li les ordres que siguin adients per la inclusió a l'esquema de les restriccions que hàgim introduït.

Aquest projecte en principi és el complement a d'altres que s'encarreguen de generar l'estructura de bases de dades de una aplicació ja sigui en un entorn de dades relacional o object relacional, també pot ser complementari a d'altres eines propietàries que només arriben a la generació d'estructures de dades.
Igualment pot ser ampliable amb connexions a gestors nous o amb l'inclusió de gramàtiques fetes amb qualssevol eina sempre que retornin l'informació requerida per l'elaboració de la sentencia particular de SQL adient.

1. UML punt de partida

Com a punt de parida d'aquest treball hem extret la informació que ens dona la OMG, més concretament la versió 1.5 de març de 2003. Aquest document conté al seu capítol 6 la especificació de l'Object Constraint Language que ha estat utilitzada per a reconèixer les restriccions definides a l'entorn d'eines CASE i que generen el fitxer XMI normalitzat per al seu treball.

Al mercat hi ha una gran quantitat d'eines CASE per utilitzar i validar UML entre elles destacarem (1):
Magic Draw UML
Poseidon for UML
Rational Rose
Describe
Raphsody
Together
Algunes d'aquestes són capaces de generar l'esquema de les bases de dades moltes d'elles poden generar codi a diferents llenguatges com Rational Rose , Raphsody o Poseidon, també existeixen eines de desenvolupament visuals propietàries per generar bases de dades a productes comercials com ORACLE o INFORMIX però només aquestes que són especialistes al seu respectiu entorn poden generar les restriccions per a les seves bases de dades.

Nosaltres hem partit dels documents XMI generats amb una d'aquestes eines Poseidon for UML i a partir d'aquest document comença el treball de la generació de restriccions. Les eines descrites anteriorment a (1) suporten i generen els documents XMI dels que partim, així doncs l'eina de partida és certament intranscendent.

Poseidon fa una verificació de la validesa de les restriccions introduïdes de forma textual i que després es tradueixen en codi XMI.

Un exemple de codi XMI d'una constraint que pot definir una clau primària escrita amb OCL a Poseidon el següent:

context Clase inv cClave : Clase . allInstances -> forAll (c1 , c2 | c2 <> c1 implies c1 . clave <> c2 . clave)

Aquesta restricció es tradueix al document XMI de la següent forma:

<UML:Constraint xmi.id = 'a17' name = 'cClave' isSpecification = 'false'>
<UML:Constraint.body>
<UML:BooleanExpression xmi.id = 'a30' language = 'OCL' body = 'context Clase inv cClave : Clase . allInstances -> forAll (c1 , c2 | c2 <> c1 implies c1 . clave <> c2 . clave) '/>
</UML:Constraint.body>
Aquest extracte és part del fitxer XMI generat per l'eina CASE utilitzada, només una petita part del document XMI és objecte d'estudi de la nostra aplicació, concretament la part que està dintre de l'element constraint body.
En aquest punt hem d'escollir el mètode per tractar el document XMI.

2 RECORREGUT DEL FITXER XMI
SAX vs DOM

A l'hora de treballar amb el fitxer XMI no te cap sentit plantejar-se el tractament del fitxer XMI d'una forma pròpia ja que existeixen línies de treball desenvolupades que tracten aquest tipus de fitxers, bàsicament hem valorat les següents opcions:
SAX , Simple API for XML.
DOM, Document Object Model
JAXP, Java API for XML Processing.

Farem una breu descripció d'algunes de les característiques que ens han fet escollir SAX per recórrer l'arbre XML, la descripció acurada d'aquestes APIs no és l'objectiu del present treball.
Bàsicament SAX és una API creada originàriament amb Java tot i que avui es troba implementada per diferents llenguatges, DOM és una API en llenguatge neutral que podem trobar implementada en diferents llenguatges com Java, C++ etc. JAXP és de fet una API desenvolupada amb Java que inclou entre d'altres facilitats als parsers dels anteriors SAX i DOM. Així doncs ens centrarem a les diferències que hi ha entre SAX i DOM.

SAX recorre el document XMI com a flux de dades, no es pot tornar enrera amb el seu parser i només revisa un cop el document i es basa en els events que es produeixen al mateix document com principi del document, fi del element...

DOM recorre el document i el guarda a memòria per a convertir-ho en un arbre DOM.
Permet recórrer i tornar a llegir l'arbre, construir un arbre nou, modificar l'arbre existent i copiar subarbres entre diferents arbres.

El nostre problema està ben definit i es troba fàcilment amb una sola passada al fitxer XMI, no ens cal realitzar un arbre com possiblement hauríem de fer si el problema que s'ens planteja és la realització de l'esquema de la base de dades, al nostre cas la complicació que afegeix DOM no ens reporta beneficis en una primera instància.
Els elements que hem de tractar estan perfectament definits dintre de les etiquetes de constraint body amb un principi i un final pel que podem prescindir de la resta d'elements del fitxer XMI tal i com s'ha plantejat el projecte.
És cert que la realització d'un arbre de constraints amb els diferents context, al nostre cas taules o relacions entre elles, ens podria ajudar a establir les possibles incongruències pròpies de l'OCL com la distinció sobre una taula entre la clau primària i les alternatives però nosaltres hem decidit tractar aquest problema per programa.
Com hem dit la simplicitat de SAX és més que suficient respecte a DOM per a la nostra aplicació i JAXP no és sinó una ampliació de les dues tècniques breument explicades.

3 Utilització de SAX versió 2

Dintre del possibles paquets existents per a recórrer el document XMI que hem trobat a www.saxproject.org ens hem decidit per la versió facilitada amb el fitxer saxon-aelfred.jar per al funcionament correcte del programa l'hem d'incloure a \lib\ext del path de la nostra versió de Java, al nostre cas a :
C:\j2sdk1.4.1_01\jre\lib\ext
Podem trobar més informació del funcionament d'aquest paquet i de les classes que el composen a la web de saxproject tal i com hem afegit a la següent taula:

Class Name	Notes
gnu.xml.aelfred2.SAXDriver	Lightweight non-validating parser; Free Software
gnu.xml.aelfred2.XmlReader	Optionally validates; Free Software
oracle.xml.parser.v2.SAXParser	Optionally validates; proprietary
org.apache.crimson.parser.XMLReaderImpl	Optionally validates; used in JDK 1.4; Open Source
org.apache.xerces.parsers.SAXParser	Optionally validates; Open Source

Requeriments:
Els requeriments mínims per a utilitzar aquest software són:
-Java 1.1 o major, nosaltres hem utilitzat la versió 1.4.1 que ve amb el j2sdj1.4.1_01.
-SAX2-compatible XML parser i la distribució SAX2 instal·lada al Classpath de Java, aquest punt es soluciona amb la instal·lació del .jar anteriorment descrita.

El funcionament detallat de les classes auxiliars que hem utilitzat per a recórrer el document XMI es pot trobar a la web de saxproject.org i no és objecte d'estudi d'aquest projecte, bàsicament utilitzarem la classe DefaultHandler per a crear un XMLReader de la classe XMLReaderFactory llegint els continguts amb els mètodes setContentHandler i setErrorHandler.

Farem una descripció mínima del funcionament d'aquestes classes juntament amb la descripció detallada de les decisions preses per a estructurar el nostre cos de programa al capítol que parla del main MySAXApp2.java, la descripció detallada de les classes auxiliars queda fora de l'abast d'aquest projecte ja que no han estat desenvolupades a aquest treball.

4 OCL, captura de les restriccions.

En primer lloc direm que la utilització de OCL te els següents avantatges:

1 OCL és un llenguatge formal que permet escriure restriccions sense ambigüitats que no es poden representar gràficament.
2 És un llenguatge de models independent de la plataforma, no és un llenguatge de programació ni pretén ser-ho.
3 La seva utilització és força més senzilla que la de llenguatges amb alt contingut matemàtic per a especificacions com Z.

També te les seves limitacions:

1 Es basa en una descripció textual de la restricció més que amb suport gràfic.
2 És necessari un coneixement previ de OCL i la seva interfície per a escriure correctament una restricció.
3 Hi ha eines CASE per UML que no suporten OCL.
4 No hi han gaires eines de suport.
5 Permet una descripció de restricció molt oberta ja que està més proper al llenguatge natural que al matemàtic, hi ha d'altres com Z-schema que són llenguatges fortament matemàtics.

Definitivament OCl és un llenguatge eminentment descriptiu que permet un grau de llibertat molt elevat.
Per exemple hi ha moltes maneres de definir que un atribut és únic, a l'entorn de bases de dades és una definició de clau primària o alternativa, així doncs ens centrarem com exemple a les vies possibles sense fer un estudi complert de les possibles definicions de clau primària:

i) Una possible definició que ja hem vist és la següent:

context Clase inv cClave :
Clase . allInstances -> forAll (c1,c2| c1 <>c2 implies c1 . clave <> c2 . clave)

1 Òbviament podem permutar el ordre de les condicions expressades als costats de implies .
2 La mateixa definició és vàlida canviant els <> per = .
3 També es pot expressar la definició per múltiples camps amb la combinació de camps de c1 i c2 igualats units amb ANDs o c1 i c2 desigualats units amb ORs.

Aquesta gramàtica és la que hem implementat a GramaticaPrimaryKey.java però tot i que és possiblement la forma més usual que es troba a la documentació de OCL existeixen moltes altres possibilitats de definició per a una clau primària i no totes suportades per totes les eines CASE.

ii) Un'altre forma d'escriure l'anterior restricció , si es tenen coneixements d'OCL, és:

Clase . allInstances -> forAll (c1| c1 <>self implies c1 . clave <> self . clave)

Aquesta forma amb les seves variacions anteriorment exposades també està incorporada a GramaticaPrimaryKey.java.

Utilitzant la navegació per OCL i les col·leccions es poden fer altres definicions de clau primària, en principi força rebuscades com per exemple utilitzar col·leccions i selects d'existència o no existència del registre de la taula que vulneri la condició de clau primària, aquest tipus de claus no estan suportats per la gramàtica implementada.

També es pot utilitzar els OCL predefinits com OclType o OclAny per definir claus primàries, o checks o claus foranes, aquest tipus de claus no estan implementats per dos motius, al igual que al paràgraf anterior les combinacions que es poden fer per definir restriccions són inabastables i d'altra banda moltes eines CASE ,entre elles la versió de POSEIDON (*) utilitzada per provar aquesta aplicació, no les suporten.
Exemple no suportat:

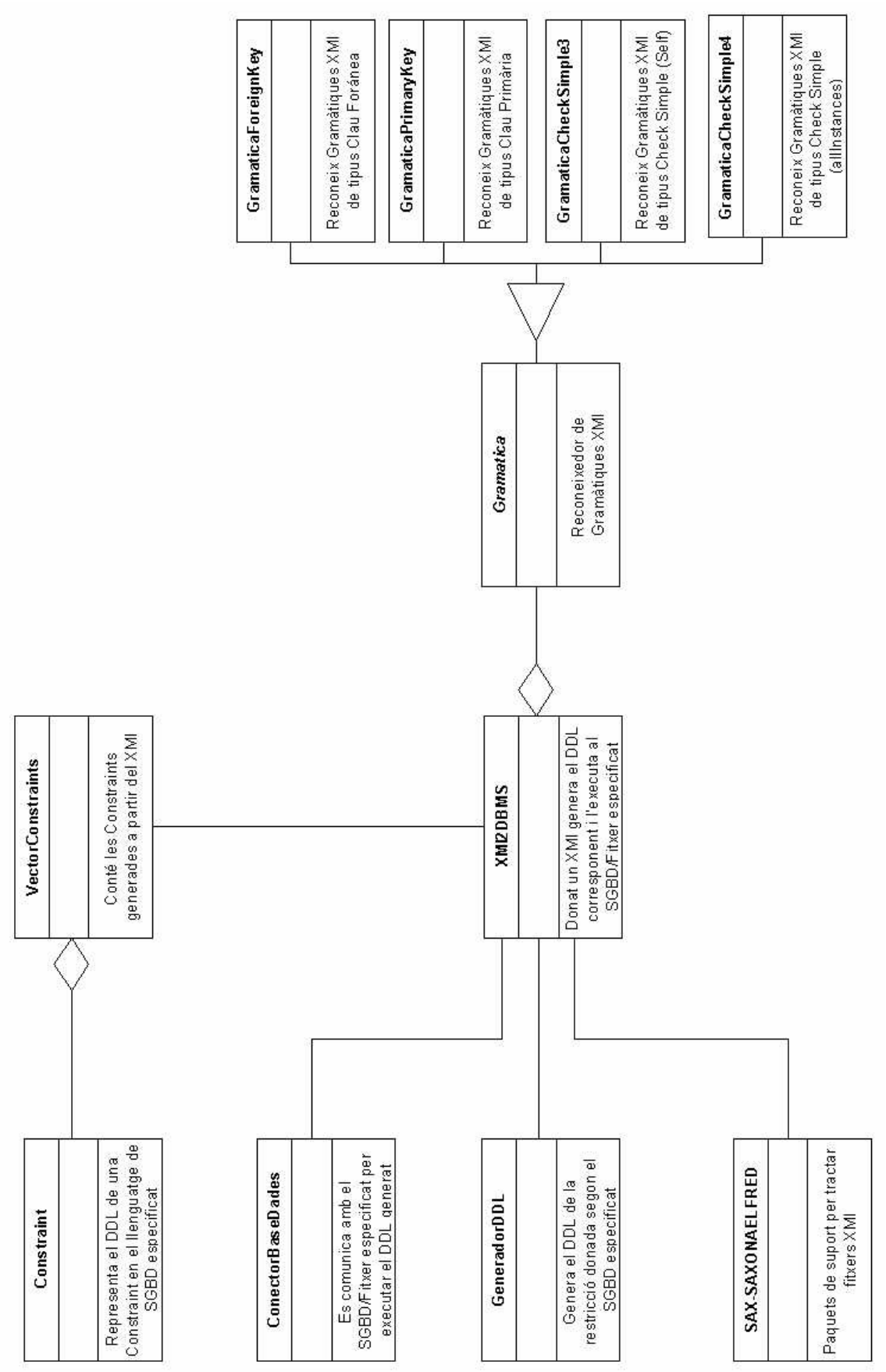
(1)
context Clase inv cClave :
oclType . allInstances -> forAll (c1\| c1 <>self implies c1 . clave <> self . clave)

En definitiva l'utilització com a entrada a la nostra aplicació d'un fitxer fet amb un llenguatge textual ens ha obligat a realitzar una eina que primés la seva extensibilitat i flexibilitat vers la seva fiabilitat per a reconèixer totes les possibles combinacions de definicions.

(*) Les eines CASE per a UML només garanteixen que l'OCL que se li ha escrit és sintàcticament correcte amb el model que se ha representat amb elles, els camps existeixen i les seves operacions són permeses o fins i tot ni això, però no tenen cura que hàgim repetit una mateixa restricció o que tinguem restriccions incongruents. Aquí partirem de la suposició que partim d'un document XMI correcte, altres projectes com XMI-DBMS es centren sobre tot en aquestes verificacions, sobre tot a la versió 1.

(1) Brian Lings, Information Systems development OCL
http://www.dcs.ex.ac.uk/~bjlings/Notes/ISD/ISD-L3-2003-12-01.pdf

5 Descripció de les Classes que composen el Projecte



Introducció a l'aplicació

La classe principal és XMI2DBMS, aquesta classe neix en principi com una aplicació definida a www.saxproject.org com a una aplicació que utilitza els parsers SAX.
Hem modificat a aquesta classe el comportament dels mètodes de tractament d'elements (startElement i endElement) per que tractin les restriccions que són perfectament conegudes per les etiquetes del fitxer XMI.

Per fer el tractament hem preparat un vector de gramàtiques ampliables amb classes hereves de la classe abstracta Gramatica. Les gramàtiques si reconeixen l'existència de una restricció coneguda ens retornaran els elements necessaris per poder construir la restricció.

Amb GeneradorDDL generem el codi de les comandes que es poden enviar al gestor de bases de dades i les guardem amb el format definit per la classe constraint.

Aquesta classe és només una estructura adient que conté el nom de la taula, el tipus de restricció, la restricció reconeguda en format adient per al SGBD i per últim la restricció alternativa si l'hem definit.

Aquestes restriccions les guardem a un vector de restriccions amb l'objectiu de ordenar-les i poder-les tractar, al nostre cas només detectem que volem enviar més d'una clau primària a una taula i traiem el missatge per línia de comandes per a escollir quina volem, les altres seran definides com a úniques.
Òbviament amb les restriccions ja ordenades podríem definir altres tractaments a XMI2DBMS.

Un cop finalitzat l'elecció de claus ConectorBaseDades s'encarrega de treure les restriccions a fitxer o és connecta al SGBD via ODBC i executa aquestes comandes.

Les gramàtiques definides estan fetes amb enfocaments força diferents.

La gramàtica que reconeix les claus primàries i alternatives està feta amb un mètode de definició més formal i el seu estudi ha estat fet de forma similar a la generació amb eines com lexx o yacc, com es pot veure aquesta gramàtica tot i tractar un nombre limitat de casos te un codi força més complicat.

Les gramàtiques dels checks, que hem anomenat simples tenen un tractament força diferent ja que és limiten a reconèixer que possiblement estem definint un check de forma seqüencial i quan entrem al cos de l'invariant fan ús del mètode restricted_word.

Aquest mètode bàsicament descarta que estiguem definint assercions, disparadors o definint accessos a dades amb condicions, ja que OCL no només defineix restriccions, d'altra banda també limita l'expressió dels checks que reconeixem ja que alguns tokens que detectem podrien formar part de la definició d'un check simple.

Per últim la definició de claus foranes s'ha fet d'una forma força dirigida a un tipus de definició trobada a la documentació, de fet ha estat més difícil que l'eina utilitzada

l'acceptés amb l'ajut de l'eina gràfica, està definida sobre una associació que la seva implantació.

5·1 **Gramatica**

Gramatica
+ faMatching (String pRegla) : boolean *+ ddlConstraint (String pRegla) : String [][]*
Reconeixedor de Gramàtiques XMI Classe Abstracta de la que heretaran totes les gramàtiques que s' incloguin en el projecte. Serveix per obligar a que totes les gramàtiques definides al projecte hagin de tenir el dos mètodes especificats.

Especificació de mètodes y Propietats de la Classe

faMatching (String pRegla) : boolean

Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

Aquest mètode ha de retornar cert o fals en funció de si la cadena del XMI que es passa com a paràmetre coincideix amb la gramàtica que suporta aquesta classe.

ddlConstraint (String pRegla) : String[][]

Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

Si la cadena del XMI que es passa com a paràmetre faMatching amb la gramàtica que suporta aquesta classe retornarà una matriu de cadenes que permetran generar el DDL corresponent a aquesta restricció, si no retornarà un nul.

Retorn [0][0] = Tipus Restricció ("PRIMARY KEY", "CHECK" ...)
Retorn [1][0] = Nom de la Taula
Retorn [2][0] = Nom de la Constraint
Retorno [3][n] = Camps i Condicions de la Constraint segons el tipus de Constraint que sigui.

Totes les gramàtiques que s' utilitzin en el projecte es poden desenvolupar amb qualsevol eina de reconeixement i generació de gramàtiques (lexx/yack, cup ...).
L' important d'aquesta classe es que retorna la informació necessària per generar posteriorment el DDL (Data Definition Language), com que el DDL es depenent del SGBD especificat no es operatiu incloure' l en aquesta classe, si no que extraiem de l' XMI la informació necessària per generar la constraint independentment del SGBD que es vagi a utilitzar (palla sintàctica).

5·2 **GramaticaPrimaryKey**

GramaticaPrimaryKey extends **Gramatica**
+ faMatching (String pRegla) : boolean + ddlConstraint (String pRegla) : String [][] + comprobar_primer_nivel(String): boolean + comprobar_prefijo(): boolean + comprobar_instancias(): boolean + comprobar_condiciones(String[]): integer - comprobar_condicion(String[]): integer
Reconeixedor de Primary Keys per a la forma més usual d'escriure aquestes a XMI: *context Taula inv cClave : Taula . allInstances -> forAll (c1 , c2 | c1 <> c2 implies c1 . clave <> c2 . clave)* Accepta múltiples camps a la clau i variacions d'ordre de les condicions, conté les famílies de definicions igualtat amb ANDs i desigualtat amb ORs. Accepta les mateixes variacions per a les formes: *context Taula inv cClave : Taula . allInstances -> forAll (c1 | c1 <> self implies c1 . clave <> self . clave)*

Aquesta classe ha estat implementada a partir d'un estudi de la gramàtica de l'expressió a analitzar, en concret m'he basat al mètode descrit per a l'utilització de CUP (Constructor of Useful Parsers, eina per a la definició de gramàtiques(2)) com a especificador de sintaxi.

Gramàtica de Primary Key

GramaticaPK ::= Prefijo '(' Instancias '|' condiciones 'implies' condiciones ')'

Prefijo ::= 'context' Tabla 'inv' Constraint ':' Tabla '.' 'allInstances' '->' 'forAll'

Instancias ::= Instancia ', ' Instancia | Instancia

condiciones ::= Instancia '=' Instancia | (1) (1->3)
Instancia '<>' Instancia | (2) (2->4)
lista_condiciones_igual | (3) (3->1)
lista_condiciones_distinto (4) (4->2)

lista_condiciones_igual ::= condicion_igual | condicion_igual 'and' lista_condiciones_igual

lista_condiciones_distinto ::= condicion_distinto | condicion_distinto 'and' lista_condiciones_distinto

condicion_igual ::= Instancia '.' Campo '=' Instancia '.' Campo

condicion_distinto ::= Instancia '.' Campo '<>' Instancia '.' Campo

(2) http://bmrc.berkeley.edu/courseware/cs164/spring98/proj/cup/manual.html

Tabla ::= cadena
Constraint ::= cadena
Instancia ::= cadena | ‘self’

Notes: Instancia de la part dreta ha de ser distinta a la Instancia de la part esquerra

He fet un esbós de la gramàtica, hi ha més condicions implementades a la classe GramaticaPrimaryKey.java.

Especificació de mètodes i Propietats de la Classe

comprobar_primer_nivel (String pRegla) : boolean

Comprova l’existència de tots les “tokens” necessaris per que puguem continuar avaluant l’expressió OCL com a vàlida i separa per referència les instàncies i condicions d’aquests tokens per després avaluar-les.

comprobar_prefijo () : boolean

Comprova que el prefix sigui de la classe:
context Taula inv cClave : Taula . allInstances -> forAll
Utilitza els paràmetres d’entrada per referència i retorna cert si el prefix s’ajusta a norma.

comprobar_instancias () : boolean

Utilitza els paràmetres d’entrada per referència, si tenim dues instàncies guarda els seus noms per a poder avaluar-les a les condicions, si només tenim una instància força que la segona instància de les condicions sigui self per establir igualtats o desigualtats, retorna cert per ambdós casos i altrament fals.

comprobar_condiciones (String[]) : integer

Rep com a paràmetre la llista de condicions a avaluar si rep una llista amb longitud 3 estarem a una condició del tipus:
c1=c2 retorna 1
c1<>c2 retorna 2
si la llista de condicions és superior utilitza el mètode **comprobar_condicion** per guardar els possibles camps de la clau primària i avaluar la condició retornant un enter.
Si les condicions que trobem totes tenen iguals separats amb ANDs retornarà 3
Si les condicions que trobem totes tenen desigualtats separats amb ORs retornarà 4

comprobar_condicion (String[]) : integer

Rep com a paràmetre les condicions simples que li passa **comprobar_condiciones**, cada condició simple te una longitud de 7 i aquest mètode emmagatzema els possibles camps de la clau primària i retorna:
1 per a les igualtats
2 per a les desigualtats.

faMatching (String pRegla) : boolean

Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

Utilitza els mètodes **comprobar_primer_nivel**, **comprobar_prefijo**, **comprobar_instancias**, **comprobar_condiciones** i **comprobar_condicion** per verificar la concordança de la restricció amb el model, les condicions són avaluades per la parella condició de la dreta de implies amb la condició de l'esquerra, si trobem una definició vàlida com:
1-3 / 3-1 que vol dir igualtat d'instàncies amb igualats de camps separats per ANDs / igual però amb ordre invertit.
2-4 / 4-2 que vol dir desigualtat d'instàncies amb desigualtats de camps separats per ORs / igual però amb ordre invertit.
retorna cert, altrament retorna fals.

Aquest mètode ha de retornar cert o fals en funció de si la cadena del XMI que es passa com a paràmetre coincideix amb la gramàtica que suporta aquesta classe.

ddlConstraint (String pRegla) : String[][]

Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

Si la cadena del XMI que es passa com a paràmetre faMatching amb la gramàtica que suporta aquesta classe retornarà una matriu de cadenes que permetran generar el DDL corresponent a aquesta restricció, si no retornarà un nul.

Retorn [0][0] = Tipus Restricció ("PRIMARY KEY")
Retorn [1][0] = Nom de la Taula
Retorn [2][0] = Nom de la Constraint
Retorno [3][n] = Camps de la Constraint

5·3 **GramaticaCheckSimple3**

GramaticaCheckSimple3 extends **Gramatica**
+ faMatching (String pRegla) : boolean + ddlConstraint (String pRegla) : String [][] - restricted_word(String): boolean
Reconeixedor de Checks simples per a la forma poc usual d'escriure aquestes a XMI: *context Taula inv cClave : Taula . allInstances -> forAll (c1 | c1 . clave <50 and c2.sou < 50000 or ....)* Accepta múltiples camps i condicions i restringeix que no sigui una asserció o un trigger amb el mètode restricted_word, accepta el que es pot fer amb SQL sota el punt de vista de Check.

Especificació de mètodes y Propietats de la Classe

faMatching (String pRegla) : boolean

Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

S'ajuda de restricted_word per discernir els checks simples de les assertions i triggers.

Aquest mètode ha de retornar cert o fals en funció de si la cadena del XMI que es passa com a paràmetre coincideix amb la gramàtica que suporta aquesta classe.

restricted_word (String pCadena) : boolean

Comprova que no s'utilitzen eines de OCL per fer seleccions, definir altres invariants, accedir o definir col·leccions, utilitzar regles if-then-else per a la definició de restriccions que normalment ens portarien a la definició de triggers i/o assertions.

ddlConstraint (String pRegla) : String[][]

Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

Si la cadena del XMI que es passa com a paràmetre faMatching amb la gramàtica que suporta aquesta classe retornarà una matriu de cadenes que permetran generar el DDL corresponent a aquesta restricció, si no retornarà un nul.

Retorn [0][0] = Tipus Restricció ("CHECK SIMPLE")
Retorn [1][0] = Nom de la Taula
Retorn [2][0] = Nom de la Constraint
Retorno [3][n] = Condicions de la restricció

5·4 **GramaticaCheckSimple4**

GramaticaCheckSimple4
+ faMatching (String pRegla) : boolean + ddlConstraint (String pRegla) : String [][] - restricted_word(String): boolean
Reconeixedor de Checks simples per a la forma més usual d'escriure aquestes a XMI: *context Taula inv cClave : self. clave <50 and self.sou < 50000 or....)* Accepta múltiples camps i condicions i restringeix que no sigui una asserció o un trigger amb el mètode restricted_word, accepta el que es pot fer amb SQL sota el punt de vista de Check.

Especificació de mètodes y Propietats de la Classe

faMatching (String pRegla) : boolean

Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

S'ajuda de restricted_word per discernir els checks simples de les assertions i triggers.

Aquest mètode ha de retornar cert o fals en funció de si la cadena del XMI que es passa com a paràmetre coincideix amb la gramàtica que suporta aquesta classe.

restricted_word (String pCadena) : boolean

Comprova que no s'utilitzen eines de OCL per fer seleccions, definir altres invariants, accedir o definir col·leccions, utilitzar regles if-then-else per a la definició de restriccions que normalment ens portarien a la definició de triggers i/o assertions.

ddlConstraint (String pRegla) : String[][]

Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

Si la cadena del XMI que es passa com a paràmetre faMatching amb la gramàtica que suporta aquesta classe retornarà una matriu de cadenes que permetran generar el DDL corresponent a aquesta restricció, si no retornarà un nul.

Retorn [0][0] = Tipus Restricció ("CHECK SIMPLE")
Retorn [1][0] = Nom de la Taula
Retorn [2][0] = Nom de la Constraint
Retorno [3][n] = Condicions de la restricció

5·5 **GramaticaForeignKey**

GramaticaForeignKey
+ faMatching (String pRegla) : boolean + ddlConstraint (String pRegla) : String [][]
Reconeixedor de claus foranes per a una forma usual d'escriure aquestes a XMI, tot i així necessita l'ajut d'eines gràfiques de l'eina CASE per validar: context Taula1 inv FK: *Taula1.allInstances -> forAll (i | Taula2.allInstances -> collect (j | i.keyTaula1 =j.keyTaula2) -> size > 0)*(3) keyTaula1 és una clau forana de Taula1 que referència keyTaula2 a Taula2 Accepta múltiples camps i canvis d'ordre a les igualtats units amb ANDs

Aquesta classe ha estat afegida al final del projecte ja que te fortes competidores als SGBD amb ajut d'eines gràfiques, de fet moltes eines gràfiques CASE no la suporten i d'altres com POSEIDON només la validen si es defineix amb l'ajut de l'eina gràfica implementant al model relacions d'associativitat.
Per aquest motiu aquesta classe es surt de l'esquema convencional de la resta de classes i només ha estat afegida per veure que el tractament és possible sempre que eina CASE i SGBD l'acceptin.

Especificació de mètodes y Propietats de la Classe

faMatching (String pRegla) : boolean
Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar

En aquest cas el reconeixement de la clau per codi es força monolític, tot i que permet certs graus de llibertat a la formulació amb múltiples camps i canvis d'ordres de les igualtats, però és una forma vàlida i un exemple de com es poden incorporar noves gramàtiques al projecte adaptant els mètodes al format requerit.

Aquest mètode ha de retornar cert o fals en funció de si la cadena del XMI que es passa com a paràmetre coincideix amb la gramàtica que suporta aquesta classe.

ddlConstraint (String pRegla) : String[][]
Paràmetre de Entrada pRegla: Cadena XMI del Constraint Body que defineix la restricció d' integritat a analitzar
Si la cadena del XMI que es passa com a paràmetre faMatching amb la gramàtica que suporta aquesta classe retornarà una matriu de cadenes que permetran generar el DDL corresponent a aquesta restricció, si no retornarà un nul.

Retorn [0][0] = Tipus Restricció ("FOREIGN KEY")
Retorn [1][0] = Nom de la Taula 1
Retorn [2][0] = Nom de la Constraint
Retorno [3][0] = Nom de la Taula 2
Retorno [3][1] = Camps de la taula1
Retorno [3][2] = Camps de la taula2

(3) Journal of Object Thecnology Vol.2,No 5, September-October 2003

5·6 **GeneradorDDL**

GeneradorDDL
+ GeneradorDDL (SGBD String); + generarDDL(String [][]) : String [] - generarDDLPK (String [][]) : String [] - generarDDLCheck (String [][]) : String [] - generarDDLForeignKey (String [][]) : String []
A partir de la informació extreta per la Classe que analitza la Gramàtica corresponent genera el DDL en funció del SGBD especificat. **La funció principal de aquesta classe es independitzar el XMI del SGBD que es vagi a utilitzar.**

Especificació de mètodes y Propietats de la Classe

String SGBD: Conté el SGBD per al qual s' han de generar les sentencies DDLs corresponents.

generadorDDL (SGBD String):String

Paràmetre de Entrada SGBD: cadena que representa el SGBD ("ORACLE", "INFORMIX" ...) per al qual es generaran les sentencies DDL. Si el SGBD que es passa no està contemplat en aquesta classe es llença una excepció informant que aquest SGBD encara no està suportat altrament valida el SGBD.

generarDDL (SGBD String[][]):String[]

Rep els paràmetres d'entrada del mètode **ddlConstraint** de la gramàtica corresponent i analitza el element [0][0] per enviar als mètodes **generarDDLPK**, **generarDDLCheck** o **generarDDLForeignKey** segons correspongui.

Retorna els paràmetres que retornen aquestes funcions.

generarDDLPK (SGBD String[][]):: String[]

Rep com a paràmetre d'entrada:
Primitivas [0][0] = Tipus Restricció ("PRIMARY KEY")
Primitivas [1][0] = Nom de la Taula
Primitivas [2][0] = Nom de la Constraint
Primitivas [3][n] = Camps de la Constraint

Te per referència el SGBD utilitzat i ens retorna:

Retorn [0] = Nom de la Taula
Retorn [1] = Tipus Restricció ("PRIMARY KEY")
Retorn [2] = Nom de la Constraint
Retorno [3] = Sentència de clau primària vàlida per al SGBD escollit
Retorno [4] = Sentència de clau alternativa vàlida per al SGBD escollit
generarDDLCheck (SGBD String[][])

Rep com a paràmetre d'entrada:
Primitivas [0][0] = Tipus Restricció ("CHECK SIMPLE")
Primitivas [1][0] = Nom de la Taula
Primitivas [2][0] = Nom de la Constraint
Primitivas [3][n] = Camps de la Constraint

Te per referència el SGBD utilitzat i ens retorna:

Retorn [0] = Nom de la Taula
Retorn [1] = Tipus Restricció ("CHECK SIMPLE")
Retorn [2] = Nom de la Constraint
Retorno [3] = Sentència de addició de restricció vàlida per al SGBD escollit
Retorno [3] =”” cadena buida, no utilitzat.

generarDDLForeignKey (SGBD String[][]):: String[]

Rep com a paràmetre d'entrada:
Primitivas [0][0] = Tipus Restricció ("FOREIGN KEY")
Primitivas [1][0] = Nom de la Taula 1
Primitivas [2][0] = Nom de la Constraint
Primitivas [3][0] = Nom de la Taula 2
Primitivas [3][1] = Camps de la Taula 1
Primitivas [3][2] = Camps de la Taula 2

Te per referència el SGBD utilitzat i ens retorna:

Retorn [0] = Nom de la Taula
Retorn [1] = Tipus Restricció ("CHECK SIMPLE")
Retorn [2] = Nom de la Constraint
Retorno [3] = Sentència de addició de clau forana vàlida per al SGBD escollit
Retorno [3] =”” cadena buida, no utilitzat.

5·7 **Constraint**

ConectorBaseDades
+ Constraint(String []);
Classe que recolza a la classe principal XMI2DBMS, munta una estructura amb la constraint amb la que es pot emmagatzemar abans d'enviar al SGBD

Constraint(String []):String [];

Mètode constructor que rep com a paràmetre d'entrada la sortida de **generarDDL** i crea una estructura que podem guardar al vector proporcionat per **VectorConstraints**.

5·8 **VectorConstraints**

VectorConstraints
+ Ordenar();
Classe que recolza a la classe principal XMI2DBMS, extè la classe vector i guarda les constraints per analitzar si s'escau abans d'enviar al SGBD

Ordenar();

Mètode utilitzat per la classe principal per ordenar el vector de constraints, el que fem és ordenar les restriccions en primer lloc per nom de taula afectada i en segon lloc per tipus de restricció.

En aquesta versió l'únic que farem des de la classe principal és verificar que només enviem una clau primària per taula, o cap si ho escollim des de línia de comandes però com es pot suposar es podria utilitzar per fer d'altres verificacions.

5·9 **ConectorBaseDades**

ConectorBaseDades
+ ConectorBaseDades(String Tipus, String ODBC, String user, String password); + executarComanda (String comanda); + finalitzarConexio();

ConectorBaseDades(String Tipus, String ODBC, String user, String password);
Mètode que rep els paràmetres entrats quan s'executa la classe principal:

- tipus, si és F realitzarà una sortida a fitxer de les comandes que se li passin, si rep O realitza una connexió ODBC amb el SGBD escollit.
- ODBC rep el gestor (ORACLE, INFORMIX, ACCESS...) amb el que ha de treballar.
- user, rep l'usuari que fa la connexió amb el SGBD via ODBC si hem escollit O al tipus, altrament accepta qualsevol cadena.
- Password, rep la password que hem d'utilitzar amb el SGBD amb el que volem connectar si hem escollit a tipus O, altrament accepta qualsevol cadena.

Retorna una excepció si no ha pogut realitzar alguna de les tasques demanades.

executarComanda (String comanda);

Rep com a paràmetre d'entrada la comanda a executar, si el tipus escollit és F la escriu al fitxer de sortida, si és O la envia al SGBD via ODBC.
Retorna una excepció si no ha pogut realitzar aquesta tasca.

finalitzarConexio();

Finalitza la connexió amb el SGBD amb el que s'ha connectat si havíem rebut com a tipus O, si havíem rebut F com a tipus tanca el flux de dades al fitxer.

Retorna la excepció corresponent si no ha pogut realitzar la tasca demanada.

5·10 **XMI2DBMS**

XMI2DBMS
+ main(String[] args); + startDocument(); + endDocument(); + startElement(String uri, String name, String qName, Attributes atts); + endElement(String uri, String name, String qName);
Classe principal de l'aplicació que utilitza les classes anteriorment descrites més el parser tipus SAX escollit per analitzar el fitxer XMI i enviar les comandes Al SGBD si s'escau o guardar-les a fitxer.

main(String[] args):

Mètode principal de la aplicació, rep per línia de comandes els arguments de treball, en cas que no siguin els correctes llença un missatge amb l'especificació d'entrada:

"XMI2DBMS FitxerXMI GestorBaseDades O/F ODBC/FitxerSortida user password"

Aquest mètode prepara un vector de gramàtiques on guardarà les gramàtiques que han estat implementades i un vector de comandes que guardarà el vector de constraints que tenim a **VectorConstraints**.

Fa ús del mètodes **startDocument**, **endDocument**, **startElement** i **endElement** heretats de la classe del parser DefaultHandler. El mètodes **startDocument** i **endDocument** no han estat modificats respecte al parser SAX pel que no els comentarem.

Després de preparar els parsers del document el parser crida a **startElement** que ens omple el vector de comandes amb les restriccions reconegudes , un cop ordenades amb el mètode **Ordenar** de la classe VectorConstraints analitza si tenim més d'una clau primària per a la mateixa taula, si és així ens demana que escollim quina volem com a clau primària i un cop escollida envia l'execució d'aquesta com a clau primària i la resta com claus alternatives.
En cas que no hi hagin definides varies claus primàries per taula envia les comandes directament al mètode **executaComanda** de la classe ConectorBaseDades.

startElement(String uri, String name, String qName, Attributes atts):

Mètode modificat a partir del parser utilitzat de SAX, bàsicament comprova que el element que estem llegint sigui una constraint i en cas afirmatiu comença a desxifrar, altrament continua seqüencialment la lectura del document.

Un cop trobada una constraint comprova si el mètode **faMatching** de alguna de les gramàtiques del vector de gramàtiques retorna cert.
El mètode **ddlConstraint** de la gramàtica que ha reconegut la restricció passarà els arguments per que **generarDDL** de la classe generadorDDL generi el codi executable al SGBD escollit i si s'escau la sentència alternativa.

El mètode finalitza la seva execució guardant els arguments i sentències de la restricció al vector vComandes amb el que treballa el mètode **main**.

endElement(String uri, String name, String qName):

Mètode heretat del parser utilitzat de SAX amb el que verifiquem que hem trobat el final de la restricció.

5.11 Comentaris de les classes

A les explicacions de les classe que hem vist no hem inclòs els atributs ni les seves explicacions, el motiu és que l'explicació sencera del codi podria fer que la memòria fos massa complida de seguir, aquesta explicació es pot trobar al codi font i als fitxers generats amb javadoc en format htm.

El diferent comportament dels gestors de bases de dades comercials per reconèixer les restriccions ens ha obligat a prendre decisions de disseny importants ja que no podem generar un codi estàndard que funcioni a diversos SGBD, tot i que les variacions són mínimes.
Les sentències dels checks gairebé no difereixen si es fan conjuntament amb la creació de les taules, però aquest no és l'escenari del nostre projecte. Nosaltres hem de fer sentències de addició de restriccions o de modificació de taula.

El reconeixement de les claus i restriccions és possible sempre i quan aquestes siguin simples, per exemple no tindria gaire sentit definir una clau forana i a continuació i sense separar amb nou "*inv:*" afegir checks o definicions d'altres claus, no obstant és podria fer amb POSEIDON, aquestes formulacions no serien reconegudes per les actuals gramàtiques.

Un'altre possibilitat és definir les claus que pertoquin a una taula i a continuació les restriccions de la següent forma:

Context Taula inv *pk1:*
Taula.allInstances -> forAll(p1,p2| p1<>p2 implies p1.Pk1<>p2.Pk1 OR p1.Pk2<>p2.Pk2)
inv *chCK1: self.Pk1<10000 and self.cK1<> '99999'*
inv *unik: Taula.allInstances -> forAll(p1| p1<>self implies p1.u1<>self.u1)*

Aquest tipus de llistes de restriccions sí que serien reconegudes com a tal per les nostres gramàtiques ja que el XMI resultant les reconeix i representa com a restriccions independents.

La ampliació de SGBDs suportats i/o les seves funcionalitats es fa modificant la classe Generador DDL tot i que podríem haver optat per un esquema similar al de gramàtica amb gramàtiques filles el fet que possiblement el número de sentències amb diferents formes sigui reduït i conseqüentment els canvis de codi mínims ens ha fet decantar per a aquesta opció.

Bàsicament la dificultat la dificultat d'ampliació d'aquesta aplicació rau en que s'acompleixin els objectius de l'OCL, que sigui una especificació clara i estructurada de definicions que no es poden representar amb UML estàndard és a dir la nostra aplicació tal i com està concebuda fracassaria en els casos en que un mal disseny especifiqués al mateix invariant diversos conceptes diferents com les definicions de checks i claus.

Una decisió de disseny que s'ha pres és obligar a que les restriccions tinguin nom abans d'enviar-les al SGBD, aquesta decisió es pren degut al diferent comportament alhora d'acceptar o no sentències ALTER TABLE dels SGBD, pot ser discutible però el criteri que s'ha adoptat és el següent:

- Les restriccions i claus foranes si no tenen nom de restricció o de clau són anomenades al moment de reconèixer-la per part de la gramàtica que ha reconegut la restricció.
- Les claus primàries i alternatives s'han de rebre amb nom de restricció al XMI, altrament no seran reconegudes.
- Aquesta decisió es podria modificar al codi sense gaire esforç.

6 Conclusions

La utilització d'eines CASE per a fer el desenvolupament UML d'una aplicació informàtica efectivament és força estesa a la indústria, no obstant la utilització de OCL com a suport del diagrama UML es troba amb alguns inconvenients.

D'una banda està força estès l'ús de les eines CASE únicament com a eines de disseny (UPER CASE), aquest fet no ajuda gaire a que hi hagi una certa cultura de seguir les recomanacions que es puguin fer per a utilitzar OCL més que com un llenguatge natural més formal que una mera explicació textual.

Les eines disponibles al mercat, per exemple ERWin, Visible Analyst i d'altres, possibiliten una creació integrada de disseny i implantació però són eines propietàries que no faciliten el seu codi, de fet en ambdós casos l'implementació de les restriccions es fa amb l'ajut d'eines gràfiques.

Hi ha d'altres projectes que treballen en direccions diferents a aquest com pot ser l'utilització de llenguatges matemàtics com Z-Schema per a la definició de restriccions i el seu posterior pas a codi, també hi ha projectes no propietaris com XML-DBMS que tracten part de la línia d'aquest projecte, en qualsevol cas l'implantació de l'ús d'OCL no és la mateixa que la dels models UML.

D'altra banda hem utilitzat com a eina de generació de fitxers XMI Poseidon for UML, de l'experiència amb aquesta eina i les anàlisi que es poden trobar a www podem dir que el suport OCL és desigual pel que possiblement ens podríem trobar amb diverses formes d'especificació de restriccions per a diferents eines per exemple amb definicions pròpies a l'esquema d'elements no suportats per OCL estàndard.

La cerca de formes de definició de restriccions no ha estat gens fàcil i sovint he trobat definicions no suportades per l'eina CASE utilitzada, pel que l'univers de proves degut a la durada d'aquest projecte òbviament s'ha vist reduït.

Donat tot l'anteriorment dit la possible utilització d'aquesta eina podríem dir que és només un punt de partida.
OCL no s'ha creat com un possible llenguatge de programació i per tant tampoc presenta possibilitats de compilació to i que hi ha projectes que treballen en aquesta línia, possiblement un del més avantatjats es fa des de la universitat de Dresde.

Hem de notar que el reconeixement de totes les possibles formulacions amb OCL de restriccions ens ha quedat fora de l'àmbit d'aquest projecte, hem preferit fer una estructura que fos fàcilment ampliable i amb un codi senzill per afegir gramàtiques i gestors de dades .
D'altra banda el reconeixement que fa dels casos proposats sempre que aquest estiguin mínimament ben definits i la seva ampliació amb d'altres casos no proposats o amb altres gestors de bases de dades com a mínim em fan creure que la línia de treball ha estat ben dirigida.

Són d'especial interès els annexos XII i XII per l'ús de l'aplicació.

7 Bibliografía i referències

Els següents documents han estat utilitzats com a font de consulta per l'elaboració del present projecte. la data de URL no és la data de la primera consulta sinó l'última data que he comprovat la validessa de l'enllaç.

SAX & DOM, CPS 196.3, Introduction to Database Systems(presentació) [www.cs.duke.edu/courses/fall02/cps196.3/lectures/17-xml-notes.pdf] [URL 16/01/2004]

OMG Unified Modeling Language Specification, March 2003 Version 1.5 formal/03-03-01

Journal of Object Technology, Vol2 No5, September-October 2003 [www.jot.fm.] [URL 05/11/2003]

Information Systems Development OCL (presentació), Brian Lings 01/12/2003 [www.dcs.ex.ac.uk/~bjlings/Notes/ISD/ISD-L3-2003-12-01.pdf] [URL 16/01/2004]

Operation Schemas and OCL,Version EIGHTc, October 2003,Alfred Strohmeier and Shane Sendall, [/lglwww.epfl.ch/teaching/software_engineering/documentation/analysis/syntax-for-operation-schema-EIGHTc-for-students.pdf][URL 16/01/2004]

Operation Schemas and the OCL, Alfred Strohmeier and Shane Sendall, Nov 2000, version FIVE [lglwww.epfl.ch/teaching/software_engineering00_01/documentation/ analysis/syntax-for-operation-schema-FIVE.pdf] [URL 16/01/2004]

Derived Classes as a basis for Views in UML/OCL Data Models, H. Balsters, [www.ub.rug.nl/eldoc/som/a/02A47/02A47.pdf] [URL 16/01/2004]

Experiments With XMI Based Transformations of Software Models, Demuth-Hussman-Obermaier, [ase.arc.nasa.gov/wtuml01/submissions/ demuth-hussman-obermaier.pdf] [URL 16/01/2004]

ANNEX I

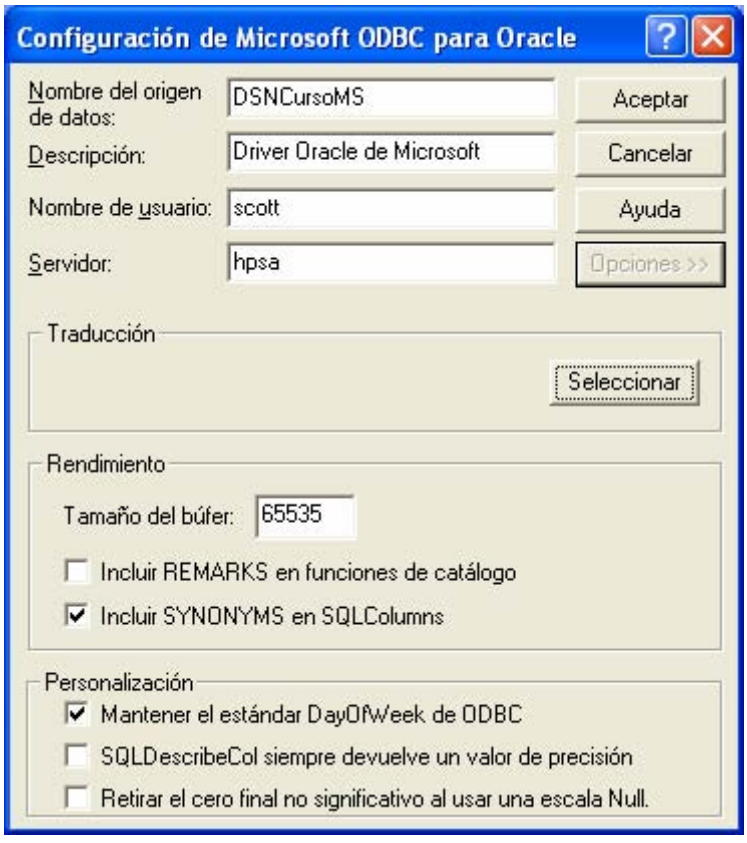
DESCRIPCIÓ DE L' ENTORN DE PROBES

Per realitzar la connexió a la Bases de Dades on-line s'ha optat per fer-ho amb la tècnica d'OBDC per que emmascara tota la complexitat de la connexió a la Base de Dades al usuari final, que tan sols ha de indicar la font de dades OBDC que vol utilitzar al cridar a l'aplicació Java. Si haguéssim optat per una connexió a Base de dades DSNLess sense haver de definir prèviament la font de dades ODBC al sistema, la crida a l'aplicació fora amb molts mes paràmetres o amb un fitxer de paràmetres que complicaria la dinàmica de les proves.

Per realitzar les proves de l'aplicació Java s'han creat 4 ODBCs al sistema:

ODBC Oracle (Driver Microsoft)

S'ha creat una font de dades ODBC amb el driver de Microsoft tal i com s'indica a la figura:



El servidor hpsa és un Oracle 9i instal·lat a la mateixa maquina que te com a sistema Operatiu un Microsoft Windows XP Home Edition Versió 2002.

La definició d'accés a haches servidor es troba al fitxer tnsnames.ora

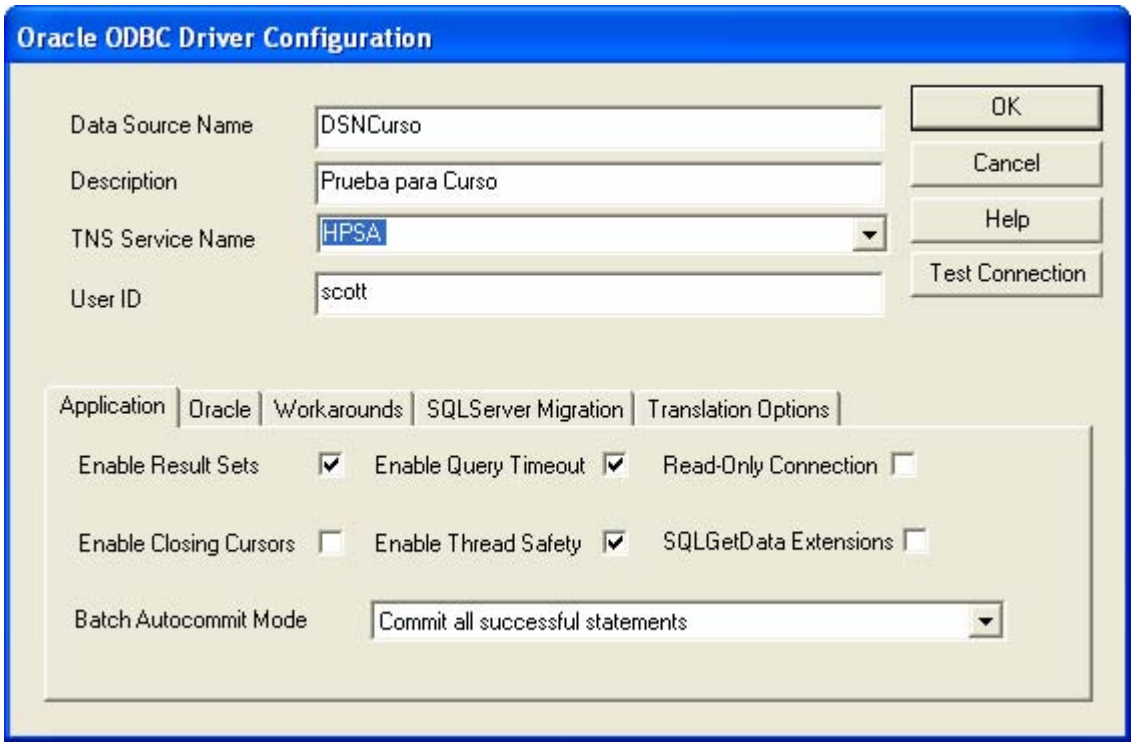
HPSA =

```
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
  )
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = HPSA)
  )
)
```

La font de dades ODBC s'ha creat com una font de dades "DNS de Sistema" per que no hi hagi problemes alhora d' accedir a la citada font de dades amb el permisos de l'usuari.

ODBC Oracle (Driver Oracle)

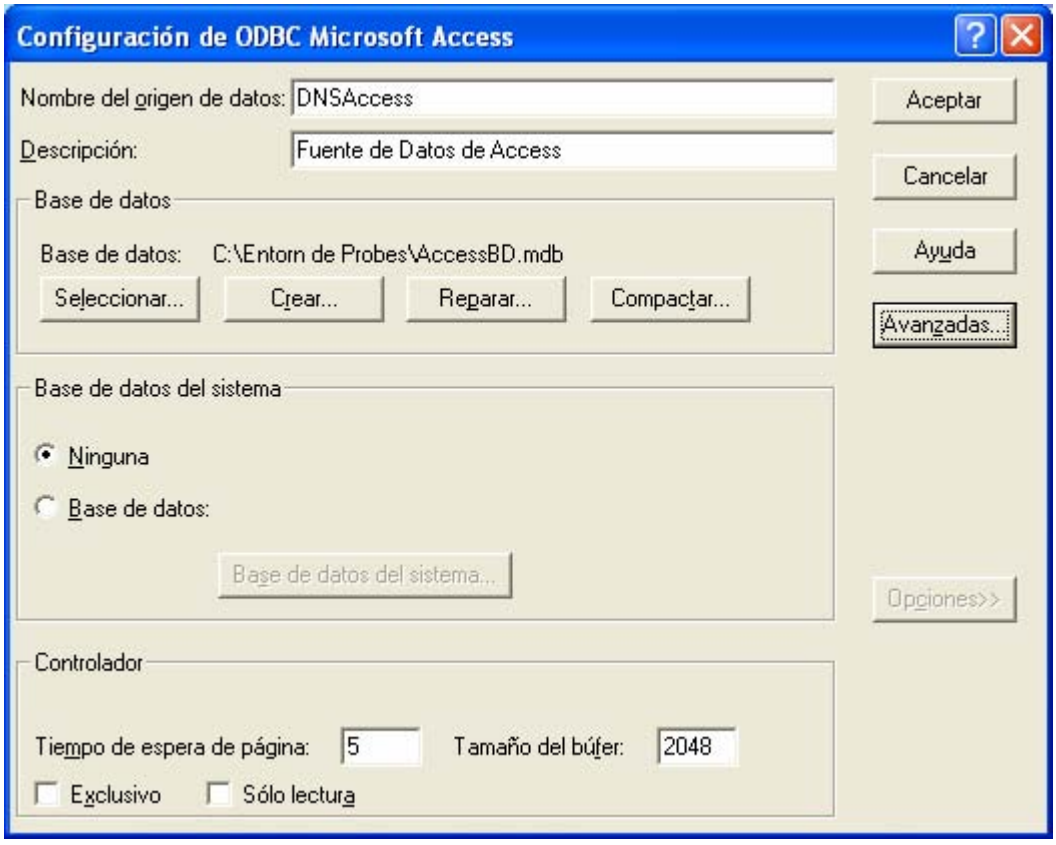
Per comprovar que el DDL generat per l'aplicació funciona independentment del Driver ODBC que s'utilitza, s'ha creat una segona font de dades contra Oracle però utilitzant el driver proporcionat per Oracle tal i com s'indica a la figura:



El servidor de Base de Dades es el mateix que l'indicat a l' apartat anterior.

ODBC Microsoft Access

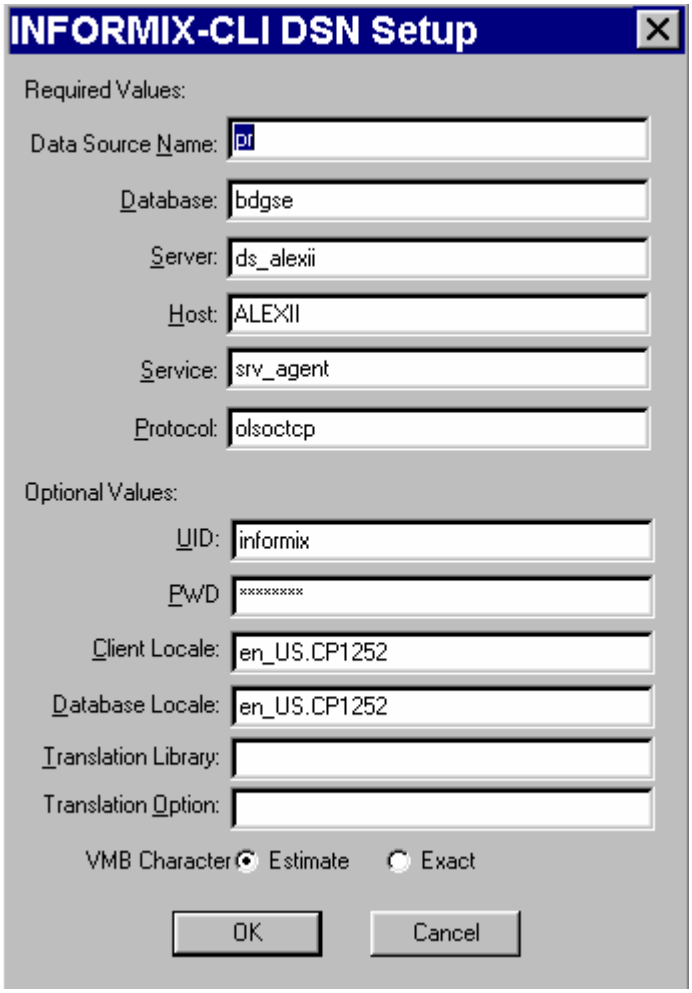
El desenvolupament de l' aplicació també contempla Bases de Dades d'entorns petits com pot ser Microsoft Access i per fer les proves s'ha creat una connexió a una Base de Dades Microsoft tal i com s'indica a la figura:



D'aquesta Base de Dades tan sols indicar que s'ha activat la seguretat de la mateixa per que l'entorn de proves sigui lo mes semblant a un entorn de desenvolupament real.

ODBC Microsoft Access

A una màquina diferent hem fet les proves amb INFORMIX, per a mantenir la coherència de l'aplicació hem utilitzat un driver ODBC, degut a que el servidor utilitzat és la versió Informix Dynamic Server Personal Edition version 7.22 facilitada per la UOC als estudiants no hem disposat d'un entorn amb les mateixes condicions que amb ORACLE, la seva configuració és :



L'entorn de treball ha estat un equip dotat amb el sistema operatiu Windows 98 SE.

ANNEX II
CODI XMI2DBMS.java

```
import java.io.*;
import java.util.*;
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;
import net.sf.saxon.aelfred.SAXDriver;

/**Classe principal de l'aplicació que utilitza els parsers de SAX
 * modificant el tractament dels elements trobats per cridar a les gramàtiques que
 * tinguem definides i amb el seu retorn generar el codi sql a GeneradorDDL. <BR>
 * Després guardem les restriccions a VectorConstraints on ordenarem aquestes
 * per a poder escollir en cas que tinguem més d'una clau primària per taula.
 * Finalment enviarem les comandes a fitxers o al SGBD seleccionat amb ConectorBaseDades.
 * @see Gramatica
 * @see GeneradorDDL
 * @see ConectorBaseDades
 * @see VectorConstraints
 */
public class XMI2DBMS extends DefaultHandler
{
        /**Vector on afegim les gramàtiques que hereten de Gramatica  */
        static Vector vGramaticas;
        /**Atribut utilitzat per a saber que estem dintre d'un element tipus bdy constraint */
        static boolean bDentroConstraint;
        /**Instància de GeneradorDDL   */
        static GeneradorDDL gDDL;
        /**Instància de ConectorBaseDades      */
        static ConectorBaseDades conBD;
        /**Instància de VectorConstraints          */
        static VectorConstraints vComandes;
        /**Funcionament del programa principal.<UL>
         * <LI>  1 Creem el vector de gramàtiques i afegim
         * les gramàtiques definides.
         * <LI> 2  Creem el vector on guardarem les restriccions reconegudes.
         * <LI> 3  Creem la instància de GeneradorDDL amb el SGBD que hem rebut del prompt.
         * <LI> 4  Preparem el parser de SAX per a llegir el document XMI, aquest tracta el document
         * amb els mètodes redefinits <B> startElement i endElement </B>.
```

```
 * <LI> 5 Creem la instància de ConectorBaseDades amb els paràmetres del prompt
 * GestorBaseDades O/F ODBC/FitxerSortida user password.
 * <LI> 6 Tractament de restriccions duplicades com claus primàries a la mateixa taula
 * amb l'ordenació i elecció de clau principal i alternatives.
 * <LI> 7 Genració de fitxer de sortida o generació de les constrains on-line
 * contra el SGBD donat.
 */
public static void main(String[] args) throws Exception {
        //Vector que inclou els parsers de las gramaticas definidas
        vGramaticas = new Vector();
        //Per cada gramatica hem de insertar una instancia nova al vector
        vGramaticas.add(new GramaticaPrimaryKey());
        vGramaticas.add(new GramaticaCheckSimple3());
        vGramaticas.add(new GramaticaCheckSimple4());
        vGramaticas.add(new GramaticaForeignkey());
        vComandes = new VectorConstraints();
        bDentroConstraint = false;
        if (args.length != 6) {
                System.out.println ("XMI2DBMS FitxerXMI GestorBaseDades O/F ODBC/FitxerSortida user password");
        }
        else{
                gDDL = new GeneradorDDL (args[1]);
                XMLReader xr = new net.sf.saxon.aelfred.SAXDriver();
                XMI2DBMS handler = new XMI2DBMS();
                xr.setContentHandler(handler);
                xr.setErrorHandler(handler);
                FileReader r = new FileReader(args[0]);
                xr.parse(new InputSource(r));//crida a startElement
                conBD = new ConectorBaseDades(args[2], args[3], args[4], args[5]);
                vComandes.ordenar();
                int i = 0;
                while (i < vComandes.size()){
                        if (((Constraint) vComandes.elementAt(i)).tipoConstraint.equalsIgnoreCase("PRIMARY KEY")){
                                int numPrimaryKeys = 0;
                                String nomTaula = ((Constraint) vComandes.elementAt(i)).nomTabla;
                                int j = i;
                                while ((j < vComandes.size()) &&
                                        (((Constraint) vComandes.elementAt(j)).tipoConstraint.equalsIgnoreCase("PRIMARY KEY")) &&
                                        (((Constraint) vComandes.elementAt(j)).nomTabla.equalsIgnoreCase(nomTaula))){
                                        j++;
                                        numPrimaryKeys++;
```

```
                                        }
                                        if (numPrimaryKeys == 1 ){
                                                conBD.executarComanda (((Constraint) vComandes.elementAt(i)).cosConstraint);
                                        }
                                        else{
                                                j = i;
                                                int pos = 0;
                                                while (j < (i + numPrimaryKeys)){
                                                        System.out.println (pos + " - " + ((Constraint) vComandes.elementAt(j)).cosConstraint);
                                                        j++;
                                                        pos++;
                                                }
                                                System.out.print("Introduzca el nº que desee que sea la Primary Key [0-" + (numPrimaryKeys-1) + "]");
                                                int opcion = System.in.read() - 48;
                                                int cr = System.in.read();
                                                int lf = System.in.read();
                                                for (j=0; j < numPrimaryKeys; j++){
                                                        if (j == opcion) conBD.executarComanda (((Constraint) vComandes.elementAt(i+j)).cosConstraint);
                                                        else conBD.executarComanda (((Constraint) vComandes.elementAt(i+j)).altConstraint);
                                                }
                                        }
                                i = i + numPrimaryKeys;
                                }
                                else{
                                        conBD.executarComanda (((Constraint) vComandes.elementAt(i)).cosConstraint);
                                        i++;
                                }
                        }
                }
        }

public void startDocument(){
}

public void endDocument(){
}
/**Mètode que detecta que estem dins un constraint body i compara el contingut
 * de la restricció amb les gramàtiques del vector de gramàtiques.
 * Després insertarem la restricció al vector de comandes per generar la sentència
 * al SGBD escollit.
 */
```

```
	public void startElement(String uri, String name, String qName, Attributes atts){
		if ((bDentroConstraint) && (name.compareTo("BooleanExpression")==0)){
			for (int i=0; i < atts.getLength(); i++){
				/* El atribut que te la definicio de la constraint es el que te QNAME = body*/
				if (atts.getQName(i).compareTo("body") == 0){
					for (int elemento=0; elemento< vGramaticas.size(); elemento++){
						/*Per cada gramatica miro si fa Matching y genero el DDL*/
						if (((Gramatica)vGramaticas.elementAt(elemento)).faMatching(atts.getValue(i))) {
							String s[][] = ((Gramatica)vGramaticas.elementAt(elemento)).ddlConstraint(atts.getValue(i));
							try{
								vComandes.addElement(new Constraint(gDDL.generarDDL(s)));
							} catch (Exception e){
								e.printStackTrace();
							}
						}
					}
				}
			}
		}

		if (name.compareTo("Constraint.body")==0) {
			bDentroConstraint = true;
		}
	}

	/**Detecta que estem al final de una restricció.
	 */
	public void endElement(String uri, String name, String qName){
		if (name.compareTo("Constraint.body")==0) {
			bDentroConstraint = false;
		}

	}
}
```

ANNEX III
CODI Gramatica.java

```
/**Classe abstracta utlilitzada com a patró per
 * afegir noves gramàtiques a XMI2DBMS per a ampliar el reconeixement
 * d'elements OCL
 */
public abstract class Gramatica
{
	/**Mètode  abstracte booleà que retornarà cert si es reconeix
	 *  una restricció segons la gramàtica
	 * hereva consultada.
	 */
	public abstract boolean faMatching(String pRegla);

	/**Mètode  abstracte que retornarà els elements necessaris
	 * i suficients per a poder construir la sentència SQL que
	 * reconegui la gramàtica hereva.
	 */
	public abstract String[][] ddlConstraint(String pRegla);
}
```

ANNEX IV
CODI GramaticaPrimaryKey.java

```
import java.util.*;
import java.io.*;
/**Classe que implementa el reconeixement de claus primàries o alternatives
 *  amb el format: <BR><I>
 * context Taula inv cClave : Taula . allInstances ->
 * forAll ( c1 , c2 | c1 <> c2 implies c1 . clave <> c2 . clave )</I><BR>
 * També accepta les permutacions de camps o instàncies, la definició
 * de claus múltiples amb ANDs o ORs
 * i la substitució de la segona instància per self,
 * mètode força utilitzat a OCL.<BR>
 * Retorna els elements suficients per que GeneradorDDL generi
 * la calu primària i la alternativa per a un gestor determinat.
 */
public class GramaticaPrimaryKey extends Gramatica {
        /**Aquí guardem la primera part de la definició de la clau:<BR>
         * context Taula inv cClave : Taula . allInstances -> forAll
         */
        private String prefijo[];
        /**Noms de les intàncies definides, a l'exemple sería:<BR>
         * instancias[0]= c1 i instancias[1]=c2
         */
        private String instancias[];
        /**Part de la definició de la clau situada després del token '|'
         * i abans del token implies */
        public String condiciones_izqda[];
        /**Part de la definició de la clau situada després del token 'implies'
         * i el final de la sentència afectada pel token forAll */
        public String condiciones_der[];

        /**Nom de la taula que conté la clau primària o alterantiva*/
        private String nombreTabla;
        /**Nom de la restricció*/
        private String nombreConstraint;
        /**Nom de la primera instància de la taula, a l'exemple c1*/
        private String nombreInstancia1;
        /**Nom de la segona instància de la taula, a l'exemple c2*/
        private String nombreInstancia2;
        /**Repositori on guardarem el camp o camps que composen
         * la clau primària o alternativa si finalment la definim així*/
```

```
private String camposClave[] = new String[100];
/**Nombre de camps que composen la clau */
int numCamposClave = 0;
/**Comprova l'existència de tots els "tokens" necessaris
 *  per que puguem continuar avaluant l'expressió OCL com a vàlida
 *  i separa per referència les instàncies i condicions
 *  d'aquests tokens per després avaluar-les.
 */
public boolean comprobar_primer_nivel(String pRegla){
        /**Paràmetre per a detectar el final de la clausula forAll
         * comptant els parèntesi, l'eina CASE és responsable de la validació
         * de la sentència OCL.
         */
        int parentesis_pendientes = 0;
        /**Cadena retornada per StringTokenizer que es va analitzant
         * al bucle while fins que hem recorregut tota la restricció */
        String elemento_actual;
        /**Paràmetre que conté la longitud del prefix*/
        int longPrefijo = 0;
        /** Paràmetre que compta el nombre d'instàncies*/
        int longInstancias = 0;
        /** Paràmetre que compta el nombre d'elements de les
         * condicions trobades abans del token implies i després del token '|'*/
        int longCondiciones_izqda = 0;
        /** Paràmetre que compta el nombre d'elements de les
         * condicions trobades després del token '|'
         * fins trobar la fí de la sentència forAll*/
        int longCondiciones_der = 0;
        /**Paràmetre on guardem el punt de la clau on estem:<UL><LI>
         * <LI> 0-Prefijo,
         * <LI> 1-Instancias,
         * <LI> 2-Condiciones_Izquierda,
         * <LI> 3-Condicones_Derecha
         * </UL>
         *  */
        int donde_estoy = 0;
        if (pRegla == null) return false;
        StringTokenizer st = new StringTokenizer(pRegla);
        while (st.hasMoreElements()){
                elemento_actual = st.nextElement().toString();
                if (donde_estoy == 4) donde_estoy++;
```

```
                if ((donde_estoy == 3) && (elemento_actual.compareTo(")")==0) && (parentesis_pendientes == 0)) donde_estoy++;
                if ((donde_estoy == 3) && (elemento_actual.compareTo(")")!=0) && (parentesis_pendientes == 0)) longCondiciones_der++;
                if ((donde_estoy == 2) && (elemento_actual.compareTo("implies")==0)) donde_estoy++;
                if ((donde_estoy == 2) && (elemento_actual.compareTo("implies")!=0)) longCondiciones_izqda++;
                if ((donde_estoy == 1) && (elemento_actual.compareTo("|")==0)) donde_estoy++;
                if ((donde_estoy == 1) && (elemento_actual.compareTo("|")!=0)) longInstancias++;
                if ((donde_estoy == 0) && (elemento_actual.compareTo("(") != 0)) longPrefijo ++;
                if ((donde_estoy == 0) && (elemento_actual.compareTo("(") == 0)) donde_estoy++;
        }
        if (donde_estoy != 4) return false;
        prefijo = new String[longPrefijo];
        instancias = new String[longInstancias];
        condiciones_izqda = new String[longCondiciones_izqda];
        condiciones_der = new String[longCondiciones_der];
        st = new StringTokenizer(pRegla);
        for (int i=0; i < longPrefijo;i++) prefijo[i] = st.nextElement().toString();
        elemento_actual = st.nextElement().toString(); //Salto el '('
        for (int i=0; i < longInstancias; i++) instancias[i] = st.nextElement().toString();
        elemento_actual = st.nextElement().toString(); //Salto el '|'
        for (int i=0; i < longCondiciones_izqda; i++) condiciones_izqda[i] = st.nextElement().toString();
        elemento_actual = st.nextElement().toString(); //Salto el 'implies'
        for (int i=0; i < longCondiciones_der; i++) condiciones_der[i] = st.nextElement().toString();
        return true;
}
/**Comprova que el prefix sigui de la classe:<BR>
* context Taula inv cClave : Taula . allInstances -> forAll
* Utilitza els paràmetres d'entrada per referència. <BR>
* Retorna cert si el prefix s'ajusta a norma.
* */
public boolean comprobar_prefijo(){
        boolean retorno = true;
        for (int i=0; i < prefijo.length; i++){
                if (i==0) retorno = (retorno && prefijo[i].equals("context"));
                if (i==1) nombreTabla = prefijo[i];
                if (i==2) retorno = (retorno && prefijo[i].equals("inv"));
                if (i==3) nombreConstraint = prefijo[i];
                if (i==4) retorno = (retorno && prefijo[i].equals(":"));
                if (i==5) retorno = (retorno && prefijo[i].equals(nombreTabla));
                if (i==6) retorno = (retorno && prefijo[i].equals("."));
                if (i==7) retorno = (retorno && prefijo[i].equals("allInstances"));
                if (i==8) retorno = (retorno && prefijo[i].equals("->"));
```

```
                if (i==9) retorno = (retorno && prefijo[i].equals("forAll"));
        }
        return (retorno && (prefijo.length==10));
}

/**Utilitza els paràmetres d'entrada per referència,
 *  si tenim dues instàncies guarda els seus noms per a
 * poder avaluar-les a les condicions, si només tenim una instància
 *  força que la segona instància de les condicions sigui self
 *  per establir igualtats o desigualtats. <BR>
 * Retorna cert per ambdós casos i altrament fals.
 */
public boolean comprobar_instancias(){
        boolean retorno = true;
        for (int i=0; i < instancias.length; i++){
                if (i==0) nombreInstancia1 = instancias[i];
                if (i==1) retorno = (retorno && instancias[i].equals(","));
                if (i==2) {
                        nombreInstancia2 = instancias[i];
                        retorno = (retorno && (!nombreInstancia1.equals(nombreInstancia2)));
                }
        }
        if (instancias.length==3) return (retorno);
        else if (instancias.length==1){nombreInstancia2 ="self"; return (retorno);}
        else retorno=false; return(retorno);
}

/**Rep com a paràmetre la llista de condicions a avaluar,
 * si rep una llista amb longitud 3 estarem a una condició del tipus:<BR>
 * c1=c2 retorna 1 <BR>
 * c1<>c2 retorna 2 <BR>
 * si la llista de condicions és superior utilitza
 *  el mètode comprobar_condicion per guardar els possibles
 *  camps de la clau primària i avaluar la condició retornant un enter.<BR>
 * Si les condicions que trobem totes tenen iguals
 * separats amb ANDs retornarà 3<BR>
 * Si les condicions que trobem totes tenen desigualtats
 *  separats amb ORs retornarà 4
 */
public int comprobar_condiciones(String condiciones[]){
        int retorno=0;
```

```
boolean valido = true;
if (condiciones.length == 3){
	//Estoy en Instancia = Instancia | Instancia <> Instancia
	for (int i=0; i<condiciones.length; i++){
		if (i==0) valido = (valido && (condiciones[i].equals(nombreInstancia1) || condiciones[i].equals(nombreInstancia2)));
		if (i==1) {
			valido = (valido && (condiciones[i].equals("=") || condiciones[i].equals("<>")));
			if (condiciones[i].equals("=")) retorno = 1;
			if (condiciones[i].equals("<>")) retorno = 2;
		}
		if (i==2) valido = (valido && (condiciones[i].equals(nombreInstancia1) || condiciones[i].equals(nombreInstancia2)));
	}
	valido = (valido && (!condiciones[0].equals(condiciones[2])));
}
else{
	//Descomponer cadena en condiciones
	numCamposClave = 0;
	int tipo_condicion = 1;
	int num_condiciones = 0;
	String condicion_simple[] = new String[7];
	int posicion_condicion_simple = 0;
	int posicion_condiciones=0;
	while (posicion_condiciones < condiciones.length){
		condicion_simple[posicion_condicion_simple] = condiciones[posicion_condiciones];
		posicion_condicion_simple++;
		posicion_condiciones++;
		if (posicion_condicion_simple == condicion_simple.length){
			int resultado_condicion=comprobar_condicion(condicion_simple);
			tipo_condicion = tipo_condicion * resultado_condicion;
			if (posicion_condiciones < (condiciones.length-1)){ //No hem arribat al fi de la cadena
				String operador =	condiciones[posicion_condiciones];
				if ((operador.equalsIgnoreCase("AND") == false)
					&& resultado_condicion==1) tipo_condicion = 0;
				if ((operador.equalsIgnoreCase("OR") == false)
					&& resultado_condicion==2) tipo_condicion = 0;
				posicion_condiciones++;
			}
			posicion_condicion_simple = 0;
			num_condiciones++;
		}
	}
```

```
                if (posicion_condicion_simple != 0) tipo_condicion = 0; //No es una cadena completa
                if (tipo_condicion == 1)
                        retorno = 3;
                else {
                        if (Math.pow(2, num_condiciones) == tipo_condicion)
                                retorno = 4;
                        else
                                retorno = 0;
                }
        }
        if (!valido) retorno=0;
        return retorno;
}

/**Rep com a paràmetre les condicions simples que li passa
 * comprobar_condiciones, cada condició simple te una longitud de 7
 * i aquest mètode emmagatzema els possibles camps de la clau primària
 *  i retorna:<BR>
 * 1 per a les igualtats <BR>
 * 2 per a les desigualtats. <BR>
 */
private int comprobar_condicion(String condicion[]){
        String nombreCampo = null;
        boolean valido = true;
        int retorno=0;

        for (int i=0; i < condicion.length; i++){
                if (i==0) valido = (valido && (condicion[i].equals(nombreInstancia1) || condicion[i].equals(nombreInstancia2)));
                if (i==1) valido = (valido && (condicion[i].equals(".")));
                if (i==2) nombreCampo = condicion[i];
                if (i==3) {
                        valido = (valido && (condicion[i].equals("=") || condicion[i].equals("<>")));
                        if (condicion[i].equals("=")) retorno = 1;
                        if (condicion[i].equals("<>")) retorno = 2;
                }
                if (i==4) valido = (valido && !(condicion[i].equals(condicion[i-4]))&& (condicion[i].equals(nombreInstancia1) || condicion[i].equals(nombreInstancia2)));
                if (i==5) valido = (valido && (condicion[i].equals(".")));
                if (i==6) valido = (valido && (condicion[i].equals(nombreCampo)));
        }
        camposClave[numCamposClave] = nombreCampo;
        numCamposClave++;
```

```
		if (!valido) retorno=0;
		return retorno;
	}

	/**Mètode booleà que avalua a cert les claus verificades
	 * segons el següent esquema:<BR>
	 * 1-3 vol dir igualtat d'instàncies amb
	 * igualats de camps separats per ANDs <BR>
	 *  3-1 és el mateix però amb ordre invertit.<BR>
	 * 2-4 vol dir desigualtat d'instàncies amb
	 * desigualtats de camps separats per ORs<BR>
	 * 4-2 és el mateix però amb ordre invertit.
	 * retorna cert pel casos anteriors, altrament retorna fals.
	 * @return boolean
	 */
	public boolean faMatching(String pRegla){
		boolean retorno;
		retorno = comprobar_primer_nivel(pRegla);
		if (retorno) {
			retorno = comprobar_prefijo();
		}
		if (retorno) {
			retorno = comprobar_instancias();
		}
		if (retorno){
			int derecha = comprobar_condiciones (condiciones_der);
			int izquierda = comprobar_condiciones (condiciones_izqda);
			retorno = ((derecha==1) && (izquierda ==3) || (derecha==2) && (izquierda ==4) ||
				(derecha==3) && (izquierda ==1) || (derecha==4) && (izquierda ==2));
		}
		return retorno;
	}

	/**Repositori que conté els elements necessaris per que GeneradorDDL
	 * confeccioni la restricció específica del SGBD determinat
	* @return <UL><LI> retorno[0][0]= "PRIMARY KEY"
	 * <LI> retorno[1][0]= nombreTabla
	 * <LI> retorno[2][0]= nombreConstraint
	 * <LI> retorno[3][0]= nombreTabla2 (referida)
	 * <LI> retorno[3][I]= nom dels camps
	 * </UL>
```

```
	 */
	public String[][] ddlConstraint(String pRegla){
		String retorno[][] = null;
		if (faMatching(pRegla)){
			retorno = new String[4][];
			retorno[0] = new String[1];
			retorno[0][0] = new String("PRIMARY KEY");
			retorno[1] = new String[1];
			retorno[1][0] = new String(nombreTabla);
			retorno[2] = new String[1];
			retorno[2][0] = new String(nombreConstraint);
			retorno[3] = new String[numCamposClave];
			for (int i=0; i < numCamposClave; i++) retorno[3][i] = new String(camposClave[i]);
		}
		return retorno;
	}
}
```

ANNEX V
CODI GramaticaCheckSimple3.java

```
import java.util.*;
import java.io.*;
import java.lang.*;

/**Classe que implementa el reconeixement de checks simples amb el format:<BR><I>
 * context Table inv nameInv : <BR>
 * Table.allInstances -> forAll(c1|c1.camp <50 and...or..)</I><BR>
 * No accepta l'inclusió al mateix invariant d'altre tipus de restriccions
 * no simples com assertions, triggers definicions de not nulls, claus...<BR>
 * Això es detecta amb el mètode implementat restricted_word.
 * Sí accepta combinacions d'operadors lògics i matemàtics.
 */
public class GramaticaCheckSimple3 extends Gramatica
{

		/**Atribut incremental per recòrrer la cadena de la restricció*/
		int estado = 0;
		/**Atribut que conté la logitud del cos de la restricció*/
		int long_check=0;
		/**Atribut utilitzat per eliminar els parèntesi,
		 * no fan falta als checks simples*/
		int compta_parentesi = 0;
		/**Atribut incremental per nombrar el check si cal fer-ho*/
		int cont_check=0;
		/**Nom de la taula*/
		String nombreTabla=null;
		/**Nom de l'invariant*/
		String nombreInvariante=null;
		/**Nom de la instància de la taula, a l'exemple sería 'c1'*/
		String nombreElemento1=null;

		/**Repositori on guardarem el cos del check:
		 * ALTER TABLE ADD CONSTRAINT nombreInvariante
		 * CHECK (cos del check);
		 */
		private String cadenaCheck[]= new String[1000];
```

```
StringTokenizer st;

/**Mètode heretat de la clase Gramatica que retorna cert si la nostra
 * gramàtica reconeix l'invariant com a restricció de check simple
 */
public boolean faMatching(String pRegla){
        estado=0;
        long_check=0;
        compta_parentesi = 0;
        st= new StringTokenizer(pRegla);

                while (st.hasMoreElements()){
                        String cadena = (String)st.nextElement();

                        if ((estado ==13)&& restricted_word(cadena)) estado=14;

                        if ((estado ==13)&& ((cadena.compareTo(")") == 0) && (compta_parentesi!=0))) {
                                cadenaCheck[long_check]=cadena;long_check++;compta_parentesi--;}
                        if ((estado ==13)&& ((cadena.compareTo("(") == 0) && (compta_parentesi!=0))) {
                                cadenaCheck[long_check]=cadena;long_check++;compta_parentesi++;}
                        if ((estado ==13)&&  (cadena.compareTo(".") != 0) &&
                           (cadena.compareTo(nombreElemento1) != 0)&& (cadena.compareTo("(") != 0)&& (cadena.compareTo(")") != 0)) {
                                cadenaCheck[long_check]=cadena;long_check++;}

                        if ((estado ==12) && (cadena.compareTo("|") == 0)) estado++;
                        if ((estado ==11)) {nombreElemento1 = cadena;estado++;}
                        if ((estado ==10) && (cadena.compareTo("(") == 0)) estado++;
                        if ((estado ==9) && (cadena.compareTo("forAll") == 0)) estado++;
                        if ((estado ==8) && (cadena.compareTo("-&gt;") == 0)) estado++;
                        if ((estado ==8) && (cadena.compareTo("->") == 0)) estado++;
                        if ((estado ==7) && (cadena.compareTo("allInstances") == 0)) estado++;
                        if ((estado ==6) && (cadena.compareTo(".") == 0)) estado++;
                        if ((estado ==5)&& restricted_word(cadena)) estado=14; //sortida de la gramàtica, no fa matching
                        if ((estado ==5) && (cadena.compareTo(nombreTabla) == 0)) estado++;
                        if ((estado ==4) && (cadena.compareTo(":") == 0)) estado++;
                        if ((estado ==3)&& (cadena.equals(":")))
                                {nombreInvariante = nombreTabla.substring(0,2)+"CK3"+cont_check;
                                 cont_check++;
                                 estado++;estado++;}
                        if  ((estado ==3)&& !(cadena.equals(":")))
                                {nombreInvariante = cadena;estado++;}
```

```
                        if ((estado ==2) && (cadena.compareTo("inv") == 0)) estado++;
                        if  (estado ==1) {nombreTabla = cadena;estado++;}
                        if ((estado ==0) && (cadena.compareTo("context") == 0)) estado++;
                }

                if (estado==13) { return (true);}
                        else return (false);
        }
/**Mètode privat que permet descartar les restriccions que necessitarien
 * de l'implementació de TRIGGERS  o ASSERTIONs i d'altres tipus
 * de restriccions que no són objecte d'aquesta gramàtica
 */
private boolean restricted_word(String pCadena){
/**Atribut booleà que indica si hem trobat algun mot que no s'ha de
 *  trobar a un check simple per impedir la construcció
 * de una sentencia DDL incorrecta
 */
boolean not_restricted=false;
if ((pCadena.equals("->")) || (pCadena.equals("implies")) || (pCadena.equals("|"))|| (pCadena.equals("["))  not_restricted=true;
if ((pCadena.equals("inv")) || (pCadena.equals("let")) || (pCadena.equals("if")))  not_restricted=true;
if ((pCadena.equals("oclIsTypeOf")) || (pCadena.equals("oclIsKindOf")) || (pCadena.equals("oclInState"))||
        (pCadena.equals("oclIsNew"))|| (pCadena.equals("oclAsType")))  not_restricted=true;

return not_restricted;
}
/**Mètode heretat de la clase Gramatica que retorna els elements
 * necessaris per a la construcció d'un check simple per part de
 * la classe GeneradorDDL
 * @return <UL><LI> retorno[0][0]= "CHECK SIMPLE"
 * <LI> retorno[1][0]= nombreTabla
 * <LI> retorno[2][0]= nombreInvariante
 * <LI> retorno[3][0]= cos del check
 * </UL>
 */
public String[][] ddlConstraint(String pRegla){
/**Repositori on guardarem el cos del check: <BR><I>
 * ALTER TABLE ADD CONSTRAINT nombreInvariante
 * CHECK (cos del check); </I>
 */
String retorno[][] = null;
if (faMatching(pRegla)){
```

```
                        retorno = new String[4][];
                        retorno[0] = new String[1];
                        retorno[0][0] = new String("CHECK SIMPLE");
                        retorno[1] = new String[1];
                        retorno[1][0] = new String(nombreTabla);
                        retorno[2] = new String[1];
                        retorno[2][0] = new String(nombreInvariante);
                        retorno[3] = new String[long_check];
                        for (int i=0; i < long_check;i++) retorno[3][i] = new String(cadenaCheck[i]+" ");

                }
                return retorno;
        }


}
```

ANNEX VI
CODI GramaticaCheckSimple4.java

```
import java.util.*;
import java.io.*;

/**Classe que implementa el reconeixement de checks simples amb el format:<BR><I>
 * context Table inv nameInv : <BR>
 * self. camp1 <50 and self.camp2 < 50000 .... )</I><BR>
 * No accepta l'inclusió al mateix invariant d'altre tipus de restriccions
 * no simples com assertions, triggers definicions de not nulls, claus...
 */
public class GramaticaCheckSimple4 extends Gramatica
{
		/**Atribut incremental per recòrrer la cadena de la restricció*/
		int estado = 0;
		/**Atribut que conté la logitud del cos de la restricció*/
		int long_check=0;
		/**Atribut utilitzat per eliminar els parèntesi,
		 * no fan falta als checks simples*/
		int compta_parentesi = 0;
		/**Atribut incremental per nombrar el check si cal fer-ho*/
		int cont_check=0;
		/**Nom de la taula*/
		String nombreTabla=null;
		/**Nom de l'invariant*/
		String nombreInvariante=null;
		/**Repositori on guardarem el cos del check: <BR><I>
		 * ALTER TABLE ADD CONSTRAINT nombreInvariante
		 * CHECK (cos del check); </I>
		 */
		private String cadenaCheck[]= new String[1000];


		StringTokenizer st;
		/**Mètode heretat de la clase Gramatica que retorna cert si la nostra
		 * gramàtica reconeix l'invariant com a restricció de check simple
		 */
		public boolean faMatching(String pRegla){
			estado=0;
			long_check=0;
			compta_parentesi = 0;
```

```
		st= new StringTokenizer(pRegla);

			while (st.hasMoreElements()){
				String cadena = (String)st.nextElement();

				if ((estado==6) &&  restricted_word(cadena)) estado=7;

				if ((estado ==6)&& ((cadena.compareTo(")") == 0) && (compta_parentesi!=0))) {
					cadenaCheck[long_check]=cadena;long_check++;compta_parentesi--;}
				if ((estado ==6)&& ((cadena.compareTo("(") == 0) && (compta_parentesi!=0))) {
					cadenaCheck[long_check]=cadena;long_check++;compta_parentesi++;}
				if ((estado ==6)&&  (cadena.compareTo(".") != 0) &&
				   (cadena.compareTo("self") != 0)&& (cadena.compareTo("(") != 0)&& (cadena.compareTo(")") != 0)) {
					cadenaCheck[long_check]=cadena;long_check++;}

				if ((estado ==5)&& restricted_word(cadena)) estado=7;//sortida de la gramàtica, no fa matching
				if ((estado ==5) && (cadena.compareTo("self") == 0)) estado++;
				if ((estado ==4) && (cadena.compareTo(":") == 0)) estado++;
				if ((estado ==3)&& (cadena.equals(":")))
					{nombreInvariante = nombreTabla.substring(0,2)+"CK4"+cont_check;
					 cont_check++;
					 estado++;estado++;}
				if  ((estado ==3)&& !(cadena.equals(":")))
					{nombreInvariante = cadena;estado++;}
				if ((estado ==2) && (cadena.compareTo("inv") == 0)) estado++;
				if  (estado ==1) {nombreTabla = cadena;estado++;}
				if ((estado ==0) && (cadena.compareTo("context") == 0)) estado++;
			}

			if (estado==6) { return (true);}
				else return (false);
		}
/**Mètode privat que permet descartar les restriccions que necessitarien
 * de l'implementació de TRIGGERS  o ASSERTIONs i d'altres tipus
 * de restriccions que no són objecte d'aquesta gramàtica
 */
private boolean restricted_word(String pCadena){
/**Atribut booleà que indica si hem trobat algun mot que no s'ha de
 *  trobar a un check simple per impedir la construcció
 * de una sentencia DDL incorrecta
 */
```

```
                boolean not_restricted=false;
        if ((pCadena.equals("->")) || (pCadena.equals("implies")) || (pCadena.equals("|"))|| (pCadena.equals("["))  not_restricted=true;
        if ((pCadena.equals("inv")) || (pCadena.equals("let")) || (pCadena.equals("if")))  not_restricted=true;
        if ((pCadena.equals("oclIsTypeOf")) || (pCadena.equals("oclIsKindOf")) || (pCadena.equals("oclInState"))||
                (pCadena.equals("oclIsNew"))|| (pCadena.equals("oclAsType")))  not_restricted=true;

        return not_restricted;
        }
        /**Mètode heretat de la clase Gramatica que retorna els elements
         * necessaris per a la construcció d'un check simple per part de
         * la classe GeneradorDDL
         * @return <UL><LI> retorno[0][0]= "CHECK SIMPLE"
         * <LI> retorno[1][0]= nombreTabla
         * <LI> retorno[2][0]= nombreInvariante
         * <LI> retorno[3][0]= cos del check
         * </UL>
         */
        public String[][] ddlConstraint(String pRegla){
        /**Repositori que conté els elements necessaris per que GeneradorDDL
         * confeccioni la restricció específica del SGBD determinat.<BR>
         */
        String retorno[][] = null;
        if (faMatching(pRegla)){
                retorno = new String[4][];
                retorno[0] = new String[1];
                retorno[0][0] = new String("CHECK SIMPLE");
                retorno[1] = new String[1];
                retorno[1][0] = new String(nombreTabla);
                retorno[2] = new String[1];
                retorno[2][0] = new String(nombreInvariante);
                retorno[3] = new String[long_check];
                for (int i=0; i < long_check;i++) retorno[3][i] = new String(cadenaCheck[i]+" ");

        }
        return retorno;
    }


}
```

ANNEX VII
CODI GramaticaForeignKey.java

```
import java.util.*;
import java.io.*;
/**Classe que implementa el reconeixement de claus foranes amb el format: <BR><I>
 * context Table1 inv nameInv : <BR>
 * Table1.allInstances -> forAll ( i | Table2.allInstances  ->
 * collect ( j | i.keyTable1 =j.keyTable2 ) -> size > 0)</I>
 */
public class GramaticaForeignkey extends Gramatica {

		/**Atribut incremental per recòrrer la cadena de la restricció*/
		int estado = 0;
		/**Atribut que conté la logitud del nombre de camps que composen
		 * la clau alternativa + 1 ja que conté el nom de la taula 2 'Table2'
		 */
		int long_check=0;
		/**Atribut utilitzat per eliminar els parèntesi,
		 * no fan falta als checks simples*/
		int compta_parentesi = 0;
		/**Atribut incremental per nombrar la clau forana si cal fer-ho*/
		int cont_foreign=0;
		/**Nom de la taula que contè la clau forana
		 * Table1*/
		String nombreTabla1=null;
		/**Nom de la taula que contè el camp referit per la clau forana
		 * Table2*/
		String nombreTabla2=null;
		/**Nom de la restricció de clau forana
		 */
		String nombreInvariante=null;
		/**Nom de l'instància de Table1, a l'exemple 'i' */
		String nombreElemento1=null;
		/**Nom de l'instància de Table2, a l'exemple 'j' */
		String nombreElemento2=null;
		/**Nom del camp de la clau forana, a l'exemple 'keyTable1'
		 * Si hi ha més d'un camp contindrà els seus nom separats per comes*/
		String nombreCampo1=null;
		/**Atribut incremental per nombrar la clau forana si cal fer-ho*/
		String nombreCampo2=null;
```

```
/**booleà que permet controlar si hem trobat l'instància i
 * (correspón a la taula de la clau)
 * per controlar canvis d'ordre a l'escriptura de la restricció*/
boolean trobat1;
/**booleà que permet controlar si hem trobat l'instància j
 * (correspón a la taula referida per la clau)
 * per controlar canvis d'ordre a l'escriptura de la restricció*/
boolean trobat2;

/**Repositori on guardarem el nom de la taula2,
 * els noms dels camps de la clau forana separats per comes
 * i els noms corresponents referits separats per comes*/
private String cadenaCheck[]= new String[1000];


StringTokenizer st;

/**Mètode heretat de la clase Gramatica que retorna cert si la nostra
 * gramàtica reconeix l'invariant com a restricció de clau forana
 */
public boolean faMatching(String pRegla){
        estado=0;
        long_check=0;
        nombreCampo1="";
        nombreCampo2="";
        trobat1=false;
        trobat2=false;
        compta_parentesi = 0;
        st= new StringTokenizer(pRegla);

                while (st.hasMoreElements()){
                        String cadena = (String)st.nextElement();
                        if (estado ==34) estado++;
                        if ((estado ==33) && (cadena.compareTo(")") == 0)) estado++;
                        if ((estado ==32) && (cadena.compareTo("0") == 0)) estado++;
                        if ((estado ==31) && (cadena.compareTo(">") == 0)) estado++;
                        if ((estado ==30) && (cadena.compareTo("size") == 0)) estado++;
                        if ((estado ==29) && (cadena.compareTo("->") == 0)) estado++;

                        if ((estado<29)&&(cadena.compareTo(")") == 0))compta_parentesi--;
                        if ((estado<29)&&(cadena.compareTo("(") == 0))compta_parentesi++;
```

```
if ((estado ==28)&& (cadena.compareTo(")") == 0) && (compta_parentesi==1)){
        estado++;
        }

//if ((estado ==28)&& (cadena.compareTo(")") == 0)) estado++;
if ((estado ==28) &&(cadena.compareTo("and") == 0)){
        nombreCampo1 = nombreCampo1+",";
        nombreCampo2 = nombreCampo2+",";
        estado= 21;
        }
if ((estado ==27&& trobat1)&&!((cadena.equals("("))||(cadena.equals(")")))){
        nombreCampo2 = nombreCampo2+cadena;estado++;
        trobat1=false;
        }
if ((estado ==27&& trobat2)&&!((cadena.equals("("))||(cadena.equals(")")))){
        nombreCampo1 = nombreCampo1+cadena;estado++;
        trobat2=false;
        }
if ((estado ==26) && (cadena.compareTo(".") == 0)) estado++;
if ((estado ==25) && (cadena.compareTo(nombreElemento2) == 0)&& trobat1) {
        estado++;
        }
if ((estado ==25) && (cadena.compareTo(nombreElemento1) == 0)&& trobat2) {
        estado++;
        }
if ((estado ==24) && (cadena.compareTo("=") == 0)) estado++;
if ((estado ==23) &&!(cadena.equalsIgnoreCase("and"))&& trobat2)
        {nombreCampo2 = nombreCampo2+cadena;estado++;}
if ((estado ==23) &&!(cadena.equalsIgnoreCase("and"))&& trobat1)
        {nombreCampo1 = nombreCampo1+cadena;estado++;}
if ((estado ==22) && (cadena.compareTo(".") == 0)) estado++;
if ((estado ==21) && (cadena.compareTo(nombreElemento1) == 0))
        {estado++;
        trobat1=true;
        }
if ((estado ==21) && (cadena.compareTo(nombreElemento2) == 0))
        {estado++;
        trobat2=true;
        }
if ((estado ==20) && (cadena.compareTo("|") == 0)) estado++;
if ((estado ==19)) {nombreElemento2 = cadena;estado++;}
```

```
                        if ((estado ==18) && (cadena.compareTo("(") == 0)) estado++;
                        if ((estado ==17) && (cadena.compareTo("collect") == 0)) estado++;
                        if ((estado ==16) && (cadena.compareTo("->") == 0)) estado++;
                        if ((estado ==15) && (cadena.compareTo("allInstances") == 0)) estado++;
                        if ((estado ==14) && (cadena.compareTo(".") == 0)) estado++;
                        if ((estado ==13)){nombreTabla2 = cadena;estado++;}
                        if ((estado ==12) && (cadena.compareTo("|") == 0)) estado++;
                        if ((estado ==11)) {nombreElemento1 = cadena;estado++;}
                        if ((estado ==10) && (cadena.compareTo("(") == 0)) estado++;
                        if ((estado ==9) && (cadena.compareTo("forAll") == 0)) estado++;
                        if ((estado ==8) && (cadena.compareTo("-&gt;") == 0)) estado++;
                        if ((estado ==8) && (cadena.compareTo("->") == 0)) estado++;
                        if ((estado ==7) && (cadena.compareTo("allInstances") == 0)) estado++;
                        if ((estado ==6) && (cadena.compareTo(".") == 0)) estado++;
                        if ((estado ==5) && (cadena.compareTo(nombreTabla1) == 0)) estado++;
                        if ((estado ==4) && (cadena.compareTo(":") == 0)) estado++;
                        if ((estado ==3)&& (cadena.equals(":")))
                                {nombreInvariante = nombreTabla1.substring(0,2)+"FK"+cont_foreign;
                                 cont_foreign++;
                                 estado++;estado++;}
                        if  ((estado ==3)&& !(cadena.equals(":")))
                                {nombreInvariante = cadena;estado++;}
                        //if  (estado ==3) {nombreInvariante = cadena;estado++;}
                        if ((estado ==2) && (cadena.compareTo("inv") == 0)) estado++;
                        if  (estado ==1) {nombreTabla1 = cadena;estado++;}
                        if ((estado ==0) && (cadena.compareTo("context") == 0)) estado++;

                }

                if (estado==34) {
                        long_check=3;


                        return (true);}
                        else return (false);
        }

/**Mètode heretat de la clase Gramatica que retorna els elements
 * necessaris per a la construcció d'un check simple per part de
 * la classe GeneradorDDL
 * @return <UL><LI> retorno[0][0]= "FOREIGN KEY"
```

```
		 * <LI> retorno[1][0]= nombreTabla1
		 * <LI> retorno[2][0]= nombreInvariante
		 * <LI> retorno[3][0]= nombreTabla2 (referida)
		 * <LI> retorno[3][1]= nom dels camps separats per comes
		 * <LI> retorno[3][1]= nom dels camps (referits) separats per comes
		 * </UL>
		 */
		public String[][] ddlConstraint(String pRegla){
		/**Repositori que conté els elements necessaris per que GeneradorDDL
		 * confeccioni la clau forana específica del SGBD determinat
		 */
		String retorno[][] = null;
		if (faMatching(pRegla)){
			retorno = new String[4][];
			retorno[0] = new String[1];
			retorno[0][0] = new String("FOREIGN KEY");
			retorno[1] = new String[1];
			retorno[1][0] = new String(nombreTabla1);
			retorno[2] = new String[1];
			retorno[2][0] = new String(nombreInvariante);
			retorno[3] = new String[long_check];
			retorno[3][0] = new String(nombreTabla2);
			retorno[3][1] = new String(nombreCampo1);
			retorno[3][2] = new String(nombreCampo2);

		}
		return retorno;
	}


}
```

ANNEX VIII
CODI Constraint.java

```
import java.util.*;
/**Constructor de l'estructura de la restrcció
* que contindrà les elements necessàris per l'enviament
* al SGBD seleccionat i dades per a ordenar i sleccionar
* sentències alternatives si s'escau*/
public class Constraint{
	/** Cadena que contè el nom de la taula
	 *  a la que s'aplicarà la restricció*/
	public String nomTabla;
	/** Cadena que contè el tipus de restricció
	 * "PRIMARY KEY", "CHECK", "FOREIGN KEY"...*/
	public String tipoConstraint;
	/** Cadena que contè el nom de la restricció*/
	public String nomConstraint;
	/** Cadena que contè el cos de la restricció
	 * tal i com l'hem d'enviar al SGBD corresponent*/
	public String cosConstraint;
	/** Cadena que contè el cos de la restricció alternativa a escollir
	 * tal i com l'hem d'enviar al SGBD corresponent
	 * sempre que l'hàgim definida a generadorDDL.java*/
	public String altConstraint;

	public Constraint(String s[]){
		//Nom Taula, Tipus, Nom Constraint, Constraint, Constraint alternativa
		nomTabla = s[0];
		tipoConstraint = s[1];
		nomConstraint = s[2];
		cosConstraint = s[3];
		altConstraint = s[4];
	}
}
```

ANNEX IX
CODI VectorConstraints.java

```
import java.util.*;
/**Constructor del vector de  restriccions, s'utilitza
* per a ordenar les restriccions que afectin a una taula,
* si s'escau el programa principal XMI2DBMS donarà a escollir
* entre les alternatives que continguin els elements constrains */
public class VectorConstraints extends Vector {
	/**Mètode per ordenar el vector de restriccions
	 * per taula a que afecten en primera instància i
	 * tipus de restrcció finalment
	 */
	public void ordenar(){
		/**Atribut utilitzat per ordenar el vector de restriccions
		 */
		boolean cambiado = true;
		while (cambiado){
			cambiado = false;
			for (int i=0; i < (this.size()-1); i++){
				if (((Constraint) this.elementAt(i)).nomTabla.compareTo(((Constraint) this.elementAt(i+1)).nomTabla) > 0){
					cambiado = true;
					Constraint c_i = (Constraint) this.elementAt(i);
					Constraint c_i1 = (Constraint) this.elementAt(i+1);
					this.removeElementAt(i);
					this.removeElementAt(i);
					this.insertElementAt(c_i1, i);
					this.insertElementAt(c_i, i+1);
				}

				if ((((Constraint) this.elementAt(i)).nomTabla.compareTo(((Constraint) this.elementAt(i+1)).nomTabla) == 0) &&
				(((Constraint) this.elementAt(i)).tipoConstraint.compareTo(((Constraint) this.elementAt(i+1)).tipoConstraint) > 0)) {
					cambiado = true;
					Constraint c_i = (Constraint) this.elementAt(i);
					Constraint c_i1 = (Constraint) this.elementAt(i+1);
					this.removeElementAt(i);
					this.removeElementAt(i);
					this.insertElementAt(c_i1, i);
					this.insertElementAt(c_i, i+1);
				}
			}
```

```
                }
        }
}
```

ANNEX X
CODI ConectorBaseDades.java

```
import java.sql.*;
import java.io.*;
/**Classe que fa la connexió al SGBD per a enviar_li les comandes
 */
public class ConectorBaseDades{
        /**Tipus de connexió pot ser la connexió vía ODBC al
         * gestor de base de dades o connexió a fitxer
         */
        String tipusConexio;
        /**Connexió a la base de dades
         */
        Connection c;
        /**Atribut utilitztat per a l'execució de comandes
         */
        Statement st;
        /**Atribut  NO utilitztat per a la recepció de recordsets
         */
        ResultSet rs;
        /**Flux de dades per a la seva utilització si es fa la sortida a fitxer
         */
        FileOutputStream fos=null;

        /**Mètode que realitza la connexió vía ODBC a la Base de dades
         * o la sortida a fitxer depenent del Tipus
         */
        public ConectorBaseDades(String Tipus, String ODBC, String user, String password){
                tipusConexio = Tipus;
                try{
                        if (tipusConexio.equalsIgnoreCase("F")){
                                fos = new FileOutputStream(ODBC);
                        }
                        else{
                                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                                c = DriverManager.getConnection("jdbc:odbc:" + ODBC, user, password);
                                st = c.createStatement();
```

```
                }
        }
        catch (Exception e){
                System.out.println("Error a la connexió a la Base de Dades");
                e.printStackTrace();
        }
}

/**Mètode que executa les comandes de creació de restriccions
 * al SGBD o escriu les comandes a fitxer depenent del Tipus
 */
public void executarComanda (String comanda){
        try{
                if (tipusConexio.equalsIgnoreCase("F")){
                        byte b[] = comanda.getBytes();
                        fos.write(b);
                        fos.write(13);
                        fos.write(10);
                }
                else{
                        st.execute(comanda);
                }
        }
        catch (Exception e){
                System.out.println("Error a la comanda " + comanda);
                e.printStackTrace();
        }
}

/**Mètode que finalitza la connexió al SGBD
 * o tanca el flux de dades a fitxer depenent del Tipus
 */
public void finalitzarConexio (){
        try{
                if (tipusConexio.equalsIgnoreCase("F")){
                        fos.close();
                }
                else{
                        st.close();
                        c.close();
                }
```

```
                }
                catch (Exception e){
                        System.out.println("Error al tancar");
                        e.printStackTrace();
                }
        }
}
```

ANNEX XI
CODI GeneradorDDL.java

```
/**Classe que rep els elements necessaris per a formar una restricció
 * en SQL de la gramàtica que ha reconegut la restricció i el nom del
 * gestor de bases de dades per a retornar la restricció ne format SQL
 * propi del gestor de dades escollit i si s'escau una sentència alternativa
 * per a poder escollir desde la classe principal XMI2DBMS
 */
public class GeneradorDDL{
	String SGBD=null;

	/**Mètode que controla que tenim el gestor de bases de dades
	 * definit a la nostra classe per a poder retornar el sql
	 * corresponent.<BR>
	 * Per a afegir un nou gestor a l'aplicació cal incloure'l
	 * aquí per que no retorni la execepció SGBD no suportat.<BR>
	 * També cal definir el seu tractament per als cassos definits,
	 * de moment són:
	 * <UL>
	 * <LI> generarDDLPK,
	 * <LI> generarDDLCheck,
	 * <LI> generarDDLForeignKey.
	 * </UL>
	 */

	public GeneradorDDL(String pSGBD) throws Exception {
		if ((pSGBD.equalsIgnoreCase("ORACLE") == false) &&
		    (pSGBD.equalsIgnoreCase("INFORMIX") == false) &&
		    (pSGBD.equalsIgnoreCase("ACCESS") == false))
			throw new Exception("SGBD no soportado");
		else
			SGBD = pSGBD;
	}

	/**Mètode que analitza el tipus de restricció reconeguda per la
	 * classse gramatica corresponent per a cridar segons el tipus
	 * als mètodes: <BR>
	 * generarDDLPK, <BR>
	 * generarDDLCheck,<BR>
	 * generarDDLForeignKey.<BR>
```

```
	 * Si ampliessim la funcionalitat de l'aplicació
	 * amb d'altre típus de restriccions  hauriem d'afegir
	 * el tipus nou retornat per a enviar-ho al nou mètode
	 * generarDDLXxxx.  <BR>
	 * Analitza el contingut de  primitivas[0][0]que conté el tipus
	 * de restricció, actualment pot ser:<UL>
	 * <LI>"PRIMARY KEY"
	 * <LI>"CHECK SIMPLE"
	 * <LI>"FOREIGN KEY"
	 * </UL>
	 */
	public String[] generarDDL(String[][] primitivas) throws Exception {
		if (primitivas[0][0].equalsIgnoreCase ("PRIMARY KEY")){
			return generarDDLPK(primitivas);
		}
		else if(primitivas[0][0].equalsIgnoreCase ("CHECK SIMPLE")){
			return generarDDLCheck(primitivas);
		}
		else if(primitivas[0][0].equalsIgnoreCase ("FOREIGN KEY")){
			return generarDDLForeignKey(primitivas);
		}
		else{
			throw new Exception("Gramatica no suportada");
		}

	}

	/**Mètode que rep a l'estructura primitivas[][]
	 * l'informació enviada per la gramàtica que ha reconegut
	 * la clau primària.  Al nostre cas utilitzarem el paràmetres
	 * d'entrada :<UL>
	 * <LI>primitivas[1][0]= nom de la taula on afegirem la clau primària
	 * o alternativa.
	 * <LI>primitivas[2][0]= nom de la clau primària
	 * o de la restricció alternativa
	 * <LI>primitivas[3][0]= nom dels camps de la clau separats per comes.
	 * </UL>
	 * ês convenient revisar si el SGBD que es vol afegir accpeta alguna
	 * sintàxi sql que estigui definida, per exemple ORACLE i ACCESS accepten
	 * la mateixa sintàxi:<BR>
	 * if ((SGBD.equalsIgnoreCase("ORACLE"))||(SGBD.equalsIgnoreCase("ACCESS"))){<BR>
```

```
 * Amb coincidència de sentències
 * podríem afegir un nou gestor de la següent forma:<BR>
 * if ((SGBD.equalsIgnoreCase("ORACLE"))||
 *     (SGBD.equalsIgnoreCase("ACCESS"))||
 *     (SGBD.equalsIgnoreCase("NewSGBD"))){<BR>
 * El retorn del mètode és :
 * @return <UL><LI>retorno[0] = nom taula,
 * <LI>retorno[1] = tipus de restricció,
 * <LI>retorno[2] = sentència SQL del SGBD de PRIMARY KEY,
 * <LI>retorno[4] = sentència SQL del SGBD de UNIQUE;</UL>
 */
public String[] generarDDLPK(String[][] primitivas){
        String retorno[] = new String[5];
        retorno[0] = primitivas[1][0];
        retorno[1] = primitivas[0][0];
        retorno[2] = primitivas[2][0];
        //ALTER TABLE TABLE_1 ADD CONSTRAINT NAME_CONSTRAINT PRIMARY KEY
        //(CAMP_1,CAMP_2, CAMP_3);
        //ALTER TABLE TABLE_1 ADD CONSTRAINT NAME_CONSTRAINT UNIQUE
        //(CAMP_1,CAMP_2, CAMP_3);
        if ((SGBD.equalsIgnoreCase("ORACLE"))||(SGBD.equalsIgnoreCase("ACCESS"))){
                retorno[3] = "ALTER TABLE " + primitivas[1][0] + " ADD CONSTRAINT " + primitivas[2][0] + " PRIMARY KEY (";
                retorno[4] = "ALTER TABLE " + primitivas[1][0] + " ADD CONSTRAINT " + primitivas[2][0] + " UNIQUE (";
                String cadena = "";
                for (int i=0; i < primitivas[3].length; i++){
                        cadena = cadena + primitivas[3][i] ;
                        if (i< (primitivas[3].length-1)) cadena = cadena + " , ";
                }
                retorno[3] = retorno[3] + cadena + " ); ";
                retorno[4] = retorno[4] + cadena + " ); ";
        }
        //ALTER TABLE TABLE_1 ADD CONSTRAINT PRIMARY KEY
        //(CAMP_1,CAMP_2, CAMP_3) CONSTRAINT NAME_CONSTRAINT;
        //ALTER TABLE TABLE_1 ADD CONSTRAINT UNIQUE
        //(CAMP_1,CAMP_2, CAMP_3) CONSTRAINT NAME_CONSTRAINT;
        if (SGBD.equalsIgnoreCase("INFORMIX")){
                retorno[3] = "ALTER TABLE " + primitivas[1][0] + " ADD CONSTRAINT PRIMARY KEY (";
                retorno[4] = "ALTER TABLE " + primitivas[1][0] + " ADD CONSTRAINT UNIQUE (";
                String cadena = "";
                for (int i=0; i < primitivas[3].length; i++){
                        cadena = cadena + primitivas[3][i] ;
```

```
                        if (i< (primitivas[3].length-1)) cadena = cadena + " , ";
                    }
                    retorno[3] = retorno[3] + cadena + " ) CONSTRAINT "+ primitivas[2][0] +" ; " ;
                    retorno[4] = retorno[4] + cadena + " ) CONSTRAINT "+ primitivas[2][0] +" ; " ;
            }

            return retorno;
    }

    /**Aquest mètode rep els elements necessaris per a formar
     * la sentència sql del check corresponent de la gramàtica que
     * ha reconegut la restricció.<BR>
     * El mètode treballa amb els paràmetres d'entrada:<UL>
     * <LI>  primitivas[1][0]= nom de la taula on afegirem la clau primària o alternativa
     * <LI>  primitivas[2][0]= nom de la restricció
     * <LI> primitivas[3][0]= contingut de la restricció
     * </UL>
     * @return <UL><LI> retorno[0] = nom taula,
     * <LI> retorno[1] = tipus de restricció,
     * <LI> retorno[2] = sentència SQL del SGBD de CHECK SIMPLE,
     * <LI> retorno[4] = "" no utilitzat;
     * </UL>
     */
    public String[] generarDDLCheck(String[][] primitivas)throws Exception{
            String retorno[] = new String[5];
            retorno[0] = primitivas[1][0];
            retorno[1] = primitivas[0][0];
            retorno[2] = primitivas[2][0];
            retorno[4] = "";
            //ALTER TABLE TABLE1 ADD CONSTRAINT NAME_CHECK
            //CHECK((CAMP_1<50 AND CAMP_2<>'CAP') OR (CAMP_1>50 AND CAMP_2='CAP')...);
            if (SGBD.equalsIgnoreCase("ORACLE")){
                    retorno[3] = "ALTER TABLE " + primitivas[1][0] + " ADD CONSTRAINT " + primitivas[2][0] + " CHECK (";
                    String cadena = "";
                            for (int i=0; i < primitivas[3].length; i++){
                                    cadena = cadena + primitivas[3][i] ;

                            }
                    retorno[3] = retorno[3] + cadena + " ); ";

                    }
```

```
		//ALTER TABLE TABLE1 ADD CONSTRAINT
		//CHECK((CAMP_1<50 AND CAMP_2<>'CAP') OR (CAMP_1>50 AND CAMP_2='CAP')...)
		//CONSTRAINT NAME_CHECK;
		if ((SGBD.equalsIgnoreCase("INFORMIX"))){
			retorno[3] = "ALTER TABLE " + primitivas[1][0] + " ADD CONSTRAINT CHECK (";
			String cadena = "";
				for (int i=0; i < primitivas[3].length; i++){
					cadena = cadena + primitivas[3][i] ;

				}
			retorno[3] = retorno[3] + cadena + " ) CONSTRAINT " + primitivas[2][0]+" ;";

			}
		if (SGBD.equalsIgnoreCase("ACCESS")){
			throw new Exception("ACCESS NO ACCEPTA CHECKS FORA DEL SEU ENTORN GRAFIC (VALIDATION RULES)");
		}
		return retorno;
	}

	/**Aquest mètode rep els elements necessaris per a formar
	 * la sentència sql de la clau forana corresponent de la gramàtica que
	 * ha reconegut la restricció.<BR>
	 * El mètode treballa amb els paràmetres d'entrada:<UL>
	 * <LI> primitivas[1][0]= nom de la taula on afegirem la clau forana
	 * <LI> primitivas[2][0]= nom de la restricció
	 * <LI> primitivas[3][0]= nom de la taula referida "referred table"
	 * <LI> primitivas[3][1]= nom dels camps que composen la clau forana
	 * pertanyen a la taula 1, separats per comes','
	 * <LI> primitivas[3][2]= nom dels camps referits per la clau forana
	 * pertanyen a la taula 2, separats per comes','  </UL>
	 * @return <UL><LI> retorno[0] = nom taula,
	 * <LI> retorno[1] = tipus de restricció,
	 * <LI> retorno[2] = sentència SQL del SGBD de FOREIGN KEY,
	 * <LI> retorno[4] = "" no utilitzat;
	 * </UL>
	 */
	public String[] generarDDLForeignKey(String[][] primitivas){
		String retorno[] = new String[5];
		retorno[0] = primitivas[1][0];
		retorno[1] = primitivas[0][0];
		retorno[2] = primitivas[2][0];
```

```
		retorno[4] = "";

		//ALTER TABLE TABLE_1 ADD CONSTRAINT NAME_FOREIGN_KEY
		//FOREIGN KEY(CAMP_1,CAMP_2,...)
		//REFERENCES TABLE2(CAMP2_1,CAMP2_2...);
		if ((SGBD.equalsIgnoreCase("ORACLE")) || (SGBD.equalsIgnoreCase("ACCESS"))){
			retorno[3] = "ALTER TABLE " + primitivas[1][0] + " ADD CONSTRAINT " + primitivas[2][0] + " FOREIGN KEY (";
			String cadena = "";
			cadena= primitivas[3][1]+") REFERENCES "+primitivas[3][0]+" ("+primitivas[3][2];
			retorno[3] = retorno[3] + cadena + " ); ";

			}

		//ALTER TABLE TABLE_1 ADD CONSTRAINT
		//FOREIGN KEY(CAMP_1,CAMP_2,...)
		//REFERENCES TABLE2(CAMP2_1,CAMP2_2...)CONSTRAINT NAME_FOREIGN_KEY;
		if (SGBD.equalsIgnoreCase("INFORMIX")){
			retorno[3] = "ALTER TABLE " + primitivas[1][0] + " ADD CONSTRAINT FOREIGN KEY (";
			String cadena = "";
			cadena= primitivas[3][1]+") REFERENCES "+primitivas[3][0]+" ("+primitivas[3][2];
			retorno[3] = retorno[3] + cadena + " ) CONSTRAINT "+ primitivas[2][0] +" ; " ;

			}
		return retorno;
		}


}
```

ANNEX XII
Arxius presentats

El projecte s'ha presentat si no s'indica el contrari com a tres fitxers zip:

1 **saleixandre_tfc_fitxers.zip**, que conté l'estructura amb directoris a la que trobem:
Els fitxers .Java al subdirectori uoc\TFC\tfc\java.
La documentació generada per javadoc al subdirectori uoc\TFC\tfc\html.
El joc de proves al subdirectori uoc\TFC\tfc\fitxers_prova on trobem:

- 1.1 La sortida en format txt pels SGBD suportats sortidaOracle.txt sortidaInformix.txt sortidaAccess.txt .
- 1.2 El fitxer generat per POSEIDON pr6.zargo, de fet es pot veure amb un extractor zip però cal l'eina per visualitzar el diagrama
- 1.3 El fitxer passat a XMI2DBMS pr6.xmi que està contingut a pr6.zargo
- 1.4 El fitxer ExecXMI2DBMS_INFORMIX.zip conté les copies de pantalla d'exemple fet a l'entorn INFORMIX sobre el entorn descrit a l'annex I, de fet les captures de pantalla d'informix són només un exmple de com s'ha vist afectada una taula per la nostra aplicació, També inclou la copia de pantalla del prompt per línia de comandes.

2 **saleixandre_TFC.zip** que conté la memòria.

3 **saleixandre_tfc_pps.zip** que conté la presentació virtual.

4 saxon-aelfred.jar Fitxer aliè utilitzat com parser SAX, per utilitzar aquesta aplicació l'hem d'incloure al path: "C:\ruta del motor java"\jre\lib\ext

ANNEX XIII
Comentaris de les proves

La nostra aplicació s'ha vist afectada, degut a l'abast d'aquest projecte, per l'entorn de probes.

Tal com ja hem comentat a la memòria la impossibilitat temporal per a utilitzar diferents eines CASE, ampliar els gestors suportats o incorporar més gramàtiques tractades ens obliga a fer certes observacions:

- Els gestors de bases de dades han estat aquests ja que eren els que tinc més fàcilment accessibles per poder utilitzar el menor temps possible amb proves de connexió i pas de jocs XMI.

- ACCESS no és un bon exemple de gestor ja que no accepta restriccions tipus check simple i l'hem tractat llençant una excepció explicativa, bàsicament ens diu que només accepta aquestes restriccions des del seu entorn gràfic o des d'eines propietàries com VisualBasic for Applications que li poden passar els checks en el format que entén (validation rules). El programa si que tractarà les altres restriccions trobades.

- POSEIDON a la versió utilitzada a l'entorn de proves **no valida la consistència de tipus d'integritat referencial**, es a dir podem definir una clau forana d'un camp integer que tingui un camp referit tipus data. La nostra aplicació tampoc fa aquesta comprovació, **per definició hem de suposar que partim d'una especificació correcta**, tal i com la que adjuntem pr6.xmi.

- POSEIDON a la versió utilitzada no detecta la definició de noms d'invariants repetits per part de l'usuari, nosaltres tampoc, es podria fer utilitzant l'ordenació de les restriccions, però no està fet a aquesta versió,

- Hem introduit al joc de proves una restricció que no s'ha de reconèixer, de fet està definida amb nom repetit, no ens donarà cap error a l'execució del programa ja que no és reconeguda com a clau primària o alternativa vàlida, les atres restriccions que podem trobar a pr6.xmi estan ben definides i són exmples de restriccions reconegudes pel joc de gramàtiques utilitzat.

Sentència no reconeguda:

```
context Taula1 inv pkmultT1 :
 Taula1 . allInstances -> forAll ( c1 , c2 | c1 <> c2 implies c1 . pk1 <> c2 . pk2 or c2 . pk2 <> c1 . pk2 or c1 . pk3 <> c2 . pk3 or c2 . pk4 <> c1 . pk4 )
```

Sentència reconeguda:

```
context Taula1 inv pkmultT1 :
Taula1 . allInstances -> forAll ( c1 , c2 | c1 <> c2 implies c1 . pk1 <> c2 . pk1 or c2 . pk2 <> c1 . pk2 or c1 . pk3 <> c2 . pk3 or c2 . pk4 <> c1 . pk4 )
```