TESINA DE LICENCIATURA

TITULO: Transformaciones entre modelos de bases de datos en el contexto de MDA

AUTORES: Nector Martín Baez

DIRECTOR: Prof. Dra Claudia Fabiana Pons

CODIRECTOR: -

CARRERA: Licenciatura en Informática - Plan 90

Resumen

La comunidad informática ha buscado siempre mejorar la calidad de los procesos de desarrollo. Ha tomado fuerza con el tiempo el concepto de desarrollo dirigido por modelos(MDD) que propone mejorar la calidad con la idea de que el hilo conductor del desarrollo sean los modelos y las transformaciones entre ellos. En respuesta a esto el Object Management Group a través de su iniciativa denominada Model Driven Architecture(MDA) ha elaborado una serie de estándares para impulsar el desarrollo dirigido por modelos. También ha tomado fuerza la idea del desarrollo de lenguajes espcificos de dominio(DSL) para llevar a cabo el desarrollo dirigido por modelos. Existen organizaciones que implementan estándares de la OMG para dar soporte al desarrollo basado en MDA y para la creación de DSLs. En el área de base de datos se utilizan modelos desde hace mucho tiempo su diseño y construcción. Estos modelos son considerados estándares de facto. Por lo tanto parece ser este un área propicia para aplicar el desarrollo dirigido por modelos. Demostramos aquí como llevar a cabo un desarrollo guiado por modelos en el área del diseño de base de datos implementando una propuesta para desarrollar un datawarehouse temporal utilizando las herramientas de modelado de la plataforma Eclipse.

Líneas de Investigación

Desarrollo dirigido por modelos
Model Driven Architecture
Domain Specific Languaje.
Diseño de Base de Datos.
Datawrehousing.

Trabajos Realizados

Se ha desarrollado un toolkit DSL utilizando MDD para dar a su vez soporte para el desarrollo basado en MDD de datawarehouses temporales.

Conclusiones

El desarrollo dirigido por modelos nos permite mejorar la calidad del desarrollo. A través de los lineamientos de la iniciativa MDA, mediante la construcción de DSLs y utilizando las herramientas que provee la plataforma eclipse podemos construir herramientas de modelado en áreas donde los modelos ya son utilizados. Este es el caso del diseño de base de datos donde los modelos son utilizados y ampliamente adoptados. La manera optima de mejorar la productividad en estas áreas es a través de la construcción de toolkits utilizando MDA y DSLs para dar soporte a su vez al desarrollo guiado por modelos en estas áreas.

Trabajos Futuros

Compatibilidad de los modelos de este trabajo con CWM, implementar aspectos del diseño de base de datos no considerados aqui(por ejemplo, jerarquías en modelos de ER), derivación atomática de de ETLs, derivación de consultas SQL. Extensionalidad para derivar código para cualquier RDBMS.

Fecha de la presentación: Febrero de 2010

Índice de contenido

Agradecimientos....5
Introducción....6
Capítulo 1....8
1. Conceptos generales....8
1.1 Desarrollo dirigido por modelos....8
1.1.1 Model Driven Development(MDD) 9
1.1.2 El Object Management Group(OMG) [1] 9
1.1.3 Model Driven Architecture(MDA)[2]....9
1.1.4 Independencia de la plataforma....9
1.1.5 Puntos de vista de MDA....9
1.1.6 Arquitectura de cuatro capas propuesta por la OMG[1]....11
1.1.7 El lenguaje de modelado MOF[5] de la OMG[1] 11
1.1.8 Modelado Específico del Dominio (DSM y DSLs)[3]....11
1.1.9 Confusión entre MDA[2] y DSM[3]....12
1.2 Diseño de Bases de Datos Relacionales....12
1.2.1 Los modelos y el diseño de bases de datos....13
1.2.2 Modelado de Bases de datos con MDA[2]....14
1.2.3 Datawarehousing....15
1.2.4 Diseño de datawarehousing....16
1.2.4.1 Diseño conceptual - Modelado Multidimensional....16
1.2.5 Modelado de datawarehousing con MDA[2]....17
1.3 Propuesta para modelar datawarehousing con MDA[14]....18
1.3.1 Aplicando MDA al diseño de un datawarehouse temporal[14]....18
Capítulo 2 25
2 Base Conceptual y herramientas 25
2.1 Modelos de propósito general 25
2.1.2 ¿Lenguajes de propósito general? ¿DSL existente? ¿DSL propio?....25
2.1.2 Visión del desarrollo dirigido por modelos utilizada....26
2.1.3 Common Warehouse Metamodel (CWM)[17] 27
2.2 Un toolkit DSL....27
2.3 Soporte de la plataforma Eclipse para el desarrollo de DSLs y basado en MDA....28
2.4 Implementación de CWM[17] con EMP(ECWM) 32
2.4.1 Un primer intento 32
2.4.2 Inconvenientes causados de implementar ECWM....32
2.4.3 Conclusión sobre el uso de CWM....34
Capítulo 3 35
3 Marco general de Desarrollo....35
3.1 Modelos de referencia 35
3.2 Implementación de modelos....35
3.3 Indice de modelos y transformaciones implementadas....36
3.3.1 Modelo Entidad-Relación(Capítulo 4) 36
3.3.2 Lenguaje de reestructuración de Entidad-Relación(Capitulo 4) 36
3.3.3 Grafo de Atributos (Capítulo 5) 37
3.2.4 Modelo Multidimensional temporal(Capítulo 6) 37
3.2.5 Modelo relacional(Capítulo 7) 38
3.3 Implementación de transformaciones 38
3.3.1 Reestructuración ER(Capítulo 4) 38
3.3.2 ER (adaptado) al Grafo de atributos(Capítulo 5) 39
3.3.3 Grafo de atributos al modelo multidimensional(Capítulo 6) 39
3.3.4 Modelo multidimensional al modelo relacional(Capítulo 7) 39
3.3.5 Modelo relacional a código(Capítulo 8) 39
3.4 Extensiones desarrolladas....39
3.4.1 Validaciones en OCL y Java....39
3.4.2 Equivalencia entre notaciones basadas en Ecore y xText 41
3.4.3 Ejecución de transformaciones ATL programaticamente....41
3.4.4 Launchers para lanzar transformaciones....43
Capítulo 4....44
4 Modelado de Entidad-Relación....44
4.1 Modelos de Entidad-Relación - Una breve definición....44
4.2 Diagramas de Entidad-Relación....44
4.3 DSL de Entidad-Relación - construcción 45
4.3.1 Metamodelo entidad relación o modelo de datos simple(mds)....45
4.3.2 Notaciones 45
4.4 DSL de Entidad-Relación - uso 50

4.4.1 Notaciones 51
Capítulo 5 54
5 Lenguajes de reestructuración....54
5.1 Lenguaje de reestructuración del modelo de Entidad-Relación....54
5.2 DSLs de reestructuración - construcción....55
5.3 DSL textual de reestructuraciones - construcción 55
5.3.1 Notaciones 55
5.4 DSL textual de reestructuraciones - uso 57
5.4.1 Notaciones 57
5.5 DSL visual de reestructuraciones - construcción 58
5.5.1 Metamodelo de reestructuración - rmds....58
5.5.2 Notaciones 58
5.6 DSL visual de reestructuraciones - uso 61
5.6.1 Notaciones 62
5.7 DSL textual y visual – correspondencia....63
5.7.1 Transformación basada en recursos EMF....63
5.7.2 Implementación de la transformación 63
5.7.3 Uso de la transformación 64
5.7.4 Ejecución de la transformación a través de una acción contextual 64
5.8 Reestructuraciones para generar el datawarehouse....64
5.9 Reestructuraciones automáticas....65
5.10 Implementación de las transformaciones y uso....65
5.10.1 Transformación automática basada en ATL para hechos de interés 66
5.10.2 Uso de la transformación para hechos de interés....66
5.10.3 Implementación de la transformación automática basada en ATL para aspectos temporales....67
5.10.4 Uso de la transformación por parte del diseñador....68
5.11 Ejecución de las transformaciones 69
Capítulo 6....70
6 Grafo de Atributo....70
6.1 Grafo de atributos - Una breve definición....70
6.2 DSL definido para el grafo de atributos - construcción 71
6.2.1 Metamodelo del grafo de atributos(mgragh)....71
6.2.2 Notaciones 72
6.3 DSL del grafo de atributos - uso 73
6.4 Modelo de Entidad-Relación al Grafo de atributos 74
6.4.1 Transformaciones automáticas 74
6.4.2 Implementación de la transformación....74
6.4.3 Uso de la transformación 75
6.4.4 Ejecución de la transformación a través del launcher....76
Capítulo 7....77
7 Modelado Multidimensional....77
7.1 Modelo Multidimensional para un Datawarehouse - Breve definición....77
7.2 Modelo Multidimensional....77
7.2.1 Esquemas y tablas....78
7.2.2 Tablas DW....78
7.2.3 Esquemas DW 78
7.3 Finalidad de un Datawarehouse 79
7.4 La Dimensión Tiempo 79
7.5 DSL Multidimensional – construcción....79
7.5.1 Metamodelo multidimensional temporal(mmt)....79
7.5.2 Notaciones definidas....80
7.6 DSL Multidimensional - uso 83
7.6.1 Notaciones del modelo multidimensional....84
7.7 Grafo de atributos al Modelo Multidimensional....86
7.7.1 Transformaciones automáticas 86
7.7.2 Implementación de la transformación....87
7.7.3 Uso de la transformación 88
7.7.4 Ejecución de la transformación 88
7.7.5 Ejecución de la transformación a través del launcher....89
Capítulo 8....90
8.1 Modelado relacional....90
8.1 Modelo relacional – Una breve definición....90
8.2 Base de datos relacional - Una breve definición 90
8.3 Diagramas relacionales....90
8.4 DSL para el modelo relacional - construcción 91
8.4.1 Metamodelo relacional(mrel)....91

8.4.2 Notaciones definidas....92
8.5 DSL para el modelo relacional – uso....96
8.5.1 Notaciones del modelo relacional....97
8.6 Transformaciones del Modelo Multidimensional a Modelo relacional 100
8.6.1 Transformaciones automáticas 100
8.6.2 Implementación de la transformación....100
8.6.3 Uso de la transformación 101
8.6.4 Ejecución de la transformación 101
8.6.5 Ejecución de la transformación a través del launcher....101
8.7 Generación de código SQL a partir del modelo lógico....102
8.7.1 Construcción de generador de código....102
Mediante un wizards el usuario selecciona el archivo que contiene la notación abstracta relacional y por medio de el mismo obtiene el código sql de creación de tablas que contiene el modelo relacional seleccionado....103
Conclusiones 105
Trabajos relacionados....106
Trabajos Futuros....106
REFERENCIAS BIBLIOGRAFICAS 108

Agradecimientos

A mis padres por darme la oportunidad y estimularme a estudiar. A mis hermanos por estar conmigo incondicionalmente siempre. A mis tios y abuelos. A mi compañera Luji por apoyarme a cerrar esta y otras etapas de mi vida. Al pequeño Benjamín que esta por llegar, quién me ha dado en envión de ánimo que necesitaba. A la profesora Claudia Pons por la motivación y sus consejos para terminar este trabajo. A Carlos Neil por sus consejos y el tiempo que dedicó a conversar conmigo sobre este trabajo y por permitirme valerme algunas de sus ideas. A la profesora Lia Molinari y mis compañeros de la cátedras de ISO y SO de la Facultad de Informática. A mis amigos de la infancia con quienes religiosamente cada fin de semana en General Belgrano compartimos momentos desde que tengo memoria. A mis ex-compañeros de trabajo del IPS. Al Lifia por permitirme ser parte del maravilloso grupo de trabajo al que pertenezco y por permitirme crecer profesionalmente. A los amigos que me dió de la Facultad de Informática con quienes preparamos parciales, finales y realizamos trabajitos. A la Facultad de Informática, un lugar donde conocí gente buena sin excepción. Y a todos los que alguna vez intentaron darme ánimo para terminar mi carrera.

Introducción

El desarrollo de software es un proceso caro, riesgoso y complicado. En el último tiempo la comunidad de la ciencia informática que se ocupa del desarrollo de software, ha puesto su esfuerzo en mitigar los aspectos anteriores mejorando tanto la calidad del proceso de desarrollo como la del producto obtenido. Es necesario aclarar que el desarrollo de software abarca un amplio espectro de dominios de aplicación, como por ejemplo sistemas con lógicas de negocios de alta complejidad, sistemas embebidos, sistemas operativos, etc. Debido al intento de mejorar la calidad han surgido varias formas de ver y de especificar los problemas de la realidad y varias propuestas de como llevar la visión del problema a una solución concreta. En informática esto significa obtener un sistema informático que resuelva el problema analizado. Esta manera de resolver problemas se viene analizando desde hace hace mucho tiempo. Siempre se ha puesto un énfasis en el análisis correcto del problema para poder tener un modelo del mismo y luego una solución acorde al modelo del problema planteado. Es por eso que han surgido tecnologías de modelado ampliamente utilizadas en la industria, como por ejemplo técnicas de diseño orientado a objetos, diseño de base de datos, etc. A pesar de esto el paso desde nuestro modelo a nuestra solución ha sido basado en técnicas informales en donde un desarrollador interpreta un modelo realizado quizás por el mismo o por otra persona y lo implementa en algún lenguaje de programación; siempre acorde a su interpretación de los modelos. Esta manera de desarrollar soluciones hace que cada vez que se cambia algo de la versión ejecutable se cambie el modelo para que no existan inconsistencias y viceversa. Pero principalmente existe una gran brecha que se se produce entre lo que el modelo expresa y lo que el desarrollador interpreta. El desarrollo guiado por modelos ataca la brecha de informalidad que se produce entre el modelado de la solución a un problema y su codificación. El foco esta puesto en modelar y en derivar el código a partir de nuestros modelos automáticamente. Existen además propuestas tan ambiciosas como estandarizar metamodelos para la industria para que sean usados en el diseño de soluciones, destacándose las incitativas propuestas por el Object Management Group(OMG) [1] a través de la iniciativa Model Driven Architecture(MDA)[2] y las aproximaciones basadas en crear lenguajes específicos del dominio para resolver problemas[3]. Siempre se estimula a utilizar modelos y transformaciones para obtener automaticamente la solución a los problemas. Es muy habitual en la industria informática el uso diagramas de bases de datos, uml, redes de petri, tablas de decisión, diagramas de workflows ,etc. Los dominios donde se utilizan estos modelos al estar muy arraigados a la industria informática parecieran ser un camino natural para la aplicaron del desarrollo dirigido por modelos. El gran paso seria que tales modelos sean los que conduzcan el desarrollo, que los modelos sean los ciudadanos de primera clase y que el código sea consecuencia automática de los modelos.

Existen áreas donde el desarrollo dirigido por modelos es mas complicado de llevar a la práctica por el poco uso (inexistente a veces) de los modelos. Por ejemplo en el área de Sistemas Operativos, no existen artefactos ni estándares de modelos para plasmar el diseño del scheduler de CPU, sus políticas de asignación de recursos a los procesos, etc .

Es fácil inventar un modelo gráfico para describir un workflow pero no es así para graficar un algoritmo de scheduling para administrar la CPU en un Sistema Operativo.
Debemos entonces poner el énfasis en aquellas áreas en donde el modelado es adoptado naturalmente, donde ya existen modelos que estándares o modelos de facto. El único paso que queda aquí es la formalización en base al metamodelado de los modelos existentes y en base a esto su utilización ya no solo a modo documentativo sino como la base para la generación del producto final. Un caso típico de esto es el área de base de datos, donde existen formalismos y modelos ampliamente adoptados y arraigados en la cultura del profesional de informática. Los modelos de Entidad-Relación, y esquemas Relacionales son conceptos muy conocidos y utilizados naturalmente en la industria informática. En el presente trabajo se propone adoptar esta filosofía para mostrar como dar soporte al desarrollo guiado por modelos en esta área de la informática. Se describe como se implementó un proceso de generación de un datawarehouse temporal adoptando la visión del desarrollo dirigido por modelos para construir una herramienta. Dicha herramienta sirve para el desarrollo guiado por modelos de datawarehousing con aspectos temporales.

Capítulo 1

En este capítulo se presentan de manera general los conceptos en los que se basa este trabajo. Nos basamos en la visión del desarrollo basado en modelos de la OMG, en la visión del desarrollo de DSLs que se utiliza en la plataforma Eclipse ,en conceptos generales sobre modelos de bases de datos y en una propuesta teórica para aplicar MDA en el diseño de base de datos.

1. Conceptos generales

Es necesario aclarar que los conceptos generales tomados como base para este trabajo a veces presentan puntos de indefinición pero que son necesarios a la hora de realizar un desarrollo. Esto ocurre debido a que las propuestas en las que se basa parte de este trabajo están basadas en algunos formalismos y en algunos conceptos definidos de manera informal y en lenguaje natural. En este trabajo por ser solo un ejemplo de caso de uso de aplicación de MDA[2] se adoptan decisiones minimalista para aquellos aspectos no especificado en dichas propuestas. Esto es para no hacer mas complejo el desarrollo. También se presentan puntos contradictorios entre la visión de la OMG[1] del desarrollo basado en modelos(MDA) y la visión de quienes desarrollan las herramientas utilizadas para el desarrollo de este trabajo. Esto hace que nuestro desarrollo deba adoptar ambas visiones y en algunos casos definirse por alguna de ellas.

1.1 Desarrollo dirigido por modelos

Se presentan aquí breves definiciones y conceptos sobre el desarrollo dirigido por modelos, la visión de la OMG[1]. Para mas detalles y definiciones formales se recomienda la lectura de las guías de la OMG[1] que figuran en las referencias del presente trabajo.

1.1.1 Model Driven Development(MDD)

La ingeniería de software establece que el problema de construir software debe ser encarado de la misma forma en que los ingenieros construyen otros sistemas complejos, como puentes, edificios, barcos y aviones. La idea básica consiste en observar el sistema de software a construir como un producto complejo y a su proceso de construcción como un trabajo ingenieril. Es decir, un proceso planificado basado en metodologías formales apoyadas por el uso de herramientas. La construcción de un sistema de software debe ser precedida por la construcción de un modelo, tal como se realiza en otros sistemas ingenieriles. El modelo del sistema es una conceptualización del dominio del problema y de su solución. El modelo se enfoca sobre el mundo real: identificando, clasificando y

abstrayendo los elementos que constituyen el problema y organizándolos en una estructura formal.
La abstracción es una de las principales técnicas con la que la mente humana se enfrenta a la complejidad. Ocultando lo que es irrelevante, un sistema complejo se puede reducir a algo comprensible y manejable. Cuando se trata de software, es sumamente útil abstraerse de los detalles tecnológicos de implementación y tratar con los conceptos del dominio de la forma más directa posible. De esta forma, el modelo de un sistema provee un medio de comunicación y negociación entre usuarios, analistas y desarrolladores que oculta o minimiza los aspectos[4].

1.1.2 El Object Management Group(OMG) [1]

La OMG fue formada para ayudar a reducir la complejidad, bajar costos y acelerar la introducción de nuevas aplicaciones. La OMG impulsa el desarrollo dirigido por modelos a través de la iniciativa Model Driven Architecture[2]. Es un consorcio internacional cuya membresía es abierta y en el cual cualquier componía puede formar parte de ella y ser parte de sus procesos.

1.1.3 Model Driven Architecture(MDA)[2]

MDA es un framework arquitectural con soporte de detalladas especificaciones. Estas especificaciones serán las conductoras hacia una industria interoperable, reusable y portable de componentes de software y modelos de datos basados en modelos estándares.
MDA es una iniciativa sobre el desarrollo de sistemas, que se sustenta en el poder de los modelos. Se dice que es un desarrollo conducido por modelos porque son ellos los que dirigen el curso de entendimiento, diseño, construcción, despliegue, operación y mantenimiento y modificaciones de los sistemas.[2].

1.1.4 Independencia de la plataforma

Es una cualidad que los modelos tienen. Ellos son independientes de cualquier plataforma[2].

1.1.5 Puntos de vista de MDA

- *CIM:Modelo Independiente de computación*

  Es una visión de un sistema desde su punto de vista independiente de aspectos computacionales. Un CIM no muestra detalles estructurales de los sistemas y a veces se lo llama *modelo de dominio o modelo de negocios* y es especificado usando un vocabulario que es familiar participantes que trabajan sobre un dominio particular.[2]. Utilizamos CIMs para especificar los requerimientos de un sistema y para mostrar en el contexto en el cual un sistema operará. Es útil para formalizar el problema y un vocabulario común.

- *PIM: Modelo Independiente de la plataforma*

  Es una visión de un sistema independiente de la plataforma. Su grado de independencia de la plataforma es tal que puede ser apto para ser utilizado en

varias de ellas. Puede consistir de información computacional que podrá ser apropiada para algún estilo arquitectural o varios[2]

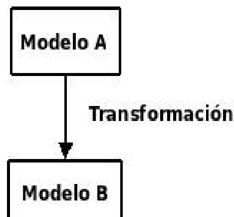
- *PSM: Modelo Especifico de la plataforma*

Es una visión de un sistema desde el punto de vista de una plataforma especifica. Combina la especificación de un PIM con los detalles de uso de un tipo particular de plataforma. MDA está basada en modelos detallados de plataforma expresados en UML, OCL y almacenados en el repositorio de de conformidad de MOF[2]. Un ejemplo de PSM definido por la OMG es CWM Common Warehouse Metamodel(CWM)[17] del cual nos ocuparemos en la sección 2.4.

- *Transformación de modelos*

Es el proceso por el cual un modelo es convertido en otro[2] de acuerdo a técnicas formalmente definidas y utilizando una herramienta de transformación automática.

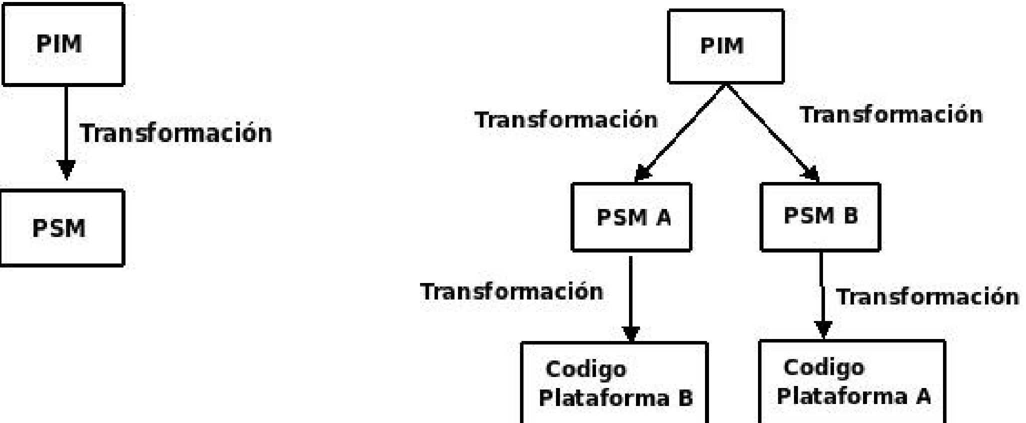
Figura 1.1 Esquema de transformación modelo a modelo



El patrón MDA especifica que un PSM es obtenido a través de un PIM por medio de una transformación.

Figura 1.2 Transformación de PIM a PSM

Figura 1.3 Transformación de un PIM a PSMs de diferentes plataformas



1.1.6 Arquitectura de cuatro capas propuesta por la OMG[1]

Un modelo se dice que conforma a su metamodelo. De la misma manera, un metamodelo debe conformar a su metametamodelo. En esta arquitectura de capas (modelos, metamodelos y metametamodelos), el metametamodelo usualmente conforma a su propia semántica, es decir, éste puede ser definido utilizando sus propios conceptos.

Capas definidas por la OMG:

- *M0: Instancias*
  En el nivel M0 se encuentran todas las instancias "reales" del sistema, es decir, los objetos de la aplicación. ejemplo: Persona, Casa, etc.
- *M1: Modelo del sistema*
  Representa el modelo de un sistema de software. Sus elementos son modelos de los datos. Ejemplo: Clase Persona.
- *M2: Metamodelo*
  Esta capa recibe el nombre de metamodelo. Ejemplo: UML
- *M3: Meta-metamodelo*
  Un meta-metamodelo es un modelo que define el lenguaje para representar un metamodelo. La relación entre un meta-metamodelo y un metamodelo es análoga a la relación entre un metamodelo y un modelo. Ejemplo: MOF.

1.1.7 El lenguaje de modelado MOF[5] de la OMG[1]

El lenguaje MOF[5], acrónimo de Meta-Object Facility, es un estándar del OMG[1] para el desarrollo conducido por modelos. Como se vio anteriormente, MOF[5] se encuentra en la capa superior de la arquitectura de 4 capas. Provee un meta- meta lenguaje que permite definir metamodelos en la capa M2. El ejemplo más popular de un elemento en la capa M2 es el metamodelo UML, definido el lenguaje MOF[5] y que describe al lenguaje UML[6].

1.1.8 Modelado Específico del Dominio (DSM y DSLs)[3]

La propuesta denominada DSM (Domain-Specific Modeling) consiste en la creación de modelos para cada dominio específico, utilizando un lenguaje focalizado y especializado para dicho dominio. Estos lenguajes se denominan DSLs(por su nombre en inglés: Domain- Specific Language)[3] y permiten especificar la solución usando directamente conceptos del dominio del problema. Estos lenguajes pueden ser gráficos o textuales.
Al igual que MDA[2], DSM[3] usa los conceptos dominio, modelo, metamodelo, meta-metamodelado e intenta ser dar una visión general del desarrollo dirigido por modelos. Los DSLs[3] son usados para construir modelos. Estos lenguajes pueden no utilizar ningún estandard de la OMG[1] ,pueden no estár basados en UML[6] y los metamodelos no son instancias de MOF[5]; aunque es de destacar que el ejemplo mas emblemático de construcción de DSLs[3] es a través de la definición de perfiles UML[7] para especializar de alguna manera la generalidad que posee UML[6]. Podemos tener DSLs[3] aplicando estándares de la OMG o sin ellos. Los que apoyan la visión general de DSLs[3] como desarrollo dirigido por modelos pregonan las siguientes principales ventajas:

- ✔ Los DSLs son mas comprensibles.
- ✔ Los DSLs tiene un mayor poder de expresividad que un lenguaje de modelado general como UML de la OMG.
- ✔ Los modelos de un DSL son autodocumentativos.

1.1.9 Confusión entre MDA[2] y DSM[3]

A veces la diferencia entre la propuesta MDA[2] de la OMG[1] y la propuesta DSM[3] parece un poco difusa. Algunas plataformas que dan soporte al desarrollo dirigido por modelos basan sus lenguajes ,modelos y herramientas en definiciones e implementaciones de estándares de la OMG pero con el objetivo de crear DSLs, lo que a veces al leer sus especificaciones hace que exista una gran confusión entre una y otra visión sobre el desarrollo dirigido por modelos. Existen autores que están apoyan una u otra visión. Como por ejemplo en [25] y [26] los autores descreen en la visión de MDA en pos de DSM debido a la rigurosidad y amplitud de MDA[2]. La crítica hacia MDA[2] radica por el basamento de esta en lenguajes complejos y por hacer hacer un paralelismo entre MDA[2] y el "fracaso " de las herramientas CASE de épocas anteriores . En los capítulos siguientes explicaremos como esta dualidad de conceptos afecta al desarrollo y cual fue la visión del desarrollo basado en modelos utilizada para desarrollar en este trabajo.

1.2 Diseño de Bases de Datos Relacionales

El proceso de diseño de una base de datos es una serie de pasos por el cual se trata de obtener un modelo que capture aquellos aspectos de la realidad que necesitamos registrar y luego su realización en un modelo relacional de tablas. Este proceso a grandes rasgos consiste en partir de una visión conceptual de la realidad y pasar a través de una serie de modelos hasta llegar a la implementación física de tablas relacionales en una base de datos. Por simplicidad se omitirán los pasos intermedios de cada parte del diseño como son la normalización y la modificación de los modelos con fines de performance o para facilitar el acceso a los datos. De estos aspectos nos ocuparemos en trabajo futuros.

Figura 1.4 pasos para diseñar una bse de datos y sus modelos mas importantes

DISEÑO CONCEPTUAL (Modelo de Entidad-Relación)

↓

DISEÑO LÓGICO (Modelo Relacional)

↓

DISEÑO FÍSICO (Esquema de tablas o código sql ddl)

1.2.1 Los modelos y el diseño de bases de datos

El desarrollo dirigido por modelos tiene un mayor grado de aplicabilidad en aquellos dominios en donde ya existen modelos de facto y en donde podemos decir que los

modelos son(o al menos forman parte) del hilo conductor del desarrollo. Tal es el caso del diseño de bases de datos, en donde los modelos gozan de una gran aceptación y están altamente arraigados entre la comunidad de informáticos que trabajan con bases de datos. Casi cualquier informático sabe entender diagramas de Entidad-Relación y todas sus extensiones(ej: agregaciones, jerarquías, etc) y como modelar la realidad utilizándolos. Los modelos relacionales no son desconocidos y se utilizan a diario en la mayoría de los proyectos informáticos de la actualidad. Desde hace largo tiempo que los diseñadores de bases de datos han estado aplicando una especie de desarrollo dirigido por modelos y lo que queda para aplicar MDA[2] al desarrollo de bases de datos es una formalización de los pasos que se llevan a cabo y que van desde el diseño conceptual de las bases de datos hasta su codificación en un motor de bases de datos relacional. A continuación describimos los pasos principales llevados a cabo en el diseño de base de datos.

1.2.1.1 Diseño conceptual - Modelado de Entidad-Relación[8]

El diseño de la realidad se lleva a cabo generalmente mediante modelos de Entidad-Relación[8] .Este es un modelo que brinda la posibilidad de especificar de manera abstracta y conceptual la representación de datos. Se utiliza como un mecanismo de modelado de base de datos. Este paso es llevado a cabo modelando aquellos aspectos de la realidad que nos interesan sin importar la plataforma en la que desarrollaremos nuestra solución al problema diseñado. Sin embargo es necesario señalar que los modelos que aquí se utilizan si bien son independientes de la plataforma tecnológica son muy aptos para ser usados en la derivación de soluciones para la plataformas relacionales Podemos decir entonces que el modelo de Entidad-Relación sirve para modelar la realidad, que es independiente de la plataforma pero que es muy útil para ser usado para derivar modelos de tablas relacionales. Dejamos para el capítulo 4 una definición mas completa de modelo de Entidad-Relación para mostrar como se relaciona el mismo con el objetivo de nuestro trabajo y ahí ahondaremos en mas cuestiones sobre el modelo de Entidad-Relación.

1.2.1.2 Diseño Lógico - Modelado Relacional[9]

Es un modelo basado en la lógica de predicados y la teoría de conjuntos su idea fundamental es el uso de relaciones para modelar problemas reales y es el mas utilizado para administrar datos en bases de datos. Este modelo si bien es independiente de la tecnología no lo es de la plataforma ya que solo sirve para modelar datos en bases de datos relacionales. Es decir que es un modelo específico para modelar datos relacionales.
Dejamos para el capítulo 7 una definición mas completa de modelo relacional para mostrar como se relaciona el mismo con el objetivo de nuestro trabajo.

1.2.1.3 Diseño físico - Lenguaje SQL DML y SQL DDL

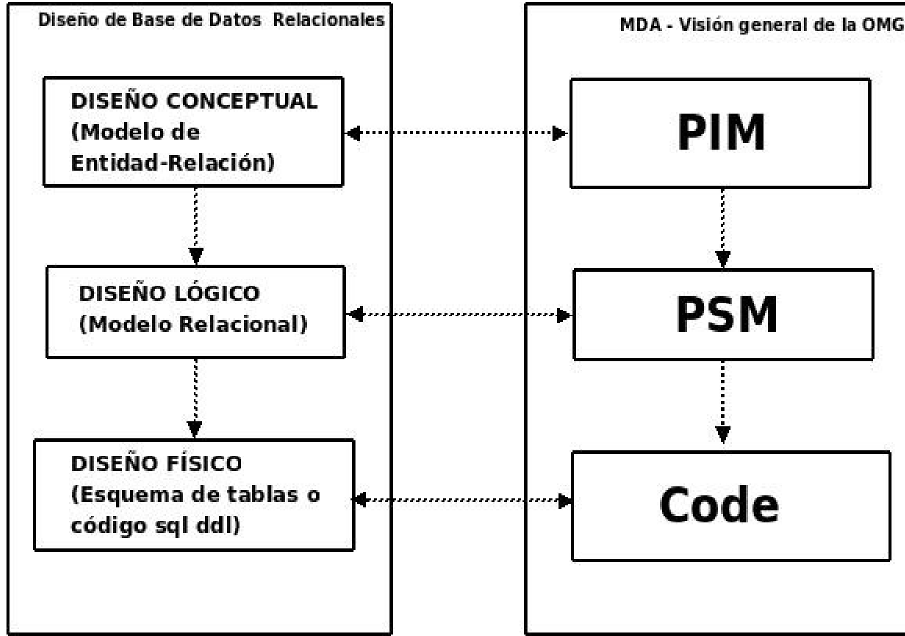
SQL DDL(Standard Query Language, Data manipulation language) es un lenguaje de Consultas Estructurado ,permite la comunicación con los Motores de Bases de Datos para consultarlos y manipularlos. El lenguaje SQL DDL(Standard Query Language, Data definition language) es el lenguaje que nos permite definir los objetos de nuestras bases de datos. Si bien existen estándares(ANSI) para el SQL DML y dicho estándar es respetado

por los fabricante de bases de datos no ocurre tal cosa para el lenguaje SQL DDL por lo tanto una herramienta que de soporte para su derivación automática debería considerar los casos para todos los fabricantes. Aquí por simplicidad solo daremos soporte para la creación de sentencias SQL DDL del motor de base de datos mysql.
Dejamos para un trabajo futuro una definición mas completa del lenguaje SQL DDL para mostrar como se relaciona el mismo con el objetivo de nuestro trabajo. No ahondaremos mas en el lenguaje SQL DML.

1.2.2 Modelado de Bases de datos con MDA[2]

Si observamos con cuidado el esquema general del diseño de Bases de Datos anteriormente descrito notaremos que perfectamente se ajusta a lo que MDA[2] pregona. Se modelan aspectos de la realidad en modelos diseñados específicamente para tal propósito .El modelo utilizado para esto es el modelo de Entidad-Relación. El mismo es ampliamente usado en el diseño conceptual de base de datos y como dijimos es independiente de cualquier plataforma subyacente en la que se vaya a implementar El modelo de Entidad-Relación es transformado luego en un modelo lógico que es el modelo de tablas relacionales, que si bien es independiente de alguna tecnología en particular no es independiente de la plataforma, ya que el modelo relacional es una manera especificar como se guardaran los datos en una base de datos relacional. Es decir el mismo es un modelo ESPECÍFICO de las bases de datos relacionales. Luego del modelo relacional derivamos las tablas y sus atributos. En nuestro caso podemos pensar esto como los scripts que crean las tablas que hemos diseñado derivando de los modelos anteriores. Esta manera general de diseñar una base de datos se ajusta perfectamente a lo que MDA[2] propone. Con lo cual debido a que los modelos que intervienen en el proceso gozan de una madurez de larga data, y que en el diseño de base de datos los modelos ya son ciudadanos de primera clase, pareciera ser que aplicar MDA[2] al diseño de base de datos relacionales es casi natural. Solo debemos formalizar un proceso que se viene realizando desde hace bastante tiempo e intentar formalizar las transformaciones que se llevan a cabo para llegar desde el modelo conceptual(PIM), al modelo lógico(PSM) y luego al código SQL.

Figura 1.5 la relación entre el proceso clásico de modelado de base de datos y MDA



1.2.3 Datawarehousing

A continuación brindamos una serie de conceptos generales necesarios para poder entender el trabajo [14] que fue uno de los motivadores del presente trabajo. Mas adelante daremos definiciones mas precisas de algunos de los conceptos expresados aquí de manera general.

1.2.3.1 Datawarehouse

Las empresas utilizan los datos acumulados durante años empleados en las transacciones comerciales, para ayudar a comprender y dirigir sus negocios; con ese propósito, los datos de las diferentes actividades se almacenan y consolidad en una base de datos central denominada datawarehouse; los analistas lo utilizan para extraer información de sus negocios que les permitan tomar mejores decisiones[11]. Un datawarehouse es una colección de datos no volátiles, que se acumulan en el tiempo, que están orientados a un tema determinado y que se utilizan para tomar decisiones organizacionales[12] el modelo multidimensional constituye la base del datawarehouse, en el la información se estructura en hechos y dimensiones; un hecho es un tema de interés para la empresa, se describe mediante atributos denominados atributos de hecho o medidas, estos están contenidos en celdas o puntos en el cubo de datos. Un cubo de datos es una representación multidimensional de datos donde estos pueden verse desde distintos puntos de vista; está formado por dimensiones que determinan la granularidad para la representación de hechos y jerarquías que muestran como las instancias de hechos pueden ser agrupadas y seleccionadas para los procesos de toma de decisión[13].

1.2.1.1 Finalidad de un Datawarehouse

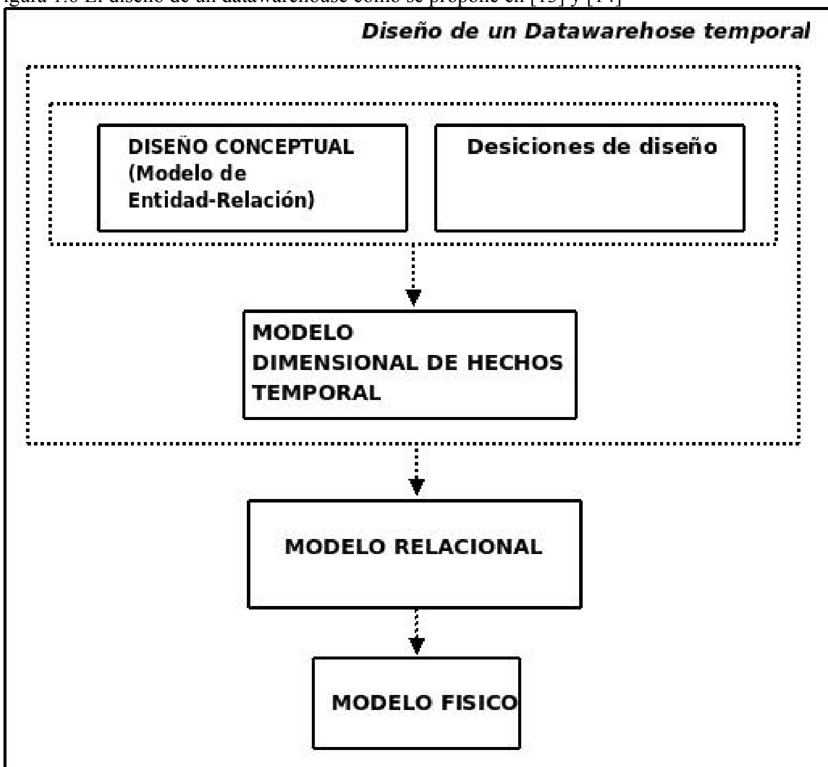
Consiste en auxiliar a comprender el pasado y planear el futuro.
Busca respuestas a preguntas como:
¿Qué están comprando nuestros clientes?
¿Qué no están comprando?
¿Qué incentivos han funcionado antes con los mismos clientes en esta época del año?.
¿Cuántos de nuestros vendedores visitan a un mismo cliente?.
¿Qué están haciendo nuestros competidores?
¿Cómo se comparan nuestros costos para cada línea de producto durante los últimos tres años?.
La promesa del datawarehousing es "sacar datos" de los sistemas operacionales para ayudar a las empresas a tomar mejores decisiones.

1.2.4 Diseño de datawarehousing

1.2.4.1 Diseño conceptual - Modelado Multidimensional

|Existen varias propuestas y formas de llevar a cabo el diseño de un datawarehouse. Aquí nos ocuparemos de aquellas que parten de una visión conceptual de la realidad y partiendo del diseño conceptual de una base de datos operacional desde el cual derivaremos el diseño del datawarehouse. De manera que el diseño del datawarehouse se realiza a través de una serie de pasos desde donde partimos de un modelo de la realidad y de decisiones que especifican aquellos aspectos que nos interesa capturar en el datawarehouse hasta llegar a su realización física en una Base de Datos. En nuestro caso nos ocuparemos del diseño que surge luego de aplicar decisiones de diseño a los modelos conceptuales de nuestras bases de datos operacionales previamente diseñadas. Para el mismo utilizaremos modelos multidimensionales derivados de nuestros modelos conceptuales de una base de datos existente. En particular utilizaremos una variante que es a su vez una variante de las numerosas propuestas que existen para el diseño de datawarehouses y que se describe mas adelante en el capítulo 7.

Figura 1.6 El diseño de un datawarehouse como se propone en [13] y [14]



1.2.5 Modelado de datawarehousing con MDA[2]

Si observamos con cuidado el esquema anterior también notaremos que perfectamente se ajusta a lo que que MDA[2] pregona. Se modelan aspectos de la realidad en modelos diseñados específicamente para tal propósito como puede ser el modelo de Entidad-Relación. Luego mediante algunas técnicas formales, otras informales y decisiones de diseño que especifican que aspecto nos interesa para la toma de decisiones derivamos el mismo en un modelo lógico que es el modelo de tablas relacional, Esta manera general de diseñar una datawarehouse también se ajusta perfectamente a lo que MDA[2] propone. Con lo cual al igual que los modelos anteriores aquí los modelos que intervienen en el proceso gozan de una madurez de larga data. Pareciera que aplicar MDA[2] al diseño de datawarehouses partiendo de modelos conceptuales(PIM) es casi natural. Solo debemos formalizar un proceso que se viene realizando desde hace bastante tiempo e intentar formalizar las transformaciones que se llevan a cabo para llegar desde el modelo conceptual(PIM), al modelo lógico(PSM) y luego al código sql.

Figura 1.7 El diseño de un datawarehouse como se propone en [13] y [14] y como abordarlo con MDA

1.3 Propuesta para modelar datawarehousing con MDA[14]

En este trabajo intentaremos demostrar la aplicabilidad de MDA implementando el desarrollo dirigido por modelo del proceso anteriormente descrito. Tomaremos una propuesta desarrollada por Carlos Neil y Claudia Pons[14] e intentaremos mostrar un ejemplo de como llevar dicho proceso formal a la práctica.

1.3.1 Aplicando MDA al diseño de un datawarehouse temporal[14]

En este trabajo nos basaremos en parte en el trabajo [14], *"Aplicando MDA al desarrollo de un datawarehouse"*, escrito por Carlos Neil y Claudia Pons. Dicho trabajo es un propuesta general sobre como utilizar el desarrollo basado en modelos y algunos de sus formalismos para especificar aquellos modelos que participan del diseño de base de datos con el objetivo final de dar soporte a la creación de la estructura de un datawarehouse temporal. En [14] se proponen varios metamodelos y transformaciones siguiendo la propuesta de MDA correspondientes a modelos muy utilizados en el área de Bases de Datos y Datawarehousing. Dicho trabajo a su vez está basado en el trabajo [14] sobre modelos multidimensionales. En [15] se presenta:

- ✔ Una notación gráfica conceptual para datawarehouses llamado MODELO HECHO DIMENSIONAL en donde se modela la realidad usando lo que se denomina un *esquema dimensional,* que consiste en un conjunto de esquemas de hechos cuyos elementos básicos son hechos de la realidad, dimensiones y jerarquías.

- ✔ Una metodología semiautomática para llevar a cabo el modelado conceptual comenzando de esquemas de entidad-relación que forman parte de modelo operacionales tradicionales

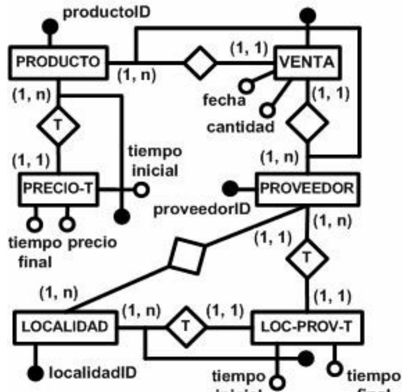
En el trabajo[14] se agregan conceptos temporales al modelo hecho dimensional de manera tal de capturar la variación en el tiempo de algunos aspectos a partir de un diagrama de Entidad-Relación[8] .

Sobre los Aspectos temporales a considerar:
En el datawarehouse el tiempo es una de las dimensiones para el análisis, pero este hace referencia al momento en que se realizó una transacción, no se detalla cuándo, en el mundo real, varían los atributos o interrelaciones involucradas en esas transacciones, La necesidad de registrar valores que permitan evaluar tendencias, variaciones, máximos y mínimos, justifican considerar en el diseño del datawarehouse cómo algunos atributos o interrelaciones pueden variar en el tiempo. Un esquema multidimensional temporal que incluya, además del hecho principal de análisis, esquemas temporales (que no pertenezcan a la jerarquía) permitirá registrar, además, la variaciones temporales de atributos y/o interrelaciones[14]. En [14] se propone que el proceso de generación del datawarehouse conste de una serie de pasos(transformaciones) que consisten en transformar modelo para luego obtener código SQL DDL[10]. El mismo fue concebido de acuerdo a la visión de la OMG[2] de como debe ser el proceso de desarrollo. Esto es: tenemos varios modelos, realizamos transformaciones entre ellos para concluir finalmente derivando en código SQL DML[10] para generar a estructura de un Datawarehouse con aspectos temporales.

Los modelos propuestos en [14] son:

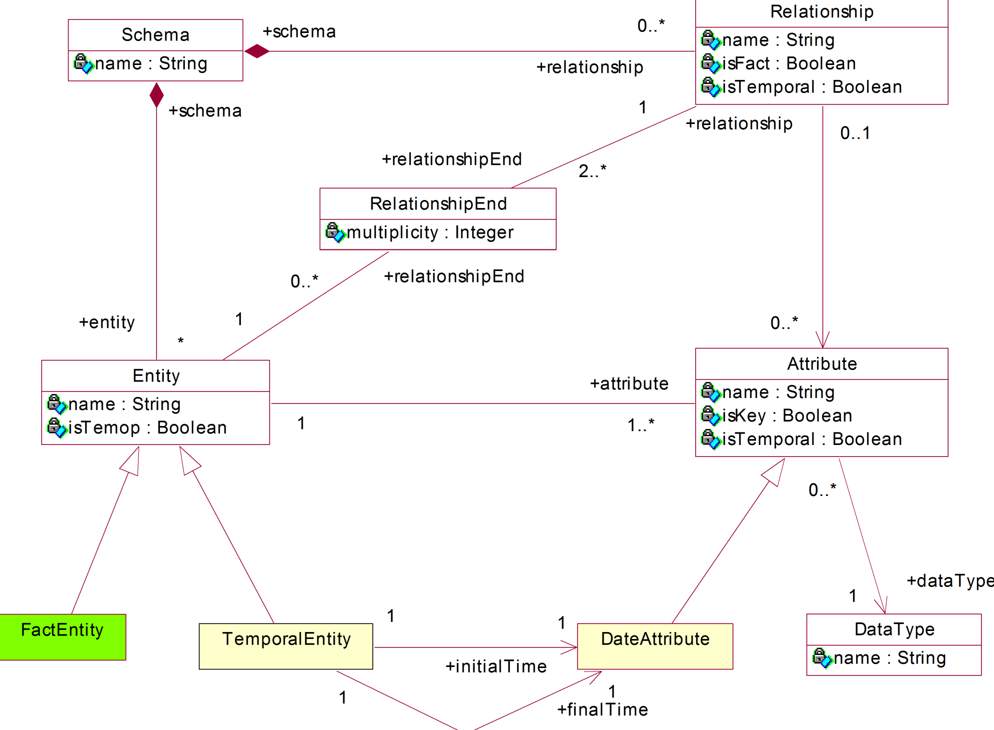
- ✔ *Entidad relación(ER) o Modelo de datos simple(mds) - PIM*
  Para el modelo ER se adopta la forma gráfica propuesta por Chen[8]

Figura 1.8 ejemplo de diagrama de entidad-relación especificado en [14]



Conforme al metamodelo: Metamodelo de datos simple(MDS)

Figura 1.9 Metamodelo ER(MDS) propuesto en [14]

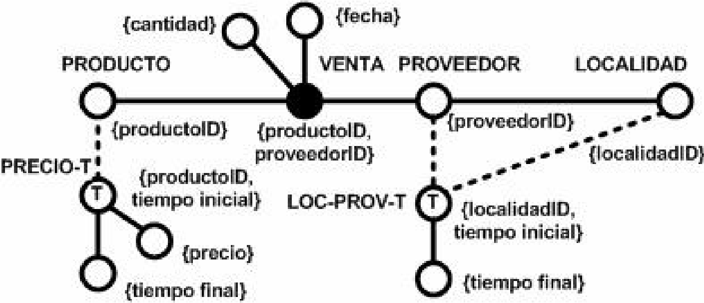


En el capítulo 6 se dan mas detalles de la base conceptual sobre el modelo de Entidad-Relación que se utiliza.

> ***Aclaración importante**:*Existe una diferencia entre la simbología con que se describen los modelos de Entidad-Relación en [14] y el metamodelo propuesto en [14], debido a que el mismo no considera el hecho de que una entidad tome como identificador un atributo de otra entidad. Con instancias del metamodelo anterior no es posible expresar este hecho. En el ejemplo la entidad *Venta* tiene como identificadores los identificadores de las entidades *Proveedor* y *Producto*. En un modelo conforme al metamodelo anterior este hecho no puede ser reflejado.

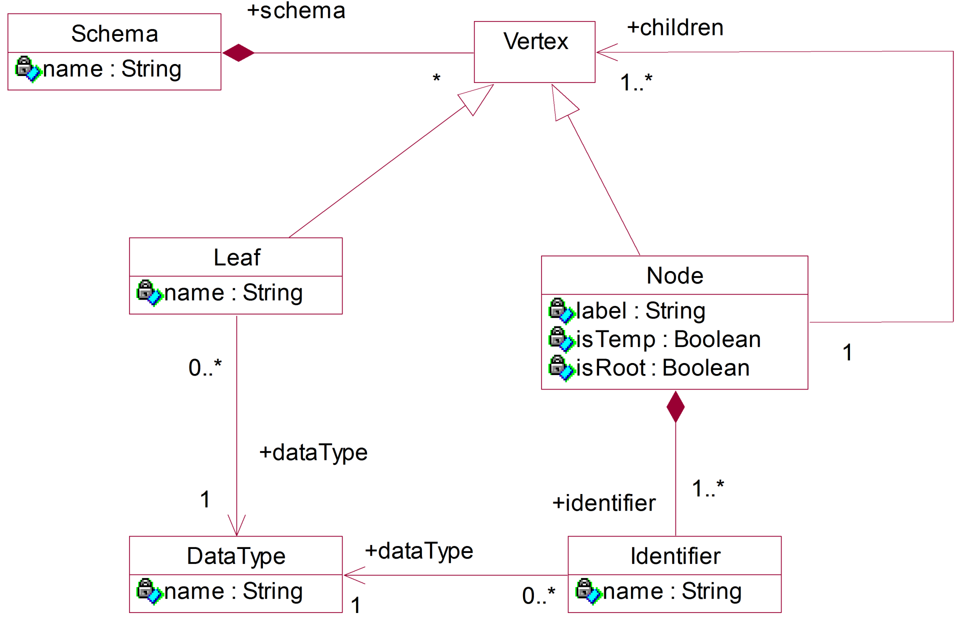
✔ *Grafo de Atributos – PIM*

Figura 1.10 Modelo de grafo de atributos propuesto en [14]



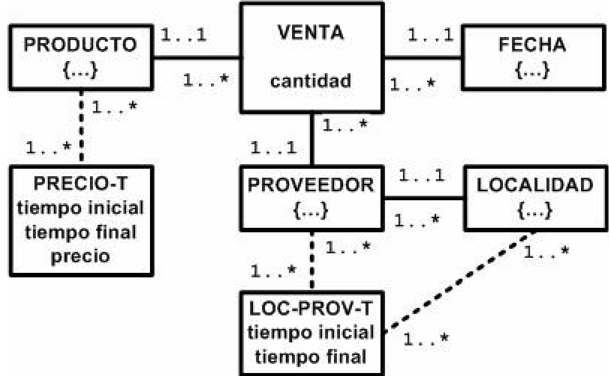
Conforme al metamodelo: Metamodelo de Grafo de Atributos(graph)

Figura 1.11 Metamodelo de grafo de atributos propuesto en [14]



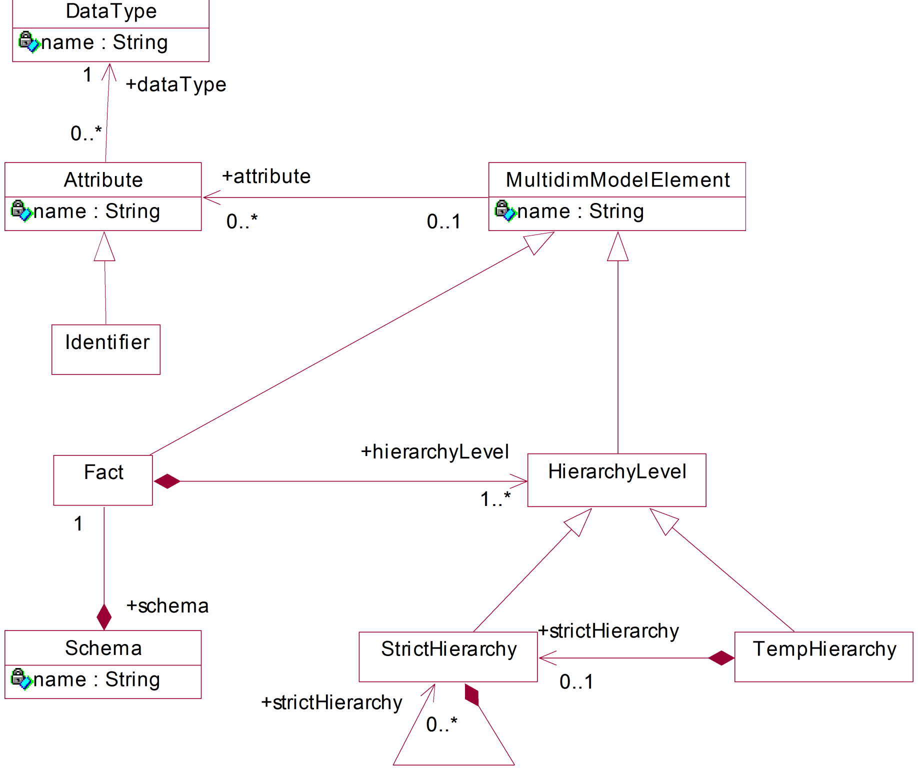
✔ *Modelo Multidimensional – PIM*

Figura 1.12 Modelo Multidimensional temporal propuesto en [14]



Conforme al metamodelo: Metamodelo multidimensional temporal(mmt)

Figura 1.13 Metamodelo Multidimensional temporal propuesto en [14]

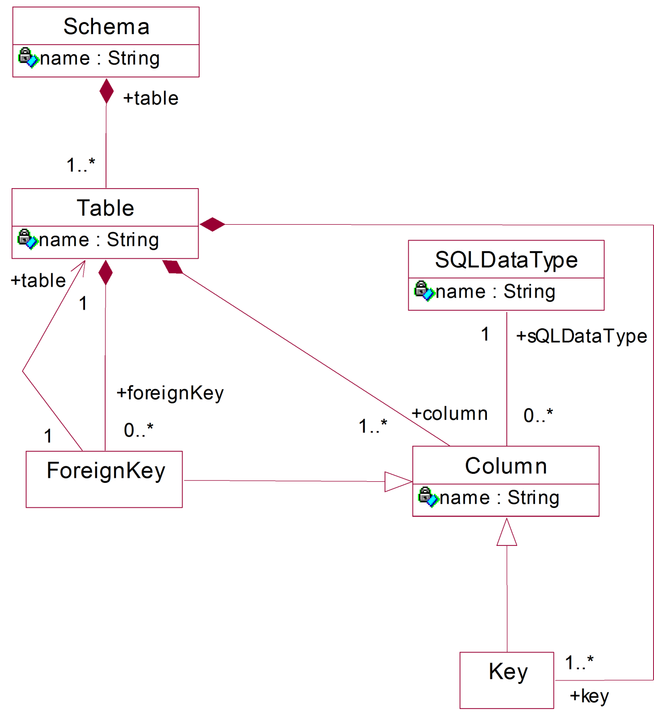


✔ *Modelo Relacional – PSM*

Conforme al metamodelo: Metamodelo relacional(mmer)

Consiste en un esquema de tablas relacionadas a través de claves foráneas

Figura 1.14 Metamodelo relacional propuesto en [14]



La transformaciones propuestas en [14] son las siguientes:

- ✔ *Obtener un modelo de datos simple reestructurado – PIM a PIM.*
  Especificación: Informal y basada en aspectos estructurales que presentan los modelos de entidad-relación así como también en conocimiento del problema modelado que tiene el diseñador.
  *Descripción:*Partiendo de un modelo de entidad-relación se intenta obtener otro tal que su estructura sea apta para ser llevada al modelo hecho dimensional. Se sigue la propuesta de [15].

- ✔ *Obtener un modelo de datos simples con aspectos temporales - PIM a PIM*
  Especificación: Informal y basada en aspectos semánticos que presentan el modelo de entidad-relación así como también en conocimiento del problema modelado que tiene el diseñador
  *Descripción:*Partiendo de un modelo de entidad relación obtener uno que exprese que aspectos temporales del mismo se desea que sean preservados.

- ✔ *Obtener un Grafo de atributos - PIM a PIM*
  Especificación: Semiformal y basada en un algoritmo adaptado del trabajo [15].
  *Descripción:*A partir de un modelo de entidad relación que se asume reestructurado se obtiene un grafo que describe la estructura del modelo de

entidad-relación . Dicho proceso esta basado en un algoritmo propuesto en [15] y contiene modificaciones para especificar aspectos temporales.

- ✔ *Obtener Modelo Multidimensional temporal – PIM a PIM*
  Especificación: Informal y basada en el conocimiento del problema que tiene el diseñador para especificar dimensiones temporales y en la estructura de los modelos anteriores para definir hechos y dimensiones no temporales.
  *Descripción:*Derivamos un modelo multidimensional basado en el modelo Hecho-Dimensional que se propone en [14] a partir del grafo de atributos.

- ✔ *Obtener Modelo relacional - PIM a PSM*
  Especificación: Informal
  *Descripción:*Obtenemos un esquema de tablas relacionales para el modelo multidimensional.

- ✔ *Derivar Código SQL - PSM a Código*
  Especificación: Informal
  Descripción:En el trabajo tomado como referencia se deriva el modelo relacional en un esquema textual de tablas al consultar con los autores propusieron que se debería derivar código SQL DDL para crear tablas en algún formato compatible con algún manejador de bases de datos relacional. Es necesario aclarar que al no haber un estandard para el SQL DDL y que cada fabricante posee su lenguaje SQL DDL.

Capítulo 2

"Cualquier plan de estandarizar un modelo de propósito general o un conjunto de modelos fallará debido a que los seres humanos necesitan inherentemente expresar su propia creatividad [NIH] . Ademas el nivel de inversión requerida para aprender y manejar un lenguaje complejo, volverse familiar con el mismo, sus herramientas asociadas y luego incorporarlo a un proceso de desarrollo hace que esto sea extremadamente costoso. Por lo tanto cuando modelar es ventajoso, lo es utilizando herramientas que permitan la construcción de DSLs en lugar de lenguajes de modelado basados en estándares"

Richard Gronback en su libro Eclipse Modeling Project as a dsl toolkit[16]

2 Base Conceptual y herramientas

El objetivo de este capítulo es dar una visión del desarrollo basado en modelos que se percibe a través de las especificaciones de la OMG[2] y de las herramientas que brindan soporte para el desarrollo basado en modelos. Se intenta transmitir como llegamos a la visión utilizada en este trabajo sobre el desarrollo basado en modelos.

2.1 Modelos de propósito general

Adhiero a la idea anterior sobre el desarrollo de DSLs[16] en lugar del uso de modelos de propósito general como UML[6]. Es buena practica refinar o implementar modelos acorde a nuestro dominio con el fin de formalizar conceptos y lograr un lenguaje formal para utilizarlo durante el desarrollo. También adhiero a la idea de que "los mapeos de modelo de propósito general a modelo de propósito general tipicamente no funcionan(UML a código java por ejemplo)"[16]. Esta es una idea que fue tenida en cuenta al tomar algunas decisiones sobre el desarrollo del presente trabajo.

2.1.2 ¿Lenguajes de propósito general? ¿DSL existente? ¿DSL propio?

La regla general es que es mejor formalizar nuestro problema mediante un DSL[3], creándolo o acotando uno existente (uml con estereotipos por ejemplo) que utilizar modelos de propósito general como UML[6] para modelar y derivar desde ese lugar nuestras soluciones. La política de resolver problemas obteniendo soluciones a partir de DSL es buena y atenta contra la informalidad causada por la distancia semántica que existe entre lenguajes de propósito general como UML[6] y el código de nuestra solución. El uso de DSLs[3] permite acortar esta brecha y obtener soluciones formales acordes a los propios formalismos del dominios en cuestión.

Si bien lo anterior es cierto; al modelar problemas recurrentes a través de DSLs se corre el riesgo de que existan muchos DSLs para el mismo problema. Existen circunstancias en las que los problemas son tan comunes y recurrentes que es una buena inversión afrontar la dificultad de estandarizar un modelo de propósito muy amplio, utilizando un modelo de propósito general como UML[6] y hacer que nuestras soluciones se ajusten a los mismos con el fin de lograr la uniformidad, portabilidad y la independencia de la tecnología. En estos casos, que son relativamente pocos es mejor tener un modelo de propósito muy amplio y abarcador que varios DSLs.
Debido a lo anteriormente descripto sobre el uso de DSLs y modelo de propósito general y debido y a que nuestra de aplicación trabaja sobre modelos de bases de datos lo ideal seria que nuestro desarrollo se adapte a un metamodelo estandard de base de datos de ser posible.

2.1.2 Visión del desarrollo dirigido por modelos utilizada

Este trabajo no tiene como objetivo primario indagar sobre cuestiones conceptuales sobre modelos, metamodelos y transformaciones pero si se propone un desarrollo dirigido por modelos y siguiendo los lineamientos principales de la OMG[1] debido a que el área de aplicación de este trabajo es muy amplio. El objetivo inicial es desarrollar un ejemplo de toolkit que servirá para el desarrollo por dirigido por modelos en el área de Base de Datos.
La visión que adoptamos del proceso está determinada por la visión de la OMG[1] de MDD a través de la iniciativa MDA[2] y por la tecnología que se utilizará para desarrollar el toolkit que es la plataforma de modelado Eclipse Modeling Project[20]. Esta tiene una visión orientada a la construcción de DSLs para el desarrollo guiado por modelos.
La plataforma Eclipse nos brinda excelentes herramientas para construir DSLs adoptando los estándares de la OMG. Para nuestro desarrollo veremos el proceso de generación del datawarehouse y sus modelos propuestos en [14] como un proceso dirigido por modelos en el que partimos de un PIM (diseño conceptual) , pasando por un PSM(diseño lógico) y finalmente pasamos al código, tal proceso se describe en la sección 1.2.5. Cada PIM y PSM se construirá como un DSL utilizando las implementaciones de Eclipse Modeling Project de los estándares de la OMG. Desdee los metamodelos(notaciones abstractas) de [14] derivaremos el código fuente de los editores(notaciones concretas) de cada DSL . El usuario del toolkit también tendrá esta visión de los modelos que manipula ya que tendrá notaciones abstractas y concretas de los modelos que utiliza e incluso podrá tener mas de una notación concreta para una misma notación abstracta(ejemplo: una notación gráfica y una textual para modificar un modelo de Entidad-Relación[8]). Esta manera de ver el desarrollo es para tener la base conceptual de nuestro toolkit en conformidad con la visión de eclipse del desarrollo basado en modelos. Es por eso que decidimos apartarnos en parte de la visión de la OMG[1] con respecto al desarrollo dirigido por modelos y con la cual fue concebido [14] para tener también una coincidencia de conceptos con Eclipse Modeling Project[20].

2.1.3 Common Warehouse Metamodel (CWM)[17]

CWM [17] es una especificación estandard propuesta por la OMG[1] y fue pensada como un mecanismo de intercambio de modelos de bases de datos para datawarehousing. A pesar de ser concebido pensando en datawarehousing, el mismo abarca metamodelos para casi todos los artefactos que intervienen en el área de las base de datos. Abarca desde metamodelos de los artefactos utilizados en diagramas de Entidad-Relación, modelo relacional, modelos OLAP ,procesos de visualización y data mining.
CWM[17] es definido heredando características propias de UML[6] y re-definiendo varias cosas del mismo. Lo ideal para nuestro desarrollo seria adoptar como norma la especificación de CWM[17] ya que sus modelos, artefactos y lenguajes son un estandard en casi cualquier proyecto informático. Hoy en día las diferencias semánticas y sintácticas en este ámbito son entendidas por todos(por ejemplo las diferentes maneras de especificar la cardinalidad en modelos de Entidad-Relación) lo que hace mas fácil el entendimiento de los modelos utilizados para su formalización(metamodelos). Es bueno en estos casos donde tenemos modelos ampliamente aceptados en la industria informática tener un modelo estandard de propósito muy amplio para modelar el problema y así facilitar el intercambio de modelos entre fabricantes.

2.2 Un toolkit DSL

> ***“Muchas consideraciones están involucradas en la creación de un DSL. ¿Existe ya un modelo que se ajuste suficientemente a nuestras necesidades?, de ser así, ¿puede este ser extendido o corregido?. ¿Necesita el modelo estar basado en algún estandard?. ¿El modelo se presta para ser mostrado y editado de manera visual y elegante? ”***
> Richard Gronback en su libro Eclipse Modeling Project as a DSL toolkit [16]

Al decidirme a comenzar a implementar un proceso de desarrollo de software basado enteramente en modelos, y sabiendo que debería implementar metamodelos, y realizar transformaciones entre ellos y sin conocer el proyecto Eclipse Modeling Project y sin conocer aun lo que era un DSL las anteriores fueron cuestiones que me planteé, a las cuales tuve que responder. Dichas respuestas serán las que intentaré explicar como respondí en este capítulo.

¿Existe ya un modelo que se ajuste suficientemente a nuestras necesidades?

Si existe y se denomina Common Warehouse Metamodel(CWM)[17] que es un estándar definido por la OMG[1] específicamente para modelar aplicaciones de bases de datos. Este estándar, tiene metamodelos específicos para el modelado de base de datos relacionales, OLAP, XML, ER, etc.

¿Puede este ser extendido o corregido?

Si, efectivamente tal como el mismo es definido re definiendo cosas de UML[6]. La idea es no corregirlo ya que el mismo intenta ser un estandard que formalice el intercambio de información con lo cual su modificación debería quedar a criterio de la OMG[1]. Por lo expuesto posteriormente la compatibilidad de nuestro trabajo a CWM[17] de la OMG[1] será abordada en el futuro.

¿Necesita el modelo estar basado en algún estandard?

Debido a que el dominio del presente trabajo es el área de Base de Datos que es cross a casi toda la industria seria bueno que si pero esto utilizar CWM[17].

¿El modelo se presta para ser mostrado y editado de manera visual y elegante?

No, a los fines de crear editores visuales CWM[17] es muy amplio, habría que utilizar una versión simplificada del mismo y luego convertirlo en un modelo conforme a CWM.

2.3 Soporte de la plataforma Eclipse para el desarrollo de DSLs y basado en MDA

Se brinda aquí un panorama de algunos de los frameworks y herramientas que nos brinda la plataforma eclipse[18][19] para el desarrollo basado en modelos y que fueron utilizados en el presente trabajo. Se da aquí un panorama general de cada herramienta y a lo largo de los capítulos que siguen se brindan detalles de como se aplicó cada una de las herramientas aquí mencionadas.

Eclipse RCP (http://wiki.eclipse.org/index.php/Rich_Client_Platform)

Fue diseñado para servir como plataforma abierta y definido de tal manera que sus componentes pueden ser utilizados para construir cualquier aplicación cliente que uno requiera. Consta de un conjunto mínimo de plugins que se necesitan para construir un cliente rico .Se denomina RCP(Rich Cliente Platform)[19]. Las aplicaciones que construimos se basan en el modelo de plugins dinámicos y la interfaz visual es construida utilizando las mismas herramientas y puntos de extensión que provee la plataforma.
Los plugins que forman parte de la plataforma mínima de RCP[19] son:

- ✔ **org.eclipse.ui**
- ✔ **org.eclipse.core.runtime**

y sus prerrequisitos.

Componentes que forman parte de RCP ademas del conjunto mínimo:

- ✔ **Eclipse Runtime**: Provee el soporte para la carga de plugins y puntos de extensión, es construido utilizando el framework OSGI.
  Plugins:
  org.eclipse.core.runtime
  org.eclipse.osgi
  org.eclipse.osgi.services

- ✔ **SWT(Standard Widget Toolkit)**:Diseñado para crear interfaces de usuarios basada en el API nativa de cada sistema operativo para crear widgets.
  Plugins:
  org.eclipse.swt + Utilidades básicas brindadas por el sistema operativos.

- ✔ **JFace:** Un framework para construir interfaces visuales construido sobre SWT para manejar aspectos comunes en la construcción de interfaces.
  Plugins:
  org.eclipse.jface

- ✔ **Workbench:** Esta construido al tope de los componente runtime, SWT, Jface para proveer un entorno abierto, escalable y de múltiples ventanas para el manejo de vistas, editores, perspectivas , wizards y páginas de preferencia.
  Plugins:
  org.eclipse.ui
  org.eclipse.ui.workbench

- ✔ **Otros prerrequisitos:** Soporte para lenguajes de expresión XML, comandos y ayuda.
  Plugins:
  org.eclipse.core.expressions
  org.eclipse.core.commands
  org.eclipse.help

Eclipse Modeling Project(http://www.eclipse.org/modeling/)

Es un subproyecto de eclipse[20] y se concentra en la evolución y promoción de desarrollo basado en modelos y provee un conjunto unificados de frameworks de modelado, herramientas y estándares de implementación. Provee herramientas para especificar notaciones abstractas, concretas y para transformar modelos. Se han evaluado múltiples herramientas pertenecientes al proyecto EMP, a continuación se brinda una

descripción de las que han sido seleccionadas para desarrollar la herramienta en respuesta a la propuesta de [14].

2.3.2.1 Notaciones abstractas

✔ *Eclipse Modeling Framework* (EMF) - http://www.eclipse.org/modeling/emf/

Es un framework de modelado y de generación de código automática para la construcción de aplicaciones basadas en un modelo de datos estructurado. Permite la construcción de herramientas y otras aplicaciones. La especificación de los modelos. Desde una aplicación descripta en XML, EMF provee herramientas soporte en runtime para producir un conjunto de clases java que implementan el framework adapter para permitir su vista y edición. Así como también nos brinda la derivación automática de editor de modelos basados en la estructura de los modelos.

Subroyecto de EMF utilizados

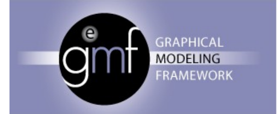
*Core:*Nos permite convertir nuestros modelos en código java que posee las características anteriormente descripta. Los modelos pueden ser especificados utilizando clases java anotadas, documentos XML o en herramientas de modelado como Racional Rose, luego ser importadas en EMF para ser utilizada luego en la generación del código de nuestros modelos.

Validación Framework:Provee definición de constraints para evaluar la validez de modelo EMF. Algunas características importantes que se utilizan en este trabajo:

- Definir constringente: Provee un API para la definición de constraints para cualquier metamodelo EMF(de forma batch y al instante)
- Constraints configurables en base a contextos de aplicación. Nos brinda un API para definir contextos basados en nuestras aplicaciones que describen nuestroxs objetos y aquellos aspectos que necesitan ser validados.
- Listeners de validación: Soporte para la validación ante la ocurrencia de eventos.

2.3.2.2 Notaciones concretas

✔ Graphical Modeling Framework(GMF)-http://www.eclipse.org/modeling/gmf/



Nos provee un componente generativo e infraestructura en runtime para el desarrollo de editores gráficos basados en EMF y GEF. Combina estos dos frameworks y nos permite tener editores basados en figuras y una paleta de componentes provistos por GEF para nuestros modelos basados en EMF

✔ *Textual Modeling Framework(TMF) - http://www.eclipse.org/modeling/tmf/*
Nos brinda la posibilidad de construir DSLs textuales para nuestros modelos. Su principal herramienta es xText.

Xtext - http://www.eclipse.org/Xtext/
xText es un framework para el desarrollo de DSLs textuales. Se basa en describir nuestros DSLs en base a un lenguaje gramáticas EBNF y el generador de xText creará un parser y un metamodelo para el mismo basado en EMF y un editor textual para la editar nuestros modelos.

Transformaciones

✔ *Model to Model Transformation*(M2M) - *http://www.eclipse.org/m2m/*
Model to Model transformation es un framework clave en el desarrollo basado en modelos acorde a la plataforma eclipse. El proyecto M2M nos brinda un framework con lenguajes para transformaciones modelo a modelo. Las transformaciones son ejecutadas por motores de ejecución de transformaciones. Existen tres motores de transformación.
En este trabajo utilizamos el lenguaje de transformación de modelos ATL(Atlas Transformation Language) que forma parte del proyecto M2M.

✔ Model to Text Transformatin(M2T)
Las herramientas que nos provee el subroyecto M2T nos permiten la generación de código a partir de un modelo. En este trabajo utilizaremos el lenguaje xPand: (http://wiki.eclipse.org/Xpand) que nos permite derivar texto a partir de modelos EMF.

ATL (ATLAS Transformation Language) - http://www.eclipse.org/m2m/atl/

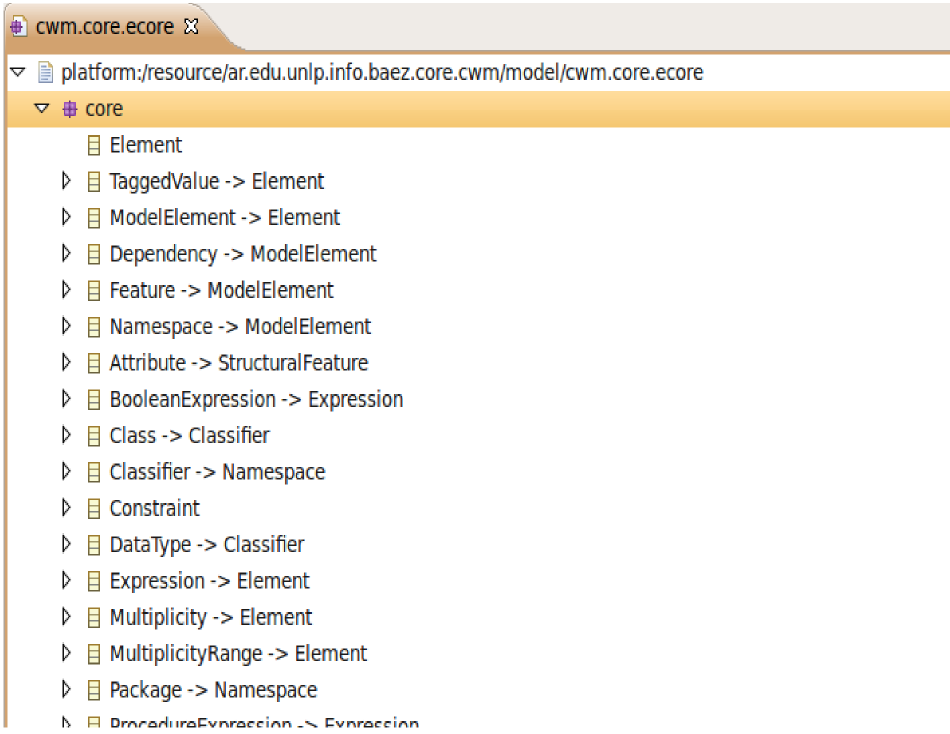
Forma parte del proyecto M2M de eclipse y es un toolkit para transformación de modelos en el ámbito del desarrollo basado en modelos. Está desarrollado utilizando la plataforma eclipse y proveen entorno de desarrollo de transformaciones integrado a Eclipse con todas las facilidades que esto implica: coloreo de sintaxis, debugger,etc.

2.4 Implementación de CWM[17] con EMP(ECWM)

2.4.1 Un primer intento

Debido a lo expuesto en 2.2 se ha implementado una versión de CWM implementada en lenguaje ecore[17]. La misma fue utilizada para intentar utilizarla como notación abstracta para los modelos que forman parte del proceso de generación del datawarehouse. En un principio esto parecía ser una opción viable debido a que los mismos modelos son muy parecidos en su estructura a los de [14] que fueron tomados como base para el trabajo. Se ha llegado a tener una muy completa versión de CWM[17] basada en ecore y derivar de la misma una versión de un editor EMF basado en la estructura del modelo.

Figura 2.1 Vista de

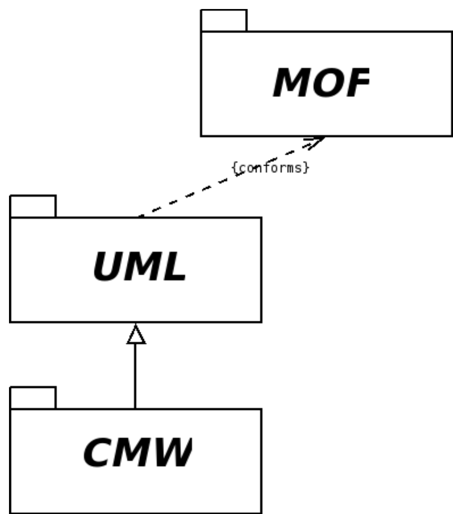


2.4.2 Inconvenientes causados de implementar ECWM

2.4.2.1 Alta Complejidad para modelar

Casi todas las especificaciones de la OMG[1], son kilométricas y CWM[17] no es la excepción. Unas de las desventajas del metamodelado es que podemos llegar a obtener algo tan largo y extenso como el propio UML. CWM[17] es un ejemplo claro de lo mismo. CWM[17] es definido heredando algunas características del propio UML. Como ya sabemos UML[6] a su vez está definido mediante el lenguaje MOF[5].

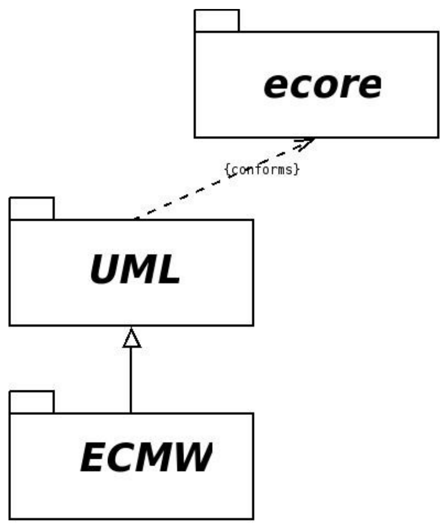
Figura 2.2 relación entre CWM y algunos metamodelos de la OMG



2.4.2.2 Diferencias entre UML2/MOF - UML2/ECORE

El lenguaje de nivel 3 de Eclipse Modeling Project[20] es ecore[22]. Eclipse a su vez posee una implementación de la especificación de UML2 de la OMG[1] realizada en lenguaje ecore[22]. Con lo cual es posible implementar CWM/MOF mediante la plataforma eclipse(CWM/ECORE) .Es necesario para esto heredar de la implementación de UML2/ECORE de eclipse y así obtener nuestra implementación de CWM[17] denominada ECWM. Ecore es similar a MOF[5] pero no totalmente acorde al mismo. Algunas definiciones hechas en CWM[17] no son totalmente realizables en nuestro intento de implementar ECWM. Por ejemplo para aprovechar las ventajas del lenguaje ecore[22] las relaciones de CWM[17] deberían ser modeladas con las references de ecore. Con lo cual nuestra implementación de CWM[17] heredará muchas decisiones adoptadas por los diseñadores de ecore y uml2 de eclipse. Y ECWM tendrá algunas diferencias con la especificación CWM[17].

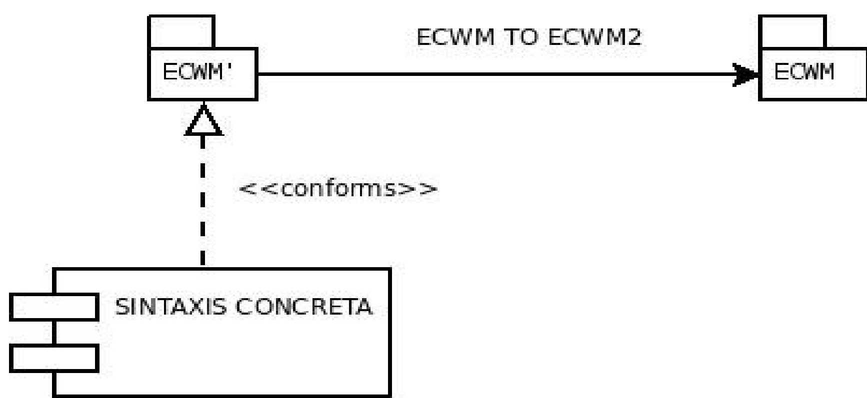
Figura 2.3 relación entre la definición ECWM y algunos metamodelos la plataforma Eclipse



2.4.2.3 Transformaciones necesarias

El hecho de que CWM[17] es altamente complejo(y por ende nuestro ECWM también) y demasiado abarcador, dificulta el uso de sus modelos para ser utilizados en las implementaciones de nuestros editores y lenguajes. De esta forma eran conveniente que nuestros metamodelos estén basados en simplificaciones de ECWM, llamada ECWM'. Luego las instancias de nuestros modelos(instancias de ECWM') debían ser transformadas a ECWM para así lograr el fin por el cual fue concebido CWM que es tener las instancias de nuestros objetos de bases de datos en un formato único que pueda eventualmente ser leído por herramientas de diferentes fabricantes.

Figura 2.4 Transformación necesaria para que nuestra versión simplificada de ECWM sea compatible con ECWM



Esta tarea a pesar de ser totalmente realizable es demandante en tiempo y análisis sobre como adaptar CWM[17] a las herramientas de modelado utilizados lo cual no es el objetivo primordial del presente trabajo.

2.4.3 Conclusión sobre el uso de CWM

Al comenzar este trabajo se intento basar el desarrollo del mismo en el estandard CWM[17] de la OMG[1] por ser nuestro dominio de aplicación algo tan común. Al no haber una implementación para el mismo la primera intención fue realizar una implementación y luego utilizarla para desarrollar este trabajo. En este punto surgieron varios inconvenientes que hacían que la mayor parte del tiempo debia ponerse todo el esfuerzo en como adaptarse a CWM[17]. Debido a eso se decidió utilizar metamodelos mas acotados que los que propone CWM[17] . La compatibilidad con CWM[17] estará dada por transformaciones desde los modelos que generamos con nuestra herramienta para generar modelos compatibles con y viceversa. Esto último será desarrollado en el futuro y en este trabajo nos concentraremos en el desarrollo en base a los metamodelos propuestos en [14] y dejando la compatibilidad de nuestra herramienta con CWM[17] para ser realizada en el futuro.

Capítulo 3

3 Marco general de Desarrollo

Cada modelo que participa del presente trabajo consta de la estructura que se detalla a continuación y cada modelo fue tomado del trabajo de [14], en donde cada uno forma parte de una serie de pasos destinados a formalizar la generación de un datawarehouse. A continuación se brinda un esquema del criterio en que fueron implementados y transformados unos en otros. Aprovechamos esta sección para introducir el simbolismo utilizado en el trabajo para expresar los artefactos que forman parte del desarrollo(modelos de dominio, definiciones de diagrama, transformaciones,etc). En algunos casos se han agregado modelos por cuestiones inherentes a la implementación y que se describen a continuación y en otros casos porque se consideró que eran útiles y aportaban características que agregan valor.

3.1 Modelos de referencia

Están basado casi estrictamente en los modelos del trabajo [14] y que se describieron en el capítulo anterior y son construidos como un DSL y utilizando la implementación de Eclipse Modeling Project[20] de los estándares de la OMG[1].

3.2 Implementación de modelos

Brindamos un esquema general del diseño de la solución implementada de cada modelo que participa en la generación del datawarehouse, también se muestra la simbología utilizada en los capítulos siguiente para describir la estructura de cada modelo.

Por cada modelo definimos las siguientes características:

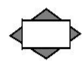
Notaciones abstractas:

- *Modelos de dominios(ecore):*
  Implementados a través del lenguaje ecore y tomados como origen para derivar el código java de los modelos de dominio.

- *Notaciones de diagramas, figuras y herramientas(gmf)*
  Para definir figuras y aspectos visuales.

- *Notaciones textuales(xtxt):*
  Gramáticas de tipo EBNF que nos describen nuestros lenguajes textuales para para manipular nuestros modelos de dominio.

Notaciones concretas

- *Definiciones de diagramas(gmfmap, genmodel, gmfgen)* 

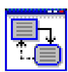
Combinación del modelo de domino con aspectos visuales para ser tomados como base en la edición gráfica de los modelos. A partir de ellos derivamos el código fuente de los editores.

Código derivado

- *Modelo de dominios:* Basados en EMF que implementa el patrón observer y son utilizados para construir editores.
- *Editores EMF:* Editores basados en la estructura del modelo de dominio
- *Editores GMF:* Editores acorde a la semántica de modelo muy potentes. Son el resultado de combinar los aspectos visuales, y el modelo de dominio y otra información.
- Lenguajes visuales: Lenguajes desarrollados para desarrollar instancias del dominio específicos(DSLs)

3.3 Indice de modelos y transformaciones implementadas

Esta sección tiene el objetivo de ser un indice de los modelos, metamodelos y transformaciones que forman parte del proceso . En los capítulos que siguen se dan detalles sobre cada modelo y las transformaciones implementada.

3.3.1 Modelo Entidad-Relación(Capítulo 4) 

✓ *Metamodelo*: **MDS**

***Notaciones* abstractas**

- *mds.ecore*: modelo de dominio
- *mds.gmfgraph*: definición de figuras
- *mds.gmftool*: definición de herramientas

***Notaciones* concretas**

- *mds.genmodel:* Definición que nos permite especificar características del código a derivar de nuestro modelo de dominios
- *mds.gmfmap:* Definición que nos permite asociar las gramáticas abstractas y obtener nuestro editor

Código derivado

- *mds.diagram*(GMF): Editor visual de diagramas de entidad relación
- *mds.editor*(ERMF): Editor visual mas rudimentario basado en la estructura del modelo de dominios y no en su “aspecto semántico”

Extensiones desarrolladas

- *Validaciones*
  - OCL
  - Java basadas el modelo de dominio generado
- *Correspondencia entre gramáticas visuales y textuales*

3.3.2 Lenguaje de reestructuración de Entidad-Relación(Capitulo 5)

✓ Metamodelo RMDS

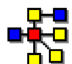
Notaciones abstractas
- *rmds.ecore:* modelo de dominio
- *rmds.xTxt* : Gramática EBNF para definir el lenguaje textual

Notaciones concretas
- *mds.genmodel:* Definición que nos permite especificar características del código a derivar de nuestro modelo de reestructuración.

Código derivado
- *rmds.editor:* Editor basado en la estructura de nuestro lenguaje. Se toma como base para ejecutar la reestructuración.

Extensiones desarrolladas:
- *Correspondencia entre el lenguaje textual y visual de reestructuraciones*
- *Wizards para realizar la conversión*

3.3.3 Grafo de Atributos (Capítulo 6)



✓ *Metamodelo*: **GRAPH**

Notaciones abstractas
- *graph.ecore*: modelo de dominio
- *graph.gmfgraph*: definición de figuras
- *graph.gmftool*: definición de herramientas

Notaciones concretas
- *graph.genmodel:* Definición que nos permite especificar características del código a derivar de nuestro modelo de dominios
- *graph.gmfmap:* Definición que nos permite asociar las gramáticas abstractas y obtener nuestro editor

Código derivado
- *graph.diagram*(GMF): Editor visual de diagramas de entidad relación
- *graph.editor*(ERMF): Editor visual mas rudimentario basado en la estructura del modelo de dominios y no en su "aspecto semántico"

Extensiones desarrolladas
- *Validaciones*

3.2.4 Modelo Multidimensional temporal(Capítulo 7)

✓ *Metamodelo*: **MMT**

Notaciones abstractas
- *mmt.ecore*: modelo de dominio
- *mmt.gmfgraph*: definición de figuras
- *mmt.gmftool*: definición de herramientas

Notaciones concretas
- *mmt.genmodel:* Definición que nos permite especificar características del código a derivar de nuestro modelo de dominios

- *mmt.gmfmap:* Definición que nos permite asociar las gramáticas abstractas y obtener nuestro editor

Código derivado

- *mmt.diagram*(GMF): Editor visual de diagramas de entidad relación
- *mmt.editor*(ERMF): Editor visual mas rudimentario basado en la estructura del modelo de dominios y no en su "aspecto semántico"

Extensiones desarrolladas

- *Validaciones*

3.2.5 Modelo relacional(Capítulo 8)

✓ *Metamodelo*: **MREL**

Notaciones abstractas

- *mrel.ecore*: modelo de dominio
- *mrelt.gmfgraph*: definición de figuras
- *mrel.gmftool*: definición de herramientas

Notaciones concretas

- *mrel.genmodel:* Definición que nos permite especificar características del código a derivar de nuestro modelo de dominios
- *mrel.gmfmap:* Definición que nos permite asociar las gramáticas abstractas y obtener nuestro editor

Código derivado

- *mrel.diagram*(GMF): Editor visual de diagramas de entidad relación
- *mrel.editor*(ERMF): Editor visual mas rudimentario basado en la estructura del modelo de dominios y no en su "aspecto semántico"

Extensiones desarrolladas

- *Validaciones*

3.3 Implementación de transformaciones

3.3.1 Reestructuración ER(Capítulo 5)

Versión Automática basada en la propuesta de [14]

Modelos origen:mds, rmds
Modelo destino: mds
Lenguaje: atl

Versión Manual

Modelo origen: mds, rmds
Modelo destino: rmds
Lenguaje: atl

3.3.2 ER (adaptado) al Grafo de atributos(Capítulo 6)

Modelo origen: mds, rmds
Modelo destino: graph
Lenguaje: atl

3.3.3 Grafo de atributos al modelo multidimensional(Capítulo 7)

Modelo origen:graph
Modelo destino:mmt
Lenguaje: atl

3.3.4 Modelo multidimensional al modelo relacional(Capítulo 8)

Modelo origen: graph
Modelo destino: mmt
Lenguaje: atl

3.3.5 Modelo relacional a código(Capítulo 8)

Modelo origen: mrel
Modelo destino: texto
Lenguaje: xPand

3.4 Extensiones desarrolladas

En esta sección se explican aquellas funcionalidades que no forman parte del desarrollo basado en modelos. En ellas hemos tenido que programar en java algunas funcionalidades para facilitar la tarea diseñador que utiliza la herramienta. En esta sección se brinda un panorama general de la tecnología utilizada para programar estas extensiones y en cada capítulo se dan detalles puntuales sobre el uso de que se le dio a cada una.

3.4.1 Validaciones en OCL y Java

Mediante el framework de validación que nos provee la plataforma eclipse(Validation Framework) se han desarrollado una serie de constraints para verificar la validez de los modelos que forman parte de nuestra herramienta. Estas validaciones sirven para validar la consistencia de los modelos que creamos y a su vez sirven para chequear que nuestros modelos sean aptos para ser utilizados como entrada para las diferentes transformaciones que forman parte de nuestra herramienta y principalmente aquellas que implementarán el proceso de generación del datawarehouse.

Las validaciones pueden ser realizadas de las siguientes maneras:

- ✓ *Batch*: Ejecutadas de manera explicita y de a lotes.
- ✓ *Live*: Ejecutadas ante la ocurrencia de algún evento.

Pueden ser de distinta gravedad:

- ✓ *Problemas:* Son errores graves que deben ser corregidos a fin de que nuestro modelo sea considerado válido.
- ✓ *Advertencias:* Son sugerencias a fin de que nuestro modelo sea mas correcto.

y pueden ser implementadas en

- ✓ *Java:* Están basadas en el modelo de dominio en lenguaje java derivado de nuestro modelo de dominio en lenguaje ecore(mrel.ecore). Las mismas son clases java que validan
- ✓ *OCL* **:** Están basadas en nuestro modelo de dominios y las validaciones las llevamos a cabo implementando invariantes del lenguaje OCL sobre nuestros modelos de dominio.

Debido a lo familiar del lenguaje java y a fin de utilizar el código derivado automáticamente, las validaciones fueron realizadas mayormente en java. Otras fueron realizadas en Ocl a fin de investigar y comparar ambas opciones. El framework de validación es muy potente y esta diseñado para soportar ambas opciones e incluso para ser extendido y construir un mecanismo propio de validaciones. Los reportes de errores podrán ser visualizados en la vista de problemas de la plataforma eclipse o a través de diálogos de error.

Figura 3.1 Reporte de una constraint fallidas validadas en forma batch

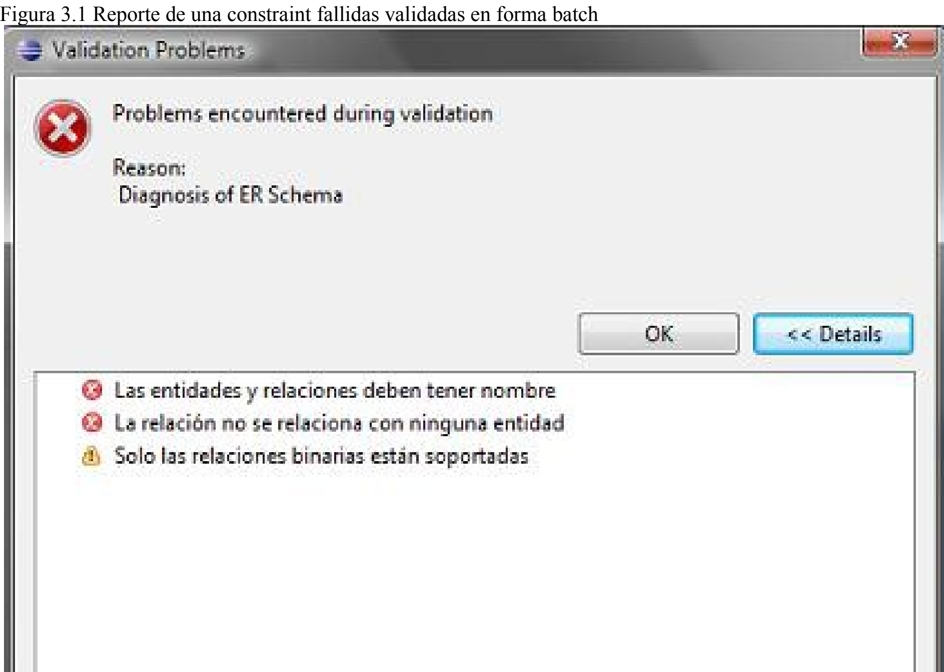


Figura 3.2 Registro de errores en la vista de problemas que provee la plataforma

Problems | Javadoc | Declaration | Error Log | Properties

3 errors, 2 warnings, 0 others

Description

Errors (3 items)

La relación no se relaciona con ninguna entidad

Las entidades y relaciones deben tener nombre

Las entidades y relaciones deben tener nombre

En cada capitulo se detallan algunas de las constraints implementadas, su forma de implementación(java u ocl), su forma de validación y detalles sobre su desarrollo.

3.4.2 Equivalencia entre notaciones basadas en Ecore y xText

Basadas en EMF Resources. EMF Resources es un mecanismo que nos que brinda EMF de leer un archivo con un modelo y serializarlo en un modelo conforme a Ecore. Esto se utilizó para transformar DSLs textuales basados en xText en modelos DSLs visuales basados en ecore que sirven como entrada para nuestras tranformaciones. En el capítulo 4 se detalla como se utilizó esta tecnología en un caso puntual.

3.4.3 Ejecución de transformaciones ATL programaticamente

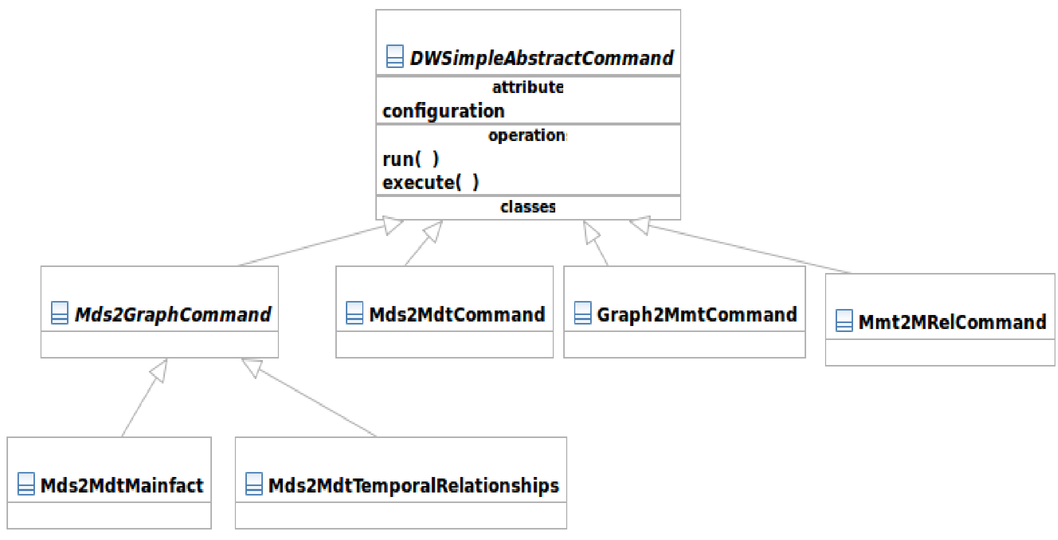
Con el objetivo de poder ejecutar las transformaciones ATL que forman parte de las transformaciones que ejecuta el diseñador fue necesario utilizar código fuente de ATL a fin de poder ejecutarlas programaticamente y de manera transparente al diseñador. Para esto hemos tenido que utilizar el código fuente de ATL, incluirlo en nuestro desarrollo y utilizarlo. Esta tarea se facilita gracias a la arquitectura de plugins de la plataforma eclipse, y que el código fuente de ATL esta disponible para utilizarlo y eventualmente modificado. Esto se lleva a cabo a través de un serie de comandos implementados en java que invocan al engine de ATL. La clase de ATL que se invoca para invocar las transformaciones programaticamente es

org.eclipse.m2m.atl.adt.launching.AtlRegularVM, e invocamos a ATL de la siguiente manera:

AtlRegularVM.runAtlLauncher(TRANSFO1_PATH, libsFromConfig, input, output, path, modelType, modelhandler, mode, superImpose, options);

A continuación mostramos un breve esquema UML con los comandos utilizados para invocar al engine de ATL.

Figura 3.3 Algunas clases importantes utilizadas para invocar al engine de ATL



El plugin que implementa la invocación al engine de ATL es y donde están las clases del diagrama UML anterior es: ***ar.edu.unlp.info.baez.launching***

> *Aclaración: En este trabajo hemos utilizado la versión 2,0 de ATL. En la versión 3 de ATL la manera de invocar al engine de ATL cambia. En el futuro será necesario migrar este código para que sea compatible con la versión 3 de ATL.*

La información sobre como invocar al engine de ATL puede obtener en la siguiente ubicación: http://wiki.eclipse.org/ATL/Developer_Guide

3.4.4 Launchers para lanzar transformaciones

Utilizando el mecanismo que nos provee la plataforma eclipse llamado Launching Framework se ha desarrollado un soporte para la creación de launchers que el diseñador puede configurar para ejecutar las transformaciones implementadas.

Se ha desarrollado de esta manera un launcher que ejecuta las transformaciones necesarias a fin de obtener el datawarehouse. Partimos de un modelo de Entidad-Relación reestructurado y realizamos la serie de transformaciones necesarias y que mas adelante se describen. El launcher utiliza el plugin ***ar.edu.unlp.info.baez.launching*** descripto anteriormente para invocar aquellas transformaciones basadas en ATL.

Se ha desarrollado el launcher de manera tal que el diseñador seleccione que transformación desea ejecutar y que modelos fuente y target desea crear. Esta tarea se realiza a través de una serie de tabs donde el diseñador configura cada transformación. A continuación mostramos una imagen del tab principal del launcher y en cada capítulo donde se describe una transformación se muestra un ejemplo del tab correspondiente a la transformación.

Figura 3.4 La creación de launchers en eclipse con la posibilidad de crear uno para las trasformaciones para crear el datawarehouse

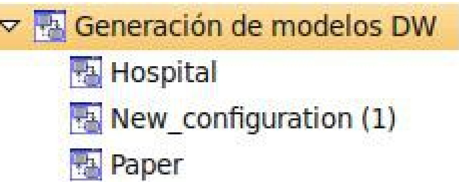
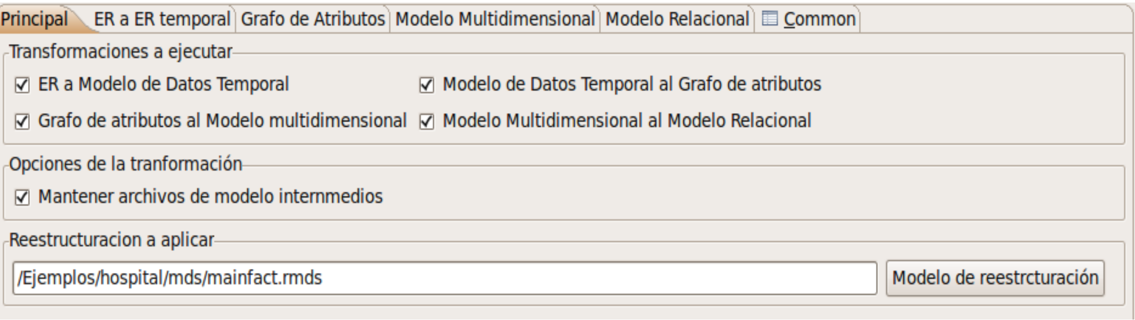
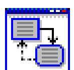


Figura 3.5 Tab principal para la creación de launchers en eclipse con la posibilidad de crear uno para las trasformaciones para crear el datawarehouse



Capítulo 4

4 Modelado de Entidad-Relación 

En este capítulo se describen los conceptos de modelo de Entidad-Relación que hemos tomado como base conceptual, el metamodelo de Entidad-Relación implementado en respuesta al propuesto en [14], las notaciones abstractas y concretas implementadas y utilizadas luego para derivar el código para los editores visuales.

4.1 Modelos de Entidad-Relación - Una breve definición

Es un modelo de diseño conceptual de datos. Un modelo conceptual está orientados a la descripción de estructuras de datos y restricciones de integridad. Se usan fundamentalmente durante la etapa de Análisis de un problema dado y están orientados a representar los elementos que intervienen en ese problema y sus relaciones. Un diagrama o modelo Entidad-Relación (a veces denominado por su siglas, E-R ,"Entidad-Relación", o, "DER" Diagrama de Entidad Relación) es una herramienta para el modelado de datos de un sistema de información. Estos modelos expresan entidades relevantes para un sistema de información así como sus interrelaciones y propiedades.

4.2 Diagramas de Entidad-Relación

El Modelo Entidad-Relación,fue propuesto por Peter Chen en 1976 [8] y desde entonces ha gozado de un gran aceptación en el ámbito de las bases de datos. Es un modelo de datos semántico utilizado para expresar el diseño conceptual de un almacén de datos. Se ha de visualizar los objetos que pertenecen a la Base de Datos como entidades (se corresponde al concepto de clase, cada tupla representaría un objeto, de la Programación Orientada a Objetos) las cuales tienen unos atributos y se vinculan mediante *relaciones*. Es una representación conceptual y gráfica de la información. Mediante una serie de procedimientos se puede pasar del modelo de Entidad-Relación a otros, como por ejemplo el modelo relacional".

Figura 4.1 Una de las tantas representaciones gráficas para el modelo propuesto por Chen

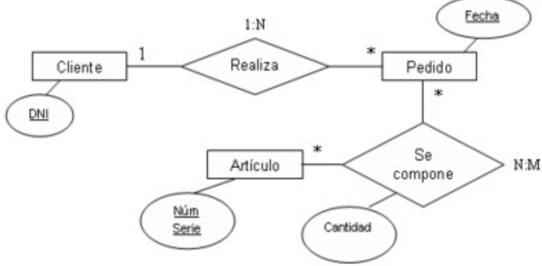
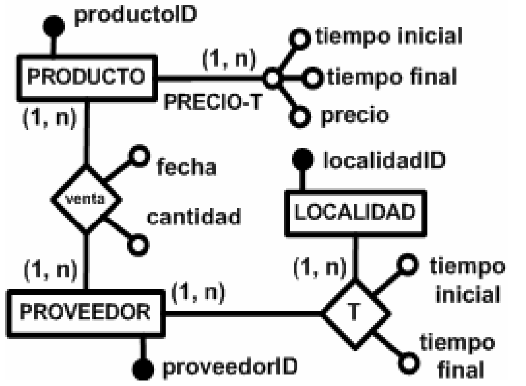


Figura 4.2 Modelos ER utilizados para documentar en el trabajo [14]



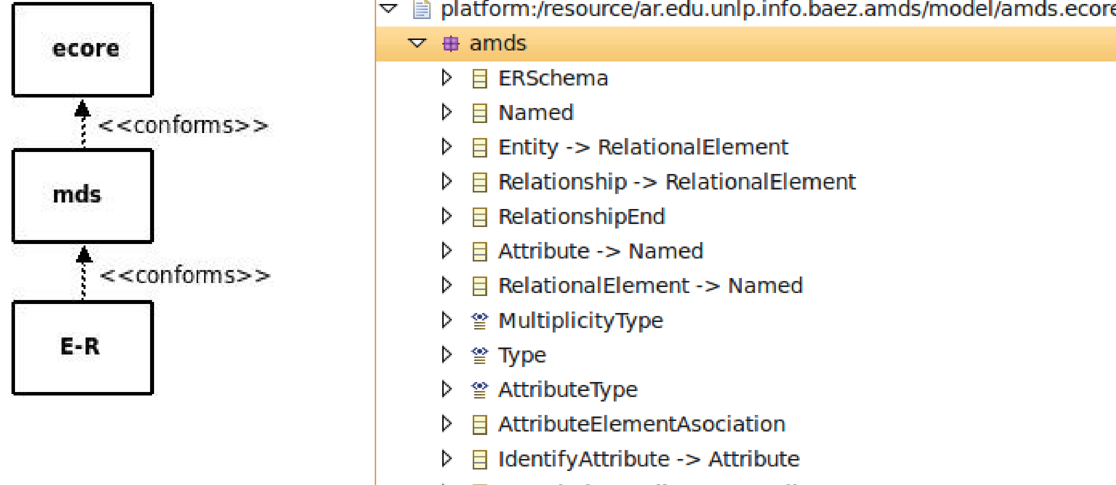
4.3 DSL de Entidad-Relación - construcción

4.3.1 Metamodelo entidad relación o modelo de datos simple(mds)

El metamodelo de Entidad-Relación nos sirve para definir un modelo de Entidad-Relación. El metamodelo que hemos utilizado es el propuesto en [14] y se ha adoptado el mismo casi en su totalidad. Se han debido hacer unas modificaciones al mismo para salvar la diferencia entre que se señala en 1.3.1.

Se adopta el mismo para ser utilizado para construir el editor y de esta manera a partir de el se desarrollan las gramáticas abstractas y concretas que utilizamos para derivar código de nuestros editores. De esta manera los modelos de Entidad-Relación que se obtienen serán conformes al metamodelo mds. El metamodelo mds es especificado en el lenguaje ecore.

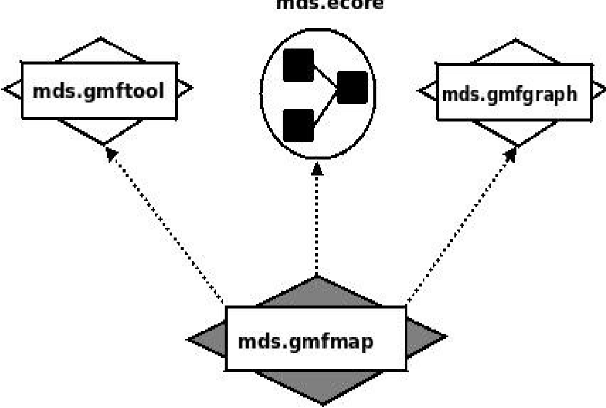
Figura 4.3 Modelo ER y sus metamodelos Figura 4.4 Implementación del mds en ecore



4.3.2 Notaciones

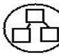
La implementación de nuestro dsl consta de varias notaciones abstractas y varias concretas desde las cuales derivamos código para generar los editores gráficos.

Figura 4.5 Esquema de las notaciones abstractas y concretas



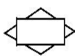
4.3.2.1 Notaciones abstractas de Entidad-Relación y gráficas

Notaciones abstractas definidas por la herramienta EMF:

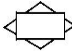


- ✔ *mds.ecore*:Modelo de dominios en lenguaje e-core, y con el que implementamos el metamodelo propuesto en [14]**.** El mismo casi no ha sido modificado a los fines de la implementación. La definición del modelo de dominio en ecore nos brinda la ventaja de poder tener varias implementaciones conformes al mismo.

Notaciones abstractas definidas por la herramienta GMF:



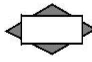
- ✔ *mds.gmgraph*: Es una definición de diagrama que especifica un conjunto de figuras y sus relaciones



- ✔ *mds.gmftool*: Es una definición de diagrama que define un conjunto de herramientas en un toolbar

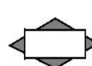
4.3.2.2 Notaciones concretas de Entidad-Relación y generación de código

Definidas por la herramienta EMF



- ✔ *mds.gemodel*: Mediante la herramienta de eclipse EMF hemos se desarrollado está gramática para indicar como convertir nuestro modelo de dominios(mds.ecore) en clases java que podrán ser utilizadas para crear instancias del modelo mds

Definidas por la herramienta GMF:



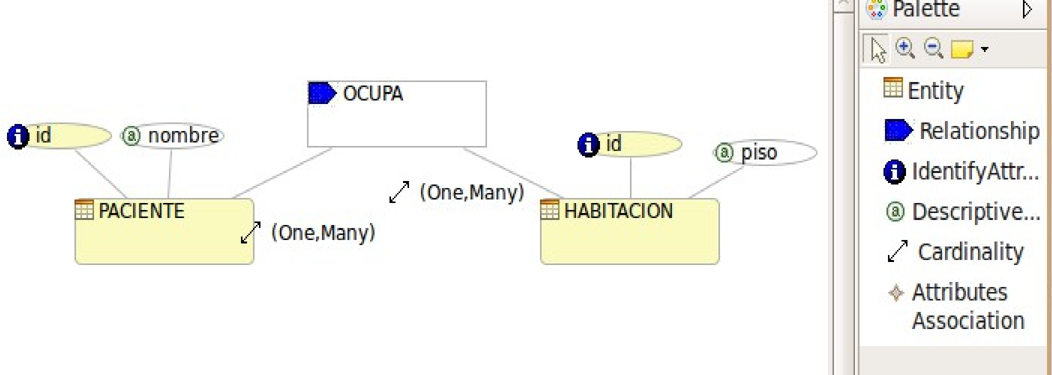
- ✔ *mds,gmfmap*: Mediante la herramienta de eclipse GMF se ha desarrollado una notación concreta que nos permite combinar las notaciones abstractas anteriores. En la misma se asocian elementos del modelo de dominio con los objetos gráficos y los elementos de la barra de herramientas. Esto se toma como base para derivar el código de nuestro editor gráfico.

4.3.2.3 Código derivado

4.3.2.3.1 Un editor de Entidad-Relación

Valiéndose las notaciones concretas anteriores derivamos código java para nuestro modelo de dominios a través de *mds.gemodel* y luego un editor gráfico para crear instancias de nuestros modelos a través de *mds.gmfmap*. Para ello utilizamos los mecanismos que nos brinda la herramienta GMF en conjunto con el lenguaje XPAND para generar código a partir de los modelos anteriores. De esta manera obtenemos un editor gráfico para desarrollar instancias del modelo mds.

Figura 4.6 Ejemplo del editor entidad-relación



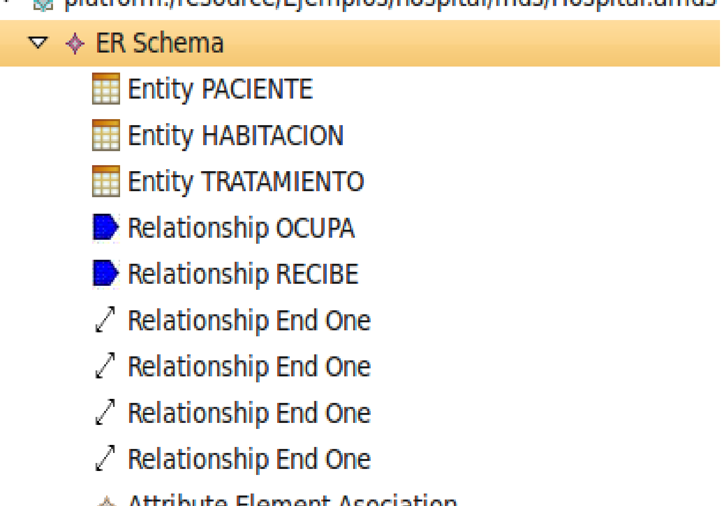
El plugin que implementa nuestro editor es:

- ✔ **ar.edu.unlp.info.baez.amds.diagram**

4.3.2.3.2 Un editor EMF del modelo basado en su estructura

Valiéndose del modelo de dominio en ecore podemos obtener a través de la plataforma EMF un editor basado en la estructura del modelo que le permitirá al diseñador editar y crear instancias del modelo relacional. Es un editor mas rústico y limitado que el editor anterior basado en GMF.

Figura 4.7 Un ejemplo del editor EMF basado en la estructura del modelo de Entidad-Relación



4.3.2.4 Extensiones desarrolladas para los modelos ER

4.3.2.4.1 Validaciones

Mediante el framework de validación que nos provee la plataforma eclipse y utilizando los mecanismos descriptos en 3.4 se ha desarrollado un plugin que forma parte de nuestro esquema de entidad-relación y que tiene como objetivo validar la consistencia de nuestros modelo de Entidad-Relación. Esto es para tener la certeza de que nuestros modelos son válidos y para que sean aptos para ser utilizados como modelos fuente para las reestructuraciones que debemos hacerle al modelo para que forman parte del proceso de generación del datawarehouse.

Las validaciones sobre el modelo de Entidad-Relación pueden ser realizadas de las siguientes maneras:

- ✔ *Batch*: Ejecutadas de manera explicita y de a lotes.

y pueden ser implementadas en

- ✔ *Java:* Están basadas en el modelo de dominio en lenguaje java derivado de nuestro modelo de dominio en lenguaje ecore(mds.ecore). Las clases java que implementan nuestras constraints realizan la validación sobre nuestro modelo de dominio en java derivado de nuestro modelo de dominio que forma parte de la sintaxis abstracta(mrel.ecore).
- ✔ *OCL:* Están escritas utilizando el soporte de la herramienta EMF para escribir invariantes que nuestro modelo debe cumplir

Constraints implementadas

- ✔ *Las entidades deben poseer nombres*

Al ser las tablas objetos de tipo Named, estos si o si deben poseer nombre.

- ✔ *Las relaciones deben poseer nombres*

Al ser las tablas objetos de tipo Named, estos si o si deben poseer nombre.

- ✔ *El esquema no puede estar vació.*

Si o si deben existir al menos una tabla. La colección llamada tables del objeto Schema de nuestro modelo de dominios no puede ser vacía.

- ✔ *Las entidades deben poseer al menos algún atributo.*

No pueden existir tablas si atributos en nuestro modelo de dominio,

el objeto Schema de nuestro modelo de dominio posee una colección llamada tables esta no puede ser vacío.

- ✔ *Los atributos deben poseer nombre y tipo.*

Si o si debemos asignarle un nombre valido a los atributos de nuestro esquema y estos deben tener un tipo de datos asignado.

- ✔ Las entidades deben tener al menos un atributo identificador(esta es una advertencia)

- ✔ Las entidades deben estar conectadas mediante relaciones(esta es una advertencia)

El plugin que implementa las validaciones en java es:

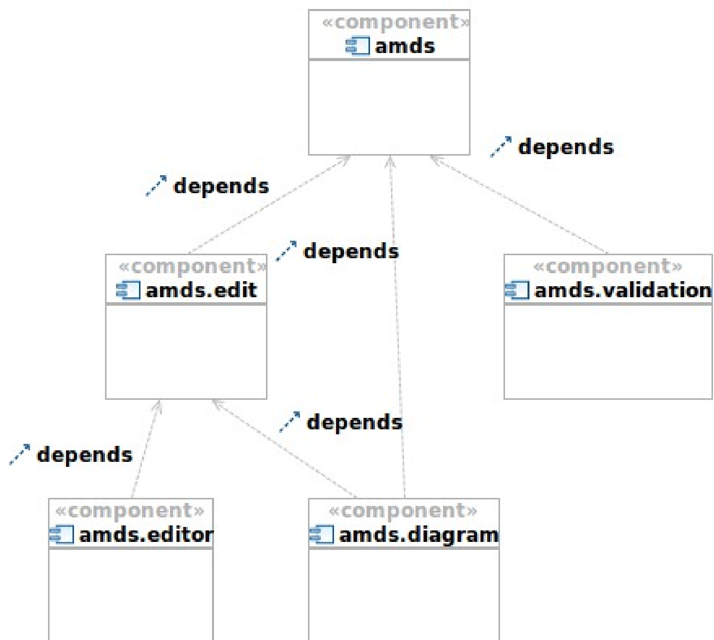
- ✔ **ar.edu.unlp.info.baez.amds.validation.general**

4.3.2.5 Plugins que forman parte del DSL de Entidad-Relación

Para resumir es necesario dejar expresado que nuestro conjunto de plugins fueron obtenidos en base a la derivación de código y la adaptación de nuestras notaciones concretas y a agregados que fueron programados. Con ellos podemos crear, manipular y validar modelos relacionales conformes a nuestro modelo de dominio y un conjunto de ellos forman parte del soporte visual que poseemos para la creación y manipulación de nuestra notación concreta visual.

En la figura 4.8 se muestra un esquema del conjunto de plugins que dan el soporte para edición visual del modelo relacionales.

Figura 4.8 esquema de componentes y sus dependencias



4.4 DSL de Entidad-Relación - uso

Desde el punto de vista del diseñador que utiliza nuestra herramienta, el mismo creará modelos de Entidad-Relación utilizando el desarrollo dirigido por modelos. Utilizando el editor creará modelos que se serializan acorde a una notación abstracta(basadas en xmi y conformes a nuestro modelo de dominios en ecore, mrel.ecore) , por convención adoptamos la extensión ".amds" para los archivos que contienen la notación abstracta de Entidad-Relación. Tener estas notaciones abstractas basadas en XMI[27] y conforme a nuestros metamodelos en ecore es particularmente útil si queremos que nuestras instancias del modelo Entidad-Relación sean utilizadas como para ser transformadas,o serializadas de manera estandard para ser leído por otra herramienta. En estos casos la información que contiene aspectos visuales no es necesaria.

También a través del editor se serializa una notación concreta que posee aspectos visuales que toma el editor para facilitar la edición. Por convención adoptamos la extensión "amds_diagram" para la notación concreta visual.

De esta manera la según la visión del diseñador la notación abstracta posee el modelo relacional y una notación concreta posee aspectos visuales. Pudiendo existir mas de una notación concreta para la misma notación abstracta.

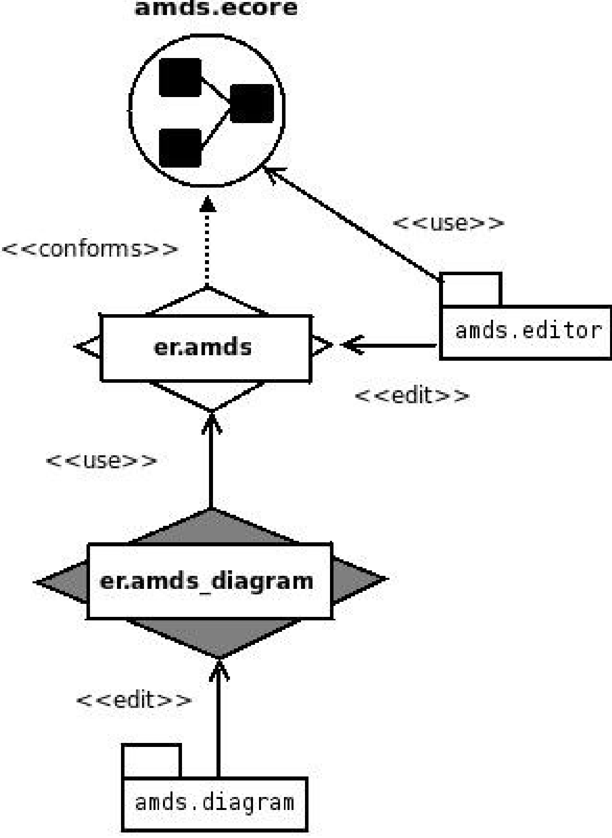
Figura 4.9 Vista de los archivos de notación abstracta y concreta que utiliza el diseñador

Hospital.amds
Hospital.amds_diagram

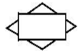
4.4.1 Notaciones

Al igual que los modelos y metamodelos utilizados para definir las notaciones abstractas y concretas que forman parte de nuestra herramienta, el diseñador que utiliza dicha herramienta desarrolla y manipula modelos de Entidad-Relación modificando notaciones abstractas y concretas para representar el modelo relacional.

Figura 4.10 relación entre los editores,las notaciones concretas, abstractas y el metamodelo desde el punto de vista del diseñador.



4.4.1.1 Notación abstracta de Entidad-Relación

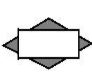


- ✔ *amds*: Consta de un archivo XML, que contiene la serialización del modelo relacional en formato XMI[27] que nos representa las instancias del modelo de entidad-relación.

Figura 4.11 Un ejemplo de la notación abstracta relacional

```
<?xml version="1.0" encoding="UTF-8"?>
<amds:ERSchema xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:amds="http://baez/mds/info/unlp/edu/ar">
  <entities name="PACIENTE"
  attributes="//@attributeAssociations.0 //@attributeAssociations.2"/>
  <entities name="TRATAMIENTO" attributes="//@attributeAssociations.1"/>
  <attributeAssociations
  attribute="//@attributes.0" relationalElement="//@entities.0"/>
  <attributeAssociations
  attribute="//@attributes.1" relationalElement="//@entities.1"/>
  <attributeAssociations
  attribute="//@attributes.2" relationalElement="//@entities.0"/>
  <attributes xsi:type="amds:IdentifyAttribute"
  name="id" owner="//@attributeAssociations.0"/>
  <attributes xsi:type="amds:IdentifyAttribute"
  name="id" owner="//@attributeAssociations.1"/>
  <attributes xsi:type="amds:DescriptiveAttribute"
   name="nombre" owner="//@attributeAssociations.2" type="TEXT"/>
</amds:ERSchema>
```

4.4.1.2 Notación concreta de Entidad-Relación

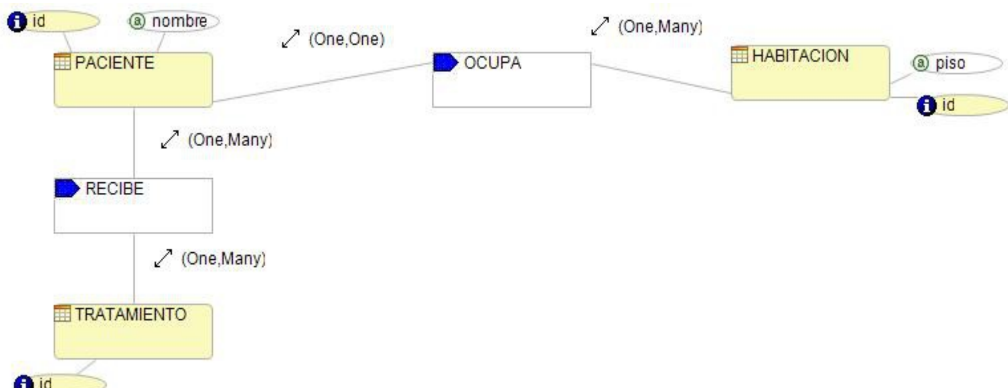


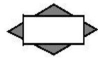
✔ *amds_diagram:* Consta de un archivo XML en formato XMI que nos representa un conjunto de figuras asociadas a la notación abstracta anterior. Son creados y modificados en base a nuestro editor gráfico relacional. Por cada elemento de la notación abstracta nos indica la figura que lo representa y su posición.

Figura 4.12 Un ejemplo de la notación concreta gráfica

```
<?xml version="1.0" encoding="UTF-8"?>
<amds:ERSchema xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:amds="http://baez/mds/info/unlp/edu/ar">
  <entities name="PACIENTE"
  attributes="//@attributeAssociations.0 //@attributeAssociations.2"/>
  <entities name="TRATAMIENTO" attributes="//@attributeAssociations.1"/>
  <attributeAssociations
  attribute="//@attributes.0" relationalElement="//@entities.0"/>
  <attributeAssociations
  attribute="//@attributes.1" relationalElement="//@entities.1"/>
  <attributeAssociations
  attribute="//@attributes.2" relationalElement="//@entities.0"/>
  <attributes xsi:type="amds:IdentifyAttribute"
  name="id" owner="//@attributeAssociations.0"/>
  <attributes xsi:type="amds:IdentifyAttribute"
  name="id" owner="//@attributeAssociations.1"/>
  <attributes xsi:type="amds:DescriptiveAttribute"
   name="nombre" owner="//@attributeAssociations.2" type="TEXT"/>
</amds:ERSchema>
```

Figura 4.12 Un ejemplo del editor de una concreta gráfica similar a la anteior





- ✔ *Editor basado en la estructura del modelo:* Podemos ver de manera visual y concreta la notación abstracta relacional utilizando el editor EMF basado en la estructura del modelo. En la figura 4.7 se observa un ejemplo de este editor. Su principal utilidad es que podemos utilizarlo para manipular un modelo relacional si tener el editor anterior instalado, Si bien esto mas complicado que hacerlo que con el editor anterior basado en GMF es mucho mas fácil que modificar la sintaxis abstracta manualmente.

4.4.1.3 ¿Notación textual?

Existen muchas herramientas en el mercado que brindan soporte para el desarrollo de base de datos. Por lo general dichas herramientas no están basadas en el desarrollo dirigido por modelos y utilizan una notación gráfica propia para los diagramas relacionales que imposibilita que sus diagramas sean exportados a un formato estandard(como CWM por ejemplo) para ser utilizados por otras herramientas. Debido a la madurez de las base de datos relacionales ,la estructura gráfica de los diagramas relacionales entre las diferentes herramientas a pesar de su informalidad es bastante uniforme y las diferencias son menores. Esta manera graficar los modelos de Entidad-Relación es bastante común y un camino natural para el diseño conceptual. Es por eso que no existen notaciones textuales ampliamente utilizadas para especificar la estructura de los modelos relacionales. Ese es el motivo por el cual se decidió no implementar un DSL textual para diseñar modelos de entidad-relación a pesar de tener la base suficiente para crear un lenguaje en el cual especificar un modelo relacional.

Capítulo 5

“Todos los aspectos temporales deberían ser ingnorados durante la mayor parte del tiempo de diseño conceptual. Un esquema ER deberia ser construido sin aspectos temporales. Una vez que el esquema de Entidad-Relacion ha sido construido en su totalidad las consideraciones temporales deben ser llevadas a cabo posteriormente[30].”

5 Lenguajes de reestructuración

Nuestra propuesta este la creación de un DSL para especificar estos aspectos temporales. Mendiante nuestro lenguaje podremos especificar el tiempo valido[31] de una entidad o relación.

Presentamos en este capítulo un lenguaje de especificaciones para indicar acciones de reestructuración a llevar a cabo en modelos de Entidad-Relación. También aprovecharemos dicho lenguaje para poder especificar las marcas de temporalidad y de hechos de interés que utilizaremos para generar la estructura del datawarehouse descripto en [14]. Presentamos también aquí la descripción de implementación y uso de las transformaciones automáticas para generar el datawarehouse. El uso de un lenguaje textual para reestructurar modelos de Entidad-Relación ha sido analizado en diversos trabajos por ejemplo en [21] se propone un lenguaje basado en ecore denominado LTR(lenguaje de transformación relacional). Aquí se toma la misma idea implementando algunos conceptos de la misma propuesta . Tendremos un lenguaje abstracto similar al de [21] y una notación concreta textual para el mismo.

5.1 Lenguaje de reestructuración del modelo de Entidad-Relación

Consiste en un lenguaje de especificaciones para indicar acciones de reestructuración a llevar a cabo en modelos de Entidad-Relación. El lenguaje nos servirá para que dichas acciones sea ejecutadas con un transformador implementado en este trabajo y que se describe mas adelante. También aprovecharemos dicho lenguaje para poder especificar las marcas de temporalidad y de hechos de interés que utilizaremos para generar la estructura del datawarehouse descripto en [14]. En el futuro seria conveniente que estos dos aspectos fuesen realizados por DSLs separados. En este trabajo por simplicidad se ha decidido que sea un solo DSL. Esta es una diferencia sustancial con la propuesta de [14], donde las marcas son partes del modelo de Entidad-Relación. Aquí hemos decidido que estos dos aspectos estés separados y utilizar el DSL de reestructuraciones para realizar estas marcas. Esto nos brinda la ventaja de tener varias reestructuraciones sobre un mismo modelo de Entidad-Relación. En [14], las marcas de temporalidad y de hechos de interés están “embebidas” en el modelo de Entidad-Relación lo que hace no podamos reestructurar un mismo modelo de varias maneras.

5.2 DSLs de reestructuración - construcción

Se presentan dos versiones equivalentes de nuestro DSL de reestructuración una textual(rmdsDSL) y una visual(rmds). La textual está basada en un gramática definida utilizando xText y desde la cual podemos obtener un ide para programar en dicho lenguaje. También tenemos una versión visual obtenida a partir de un modelo de dominios en ecore(rmds.ecore) equivalente gramática definida en xText. Estos dos DSLs son equivalentes y es posible transformar uno en otro. Solo los modelos de nuestra versión visual basada en ecore del lenguaje de reestructuración son aptos para ser utilizado en la generación del datawarehouse.

Mediante nuestro lenguaje podremos:

- ✔ Especificar reestructuraciones:
  - Agregar Entidades
  - Agregar Relaciones
  - Conectar Entidades con relaciones:
- ✔ Marcar aspectos temporales y de interés para toma de decisiones acorde a [14]
  - Declarar relaciones temporales.
  - Declarar hechos de interés para la toma de decisiones.

5.3 DSL textual de reestructuraciones - construcción

5.3.1 Notaciones

La implementación de nuestro DSL de reestructuración textual denominado rmdsDSL consta de una notación abstracta y un editor para un lenguaje textual para definir reestructuraciones. Lo que nos interesa aquí es definir las reestructuraciones en el lenguaje textual y conforme a rmdsDSL.xTxt y que nos servirá para ser tomado como modelo fuente para las transformaciones que se utilizan para generar el datawarehouse.

5.3.1.1 Notaciones abstracta

Definidas con la herramienta xText

{} **rmdsDSL.xTxt :** Está definida mediante el lenguaje xText. Es una gramática similar a EBNF y en ella definimos la estructura que tendrá nuestro lenguaje. Es un metamodelo de nuestro lenguaje.

Figura 5.2 Notación de reestructuración, versión en xText

```
Refactors :
 (refactors +=Refactor)*;

Refactor: NewEntity | NewRelationship | RemoveEntity |
RemoveRelationship | Type | DeclareMainFact | DeclareTemporal |
ConnectEntityRelationship;

Enum Cardinality : Zero="Zero" | One="One" | Many="Many";

ConnectEntityRelationship:
connect' 'relationship' nameRelationship=ID 'entity' nameEntity=ID '('
        minCard=Cardinality ',' maxCard=Cardinality ')';

 RemoveRelationship: 'removeRelationship' name=ID '{}';

 RemoveEntity: 'removeEntity' name=ID '{}';
```

5.3.1.3 Código derivado

4.7.2.3.1 Un editor visual para el lenguaje textual de reestructuraciones.

Valiéndose de las notaciones concretas anteriores derivamos código java para nuestro Ide a través de los archivos que utiliza xText para generar código java. En ellos se dan datos concretos sobre el ide para nuestro lenguaje textual. Para ello utilizamos los mecanismos que nos brinda la herramienta xText en conjunto con el lenguaje XPAND para generar código a partir de los modelos anteriores. De esta manera obtenemos un ide para desarrollar instancias del modelo de reestructuraciones.

Figura 5.5 Ejemplo del editor generado para nuestro lenguaje textual

```
type String
type Date
type int

newEntity PERSONA{
 property nombre : type string
 property apellido : type string
 property id : type int
}

newEntity DOMICILIO{
 property id : type int
 property direccion : type string

}
```

El plugin que implementa nuestro editor gráfico es:

- ✔ **ar.edu.unlp.info.baez.mds.refactor**

5.4 DSL textual de reestructuraciones - uso

Desde el punto de vista del diseñador que utiliza nuestra herramienta, el mismo especificará reestructuraciones a llevar a cabo en los modelos de Entidad-Relación utilizando el desarrollo dirigido por modelos. Utilizando el editor creará modelos que se serializan acorde a una notación abstracta. Por convención adoptamos la extensión ".rmdsDSL" para los archivos que contienen la notación abstracta de reestructuración. En estos casos la información que contiene aspectos visuales no es necesaria.
De esta manera la según la visión del diseñador la notación abstracta posee el modelo de reestructuración.

Figura 5.6 Vista de la notación abstracta que utiliza el diseñador

mainfact.rmds

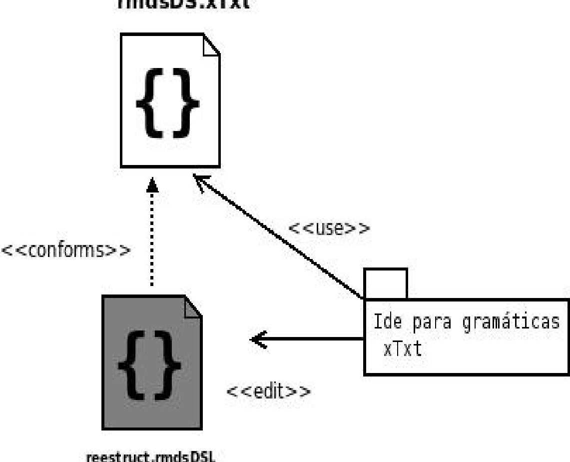
mainfact.rmdsDsl

.

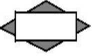
5.4.1 Notaciones

El diseñador manipula el DSL textual a través del editor derivado en 5.3.1.3 modificando por medio de este la gramática abstracta . Es decir que la gramática concreta esta implícita y es la visión que tiene el diseñador a través del editor que se describe en 5.3.1.3 Es un caso similar a los editores EMF basados en la estructura de los modelos. Es decir nos dan una visión concreta de la notación abstracta "en memoria", y esta notación concreta no se serializa. Esto ocurre generalmente en DSLs textuales que no poseen aspectos gráficos(ubicación y colores para guardar). La gramática abstracta(rmdDSL) textual podrá ser convertida en la notación abstracta basada en ecore descripta en 5.5.2.1(rmds).

Figura 5.7 relación entre los editores las notaciones concretas, abstractas y la gramática abstracta desde el punto de vista del diseñador.



5.4.1.1 Notaciones concretas



✔ *Ide para gramáticas basadas en xText, un editor para nuestro lenguaje:* Podemos ver de manera visual y concreta la notación abstracta de

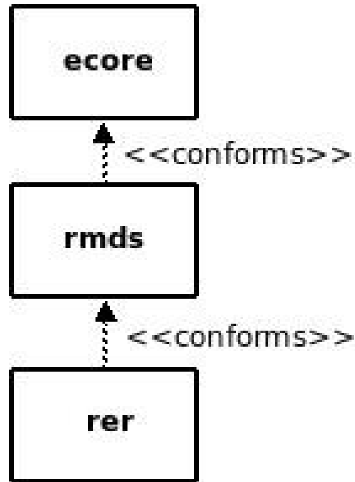
reestructuración utilizando el editor para lenguajes definidos utilizando xText. En la figura 7.14 se observa un ejemplo de este editor.

5.5 DSL visual de reestructuraciones - construcción

5.5.1 Metamodelo de reestructuración - rmds

Para implementar la reestructuración del modelo de Entidad-Relación se propone un DSL visual equivalente al textual para especificar reestructuraciones a nuestros modelos de Entidad-Relación así como para especificar los aspectos temporales que se desean preservar. De esta manera quien utiliza nuestro editor podrá proponer distintas reestructuraciones o decisiones de temporalidad y obtener distintos diagramas reestructurados para un mismo diagrama origen.

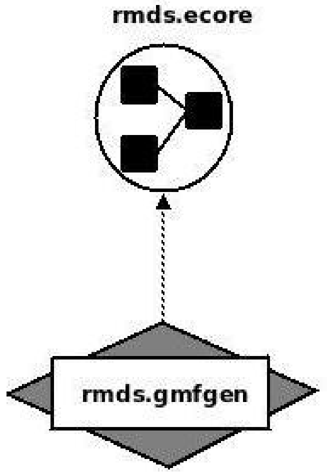
Figura 5.8 Relación entre el RER y sus metamodelos



5.5.2 Notaciones

La implementación de nuestro DSL de reestructuración visual consta de dos gramáticas abstractas y varias concretas desde las cuales derivamos código para generar un editor gráfico. Lo que nos interesa aquí es definir las reestructuraciones en el lenguaje textual conforme a rmdsDSL.xTxt y luego convertirlo en un modelo conforme a ecore que es rmds.ecore el cual nos servirá para ser tomado como modelo fuente para las trasformaciones que se utilizan para generar el datawarehouse.

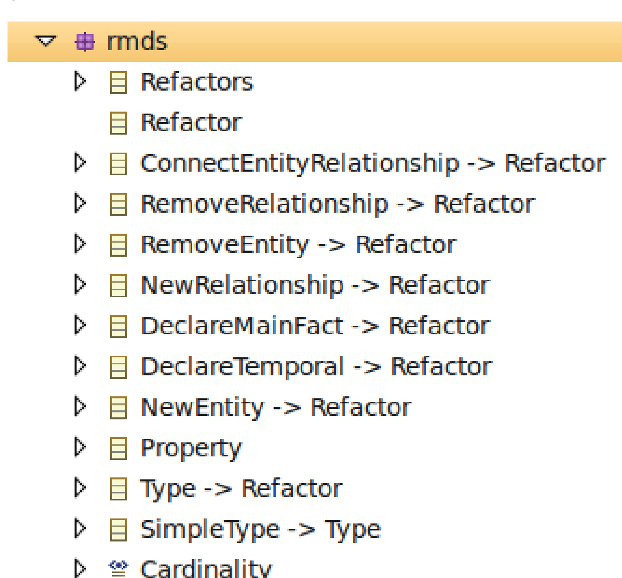
Figura 5.9 Esquema de las notaciones de reestructuración visual definidas



5.5.2.1 Notaciones abstractas

rmds.ecore: Al generar nuestro lenguaje de reestructuración del modelo de entidad relación mediante xText obtenemos automáticamente una versión en ecore del mismo(rmds.ecore) que puede ser utilizada para construir notaciones concretas gráficas.

Figura 5.10 Notación de reestructuración visual



5.5.2.2 Notaciones concretas

Definidas con la herramienta EMF

Notación concreta visual: Definida en base a la versión en ecore de nuestro lenguaje de reestructuración. Debido a que tenemos nuestra notación abstracta

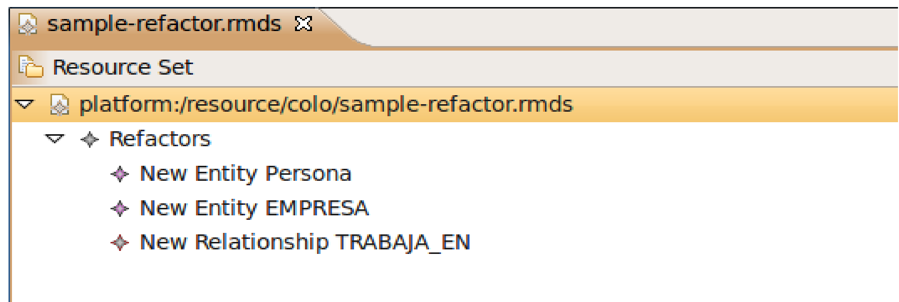
definida ecore podemos utilizar la potencia que nos brinda EMF para generar editores para nuestros modelos en ecore.

5.5.2.3 Código derivado

5.5.2.3.1 Un editor de modelo rmds basado en su estructura

Valiéndose del modelo de dominio en ecore(rmds.ecore) podemos obtener a través de la plataforma EMF un editor basado en la estructura del modelo que le permitirá al diseñador editar y crear instancias del modelo relacional.

Figura 5.11 Una serie de refactors en nuestra notación concreta visual(refactor.rmds)



5.5.2.4 Extensiones desarrolladas

5.5.2.4.1 Validaciones

Mediante el framework de validación que nos provee la plataforma eclipse y utilizando los mecanismos descriptos en 3.2 se ha desarrollado un plugin que forma parte de nuestro esquema de reestructuraciones y que tiene como objetivo validar la consistencia de nuestros modelo relacionales. Esto para tener la certeza de que nuestros modelos son válidos y para que sean aptos para ser utilizados como modelos fuente para transformaciones que forman parte del proceso de generación del datawarehouse.

El validador funciona sobre el modelo de reestructuración visual que es el que se utiliza como entrada para las transformaciones que se utilizan para generar el datawarehouse.

Las validaciones sobre el modelo relacional pueden ser realizadas de las siguientes maneras:

- ✔ *Batch*: Ejecutadas de manera explicita y de a lotes.

y pueden ser implementadas en

- ✔ *Java:* Están basadas en el modelo de dominio en lenguaje java derivado de nuestro modelo de dominio en lenguaje ecore(mrel.ecore). Las clases java que implementan nuestras constraints realizan la validación sobre nuestro modelo de dominio en java derivado de nuestro modelo de dominio que forma parte de la sintaxis abstracta(mrel.ecore).

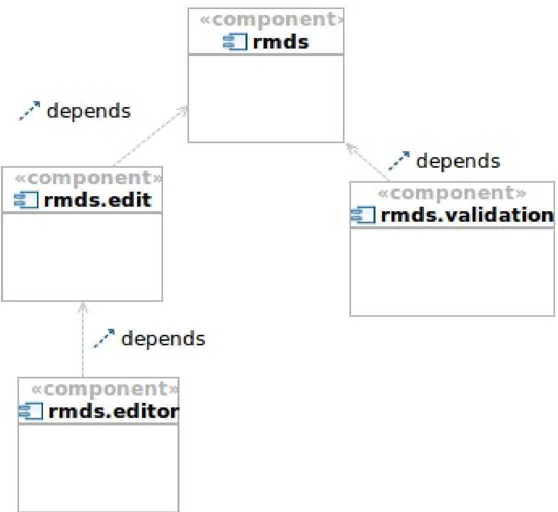
El plugin que implementa las validaciones en java es:

- ✔ **ar.edu.unlp.info.baez.mrel.validation.general**

5.5.2.5 Plugins que forman parte del DSL visual

Para resumir es necesario dejar expresado que nuestro conjunto de plugins fue obtenido en base a la derivación de código y la adaptación de nuestras notaciones concretas. Con ellos podemos crear y manipular del lenguaje de reestructuraciones
En la figura 5.12 se muestra un esquema del conjunto de plugins que dan el soporte para edición visual del modelo relacionales y plugins desarrollados.

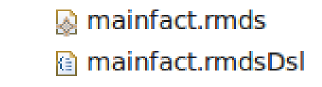
Figura 5.12 esquema de componentes y sus dependencias



5.6 DSL visual de reestructuraciones - uso

Desde el punto de vista del diseñador que utiliza nuestra herramienta, el mismo especificará reestructuraciones a llevar a cabo en los modelos de Entidad-Relación utilizando el desarrollo dirigido por modelos. Utilizando el editor creará modelos que se serializan acorde a una notación abstracta. Por convención adoptamos la extensión ".rmds" para los archivos que contienen la notación abstracta de reestructuración. Tener estas notaciones abstractas basadas en XMI[27] y conforme a nuestros metamodelos en ecore es particularmente útil si queremos que nuestras instancias del modelo relacional sean utilizadas como para ser transformadas,o serializadas de manera estandard para ser leído por otra herramienta y de diferentes manera , así es en este caso ya que tendremos dos notaciones concretas para nuestra notación abstracta de reestructuración. En estos casos la información que contiene aspectos visuales no es necesaria.

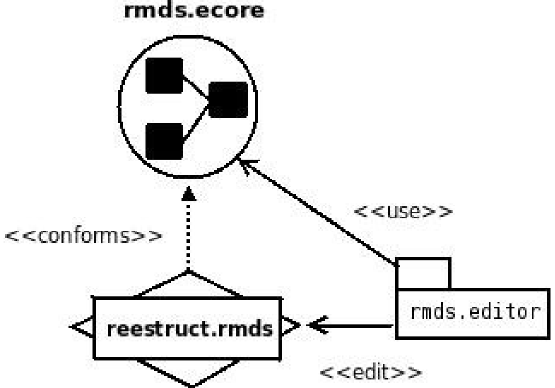
Figura 5.13 Vista de la notación abstracta que utiliza el diseñador



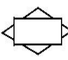
5.6.1 Notaciones

Al igual que los modelos y metamodelos utilizados para definir las notaciones abstractas y concretas que forman parte de nuestra herramienta, el diseñador que utiliza dicha herramienta desarrolla y manipula modelos relacionales modificando notaciones abstractas y concretas para representar el lenguaje de reestructuración.

Figura 4.13 relación entre el editor ,las notaciones concretas, abstractas y el metamodelo desde el punto de vista del diseñador.



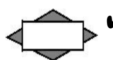
5.6.1.1 Notación abstracta de reestructuración basada en el modelo de Reestructuración



- ✔ *rmds*: Consta de un archivo XML, que contiene la serialización del modelo relacional en formato XMI(xml metada interchange)[27] que nos representa las instancias de reestructuraciones.

Figura 5.14 Un ejemplo de la notación abstracta de reestructuración correspondiente al diagrama de la figura 7.5

```
<?xml version="1.0" encoding="ASCII"?>
<rmds:Refactors xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rmds="http://ar/edu/unlp/info/baez/mds/refactor">
  <refactors xsi:type="rmds:DeclareMainFact" name="RECIBE"/>
  <refactors xsi:type="rmds:DeclareTemporal" name="OCUPA"/>
</rmds:Refactors>
```



- ✔ *Editor basado en la estructura del modelo:* Podemos ver de manera visual y concreta la notación abstracta de reestructuración utilizando el editor basado en la estructura del modelo. En la figura 4.10 se observa un ejemplo de este editor. Su principal utilidad es que podemos utilizarlo para manipular un modelo relacional si tener el editor anterior instalado, Si bien esto mas complicado que hacerlo que con el editor anterior basado en GMF es mucho mas fácil que modificar la sintaxis abstracta manualmente. Si bien no se serializa ningún archivo para representar nuestro editor basado en el modelo este puede ser visto como una notación concreta ya que nos brinda una representación adaptada de la notación abstracta anterior.

5.7 DSL textual y visual – correspondencia

5.7.1 Transformación basada en recursos EMF

Debido al hecho que la sintaxis abstracta de nuestro DSL de reestructuración fue definida en xText(rmds.xtxt) y que xText nos permite inferir una versión en lenguaje ecore de la misma(rmds.ecore) obtenemos de esta manera un metamodelo basado en xText y otro en lenguaje ecore para nuestro lenguaje de reestructuración. Gracias a la correspondencia entre estos dos lenguajes podemos convertir nuestros scripts de reestructuración desarrollados a través de nuestro lenguaje textual en instancias de los mismos conformes a rmds.ecore que representa reestructuraciones visuales. Esto es particularmente útil ya que nos permite a través de xText tener un editor para nuestro lenguaje de scripts de reestructuraciones y luego transformarlo en uno basado en ecore que sirve de entrada para editores visuales o para transformaciones basadas en ATL[23].En el marco del presente trabajo solo hemos desarrollado la conversión del lenguaje textual al gráfico basado en ecore ya que este último es que utilizaremos como entrada para nuestras transformaciones que ejecutarán el refactor. La transformación ha sido realizada mediante la utilización de un recurso EMF, descripto a continuación.

5.7.2 Implementación de la transformación

Esta conversión de nuestras instancias de nuestra notación concreta textual en instancias de nuestra notación concreta visual se desarrolló utilizando un recurso EMF que nos brinda la herramienta xText descripto anteriormente y que nos permite leer nuestros modelos textuales como modelos basados en ecore e incluso exportar nuestro modelo textual en un modelo basado en ecore.

Figura 5.15 Relación entre la reestructuraciones en xtxt y en ecore

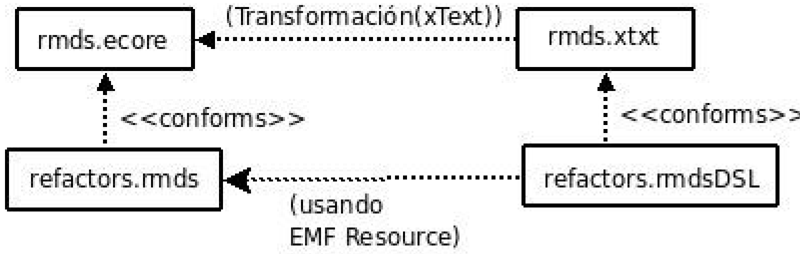
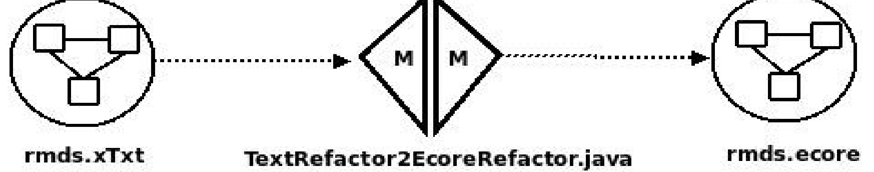


Figura 5.16 Esquema de la transformación en java-EMF Resource entre notaciones de reestructuración textual y visual

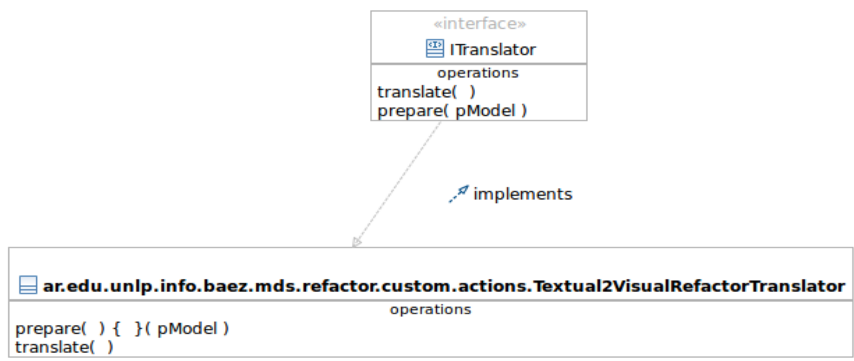


El plugin que donde se implementa está transformación es:

- ✔ **ar.edu.unlp.info.baez.mds.refactor.custom**

La implementación de la la misma está en el método *translate()* de la clase ***ar.edu.unlp.info.baez.mds.refactor.custom.actions.Textual2VisualRefactorTranslator***

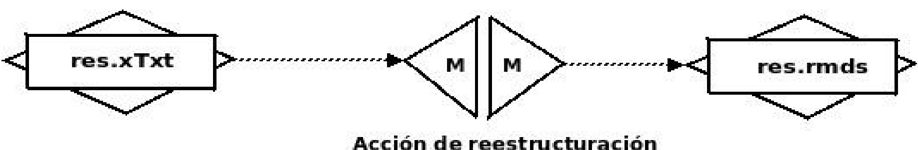
Figura 5.17 Transformador que traduce reestructuraciones textuales(xtxt) en visuales(ecore)



5.7.3 Uso de la transformación

El diseñador transforma reestructuraciones textuales en reestructuraciones visuales. La transformación convierte notaciones abstractas acorde a la visión del diseñador de proceso. Toma una instancia de una reestructuración textual y a partir de esta obtendrá una versión visual en ecore de la misma que podrá usar para el proceso de generación del datawarehouse.

Figura 5.18 Transformación de una reestructuración textual en una visual



5.7.4 Ejecución de la transformación a través de una acción contextual

La transformación es invocada a través de una acción en el menú contextual de la plataforma.

5.8 Reestructuraciones para generar el datawarehouse

Describimos hasta ahora los pasos a llevar a cabo para reestructurar el modelo de Entidad-Relación , mediante nuestro lenguaje de reestructuraciones especificamos las acciones de reestructuración. Dichas acciones serán ejecutadas a través de un transformador implementado en este trabajo y que se describe mas adelante. También aprovecharemos dicho lenguaje para poder especificar las marcas de temporalidad y de hechos de interés que utilizaremos para dar soporte a la cadena de transformaciones de [14]. Recordamos

que en el futuro sería conveniente que estos dos aspectos sean realizados por DSLs separados. En este trabajo por simplicidad se decidió que ambos aspectos estén en un único DSL.

5.9 Reestructuraciones automáticas

El modelo de Entidad-Relación necesita ser reestructurado para ser utilizado para derivar el datawarehouse[14] .Para ello necesitaremos expresar temporalidad en función de las decisiones del diseñador donde se especifican que relaciones,y atributos se desean preservar y cuales relaciones son hechos de interés principal. Para esto se debe modificar el modelo de Entidad-Relación original y debemos realizar una serie de reestructuraciones agregando y/o quitando entidades, relaciones o atributos acorde a los descripto en [14].

Figura 5.19 Reestructuración a realizar si la relación LOC_CLI se desea preservar en el tiempo según [14]

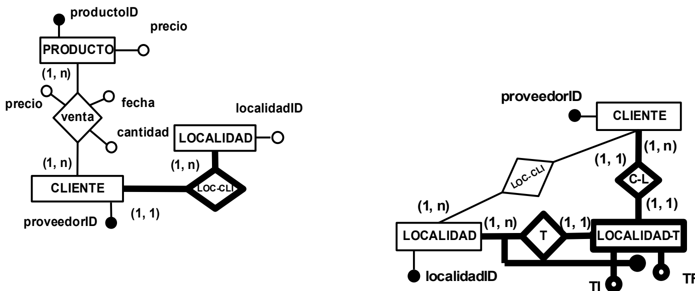
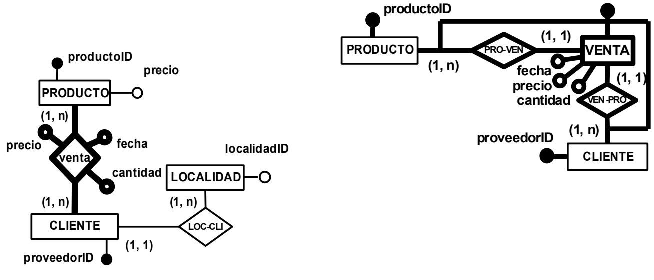


Figura 5.20 Reestructuración a realizar si la relación si la relación VENTA se decide que es un hecho principal según [14]



TI

5.10 Implementación de las transformaciones y uso

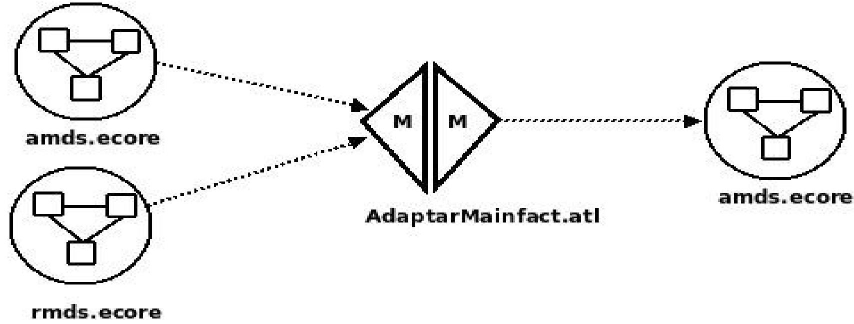
La transformaciones fueron implementadas utilizando el lenguaje ATL. Toma como modelos de entrada el modelo de Entidad-Relación y el lenguaje de reestructuración donde especificamos los aspectos que deseamos preservar y los hechos para la toma de decisiones y obtenemos un modelo de entidad-relación reestructurado. A continuación se describe como fue implementada la reestructuración para aquellas interrelaciones que se deciden que son hechos para la toma de decisiones y posteriormente para aquellas interrelaciones que se deciden preservar en el tiempo. En la sección 5.10.1 se dan detalles

de como se implementan las transformaciones para reestructurar hechos principales y en 5.10.2 como la utiliza el diseñador con un ejemplo de case de uso. En la sección 5.10.3 se dan detalles de como se implementan las transformaciones para reestructurar aspectos temporales y en 5.10.4 como la utiliza el diseñador con un ejemplo de caso de su uso. Luego en 4.4.8 se describe el launcher que utiliza nuestra herramienta.

5.10.1 Transformación automática basada en ATL para hechos de interés

Dado un modelo de Entidad-Relación, y a través de rmds especificamos que relación será un hecho de interés para la toma de decisiones acorde a [14] y una vez especificado en el lenguaje de reestructuración podremos transformar nuestro modelo y obtener el mismo reestructurado a fin de representar nuestro hecho de interés como se especifica en [14].

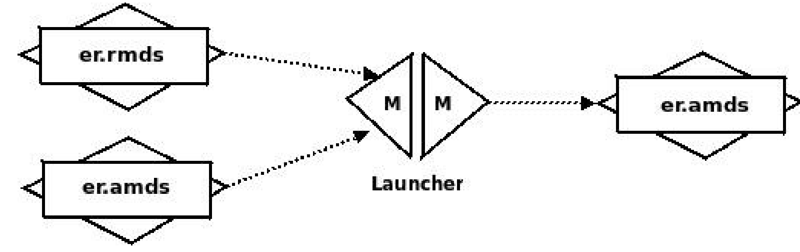
Figura 5.21 Transformación para reestructurar el modelo en base a hechos temporales



5.10.2 Uso de la transformación para hechos de interés

Las siguientes imágenes muestran un ejemplo de un modelo de Entidad-Relación en donde se quiere marcar como hecho de interés para la toma de decisión la ocupación de habitaciones de un hospital. En este caso nuestro hecho de interés principal es la ocupación de habitaciones de un hospital.

Figura 5.22 transformación automática que efectúa la reestructuración del modelo de Entidad-Relación



<u>Ejemplo:</u>

Figura 5.23 El modelo origen

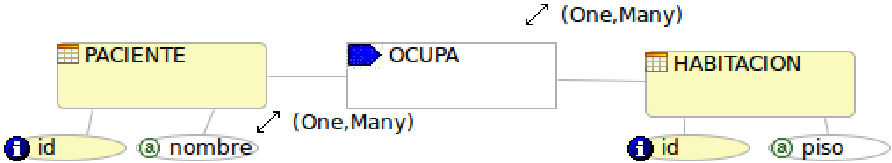
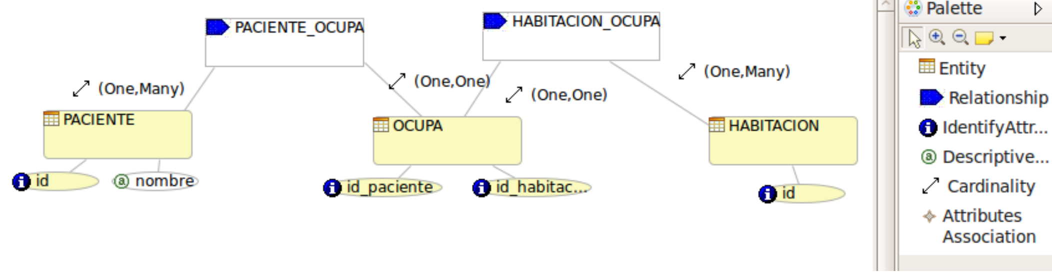


Figura 5.24 La declaración del hecho de interés en el lenguaje de reestructuración

```
declareMainFact OCUPA{
}
```

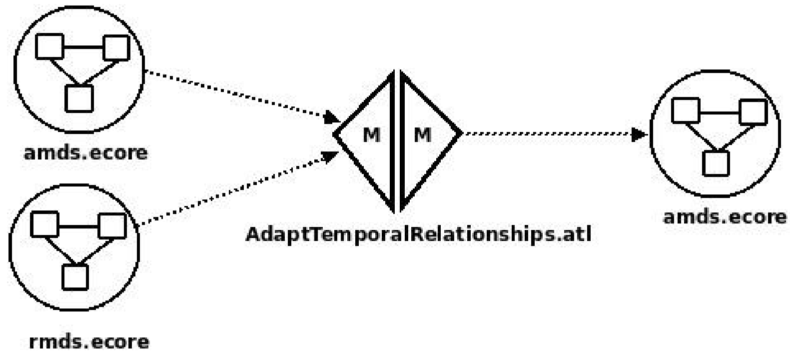
Figura 5.25 El hecho principal reestructurado



5.10.3 Implementación de la transformación automática basada en ATL para aspectos temporales

Dado un modelo de Entidad-Relación, y a través de rmds especificamos que relación será un hecho que deseamos que sea registrado en el tiempo acorde a [14] .Una vez especificado esto en el lenguaje de reestructuración podremos transformar nuestro modelo y obtener el mismo reestructurado a fin de preservar en el tiempo algún hecho como se especifica en [14].

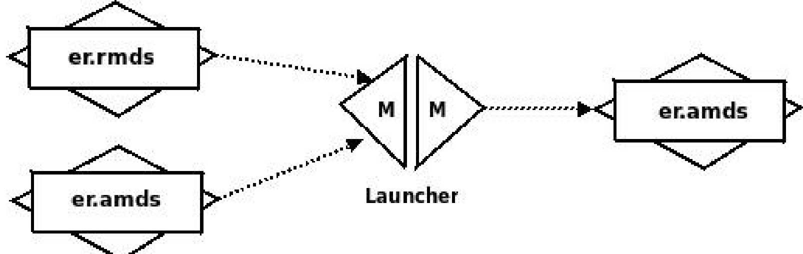
Figura 5.26 esquema de la transformación que invoca el toolkit



5.10.4 Uso de la transformación por parte del diseñador

Las siguientes imágenes muestran un ejemplo de un modelo Entidad-Relación en donde se quiere marcar aquellas interrelaciones que se desean preservar en el tiempo. En este caso nuestro hecho de interés principal es la ocupación de habitaciones de un hospital.

Figura 5.27 esquema de la transformación que invoca el diseñador



Las siguientes imágenes muestran un ejemplo de un modelo de Entidad-Relación en donde se quiere marcar como hecho de interés para la toma de decisión la ocupación de habitaciones de un hospital. En este caso nuestro hecho de interés principal son los tratamientos que reciben los pacientes de un hospital.

4.5.4.1 Ejemplo de modelo que intervienen en la transformación

Figura 5.28 El modelo origen

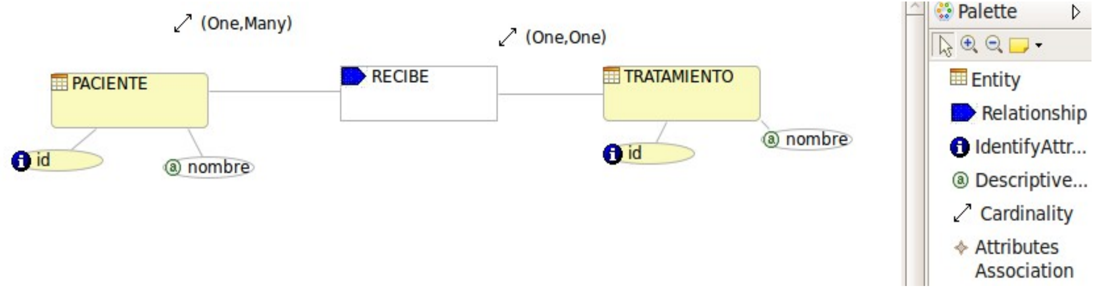
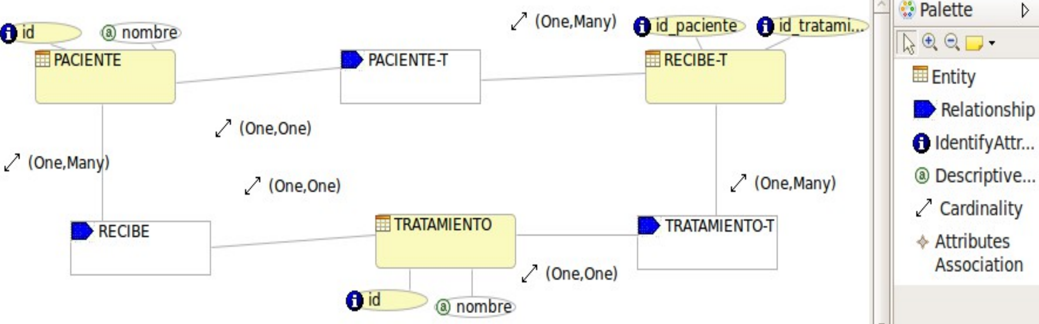


Figura 5.29 La declaración de temporalidad de la interrelación RECIBE

```
declareTemporal OCUPA{
}
```

Figura 5.30 El modelo reestructurado

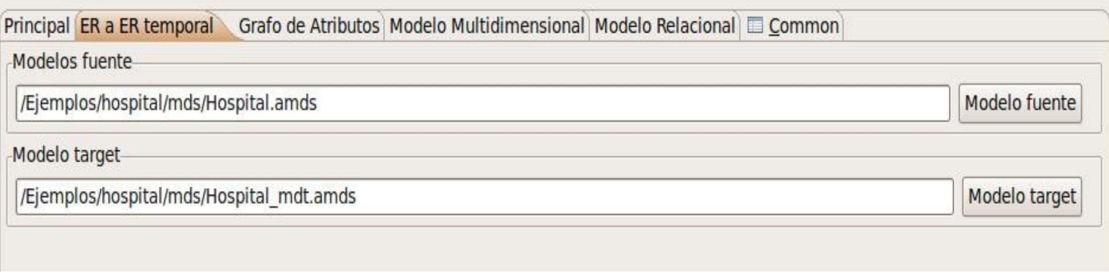


5.11 Ejecución de las transformaciones

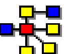
Para ejecutar la transformación el diseñador debe configurar la misma en el launcher de transformaciones que se describe en la sección 3.2.4.

Para ello se debe configurar el tab principal del launcher donde se debe seleccionar la opción “ER al ER temporal”, la reestructuración visual en ecore, con extensión rmds y luego configurar en el tab “ER temporal” los modelos de origen y target de la transformación. Como modelo origen seleccionamos un modelo de Entidad-Relación con extensión “.amds” y como modelo target seleccionamos el archivo donde queremos que deje la serialización del modelo reestructurado. El mismo es un archivo con extensión “.amds”. En las imágenes se muestra un ejemplo configuración de la transformación.

Figura 5.31 Configuración de la transformación en el tab “ER a ER temporal del launcher”



Capítulo 6

6 Grafo de Atributo 

En este capítulo se describen los conceptos de grafo de atributos, el metamodelo del grafo propuestos en [14], las notaciones abstractas y concretas implementadas en respuesta a las propuestas de dicho trabajo y utilizadas luego para derivar el código para implementar nuestros editores visuales para manipular modelos de grafo de atributos.
No especificaremos conceptos teóricos sobre grafos ya que consideramos que no es necesario en este caso debido a que el modelo de grafo utilizados es un modelo ad-hoc y cuya única utilidad es para construir un modelo multidimensional.

6.1 Grafo de atributos - Una breve definición

En el trabajo [13], en base a un modelo de Entidad-Relación reestructurado y sobre cuya estructura será aplicado un algoritmo recursivo permitirá obtener un grafo de atributos temporal; a partir de éste y mediante decisiones del diseñador respecto a qué medidas, dimensiones y niveles de jerarquías obtendremos luego un modelo multidimensional[13].
Según [14]:
“Dada un área de interés en un modelo entidad interrelación temporal y una entidad E que pertenece a él, denominamos grafo de atributos al grafo tal que:

- *Cada vértice corresponde a un atributo, simple o compuesto del modelo entidad interrelación.*
- *La raíz corresponde al identificador de E.*
- *El atributo correspondiente a cada vértice v, determina funcionalmente a todos los atributos descendientes de v.*

Los vértices temporales representan esquemas que tienen como foco de interés la variación de atributos e interrelaciones en función del tiempo. Dado un identifier(E) que indica un conjunto de atributos que identifican a la entidad E, el grafo de atributos será construido semi automáticamente mediante la aplicación de la siguiente función recursiva modificada de [14]”

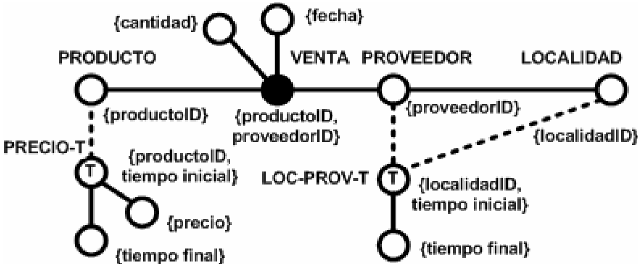
En [14] para construir el grafo se utiliza la función basada en el modelo ER reestructurado:

```
  Function translate (E: Entity): Vertex
{v = newVertex(E);
// newVertex(E) crea un nuevo vértice,
// conteniendo el nombre y el identificador
// del objeto E
for each attribute a ∈E□  a ∉identifier(E)
do
  addChild (v, newVertex(a));
  // se agrega un hijo a al vértice v
  for each entity G connected to E
```

```
   by relationship R□card-max(E,R)=1 xor R is
 temporal do
 // se consideran interrelaciones y atributos
 // temporales
 {for each attribute b ∈ R do
    addChild (v, newVertex(b));
    addChild (v, translate(G));
 }
 return(v)}
```

La misma está basada en el trabajo [15] y fue modificada en [14] por sus autores a los fines de poder preservar aspectos temporales además de hechos de interés para la toma de decisiones.

Figura 6.1 Ejemplo de grafo de atributos de [14]

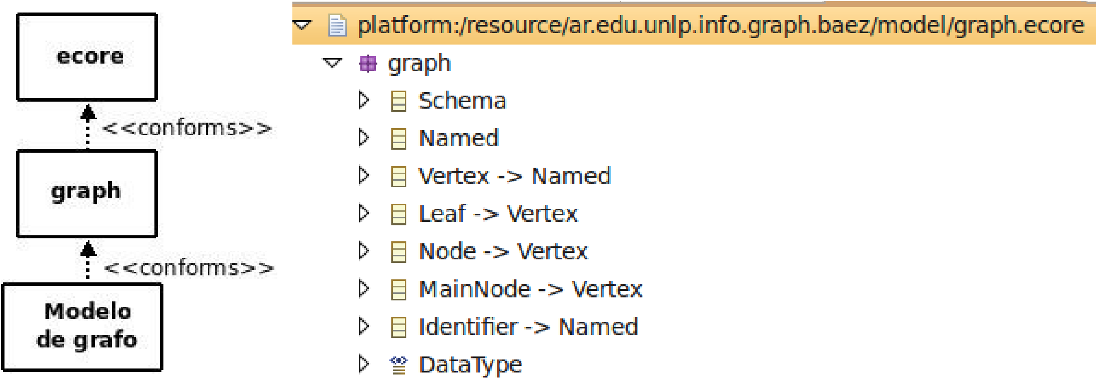


6.2 DSL definido para el grafo de atributos - construcción

6.2.1 Metamodelo del grafo de atributos(mgragh)

El metamodelo del grafo de atributos que hemos utilizados es el propuesto en [14] y se ha adoptado el mismo casi en su totalidad. Se adopta el mismo como metamodelo y de esta manera a partir de una versión en ecore del mismo se desarrollan las notaciones abstractas y concretas que utilizamos para derivar código de nuestros editores y lenguajes textuales.

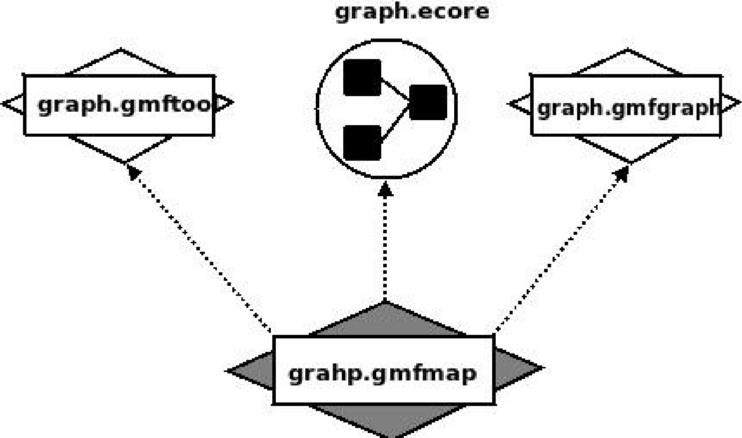
Figura 6.2 Grafo y metamodelos   Figura 6.3 Implementación del grafo en ecore



6.2.2 Notaciones

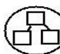
La implementación de nuestro DSL consta de varias notaciones abstractas y varias concretas desde las cuales derivamos código para generar los editores gráficos.

Figura 6.4 Esquema de las notaciones abstractas y concretas



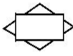
6.2.2.1 Notaciones abstractas del grafo aspecto gráficos

Notaciones abstractas definidas por la herramienta EMF:

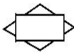


- ✔ *graph.ecore*:Se ha desarrollado un modelo de dominio en lenguaje ecore, y con ella implementamos el metamodelo propuesto en [14]**.** El mismo casi no ha sido modificado a los fines de la implementación. Este es nuestro modelo de dominios de grafo de atributos.

Notaciones abstractas definidas por la herramienta GMF:



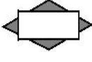
- ✔ *graph.gmgraph*: Es una definición de diagrama que especifica un conjunto de figuras y sus relaciones



- ✔ *graph.gmftool*: Es una definición de diagrama que define un conjunto de herramientas en un toolbar

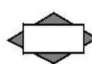
6.2.2.2 Notaciones concretas del grafo

Definidas por la herramienta EMF



- ✔ *graph.gemodel*: Mediante la herramienta de eclipse EMF hemos se desarrollado está gramática para indicar como convertir nuestro modelo de dominios en clases java que podrán ser utilizadas para crear instancias del modelo del grafo.

Definidas por la herramienta GMF:



- ✔ *graph,gmfmap*: Mediante la herramienta de eclipse GMF se ha desarrollado una notación concreta que nos permite combinar las notaciones abstractas anteriores. En la misma se asocian elementos del modelo de dominio con los objetos gráficos y los elementos de la barra de

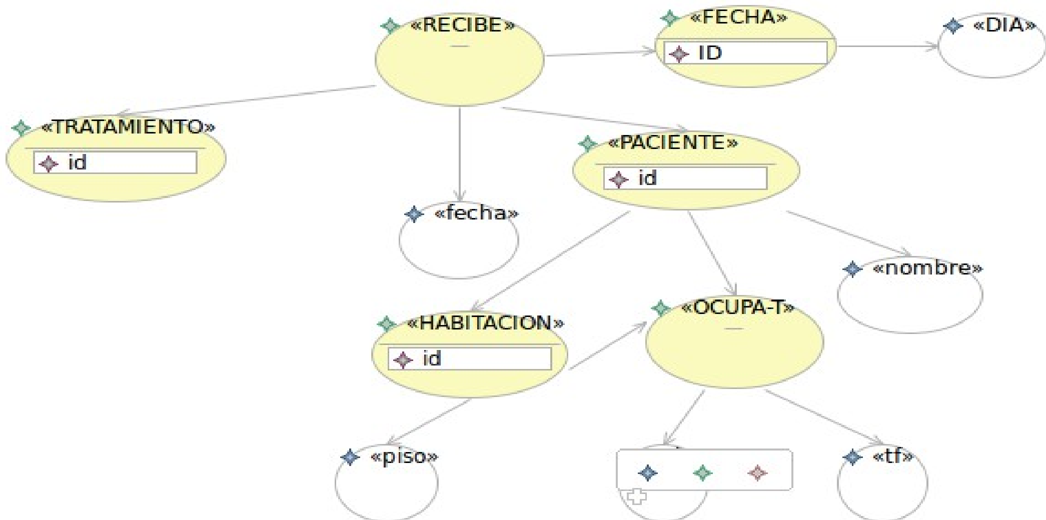
herramientas. Esto se toma como base para derivar el código de nuestro editor gráfico.

6.2.2.3 Código derivado

6.2.2.3.1 Un editor de para el grafo de atributos

Valiéndose de las notaciones concretas anteriores derivamos código java para nuestro modelo de dominios a través de graph.*gemodel* y luego un editor gráfico para crear instancias de nuestros modelos a través de mds.gmfmap. Para ello utilizamos los mecanismos que nos brinda la herramienta GMF en conjunto con el lenguaje XPAND para generar código a partir de los modelos anteriores. De esta manera obtenemos un editor gráfico para desarrollar instancias del modelo mds.

Figura 6.5 Ejemplo del editor del grafo de atributos



6.2.2.4 ¿Extensiones al grafo? ¿Validaciones?

Debido a que el grafo de atributos es un mecanismo ad-hoc para el proceso de generación del datawarehouse y que solo es utilizado como un mecanismo "de paso" para la generación del datawarehouse, sin ningún otro uso, se decidió no profundizar en aspectos ni en extensiones que faciliten la manipulación del grafo de atributos. Queda para el futuro analizar otros usos que se pueden dar al grafo de atributos, como puede ser la generación automática de consultas. Pero dichos usos del grafo de atributos están fuera del alcance del presente trabajo.

6.3 DSL del grafo de atributos - uso

Desde el punto de vista del diseñador que utiliza nuestra herramienta, el mismo podría crear modelos del grafo utilizando el desarrollo dirigido por modelos. Utilizando el editor puede modificar modelos que se serializan acorde a una notación abstracta. Por convención adoptamos la extensión ".graph" para los archivos que contienen la notación

abstracta relacional. Tener estas notaciones abstractas basadas en XMI y conforme a nuestros metamodelos en ecore es particularmente útil si queremos que nuestras instancias del modelo relacional sean utilizadas como para ser transformadas,o serializadas de manera estandard para ser leído por otra herramienta. En estos casos la información que contiene aspectos visuales no es necesaria.
También a través del editor serializa una notación concreta que posee aspectos visuales que toma el editor relacional para facilitar la edición relacional, Por convención adoptamos la extensión “graph_diagram” para la notación concreta visual.

Queda a disposición del diseñador el editor de grafo para manipulaciones manuales solo para casos en que haya que efectuarle modificaciones debido a aspectos no considerados en el trabajo y que hagan que el grafo pueda no ser apto para seguir en el proceso de generación del datawarehouse.

6.4 Modelo de Entidad-Relación al Grafo de atributos

6.4.1 Transformaciones automáticas

Para construir un grafo de atributos nos basamos en el modelo de Entidad-Relación que se describe en el capítulo 4 , el mismo se asume fue reestructurado y que el mismo ya posee su estructura adecuada de acuerdo a los cambios que se proponen en [14].
Para realizar la transformación utilizamos las marcas de temporalidad y de hechos para la toma de decisión que se especifican mediante el lenguaje de reestructuración descripto en el capítulo 5. Esta transformación toma como modelos de entrada un modelo de Entidad-Relación reestructurado temporalmente (amds) , un modelo de reestructuración(rmds), descripto en el capítulo 5 con la información sobre las marcas sobre hechos y temporalidad y obtenemos un grafo de atributos.

6.4.2 Implementación de la transformación

La transformación fue implementada utilizando el lenguaje ATL. Toma como modelos de entrada en modelo de Entidad-Relación reestructurado y el una especificación de reestructuración donde se especifican aspectos que deseamos preservar y los hechos para la toma de decisiones.

Figura 6.6.Transformación del modelo entidad-relación al grafo de atributos

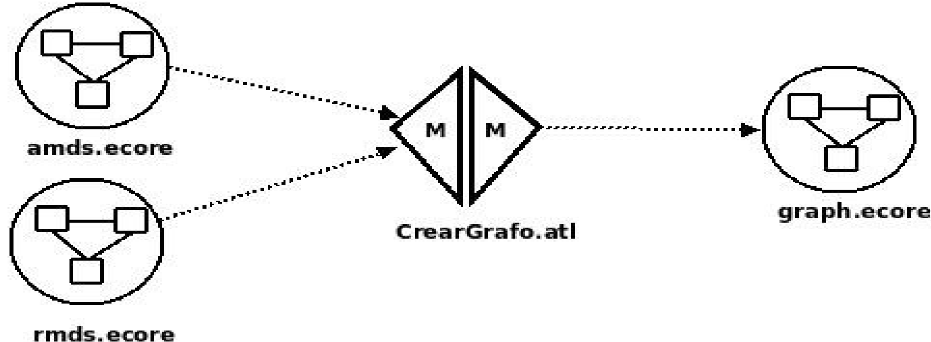


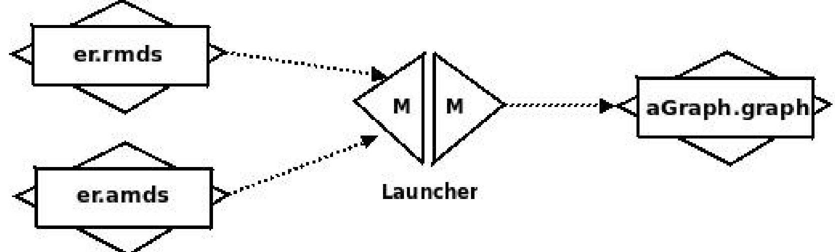
Figura 6.7 Algunas reglas que conforman la transformación ATL para crear el grafo

```
rule Node{
      from
      e : amds!Entity(not e.selectedAsMainFact)
      to
      eo : graph!Node(
            name<- e.name,
            identifiers <- e.getIdentifyAttributes(),
            isRoot<- false,
            children <- e.getDescriptiveAttributes(),
          children <- e.getEntidadesConectadas->
            select( c | e.conectadaTemporalmenteConCardMaxMany(c)
                  xor  c.conectadaConCardinalidadMaxOne(e) )
            -> union(e.getDescriptiveAttributes())
           )
}

rule Leaf{
from e:amds!DescriptiveAttribute
to eo:graph!Leaf(
name<-e.name,
type<- thisModule.getMyType(e.type)
)
}

rule Identifier{
from e:amds!IdentifyAttribute
to eo:graph!Identifier(
name<-e.name,
type <- thisModule.getMyType(e.type)
)
}
```

6.4.3 Uso de la transformación

El diseñador transforma un modelo de Entidad-Relación reestructurado en un grafo de atributos. La transformación convierte notaciones abstractas acorde a la visión del diseñador de proceso. Toma una notación abstracta de un modelo de entidad-relación, una notación abstracta de reestructuración y a partir de estas obtendrá un grafo de atributos.

Figura 6.8 Transformación del modelo ER reestructurado al grafo de atributos



6.4.4 Ejecución de la transformación a través del launcher

Para ejecutar la transformación el diseñador debe configurar la misma en el launcher de transformaciones que se describe en la sección 3.2.4.

Para ejecutarla el diseñador debe configurar en el tab principal del launcher debe seleccionar la opción "Modelo de datos temporal al Grafo de atributos", y luego configurar en el tab "Grafo de atributos" los modelos de origen y target de la transformación. Como modelo origen seleccionamos un modelo de Entidad-Relación que se asume reestructurado, con extensión ".amds" y como modelo target seleccionamos el archivo donde queremos que deje la serialización del grafo resultante. El mismo es un archivo con extensión ".graph". En las imágenes se muestra un ejemplo configuración de la transformación.

Figura 6.9 Vista del tab principal

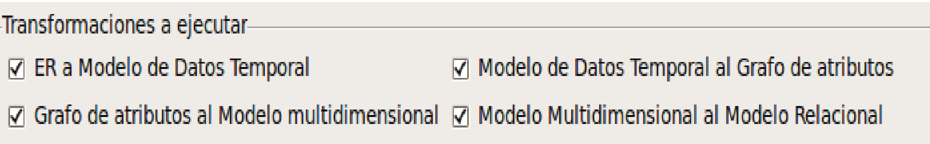


Figura 6.10 configuración de la transformación en el tab "Grafo de Atributos"

Modelos fuente
/Ejemplos/hospital/mds/Hospital_mdt.amds
Modelo fuente
Modelo target
/Ejemplos/hospital/graph/Hospital.graph
Modelo target

Capítulo 7

"El Data WareHouse es una colección de datos orientados al tema, integrados, no volátiles e historiados, organizados para el apoyo de un proceso de ayuda a la toma de decisiones". Bill Inmon en su obra "Using the Data WareHouse".

7 Modelado Multidimensional

En este capítulo se describen los conceptos de modelo multidimensional que hemos tomado como base conceptual, el metamodelo multidimensional temporal propuesto en [14], las notaciones abstractas y concretas implementadas en respuesta a las propuestas de dicho trabajo y utilizadas luego para derivar el código para los editores visuales.

7.1 Modelo Multidimensional para un Datawarehouse - Breve definición

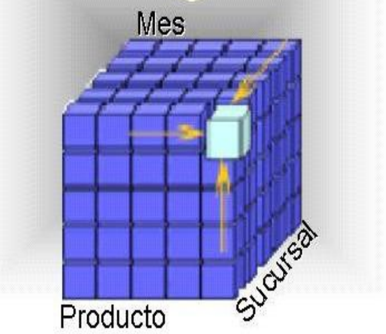
El modelado dimensional es una técnica para modelar bases de datos simples y entendibles al usuario final. Consideremos un punto en el espacio. El espacio se define a través de sus ejes de coordenadas (por ejemplo X, Y, Z). Un punto cualquiera de este espacio quedará determinado por la intersección de tres valores particulares de sus ejes.

7.2 Modelo Multidimensional

Representa los datos como matrices N-Dimensionales denominadas **Hipercubos**

• Las dimensiones definen dominios de datos como geografía, producto, tiempo, cliente...
• Los miembros de una dimensión se agrupan de forma jerárquica de acuerdo a(dimensión geográfica: ciudad, provincia,autonomía, país).
• Cada celda contiene datos agregados que relacionan los elementos de las dimensiones.

Figura 7.1 Ejemplo de cubo de datos de ventas por mes, producto y sucursal



La estructura básica de un DW para el Modelo Multidimensional está definida por dos elementos: esquemas y tablas.

7.2.1 Esquemas y tablas

Se brinda aquí una breve descripción de los tipos de tablas que forman parte un modelo multidimensional.

7.2.2 Tablas DW

El datawarehouse es una base de datos relacional, de esta manera el mismo se compone de tablas.

7.2.2.1 Tablas Fact

Es la tabla central en un esquema dimensional, contiene los valores de las medidas de negocios. Cada medida es tomada de la intersección de las dimensiones que la definen.
La clave de la tabla fact recibe el nombre de clave compuesta o concatenada debido a que se forma de la composición (o concatenación) de las llaves primarias de las tablas dimensionales a las que está unida.
Así entonces, se distinguen dos tipos de columnas en una tabla fact: columnas fact y columnas key. Donde la columna fact es la que almacena alguna medida de negocio y una columna key forma parte de la clave compuesta de la tabla

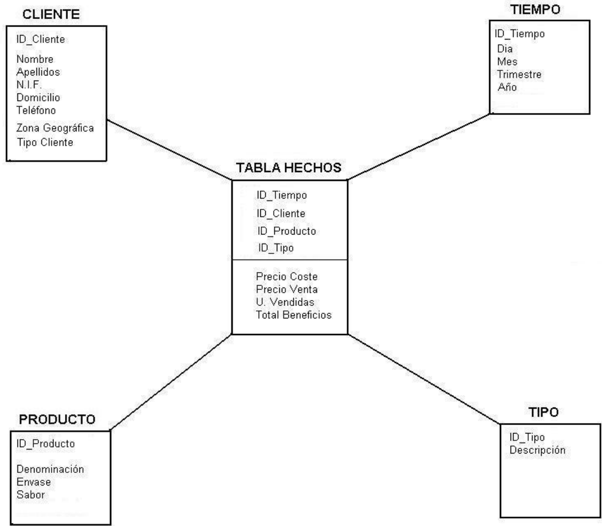
7.2.2.2 Tablas dimensionales

Estas tablas son las que se conectan a la tabla fact. Una tabla Lock-up almacena un conjunto de valores que están relacionados con una dimensión particular. Están compuestas por una clave primaria.
En las tablas de hechos hay un conjunto de atributos son llamados medidas (measures).
Las relaciones entre las dimensiones y dimensiones es m – m.
Las relaciones entre hecho y dimensiones 1 – m

7.2.3 Esquemas DW

Es una colección de tablas en el DW se conoce como esquema. Pueden ser de dos tipos: estrella y copos de nieve.
Esquema estrella: Consiste en estructurar la información en procesos, vistas y métricas recordando a una estrella (por ello el nombre star schema). Es decir, tendremos una visión multidimensional de un proceso que medimos a través de unas métricas. A nivel de diseño, consiste en una tabla de hechos en el centro para el hecho objeto de análisis y una o varias tablas de dimensión.
Esquema en copo de nieve: El esquema en copo de nieve (snowflake schema) es un esquema de representación derivado del esquema en estrella, en el que las tablas de dimensión se normalizan en múltiples tablas. Por esta razón, la tabla de hechos deja de ser la única tabla del esquema que se relaciona con otras tablas, y aparecen nuevas joins gracias a que las dimensiones de análisis se representan ahora en tablas de dimensión normalizadas. En este trabajo no nos ocuparemos de este tipo de esquemas.

Figura 7.2 Ejemplo de esquemas estrella.



7.3 Finalidad de un Datawarehouse

Consiste en auxiliar a la administración a comprender el pasado y planear el futuro.
La promesa del datawarehousing es "sacar datos" de los sistemas operacionales para ayudar a las empresas a tomar mejores decisiones.

7.4 La Dimensión Tiempo

Virtualmente se garantiza que cada DDW tendrá una tabla dimensional de tiempo, debido a la perspectiva de almacenamiento histórica de la información. Usualmente es la primera dimensión en definirse, con el objeto de establecer un orden, ya que la inserción de datos en la base de datos multidimensional se hace por intervalos de tiempo, lo cual asegura un orden implícito.

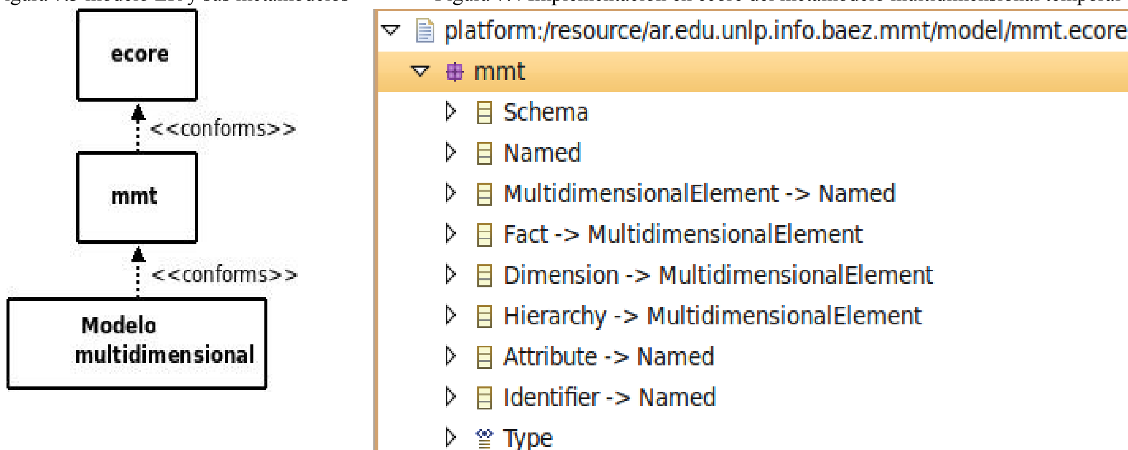
7.5 DSL Multidimensional – construcción

7.5.1 Metamodelo multidimensional temporal(mmt)

El metamodelo multidimensional que hemos utilizados es el propuesto en [14] y se ha adoptado el mismo casi en su totalidad. El mismo esta basado a su vez en la propuestas de [14] y de [16].Se adopta el mismo el mismo como metamodelo y de esta manera a partir de el se desarrollan las gramáticas abstractas y concretas que utilizamos para derivar código de nuestros editores . Los modelos multidimensionales que se obtienen serán conformes al metamodelo mmt. El metamodelo multidimensional es especificado en el lenguaje ecore.

Figura 7.3 modelo ER y sus metamodelos

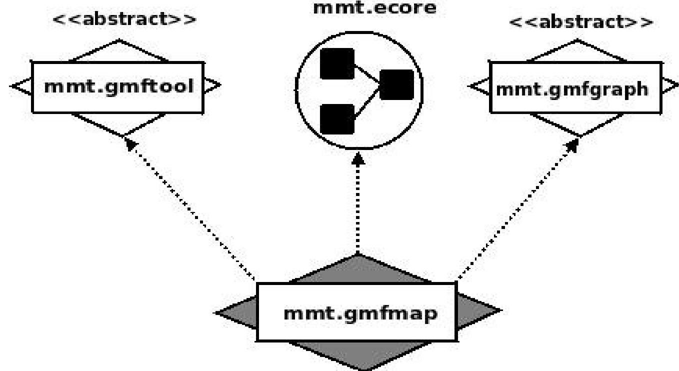
Figura 7.4 Implementación en ecore del metamodelo multidimensional temporal



7.5.2 Notaciones definidas

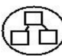
La implementación de nuestro dsl consta de varias notaciones abstractas y varias concretas desde las cuales derivamos código para generar los editores gráficos

Figura 7.5 Esquema de las gramáticas definidas y sus relaciones



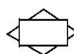
7.5.2.1 Notaciones abstractas

Notaciones abstractas definidas por la herramienta EMF:

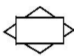


- ✔ *mmt.ecore*:Se ha desarrollado un modelo de dominio en lenguaje e-core, y con ella implementamos el metamodelo propuesto en [14]. El mismo casi no ha sido modificado a los fines de la implementación. Este es nuestro modelo de dominios. La definición del modelo de dominio en ecore nos brinda la ventaja de poder tener varias implementaciones conformes al mismo.

Notaciones abstractas definidas por la herramienta GMF:



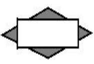
- ✔ *mmt.gmgraph*: Es una definición de diagrama que especifica un conjunto de figuras y sus relaciones



- ✔ *mmt.gmftool*: Es una definición de diagrama que define un conjunto de herramientas en un toolbar

7.5.2.2 Notaciones concretas

Definidas por la herramienta EMF



- ✔ *mmt.gemodel*: Mediante la herramienta de eclipse EMF hemos se desarrollado está gramática para indicar como convertir nuestro modelo de dominios multidimensional en clases java que podrán ser utilizadas para crear instancias del modelo mds

Definidas por la herramienta GMF:

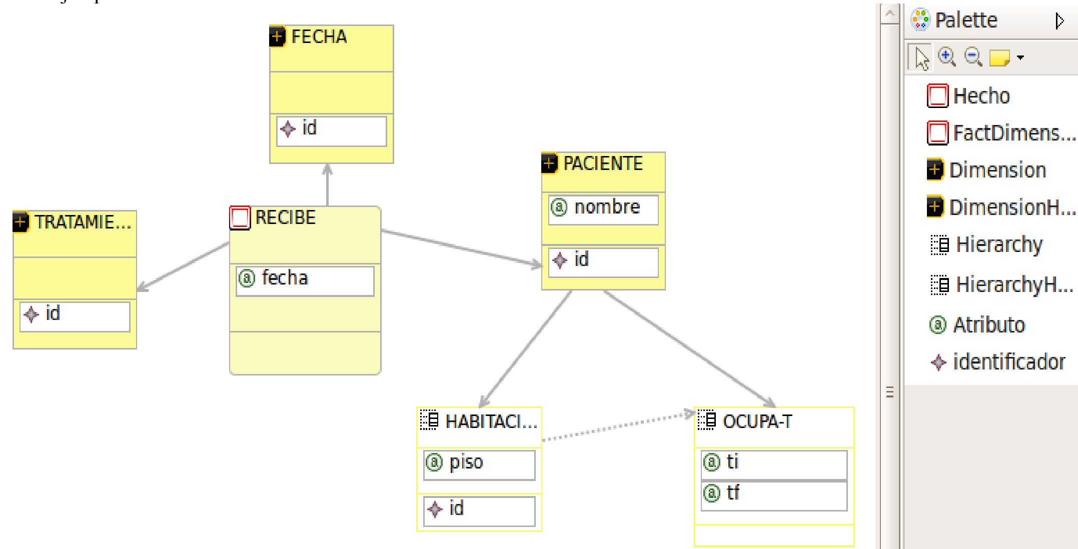
- ✔ *mmt.gmfmap*: Mediante la herramienta de eclipse GMF se ha desarrollado una notación concreta que nos permite combinar las notaciones abstractas anteriores. En la misma se asocian elementos del modelo de dominio con los objetos gráficos y los elementos de la barra de herramientas. Esto se toma como base para derivar el código de nuestro editor gráfico.

7.5.2.3 Código derivado

7.5.2.3.1 Un editor Multidimensional

Valiéndose las notaciones concretas anteriores derivamos código java para nuestro modelo de dominios a través de *mmt.gemodel* y luego un editor gráfico para crear instancias de nuestros modelos a través de mmt.*gmfmap*. Para ello utilizamos los mecanismos que nos brinda la herramienta GMF en conjunto con el lenguaje XPAND para generar código a partir de los modelos anteriores. De esta manera obtenemos un editor gráfico para desarrollar instancias del modelo mmt.

Figura 7.6 Ejemplo del editor multidimensional



7.5.2.3.2 Validaciones

Mediante el framework de validación que nos provee la plataforma eclipse y utilizando los mecanismos descriptos en 3.2 se ha desarrollado un plugin que forma parte de nuestro esquema relacional y que tiene como objetivo validar la consistencia de nuestros modelo relacionales. Esto para tener la certeza de que nuestros modelos son válidos y para que sean aptos para ser utilizados como modelos fuente para transformaciones que forman parte del proceso de generación del datawarehouse.
Las validaciones sobre el modelo relacional pueden ser realizadas de las siguientes maneras:

- ✔ *Batch*: Ejecutadas de manera explicita y de a lotes.

y pueden ser implementadas en

- ✔ *Java*: Están basadas en el modelo de dominio en lenguaje java derivado de nuestro modelo de dominio en lenguaje ecore(mrel.ecore). Las clases java que implementan nuestras constraints realizan la validación sobre nuestro modelo de dominio en java derivado de nuestro modelo de dominio que forma parte de la sintaxis abstracta(mrel.ecore).

Constraints implementadas

- ✔ *Las tablas deben poseer nombres*

  Al ser las tablas objetos de tipo Named, estos si o si deben poseer nombre.

- ✔ *El esquema no puede estar vació.*

  Si o si deben existir al menos una tabla. La colección llamada tables del objeto Schema de nuestro modelo de dominios no puede ser vaciá.

- ✔ *Las tablas deben poseer al menos algún atributo.*

  No pueden existir tablas si atributos en nuestro modelo de dominio,

  el objeto Schema de nuestro modelo de dominio posee una colección llamada tables esta no puede ser vació.

- ✔ *Las claves foráneas deben apuntar a un atributo clave de otra tabla.*

  Cuando agregamos una columna a una tabla y esta es de tipo clave foránea, esta en nuestro modelo de dominio esta representada por la clase ForeignkeyColumn que posee una referencia a la clave primaria de otra tabla. Esta referencia no puede ser nula. El objeto de nuestro modelo de dominios que nos representan esta asociación es KeyReference

- ✔ *Los atributos deben poseer nombre y tipo.*

  Si o si debemos asignarle un nombre valido a los atributos de nuestro esquema y estos deben tener un tipo de datos asignado.

- ✔ *El esquema debe poseer al menos un hecho*
- ✔ *El hecho principal debe tener la dimension temporal*

Esto es una advertencia, no un error de constraint. Debido a que [14] exige que el hecho principal tenga una dimensión temporal.

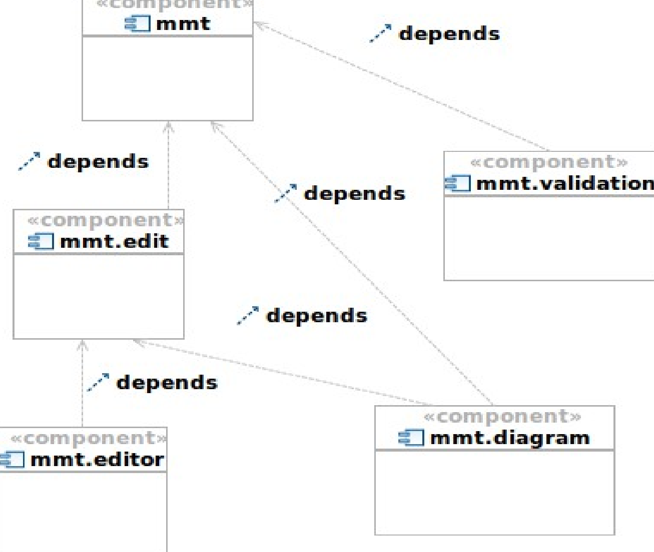
El plugin que implementa las validaciones en java es:

- ✔ **ar.edu.unlp.info.baez.mmt.validation.general**

7.5.2.4 Plugins que forman parte del DSL multidimensional

Para resumir es necesario dejar expresado que nuestro conjunto de plugins fue obtenidos en base a la derivación de código y la adaptación de nuestras notaciones concretas con ellos podemos crear y manipular modelos relacionales conformes a nuestro modelo de dominio y un conjunto de ellos forman parte del soporte visual que poseemos para la creación y manipulación de nuestra notación concreta visual.
En la figura 6.7 se muestra un esquema del conjunto de plugins que dan el soporte para edición visual del modelo relacionales y plugins desarrollados.

Figura 7.7 esquema de componentes multidimensionales y sus dependencias



7.6 DSL Multidimensional - uso

Desde el punto de vista del diseñador que utiliza nuestra herramienta, el mismo creará modelos relacionales utilizando el desarrollo dirigido por modelos. Utilizando el editor creará modelos que se serializan acorde a una notación abstracta . Por convención adoptamos la extensión ".mmt" para los archivos que contienen la notación abstracta

relacional. Tener estas notaciones abstractas basadas en xmi y conforme a nuestros metamodelos en ecore es particularmente útil si queremos que nuestras instancias del modelo relacional sean utilizadas como para ser transformadas,o serializadas de manera estandard para ser leído por otra herramienta. En estos casos la información que contiene aspectos visuales no es necesaria.
También a través del editor serializa una notación concreta que posee aspectos visuales que toma el editor relacional para facilitar la edición relacional, Por convención adoptamos la extensión “mmt_diagram” para la notación concreta visual.
De esta manera la según la visión del diseñador la notación abstracta posee el modelo relacional y una notación concreta posee aspectos visuales. Pudiendo existir mas de una notación concreta para la misma notación abstracta.

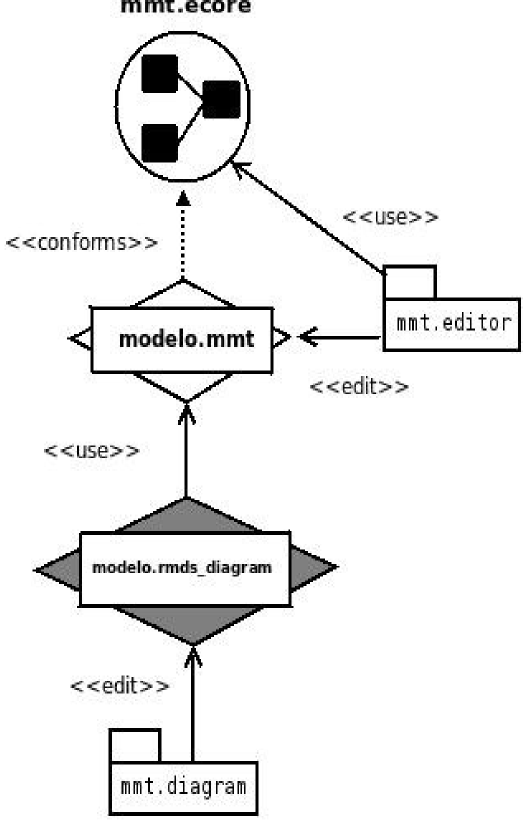
Figura 7.8 Vista de la notación abstracta y concreta que utiliza el diseñador

Hospital.mmt
Hospital.mmt_diagram

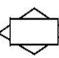
7.6.1 Notaciones del modelo multidimensional

Al igual que los modelos y metamodelos utilizados para definir las notaciones abstractas y concretas que forman parte de nuestra herramienta, el diseñador que utiliza dicha herramienta desarrolla y manipula modelos relacionales modificando notaciones abstractas y concretas para representar el modelo relacional.

Figura 7.9 relación entre los editores,las notaciones concretas, abstractas y el metamodelo desde el punto de vista del diseñador.



7.6.1.1 Notación abstracta multidimensional y gráfica

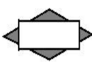


- ✔ *mmt*: Consta de un archivo XML, que contiene la serialización del modelo relacional en formato XMI(xml metada interchange) que nos representa las instancias de nuestro modelo relacional.

Figura 7.10 Un ejemplo de la notación abstracta multidimensional

```
<?xml version="1.0" encoding="UTF-8"?>
<mmt:Schema xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:mmt="http://baez/mmt/info/unlp/edu/ar">
  <facts name="OCUPA"
  dimensions="//@dimensions.0 //@dimensions.1 //@dimensions.2"/>
  <dimensions name="PACIENTE">
    <attributes name="nombre"/>
    <identifiers name="id"/>
  </dimensions>
  <dimensions name="FECHA">
    <identifiers name="id"/>
  </dimensions>
  <dimensions name="CAMA">
    <attributes name="numero"/>
    <identifiers name="ID"/>
  </dimensions>
</mmt:Schema>
```

7.6.1.2 Notación concreta



- ✔ *mmt_diagram:* Consta de un archivo XML en formato xmi que nos representa un conjunto de figuras asociadas a la notación abstracta anterior. Son creados y modificados en base a nuestro editor gráfico relacional. Por cada elemento de la notación abstracta nos indica la figura que lo representa y su posición.

Figura 7.11 Un ejemplo de la notación abstracta relacional

```
<?xml version="1.0" encoding="UTF-8"?>
<notation:Diagram xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mmt="http://baez/mmt/info/unlp/edu/ar"
xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.1/notation"
xmi:id="_1yAnoQwrEd-eI8E9HnsGJA" type="Mmt" name="default.mmt_diagram"
measurementUnit="Pixel">
<children xmi:type="notation:Node" xmi:id="_5dez0AwrEd-eI8E9HnsGJA" type="2004">
<children xmi:type="notation:Node" xmi:id="_5ds2QAwrEd-eI8E9HnsGJA" type="5004"/>
<children xmi:type="notation:Node" xmi:id="_5d31YAwrEd-eI8E9HnsGJA" type="7003">
   <styles xmi:type="notation:SortingStyle" xmi:id="_5d31YQwrEd-eI8E9HnsGJA"/>
   <styles xmi:type="notation:FilteringStyle" xmi:id="_5d31YgwrEd-eI8E9HnsGJA"/>
  </children>
<children xmi:type="notation:Node" xmi:id="_5d8t4AwrEd-eI8E9HnsGJA" type="7004">
   <styles xmi:type="notation:SortingStyle" xmi:id="_5d8t4QwrEd-eI8E9HnsGJA"/>
  <styles xmi:type="notation:FilteringStyle" xmi:id="_5d8t4gwrEd-eI8E9HnsGJA"/>
 </children>
```

7.6.1.3 ¿Notación concreta textual?

Como ocurre con el modelo del modelo de entidad-relación, existen muchas herramientas en el mercado que brindan soporte para el desarrollo de base de datos. Por lo general dichas herramientas no están basadas en el desarrollo dirigido por modelos y utilizan una notación gráfica propia para los diagramas relacionales que imposibilita por lo general que sus diagramas sean exportados a un formato estandard(como CWM por ejemplo) para ser utilizados por otras herramientas. Debido a la simplicidad estructural de los diagramas multidimensionales y a la madurez de estos modelos ,la estructura gráfica de los diagramas multidimensionales entre las diferentes herramientas es bastante uniforme y las diferencias son menores. Esta manera graficar los modelos multidimensionales es bastante común y un camino natural para el diseño relacional. Es por eso que no existen notaciones textuales ampliamente utilizadas para especificar la estructura de los modelos relacionales. Ese es el motivo por el cual se decidió no implementar un DSL textual para diseñar modelos relacionales a pesar de tener la base suficiente para crear un lenguaje en el cual especificar un modelo relacional.

7.7 Grafo de atributos al Modelo Multidimensional

7.7.1 Transformaciones automáticas

Para construir un modelo multidimensional nos basamos en el modelo de modelo multidimensional que se describe en 4.1.

7.7.2 Implementación de la transformación

La transformación fue implementada utilizando el lenguaje ATL. Toma como modelos de entrada un grafo de atributos y a través de una transformación obtenemos un modelo multidimensional.

Figura 7.12 Transformación del grafo de atributos al modelo multidimensional

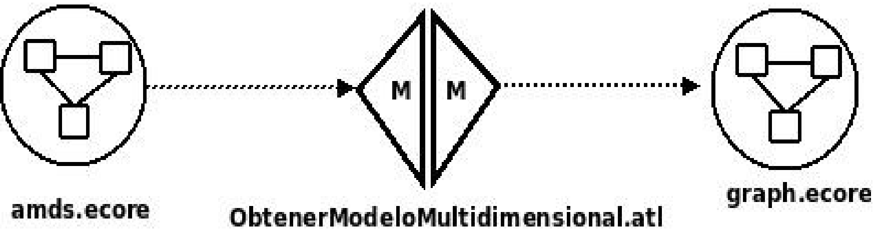


Figura 7.13 Algunas reglas que conforman la transformación ATL para crear el modelo multidimensional

```
rule MainFact {
from n:graph!Node(n.isRoot)
to f:mmt!Fact(
      name<-n.name,
      dimensions <- graph!Node.allInstances() ->
      select(c| c.isDimensionNode) ->
      union(n.children->
      select(c| c.isLeafWithTemporalData )),
      identifiers <- n.identifiers,
      attributes <- n.myAttributes
)
}

rule Dimension {
from n:graph!Node(n.isDimensionNode)
to f:mmt!Dimension(
      name<-n.name,
      identifiers <- n.identifiers,
      hierarchies<- n.children -> select(c| c.isHierarchyNode),
      attributes <- n.myAttributes
)
}

rule TemporalDimension {
from n:graph!Leaf(n.isLeafWithTemporalData)
to f:mmt!Dimension(
      name<-n.name,
      identifiers <- Sequence{newIdentifier},
      attributes <- Sequence{newAttribute}),
      newIdentifier:mmt!Identifier(
        name<- 'id'
      ),
      newAttribute:mmt!Attribute(
        name<- n.name
      )
}
```

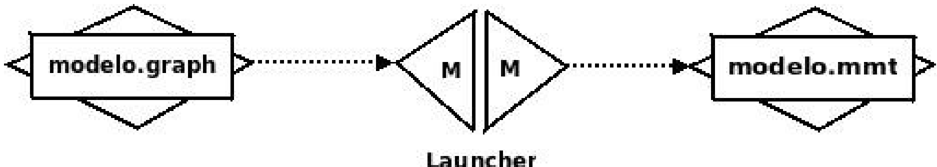
7.7.3 Uso de la transformación

El diseñador transforma un grafo de atributos en un modelo multidimensional. La transformación convierte notaciones abstractas acorde a la visión del diseñador de proceso. Toma una notación abstracta de un grafo de atributos y a partir de estas obtendrá un grafo de atributos.

7.7.4 Ejecución de la transformación

Para ejecutar la transformación el diseñador debe configurar la misma en el launcher de transformaciones que se describe en la sección 3.2.4.

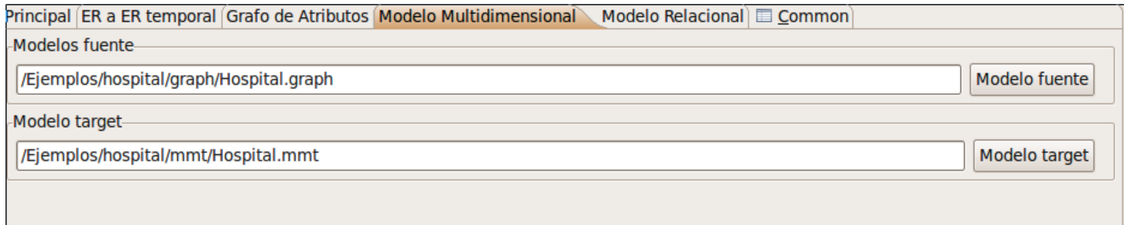
Figura 7.14 Transformación del grafo reestructurado al modelo multidimensional



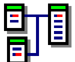
7.7.5 Ejecución de la transformación a través del launcher

Para ejecutar la transformación el diseñador debe configurar la misma en el launcher de transformaciones que se describe en la sección 3.2.4. Para ejecutarla el diseñador debe configurar en el tab principal del launcher debe seleccionar la opción "Grafo de atributos a Modelo Multidimensional", y luego configurar en el tab "Modelo Multidimensonal" los modelos de origen y target de la transformación. Como modelo origen seleccionamos un modelo de grafo de atributos, con extensión ".graph" y como modelo target seleccionamos el archivo donde queremos que deje la serialización del modelo multidimensional resultante. El mismo es un archivo con extensión ".mmt". En las imágenes se muestra un ejemplo configuración de la transformación. En las imágenes se muestra un ejemplo configuración de la transformación.

Figura 7.15 Un ejemplo de configuración del tab correspondiente a la transformación al modelo multidimensional



Capítulo 8

8.1 Modelado relacional 

En este capítulo se describen los conceptos del modelo relacional que hemos tomado como base conceptual, el metamodelo relacional propuesto en [14], las notaciones abstractas y concretas implementadas en respuesta a las propuestas de dicho trabajo y las modificaciones a dicho modelo propuesto en [14] que debieron ser llevadas a cabo a los fines de poder ser utilizado para construir las notaciones concretas utilizadas para implementar nuestros editores visuales.

8.1 Modelo relacional – Una breve definición

Los modelos lógicos están orientados a las operaciones más que a la descripción de una realidad. Usualmente están implementados en algún manejador de base de datos. El modelo relacional es un modelo de datos lógico que tiene su basamento teórico en la lógica de predicados y en la teoría de conjuntos. Hoy en día es el mas utilizado fue propuesto por Edgard Frank Codd en 1970. Se basa en el uso de "relaciones" que son conjuntos de tuplas en dicho modelo cada relación tiene la forma de una tabla. En el modelo relacional todos los datos son almacenados en relaciones, y como cada relación es un conjunto de datos, el orden en el que estos se almacenen no tiene relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar por un usuario no experto. La información puede ser recuperada o almacenada por medio de consultas que ofrecen una amplia flexibilidad y poder para administrar la información.
Este modelo considera la base de datos como una colección de relaciones. De manera simple, una relación representa una tabla que no es más que un conjunto de filas, cada fila es un conjunto de campos y cada campo representa un valor que interpretado describe el mundo real. Cada fila también se puede denominar tupla o registro y a cada columna también se le puede llamar campo o atributo.

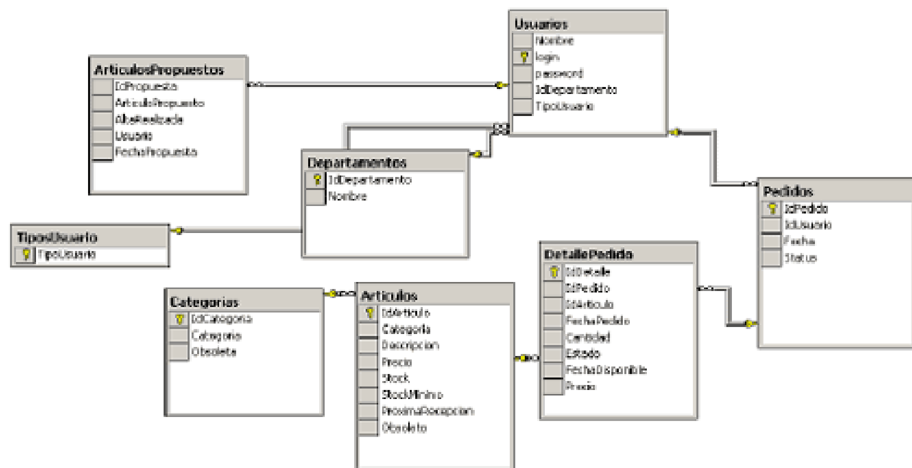
8.2 Base de datos relacional - Una breve definición

Una base de datos relacional es un conjunto de una o más tablas estructuradas en registros (líneas) y campos (columnas), que se vinculan entre sí por un campo en común, en ambos casos posee las mismas características como por ejemplo el nombre de campo, tipo y longitud; a este campo generalmente se le denomina ID, identificador o clave. *Para diseñarlas utilizamos el modelo relacional.*
Las bases de datos relacionales pasan por un proceso al que se le conoce como normalización de base de datos, el cual es entendido como el proceso necesario para que una base de datos sea utilizada de manera óptima. Queda fuera del alcance de este trabajo la normalizan de los modelos de bases de datos.

8.3 Diagramas relacionales

Por lo general luego de normalizar un diagrama conceptual de entidad relación ya sea por medio de la aplicación de las reglas de Codd para tal efecto o por diseñar nuestros esquema de entidad-relación de manera concienzuda y cuidándonos de respetar las formas normales podremos convertir el esquema conceptual de entidad-relación en un esquema lógico relacional para luego crear nuestras tablas en un manejador de base de datos. No existe un estandard para gráfico para este modelo y pero por lo general existen varios.

Figura 8.1 Un ejemplo de modelo relacional de las herramientas de Microsoft

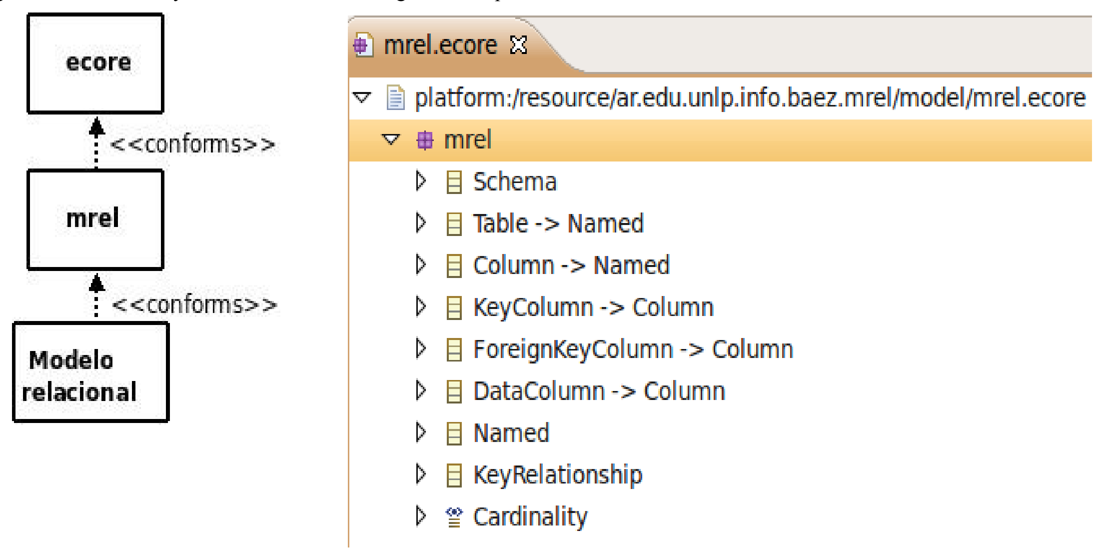


8.4 DSL para el modelo relacional - construcción

8.4.1 Metamodelo relacional(mrel)

El metamodelo relacional que hemos utilizados es el propuesto en [14], llamado metamodelo temporal relacional;se dice temporal porque se asume que preserva los conceptos temporales descriptos en los capítulos anteriores. Se ha adoptado el mismo casi en su totalidad. Se adopta el mismo metamodelo y de esta manera a partir de el se desarrollan las gramáticas abstractas y concretas que utilizamos para derivar código de nuestros editores y lenguajes textuales. Los modelos de entidad relación que se obtienen serán conformes al metamodelo mds. El metamodelo mrel es especificado en el lenguaje ecore.

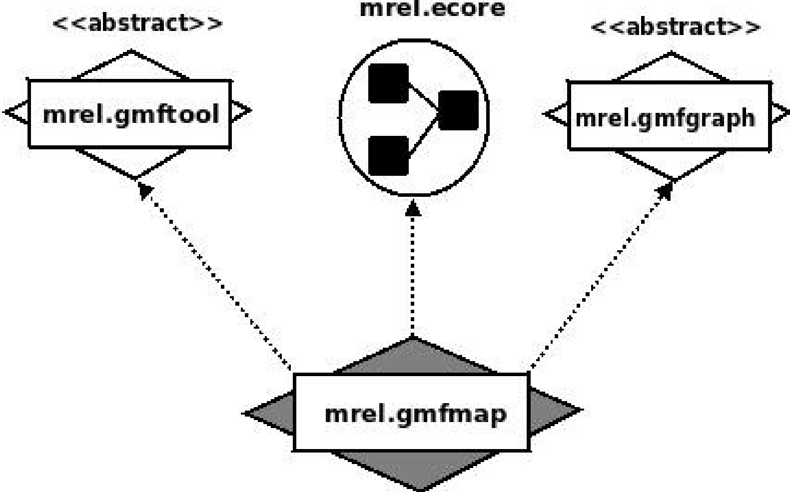
Figura 8.2 Modelo ER y sus metamodelos Figura 8.3 Implementación en ecore de mrel



8.4.2 Notaciones definidas

La implementación de nuestro dsl consta de varias notaciones abstractas y varias concretas desde las cuales derivamos código para generar los editores gráficos.

Figura 8.4 Esquema de las gramáticas definidas y sus relaciones



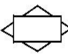
8.4.2.1 Notaciones abstractas relacionales y gráficas

Notaciones abstractas definidas por la herramienta EMF:

- ✔ *mrel.ecore*:Se ha desarrollado un modelo de dominio en lenguaje e-core, y con ella implementamos el metamodelo propuesto en [14]. El mismo casi no ha sido modificado a los fines de la implementación. Este es nuestro modelo de dominios. La definición del modelo de dominio en ecore nos

brinda la ventaja de poder tener varias implementaciones conformes al mismo.

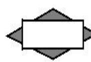
Notaciones abstractas definidas por la herramienta GMF:



- ✔ *mrel.gmgraph*: Es una definición de diagrama que especifica un conjunto de figuras y sus relaciones
- ✔ *mrel.gmftool*: Es una definición de diagrama que define un conjunto de herramientas en un toolbar.

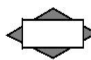
8.4.2.2 Notaciones concretas relacionales

Definidas por la herramienta EMF



- ✔ *mrel.gemodel*:Mediante la herramienta de eclipse EMF hemos se desarrollado está gramática para indicar como convertir nuestro modelo de dominios multidimensional en clases java que podrán ser utilizadas para crear instancias del modelo relacional.

Definidas por la herramienta GMF:



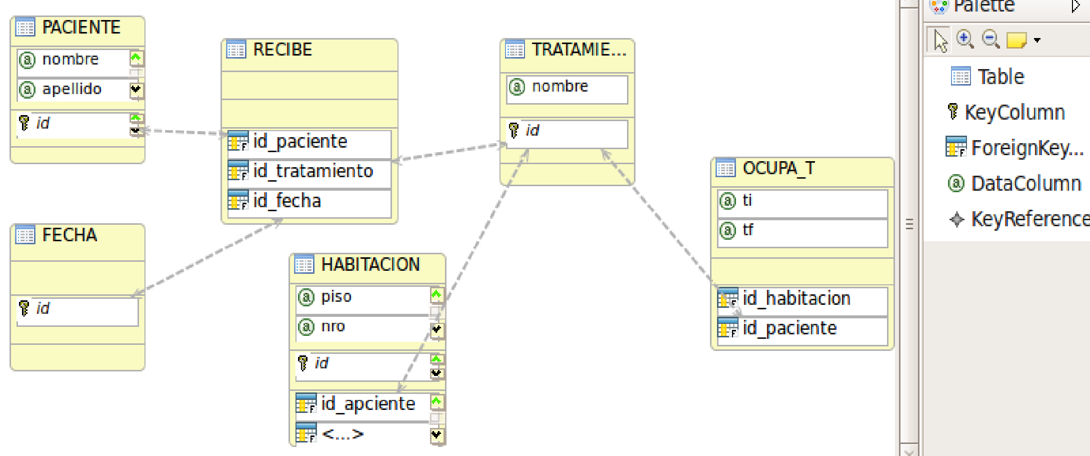
- ✔ *mrel.gmfmap*:Mediante la herramienta de eclipse GMF se ha desarrollado una notación concreta que nos permite combinar las notaciones abstractas anteriores. En la misma se asocian elementos del modelo de dominio con los objetos gráficos y los elementos de la barra de herramientas. Esto se toma como base para derivar el código de nuestro editor gráfico.

8.4.2.3 Código derivado

8.4.2.3.1 Un editor relacional

Valiéndose las notaciones concretas anteriores derivamos código java para nuestro modelo de dominios a través de *mrel.gemodel* y luego un editor gráfico para crear instancias de nuestros modelos a través de mrel.*gmfmap*. Para ello utilizamos los mecanismos que nos brinda la herramienta GMF en conjunto con el lenguaje XPAND para generar código a partir de los modelos anteriores. De esta manera obtenemos un editor gráfico para desarrollar instancias del modelo mmt.

Figura 8.5 Ejemplo del editor relacional



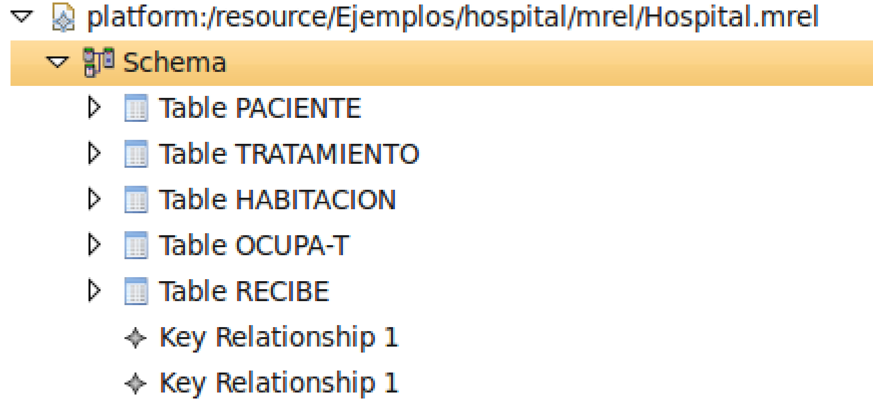
El plugin que implementa nuestro editor gráfico es:

✔ **ar.edu.unlp.info.baez.mrel.diagram**

8.4.2.3.1 Un editor de modelo basado en su estructura

Valiéndose del modelo de dominio en ecore podemos obtener a través de la plataforma EMF un editor basado en la estructura del modelo que le permitirá al diseñador editar y crear instancias del modelo relacional. Es un editor mas rústico y limitado que el editor anterior basado en GMF.

Figura 8.6 Un ejemplo del editor EMF basado en la estructura del modelo relacional



El plugin que implementa nuestro editor basado en la estructura del modelo es:

✔ **ar.edu.unlp.info.baez.mrel.editor**

8.4.2.4 Extensiones desarrolladas para los modelos relacionales

8.4.2.4.1 Validaciones

Mediante el framework de validación que nos provee la plataforma eclipse y utilizando los mecanismos descriptos en 3.2 se ha desarrollado un plugin que forma parte de nuestro esquema relacional y que tiene como objetivo validar la consistencia de nuestros modelos relacionales. Esto para tener la certeza de que nuestros modelos son válidos y para que sean aptos para ser utilizados como modelos fuente para transformaciones que forman parte del proceso de generación del datawarehouse.

Las validaciones sobre el modelo relacional pueden ser realizadas de las siguientes maneras:

- ✔ *Batch*: Ejecutadas de manera explicita y de a lotes.
- ✔ *Implementadas en Java:* Están basadas en el modelo de dominio en lenguaje java derivado de nuestro modelo de dominio en lenguaje ecore(mrel.ecore). Las clases java que implementan nuestras constraints realizan la validación sobre nuestro modelo de dominio en java derivado de nuestro modelo de dominio que forma parte de la sintaxis abstracta(mrel.ecore).

Constraints implementadas

- ✔ *Las tablas deben poseer nombres*

  Al ser las tablas objetos de tipo Named, estos si o si deben poseer nombre.

- ✔ *El esquema no puede estar vació.*

  Si o si deben existir al menos una tabla. La colección llamada tables del objeto Schema de nuestro modelo de dominios no puede ser vaciá.

- ✔ *Las tablas deben poseer al menos algún atributo.*

  No pueden existir tablas si atributos en nuestro modelo de dominio,

  el objeto Schema de nuestro modelo de dominio posee una colección llamada tables esta no puede ser vació.

- ✔ *Las claves foráneas deben apuntar a un atributo clave de otra tabla.*

  Cuando agregamos una columna a una tabla y esta es de tipo clave foránea, esta en nuestro modelo de dominio esta representada por la clase ForeignkeyColumn que posee una referencia a la clave primaria de otra tabla. Esta referencia no puede ser nula. El objeto de nuestro modelo de dominios que nos representan esta asociación es KeyReference.

- ✔ *Los atributos deben poseer nombre y tipo.*

  Si o si debemos asignarle un nombre valido a los atributos de nuestro esquema y estos deben tener un tipo de datos asignado.

El plugin que implementa las validaciones en java es:

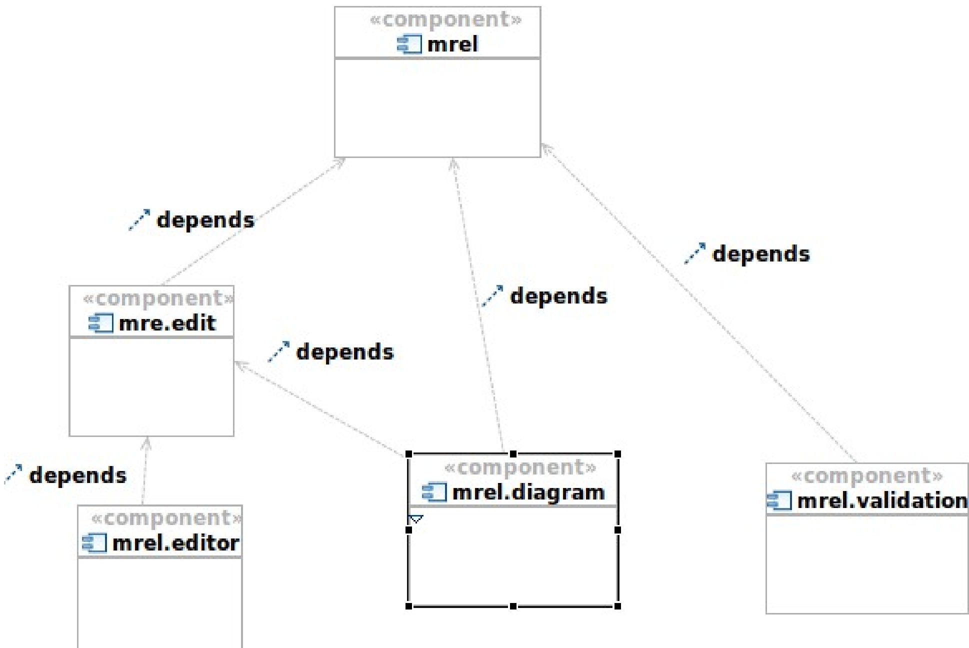
✔ **ar.edu.unlp.info.baez.mrel.validation.general**

8.4.2.5 Plugins que forman parte del DSL relacional

Para resumir es necesario dejar expresado que nuestro conjunto de plugins fue obtenidos en base a la derivación de código y la adaptación de nuestras notaciones concretas con ellos podemos crear y manipular modelos relacionales conformes a nuestro modelo de dominio y un conjunto de ellos forman parte del soporte visual que poseemos para la creación y manipulación de nuestra notación concreta visual.

En la figura 7.7 se muestra un esquema del conjunto de plugins que dan el soporte para edición visual del modelo relacionales y plugins desarrollados.

Figura 8.7 esquema de componentes para el DSL relacional y sus dependencias

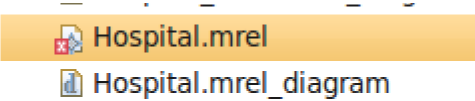


8.5 DSL para el modelo relacional – uso

Desde el punto de vista del diseñador que utiliza nuestra herramienta, el mismo creará modelos relacionales utilizando el desarrollo dirigido por modelos. Utilizando el editor creará modelos que se serializan acorde a una notación abstracta. Por convención adoptamos la extensión ".mrel" para los archivos que contienen la notación abstracta relacional. Tener estas notaciones abstractas basadas en XMI y conforme a nuestros metamodelos en ecore es particularmente útil si queremos que nuestras instancias del modelo relacional sean utilizadas como para ser transformadas,o serializadas de manera estandard para ser leído por otra herramienta. En estos casos la información que contiene aspectos visuales no es necesaria.

También a través del editor serializa una notación concreta que posee aspectos visuales que toma el editor relacional para facilitar la edición relacional, Por convención adoptamos la extensión "mrel_diagram" para la notación concreta visual.
De esta manera la según la visión del diseñador la notación abstracta posee el modelo relacional y una notación concreta posee aspectos visuales. Pudiendo existir mas de una notación concreta para la misma notación abstracta.

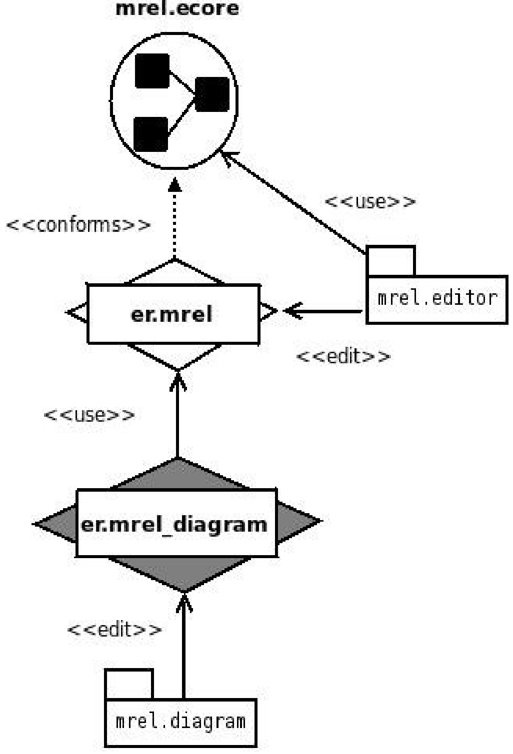
Figura 8.8 Vista de la notación abstracta y concreta que utiliza el diseñador



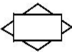
8.5.1 Notaciones del modelo relacional

Al igual que los modelos y metamodelos utilizados para definir las notaciones abstractas y concretas que forman parte de nuestra herramienta, el diseñador que utiliza dicha herramienta desarrolla y manipula modelos relacionales modificando notaciones abstractas y concretas para representar el modelo relacional.

Figura 8.9 relación entre los editores,las notaciones concretas, abstractas y el metamodelo desde el punto de vista del diseñador.



8.5.1.1 Notación abstracta relacional basada en el modelo relacional

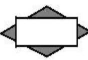


- ✔ *mrel*: Consta de un archivo XML, que contiene la serialización del modelo relacional en formato XMI(xml metada interchange) que nos representa las instancias de nuestro modelo relacional.

Figura 8.10 Un ejemplo de la notación abstracta relacional


```xml
<?xml version="1.0" encoding="UTF-8"?>
<mrel:Schema xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:mrel="http://ar/edu/unlp/info/baez/mrel">
  <tables name="PACIENTE">
    <keyColumns name="id"/>
    <attributeColumns name="nombre"/>
    <fkColumns name="id-HABITACION"/>
  </tables>
  <tables name="TRATAMIENTO">
    <keyColumns name="id"/>
  </tables>
  <tables name="HABITACION">
    <keyColumns name="id"/>
    <attributeColumns name="piso"/>
  </tables>
  <tables name="OCUPA-T">
    <attributeColumns name="ti"/>
    <attributeColumns name="tf"/>
    <fkColumns name="id-HABITACION"/>
    <fkColumns name="id-PACIENTE" referenced="//@fkReferences.1"/>
  </tables>
  <tables name="RECIBE">
    <attributeColumns name="fecha"/>
    <fkColumns name="id-PACIENTE" referenced="//@fkReferences.0"/>
    <fkColumns name="id-TRATAMIENTO"/>
  </tables>
  <fkReferences from="//@tables.4/@fkColumns.0"
       to="//@tables.0/@keyColumns.0"/>
  <fkReferences from="//@tables.3/@fkColumns.1"
to="//@tables.0/@keyColumns.0"/>
</mrel:Schema>
```

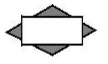

8.5.1.2 Notación concreta relacional basada en el modelo relacional



- ✔ *mrel_diagram:* Consta de un archivo XML en formato xmi que nos representa un conjunto de figuras asociadas a la notación abstracta anterior. Son creados y modificados en base a nuestro editor gráfico relacional. Por cada elemento de la notación abstracta nos indica la figura que lo representa y su posición.

Figura 8.11 Un ejemplo de la notación concreta gráfica correspondiente al diagrama de la figura

```
<?xml version="1.0" encoding="UTF-8"?>
<notation:Diagram xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:mrel="http://ar/edu/unlp/info/baez/mrel"
xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.1/notation"
xmi:id="_no5-seNJEd6BReiw_8pNpA"
type="Mrel" measurementUnit="Pixel">
<children xmi:type="notation:Node"
xmi:id="_oMQjYONJEd6BReiw_8pNpA" type="2002">
<children xmi:type="notation:Node"
xmi:id="_oMhpIONJEd6BReiw_8pNpA" type="5003"/>
<children xmi:type="notation:Node"
xmi:id="_oMo94ONJEd6BReiw_8pNpA" type="7001">
<styles xmi:type="notation:SortingStyle"
xmi:id="_oMo94eNJEd6BReiw_8pNpA"/>
<styles xmi:type="notation:FilteringStyle"
xmi:id="_oMo94uNJEd6BReiw_8pNpA"/>
</children>
<children xmi:type="notation:Node" xmi:id="_oMraIONJEd6BReiw_8pNpA"
type="7002">
<children xmi:type="notation:Node" xmi:id="_oY-wwONJEd6BReiw_8pNpA"
type="3002">
```

- ✔ *Editor basado en la estructura del modelo:* Podemos ver de manera visual y concreta la notación abstracta relacional utilizando el editor basado en la estructura del modelo. En la figura 7.6 se observa un ejemplo de este editor. Su principal utilidad es que podemos utilizarlo para manipular un modelo relacional si tener el editor anterior instalado, Si bien esto mas complicado que hacerlo que con el editor anterior basado en GMF es mucho mas fácil que modificar la sintaxis abstracta manualmente.

8.5.1.3 ¿Notación concreta textual?

Como ocurre con el modelo conceptual de base de datos datos y el modelo de entidad-relación, existen muchas herramientas en el mercado que brindan soporte para el desarrollo de base de datos. En la figura 7.1 se muestra un ejemplo de la notación gráfica utilizada por las herramientas de diseño relacional de la familia Microsoft en Access Sql Server y Visual Studio. Por lo general dichas herramientas no están basadas en el desarrollo dirigido por modelos y utilizan una notación gráfica propia para los diagramas relacionales que imposibilita por lo general que sus diagramas sean exportados a un formato estandard(como CWM[17] por ejemplo) para ser utilizados por otras herramientas. Debido a la simplicidad estructural de los diagramas relacionales y a la madurez de las base de datos relacionales , la estructura gráfica de los diagramas relacionales entre las diferentes herramientas es bastante uniforme y las diferencias son menores. Esta manera graficar los modelos relacionales es bastante común y un camino

natural para el diseño relacional. Es por eso que no existen notaciones textuales ampliamente utilizadas para especificar la estructura de los modelos relacionales. Ese es el motivo por el cual se decidió no implementar un DSL textual para diseñar modelos relacionales a pesar de tener la base suficiente para crear un lenguaje en el cual especificar un modelo relacional.

8.6 Transformaciones del Modelo Multidimensional al Modelo relacional

8.6.1 Transformaciones automáticas

Para construir un modelo nos basamos en el modelo de modelo multidimensional que se describe en 4.1.

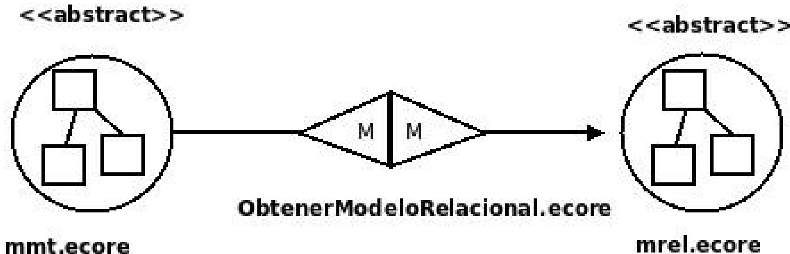
8.6.2 Implementación de la transformación

La transformación fue implementada utilizando el lenguaje ATL. Toma como modelos de entrada modelo multidimensional y a través de una transformación obtenemos un modelo relacional.

Figura 8.12 Algunas reglas que conforman la transformación ATL para crear el modelo relacional

```
rule Schema{
from is: mmt!Schema

to os: mrel!Schema(
 tables <- is.dimensions ->
union(is.hierarchies) -> including(is.facts)
)}

rule TableFromFact{
from n:mmt!Fact
using {IdsACrear:Sequence(String) =
n.dimensions -> collect(x| x.createIdentifiersWithSufix)->flatten();
}
to   t:mrel!Table(
  name <- n.name,
  keyColumns <- n.identifiers,
  attributeColumns <- n.attributes,
  fkColumns <- newFkColumns

),
 newFkColumns:distinct mrel!ForeignKeyColumn
foreach (r in  IdsACrear )(
   name<-r
 ) }
```

Figura 8.13 Transformación del modelo multidimensional al modelo relacional

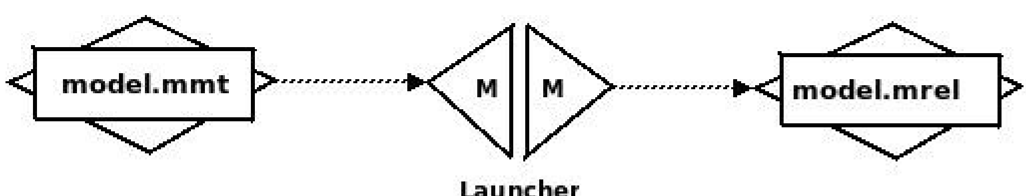


8.6.3 Uso de la transformación

El diseñador transforma un modelo multidimensional en un modelo relacional. La transformación convierte notaciones abstractas acorde a la visión del diseñador de proceso. Toma una notación abstracta de un modelo multidimensional y a partir de esta obtendrá una correspondiente a un modelo relacional.

8.6.4 Ejecución de la transformación

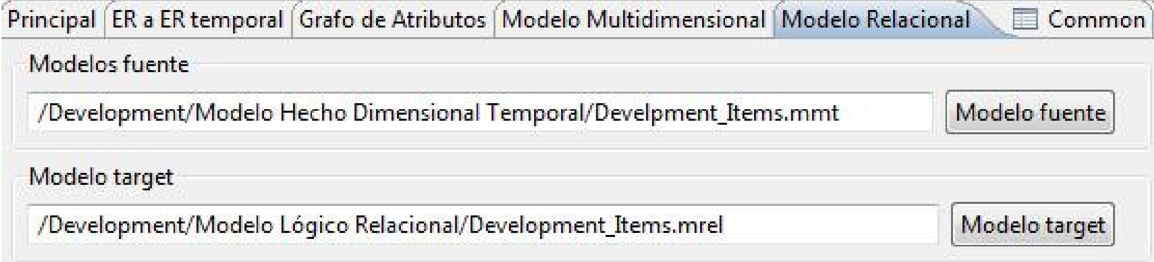
Para ejecutar la transformación el diseñador debe configurar la misma en el launcher de transformaciones que se describe en la sección 3.2.4.



8.6.5 Ejecución de la transformación a través del launcher

Para ejecutar la transformación el diseñador debe configurar la misma en el launcher de transformaciones que se describe en la sección 3.2.4. Para ejecutarla el diseñador debe configurar en el tab principal del launcher debe seleccionar la opción "Modelo Multidimensional al Modelo Relacional", y luego configurar en el tab "Modelo Relacional" los modelos de origen y target de la transformación. Como modelo origen seleccionamos un modelo multidimensional, con extensión ".mmt" y como modelo target seleccionamos el archivo donde queremos que deje la serialización del modelo multidimensional resultante. El mismo es un archivo con extensión ".mrel". En las imágenes se muestra un ejemplo configuración de la transformación. En las imágenes se muestra un ejemplo configuración de la transformación.

Figura 8.15 Un ejemplo de configuración del tab correspondiente a la transformación al modelo relaciona

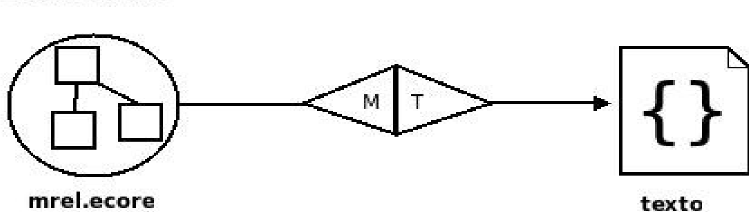


8.7 Generación de código SQL a partir del modelo lógico

8.7.1 Construcción de generador de código

Mediante una transformación modelo a texto llevamos a cabo la transformación con la que obtenemos el código sql que nos permite crear el datawarehouse en una base de datos relacional mysql[29].

Figura 8.15 Transformación modelo a texto que genera código sqll



Nos hemos valido del lenguaje xText para desarrollar una transformación Modelo a Texto que toma una notación abstracta de nuestro modelo relacional y obtenemos a partir de ella código sql.

Figura 8.16 Parte de la transformación modelo a texto que genera código sql

```
«IMPORT mrel»
«DEFINE Root FOR mrel::Schema»
  «FILE "ddlgen.sql"»
  «EXPAND Table FOREACH tables»
   «ENDFILE»
«ENDDEFINE»

«DEFINE Table FOR       mrel::Table»
create Table «name» (
«EXPAND newColumn(this.name)
    FOREACH attributeColumns.union(keyColumns).
             union(fkColumns) SEPARATOR ","»)
«ENDDEFINE»

«DEFINE newColumn(String tableName) FOR KeyColumn-»
CONSTRAINT «this.name»  AUTO_INCREMENT  PRIMARY KEY
«EXPAND type  FOR this.type»
«ENDDEFINE»

«DEFINE newColumn(String tableName) FOR DataColumn-»
«EXPAND type  FOR this.type-» «this.name»
«ENDDEFINE»
```

Mediante un wizard el usuario selecciona el archivo que contiene la notación abstracta relacional y por medio de el mismo obtiene el código sql de creación de tablas que contiene el modelo relacional seleccionado

Figura 8.17 Vista del wizards que dispara la transformación del modelo relacional a código sql

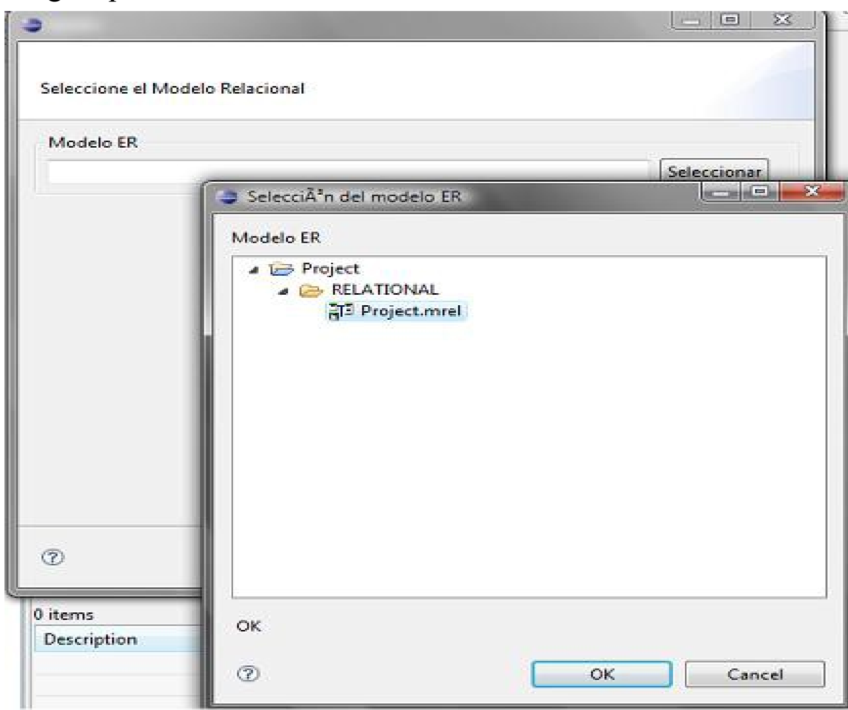


Figura 8.18 Vista del código SQL generado

```
create Table DEVELOPER (
CONSTRAINT id-TEAM  FOREIGN KEY INT(10) REFERENCES DEVELOPER id ON
DELETE RESTRICT  ON UPDATE  RESTRICT
,CONSTRAINT id  AUTO_INCREMENT  PRIMARY KEY INT(10)
,INT(10) name
)

create Table ITEM (
CONSTRAINT id  AUTO_INCREMENT  PRIMARY KEY INT(10)
,INT(10) name
)

create Table fecha (
INT(10) fecha
,CONSTRAINT id  AUTO_INCREMENT  PRIMARY KEY INT(10)
)

create Table TEAM (
CONSTRAINT id  AUTO_INCREMENT  PRIMARY KEY INT(10)
,INT(10) name
)
```

Conclusiones

✔ Aplicabilidad del desarrollo dirigido por modelos

El objetivo principal de este trabajo es demostrar que MDD es aplicable y en ser una referencia de como aplicar MDD. Esto puede ser llevado a cabo utilizando los lineamientos de la de la iniciativa MDA[2], adoptando la visión del desarrollo específico del dominio y utilizando las herramientas que nos brinda la plataforma Eclipse para el modelado. Podemos así construir poderosas herramientas de diseño que dan soporte al desarrollo dirigido por modelos. Usamos MDD para implementar la propuesta del trabajo [14] y construimos una herramienta que da soporte a un proceso basados en MDD. Cumplimos un rol que en [16] se denomina *"Toolsmith".*Existen casos en los que podemos aplicar desarrollo dirigido por modelos para construir una aplicación final. En la cual, por ejemplo en base al modelado podemos obtener una aplicación web que se deriva a partir de un modelo de la misma. Un claro ejemplo de este tipo de herramientas es AndroMDA[29]. Como puede deducirse hemos optado aquí trabajar sobre aquellos casos en los cuales tanto quien que construye el toolkit(nosotros) como el usuario final(el diseñador de base de datos) utilizan una visión de desarrollo guiado por modelos. De esta manera existen aquí los roles definidos en [16], el *Toolsmith*(quien desarrolla la herramienta) y el *practicioner*(quien la utiliza). Ambos tienen una "visión MDD". Es por eso que en cada DSL construido hemos mostramos como se desarrollo y como se usa. Ambos son casos de desarrollo guiado por modelos y hemos comprobado que es el camino correcto para construir herramientas de modelado y de generación de código. Se logra de esta manera el objetivo que se intento alcanzar sin éxito con las denominadas herramientas CASE de épocas pasadas.

✔ Aplicabilidad del desarrollo dirigido por modelos para diseñar base de datos y datawarehousing

Existen muchos dominios en los cuales los modelos se vienen utilizando de manera sistemática desde hace mucho tiempo. Estos dominios son el lugar mas fértil para aplicar la visión del desarrollo guiado por modelos. Esto se debe a que los modelos ya son ciudadanos de primera clase, en muchos casos se han convertido en estándares de facto y son utilizados de manera natural. Un objetivo primordial de este trabajo es mostrar como aplicar un desarrollo dirigido por modelos en estos ámbitos. Demostramos mediante un ejemplo lo conveniente de aplicar el desarrollo guiado por modelos en un ámbito "acostumbrado" a los modelos como es el área de las bases de datos y datawarehousing. La pauta sería formalizar los modelos existentes en base al metamodelado, escribir las transformaciones llevadas a cabo para pasar de un modelo a otro y que hoy en día se realizan de manera informal. Así como también dar un soporte a quien las usa,

de manera tal que estas herramientas no solo sean un juguete de laboratorio. Para ello hay que proveer a los diseñadores que ya trabajan en esos dominios con herramientas de modelado(toolkits) que den soporte a la manipulación y transformación de sus modelos y son *específicos* de su dominio). Es por eso que en este trabajo demostramos estos hechos mostrando como construir un Toolkit DSL para aplicar MDD en la construcción de un datawarehouse temporal siguiendo los lineamientos de la propuesta de [14].

✔ El desarrollo dirigido por modelos con la tecnología y estándares actuales

Este trabajo se basó y alienta el uso de las herramientas de modelado de la plataforma eclipse a través de su iniciativa de código abierto Eclipse Modeling Project[20]. En tal plataforma se brinda da soporte a la construcción de DSLs utilizando frameworks que implementan muchos de los estándares de la OMG. Estos frameworks gozan de un alto grado de madurez , una gran comunidad de personas que usan dichas herramientas y muchos casos de uso para tomar de referencia.

✔ Sobre el estandard CWM[17] de la OMG[2]

Hemos analizado aquí el estandard CWM[17] que fue pensado para el intercambio de modelos de base de datos y datawarehousing. Pudimos concluir que el mismo es muy amplio ,abarcador y poco apto para ser utilizado como notación abstracta en la cual basarse para construir editores y lenguajes específicos en el área de base de datos. Es por eso que las herramientas de diseño base de datos y datawarehousing no deben basarse en tal estandard para sus DSLs pero si brindar compatibilidad con el mismo a través de transformaciones.

Trabajos relacionados

- ✔ El trabajo “Aplicando MDA al Diseño de un Datawarehouse Temporal.” que es una propuesta formal y fue utilizado como objetivo para mostrar la implementación de un toolkit DSL.
- ✔ En el trabajo Usando ATL en la Transformación de Modelos Multidimensionales Temporales” se realizó un análisis del uso del lenguaje ATL para llevar a cabo las transformaciones modelo a modelo propuestas en [14]. Debido a las conclusiones de ese trabajo se decidió el uso del lenguaje ATL frente a otros(QVT por ejemplo).

Trabajos Futuros

✔ Compatibilidad con CWM[17]

Debemos llevar a cabo las transformaciones necesarias para que las instancias de los modelos puedan ser exportadas a formatos compatibles con CWM[17] y poder importar modelos serializados en formatos compatibles con CWM[17] a nuestro toolkit a fin de lograr la portabilidad basada en dicho estandard y lograr la independencia de la plataforma.

✔ Implementación de aspectos del diseño de base de datos y datawarehousing no considerados

Se deben implementar aspectos no considerados en nuestros modelos, como pueden ser los conceptos de generalizaciones, agregaciones y normalizaciónes de los modelos de Entidad-relación.

✔ Derivación automática de consultas

Es posible a a partir del grafo obtener consultas SQL para obtener datos y reportes desde el datawarehouse construido. Este tema está siendo estudiado por los autores del trabajo [14] y podría ser tomado su trabajo como objetivo para llevarlo a la práctica.

✔ Derivación automática de ETL.

A partir de nuestros modelos para derivar el datawarehouse se debe derivar también aquellas funciones que realizan la extracción, transformación y carga de datos del datawarehouse desde las bases de datos operacionales existentes.

REFERENCIAS BIBLIOGRAFICAS

1. Object Management Group OMG
2. Model Driven Architecture(MDA). MDA Guide Version 1.0.1, http://www.omg.org/mda/
3. Domain Specific Language. Martin Fowler http://martinfowler.com/bliki/DomainSpecificLanguage.html
4 Desarrollo de Software Dirigido por Modelos, Conceptos teóricos y su aplicación práctica. Claudia Pons, Roxana Giandini y Gabriela Pérez
5 MOF Meta Object Facility (MOF) 2.0. OMG Adopted Specification. Octubre 2003, http://www.omg.org/mda/
6. Unified Modeling Language(UML) http://www.uml.org/
7. UML profiles http://www.uml.org/#UMLProfiles
8. The Entity-Relationship Model: Toward a Unified View of Data (1976). Peter Pin-shan Chen
9.. A Relational Model of Data for Large Shared Data Banks(1970). Edgar Frank Codd
10. Standard Query Language(SQL). http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt
11. Gupta, H., Harinarayan, V. Rajaraman, A. and J. Ullman. Index Selection for OLAP. Proceeding ICDE 1997.
12. W. Inmon. Building the Data Warehouse. John Wiley & Sons, 2002.
13. Agrawal R, Gupta A, Sarawagi S., Modeling Multidimensional Databases, Research Report, IBM
14. Carlos Neil y Claudia F. Pons Aplicando MDA al Diseño de un Datawarehouse Temporal.
15. M Golfarelli, D. Maio y Stefano Rizzi. A conceptual model for datawarehouses.
16 Eclipse Modelling Project as a dsl toolkit. Richard Gronback
17 Common Warehouse Metamodel™ (CWM™) Specification, v1.1 http://www.omg.org/spec/CWM/1.1/
18 Eclipse project http://www.eclipse.org
19 Eclipse Rich Client Platform(RCP). http://wiki.eclipse.org/index.php/Rich_Client_Platform
20 Eclipse Modeling Project. http://www.eclipse.org/modeling/
21.El modelo relacional en el marco de transformación de modelos. Jerónimo Irazabal.
22 Lenguaje ecore. http://www.eclipse.org/modeling/emf/?project=emf
23 Atlas transformation Languaje(ATL) http://www.eclipse.org/m2m/atl/
24 DSM Forum http://www.dsmforum.org/
25 Domain-Specific Modeling and Model Driven Architecture. Steve Cook
26 Language Workbenches and Model Driven Architecture. Martin Fowler http://martinfowler.com/articles/mdaLanguageWorkbench.html
27 XMI(XML Metada Interchange), http://www.omg.org/technology/documents/formal/xmi.htm
28 Mysql 5.0 Reference manual. http://dev.mysql.com/doc/refman/5.0/en/index.html

29 AndroMDA, http://www.andromda.org/index.php
30 Developing time oriented Database Applications in SQL. Richard T Snodgrass