Diseño de un Almacén de Datos Histórico en el Marco del Desarrollo de Software Dirigido por Modelos

Mg. Carlos Gerardo Neil

Directora: Dra. Claudia Pons
Codirector: Dr. Gustavo Rossi

Tesis presentada para obtener el grado de Doctor en Ciencias Informáticas

Facultad de Informática
Universidad Nacional de La Plata

- Noviembre de 2010 -

Jurados

Dr. **José Angel Olivas Varela**
Departamento de Tecnologías y Sistemas de Información
Universidad Castilla-La Mancha. España

Dra. **Alejandra Cechich**
Facultad de Informática
Universidad Nacional del Comahue. Argentina

Dr. **Juan María Ale**
Facultad de Ingeniería
Universidad de Buenos Aires. Argentina
Facultad de Ingeniería (Departamento de Informática)
Universidad Austral. Argentina

Agradecimientos

Quiero agradecer inicalmente a mis directores, Claudia Pons y Gustavo Rossi, y al decano de la facultad de Tecnología Informática de la UAI, Marcelo De Vincenzi.

A Claudia porque me acompaño la mayor parte del camino, siempre dispuesta con su generosidad y simpatía; a Gustavo, porque me ayudó en los comienzos y me dio la oportunidad de acercarme a la Universidad Nacional de la Plata para iniciar mis estudios y, a Marcelo, por su apoyo constante y su confianza en mi tarea en la Universidad Abierta Interamericana.

Extiendo mi agradecimiento a mi familia que siempre fue y será un constante apoyo, a mis seres queridos, a mis compañeros de trabajo, a mis amigos del deporte, a mis amigos de la vida y al amor de mi vida.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Un ***Decision Support System*** **(DSS)** asiste a los usuarios en el proceso de análisis de datos en una organización con el propósito de producir información que les permita tomar mejores decisiones. Los analistas que utilizan el **DSS** están más interesados en identificar tendencias que en buscar algún registro individual en forma aislada [HRU96]. Con ese propósito, los datos de las diferentes transacciones se almacenan y consolidan en una base de datos central denominada ***Data Warehouse*** **(DW)**; los analistas utilizan esas estructuras de datos para extraer información de sus negocios que les permita tomar mejores decisiones [GHRU97].

Basándose en el esquema de datos fuente y en los requisitos de información de la organización, el objetivo del diseñador de un **DSS** es sintetizar esos datos para reducirlos a un formato que le permita, al usuario de la aplicación, utilizarlos en el análisis del comportamiento de la empresa.

Dos tipos diferentes (pero relacionados) de actividades están presentes: el diseño de las estructuras de almacenamiento y la creación de consultas sobre esas estructuras. La primera tarea se desarrolla en el ámbito de los diseñadores de aplicaciones informáticas; la segunda, en la esfera de los usuarios finales. Ambas actividades, normalmente, se realizan con escasa asistencia de herramientas automatizadas.

1.1. Problemas y Soluciones

A partir de lo expresado anteriormente Identificamos, por consiguiente, tres problemas a resolver: a) la creación de estructuras de almacenamiento eficientes para la toma de decisión, b) la simplificación en la obtención de la información sobre esas estructuras para el usuario final y, c) la automatización, tanto del proceso de diseño de las estructuras de almacenamiento, como en la elaboración iterativa de consultas por parte del usuario de la aplicación.

La solución propuesta es el diseño de una nueva estructura de almacenamiento que denominaremos ***Historical Data Warehouse*** **(HDW)** que combina, en un modelo integrado, un ***Historical Data Base*** **(HDB)** y un **DW;** el diseño de una interface gráfica, derivada del **HDW,** que permite realizar consultas en forma automática y, por último, el desarrollo de un método de

diseño que engloba ambas propuestas en el marco del ***Model Driven Software Development* (MDD)**.

1.1.1. Creación de Estructuras de Almacenamiento Eficientes

El **DW** es una copia de los datos de las transacciones de una organización, estructurados específicamente, para realizar consultas y análisis [Kim96]; este tipo de almacenamiento juega un rol central en los actuales **DSS** debido a que brindan información crucial para el proceso de toma de decisión estratégica [Inm02].

Una característica distintiva del **DW** es que el tiempo es una de las dimensiones para el análisis [CD97], [GMR98a], pero éste hace referencia al instante en que se realizó una transacción, por lo tanto, no especifica ni cómo ni cuándo han variado, a través del paso del tiempo, los valores de las entidades, atributos e interrelaciones vinculadas a esas transacciones. Si bien el ***Temporal Data Warehouse* (TDW)** contempla, además de la dimensión temporal, otros aspectos vinculados con el tiempo [HMV99], [EC00], [EKM01], este modelo considera solo las modificaciones que se producen en el esquema del **DW**, tanto en las dimensiones como en las jerarquías.

Por lo tanto, un problema a resolver en este tipo de estructura ***Multidimensional* (MD),** en vistas a la necesidad de registrar valores que permitan evaluar tendencias, variaciones, máximos y mínimos, es de qué manera plasmar en el diseño de la estructura **MD** cómo los valores de las entidades, atributos o interrelaciones pueden variar en el tiempo; ya que, aunque los datos necesarios estuvieren almacenados, los mecanismos de búsqueda temporales resultarían complejos [NA02].

1.1.2. Simplificación en la Obtención de la Información

Las herramientas de consulta que dependen de la habilidad de los programadores para un uso eficaz y eficiente, imponen una carga cognitiva que puede disminuir la productividad de los usuarios [ON01]. Resulta, por lo tanto, un desafío para usuarios no técnicos especificar consultas en una estructura de **DB** [LJ09], más aun si los datos están almacenados en una estructura ***Temporal Multidimensional* (TMD)**.

La forma tradicional de acceder a una base de datos ha sido mediante consultas por medio del ***Structured Query Language* (SQL)**, un lenguaje diseñado específicamente para crear, organizar y consultar base de datos.

Debido a la complejidad en la formulación de consultas no triviales se han propuesta diversos enfoques para hacerlas más accesible a un espectro mayor de usuarios [FKSS06]; siguiendo esta línea, el uso de lenguajes gráficos para la realización de consultas, comparado con la escritura de expresiones algebraicas, facilitaría sobremanera las tares del usuario final. Por lo tanto, un lenguaje gráfico debería operar sobre una vista gráfica explícita del esquema conceptual y las consultas beberían ser expresadas sobre la representación gráfica en forma incremental [RTTZ08].

1.1.3. Automatización del Proceso de Desarrollo de Software

Aunque han sido propuestos diversos métodos[1] que permiten derivar el esquema conceptual **MD** a partir de los datos fuentes de la organización y/o de los requerimientos del usuario (ver [CT98], [GMR98a], [TBC99]), la mayoría de ellos deben ser realizados manualmente [RA08]. Por otro lado, un ambiente visual

[1] Utilizaremos el término método para referirnos a una forma específica de resolver un problema y metodología como el estudio de un método.

centrado en el usuario debería incluir herramientas automatizadas para la formulación de consultas y proveer diferentes metáforas de visualización [KG95].

Una solución a estos problemas lo plantea **MDD**, este enfoque se ha convertido en un nuevo paradigma de desarrollo de software que promete mejoras en la construcción de software basándose en un proceso guiado por modelos y soportado por potentes herramientas. En **MDD**, la construcción de software se realiza a través de un proceso guiado por modelos y soportado por potentes herramientas que generan código a partir de aquellos. Este nuevo paradigma tiene como objetivos mejorar la productividad y la calidad del software generado mediante una reducción del salto semántico entre el dominio del problema y de la solución [PGP09].

La idea clave subyacente es que, si se trabaja con modelos, se obtendrán importantes beneficios tanto en productividad, portabilidad, interoperatividad y mantenimiento y documentación [KWB03].

1.2. Nuestra Propuesta: El Data Warehouse Histórico

Por lo considerado anteriormente, respecto al almacenamiento de datos, la recuperación de información y la automatización de ambos procesos, surge la necesidad de un método, en un ambiente asistido por herramientas automatizadas, que faciliten el proceso de diseño de una nueva estructura de almacenamiento que denominaremos **HDW**, propuesta que combina, en un modelo integrado, un **HDB** y un **DW** y cuyo objetivo es resolver las limitaciones temporales de las estructuras **MD** tradicionales y que, además, le permita, al usuario final, realizar sobre esa estructura **TMD**, en forma automática y asistido por una intuitiva interface gráfica, consultas temporales, de toma de decisión o combinaciones de ambas.

1.3. Objetivo de la Tesis

Los objetivos plateados en la tesis son: a) presentar una solución a las necesidades temporales explícitas en los modelos **MD** mediante el diseño de un **HDW**, b) enmarcado en el enfoque **MDD**, derivar una implementación física en un ***Relational Data Base Management System*** **(RDBMS)** del **HDW**, c) proponer el diseño de una interface gráfica, derivada del **HDW,** en el marco de **MDD**, para la recuperación de información y, d) resolver en forma automática, mediante sentencias **SQL**, las consultas realizadas por el usuario final en el **HDW.**

La propuesta está resumida en los siguientes ítems:

- La creación de un modelo y un método para el diseño automático de un **HDW** que incluya, además del hecho principal de análisis, estructuras temporales vinculadas a los niveles de las jerarquías dimensionales que posibiliten registrar los datos y recuperar la información que variase en el tiempo.

- La creación de un lenguaje visual de consultas, derivado de la estructura **TMD** que permita realizar, en forma automática, las consultas temporales y **MD**.

- La aplicación del paradigma **MDD** en la construcción de un **HDW** y de una interface gráfica, que facilite el diseño mediante un proceso guiado por modelos y asistido por herramientas automatizadas que

generen código a partir de aquellos, reduciendo, de este modo, el salto semántico entre el dominio del problema y la implementación de su solución.

1.4. Contribuciones Principales

Las contribuciones principales de la tesis son:

- La propuesta de un nuevo modelo de datos temporal (**HDB**) simplificado que permite registrar la variación de los valores de atributos, entidades e interrelaciones que se modifiquen en el tiempo.

- La propuesta de una nueva estructura de almacenamiento de datos (**HDW**), que combina e integra en un solo modelo, un **DW** y un **HDB.**

- La creación de un método de diseño que, a partir de un modelo de datos conceptual y mediante sucesivas transformaciones, permite obtener una implementación lógica de un **HDW** en un **RDBMS.**

- La implementación, mediante el enfoque **MDD,** del método de diseño de un **HDW** mediante la transformación automática de sus modelos abstractos a modelos concretos.

- El desarrollo de un entorno gráfico derivado automáticamente del **HDW,** en el marco **MDD,** para la realización de consultas sobre la estructura **TMD.**

- La generación automática, utilizando el enfoque **MDD**, de sentencias **SQL** que permite realizar, sobre el **HDW,** tanto las consultas características de un **DW** como las típicas de un **HDB.**

- La creación de un prototipo, basada en tecnología ECLIPSE**,** que implemente el método de diseño del **HDW,** la interface gráfica de consultas y la realización de sentencias **SQL**.

1.5. Estructura General de la Tesis

La estructura general de la tesis está compuesta por: trece capítulos, cada uno de ellos desarrollan los aspectos principales del trabajo; cinco anexos, que describen los lenguajes de transformación utilizados, el entorno de desarrollo, las transformaciones y la corroboración empírica; por último, los acrónimos y las referencias.

1.5.1. Capítulos

- En el capítulo 2, ***Base de Datos Temporales***, se detallan las características principales de las Bases de Datos Temporales y su diferencia con las Bases de Datos Operacionales. Se presentan los principales conceptos temporales y los diversos tipos de bases de datos a partir de las dimensiones temporales. Por último, se muestran las características principales de los modelos de datos temporales y sus posibles implementaciones y se resumen las diferentes propuestas

sobre extensiones del modelo Entidad Interrelación para capturar aspectos temporales.

- En el capítulo 3, ***Diseño de una Base de Datos Histórica,*** se describe el modelo de datos temporal propuesto, que será utilizado en el diseño de Data Warehouse Histórico. Se detallan los principales componentes del modelo y se describen las transformaciones informales para obtener, a partir de un modelo de datos, un modelo de datos temporal que admite preservar la historia de entidades, atributos e interrelaciones. Luego, se establecen los criterios para su implementación mediante sentencias SQL en un Sistema Administrador de Base de Datos**.** Por último se detallan los alcances y limitaciones del modelo.

- En el capítulo 4, ***Data Warehouse,*** se presentan las características principales de un Data Warehouse y sus diferencias con los Bases de Datos Operacionales. Se detallan los distintos tipos de jerarquías dimensionales, las diferentes formas de visualizar los datos en la estructura Multidimensional y las diferentes alternativas de implantación. Se contrastan las diferencias entre el Data Warehouse con el Data Warehouse Temporal y el Data Warehouse Histórico y, por último, se describen los más importantes trabajos donde se vincula al modelo Entidad Interrelación con el diseño de un Data Warehouse**.**

- En el capítulo 5, ***Diseño de un Data Warehouse Histórico,*** se desarrolla el método de diseño de un Data Warehouse Histórico a partir de un modelo Entidad Interrelación**.** Primero**,** se describe el modelo **MD** propuesto. Luego, se detallan, de manera informal, todas las transformaciones necesarias para obtener sentencias SQL que permitirán implementar el modelo propuesto en un Sistema Administrador de Base de Datos.

- En el capítulo 6, ***Recuperación de información en Estructuras de Almacenamiento,*** se presenta, primeramente, un conjunto de estructuras de almacenamientos, junto con sus limitaciones a la hora de resolver las necesidades de información del usuario, Luego, con el objetivo de resolver aquellas y a partir del modelo Temporal y del Multidimensional propuestos, se detallan las principales consultas temporales y de toma de decisión que pueden realizarse sobre dichos modelos. Por último, se describe cómo implementar las consultas Temporales y Multidimensionales mediante sentencias SQL.

- En el capítulo 7, ***Consultas en un Data Warehouse Histórico,*** se presenta una interface gráfica, derivada del modelo Multidimensional Temporal, que permite realizar, mediante marcas en un gráfico, consultas temporales y de toma de decisión. Primeramente, se muestra informalmente la transformación, paso a paso, del modelo Multidimensional Temporal al Grafo de Atributos. Luego, se detallan los patrones de consultas temporales y de toma de decisión y, a continuación se especifica cómo, mediante marcas en el Grafo de Consulta, se pueden implementar, a través de transformaciones, las consultas temporales y multidimensionales que permiten expresarlas en sentencias SQL**.**

- En el capítulo 8, ***Arquitectura de Software Dirigida por Modelos***, se presenta, primeramente, una visión general del enfoque de la arquitectura de software dirigida por modelos. Luego, se describen los diferentes niveles de madurez de los modelos. Posteriormente, se detallan los diferentes modelos usados y los tipos básicos de transformaciones entre ellos. luego, se detallan las ventajas de la arquitectura de software dirigida por modelos sobre el proceso tradicional de desarrollo de software; posteriormente, se especifican las diferentes propuestas del enfoque dirigida por modelos. Por último, se detallan los distintos grados y métodos de transformación de modelos.

- En el capítulo 9, ***Lenguajes para Transformación de Modelos***, se plantea, primero, una visión general del proceso de transformación. Luego, se analizan los principales mecanismos existentes para la definición de transformaciones modelo a modelo; a continuación se introduce el lenguaje QVT, un estándar para transformaciones especificado por el OMG; luego, se analizan algunos de los principales requisitos para que un lenguaje de transformaciones modelo a modelo sea práctico y usable; a continuación, se presentan las principales características de las transformaciones modelo a texto, en particular, se introducirá el lenguaje estándar MOF2Text. Por último, se detallan las principales herramientas, tanto de transformación modelo a modelo, como de transformación modelo a texto.

- En el capítulo 10, ***Diseño de un Data Warehouse Histórico en el Marco MDD***, se muestra, primeramente, una visión general del concepto de metamodelo. Luego, se detalla la descripción de todos los metamodelos usados en las transformaciones: el metamodelo de datos, el metamodelo de datos temporal, el metamodelo del grafo de atributos, el metamodelo multidimensional temporal, el metamodelo relacional y, por último, el metamodelo del grafo de consultas. Por último, para cada uno de ellos, se establecen un conjunto de restricciones OCL asociados a los mismos.

- En el capítulo 11, ***Prototipo de Implementación,*** se detalla el prototipo desarrollado en ECLIPSE que permite implementar el DW histórico. Mediante el uso del ejemplo desarrollado en los capítulos precedentes detallamos, para a paso, cómo generar los modelos y las transformaciones necesarias, tanto para la obtención del modelo multidimensional temporal, como así también obtener las consultas automáticas sobre el mismo.

- En el capítulo 12**, *Trabajos Relacionados***, se presentan, primeramente, los principales trabajos de investigación vinculados al uso del enfoque conducido por modelos en el diseño de estructuras de almacenamiento; a continuación, se detallan diferentes trabajos de investigación relacionados con el uso de consultas gráficas sobre estructuras de datos. Posteriormente, establecemos las diferencias de nuestra propuesta respecto de los trabajos relacionados. Al final, presentamos los principales trabajos que hemos desarrollado vinculados a la temática propuesta.

- En el capítulo 13**, *Conclusiones***, se presenta, primero, una síntesis de la tesis respecto del diseño de Data Warehouse histórico en el contexto de la arquitectura de software dirigida por modelos**,** donde se muestran las diferencias entre nuestro planteo respecto de los trabajos vinculados. Luego, se detallan los aportes más significativos de nuestra tesis. Finalmente, se presentan distintas líneas de investigación que permitirán continuar con el trabajo presentado.

1.5.2. Anexos

- En el Anexo I**, *Eclipse*,** se describe, primeramente, las características principales del EMF**,** el *framework* de Eclipse para modelado, que permite la generación automática de código para construir herramientas y otras aplicaciones a partir de modelos de datos estructurados; luego detallamos el GMF**,** un *framework* de código abierto, completamente integrados a Eclipse, que permite construir editores gráficos. Ambos *frameworks* son utilizados para el desarrollo del prototipo de implementación del capítulo 11.

- En el Anexo II, ***ATL,*** se detallan las construcciones más importantes del lenguaje de transformación ATL, lenguaje que permite, al desarrollador, especificar la forma de producir un conjunto de modelos destinos a partir de un conjunto de modelos fuentes. El lenguaje ATL es utilizado para describir las transformaciones.

- En el Anexo III, ***MOFScript,*** *se* detallan las construcciones más importantes del lenguaje de transformación, modelo a texto, MOFScript, herramienta que asiste en el proceso de desarrollo de software, tanto en la generación de código de implementación, como de documentación, a partir de modelos.

- En el Anexo IV, ***Transformaciones en ATL y MOFScript****,* se detallan todas las transformaciones que inicialmente fueron descritas de manera informal en el capítulo 5 (Diseño de un Data Warehouse Histórico) y en el capítulo 7 (Consultas en un Data Warehouse Histórico). Las transformaciones M2M, serán descritas en ATL y las transformaciones M2Text, en MOFScript.

- En el Anexo V, ***Corroboración Empírica de la Propuesta,*** se evalúa empíricamente la propuesta de la tesis a partir del uso, por parte de usuarios reales y en un ambiente controlado, del método de diseño de un Data Warehouse Histórico, la estructura de almacenamiento integrada y la interface gráfica de consultas a partir del uso del prototipo que implementa las ideas principales desarrolladas en la tesis.

1.5.3. Acrónimos

1.5.4. Referencias

Capítulo 2

Base de Datos Temporales

2.1. Introducción

Los modelos de datos tradicionales capturan un único estado de la realidad que representan, usualmente el actual; un ***Operational Data Base*** (**ODB)** se diseña con el objetivo de representar esa visión particular de la información; los ***Data Base Management System*** **(DBMS)** que lo implementan permiten, mediante operaciones de actualización, la transición de un estado a otro, reemplazando los valores anteriores por otros nuevos. Respecto de las consultas, se asume que todos los hechos almacenados en la base de datos son válidos en el instante en que ésta es evaluada. Por lo tanto, si no se dispone de un modelo de representación adecuado, esa información tendrá validez en un lapso determinado, no siempre explicitado.

Por ejemplo, supongamos que el precio de un producto, en un momento dado, es de "p" pesos; si ese valor se modificara, ese hecho debería ser reflejado en la base de datos mediante actualizaciones o borrados, según sea la naturaleza del cambio; si el precio ahora aumentara "a" pesos, el nuevo valor debería ser "p+a" pesos. Por lo tanto, debido a esta modificación, el estado previo de la base de datos (en particular, el precio anterior del producto) se perdería. A este tipo de base de datos se la denomina "*instantánea*" (*snapshot)*, ya que ofrece una "*fotografía*" de la realidad en un momento determinado.

En muchas organizaciones, este tipo de base de datos satisfaría las necesidades de información del usuario. Por otro lado, el aspecto histórico de las base datos, aquel que captura la variación temporal de los hechos que ocurren en la realidad se torna necesario en aplicaciones tales como la gestión de proyectos, las historias clínicas en hospitales, el registro temporal del mantenimiento de equipos, el periodo de validez de leyes, etc. En este contexto, y siguiendo con el ejemplo anterior, si ahora el precio del producto fuera de "p" pesos, desde una fecha "f", hasta el día de hoy, estaríamos registrando, en este caso, no solamente la información del precio del producto sino que, además, el período de validez de ese precio.

Presentaremos en este capítulo las características principales de las **TDB**, su diferencia con las **ODB**, los principales conceptos temporales y los diversos tipos de **TDB** a partir de las dimensiones temporales. A continuación, se expondrán las características principales de los modelos de datos temporales y

sus posibles implementaciones y, por último, se resumirán las diferentes propuestas sobre extensiones del modelo **ER** para capturar aspectos temporales.

2.2. Base de Datos Temporal

En un **ODB**, la información se hace efectiva en el momento en que se asienta en él y ésta se considera válida hasta que una nueva actualización la modifique; por lo tanto, no existe una distinción entre el tiempo de registro de esa información en la base de datos y el período durante el cual los valores específicos de los hechos vinculados a esa información son válidos en el universo de discurso. Así, la base de datos representa únicamente el estado actual y no la historia de los hechos de la realidad que se estuviere modelando.

Una base de datos temporal contiene datos actuales, históricos (y futuros); esto implica, como posición extrema, que los datos serán solo insertados pero nunca eliminados ni actualizadas (posición asumida por los **DW** usados en procesos de toma de decisión); en el otro extremo están las bases de datos instantáneas, que solo contienen datos actuales, donde éstos son actualizados o eliminados cuando los hechos que representan dejan de ser ciertos.

Existen varios dominios de aplicaciones donde es necesario acceder no solamente al más reciente estado de la base de datos, sino también a estados pasados y aun futuros. Para satisfacer los requerimientos vinculados con el tiempo se precisarán modelos de datos que incorporen explícitamente los aspectos temporales y que permitan registrar, tanto la información que variase con el tiempo, como la que no. El término dato temporal se refiere al concepto en el cual datos u objetos tienen algún tipo de información temporal asociada con él; por ejemplo, mediante alguna marca de tiempo.

El objetivo de este tipo de almacenamientos, denominado ***Temporal Data Base*** **(TDB),** está descrito sintéticamente en la definición planteada en [Jen+94] *"Una base de datos que registra algún aspecto vinculado al tiempo, excluyendo al tiempo definido por el usuario"*

Surgen ahora dos cuestiones en el diseño de una **TDB**; primero, definir cómo representar el tiempo en los modelos temporales y, luego, identificar diferentes dimensiones temporales respecto de las cuales los datos serán almacenados. Por ejemplo, el momento en que los datos son actualizados en las base de datos no necesariamente deberían ser los mismos de aquel donde parte de la realidad representada por esos datos ha cambiado.

2.2.1. Definiciones Básicas

Definiremos, a continuación, los principales conceptos temporales [Jen+94].

2.2.1.1. Instante

Un instante (*instant*) se define como un punto en un eje que representa al tiempo. Se han propuesto varios modelos, tanto en la literatura filosófica como en la lógica, relacionados con el tiempo; un instante puede ser considerado discreto o continuo; los instantes en un modelo discreto son isomórficos a los números naturales, es decir, cada instante tiene un único sucesor; por otro lado, los instantes en el modelo continuo son isomórficos a los números reales, esto es, entre cualesquiera de dos instantes siempre puede haber otro.

2.2.1.2. Intervalo de Tiempo

Un intervalo de tiempo (*time interval*) es el tiempo transcurrido entre dos instantes. En general, se considera que un modelo de datos discreto es

adecuado en la mayoría de las aplicaciones de **TDB**; el intervalo de tiempo, por lo tanto, se define como el tiempo transcurrido entre dos instantes, uno inicial y otro final que detallan, respectivamente, el instante inicial y final del mismo [JS97a]. Además, se define a un elemento temporal (*temporal elements*) como la unión finita de intervalos de tiempo.

2.2.1.3. Crono

Se denomina crono *(chronon)* al intervalo temporal de duración mínima y fija; su tamaño (granularidad) puede ser expresado, dependiendo de las necesidades del usuario, en segundos, minutos, horas, días, semanas, mes, año, etc. La granularidad, por lo tanto, es la duración de la menor unidad de tiempo almacenada en la **TDB**. Este concepto denota el grado de precisión en la representación de los registros temporales.

2.2.1.4. Marca de Tiempo

Una marca de tiempo (*timestamp*) es una marca temporal asociada a un objeto o atributo de la base de datos.

2.2.1.5. Tiempo de Vida

El tiempo de vida (*lifespan*) de un objeto de la base de datos es el tiempo en el cual éste está definido. En los objetos temporales que tienen un tiempo de vida asociado, éste puede ser representado por intervalos de tiempo no necesariamente contiguos.

2.3. Dimensiones Temporales

Los conceptos previamente expresados sirven de fundamento para definir un conjunto de registros temporales provistos por las **TDB**. No necesariamente todos ellos deben estar presentes en todos los modelos que pretendan preservar aspectos temporales.

La información temporal puede tener diferentes semánticas dependiendo de la dimensión temporal considerada. Una **TDB** usualmente define una o dos de las siguientes dimensiones temporales: tiempo válido (*valid time*) y tiempo de transacción (*transaction time*). Estas dos dimensiones constituyen, respectivamente, la historia de los objetos del mundo real y la historia de su registro en el **DBMS** y ambas son ortogonales, esto es, independientes entre sí.

La semántica asociada al tiempo válido y al tiempo de transacción es, en general, parte de la base de datos, no solo de una aplicación en particular.

2.3.1. Tiempo Válido

El tiempo válido se aplica a los hechos; un hecho denota cualquier declaración que pueda serle asignado un valor de verdad (verdadero o falso) [GJ98]. El tiempo válido de un hecho es aquel en el cual ese hecho es, fue o será cierto en el modelo de la realidad que se está representando; todos los hechos tienen un tiempo válido, pero éste puede o no estar registrado explícitamente en la base de datos. Un hecho puede tener asociado varios instantes o intervalos temporales. El tiempo válido usualmente lo suministra el usuario y puede ser modificado.

Consideremos, por ejemplo, el siguiente hecho: un empleado "E" trabaja para la compañía "C" desde el 1° de enero de 2009 hasta el 31 de diciembre del

mismo año; el tiempo válido asociado a ese hecho será, si usamos un modelo de datos discreto, el intervalo [01/01/2009, 31/12/09). En la práctica, el tiempo válido es el más importante ya que modela la veracidad de los hechos registrados en el universo de discurso, principal objetivo de los sistemas de información [BSW92].

2.3.2. Tiempo de Transacción

Un hecho se registra en una base de datos en un momento particular; luego de ser almacenado puede ser utilizado hasta que sea eliminado lógicamente. El tiempo de transacción de un hecho que es registrado en la base de datos es aquel en el cual ese hecho es actual en ella y puede ser recuperado. El tiempo de transacción podrá ser solamente modificado por el **DBMS**; así, cualquier cambio que el usuario realice sobre los datos, será expresado mediante un nuevo tiempo de transacción que reflejará el momento en que se hizo la actualización [GMJ98].

Siguiendo con el ejemplo anterior, si se registra en la base de datos el hecho de que un empleado "E" trabaja para la compañía "C", el día 3 de enero de 2009 y se elimina esa información el día 5 de enero de 2010, el tiempo de transacción del hecho se puede representar mediante el intervalo [03/01/2009, 05/01/2010).

Como en el caso del tiempo válido, el tiempo de transacción puede tener asociado varios instantes o intervalos de tiempo.

A diferencia del tiempo válido, el tiempo de transacción puede ser asociado con cualquier estructura almacenada en la base de datos, no sólo con hechos. De este modo, todas las estructuras almacenadas tienen aspectos relacionados con el tiempo de transacción. Esta dimensión temporal es utilizada solamente en meta reglas, por ejemplo, para conocer qué información el sistema ha usado en algún punto de su ejecución [BSW92].

El tiempo de transacción no tiene que coincidir, necesariamente, con el tiempo de validez, éste puede ser posterior al tiempo de transacción, lo que implica (y, por lo tanto, se denomina) actualización proactiva; anterior al tiempo de transacción, actualización retroactiva o, en el caso en que coincida el tiempo válido y el tiempo de transacción, actualización simultánea.

2.3.3. Tiempo Definido por el Usuario

El tiempo definido por el usuario (*user-defined time*) representa algún aspecto del tiempo que no es interpretado por el **DBMS** como un tipo especial de dato, por ejemplo, una fecha de cumpleaños. Los modelos de datos soportan tiempos definidos por el usuario mediante, por ejemplo, tipos *DATE, DATETIME,* etc., para valores de atributos; el tiempo definido por el usuario puede ser cualquier instante referido al pasado, presente o futuro, son suministrados por el usuario y pueden actualizarse.

La distinción entre dato temporal interpretado y no interpretado surge debido a que modelos de datos temporales propuestos utilizan estructuras de datos de modelos de datos no temporales y extienden los esquemas con atributos temporales. En el caso en que las consultas sean evaluadas usando semántica temporal, los atributos temporales son interpretados como tal. En cambio, los atributos definidos por el usuario, no son utilizados de esa forma.

2.3.4. Tiempo de Vida de una Entidad

El tiempo de vida (*lifespan*) de una entidad captura el tiempo de existencia de esa entidad. El tiempo de vida de una entidad "E" puede ser modelado como el tiempo válido del hecho "´E´ existe"; sin embargo, en muchas aplicaciones es importante considerar el tiempo de vida como un aspecto separado. El

concepto de tiempo de vida puede aplicarse, con el mismo significado, a las interrelaciones [GMJ98].

2.3.5. Tiempo de Decisión

En una **TDB** puede considerarse también el tiempo de decisión (*decision time*) de un hecho; se lo define como aquel en que el hecho fue decidido. Un hecho puede tener asociado varios tiempos de decisión. Debido a que la cantidad y el significado de *"el tiempo de decisión de"* varia de aplicación en aplicación y, además, a diferencia del tiempo válido y de transacción, aquel no exhibe propiedades especializadas, en general no son considerados en los modelos temporales [GJ98].

2.3.6. Tiempo Actual

Una variable especial denominada "ahora" (*now*) modela la intuitiva noción de tiempo actual en una base de datos [FM96]. *Now* es una palabra que en ingles significa "en este momento". Una variable con este nombre ha sido utilizada en modelos de datos temporales relacionales, primariamente como una marca de tiempo asociada a tuplas o valores de atributos en relaciones temporales. La noción de un valor de tiempo actual siempre creciente se ha expresado en varios modelos de **TDB** mediante variables tales como "*until-changed*", "@", "∞" y "-" [Cli+97].

2.4. Tipos de Bases de Datos Temporales

Los diferentes tipos de bases de datos están vinculados, en su definición, a los conceptos vertidos anteriormente. Una base de datos que modela solamente el tiempo válido se la denomina **HDB**, la que modela solamente el tiempo de transacción, ***RollBack Data Base* (RBDB)** y la que modela el tiempo válido y el de transacción, ***BiTemporal Data Base* (BTDB)** [Jen+94].

Podemos graficar los aspectos temporales mediante dos ejes ortogonales que representan el tiempo válido y el tiempo de transacción.

Las bases de datos no temporales capturan la realidad del universo de discurso que modelan y sus cambios en forma dinámica, pero solo registrando un estado: el actual. La modificación del estado de la base de datos se realiza mediante operaciones de inserción, borrado y modificación y los estados anteriores de la base de datos son perdidos. En la Figura 2.1 se muestran dos estados diferentes (e_1 y e_2) de la base de datos, que corresponden, respectivamente, a los instantes ¨ahora$_1$¨ y ¨ahora$_2$¨.

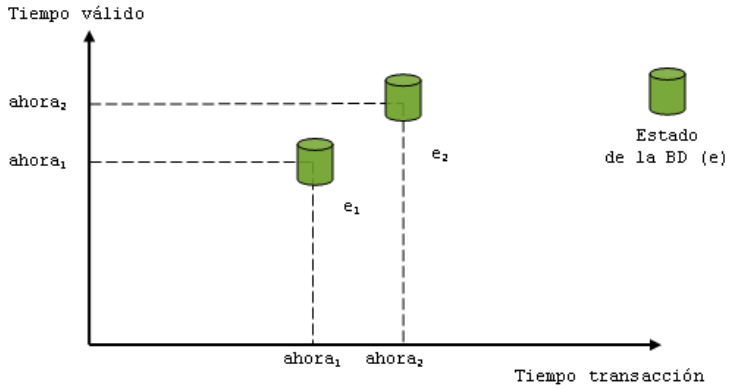


Figura 2.1. Base de Datos no-Temporal

Las **HDB** registran la historia de los datos que modelan al mundo real, las modificaciones se registran a lo largo de la dimensión tiempo válido, por lo tanto, el pasado, presente y el futuro pueden ser registrados. Como consecuencia de esto, los lenguajes de consulta son más complejos ya que precisan recuperar datos de diferentes estados de la base de datos. En la Figura 2.2 se muestran diferentes estados de la base de datos sobre la dimensión tiempo válido; se observa que los estados ev_1 y ev_2 corresponden a hechos pasados, el estado ev_3, al actual y el ev_4 a un estado futuro.

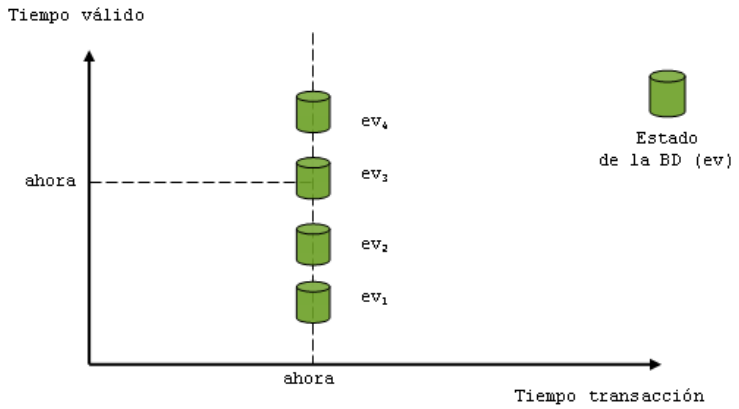


Figura 2.2.

Las **RBDB** registran los cambios en la base de datos, las modificaciones se registran a lo largo de la dimensión tiempo de transacción. A diferencia de las **HDB**, no registra estados futuros de la base de datos debido a que el sistema mantiene registros de tiempo de transacción y no conoce nada referido a hechos futuros. En la Figura 2.3 se muestran diferentes estados de la base de datos sobre la dimensión tiempo de transacción; se observa que los estados et_1 y et_2 corresponden a hechos pasados, el estado et_3, al actual y estado et_4 que representaría a hechos futuros no puede ser registrado.

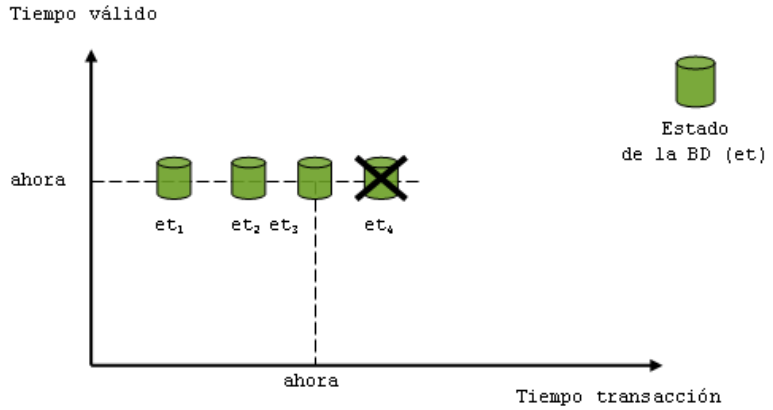


Figura 2.3. Base de Datos RollBack

Por último, Un **BTDB** es una combinación de una **HDB** y una **RBDB;** en este tipo de base de datos temporal, se preservan, para cada estado de la base de datos, tanto el tiempo válido como el tiempo de transacción. En la Figura 2.4 se muestran diferentes estados de la base de datos sobre las dimensiones tiempo válido y tiempo de transacción.

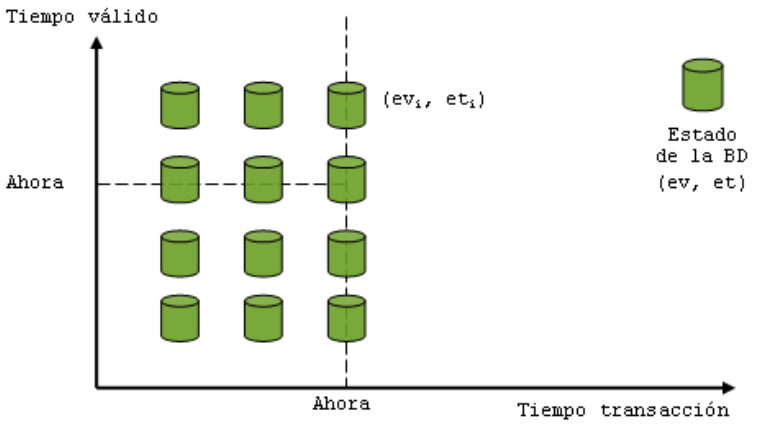


Figura 2.4. Base de Datos BiTemporal

2.5. Actualización en una Base de Datos Temporal

La administración de los datos en un **ODB** se efectúa mediante las operaciones de *insert, update, delete y select*, las restricciones de integridad son mantenidas por el **DBMS.** En las **TDB** se realizan las mismas operaciones, pero su implementación no es exactamente la misma; en las operaciones de actualización, los valores pasados no son solamente sustituidos por otros sino también son preservados para su posterior utilización, esto es, todos los valores son almacenados en la base de datos con marcas de tiempo asociadas. Por lo tanto, el borrado de datos se realiza de manera lógica. No obstante, el borrado físico puede ser efectuado eliminando la información que resulte irrelevante (o porque existen imposiciones legales que lo hacen necesario); esta operación se denomina *vacuuming* [HE00].

2.5.1. Esquemas de Base de Datos Temporal

De la misma forma que los datos en la base de datos (extensión) requieren consideraciones especiales para registrar hechos pasados, el esquema conceptual (intensión) puede sufrir modificaciones y, por lo tanto, precisar mecanismos para registrar su evolución. Con este objetivo surgieron las bases de datos temporales que consideran la evolución de esquemas en el tiempo; estas se diseñan para resolver, por un lado, el almacenamiento registrado en versiones sucesivas del esquema de la base de datos y sus datos asociados y, por otro, el procesamiento de consultas que involucran más de un esquema [ME99].

2.6. Modelo de Datos Temporales

Aunque los aspectos temporales en las bases de datos son importantes, para la mayoría de las aplicaciones éstos son difíciles de capturar, en forma elegante, mediante un modelo de datos. Las características temporales han sido modeladas, frecuentemente, por medio del ***Entity-Relationship*** **(ER)** [Che76] original, con el agregado de entidades, atributos e interrelaciones temporales y, debido a ellos, los diagramas se transformaban en complejos y difíciles de leer. Por tal razón muchos usuarios simplemente ignoraban los aspectos temporales en sus diagramas, agregando frases textuales indicando que algunas características temporales, de alguna manera, eran capturadas. Como resultado de esto, la transformación del modelo **ER** a las tablas relacionales de un **RDBMS**

subyacente (no la única, pero si las más habitual) debía realizarse "*a mano*" [GJ99]. El modelo, por consiguiente, no permitía una adecuada documentación de las extensiones temporales de los esquemas relacionales utilizados por los diseñadores. Como respuesta a estos problemas se han desarrollado extensiones al modelo **ER,** denominado ***Extended Entity-Relationship*** **(EER)** que incorporaron construcciones que permitieron al diseñador, en forma natural y elegante, capturar características temporales.

El modelo **ER** es fácil de comprender y brinda, además, una visión intuitiva de la base de datos. Desde su publicación, el modelo ha tenido variaciones tanto en su notación como en su semántica. Los diferentes aspectos temporales pueden o no, dependiendo de los requerimientos de la aplicación ser registrados en la base de datos; por lo tanto, debería ser una decisión del diseñador la elección de cuales entidades, atributos e interrelaciones serán o no temporales; incluso una interrelación o entidad temporal debería poder tener atributos temporales o no.

Un ***Temporal Entity-Relationship*** **(TER)** ideal debería ser fácil de entender en términos del modelo original, no debería invalidar los diagramas legados ni las aplicaciones sobre la base de datos y tampoco debería restringir a la base de datos para que sea solamente temporal, sino permitir combinar características temporales y no temporales [JS97a].

2.6.1. Soporte Temporal

Dos enfoques diferentes han sido ideados, en el modelo **ER,** para proveer soporte en la conceptualización de las características temporales. Uno, denominado implícito, oculta la dimensión temporal en la interpretación de las estructuras en las construcciones del modelo. De esta manera, el modelo no incluye ninguna nueva construcción temporal explicita con respecto al estándar. Cada construcción del modelo siempre es interpretada con semántica temporal, de esta forma las instancias de las entidades e interrelaciones son potencialmente variables en el tiempo; en este modelo, los atributos marcados temporalmente para registrar los aspectos temporales están ocultos, por lo tanto, no hay marcas temporales explicitas para registrar aspectos temporales; el problema de este enfoque es que excluye la posibilidad de diseñar bases de datos no temporales [GJ98].

El otro enfoque, denominado explícito, retiene la semántica no temporal del modelo convencional y adiciona nuevas construcciones sintácticas para la representación de las entidades e interrelaciones temporales y sus interdependencias. En este modelo, los atributos marcados temporalmente son representados explícitamente, conservando la semántica del modelo **ER** original.

El soporte temporal puede ser tanto explicito como implícito. El inconveniente de este enfoque está vinculado al aprendizaje y entendimiento, por parte del diseñador de la base de datos, si las extensiones para marcas temporales son excesivas. Por otro lado, la ventaja del enfoque explicito es la denominada *compatibilidad ascendente*; esto es, el significado de los modelos **ER** convencionales (legados) utilizados en los modelos temporales permanecerá inalterable. Esto es importante, por ejemplo, en el modelado de **DW** y bases de datos federadas, donde los datos fuente pueden ser tanto provenientes de **TDB** como de bases de datos legadas [AF99].

2.7. Implementación de Base de Datos Temporales

Un objetivo central en el diseño convencional de los **ODB** es producir una estructura de almacenamiento eficiente conformada por un conjunto de esquemas de relaciones. El uso de las formas normales, tanto la tercera forma

normal (3FN) como la forma normal de Boyce-Codd, constituyen buenas prácticas para lograr esquemas de relación "buenos". Por otro lado, los conceptos de normalización existentes no son aplicables a las **TDB**, debido a que éstas emplean estructuras relacionales diferentes de las usadas en las bases de datos convencionales [JS97b].

La implementación de un **TDB** puede realizarse en un **DBMS**. En [EHMD00] argumentan que, si bien se han presentado varios modelos de datos temporales, no existía, hasta ese momento, implementaciones comerciales disponibles y presentan, en su propuesta, un ambiente integrado de **TDB** implementado sobre un **DBMS**; en [HE00] proponen implementar un modelo de datos temporal utilizando un **RDBMS**, mapeando en ella los modelos de datos temporales; en [Tan04] se plantea un modelo de datos temporal y restricciones de integridad temporal en una extensión temporal sobre un ***Relational Model (RM)***. En [DW02] se presenta un enfoque completamente relacional para los problemas relacionados con los datos temporales. Siguiendo con esta línea, en [Zim06] se muestra cómo realizar consultas temporales usando el estándar **SQL**.

A pesar de la enorme aceptación de los **RDBMS**, éstos presentan algunas limitaciones en vistas a las necesidades de algunas de las aplicaciones actuales. La nueva generación de sistemas ***Object-Oriented Data Base Management System*** (**OODBMS**) que incluyen a las ***Object-Relational Data Base Management System*** (**ORDBMS**) proveen soporte para implementar tipos de datos complejos y relaciones, datos multimedia, herencia, etc. En [UD03] se detallan diversas estrategias para la transformación de un modelo de datos conceptual a tres modelos lógicos soportados por: **OODBMS**, **ORDBMS** y **RDBMS**; en [VVCM07] se presenta una transformación, en el contexto de **MDA** de un modelo de datos, expresados en ***Unified Modeling Language*** **(UML)** [UML], hacia un modelo lógico Objeto/Relacional, para el estándar SQL:2003. En [GCR06] compara y evalúa técnicas para transformar un modelo conceptual de datos usando **UML** a un modelo lógico soportado por un **ORDBMS**.

2.8. Trabajos Relacionados

Presentaremos, a continuación, diferentes propuestas sobre extensiones al modelo **ER,** para capturar aspectos temporales.

TERM [Klo81], fue el primer modelo temporal, con la característica principal de no tener una representación gráfica, sino una sintaxis similar al lenguaje de programación *PASCAL*; permitía diseñar aspectos temporales a través del concepto de *historia,* ésta estaba definida como una función desde un dominio temporal a algún dominio de valores que era utilizada para modelar aspectos vinculados con la variación temporal; este modelo considera solo el tiempo válido.

En *RAKE* [Fer85] se mantienen la mayoría de las construcciones y semántica del modelo **ER** original e introduce otras nuevas para modelar interrelaciones y entidades temporales, mediante entidades débiles que poseen un periodo de tiempo implícito que determina el periodo de validez de los elementos temporales; este modelo considera solo el tiempo válido.

MOTAR [Nar88] presenta características internas que permiten describir, tanto a nivel conceptual como lógico, los aspectos temporales; el modelo provee soporte implícito, en el modelo conceptual, de marcas de tiempo para registrar el tiempo válido, haciéndolos explicito en el modelo lógico, mediante atributos marcados temporalmente, a través de una transformación al modelo relacional; este modelo considera solo el tiempo válido.

TER [Tau91] está basado en el modelo **ER**, en particular, reemplaza el concepto de restricción de cardinalidad por el de restricción cardinalidad *instantánea* y *tiempo de vida*, mediante la cual redefine la clasificación de

interrelaciones y de opcionalidad. La noción de tiempo está implícita en el modelo, haciendo explícita a través del mapeo al modelo **ER**; este modelo considera solo el tiempo válido.

En *ERT* [TLW91], [MSW92] se utiliza como estructura básica el enfoque del modelo **ER** original; sobre esta base, el modelo extiende su semántica y su construcción gráfica en dos direcciones: el modelado del tiempo y el modelado de objetos complejos. Utiliza las marcas temporales para representar la variación temporal de objetos con respecto a otros, en lugar de modelar simplemente que alguna variación temporal ha ocurrido este modelo considera el tiempo válido y el tiempo de transacción

TEER [EWK93] no adiciona ninguna nueva construcción al modelo **ER**, sino que cambia la semántica de todas las construcciones del modelo original (entidades, interrelaciones clases/subclases y atributos) transformándolas en temporales; este modelo considera solo el tiempo válido.

STERR [EK93] es un modelo temporal semántico, distingue entre entidades, interrelaciones y atributos conceptuales y temporales, donde éstas últimas son tratados (a diferencias de aquellas) como de existencia permanente, la forma de representar el tiempo es similar a la planteada en *TEER;* este modelo considera solo el tiempo válido.

En *TempRT* [Kra96] se incorpora soporte para tiempo válido a través de interrelaciones, atributos y entidades temporales. La estructura temporal básica es la interrelación temporal, siendo su semántica una extensión de la planteada en el modelo **ER;** de este modo, tanto los atributos como las entidades temporales se definen a partir de interrelaciones temporales; este modelo considera solo el tiempo válido.

TERC+ [ZPSP97] incluye cuatro tipos de construcciones, cada uno de ellas con su homóloga temporal: tipos de atributos, entidades, interrelaciones e interrelaciones dinámicas. Utiliza para representar el tiempo, intervalos y elementos temporales e incorpora una notación que permite describir comportamiento Inter-objetos; modela el tiempo válido.

TIMEER [GJ98] tiene soporte temporal explicito, reteniendo las construcciones y la semántica del modelo **ER** e incorporando, además, soporte temporal para entidades, interrelaciones, clases/subclases y atributos. Define los tipos de entidades regulares donde el diseñador debe decidir si captura, o no, las características temporales en la base de datos. Tanto el tiempo válido como el tiempo de transacción son capturados para un tipo de entidad. Las entidades que capturan, al menos, un aspecto temporal se denominan tipo de entidad temporal, en caso contrario no temporal.

Por último, en [GJ99] se detallan un conjunto de modelos temporales y se examina cómo los conceptos y las construcciones de modelado utilizadas permiten capturan la información variante en el tiempo. Además, define un conjunto de propiedades consideradas convenientes en el diseño de un modelo **TER,** caracterizando cada uno de los modelos estudiados de acuerdo con ellas.

2.9. Resumen del Capítulo

El objetivo de este capítulo fue mostrar las características principales de las **TDB** y su diferencia con las **ODB**. Se presentaron los principales conceptos temporales y los diversos tipos de **TDB** a partir de las dimensiones temporales. Por último, se mostraron las características principales de los modelos de datos temporales y sus posibles implementaciones y se resumieron las diferentes propuestas sobre extensiones del modelo **ER** para capturar aspectos temporales.

Capítulo 3

Diseño de una Base de Datos Histórica

3.1. Introducción

En el diseño de un **TDB** se deben considerar, principalmente, dos aspectos; primero, definir cómo representar el tiempo en los modelos temporales y, luego, identificar diferentes dimensiones temporales respecto de las cuales los datos serán almacenados. La característica del modelo temporal simple propuesto (atributos explícitos para representar intervalos temporales) está vinculada a su ulterior uso en el método de diseño de un **HDW**; éste tomará datos de diversas fuentes, muchas de las cuales no serán **TDB**[2]**.** Utilizaremos para la creación del modelo temporal un enfoque explícito, con soporte temporal explicito; éste enfoque retiene la semántica no temporal del modelo convencional y adiciona nuevas construcciones sintácticas para la representación de las entidades, atributos e interrelaciones temporales. Respecto del segundo punto, consideramos solo la dimensión tiempo válido, por lo tanto la **TDB** será, en particular, una **HDB**.

En este capítulo se presentará el modelo de datos temporal propuesto que utilizaremos en el diseño de **HDW** (Capítulo 5). Luego, se describirán las transformaciones informales para obtener un modelo de datos temporal que permita preservar la historia de entidades, atributos e interrelaciones. Por último, se establecerán los criterios para su implementación mediante sentencias **SQL** en un **DBMS**.

3.2. Modelo Temporal

En el proceso de diseño consideraremos que el ***Temporal Schema*** (**TS**) se derivará de un ***Non-Temporal Schema*** (**NTS**) previo; el diseñador decidirá, sobre éste

[2] Esto implicará que en el proceso de extracción, transformación y carga de datos, se deberán considerar, entre otras cuestiones a resolver, el tratamiento de la inexistencia de los datos temporales y la semántica asociada a su ausencia.

último, qué entidades, atributos e interrelaciones considerará necesario transformarlas en temporales.

El **TS** resultante admitirá tanto entidades, atributos e interrelaciones temporales como no temporales. La consecuencia práctica de no especificar soporte temporal para algunas construcciones del modelo (entidades, atributos e interrelaciones) será que la supresión, tanto de una instancia de una entidad o interrelación o del valor de un atributo implicará un borrado lógico de la base de datos sin un registro ulterior de su historia.

Emplearemos, en la construcción del modelo temporal, el concepto de entidad compleja, que adaptaremos como atributo compuesto multivaluado, para una inicial representación de los atributos temporales; utilizáremos un intervalo temporal como rango de valores durante el cual el hecho es, fue o será cierto (tiempo válido), usaremos la notación [TI, TF) para representar un conjunto de instantes consecutivos y equidistantes, donde el atributo TI será el primer instante y el atributo TF el último instante del intervalo; el intervalo será *cerrado/abierto*, esto es, incluirá el instante TI y excluirá el instante TF; utilizaremos la palabra reservada "*now*" para designar el tiempo actual, el cual se lo considerará en continua expansión; adoptaremos el marcado de las interrelaciones temporales, de modo que expliciten visualmente su condición de tal.

La solución propuesta para registrar el intervalo temporal mediante dos atributos, TI y TF, tiene como principal ventaja la simplicidad, pero presenta algunos inconvenientes a la hora de implementarlo, principalmente, la introducción por parte del usuario de estos dos atributos por separado y la responsabilidad de evitar solapamientos temporales. La necesidad de evitar estos problemas generó el uso de intervalos temporales como tipo de dato primitivo [LM95].

3.2.1. Componentes del Modelo Conceptual Temporal

El diseño conceptual de sistemas de información utilizando el modelo **ER** es una técnica estándar aceptada en el diseño de software [GL03]; este modelo, que es una herramienta básica en el diseño de base de datos es, además, un eficiente método para capturar aspectos semánticas en diferentes ámbitos [Bad04].

Presentaremos, a continuación, las principales construcciones del modelo temporal, cuyos componentes básicos serán tomados del modelo **ER**; para la representación de entidades débiles y la multiplicidad máxima y mínima adoptaremos la gráfica y notación utilizada en [GMR98].

3.2.1.1. Entidad Temporal

Una entidad representa a un conjunto de instancias que comparten los mismos atributos y semántica. Éstas pueden ser regulares o débiles, las primeras tendrán existencia propia (identificador único interno), las segundas, que dependen de aquellas, precisarán para su identificación del atributo identificador de la entidad regular de la que dependen, más un atributo discriminante (cuyo valor no se repite en el contexto de la entidad débil).

Dado que el tiempo de vida de una entidad "E" puede ser modelado como el tiempo válido del hecho "´E´ existe" [GMJ98], éste enfoque será el tratamiento dado a las entidades temporales (regulares o débiles) para registrar su tiempo de vida.

- *Dada una entidad (regular o débil) E, consideramos que es una entidad temporal si nos interesa conservar su tiempo de vida (lifespan).*

En el modelo propuesto, toda entidad en el **NTS** que se precise temporal tendrá asociada, en el **TS**, una entidad temporal que tendrá el mismo nombre que la entidad en el **NTS** (será una entidad débil de ésta) y terminará en "-T"; el identificador de la entidad temporal estará formado por la unión del identificador de la entidad no temporal más el atributo denominado TI; además, tendrá un atributo descriptivo denominado TF. Los atributos TI y TF determinarán los instantes inicial y final, respectivamente, del rango dentro del cual la entidad E tendrá existencia en el universo de discurso.

Las propiedades temporales de las entidades débiles serán las mismas que las de las entidades regulares.

Por ejemplo, la entidad *CLIENTE* (Figura 3.1, izquierda) tendrá asociada una entidad temporal denominada *CLIENTE-T* (Figura 3.1, derecha), que modelará el tiempo de vida de aquella.

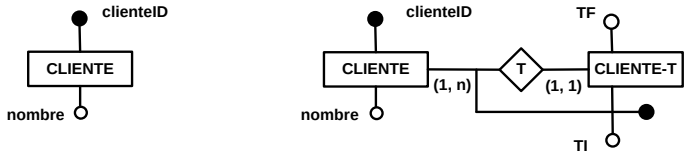


Figura 3.1. Representación de una Entidad Temporal

Observamos en la Tabla 3.1 distintos valores, tanto de la entidad *CLIENTE* como de *CLIENTE-T*. En la misma, se observa que el cliente Pérez estuvo activo entre la fecha 19/02/06 hasta la fecha 24/08/07; luego, estuvo inactivo desde la fecha 25/08/07 hasta la fecha 23/06/08, para luego retornar como cliente desde la fecha 24/06/08 hasta la fecha 30/12/08, después tuvo otro periodo de inactividad hasta la fecha 01/01/09, siendo actualmente, nuevamente, cliente activo.

CLIENTE

clienteID	Nombre
01	Pérez
02	Paz
03	López

CLIENTE-T

clienteID	TI	TF
01	19/02/06	25/08/07
01	24/06/08	31/12/08
01	02/02/09	now

Tabla 3.1. Representación Tabular de las Entidades *CLIENTE* y *CLIENTE-T*

3.2.1.2. Atributo Temporal

Un atributo es una propiedad que caracteriza tanto a entidades como a interrelaciones y que está asociado a un conjunto de valores que constituyen su dominio. Los atributos pueden ser descriptivos o identificadores, estos últimos solo se consideran para las entidades y tienen la propiedad de identificar unívocamente, dentro de la entidad, una instancia de otra.

Los atributos se pueden clasificar en simples o compuestos, un atributo compuesto es una agrupación de atributos simples. Los atributos pueden ser monovaluados o multivaluados, los primeros tomarán valores de un dominio de valores atómicos, los segundos pueden tomar más de un valor; también pueden ser obligatorios u optativos. Existen, por lo tanto, cuatro posibles combinaciones: atributos monovaluados obligatorios u optativos y atributos multivaluados obligatorios u optativos. Estas características se representan en el modelo, asociado a cada atributo, las multiplicidades mínima y máxima: (1, 1), (0, 1), (1, n), (0, n) respectivamente. El caso más común (1, 1), normalmente se omite.

Los atributos simples se representan mediante pequeños círculos vacíos unidos, mediante arcos, a la entidad y, cercano al círculo, los nombres asociados que lo describen; los atributos compuestos se representan mediante atributos simples ligados al compuesto; el identificador de la entidad se explicita del mismo modo que los atributos pero con el círculo relleno (Figura 3.2, centro)

- *Dado un atributo A, consideramos que es un atributo temporal si nos interesa conservar su historia (valid time).*

Un atributo compuesto multivaluado AT = {A, TI, TF}, será utilizado para representar a un atributo temporal de una entidad o interrelación. Los atributos denominados TI y TF determinarán los instantes iniciales y finales, respectivamente, del rango dentro del cual el valor A es cierto en el universo de discurso.

En el modelo propuesto, todo atributo en el **NTS** que varíe en el tiempo y que, además, resulte necesario conservar sus valores, será transformado, en su representación en el **TS**, en un atributo temporal.

La imposibilidad de representar atributos multivaluados en el modelo de implementación utilizado (**RM**), obliga a realizar una transformación en su representación.

Una entidad E, en el **TS,** con *m* atributos temporales (compuestos y multivaluados) puede representarse, en el modelo conceptual, mediante una entidad regular E con *m* entidades débiles vinculadas, la interrelación que las vincula estará marcada con una "T"; el nombre de la entidad temporal será igual al nombre del atributo temporal y terminará en "-T". El identificador de la entidad temporal estará formado por la unión del identificador de la entidad regular más el atributo denominado TI, además, tendrá un atributo descriptivo denominado TF.

Por ejemplo (Figura 3.2, izquierda), la entidad *PRODUCTO* tiene un atributo (*precio*) del cual se pretende conservar su historia; por lo tanto, se lo transformará en un atributo multivaluado compuesto (Figura 3.2, centro), donde el atributo precio representará el hecho y el TI y TF, los extremos del intervalo temporal. Luego, el atributo *precio* se transformará en una entidad débil *PRECIO-T* (Figura 3.2, derecha).

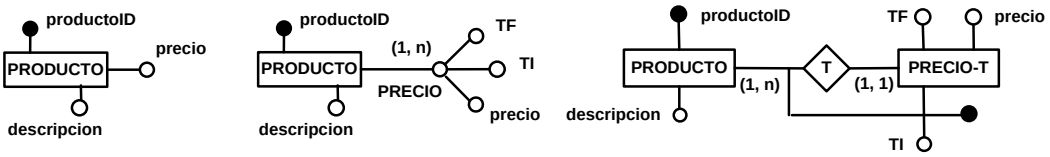


Figura 3.2. Transformación de un Atributo Temporal en Entidad Temporal

Cuando los atributos temporales describan a una interrelación, el tratamiento será el mismo, previa transformación de la interrelación en entidad.

En el caso particular en que todos los atributos de la entidad variasen sincrónicamente, esto es, compartan todos el mismo tiempo válido, podremos simplificar la estructura considerando que toda la entidad es temporal y transformarla como tal al **TS.**

En la Tabla 3.2, observamos posibles instancias de las entidades *PRODUCTO* Y *PRECIO-T*, en ésta última se detalla el tiempo válido de los diferentes precios que tuvo el producto a través del tiempo, por ejemplo, el precio del azúcar fue de $12, desde la fecha 19/02/06 hasta la fecha

24/08/07; luego, aumento a $15 entre la fecha 25/08/07 hasta la fecha 30/12/08, actualmente el precio es, desde la fecha 31/12/08 de $21.

PRODUCTO

ProductoID	Descripción
01	azúcar
02	pan
03	leche

PRECIO-T

ProductoID	TI	Precio	TF
01	19/02/06	12	25/08/07
01	25/08/07	15	31/12/08
01	31/12/08	21	now

Tabla 3.2. Representación Tabular de las Entidades *PRODUCTO* y *PRECIO-T*

3.2.1.3. Interrelación Temporal

Las interrelaciones representan vínculos entre entidades, sus instancias pueden interpretarse como tuplas que contienen elementos que pertenecen (y, además, referencian) a las instancias de las entidades involucradas. Una interrelación de grado ¨*n*¨ tendrá ¨*n*¨ entidades participantes. Se representará mediante un rombo con arcos que vinculan a las entidades que conecta. Nos limitaremos a interrelaciones binarias.

- *Dada una interrelación que vincula a dos entidades, consideramos que es una interrelación binaria temporal si el vínculo entre esas entidades varía en el tiempo y, además, amerita preservar su historia (valid time)*

En el modelo propuesto, si una interrelación en el **NTS** varía en el tiempo, ésta se denominará interrelación temporal y estará marcada con una "T", tendrá atributos descriptivos denominados TI y TF, además de los atributos (temporales o no) que tuviere la interrelación. Por otro lado, se deberá cumplir que la multiplicidad de rol máxima (E_1, T) = multiplicidad de rol máxima (E_2, T) = n, (n > 1). Los atributos TI y TF determinarán los instantes iniciales y finales, respectivamente, del rango dentro del cual la interrelación es cierta (tiempo válido) en el universo de discurso.

Por ejemplo, (Figura 3.3, superior) modelamos a las entidades *LOCALIDAD* y *CLIENTE* y la interrelación temporal que describe en qué momento (intervalo entre el TI y TF) un cliente estuvo ubicado en una localidad en particular.

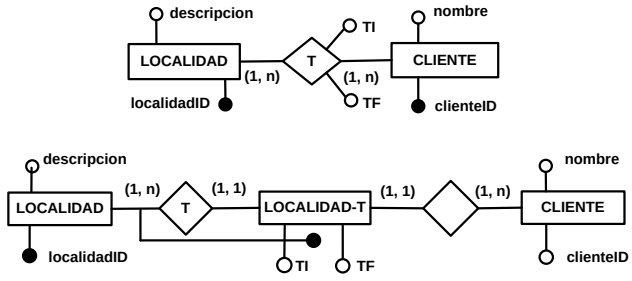


Figura 3.3. Transformación de una Interrelación Temporal en un Entidad Temporal

Dada una interrelación temporal binaria "T" entre entidades E_1 y E_2, transformaremos a la interrelación "T" en una entidad temporal, reemplazando cada rama E_1, E_2 con una interrelación binaria entre T y E_1 y T y E_2. El nombre de la entidad temporal será igual al nombre de la entidad E_1 y terminará en "-T"; los atributos descriptivos de la nueva entidad serán el atributo denominado TF, además de los atributos que tuviere la interrelación; el identificador de la

entidad temporal estará formado por la combinación del identificador de E_1 (del cual es entidad débil y cuyo vinculo estará marcado con "T") más el atributo TI.

Por ejemplo (Figura 3.3, inferior), la interrelación temporal se transforma en una entidad temporal, denominada *LOCALIDAD-T*, su identificador está formado por la unión del identificador de la entidad regular (*localidadID*) más el atributo TI, además, tendrá a TF como atributo descriptivo.

En la Tabla 3.3, observamos algunas posibles instancias de las entidades *LOCALIDAD*, *LOCALIDAD-T* Y *CLIENTE*; la interpretación es la siguiente: el cliente Pérez vivió en la localidad de Merlo desde la fecha 19/02/06, hasta la fecha 24/08/07; luego, se mudo a Padua, a partir de la fecha 25/08/07 hasta la fecha 30/12/08; por último, desde la fecha 31/12/08 hasta ahora, vive en Morón.

LOCALIDAD

LocalidadID	Descripción
01	Merlo
02	Padua
03	Morón

LOCALIDAD-T

LocalidadID	TI	ClienteID	TF
01	19/02/06	01	25/08/07
02	25/08/07	01	31/12/08
03	31/12/08	01	now

CLIENTE

clienteID	Nombre
01	Pérez
02	Paz
03	López

Tabla 3.3. Representación Tabular de las Entidades

3.2.2. Restricciones Temporales en el Modelo

Debido a que el modelo temporal admite combinaciones de entidades, atributos e interrelaciones temporales y no temporales, deberemos establecer en el mismo ciertas restricciones que evitarán inconsistencias en la **TDB.**

Por ejemplo:

- El tiempo válido de los atributos temporales deberá ser un subconjunto del tiempo de vida de la entidad temporal a la cual pertenece el atributo temporal.
- El tiempo válido de los atributos temporales deberá ser un subconjunto del tiempo válido de la interrelación temporal a la cual pertenece el atributo temporal.
- El tiempo de vida de las entidades temporales vinculadas a una interrelación temporal deberá ser un subconjunto del tiempo valido de la interrelación temporal.
- El tiempo de vida de la entidad débil deberá ser un subconjunto del tiempo de vida de la entidad regular de la cual depende

3.3. Transformaciones Descritas Informalmente

A continuación, describiremos en detalle las transformaciones informales del **NTS** al **TS**. El modelo resultante mantendrá las características del inicial y agregará, además, semántica temporal.

3.3.1. Transformación del Modelo de Datos al Modelo de Datos Temporal

En esta primera transformación consideraremos qué entidades, atributos e interrelaciones interesan ser preservadas en el tiempo, estos criterios serán utilizados, en el capítulo 5, para realizar el diseño del **HDW**. El modelo, como se detalló anteriormente, contemplará el tiempo de vida para las entidades y tiempo válido tanto para los atributos como para las interrelaciones.

3.3.1.1. Entidad temporal

Cada entidad identificada como temporal, en el **NTS** fuente, se representará en el **TS** destino, como una entidad (débil) temporal que estará vinculada a la entidad original (regular) que será preservada en el **TS**. La multiplicidad de rol de la entidad temporal será (1, 1), la multiplicidad de rol de la entidad regular será (1, N). El nombre de la nueva entidad será igual al de la entidad en el **NTS** y terminará en "-T". Tendrá como atributo descriptivo a TF; el identificador será la unión del identificador en el **NTS**, más el atributo TI.

> Por ejemplo, la entidad *CLIENTE* (Figura 3.4, izquierda), se mantendrá como tal en el modelo temporal y tendrá asociada a la entidad temporal *CLIENTE-T* (Figura 3.4, derecha).

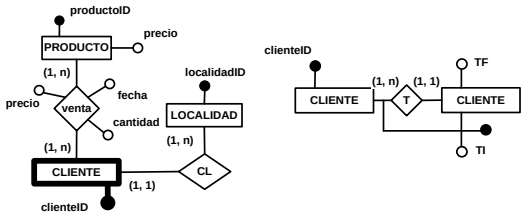


Figura 3.4. Representación de una Entidad Temporal

3.3.1.2. Atributo Temporal[3]

Cada atributo identificado como temporal, en el **NTS** fuente, se transformará en el **TS** destino, en una entidad (débil) temporal, estará vinculada a la entidad (regular) a la que pertenece dicho atributo; la multiplicidad de rol de la entidad temporal será (1, 1), la multiplicidad de rol de la entidad regular será (1, N). El nombre de la entidad temporal será igual al nombre del atributo temporal y terminará en "-T"; tendrá como atributos descriptivos a TF y un atributo que tendrá el mismo nombre y dominio que el atributo transformado en temporal en el modelo fuente; además, el identificador único de la entidad temporal será el atributo compuesto formado por TI más el identificador de la entidad regular. El atributo que originó la transformación desaparecerá en el **TS** destino.

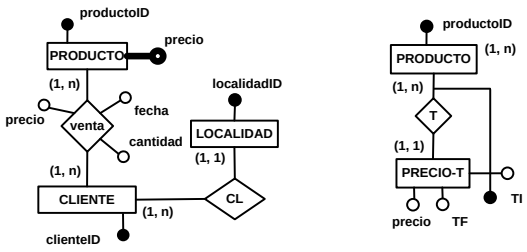


Figura 3.5. Representación de un Atributo Temporal

[3] Omitiremos, por simplicidad, el paso intermedio de transformación de la entidad temporal al un atributo multivaluado compuesto.

Por ejemplo, el atributo precio de la entidad *PRODUCTO* (Figura 3.5, izquierda), se transformará en entidad temporal *PRECIO-T* asociada a *PRODUCTO* (Figura 3.5, derecha).

3.3.1.3. Interrelación Temporal

Cada interrelación identificada como temporal, en el **NTS** fuente, se transformará, en el **TS** destino, como entidad (débil) temporal vinculada a una de las dos entidades que forman la interrelación (regular). El nombre de la nueva entidad se establecerá como combinación del nombre de la entidad regular elegida y terminará en "-T", las multiplicidad del rol de la entidad regular será (1, n), las multiplicidad del rol de la entidad temporal será (1, 1). Por otro lado, la entidad temporal estará vinculada a la otra entidad perteneciente a la interrelación en el **NTS,** la multiplicidad del rol de la entidad temporal será (1, 1), las multiplicidad del rol de la otra entidad será (1, n). La nueva entidad creada tendrá como atributo descriptivo a TF más los atributos que tuviere la interrelación en el modelo fuente. El identificador de la entidad transformada será el atributo compuesto formado por TI más el identificador de la entidad que le dio el nombre (representará una entidad débil de aquella).

Por ejemplo, la interrelación CL que vincula a las entidades *LOCALIDAD* y *CLIENTE* (Figura 3.6, izquierda), se transformará en entidad temporal *LOCALIDAD -T* asociada (como entidad débil) a la entidad *LOCALIDAD* y a la entidad *CLIENTE* (Figura 3.6, derecha).

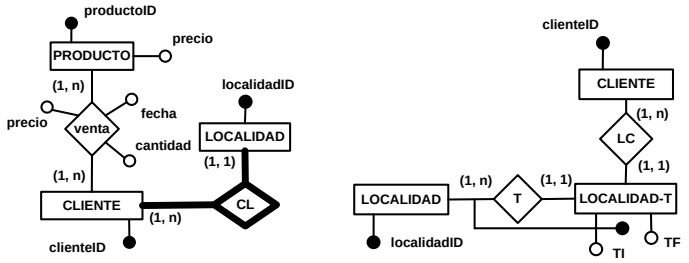


Figura 3.6. Representación de una Interrelación Temporal

En resumen, el modelo utiliza distintos tipos de entidades temporales: a) entidad temporal propiamente dicha, de la que se registra su tiempo de vida; b) entidad temporal que proviene de (y modela a) un atributo temporal y c) entidad temporal que proviene de (y modela a) una interrelación temporal. En estas dos últimas, el registro temporal corresponde al tiempo válido.

En la Figura 3.7 se muestra la transformación completa

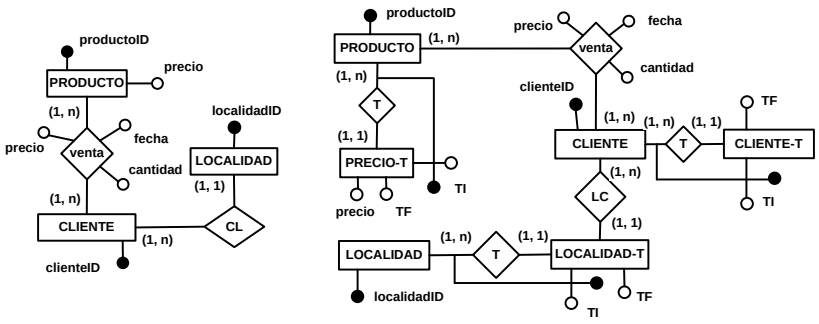


Figura 3.7. Transformación de un Modelo de Datos a un Modelo de Datos Temporal

3.3.1.4. Entidades e Interrelaciones

Todas las entidades e interrelaciones no temporales, en el **NTS**, se mantendrán sin modificaciones en el **TS.** Las jerarquías de clasificación (si las hubiere) serán consideradas como tales en el **TS**, sin comportamiento temporal.

3.3.1.5. Atributos

Todos los atributos (tanto de entidades cómo de interrelaciones) no temporales, en el **NTS**, se mantendrán en el **TS**; no así los atributos transformados en temporales.

3.3.2. Transformación del Modelo de Datos al Modelo Relacional

El modelo de datos temporal propuesto estará diseñado para complementar un modelo **MD**. La transformación se implementará usando el estándar **SQL**. La semántica temporal de los atributos utilizados para denotar intervalos temporales no será considerada como tal en el **DBMS**; la interpretación temporal será responsabilidad de la aplicación que los utilice.

3.3.2.1. Entidad Temporal (considerada como tal)

Toda entidad temporal (entidad débil) se transformará en una tabla; sus atributos serán columnas de la tabla, el identificador de la entidad (compuesto) será clave primaria de la tabla; el atributo identificador, clave primaria de la entidad regular, será clave foránea y hará referencia a ésta.

> Por ejemplo, (Figura 3.8, izquierda) las entidades *CLIENTE* y *CLIENTE-T*, se transformarán en las tablas *CLIENTE* y *CLIENTE-T*, respectivamente (Figura 3.9, derecha).

Figura 3.8. Transformación de Entidad Temporal en Tabla

3.3.2.2. Entidad Temporal (derivada de un atributo temporal)

La entidad temporal (entidad débil) se transformará en una tabla; sus atributos serán columnas de la tabla, el identificador de la entidad (compuesto y externo) será clave primaria de la tabla, el identificador de la entidad regular, que forma parte de la clave primaria, será clave foránea que hará referencia a la entidad regular.

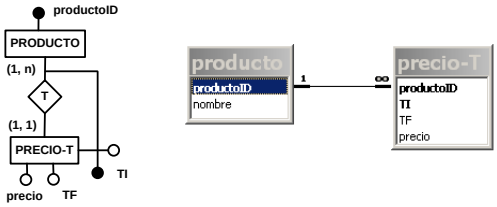


Figura 3.9. Transformación de Atributo Temporal en Tabla

Por ejemplo (Figura 3.9, izquierda) las entidades *PRODUCTO* y *PRECIO-T*, se transformarán en las tablas *PRODUCTO* y *PRECIO-T*, respectivamente (Figura 3.9, derecha).

3.3.2.3. Entidad Temporal (derivada de una interrelación temporal)

La entidad temporal (entidad débil) se transformará en una tabla; sus atributos serán columnas de la tabla, el identificador de la entidad (compuesto y externo) será la clave primaria de la tabla; el identificador de la entidad regular, que forma parte de la clave primaria, será clave foránea que hará referencia a una de las entidades regulares intervinientes, además tendrá un atributo clave foránea que tendrá el rol de clave primaria en la otra entidad vinculada y hará referencia a dicha entidad.

Por ejemplo (Figura 3.10, izquierda) las entidades *CLIENTE*, *LOCALIDAD* y *LOCALIDAD-T*, se transformarán en las tablas *CLIENTE*, *LOCALIDAD* y *LOCALIDAD-T*, respectivamente (Figura 3.10, derecha).

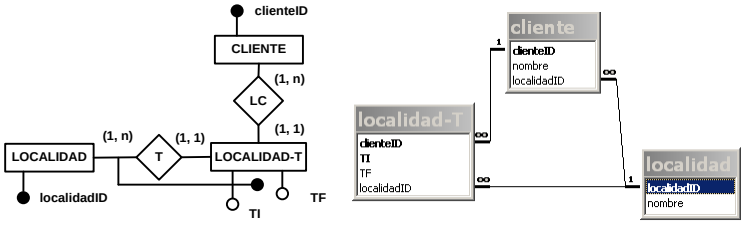


Figura 3.10. Transformación de Interrelación Temporal en Tabla

Para el resto de las transformaciones (entidades e interrelaciones no temporales) usaremos los criterios generales ya establecidos que detallaremos en el Anexo IV donde describimos las transformaciones formales completas.

3.3.3. Transformación del Modelo de Datos a Sentencias SQL

La transformación del modelo relacional al modelo de datos expresado en sentencias **SQL**, es inmediata. Cada atributo tiene un tipo de datos, que deberá ser transformado (mediante una tabla de transformación definida) a los tipos de datos **SQL** existentes.

La sintaxis general del sublenguaje ***Definition Data Languague*** **(DDL)** (simplificada) para crear tablas mediante sentencias **SQL** es:

```
CREATE TABLE table_name ( [
column_name data_type  [ NOT NULL | NULL] [, ... ]
PRIMARY KEY (column_name_pk [, ...] )
FOREIGN KEY (column_name_fk [, ...] ) REFERENCES reftable
] )
```

Utilizando los mismos criterios de transformación del apartado anterior, la transformación es directa.

Por ejemplo, dado las siguientes entidades e interrelaciones (Figura 3.8), las sentencias **SQL** que permitirán su implantación en un **RDBMS** será:

```
CREATE TABLE CLIENTE(
     clienteID text NOT NULL,
     PRIMARY KEY (clienteID)
     )
```

```
CREATE TABLE CLIENTE-T
     clienteID text NOT NULL,
     TI date        NOT NULL,
     TF date        NOT NULL,
     PRIMARY KEY (clienteID, TI),
     FOREIGN KEY (clienteID) REFERENCES CLIENTE
     )
```

3.4. Alcance y Limitaciones del Modelo

El modelo propuesto permitirá registrar el tiempo de vida (*lifespan*) de las entidades y el tiempo válido (*valid time*) tanto de atributos como de interrelaciones. No contemplará tiempo de transacción (*transaction time*) ni jerarquías de clasificación temporales.

Respecto de la consultas que podrán realizarse sobre su estructura, las resumimos a continuación[4]: para las entidades temporales, se podrá obtener su tiempo de existencia (tiempo de vida); para los atributos temporales, la determinación de en qué fechas se modifico su valor, además de determinar sus diferentes valores en diversos momentos (tiempo válido) y, por último, para las interrelaciones temporales, se podrá establecer en qué fechas se modificó el vinculo entre dos entidades y, para un instante particular, cual fue el vínculo entre ellas (tiempo válido). Estas características permitirán establecer patrones de consultas, que podrán automatizarse conjuntamente con las consultas **DSS** (capítulo 7).

3.5. Resumen del Capítulo

El objetivo de este capítulo fue describir el modelo de datos temporal simple propuesto, que será utilizado en el diseño de **HDW**. Se detallaron los principales componentes del modelo y se describieron las transformaciones informales para obtener, a partir de un modelo de datos, un modelo de datos temporal que admite preservar la historia de entidades, atributos e interrelaciones. Luego, se establecieron los criterios para su implementación mediante sentencias **SQL** en un **DBMS.** Por último se detallaron los alcances y limitaciones del modelo.

[4] Nos referimos al conjunto de consultas que serán automatizadas, otro tipo de consultas podrán realizarse también sobre el modelo

Capítulo 4

Data Warehouse

4.1. Introducción

Los **ODB´s** surgieron ante la necesidad de almacenar y organizar los datos de las organizaciones, producto de sus transacciones diarias. En este tipo de almacenamiento, las estructuras de datos se diseñan con el objetivo de evitar redundancias en los datos almacenados y, por consiguiente, posibles inconsistencias en la información obtenida, debido a los procesos de actualización.

En los sistemas ***OnLine Transaction Processing*** (**OLTP)**, cuya estructura de datos subyacente es un **ODB**, las operaciones automatizan el procesamiento de datos (por ejemplo, las operaciones bancarias, las ventas en un supermercado, etc.); estas actividades son rutinarias y repetitivas y, generalmente, consisten en operaciones aisladas, atómicas y breves; en este ambiente, las transacciones requieren actualizaciones o lecturas de unos pocos registros accedidos normalmente (en los **RDBMS**) mediante su clave primaria; la consistencia y la recuperación son elementos críticos en la base de datos, tanto como la maximización de las transacciones; por tales motivos, éstas se diseñan para reflejar la semántica de las operaciones de las aplicaciones conocidas y, en particular, para minimizar los conflictos de concurrencia. Las consultas que se realizan sobre las estructuras de almacenamiento son relativamente simples y, generalmente, están predefinidas y testeadas con el objetivo de obtener la salida adecuada a las necesidades de la organización.

Si, por el contrario, el objetivo de la base de datos fuera proveer vistas resumidas de las transacciones actuales y pasadas, si las consultas realizadas sobre los datos almacenados no fueran estructuradas y estuvieran, además, definidas en función de resultados de consultas previas y, si también se requirieran administrar grandes volúmenes de datos, las estructuras de datos previstas en los sistemas **OLTP** no serían las más adecuadas para estos fines; por lo tanto, se requerirá otro tipo de estructuras de almacenamiento.

En este capítulo se presentarán las características principales de un **DW** y sus diferencias con los **ODB**. Se detallarán los distintos tipos de jerarquías, las diferentes formas de visualizar los datos en la estructura **MD** y las alternativas de implantación. Se contrastarán las diferencias entre el **DW** con el **TDW** y **HDW** y, por

último, se describirán los más importantes trabajos donde se vincula al modelo **ER** con el diseño de un **DW.**

4.2. Data Warehouse

Actualmente, las empresas[5] utilizan los datos operacionales acumulados durante años y almacenados en estructuras *ad hoc* denominadas **DW** para ayudar a comprender y dirigir sus actividades. El procesamiento analítico que incluye consultas muy complejas (frecuentemente con funciones de agregado) y poco o nada de actualizaciones (denominado **DSS**) es uno de los usos primarios del **DW** [CD97]. La importancia del **DW** en el segmento comercial surge debido a la necesidad de reunir todos los datos en un sólo lugar para realizar, en detalle, el análisis de los mismos.

Los usuarios de los **DSS** están más interesados en identificar tendencias que en buscar algún registro individual en forma aislada [HRU96]. Con ese propósito, los datos de las diferentes transacciones se guardan y consolidan en una base de datos central denominada **DW**; los analistas lo utilizan para extraer información de sus negocios que les permita tomar mejores decisiones [GHRU97]. Mientras que un **ODB** mantiene datos actuales, el **DW** mantiene datos históricos de la empresa; como resultado, las estructuras de datos subyacentes, por crecer constantemente en el tiempo, requiere una alta capacidad de almacenamiento.

Las operaciones en los sistemas **OLTP** automatizan el procesamiento de datos, estas actividades son rutinarias y repetitivas. El **DW** está pensado para el **DSS**; los datos históricos, resumidos y consolidados son más importantes que los detalles y los registros individuales. Debido a que el **DW** tiene datos consolidados, quizás de varios **ODB**, recolectados en largos periodos, éstos tienden a ser, en orden de magnitud, más grandes que las **ODB**. La actividad principal sobre un **DW** es la realización de consultas complejas que pueden acceder a millones de registros, realizar búsquedas, uniones y agrupamientos. Aquí las consultas y el tiempo de respuesta son más importantes que las transacciones individuales.

Generalmente, el **DW** se mantiene separado del **ODB** de la organización, principalmente porque los requerimientos de rendimiento y la funcionalidad de los procesos **OLAP** difieren de las aplicaciones **OLTP**, características de los **ODB** [VS99].

Codd [CCS93] presentó el término ***OnLine Analitycal Processing* (OLAP)**, en el año 1993, para caracterizar los requerimientos de resumen, consolidación, visión y síntesis de datos a través de múltiples dimensiones. Este nombre se estableció para distinguirlo del tradicional **OLTP**, cuyo objetivo principal era administrar los datos de las transacciones u operaciones rutinarias de las organizaciones.

Una definición muy extendida de Inmon [Inm02], establece que un **DW** es "*una colección de datos **no volátiles**, **integrados**, que se acumulan en el **tiempo**, que están orientados a un **tema** determinado y que es utilizada para ayudar a tomar **decisiones** organizacionales*". Explicaremos en detalle esta definición:

- La ***no volatilidad*** de los datos se refiere a que, a diferencia de un **ODB**, donde son frecuentes las operaciones de altas, bajas y modificaciones sobre los datos (de hecho, las estructuras de datos subyacentes están diseñadas para realizar eficientemente esas acciones), en el **DW**, los datos son almacenados sin que se realicen (normalmente) posteriores modificaciones y bajas, hecho que

[5] Usaremos el término "empresa" en sentido general, no necesariamente de índole comercial.

también influye en la estructura de organización de sus datos.

- La ***integración de datos*** expresa que éstos son extraídos de diferentes **ODB** u otras fuentes de datos e integrados en un almacenamiento común (al menos, lógicamente). En este proceso podrían surgir ciertos problemas, debido entre otros, a los diferentes formatos de datos, a las diferentes formas de codificación, a los datos homónimos (con igual nombre y diferente significado), a los datos sinónimos (con distinto nombre e igual significado), a la presencia de valores nulos, etc.
- El concepto de ***acumulación temporal*** está relacionado con la particularidad de que, en el **DW**, siempre está incluida la noción de tiempo; esto permite tener diversos valores de un mismo objeto conforme pase el tiempo. Uno de los objetivos de nuestra tesis es hacer explícita, en la estructura de almacenamiento, esta característica temporal.
- La ***orientación a un tema determinado*** determina que los datos están organizados por temas (a diferencia de los **ODB´s**, en donde los datos están organizados para soportar aplicaciones particulares), cuyo objetivo es satisfacer las necesidades de información para la toma de decisión sobre aspectos determinados de la organización.
- Por último, las estructuras de almacenamiento están organizadas para el análisis de datos, tanto a nivel estratégico como táctico. Los datos están organizados para comprender y administrar la empresa, esto es, para **ayudar a tomar decisiones organizacionales.**

Por otro lado, más concisamente, Kimball [Kim96] definió al **DW** como "*una copia de los datos de transacción específicamente estructurados para consultas y análisis*". Propuso, además, que los datos en el **DW** deben ser consistentes, y que éste no solamente incluirá datos, sino también herramientas para consulta, análisis y presentación de los datos que se transformarán en información para el usuario.

4.3. Proceso de Construcción de un Data Warehouse

Para comprender mejor qué es un sistema **DW**, es interesante considerar los tres procesos generales que intervienen en su construcción y uso: extracción, transformación y carga (***Extract, Transform and Load* (ETL)**) y, por último, explotación. A continuación se describen, sintéticamente, cada uno de ellos [VSS02], [Vas+05]:

4.3.1. Extracción

La extracción consiste en la obtención de datos de las distintas fuentes operacionales tanto internas como externas a la organización. El principal problema en este proceso, reside en poder acceder a la información que se desea tener almacenada en el **DW**.

4.3.2. Transformación y Carga

El proceso de transformación lo componen una serie de tareas: limpieza, integración, agregación. Los dos problemas más importantes en la integración son la integración de formato y la integración semántica.

- *Integración de formato*: Se refiere a la unificación de tipos de datos, unidades de medida, codificaciones, etc. Una situación normal en estos entornos es que cada sistema operacional haya sido desarrollado independiente de los otros, dándose situaciones de inconsistencia en el formato y representación de los mismos datos.
- *Integración semántica*: La integración semántica se refiere a la integración de los datos de acuerdo a su significado. Debido a que la información de un **DW** proviene de diferentes sistemas operacionales diseñados con fines distintos, pueden darse situaciones en las que datos similares tengan significado distinto, o al revés, que datos distintos tengan el mismo significado, pudiéndose plantear problemas en el momento de integrarlos en el **DW**.

4.3.3. Explotación

La explotación consiste en la consulta y análisis de los datos en el **DW**. Desde el punto de vista del usuario, el único proceso visible es el de la explotación del **DW**, aunque la calidad del **DW** y su éxito radican en los dos procesos anteriores, que durante el desarrollo del **DW**, consumen la mayor parte de los recursos.

4.4. Modelo Multidimensional

El modelo **MD** es el fundamento del **DW**; en él los datos se estructuran mediante hechos y dimensiones. Los hechos representan el foco de análisis en el proceso de toma de decisión (por ejemplo, las ventas en una empresa) y contienen, generalmente[6], medidas que representan los elementos específicos de análisis (por ejemplo, la cantidad de productos vendidos). Las dimensiones (por ejemplo, fecha de venta, producto vendido y cliente) permiten explorar las medidas desde diferentes perspectivas de análisis (por ejemplo, la cantidad de ventas realizadas por producto, por cliente y por fecha). No existe una manera formal de decidir cuáles atributos serán dimensiones y cuáles corresponderán a medidas. Esto suele hacerse basándose en los procesos del negocio y, si bien, existen lineamientos generales que asisten en el proceso de construcción, es decisión del diseñador del **DW.**

4.4.1. Dimensiones y Jerarquías

Las dimensiones, usualmente, se asocian con sus jerarquías para describir niveles de agrupamiento específicos (explícitos o implícitos) y, por lo tanto, las diferentes granularidades (nivel de detalle) en la visión de los datos. Las jerarquías forman distintos niveles, relacionados entre sí, y son utilizados para realizar operaciones de agrupamiento.

Las jerarquías se pueden definir como relaciones binarias entre niveles de una dimensión [MZ04]; dado dos niveles consecutivos, al de más alto nivel (menor granularidad) se lo denomina "*padre*" y al de más bajo nivel (mayor granularidad) se lo denomina "*hijo*", el nivel que no tiene hijos se lo denomina "*hoja*" y el que no tiene padre, "*raíz*" (Figura 4.1).

Una relación en la jerarquía entre niveles *padre* e *hijo* puede pensarse como una interrelación en el modelo **ER**; si la multiplicidad de rol *padre,* en la interrelación entre *padre* e *hijo,* es (0,1) o (1,1)[7], corresponderá a una relación "x-

[6] Un hecho puede carecer de medidas; solamente registra la ocurrencia del hecho.

[7] El par (min, max) determina a la multiplicidad mínima y máxima del rol que le corresponde al nivel en la relación.

a-uno" (Figura 4.2, izquierda). Si la multiplicidad de rol *padre,* en la interrelación entre *padre* e *hijo,* es (0, N) o (1, N), corresponde a una relación "*x-a-muchos"* (Figura 4.2, derecha).

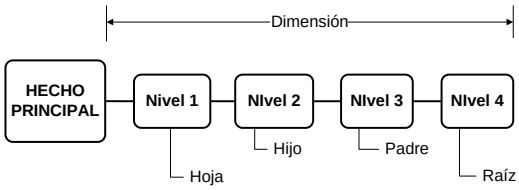


Figura 4.1. Niveles de Jerarquía

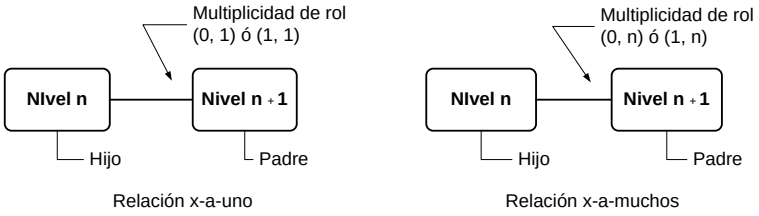


Figura 4.2. Relaciones entre Niveles de la Jerarquía

Los niveles de la jerarquía también pueden tener atributos descriptivos, denominados atributos no-dimensión, éstos no son utilizados para formar niveles de jerarquía, sino para describir detalles en los mismos, por ejemplo, el número de teléfono, la dirección de correo electrónico, etc.

4.4.2. Dimensiones "*Degeneradas*"

Existen casos donde a ciertas dimensiones no se las consideran en forma explicita debido a que la mayoría de sus propiedades están representadas mediante otros elementos (hechos o dimensiones); no obstante, se considera que es necesario algún atributo en el *hecho* que permita identificar unívocamente instancias del mismo. A este tipo de dimensiones se las denominan "*degeneradas"* (degenerated dimension). Por lo tanto, una dimensión *degenera* es aquella cuyo identificador existe solo en el hecho, pero no está materializada como una dimensión [MT02].

4.5. Clasificación de Jerarquías

Las jerarquías se pueden clasificar en diversos tipos [MZ05]: ***simples***, que, a su vez, pueden ser ***simétricas*** o ***asimétricas***, además de ***estrictas*** o ***no-estrictas*** y ***múltiples***, las que están compuestas por una o más jerarquías simples con un mismo criterio de análisis (Figura 4.3).

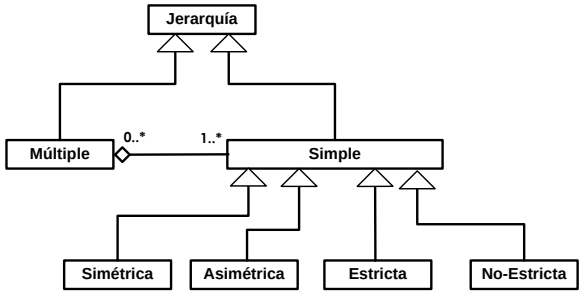


Figura 4.3. Clasificación de Jerarquías

4.5.1. Jerarquías Simétricas

Las jerarquías simétricas se caracterizan porque todas las instancias de un nivel *padre* deben vincularse, al menos, con una instancia de nivel *hijo* y éste no puede pertenecer a más un *padre*; esto implica que la multiplicidad del rol *hijo* debe ser (1, N) y la multiplicidad en el rol padre (1, 1).

Por ejemplo, la jerarquía "*producto → tipo de producto → departamento*" puede considerarse una jerarquía simétrica si exigimos que todos los productos pertenezcan, si o si, a un tipo de producto y éstos a no más que a un departamento. Observamos un ejemplo de jerarquía simétrica (Figura 4.4, superior) y posibles instancias de la misma (Figura 4.4, inferior).

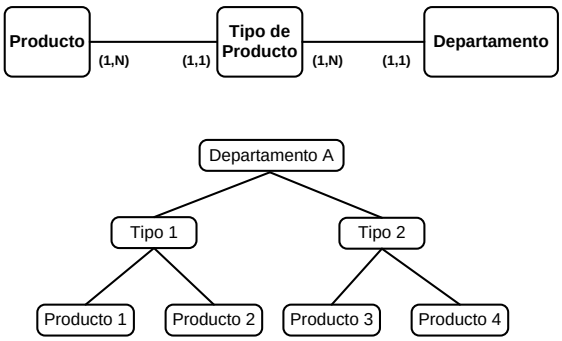


Figura 4.4. Jerarquías Simétricas

4.5.2. Jerarquías Asimétricas

Las jerarquías asimétricas se caracterizan porque no todas las instancias de un nivel *padre* deben vincularse, necesariamente, a una instancia de nivel *hijo*; de todos modos, como en las jerarquías simétricas, éste no puede pertenecer a más un nivel *padre*; esto implica que la multiplicidad del rol *hijo* debe ser (0, N) y la multiplicidad en el rol padre (1, 1).

Por ejemplo, la jerarquía "*producto → tipo de producto → departamento → región*" puede considerarse como una jerarquía asimétrica si no exigimos que algunos productos pertenezcan a un tipo de producto determinado y que, tampoco, algunos tipos de productos pertenezcan a un departamento, etc. Observamos un ejemplo de jerarquía asimétrica (Figura 4.5, superior) y posibles instancias de la misma (Figura 4.4, inferior).

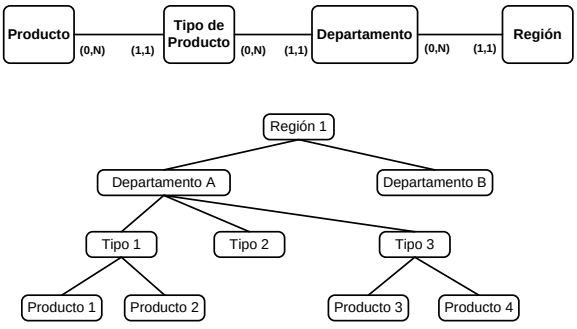


Figura 4.5. Jerarquías Asimétricas

4.5.3. Jerarquías Estrictas

Las jerarquías estrictas son aquellas en donde las instancias de un nivel *hijo* están vinculadas con no más que una instancia del nivel *padre*; esto implica que la multiplicidad del rol *padre* debe ser (0, 1) o (1, 1) y constituyen relaciones del tipo "*x-a-uno*". Este es el caso más común en la relaciones entre niveles y es la que permite aplicar sobre ella funciones de agrupamiento.

> Por ejemplo, la jerarquía "*producto → tipo de producto → departamento*" (Figura 4.4, superior) *puede considerarse una jerarquía estricta si consideramos que todos los productos pertenecen a un solo tipo de producto y cada tipo de productos a un solo departamento.*

4.5.4. Jerarquías No-Estrictas

Las jerarquías no-estrictas son aquellas en donde las instancias de un nivel *hijo* pueden tienen más de una instancia del nivel padre; esto implica que la multiplicidad del rol *padre* debe ser (0, N) o (1, N) constituyen relaciones del tipo "*x-a-muchos*".

> Por ejemplo, la jerarquía "*producto → tipo de producto → departamento → región*" *puede considerarse una jerarquía no-estricta si un tipo de producto pertenece a más de un departamento.* Observamos un ejemplo de jerarquía No-Estrictas (Figura 4.6, superior) y posibles instancias de la misma (Figura 4.6, inferior).

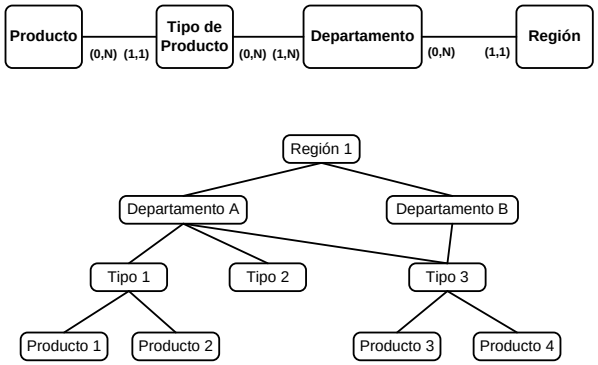


Figura 4.6. Jerarquías No-Estrictas

4.5.5. Jerarquías Múltiples

Las jerarquías múltiples son aquellas en donde existen varias jerarquías simples que comparten algún nivel. Por ejemplo, la dimensión tiempo puede tener una jerarquía formada por *fecha → mes → trimestre → año*; además de otra jerarquía formada por *fecha → semana → año*. (Figura 4.6)

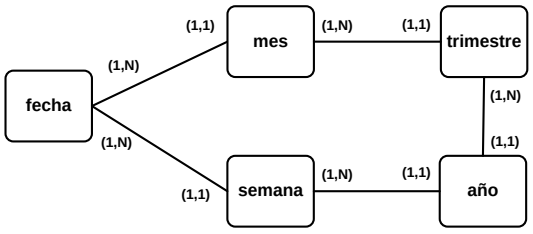


Figura 4.7. Jerarquías Múltiples

4.5.6. Relaciones Entre Hechos y Dimensiones

De la misma forma que existen vínculos entre los distintos niveles de las jerarquías en las dimensiones, también encontramos distintos tipos de relaciones entre el hecho y las dimensiones.

El *hecho* está vinculado directamente a las dimensiones (las denominadas *hojas* en el nivel de jerarquía); una relación entre el hecho y el mayor nivel de granularidad (hoja) puede pensarse como una interrelación en el modelo **ER**; normalmente, la multiplicidad del rol *hecho,* en la interrelación entre *hecho* y *hoja,* es (1, N), y la multiplicidad del rol hoja es (0, 1) o (1,1) y corresponde a una relación del tipo "*muchos-a-uno*". Este es el tipo de relación más frecuente. Si, por el contrario, la multiplicidad del rol hoja, en la interrelación entre *hecho* y *hoja*, es (0, N) o (1, N), corresponderá a una relación del tipo "*muchos-a-muchos*" (Figura 4.8). Este tipo de relaciones tienen el inconveniente de disminuir la simplicidad en la estructura del esquema de almacenamiento (en particular si la implementación es relacional), incrementar la complejidad en las consultas y degradar el rendimiento por el agregado de joins.

En [SRME01] se analizan las relaciones *muchos-a-muchos* entre un hecho y dimensiones y se ilustran mediante un ejemplo, mostrando ventajas y desventajas, seis diferentes enfoques que permiten mejorar el rendimiento de las consultas.

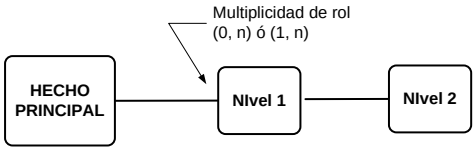


Figura 4.8. Relaciones entre Hechos y Dimensiones

4.6. Visión Multidimensional de los Datos

Los datos en el **DW** pueden ser representados de diversas maneras. Utilizaremos para ejemplificar las diferentes representaciones, una estructura de almacenamiento que contiene datos sobre ventas de productos, con clientes que los adquieren y las fechas de venta de esos productos.

Observamos, en todos los casos, que las ventas (hecho) están determinadas funcionalmente por los demás atributos (dimensiones); esto significa que, en el ejemplo planteado, dados los valores de los atributos producto, cliente y fecha, se puede determinar la cantidad de ventas realizada. En forma intuitiva, observamos que cualquiera de los otros tres atributos (producto, cliente y fecha) puede variar y, de acuerdo a ello, variar también el valor de la cantidad venida.

Detallaremos a continuación diferentes formas de representar esos mismos datos.

4.6.1. Visión Bidimensional

Mostramos (Tabla 4.1) una representación tabular (Bidimensional) de posibles datos de la estructura descrita anteriormente. En ella, las dimensiones (Producto, cliente y fecha) y la medida (cantidad), se representan como cabeceras de la tabla; las tres primeras columna de esa tabla representan las coordenadas específicas (valores de las dimensiones) y. la última, el valor asignado a esas coordenadas (la medida).

VENTAS

Producto	Cliente	Fecha	Cantidad
Azúcar	Fernández	19-02-2008	20
Azúcar	Rodríguez	19-02-2008	40
Arroz	Fernández	20-02-2008	30
Leche	Pérez	20-02-2008	10

Tabla 4.1. Visión Bidimensional de los Datos

4.6.2. Visión Multidimensional

Los mismos datos pueden ser descritos multidimensionalmente[8]. Esta representación, también denominada genéricamente cubo, es una vista **MD** de la base de datos en donde el valor crítico de análisis (las ventas) está organizado mediante un conjunto de dimensiones (producto, fecha y cliente).

> Por ejemplo, la Figura 4.9 muestra al cliente Fernández, al que se le vendió 20 (kilos) de Azúcar, el 19 de febrero de 2008. La cantidad vendida, 20 kilos (la medida) puede interpretarse como un punto en el espacio tridimensional, que tiene como coordenadas las dimensiones {Fernández, Azúcar, 19 de febrero de 2008}.

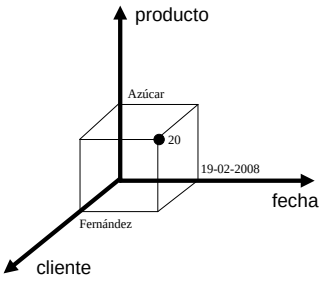


Figura 4.9. Visión Multidimensional de los Datos

4.6.3. Visión Esquemática

En la Figura 4.10 presentamos otra forma de exponer más detalladamente los datos, mediante un *esquema de hecho* [GMR98a].

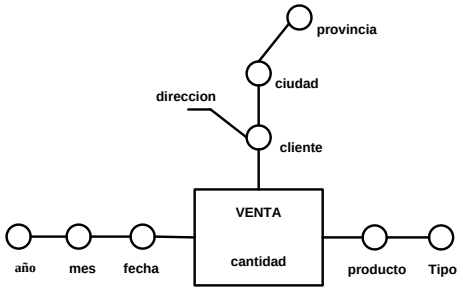


Figura 4.10. Esquema de Hecho

En este tipo de representación, el hecho de análisis se muestra mediante un rectángulo, que incluye la medida (cantidad), mientras que las dimensiones se representan mediante nodos unidos al hecho (producto, cliente y fecha), los atributos de las dimensiones se muestran también mediante nodos unidos a las

[8] Esta forma de representación gráfica admite, como máximo, tres dimensiones.

dimensiones que constituyen jerarquías de esa dimensión (ciudad y provincia en cliente) o, mediante líneas que representan atributos que describen tanto a dimensiones o jerarquías (dirección en cliente).

4.6.4. Visión "Relacional"

Otra forma de presentar la mismos datos **MD** (en el **RM**) es mediante el esquema "estrella" (*star schema*) (Figura 4.11) y su versión extendida "copo de nieve" (*snowflake schema*) [Kim96] (Figura 4.12). En el primero, la estructura **MD** se muestra mediante una tabla de hecho que representa el tema de análisis (por ejemplo, las ventas) y un conjunto de tablas dimensión (por ejemplo, la fecha, el producto y el cliente) que determinan la granularidad en la determinación del hecho.

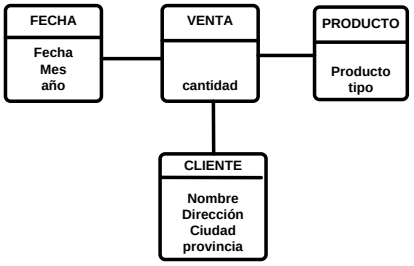


Figura 4.11. Esquema Estrella

En el esquema estrella, las jerarquías están implícitas como atributos de las dimensiones. En el esquema copo de nieve, las jerarquías se representan en forma explícita mediante tablas asociadas a las dimensiones; por ejemplo, la dimensión temporal está representada explícitamente mediante los niveles de jerarquía fecha, mes y año (Figura 4.12).

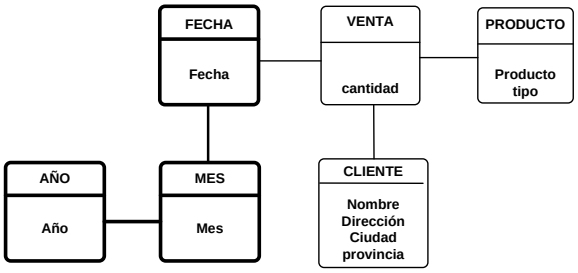


Figura 4.12. Esquema Copo de Nieve

4.7. Arquitectura General de un Sistema de Data Warehouse

La arquitectura típica de un **DW** [CD97] (Figura 4.13) incluye herramientas para la extracción de datos de múltiples **ODB´s** y otras fuentes externas; herramientas para "limpieza", transformación e integración de esos datos; para la carga y el refresco periódico del **DW** que refleje las actualizaciones de las fuentes y la purga del almacenamiento.

Además del **DW** principal, existen varios *data marts*; estos son subconjuntos departamentales que focalizan subobjetos seleccionados (por ejemplo, un *data marts* de comercialización puede incluir clientes, productos y datos de ventas). Los datos en el **DW** y los *data marts* son administrados por uno o más servidores, los cuales presentan vistas **MD** a una variedad de herramientas

front end, esto es, herramientas de consulta, generadores de informes, herramientas de análisis y herramientas de *data mining*.

Otros componentes importantes en la arquitectura son el depósito para el almacenamiento y la administración de los metadatos y las herramientas para monitoreo y administración del sistema. El **DW** puede ser distribuido para el balanceo de la carga, para permitir la escalabilidad y para que exista una alta disponibilidad. En el caso del **DW** distribuido, el depósito de los metadatos se replica en cada fragmento **DW** y éste, en forma completa, se administra centralmente.

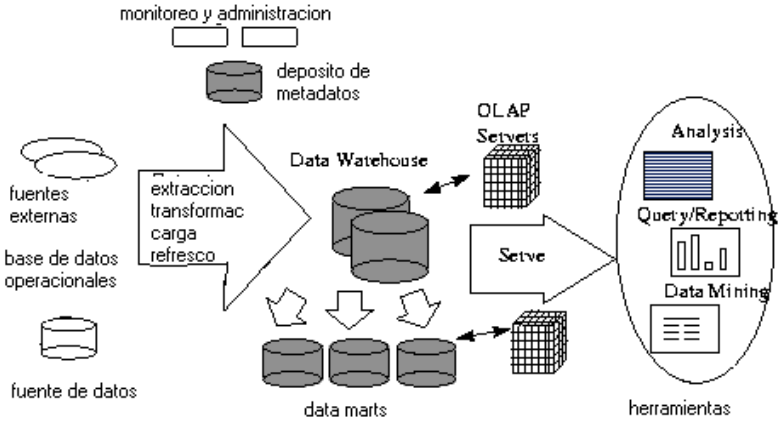


Figura 4.13. Arquitectura General de un Data Warehouse [CD97]

Desde un punto de vista funcional, el proceso de construcción de un **DW** consta de tres fases: a) extracción de datos de diferentes fuentes operacionales distribuidas, b) organización e integración de esos datos en forma consistente dentro del **DW** y, por último, c) acceso a los datos integrados en forma eficiente y flexible. La primera fase incluye los típicos temas vinculados a los servicios de información distribuidos y heterogéneos, tales como inconsistencia de datos, estructuras de datos incompatibles, granularidad de los datos, etc. La segunda fase requiere de técnicas diferentes a las utilizadas para los sistemas de información operacional que ameritan diferentes propuestas metodológicas (como la que planteamos en la tesis). Por último, la tercera fase requiere de capacidades de navegación, optimización de consultas complejas, técnicas de indexación adecuadas y amigables interfaces visuales de usuario para ser utilizados en **OLAP** y *data mining*. [GMR98b].

4.8. Implementaciones de un Data Warehouse

Existen diferentes opciones de implementación física de un **DW.** Una es utilizar un *back end* relacional, donde las operaciones sobre el cubo de datos son traducidas a consultas relacionales (planteadas en un posible dialecto mejorado de **SQL**). La construcción de índices sobre vistas materializadas también se utiliza en estos sistemas, esta alternativa está también planteada en varios productos.

Un **DW** implementado sobre un **RDBMS** se denomina ***Relational OnLine Analitycal Processing*** **(ROLAP)** (**OLAP** Relacional). Estos servidores guardan los datos en un **RDBMS** además de soportar extensiones de **SQL**, accesos especiales y métodos de implementación para hacer más eficiente el modelo **MD** y sus operaciones. En contraste, los servidores ***Multidimensional OnLine Analitycal Processing*** (**MOLAP**), almacenan directamente los datos **MD´s** en estructuras especiales (por ejemplo arreglos) e implementan las operaciones **OLAP** sobre esas estructuras de datos [CD97].

4.8.1. Implementación Relacional

Las complejas consultas requeridas en un ambiente **OLAP** resultan difíciles de realizar en un esquema de tablas altamente normalizadas; esto es debido a que requerirían de múltiples y complejos *joins*. La necesidad o no del uso de las formas normales en las base de datos **MD** surgió desde que éstas fueron utilizadas inicialmente para realizar análisis estadísticos. A diferencia del **RM**, en donde las formas normales fueron definidas con el objetivo de eliminar anomalías en el proceso de actualización, el uso de las formas normales **MD** fue debido a la necesidad de asegurar la sumarización en el esquema y, por consiguiente, el uso correcto de las funciones de agregación en las operaciones de *roll-up* y *dril-down* [LAW98].

La mayoría de los **DW** utilizan el esquema estrella (Figura 4.11) o su versión más normalizada, el copo de nieve (Figura 4.12), que simplifican la estructura de las tablas utilizadas y, por lo tanto, minimizan los *joins* necesarios para realizar las complejas consultas propias de un ambiente **OLAP**.

La elección del enfoque relacional para la implementación del modelo **MD** posee características distintivas que la hacen más convenientes, entre otras, los **RDBMS** tienen estrategias de almacenamiento de datos bien establecidas y estandarizadas; además, se ha realizado importantes investigaciones en este ámbito en temas tales como técnicas de optimización en indexación, operaciones de *join* y materialización de vistas [MZ05].

4.8.2. Esquema Estrella

En el esquema estrella (Figura 4.11), la base de datos está formada por una tabla de hechos y una tabla para cada dimensión. Cada tupla en una tabla de hechos tiene un puntero (clave externa) a cada una de las dimensiones, que provee su coordenada **MD** y, además, almacena la/s medida/s de sus coordenadas. Cada tabla dimensión está compuesta por columnas que corresponden a los atributos de cada dimensión. En el esquema estrella, en general, se establecen relaciones *muchos-a-uno* entre la tabla de hecho y las dimensiones, esta característica simplifica el proceso de consultas tanto como el de **ETL.**

4.8.3. Esquema Copo de Nieve

El esquema estrella no provee explícitamente soporte para niveles de jerarquía. El esquema copo de nieve (Figura 4.12) es un refinamiento del esquema estrella, donde algunos (o todos) los niveles de jerarquías se representa explícitamente, mediante la normalización de las tablas dimensiones, esto conduce a un ventajoso mantenimiento de las mismas. Las estructuras normalizadas de éstas tablas en el esquema copo de nieve son más apropiadas para la inspección de los diferentes niveles en las dimensiones**.**

4.8.4. Esquema Multiestrella

El esquema multiestrella (multi-star) es un ejemplo de estructura más compleja en donde múltiples tablas de hechos son compartidas con las tablas dimensión. Por ejemplo, los gastos proyectados y los gastos actuales pueden formar una constelación de hechos en la medida que comparten una o más dimensiones.

Además de la tabla de dimensiones y de hechos, el **DW** almacena tablas resumidas que contienen datos pre-agrupados. En un caso simple, éstos se corresponden con la agrupación de tablas de hechos sobre una o más dimensiones seleccionadas. De esta manera, la pre-agrupación de datos

resumidos puede ser representada en la base de datos en al menos dos formas [CD97], [VS99].

4.8.5. Esquema K-Dimensional

Otra alternativa en la construcción de base de datos **MD** es el uso de un cubo k-dimensional sobre una estructura especializada no relacional para el almacenamiento de datos. El diseñador especifica todos los agrupamientos que considere útiles. Mientras se construye la estructura de almacenamiento, esos agrupamientos asociados con todos los posibles *roll-ups* son precalculados y almacenados. De esa manera los roll-ups y los *drill-downs* son consultados en forma interactiva. Muchos productos han adoptado esta alternativa [AGS97].

4.9. Proceso de Desarrollo de un Data Warehouse

Construir un **DW** es un proceso largo y complejo que requiere un modelo completo del negocio que puede llevar muchos años en llevarse a cabo. Algunas organizaciones han resuelto este problema mediante los *data marts*; éstos permiten un despliegue más rápido debido a que no precisan consensuar la información de toda la empresa; pero, si no se desarrolla el modelo de la empresa completo, puede crearse un complejo problema de integración en el futuro [CD97].

El esquema **MD** está fuertemente influido por los requerimientos del usuario, además de la disponibilidad y estructura de los datos del **ODB**. La mayoría de los **DW** proyectados tienen un enfoque evolutivo, comienzan con un prototipo que provee cierta funcionalidad y que es posteriormente adaptado a los cambios requeridos a partir de la retroalimentación de los usuarios. Para asegurar la flexibilidad y la reusabilidad del esquema, estos deben ser especificados en un nivel conceptual. Esto significa, especialmente, que no debe asumirse ningún hecho que sea resultado de una posterior etapa de diseño, por ejemplo, decisiones sobre la tecnología de base de datos utilizada [SBHD98].

El diseño de un **DW** que depende de algún tipo de específico de implantación tiene consecuencias negativas; por un lado, es difícil definir un método de diseño que incluya pasos conceptuales generales; además, en consultas específicas, el analista precisa tener en cuenta tediosos detalles vinculados con la organización física de los datos más que en la esencia de los aspectos lógicos. Por lo tanto, tal como sucede en las **RDBMS**, una mejor comprensión de los problemas relacionados con la administración de los bases de datos **MD´s** puede ser alcanzada proveyendo una descripción conceptual de los datos del negocio, independientemente de la forma en que estos sean almacenados [CT98].

El proceso de diseño de un **DW** puede ser desarrollado conforme a tres enfoques: conducido por oferta (supply-driven approach), conducido por demanda (demand-driven approach), o una combinación de ambos (supply/demand-driven approach) [WS03], [RA06]. El enfoque conducido por demanda se focaliza en la determinación de los requerimientos **MD** del usuario para luego derivar las estructuras de almacenamiento adecuadas que los satisfagan. El enfoque conducido por oferta, comienza analizando las fuentes de datos para determinar los conceptos **MD** en un proceso de reingeniería. En el enfoque combinado, conducido por oferta y demanda, se comienza por los requerimientos del usuario (en términos de consultas **SQL**) y, paralelamente, se analizan las fuentes de datos operacionales, para extraer de ellas información que permita validar los requerimientos del usuario.

4.10. Data Warehouse Temporal

Los sistemas **OLAP** permiten analizar los cambios en los valores de los datos a través del tiempo ya que tienen, generalmente, una dimensión "tiempo" que permite la comparación de diferentes periodos, pero no están diseñados para aceptar modificaciones en las dimensiones ya que éstas se consideran (implícitamente) ortogonales.

Por ejemplo, cuando una región de un país se divide en dos o cuando se establece una nueva división departamental, este tipo de modificaciones afectará las nuevas consultas. Esto ocurre porque se considera que los datos reflejan la parte dinámica del **DW** mientras que las dimensiones son relativamente estáticas. Además, se asume que las dimensiones son ortogonales, esto es, independientes entre sí. En particular, la ortogonalidad de la dimensión tiempo significa que las demás dimensiones deberían ser invariantes respecto de la variación temporal [EC00].

4.10.1. Modificaciones en un Data Warehouse

En una implementación **ROLAP**, pensamos a los *hechos* como almacenados en una tabla de *hechos*, en donde cada dimensión se describe mediante tablas dimensión. Es usual asumir que las tablas de *hecho* representan los aspectos dinámicos del **DW** mientras que los datos en las tablas dimensión son relativamente estáticos, pero estos en realidad pueden sufrir modificaciones; por lo tanto, pueden también ocurrir cambios estructurales [HMV99].

Cuando analizamos los cambios que puede sufrir un **DW**, podemos distinguir dos tipos [EKK02]: cambios relacionados con el esquema y cambios relacionados con las instancias; además, estos últimos pueden ser subdivididos en datos de transacción, esto es, los datos introducidos en las celdas o medidas en el esquema **MD** y datos maestros, esto es, las extensiones de los niveles dimensión, por ejemplo, los valores Argentina y Uruguay son datos maestros en la dimensión "País".

En el **DW**, las modificaciones en las medidas pueden ser rastreadas a través del tiempo debido a que, normalmente, existe una dimensión tiempo que permite analizar temporalmente estos cambios; por ejemplo, puedo analizar cómo han sido las ventas de una empresa y compararlas en distintos momentos. Sin embargo, la dimensión tiempo no ayuda a rastrear las modificaciones de los datos maestros. Hay muchas causas por los que los datos maestros pueden ser modificados: cambios en la identificación de los países, reestructuración de los departamentos en una empresa, cambios en los atributos, modificaciones en las unidades de medida, etc.

Otro tipo de modificación que puede sufrir el **DW** está vinculada al esquema. El esquema del **DW** define el tipo de consultas que el usuario puede realizar, puede suceder que surjan nuevas consultas y para ello se deba añadir una nueva dimensión o modificar una jerarquía para establecer un nuevo nivel de agrupamiento; por ejemplo, agregar una dimensión sexo, para poder discriminar si las compras la realizan hombres o mujeres o agregar una nueva jerarquía "bimestre" en la dimensión tiempo, donde antes solo existía año→ trimestre → mes; o bien, eliminar un nivel en la jerarquía; por ejemplo, trimestre. Por otro lado, también es probable que, debido a cambios en los datos fuentes, una dimensión completa deba discontinuarse. Lo mismo puede suceder con las jerarquías.

4.10.2. Actualización de Dimensiones

El problema de la actualización de las dimensiones o la agregación o modificación de jerarquías fue tratado en varios trabajos. En [HMV99], se presentó un modelo formal para la actualización de dimensiones en un modelo **MD** que considera las actualizaciones en los dominios de las dimensiones y las actualizaciones estructurales en las jerarquías, junto con una definición de un conjunto operadores primitivos para realizar dichas actualizaciones; además de un estudio de los efectos de esas actualizaciones sobre las vistas materializadas en los niveles dimensión, más un algoritmo para el eficiente mantenimiento de las vistas.

En [EC00], [EKK02], [EKM01] se estudió el concepto de "versiones de estructuras" que representan una vista sobre el **TDW** que mantiene una estructura válida para un cierto intervalo de tiempo. En [EKM02] se presentó el metamodelo *COMET*, para **DW**, el cual es una **TDB** para todos los elementos del **DW**: esquemas, datos maestros, jerarquías y relaciones entre estas, además de un conjunto de reglas que permitan mantener la integridad de los datos almacenados en el **DW**. Esto lo logra agregando en todas las instancias de las dimensiones y jerarquías una marca de tiempo, mediante un intervalo $[T_i, T_f]$, que representa el tiempo válido. También, para poder registrar todas las modificaciones del esquema del **DW**, realizan marcas temporales a todas las dimensiones y jerarquías. De esta manera se establecen diferentes versiones de la estructura del **DW**, cada de una de ellas estable dentro de un intervalo $[t_i, t_f]$, que son analizadas, cuando se realizan consultas, de modo tal de establecer si ésta sobrepasa una estructura determinada, de modo de evitar consultas erróneas. Plantea, además, funciones de transformación que permiten establecer relaciones entre distintas versiones de estructuras. En la misma línea de investigación, en [EC02], se analizan distintas posibilidades de tratamientos temporales en **OLAP** no temporales, por último, en [EK02] se detallan diferentes enfoques para representar el comportamiento temporal de datos maestros en los **DW** no temporales existentes

Por otro lado, en [AM03a] y [AM03b] identifica similitudes entre las **TDB** y los **DW** y presenta cómo las contribuciones de las primeras pueden beneficiar a las segundas.

4.11. Data Warehouse Histórico

Los **ODB´s** son la fuente de datos del **DW**, aquellos representan sólo el estado actual de los datos y no su historia. Por lo tanto, si bien en el **DW** el tiempo está implícito en su estructura y se utiliza en el análisis de datos, éste hace referencia al momento en que se realizó una transacción, no explícita cómo y cuándo variaron los datos involucradas en ellas. Por consiguiente, aunque la información estuviere almacenada, los mecanismos de búsqueda temporales resultarían complejos [NA02]. La necesidad de evaluar tendencias, variaciones, máximos y mínimos en lapsos determinados, información que se obtiene de una **HDB**; además de realizar las consultas típicas de un **DW**, justifican considerar en el diseño, y representar en forma explícita, cómo los datos pueden variar en el tiempo.

EL concepto de **HDW** es parte de nuestra propuesta de tesis. Este tipo de estructura de almacenamiento, que relaciona un **DW** y una **HDB**, no solamente permite realizar consultas propias de cada uno de las estructuras por separado, sino que también realizar consultas en forma vinculadas (capítulo 7).

4.12. Trabajos Relacionados

Debido que las fuentes de datos transaccionales normalmente provienen de **RDBMS** y además, generalmente, éstas fueron modeladas utilizando el modelo **ER**, éstos pueden proveer una base (enfoque conducido por oferta) para la traducción al esquema **MD** en la medida en que pueden ser desarrolladas un conjunto de reglas de transformación o algoritmos que guíen el proceso. Además, debido a que gran parte del tiempo utilizado en el desarrollo del **DW** se utiliza en la etapa de modelado, la aplicación de las técnicas del modelo **ER**, ampliamente empleadas y estandarizadas, lo reduciría considerablemente. Más aún, si el proceso puede ser automatizado bajo reglas de transformación [KS97]. Por otro lado, el modelo **ER** está extendido como un formalismo conceptual que brinda documentación estándar para sistemas de datos en el **RM** [GMR98a].

Describiremos, a continuación, diversos enfoques que vinculan al modelo **ER** con el diseño de un **DW**.

En [KS97] se examinó la relación entre el esquema estrella (con sus variantes más representativas: copo de nieve y multiestrella) y el modelo **ER**. Se exploraron las relaciones entre el modelo **ER** y el esquema estrella mediante la búsqueda de patrones recurrentes en el proceso de traslación del primer esquema al segundo. Las reglas de transformación, para ambos casos, las presenta mediante una lista informal de acciones que permiten, partiendo de una modelo estrella o multiestrella, obtener un modelo **ER**. Si bien no desarrolla la heurística inversa (**ER** a esquema estrella) sugiere que las consideraciones son también validas para el proceso inverso.

En [SBHD98] se presentó el *M/ER Model* (Modelo **MD ER**). El modelo fue diseñado como una especialización del modelo **ER**; todos los elementos introducidos son casos especiales del modelo original de modo tal de no diezmar su flexibilidad y expresividad. Además, se redujo al mínimo la extensión de las construcciones al modelo para que pueda ser utilizado sin inconvenientes por usuarios experimentados del modelo **ER** clásico, de modo tal que permita representar la semántica **MD** a pesar de la minimalidad de la extensión.

En [CT98] se presentó formalmente el modelo **MD** y un método general que permite el diseño de un **DW** comenzando por una **ODB** descrita en términos del modelo **ER**. El modelo está basado en dos construcciones principales, las dimensiones y las "*f-tables*". Las dimensiones son categorías lingüísticas que describen distintas formas de ver los datos; además, cada dimensión está organizada en niveles de jerarquía que corresponden a la diferentes granularidades en la representación de los datos. Dentro de las dimensiones, los niveles están relacionados a través de funciones "*roll up*". Los atributos de hecho están representados mediante las "*f-tables*" que son el homólogo lógico de los arreglos **MD** que representan funciones que asocian medidas con sus coordenadas simbólicas. Para el diseño de un **DW**, asume la existencia de un modelo **ER** que describe un "primitivo" **DW** que contiene todos los datos operacionales que pueden ser soportadas por el proceso de negocio pero que aun no está adaptada para esa actividad. El método de diseño consta de cuatro pasos: la identificación de los hechos y dimensiones en el modelo **ER** a partir de consideraciones hechas por el diseñador, la reestructuración del mismo para describir los hechos y dimensiones en un formato más explícito, la derivación del grafo dimensional que representa en forma sucinta los hechos y dimensiones del modelo **ER** reestructurado y, por último, la traducción al modelo **MD**. Como punto final de trabajo, propone dos implementaciones del modelo: a un **RDBMS** y a un arreglo **MD**.

En [GMR98a], [GMR98b] se presentó el modelo conceptual gráfico para **DW**, denominado modelo de hechos dimensional (*Dimensional Fact Model, DFM*). En el trabajo propuesto formalizaron el modelo y propusieron un método

semiautomático para construirlo a partir de un modelo **ER** preexistente que representa a un **ODB**. El *DFM* es una colección de esquemas de hechos en estructura de árbol, cuyos elementos básicos son los hechos, los atributos, las dimensiones y las jerarquías. Otras características que pueden ser representadas sobre el esquema de hechos son la aditividad de los atributos a lo largo de las dimensiones, la opcionalidad de los atributos dimensión y la existencia de atributos no-dimensión. En algunos casos, si la información del modelo **ER** no está disponible, el método propuesto puede ser aplicada también comenzando con el esquema de un **RM**, con la condición de que sea conocida la multiplicidad (a-uno o a-muchos) de las asociaciones lógicas establecidas mediante las restricciones de claves foráneas. El método consta de los siguientes pasos: definición de los hechos a partir del modelo **ER** existente; por cada hecho, se construye un cuasi-árbol de atributos a partir de un algoritmo recursivo; sobre ese cuasi-árbol, a partir de consideraciones de diseño, se "injerta" y "poda" con el objetivo de eliminar detalles innecesarios; luego se definen, también a partir de consideraciones del diseñador, las dimensiones, las jerarquías sobre esas dimensiones y los atributos de hecho. El algoritmo utilizado en este trabajo será adaptado y formalizado en nuestra propuesta.

En [TBC99] se presentó el *starER*, un modelo conceptual de alto nivel de abstracción, cuyo objetivo principal fue trasladar los requerimientos del usuario a una representación abstracta compresible por él, independientemente de la implantación, pero formal y completa de modo tal que pueda ser transformado a un modelo lógico sin ambigüedades. Combina la estructura del esquema estrella, dominante en el ambiente del **DW**, con las construcciones semánticamente ricas del modelo **ER** utilizadas desde hace muchos años y que provee suficiente poder para modelar aplicaciones complejas agregando extensiones al modelo original, pero siempre bajo una misma base conceptual. Debido a que el **DW** impone nuevas estructuras de modelado, el nuevo modelo adiciona construcciones que permiten modelar hechos, dimensiones y jerarquías. Presentado el modelo lo evalúa basado en los criterios expuestos en [PJ98a].

4.13. Resumen del Capítulo

El objetivo de este capítulo fue presentar las características principales de un **DW** y sus diferencias con los **ODB**. Se detallaron los distintos tipos de jerarquías, las diferentes formas de visualizar los datos en la estructura **MD** y las alternativas de implantación. Luego, se contrastaron las diferencias entre el **DW** con el **TDW** y **HDW** y, por último, se describieron los más importantes trabajos donde se vincula al modelo **ER** con el diseño de un **DW.**

Capítulo 5

Diseño de un Data Warehouse Histórico

5.1. Introducción

En este capítulo detallaremos el proceso de transformación que permitirá diseñar, a partir de un modelo **ER**, un **HDW**. Las transformaciones serán descritas de manera informal. El proceso de diseño comienza utilizando un modelo **ER** que representa al modelo de datos fuente y que expresa en forma abstracta al **ODB** de la organización; en él, el diseñador determinará qué entidades, atributos e interrelaciones querrá preservar temporalmente, además de establecer cuál será el hecho principal de análisis del **DW**; luego, se derivará un modelo **TER** sobre cuya estructura será aplicado un algoritmo recursivo que permitirá obtener un ***Temporal Attribute Graph*** **(TAG)**; a partir de éste y mediante decisiones del diseñador respecto a qué medidas, dimensiones y niveles de jerarquías considera adecuadas, se obtendrá un modelo **TMD**; luego, se derivará, primeramente, el **RM** que describirá al **TMD** en una implementación relacional y, posteriormente, las tablas relacionales expresadas mediante sentencias **SQL,** que permitirán implementarlo en un **RDBMS**.

5.2. Modelo Multidimensional Temporal

El modelo **TMD** está compuesto por un hecho[9] y un conjunto de dimensiones, éstas últimas se representan mediante niveles de jerarquías (temporales y no temporales) simples o múltiples. El hecho principal puede tener uno o más atributos de hecho (de ahora en más, medidas), además de atributos que no se interpretan como medidas pero que si podrán utilizarse para identificar una instancia particular del hecho (dimensiones degeneradas).

Esquemáticamente[10] (Figura 5.1), el modelo está compuesto por un *hecho* (H) que contiene un conjunto de atributos (idi_0) que, en forma individual,

[9] El método propuesto considera en su desarrollo a un *hecho* por vez

[10] Los atributos identificadores estarán representados entre "{ }" , los atributos referencias entre "[ ]" y las medidas y atributos no dimensión entre "( )"

hacen referencia a cada una de los niveles *hoja* de las dimensiones (ni_0) y, en conjunto, identifican a una instancia particular del *hecho;* además, el hecho puede contener una o más medidas (med_i).

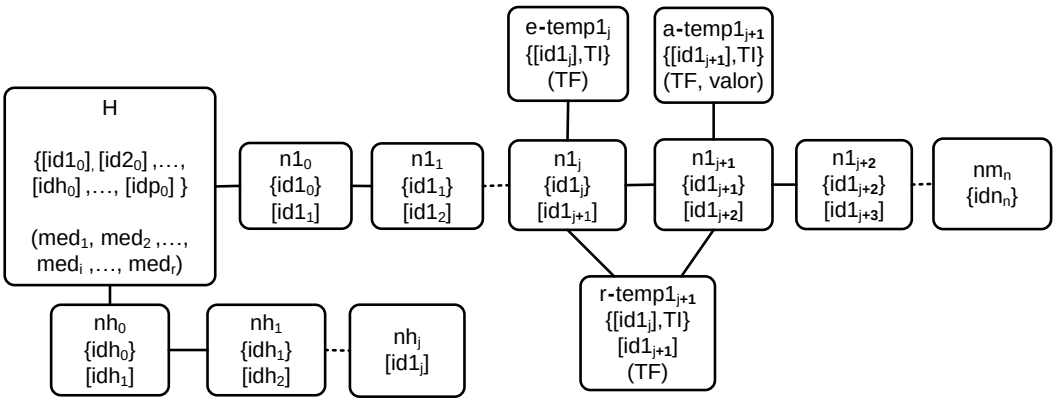


Figura 5.1. Modelo Multidimensional Temporal

Cada dimensión n_i (i, i= 1, 2,…, n) está formada por niveles de jerarquías n_{ij} (i, i= 1, 2,…, n; j, i= 1, 2,…, m); todos los niveles tendrán un atributo identificador (idn_i) más un atributo que hace referencia al nivel *padre* (idn_{i+1}), a excepción del nivel *raíz,* por ser el último nivel de la jerarquía; además, podrán tener atributos descriptivos[11] (atributos no-dimensión).

Los niveles de jerarquía ($e\text{-}temp_i$, $a\text{-}temp_i$, $r\text{-}temp_i$), que representan, respectivamente, entidades, atributos, e interrelaciones temporales, constituyen jerarquías no estrictas.

El nivel de jerarquía que simboliza a una entidad temporal ($e\text{-}temp_i$) tiene un atributo compuesto que lo identifica, formado por el identificador del nivel que representa a la entidad (hará referencia a ella) más el atributo denominado TI ({idn_i, TI}); además, posee un atributo denominado TF.

El nivel de jerarquía que representa a un atributo temporal ($a\text{-}temp_i$) tiene un atributo compuesto que lo identifica ({idn_i, TI}), formado por el identificador del nivel que representa la entidad (hará referencia a ella) más el atributo denominado TI, además, posee dos atributos denominados TF y valor.

El nivel de jerarquía que representa a una interrelación temporal ($r\text{-}temp_i$) tiene un atributo compuesto que lo identifica ({idn_i, TI}), formado por el identificador de uno de los niveles que vincula (además, hará referencia a él) más el atributo denominado TI, también posee un atributo que hará referencia al otro nivel que vincula (idn_{i+1}), más un atributo denominados TF.

En las jerarquías temporales, los atributos TI y TF representan a los valores extremos del intervalo temporal [TI, TF).

El modelo **TMD** no admite relaciones *muchos-a-muchos* entre el hecho y los niveles *hoja* en las jerarquías, solo relaciones *muchos-a-1*[12], como tampoco permitirá modelar jerarquías asimétricas, todas se consideran simétricas ya que el algoritmo no contempla la multiplicidad mínima y, por lo tanto, el método asume a la multiplicidad de rol como (1, x)[13]. Todas las jerarquías serán estrictas (relaciones *x-a-1*) entre niveles *padre* e *hijo*, con excepción, como se aclaro anteriormente, de los niveles de jerarquía temporal. Las multiplicidades de rol en los niveles de jerarquía y entre el hecho principal y los niveles hoja, no estarán

[11] Por simplicidad, no serán detallados explícitamente en el gráfico

[12] Esto es debido a una restricción del algoritmo utilizado para la construcción del **TMD**

[13] Esta restricción podría ocasionar, en el proceso de carga de datos, algunos inconvenientes; debido a que, si la multiplicidad de los roles asociados a las interrelaciones del **TER** fueran (0, 1), podría implicar que, si algunos valores no estuvieran presentes, las consultas devolverían valores anómalos.

descritos explícitamente en el modelo, se asumirán a partir de lo explicado anteriormente.

El modelo no discriminará entre medidas aditivas, semi-aditivas o no aditivas; esto corresponderá a un análisis posterior a partir de la semántica de los mismos.

5.3. Método de Diseño de un Data Warehouse Histórico

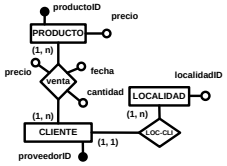
El proceso de transformación propuesto se iniciará a partir de los datos fuente expresados en un modelo **ER** (enfoque conducido por oferta) que representa a la **ODB** de la organización; si bien, para la obtención del **HDW,** el método considera sólo un hecho (para cada esquema resultante de la aplicación del método), este se podrá combinar con otros hechos (incluso a partir del mismo **TER)**, e integrarlos entre sí en la medida que tengan, al menos, una dimensión en común, para formar un esquema multiestrella, de modo de poder realizar consultas del tipo *drill-across.*

5.3.1. Visión General del Proceso de Transformación Informal

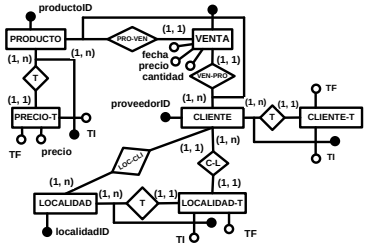
El método de transformación, que comienza con un modelo **ER** que describe el esquema de datos fuente de la **ODB**, hasta la obtención de un **HDW** implementado en un **RDBMS,** plantea una serie de pasos, descritos de manera informal, que detallaremos a continuación y que, en resumen, comenzará con un modelo **ER** y, mediante sucesivas transformaciones, permitirá obtener un conjunto de tablas en un **RM** expresadas en sentencias **SQL**.

Presentaremos, a continuación, una visión general del método y, mediante un ejemplo, detallaremos cómo es el proceso de transformación. Luego, de manera informal, y paso por paso, detallaremos cada una de las transformaciones en forma detallada.

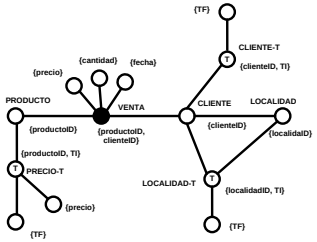
Inicialmente, comenzando con un modelo **ER**, obtendremos un **TER** adaptado (incluye la transformación de la interrelación *hecho* a un entidad *hecho*) (Figura 5.2, transformación #1 y #2); luego, del **TER** adaptado modelaremos un **TGA** (Figura 5.2, transformación #3); después, a partir de éste obtendremos un **TMD** (Figura 5.2, transformación #4); a continuación, del **TMD** modelaremos un **RM** (Figura 5.2, transformación #5) y, finalmente, obtendremos sentencias **SQL** para su implementación en un **RDBMS** (Figura 5.2, transformación #6). Las transformaciones #7 y #8 (Figura 5.2), serán descritas en el capítulo 7.



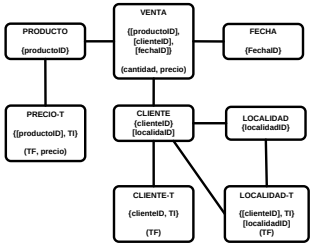
TRANSFORMACIÓN #1, #2



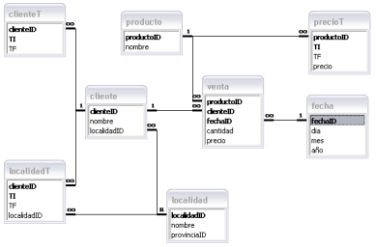
TRANSFORMACIÓN #3



TRANSFORMACIÓN #4

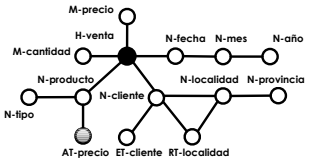


TRANSFORMACIÓN #5



TRANSFORMACIÓN #6

```
CREATE TABLE VENTA (
    productoID numeric NOT NULL,
    clienteID numeric NOT NULL,
    fechaID date NOT NULL,
    cantidad numeric NOT NULL,
    precio numeric NOT NULL,
    PRIMARY KEY (productoID,
                 clienteID,
                 fechaID),
    FOREIGN KEY (clienteID)
    REFERENCES PRODUCTO,
    FOREIGN KEY (fechaID)
    REFERENCES CLIENTE,
    FOREIGN KEY (productoID)
    REFERENCES FECHA,
    )
```



TRANSFORMACIÓN #7

```
PARAMETERS t datetime, p text;
SELECT pt.precio
FROM producto AS p, preciot AS pt
WHERE p.productoid  = pt.productoid
AND p.productoid = p
AND ti <= t
AND tf >= t;
```

TRANSFORMACIÓN #8

Figura 5.2. Proceso Completo de Transformaciones

5.4. Transformaciones Descritas Informalmente

A continuación, describiremos el conjunto de transformaciones que luego serán formalizados en el Anexo IV.

- Transf. #1: modelo de datos al modelo de datos temporal.
- Transf. #2: modelo de datos temporal al modelo temporal adaptado.
- Transf. #3: modelo temporal adaptado al grafo de atributos temporal
- Transf. #4: grafo de atributos temporal al modelo multidimensional temporal
- Transf. #5: modelo multidimensional temporal al modelo relacional.
- Transf. #6: modelo relacional a tablas mediante sentencias **SQL**.

5.4.1. Transformación del Modelo de Datos al Modelo Temporal Adaptado

El método propuesto en la transformación del modelo **ER** al **TER** adaptado permitirá modelar, en este último, el tiempo de vida de entidades además del tiempo válido, tanto de atributos como de interrelaciones.

El tiempo de vida de las entidades no será considerado como tal, ya que complejizaría innecesariamente la estructura del **TMD**[14]. Como hemos descrito en el capítulo 3, el tiempo de vida de una entidad "E" puede ser vista como el tiempo válido del hecho "´E´ existe"; por lo tanto, en el modelo temporal, el tiempo de vida de una entidad será registrada, en forma indirecta, mediante el concepto de tiempo válido.

Para expresar el tiempo válido, utilizaremos la notación [TI, TF) como representación del intervalo de validez, donde TI será el primer instante y TF el último en el intervalo descrito; Por otro lado, al intervalo lo consideramos *cerrado/abierto*, esto es, incluirá el instante TI y excluirá el instante TF; se utilizará la palabra reservada *now* para designar el tiempo actual, el cual estará en continua expansión; por último, se adoptará el marcado de las interrelaciones temporales, para que expliciten visualmente su condición de tal.

Se utilizará, para la representación del modelo temporal, un enfoque explícito, con soporte temporal explicito; esto es, se retendrá la semántica no temporal del modelo **ER** y se adicionarán nuevas construcciones sintácticas para la representación de las entidades, atributos e interrelaciones temporales en el **TER.**

La transformación del modelo **ER** al **TER** estará subdividida en dos pasos; en el primero, se transformará el modelo **ER** en un **TER**; en particular, aquellas entidades, atributos e interrelaciones que el diseñador considerase conveniente mantener su historia; luego, en el **TER** resultante**,** la interrelación que representa al hecho principal del **HWD** (si amerita) será transformada en entidad. El resultado es un **TER** adaptado. En el caso (no muy usual) en que el hecho ya esté expresado mediante una entidad en el **TER**, se omitirá este último paso.

5.4.1.1. Transf. #1: Modelo de Datos al Modelo de Datos Temporal

En esta primera transformación se considerarán cuáles entidades, atributos e interrelaciones interesarán ser preservadas en el tiempo. El diseñador del **HDW** decidirá cuáles de estas construcciones se convertirán en temporales. La transformación se detallará a continuación.

[14] Principalmente en la creación de consultas que precisaran de claves foráneas compuestas y que necesitaran, además, especificar valores temporales de existencia actual en los niveles de jerarquía que hayan sido considerados en el modelo **TER**, como entidad temporal.

5.4.1.1.1. Entidad temporal

Toda entidad identificada como temporal, en el modelo **ER** fuente, se representará, en el modelo **TER** destino mediante una entidad (débil) temporal asociada a la entidad (ex) temporal (regular). La nueva entidad temporal llevará como nombre el mismo de la entidad regular y terminará en "-T"; contendrá un atributo denominado TF[15] y un identificador único (compuesto) formado por el identificador de la entidad regular más un atributo denominado TI; el intervalo [Ti, TF), representará, ahora, el tiempo de vida de la entidad temporal.

El vinculo entre estas dos entidades estará marcado con una "T", la multiplicidades de rol de la entidad débil será (1, 1), La multiplicidad de rol de la entidad regular (1, N).

> Por ejemplo, la entidad *CLIENTE* (Figura 5.3, izquierda) tendrá asociada a la entidad temporal *CLIENTE-T* (Figura 5.3. derecha) que modelará (en forma indirecta) su tiempo de vida.

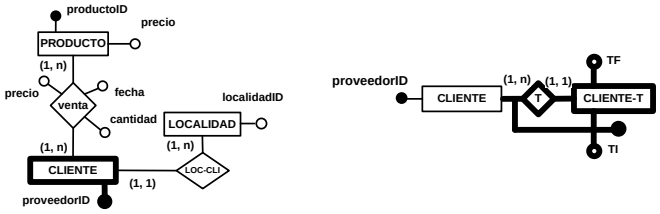


Figura 5.3. Transformación de una Entidad Temporal

5.4.1.1.2. Atributo Temporal[16]

Todo atributo identificado como temporal, en el modelo **ER** fuente, será transformado en el modelo **TER** destino en una entidad (débil) temporal vinculada a la entidad a la que pertenece dicho atributo. El nombre de la entidad temporal será igual al nombre del atributo temporal y terminará en "-T"; tendrá como atributos descriptivos, el atributo denominado TF además de un atributo que tendrá el mismo nombre y dominio que el atributo transformado en temporal en el modelo fuente; el identificador único (compuesto) de la entidad temporal estará formado por el atributo denominado TI más el identificador de la entidad regular.

El intervalo [Ti, TF), representará el tiempo válido del atributo temporal. La multiplicidad de rol de la entidad temporal será (1, 1), la multiplicidad de rol de la entidad regular será (1, N). El atributo que fue transformado a temporal desaparecerá como tal en la entidad regular del modelo **TER**.

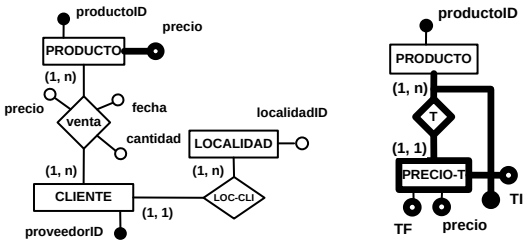


Figura 5.4. Transformación de un Atributo Temporal

[15] Los atributos denominados TI y TF tendrán como dominio un conjunto de valores temporales.

[16] Omitiremos, por simplicidad y sin pérdidas semánticas, el paso intermedio de transformación de la entidad temporal al un atributo multivaluado compuesto tal como fue expresado en la unidad 2

Por ejemplo, el atributo precio de la entidad *PRODUCTO* (Figura 5.4, izquierda) se transformará en la entidad temporal *PRECIO-T* (Figura 5.4, derecha).

5.4.1.1.3. Interrelación Temporal

Toda interrelación identificada como temporal, en el modelo **ER** fuente, se transformará, en el **TER** destino en una entidad (débil) temporal vinculada a una de las entidades que forman la interrelación. El nombre de la nueva entidad se establecerá como combinación del nombre de la entidad regular elegida y terminará en "-T", las multiplicidades de rol de las entidades vinculadas será (1, N), las multiplicidades de rol de la entidad temporal será (1, 1). La nueva entidad temporal tendrá como atributo descriptivo el atributo denominado TF más los atributos que tuviere la interrelación en el modelo **ER** fuente. El identificador único (compuesto) de la entidad temporal estará formado el atributo denominado TI más el atributo identificador de la entidad que le dio el nombre (representará una entidad débil de aquella). La interrelación que se transformó en temporal, se mantendrá como no temporal, para permitir en el proceso de transformación ulterior ser fuente de un posible nivel agrupamiento en la dimensión.

Por ejemplo, la interrelación *LOC-CLI* entre las entidades *LOCALIDAD* y *CLIENTE* (Figura 5.5, izquierda) se transformará en la entidad temporal *LOCALIDAD-T* (Figura 5.5, derecha)

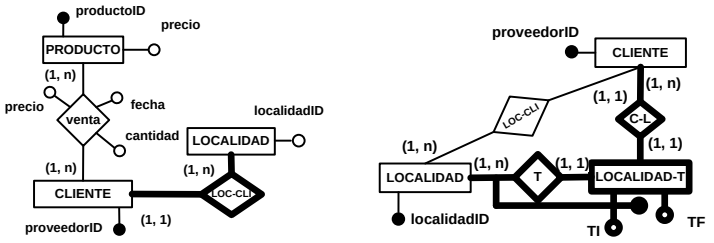


Figura 5.5. Transformación de una Interrelación Temporal

5.4.1.1.4. Entidades, Interrelaciones y Atributos

Todas las entidades e interrelaciones no temporales, en el modelo **ER** fuente, se mantendrán sin modificaciones en el modelo **TER**[17].

Todos los atributos (tanto de entidades cómo de interrelaciones) no temporales, en el modelo **ER** fuente, se mantendrán en el modelo **TER**; no así los atributos transformados en temporales.

5.4.1.2. Transf. #2: Modelo de Datos Temporal al Modelo Temporal Adaptado

La segunda transformación representará, a partir de un *hecho* modelado por una interrelación, el mismo hecho descrito mediante una entidad. Esta transformación que no agrega, semánticamente, modificaciones en el modelo, se realizará a efectos de la aplicación ulterior del algoritmo recursivo para la obtención del **TAG**.

17 Se preservará la interrelación que se trasformó a temporal como interrelación estándar para no anular un posible nivel de jerarquía en el **HDW**.

5.4.1.2.1. Interrelación "Hecho"

El hecho (si fuese representado por una interrelación) en el modelo **ER** fuente, se transformará en el modelo **TER** destino, en una entidad vinculada a las dos entidades que forman parte de la interrelación. El nombre de la entidad será el mismo que el de la interrelación; la multiplicidad de rol de las entidades vinculadas será (1, N), la multiplicidades de rol de la entidad transformada será (1, 1); tendrá como atributos descriptivos los atributos de la interrelación (si los tuviere) y, como identificador único, la unión de los atributos identificadores de las entidades vinculadas a la interrelación.

Por ejemplo, el hecho "venta" presentado por la interrelación *VENTA* (Figura 56, izquierda) se transformará en el mismo hecho "venta" pero ahora modelado por la entidad *VENTA* ((Figura 5.6, derecha)

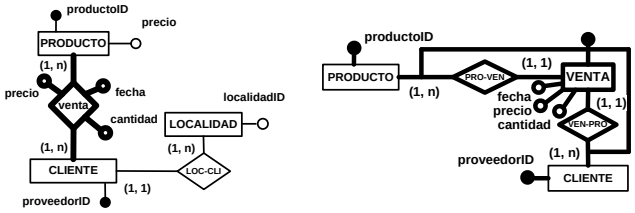


Figura 5.6. Transformación de una Entidad *hecho* a Interrelación *hecho*

En la figura 5.7, izquierda, se muestra la transformación completa del modelo de datos al modelo de datos temporal adaptado

5.4.2. Transf. #3: Modelo de Datos Temporal al Grafo de Atributos Temporal

Como paso intermedio en el diseño del **HDW** se construirá, a partir del modelo **TER** adaptado, un **TAG** modificado de [GMR98a] que se utilizará, luego, como modelo fuente para la construcción del **TMD**.

Para la construcción del **TAG** se identificará, primeramente, el hecho principal[18] en el **TER** adaptado; aquel, como concepto de interés primario para el proceso de toma de decisión en el **HDW**, corresponderá a eventos que ocurren dinámicamente en la realidad y estará representado, en el modelo **TER** adaptado, mediante una entidad.

El grafo estará compuesto por vértices; cada uno de los cuales representará a un atributo, que podrá ser simple o compuesto y que pertenecerá al modelo **TER** fuente.

Detallaremos las características del grafo:

- La raíz del grafo corresponderá al identificador del hecho en el modelo **TER.**
- Para cada vértice no temporal *v*, su/s atributo/s asociado/s correspondiente/s determinará/n funcionalmente a todos los atributos que correspondan a los descendientes de v.
- Cada vértice *v* marcado con una "T" corresponderá a un vértice temporal; este a su vez, puede ser "Terminal" o "no-Terminal", el

[18] La decisión en la elección del cual será el hecho principal, corresponderá a criterios del diseñador en la transformación anterior.

primero se habrá derivado de un atributo o una entidad temporal en el modelo **TER**; el segundo, de una interrelación temporal.

- Los vértices "Terminal" no tienen vértices descendientes cuyos atributos sean identificadores.
- Los vértices "no-Terminal" tienen vértices descendientes cuyos atributos sean identificadores.

5.4.2.1. Construcción del Grafo de Atributos Temporal

Dado un identifier(E) que indica un conjunto de atributos que identifican a la entidad E en el modelo **TER**, el **TAG** (Figura 5.7, derecha) podrá ser construido automáticamente mediante la aplicación de la siguiente función recursiva.

```
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| mult-max(E,R)=1 or
            (R is temporal and E is no temporal) do
   //se consideran entidades temporales
   //y se evitan bucles infinitos
            {for each  attribute  b ∈ R do
            addChild (v, translate(G)); }
   return(v)
}
```

Por ejemplo, tomando el modelo fuente **TER** adaptado (Figura 5.7, izquierda) se obtendrá automáticamente, mediante el algoritmo, el siguiente **TAG** (Figura 5.7, derecha)

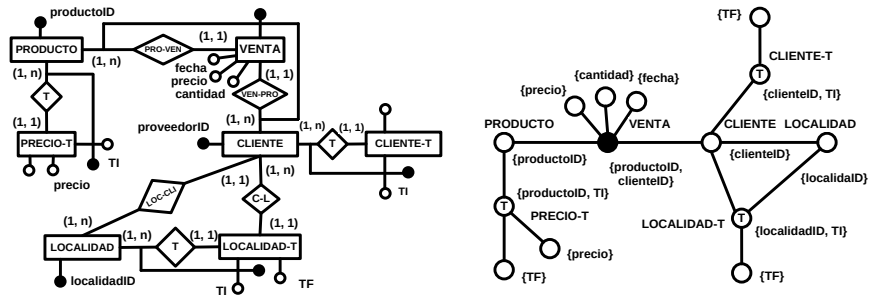


Figura 5.7. Transformación del Modele temporal Adaptado al Grafo de Atributos

A continuación, ilustraremos (Figura 5.8) con el ejemplo del modelo **TER** propuesto (Figura 5.7, izquierda), paso por paso, la construcción del **TAG** (Figura 5.7, derecha), utilizando el algoritmo recursivo.

```
traslate(VENTA)
       v = <label = VENTA, identifier = {productoID, clienteID}>
       addchild({productoID, clienteID}, {cantidad})
       addchild({productoID, clienteID}, {fecha})
       addchild({productoID, clienteID}, {precio})
       G= PRODUCTO
       addchild({productoID, clienteID}, traslate(PRODUCTO))
traslate(PRODUCTO)
       v = <label = PRODUCTO, identifier = {productoID}>
```

```
       G= PRECIO-T
       addchild({productoID}, traslate(PRECIO-T))
traslate(PRECIO-T)
       v = <label= PRECIO-T, identifier = {productoID, TI}>
       addchild({productoID, TI }, {TF})
       addchild({productoID, TI }, {precio})
       G = CLIENTE
       addchild({productoID, TI}, traslate(CLIENTE))
traslate(CLIENTE)
       v = <label = CLIENTE, identifier = {clienteID}>
       G = CLIENTE-T
       addchild({clienteID }, traslate(CLIENTE-T))
traslate(CLIENTE-T)
       v = <label = CLIENTE-T, identifier = {clienteID, TI}>
       addchild({clienteID, TI}{TF})
       G= LOCALIDAD
       addchild({clienteID}, traslate(LOCALIDAD))
traslate(LOCALIDAD)
       v = <label = LOCALIDAD, identifier = {localidadID}>
       G= LOCALIDAD-T
       addchild({localidadID}, traslate(LOCALIDAD-T))
traslate(LOCALIDAD-T)
       v = <label= LOCALIDAD-T, identifier={localidadID, TI}>
       addchild({localidadID, TI}, {TF))
```

Figura 5.8. Instanciación del Grafo de Atributos

5.4.2.2. Algunas Consideraciones Sobre el Grafo de Atributos Temporal

A continuación estableceremos algunas consideraciones a tener en cuenta en la creación del **TAG,** vinculadas con las interrelaciones, atributos y eliminación de detalles innecesarios.

5.4.2.2.1. Interrelación uno-a-uno

Las interrelaciones uno-a-uno deben ser interpretadas como un tipo particular de relación muchos-a-uno, por lo tanto éstas pueden ser incluidas en el **TAG**. Sin embargo, en una consulta en un **DW**, el *drill-down* a lo largo de una relación uno-a-uno significa adicionar una fila cabecera al resultado sin introducir mayores detalles. Por lo tanto en algunos casos es conveniente modificar el **TAG** representándolos como atributos no-dimensión.

5.4.2.2.2. Interrelación x-a-muchos

Las interrelaciones x-a-muchos (no temporales) en el modelo **TER**, por restricciones propias del algoritmo, no podrán incluidas en el **TAG**[19].

5.4.2.2.3. Entidad, Atributo e Interrelación Temporales

Los componentes temporales del modelo se transformarán en entidades vinculadas con interrelaciones marcadas con "T" del tipo x-a-muchos, Este tipo de interrelaciones sí serán incluidas en el **TAG**; no obstante, por lo comentado anteriormente, no podrán ser utilizadas para realizar agregaciones, corresponderán a jerarquías no estrictas en el **HDW** que permitirán consultas temporales.

[19] Si así no fuere, originarían en el **HDW**, jerarquías no estrictas.

5.4.2.2.4. Atributo Multivaluado

Los atributos multivaluados, debido a las restricciones impuestas por el algoritmo, no podrá ser incluido como tal el **TAG**.

5.4.2.2.5. Eliminación de Detalles Innecesarios

Probablemente, no todos los atributos representados como vértices en el **TAG** sean interesantes para el **HDW**. Por tal motivo, éste podrá ser modificado, por el diseñador, para descartar los niveles de detalles innecesarios.

5.4.3. Transf. #4: Grafo de Atributos Temporal al Modelo Multidimensional

El proceso de transformación del **TAG** al modelo **TMD**, que implica la elección de cuáles vértices del grafo serán medidas en el hecho y cuales dimensiones o niveles de jerarquías (temporales o no) dependerá de las decisiones del diseñador. No así la elección del hecho, ya que éste se derivará directamente de la raíz del grafo.

Los atributos identificadores de los nodos en el grafo serán conservados en el **TMD** para la posterior transformación al **RM**.

El criterio general que utilizaremos en la transformación es el siguiente:

5.4.3.1. Hecho

La raíz del grafo en el **TAG** corresponderá al hecho en el modelo **TMD**, su nombre será igual al de la raíz. Los atributos identificadores en el nodo raíz serán atributos identificadores en el hecho[20].

> Por ejemplo, la raíz del **TAG**, *VENTA* (Figura 5.9, izquierda) se transformará en el hecho *VENTA* en el modelo **TMD** (Figura 5.9, derecha)

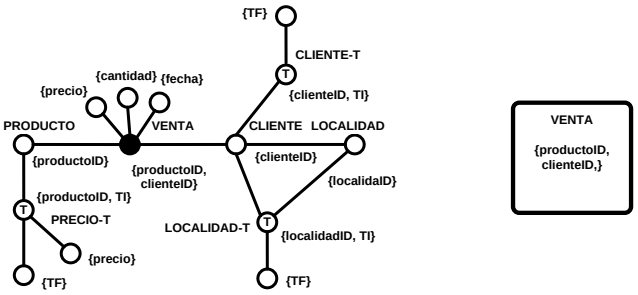


Figura 5.9. Transformación de la Raíz en Hecho

5.4.3.2. Medidas

Todos los vértices vinculados con la raíz en el **TAG**, que no sean ni identificadores ni atributos que denoten aspectos temporales (estos últimos serán transformados en *hojas* del nivel de jerarquía temporal), corresponderán a medidas. Un hecho podría carecer de medidas[21].

[20] El atributo identificador del nivel hoja de la dimensión temporal, será agregado cuando se considere dicha dimensión como tal.
[21] Cuando solamente la información a ser registrada sea la ocurrencia del hecho mismo.

Por ejemplo, los vértices *cantidad* y *precio* asociados a la raíz *VENTA* en el **TAG** (Figura 5.10, izquierda) se transformarán en las medidas *cantidad* y *precio* del hecho *VENTA* en el modelo **TMD** (Figura 5.10, derecha).

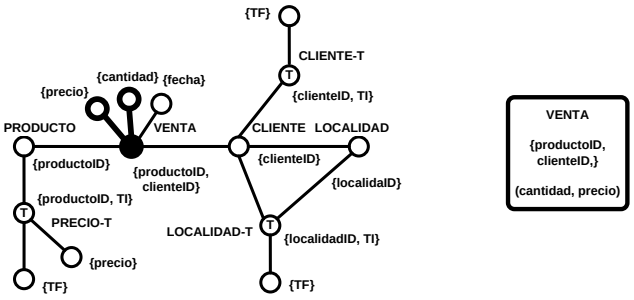


Figura 5.10. Transformación de Vértices en Medidas

5.4.3.3. Nivel Hoja en la Jerarquía

Los niveles hoja en la jerarquía deberán ser elegidas en el **TAG** entre los vértices *hijos* de la raíz. Su elección será crucial para el diseño del **HDW** dado que éstos determinarán el más bajo nivel de granularidad de las instancias de hecho.

En el **TAG**, los vértices vinculados al hecho, que sean identificadores, serán las *hojas* en el modelo **TMD**; los vértices vinculados a ellas, que no sean identificadores, serán atributos no dimensión, el nombre del vértice será igual al de la *hoja*. La multiplicidad de rol del hecho será (1, N), la multiplicidad de rol *hoja* será (1, 1)[22]. Los niveles *hoja* en la jerarquía están vinculados con el hecho principal, por lo tanto, cada uno de los atributos identificadores del hecho será una referencia a cada uno de niveles *hoja* de la jerarquía, de modo tal de establecer una relación "muchos-a-1" entre el hecho y sus niveles *hoja*.

Por ejemplo, los vértices *PRODUCTO* y *CLIENTE*, asociados a la raíz (Figura 5.11, izquierda) en el **TAG,** se transformarán en los niveles *hoja PRODUCTO* y *CLIENTE* (Figura 5.11, derecha) en el modelo **TMD**.

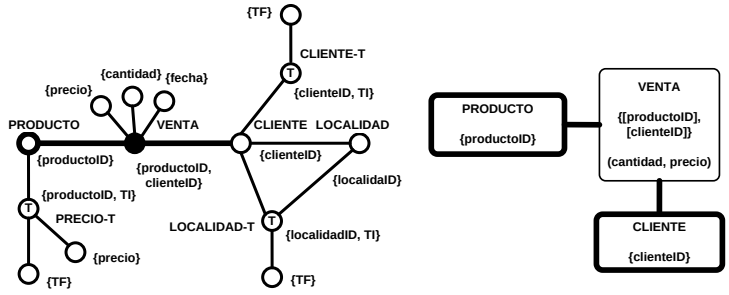


Figura 5.11. Transformación de Vértices en Dimensiones

5.4.3.4. Nivel Hoja Temporal

El hecho debería tener asociado un vértice que denote aspectos temporales; éste se convertirá en *hoja* temporal de la dimensión tiempo en el **TMD;** si así no fuere, debido a que en el modelo **TER**, no existiese un atributo de esas características, la hoja temporal se forzará[23] directamente en el modelo **TMD** y se agregará el identificador de la dimensión temporal al identificador del hecho. La

[22] Las multiplicidades serán las características distintivas del modelo, por lo tanto no se las detallarán explícitamente en el modelo **TMD.**
[23] De esta forma se obtendrá la dimensión tiempo que caracteriza a un **DW**

multiplicidad de rol del hecho será (1, N), la multiplicidad de rol *hoja* temporal será (1, 1).

Por ejemplo, el vértice *fecha*, asociado a la raíz en el **TAG** (Figura 5.12, izquierda), se transformará en *hoja temporal FECHA* (Figura 5.12, derecha), correspondiendo a la necesaria dimensión temporal que caracteriza a un **HDW.** Además, ahora al identificador del hecho se le agregará, como parte componente, el identificador del nivel *hoja* fecha, que, además, será una referencia al nivel fecha.

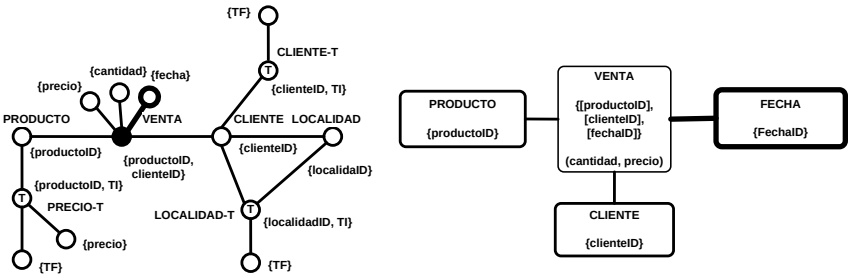


Figura 5.12. Transformación del Vértice en Dimensión Temporal

5.4.3.5. Niveles en la Jerarquía

Los vértices vinculados a los nodos adyacentes a la raíz, en el **TAG**, que sean identificadores, serán transformados en niveles *padre* de la jerarquía en el **TMD** (Figura 5.1), el nombre del nivel *padre* será igual al del vértice y su identificador será el mismo que el identificador del nodo; todos los vértices vinculados a ese nivel, que sean identificadores, serán también niveles *padres* vinculados. Los vértices que no sean identificadores, serán atributos del nivel.

Todos los niveles *hijo* en la jerarquía tendrán un atributo que hará referencia al nivel *padre*. En todos los niveles de la jerarquía, la multiplicidad de rol del *padre* será (1, 1), la multiplicidad de rol *hijo* será (1, N).

Por ejemplo, el vértice *LOCALIDAD* asociado al vértice *CLIENTE* en el **TAG** (Figura 5.13, izquierda), se transformará en nivel *padre LOCALIDAD* (Figura 5.13, derecha) perteneciente a la dimensión *CLIENTE* en el modelo **TMD**.

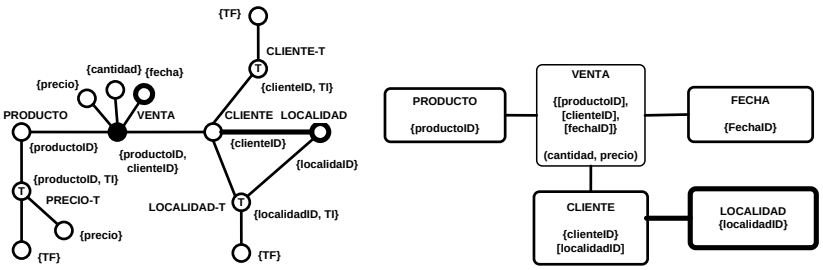


Figura 5.13. Transformación del Vértice en Jerarquía

5.4.3.6. Nivel Temporal

Los vértices temporales (marcados con "T"), en el **TAG,** vinculados a los vértices transformados en niveles de la jerarquía en el **TMD**, serán trasformados en niveles temporales, el nombre del nivel será igual al del vértice y el identificador será el mismo que el identificador del nodo; si el vértice fuera "Terminal", todos los vértices vinculados, serán hojas y por lo tanto, no serán identificadores,

serán atributos de la jerarquía temporal en el modelo **TMD,** además, el atributo que no forma parte del intervalo temporal, hará referencia al nivel de jerarquía vinculado. Si el vértice fuera "no-Terminal", estará vinculado a las dos jerarquías no temporales descritas en el grafo, por lo tanto, el nivel temporal tendrá referencias a esos dos niveles. Todos los vértices vinculados a los nodos temporales que no sean identificadores, serán atributos del vértice temporal.

Por ejemplo, los vértices temporales *PRECIO-T, CLIENTE-T* y *LOCALIDAD-T* en el **TAG** (Figura 5.14, izquierda) se transformarán en los niveles temporales *PRECIO-T, CLIENTE-T* y *LOCALIDAD-T*, respectivamente en el modelo **TMD** (Figura 5.14, derecha).

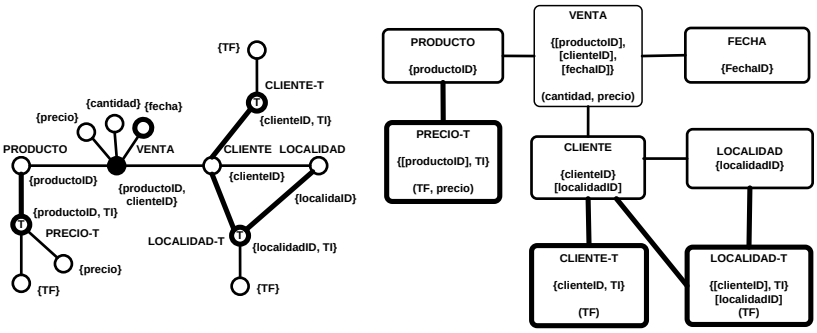


Figura 5.14. Transformación de Vértices en Jerarquías Temporales

5.4.4. Transf. #5: Modelo Multidimensional Temporal al Modelo Relacional

El paso siguiente en la transformación es la creación de un **RM**. Por lo tanto, a partir del **TMD** obtendremos, aplicando las siguientes reglas de transformación, un conjunto de estructuras de datos relacionales.

El proceso de transformación será detallado a continuación.

5.4.4.1. Hecho

La representación del hecho en el **TMD,** se transformará en una tabla en el **RM**; las medidas, serán columnas de la tabla; la clave primaria estará compuesta por el conjunto de los atributos identificadores de los niveles hoja; los atributos que forman la clave primaria serán, además, claves foráneas que harán referencia a cada una de las tablas resultantes de las transformaciones de niveles hoja asociados al hecho.

Por ejemplo, el hecho *VENTA* (Figura 5.15, izquierda) en el **TMD,** se transformará en la tabla *VENTA* (Figura 5.15, derecha) en el **RM.**

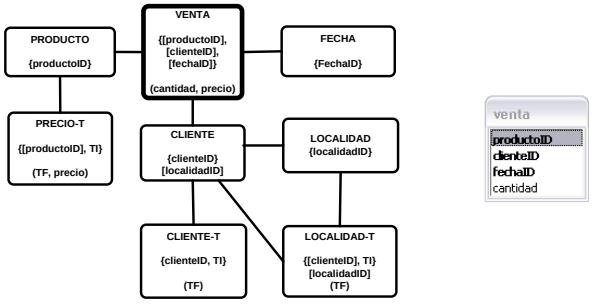


Figura 5.15. Transformación del Hecho en Tabla de Hecho

5.4.4.2. Nivel Hoja

Los niveles *hoja* se transformarán en tablas en el **RM**; los atributos serán columnas de la tabla; la clave primaria estará compuesta por el conjunto de los atributos identificadores de cada nivel; además, cada tabla *hoja* tendrá una clave foránea que hará referencia a cada una de las tablas que formen los niveles de las jerarquías

Por ejemplo, los niveles *hoja PRODUCTO, CLIENTE y FECHA* (Figura 5.16, izquierda) en el **TMD,** se transformarán, respectivamente, en las tablas *PRODUCTO, CLIENTE y FECHA* (Figura 5.16, derecha) en el **RM**

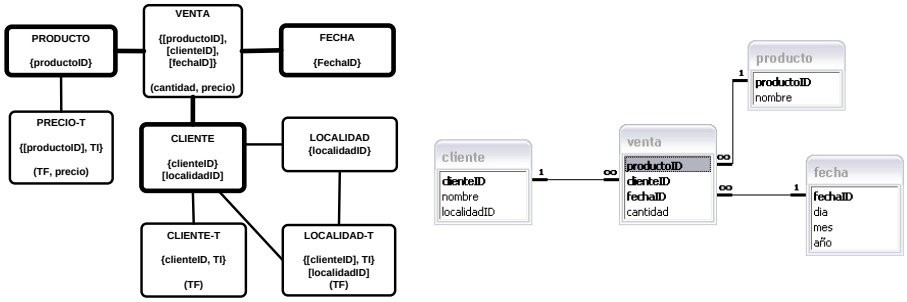


Figura 5.16. Transformación de Dimensiones como Tablas Dimensión

5.4.4.3. Niveles en la Jerarquía

Los niveles en las jerarquías en el **TMD** se transformarán en tablas en el **RM**; los atributos del nivel serán columnas de la tabla; la clave primaria estará compuesta por el conjunto de los atributos identificadores del nivel; además, cada tabla "*hijo*" tendrá una clave foránea que hará referencia a cada una de las tablas "*padre*" vinculadas.

Por ejemplo, el nivel *LOCALIDAD* (Figura 5.17, izquierda) en el **TMD,** se transformará en la tabla *LOCALIDAD* (Figura 5.17, derecha) en el **RM.**

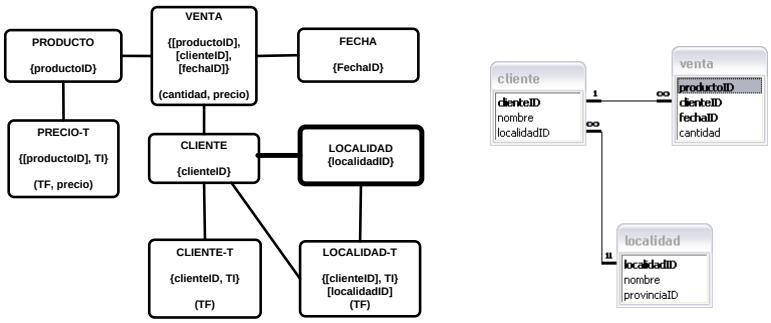


Figura 5.17. Transformación de Jerarquías en Tablas Jerarquía

5.4.4.4. Niveles Temporales

Los niveles de jerarquía temporal en el **TMD** se transformarán en tablas en el **RM**. Si este deviene de una entidad temporal, tendrá como atributo el TF; la clave primaria estará compuesta por la unión de la clave primaria de la tabla de la jerarquía vinculada (además, será la clave foránea que hará referencia a dicha tabla jerarquía) más el atributo TI. Si la jerarquía temporal deriva de un atributo temporal, tendrá como atributos el TF más el atributo que se precisa

conservar temporalmente; la clave primaria estará compuesta por la unión de la clave primaria de la tabla de la jerarquía vinculada (además, será la clave foránea que hará referencia a dicha tabla jerarquía) más el atributo TI. Si el nivel de jerarquía temporal deviene de una interrelación temporal, tendrá como atributo el TF y el atributo que represente a la clave primaria de una de las tablas jerarquías vinculadas (además, será la clave foránea que hará referencia a la tabla jerarquía); la clave primaria estará compuesta por la unión de la clave primaria de la otra tabla jerarquía vinculada (además, será la clave foránea que hará referencia a dicha tabla jerarquía) más el TI.

Por ejemplo, los niveles de jerarquía temporal *PRECIO-T, CLIENTE-T y LOCALIDAD-T* (Figura 5.18, izquierda) en el **TMD,** se transformarán, respectivamente, en las tablas *PRECIO-T, CLIENTE-T y LOCALIDAD-T* (Figura 5.18, derecha) en el **RM**.

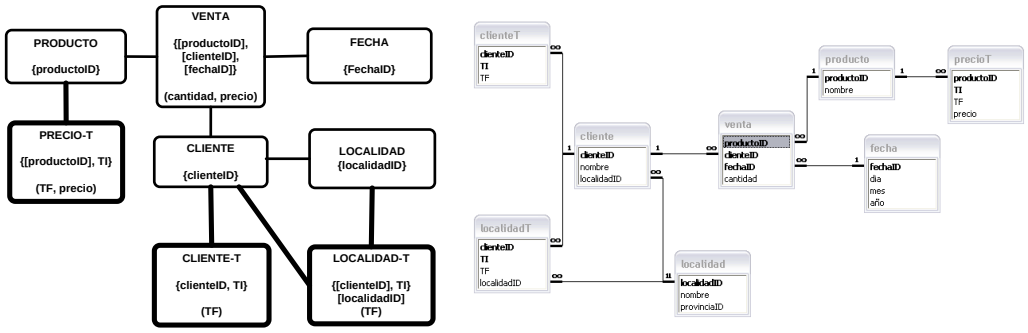


Figura 5.18. Transformación de Jerarquías Temporales en Tablas Jerarquías

5.4.5. Transf #6: Modelo Relacional a Tablas Mediante Sentencias SQL

La transformación del **RM** al modelo de datos expresado en sentencias **SQL**, es inmediata. Cada atributo tiene un tipo de datos, que deberá ser transformado (mediante una tabla de transformación definida) a los tipos de datos **SQL** existentes.

La sintaxis general del sublenguaje **DDL** (simplificado) para crear tablas mediante sentencias **SQL** es:

```
CREATE TABLE table_name ( [
column_name data_type  [ NOT NULL | NULL] [, ... ]
PRIMARY KEY (column_name_pk [, ...] )
FOREIGN KEY (column_name_fk [, ...] ) REFERENCES reftable
] )
```

Utilizando los mismos criterios de transformación del apartado anterior, la transformación es directa.

Por ejemplo, para la tabla *VENTA* (Figura 5.18, derecha), las sentencias **DDL** del **SQL** permitirán su implantación en un **RDBMS**:

```
CREATE TABLE VENTA (
        productoID numeric NOT NULL,
        clienteID numeric NOT NULL,
        fechaID date NOT NULL,
        cantidad numeric NOT NULL,
        precio numeric NOT NULL,
        PRIMARY KEY (productoID, clienteID, fechaID),
        FOREIGN KEY (clienteID) REFERENCES CLIENTE,
```

```
FOREIGN KEY (fechaID) REFERENCES FECHA,
FOREIGN KEY (productoID) REFERENCES PRODUCTO,
);
```

5.5. Resumen del Capítulo

El objetivo de este capítulo fue desarrollar el método de diseño de un **HDW** a partir de un modelo **ER.** Primero, se describió el modelo **MD** propuesto. Luego, se detallaron, de manera informal, todas las transformaciones: comenzando con un modelo **ER** donde el diseñador decidía que construcciones temporales preservar; se obtuvo, primero, un modelo de **TER**; luego, en base a aquel, un **AG**; a partir de éste y de decisiones del diseñado sobre qué aspectos **MD** considerar, un **TMD**; a continuación, un conjunto de tablas del **RM** y, por último, sentencias **SQL** que permiten implementarlo en un **RDBMS**. En el anexo IV detallaremos todas las transformaciones presentadas en este capítulo.

Capítulo 6

Recuperación de Información en Estructuras de Almacenamiento

6.1. Introducción

Con el objetivo de precisar las ventajas de nuestra propuesta, presentaremos en este capítulo un conjunto de estructuras de almacenamientos (ya establecidas en la comunidad informática) junto con sus limitaciones a la hora de resolver las necesidades de información del usuario. Luego, caracterizaremos a las principales consultas **MD**; seguidamente, detallaremos cómo resolver las consultas de toma de decisión sobre el modelo **TMD** desarrollado en el capítulo 5; a continuación, desarrollaremos las consultas temporales en el modelo de datos propuesto en el capítulo 3. Por último, estableceremos la forma de implementar las consultas mediante sentencias **SQL.**

6.2. Sistemas de Información Automatizados

Un sistema de información automatizado utiliza estructuras de datos para el almacenamiento digital y lenguajes de consulta para la recuperación de la información[24]. Analizaremos, a partir de un ejemplo, diversas consultas y cómo los diferentes tipos de almacenamiento de datos admiten, o no, resolverlas en forma eficiente.

Clasificaremos a las consultas en ***tipos A, B, C*** y ***D*** según correspondan, respectivamente, a usos operacionales (***A***), temporales (***B***), de toma de decisión (**C**) o de toma de decisión temporal (**D**), ésta última como combinación de consultas **tipo B** y **C.**

Caractericemos el sistema ejemplo: "*La empresa fabrica y centraliza sus operaciones comerciales en Buenos Aires, pero sus clientes están distribuidos*

[24] Utilizaremos "*dato*" para describir a los símbolos que describen objetos de la realidad e "*información*" para referirnos a esos datos analizados por el usuario que le permitan tomar decisiones.

geográficamente en todo el país. Realiza ventas de productos a partir de pedidos de clientes que están ubicados en distintas provincias, registra las transacciones realizadas, las fechas y cantidades vendidas".

Las posibles consultas, clasificadas por tipo y ejemplificadas, se detallan a continuación:

- ***tipo A***. Son consultas realizadas en un sistema **OLTP**; por ejemplo: ¿Qué productos y cantidades se vendieron a un cliente dado en una fecha determinada? ¿Cuál es el precio actual de un producto?, ¿Cuál es la localidad actual donde está ubicado un cliente determinado?, etcétera.
- ***tipo B***. Son consultas realizadas en un **TDB**; por ejemplo: ¿En qué fechas varió y cuál fue el precio de un producto específico, en el transcurso del año 2007? ¿Cuál fue el precio de un producto determinado el 18 de marzo de 2004? ¿En qué fechas de 2006 se mudo un cliente específico y a dónde? ¿Cuál fue la ubicación de un cliente determinado el 20 de octubre de 2004?, ¿En qué periodos un cliente específico estuvo activo como tal?, etcétera.
- ***tipo C***. Son consultas realizadas en un **DSS**; por ejemplo, ¿Cuál fue el promedio de las ventas de productos, por cliente y por localidad, en el año 2008?, ¿Cuál fue la cantidad de ventas, por tipo de cliente y por provincia, en el 2º semestre del año 2007?, etcétera.
- ***tipo D***. Son consultas implementadas en un ***Temporal Decision Support System* (TDSS)** en él pueden realizarse consultas de ***tipo B***, ***tipo C*** o, incluso, combinación de ambas (**B**+**C**); por ejemplo, ¿Cuál fue la incidencia de la variación del precio de un producto específico en las ventas anuales a los clientes?, ¿Cómo variaron las ventas a los clientes en función de los cambios de dirección de los mismos en un periodo dado?, etcétera.

6.3. Almacenamiento de Datos

Analizaremos, a continuación, distintos tipos de almacenamiento de datos y sus ámbitos de utilización: **ODB**, **TDB**, **DW, TDW** y, por último, **HDW**, nuestra propuesta de estructura de almacenamiento **TMD**.

6.3.1. Base de Datos Operacionales

Los **ODB´s** surgieron ante la necesidad de almacenar y organizar los datos de las empresas, producto de sus transacciones diarias; las estructuras de datos se diseñan con el objetivo de evitar redundancias en los datos almacenados y, por consiguiente, posibles inconsistencias en la información, debido a los procesos de actualización. Las operaciones típicas en los sistemas **OLTP** automatizan el procesamiento de datos (por ejemplo, transacciones bancarias, ventas de productos, etc.); en este tipo de **DBMS**, las actividades son rutinarias y repetitivas y, generalmente, consisten en transacciones aisladas, atómicas[25] y breves [CD97]. Las consultas que se realizan son relativamente simples y, generalmente, están predefinidas y testeadas con el objetivo de obtener la respuesta adecuada a las necesidades de la empresa. Un **ODB** se diseña para reflejar la semántica de las operaciones de aplicaciones ya conocidas, evitando

[25] Una transacción es una secuencia de modificaciones y consultas tratada como una operación atómica simple.

anomalías de actualización[26] mediante estructuras de datos normalizadas en un **RDBMS**.

Un **ODB** permitirá resolver consultas de ***tipo A***. Las consultas de ***tipo B***, no, debido a que las actualizaciones eliminan los datos anteriores y pierden la historia de las mismas. Por otro lado, las consultas de ***tipo C***, si bien se podrían resolver[27], si la cantidad de datos almacenados fuere considerable, requerirían para su ejecución un excesivo tiempo de procesamiento, pudiendo, incluso, entorpecer el funcionamiento normal del sistema **OLTP.** Como consecuencia de lo descrito anteriormente, tampoco resolverá las consultas de ***tipo D.***

6.3.2. Bases de Datos Temporales

Un **ODB** se diseña con el objetivo de representar una visión particular de la información sobre el mundo real, esa información tiene validez en un lapso determinado, no siempre explicitado; por ejemplo, si el valor de un atributo se modificara, ese hecho deberá ser reflejado en la base de datos mediante actualizaciones o borrados, según sea la naturaleza del cambio. Por lo tanto, debido a esa modificación, el estado previo de la base de datos se perderá.

Una limitación de un **ODB** es que cada actualización destruye datos antiguos. Capturar la variación temporal de los datos se torna necesario en aplicaciones que requieren preservar los cambios, por tal motivo, las estructuras de datos de las **TDB** se diseñan con el objetivo de permitir conservar en el tiempo la historia del cambio de los valores de los datos [EN97]

Podemos considerar a las **TDB´s** como un súper conjunto de los **ODB´s**, por lo tanto, permitirán resolver las consultas de ***tipo A*** y ***tipo B***, respecto de las consultas de ***tipo C,*** las consideraciones son similares a las expuestas para un **ODB** y por consiguiente, tampoco resolverá las consultas de ***tipo D.***

6.3.3. Data Warehouse

Los usuarios de los **DSS** están más interesados en identificar tendencias que en buscar algún registro individual en forma aislada [HRU96]. Con ese propósito, los datos de las diferentes transacciones de los **OLTP** se guardan y consolidan en una base de datos central denominada **DW** que los analistas utilizan para extraer información de sus negocios que les permita tomar mejores decisiones [GHRU97].

Si un **ODB** mantiene información actual, el **DW** mantiene información histórica de la organización y éste se diseña para asistir en el proceso de toma de decisiones, no suple a los **ODB´s** ni a las **TDB´s**, los complementa. Los datos históricos, resumidos y consolidados, son más importantes que los detalles y los registros individuales. El procesamiento analítico incluye consultas muy complejas (frecuentemente con funciones de agregado) y poco o nada de actualizaciones.

Un **DW** permitirá resolver consultas de **tipo C**. Por carecer de estructuras temporales explicitas, no será adecuado para responder a consultas de ***tipo B y D.*** Tampoco será adecuado para resolver las consultas de ***tipo A.***

[26] Problemas de inconsistencia que pueden ocasionar estructuras de datos diseñadas ineficientemente, cuando se realizan sobre ellas procesos de inserción, actualización o borrado de datos.

[27] Las consultas de *tipo C* normalmente no se realizan en un sistema **OLTP** sino, en forma eficiente, en un **DSS** basado en un **DW**.

6.3.4. Data Warehouse Temporal

Un **DW** podría admitir dos tipos de modificaciones: en la estructura del esquema de almacenamiento y en los valores de las instancias (o en ambas). Los sistemas **OLAP,** permiten analizar los cambios en los valores de los datos a través del tiempo, pero no están diseñados para aceptar modificaciones en las dimensiones ya que éstas se consideran (implícitamente) ortogonales. Esto es debido a que los datos reflejan la parte dinámica del **DW** mientras que las dimensiones son relativamente estáticas [EKK02].

El **DW** está diseñado para reflejar las modificaciones de los datos transaccionales. El **TDW** contempla, además, modificaciones que puede sufrir el **DW** en el esquema, por ejemplo, añadir una nueva dimensión o modificar una jerarquía para establecer un nuevo nivel de agrupamiento.

El **TDW**, tiene como objetivo permitir resolver consultas del **tipo C** y, en particular, solucionar problemas vinculados con cambios en el esquema de almacenamiento. No facilita las consultas de **tipo B** y **D.** Tampoco es adecuado para resolver las consultas de ***tipo A.***

6.3.5. Data Warehouse Histórico

Los **ODB´s** son la fuente de datos del **DW**, aquellos representan sólo el estado actual de los datos y no su historia. Por lo tanto, si bien en el **DW** el tiempo está implícito en su estructura y se utiliza en el análisis de datos, éste hace referencia al momento en que se realizó una transacción, no explícita cómo variaron los datos involucradas en ellas. Por consiguiente, aunque la información estuviere almacenada, los mecanismos de búsqueda temporales resultarían complejos [NA02]. La necesidad de evaluar tendencias, variaciones, máximos y mínimos en lapsos determinados, información que se obtiene de una **HDB**; además de realizar las consultas típicas de un **DW**, justifican considerar en el diseño, y representar en forma explícita, cómo los datos pueden variar en el tiempo. El **HDW** es una propuesta que combina, en un modelo integrado, **HDB** y un **DW.**

Un **HDW** permite resolver consultas de **tipo D.** No es adecuado para resolver consultas de **tipo A**, pero si, al menos parcialmente, consultas de **tipo B.**

6.4. El Lenguaje de Consulta Estructurado

SQL [SQL] es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características distintivas es el manejo del álgebra y el cálculo relacional, que permite realizar consultas con el fin de recuperar, de una forma sencilla, información de interés de una base de datos, así como también hacer cambios sobre la misma.

Los orígenes de **SQL** están ligados a los de las bases de datos relacionales. En 1970 E. F. Codd propone el modelo relacional y asociado a éste un sublenguaje de acceso a los datos basado en el cálculo de predicados. **SQL** es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos. Es un lenguaje declarativo de "alto nivel" o "no procedimental" orientado al manejo de conjuntos de registros. Es declarativo por cuanto especifica qué es lo que se quiere y no como conseguirlo.

SQL permite definir la estructura de los datos, recuperar y manipular datos, administrar y controlar el acceso a los datos, compartir datos de forma concurrente y asegurar su integridad. Además, los comandos del lenguaje

pueden ser ejecutados interactivamente o bien formando parte de un programa de aplicación, donde las sentencias **SQL** están insertas dentro del código fuente y mezcladas con las sentencias del código.

6.4.1. Versiones de SQL

Desde su aparición, se han presentado diferentes versiones:

- **SQL-86**: en el año 1986 surge la primera estandarización por el ***American Nacional Standards Institute*** **(ANSI)**, también llamada **SQL**1. Al año siguiente, este estándar es adoptado por la ***International Standards Organization*** **(ISO)**. Esta versión no incluía integridad referencial.
- **SQL-89**: está basado en el anterior **SQL**-86, pero incorpora la integridad referencial (no actualizaciones en cascada), definición de claves primarias, etc. y puede usarse a través de dos interfaces: interactivamente o dentro de programas de aplicación.
  En su primera versión, **SQL**-89 tiene tres partes:

    - ***Data Description Language*** **(DDL)**: lenguaje de definición de datos. Permite definir, modificar o borrar las tablas en las que se almacenan los datos y las relaciones entre éstas.
    - ***Data Control Language*** **(DCL)**: lenguaje de control de datos. Permite trabajar en un entorno multiusuario, donde es importante la protección y seguridad de los datos y la compartición de los mismos.
    - ***Data Manipulation Language*** **(DML**): lenguaje de manipulación de datos. Permite recuperar datos almacenados en la base de datos así como su actualización tanto modificación como eliminación.

- **SQL-92**: Incorpora nuevos operadores relacionales (*OUTER JOIN y JOIN*), **SQL** dinámico, gestión de errores, cursores de desplazamiento (*scroll cursor*), modo de acceso (lectura o lectura/escritura) y nivel de aislamiento de las transacciones y la definición de dominios. No es un lenguaje fuertemente tipado.
- **SQL:1999** (parte del estándar **SQL**3): es un lenguaje fuertemente tipado. Soporta la tecnología objeto relacional. Sus características más relevantes son: nuevos tipos de datos, posibilidad de definir nuevos tipos de datos por parte del usuario, disparadores o *triggers*, vistas actualizables, cursores sensitivos. *Queries* recursivos, definición de roles de usuario e Incorporación de las características de orientación a objetos: tipos de datos abstractos, generalización, herencia y polimorfismo.
- **SQL:2003**: intento de transformación en un lenguaje *stand-alone* y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimediales. Está caracterizado como "**SQL** orientado a objetos" y es la base de algunos sistemas de manejo de bases de datos orientadas a objetos.
  El borrador de **SQL3** está dividido en partes:

    - **Primera Parte SQL**/Estructura: se ofrecen definiciones básicas y se explica la estructura de la especificación **SQL**3.
    - **Segunda Parte SQL**/Fundamentos: incluye las principales novedades de **SQL**3. Comprende disparadores (*triggers*), funciones, consultas recurrentes, colecciones y **SQL** para objetos, incluidos los tipos de datos abstractos definidos por el usuario.

- **Tercera Parte SQL**/CLI: se define la interface del nivel de llamada. Es una ampliación de la propuesta de **SQL** *Access Group* y de gran aceptación en el medio.
- **Cuarta Parte SQL**/PSM: define los módulos de almacenamiento persistente de **SQL**, que son los procedimientos almacenados y las extensiones de lenguaje de procedimientos de **SQL**.
- **Quinta Parte SQL**/Acoplamientos: se refiera a los mecanismos de combinación de **SQL** con otros lenguajes a través de precompiladotes y **SQL** incrustado.
- **Sexta Parte SQL**/Transacciones: especifica la forma en que las bases de datos **SQL** participan en transacciones globales.
- **Séptima Parte SQL**/Temporal: trata el modo de empleo de datos de series de tiempo por las bases de datos **SQL**, lo que permitirá hacer consultas con el tiempo como variable.

- **SQL:2006**: define las formas en las cuales el **SQL** se puede utilizar conjuntamente con ***Extensible Markup Language* (XML)**. Define maneras de importar y guardar **XML** en una base de datos **SQL**, manipulándolos dentro de la base de datos y publicando el **XML** y los datos **SQL** convencionales en forma **XML**. Además proporciona facilidades que permiten a las aplicaciones integrar dentro de su código **SQL** el uso de XQuery, lenguaje de consulta **XML** para acceso concurrente a datos ordinarios **SQL** y documentos **XML**.
- **SQL/MM**: parte del **SQL**:1999, estandarizado en mayo del 2001 por el **ISO** introduce tipos de objetos y métodos asociados para: texto libre (full text), datos espaciales e imágenes. Para imágenes, provee tipos de objetos estructurados y métodos para almacenar, manipular y representar datos de imágenes por contenido, a nivel físico.

6.5. Consultas Multidimensionales

Una característica distintiva de las herramientas **OLAP** es su acento sobre la agrupación de medidas considerando una o más dimensiones, La visión **MD** permite analizar a los distintos niveles de jerarquías en cada dimensión de una manera lógica.

Aunque no hay consenso sobre un conjunto mínimo de operaciones sobre el modelo **MD**, la mayoría de los trabajos de investigación presentan dos grandes grupos de consultas [RTT08]:

- ***Drilling***: estas operaciones permiten navegar a través de los distintos niveles de jerarquías de las dimensiones, con el objetivo de analizar una medida con mayor o menor nivel de detalle. Cuando la consulta navega de mayor a menor nivel de detalle en los datos, la operación se denomina *ROLL-UP*, la navegación inversa es *DRILL-DOWN*.
- ***Selections:*** estas operaciones permiten al usuario trabajar con un subconjunto de los datos disponibles. *SLICE* especifica restricciones en los datos de las dimensiones y *DICE* establece restricciones en los datos del hecho.

Las estructuras **MD** son considerados, en general, en forma aislada, sin embargo cuando son utilizados para realizar consultas puede ser necesario navegar de una estructura a otra *(drill-across)*. Esto significa que se pueden analizar un cubo desde un punto de vista y querer luego ver datos, en otra estructura, desde otro punto de vista; para que esto sea posible, las estructuras

MD necesitan tener puntos de equivalencias, esto es, dimensiones en común [AAS01]. En [ASS03] se analiza la posibilidad de establecer relaciones entre diferentes esquemas, sin imponer la restricción de total coincidencia en las dimensiones.

Se han elaborado diversos trabajos relacionados a las consultas **MD**. En [RA05] se definen y clasifican los problemas que surgen cuando se traducen automáticamente consultas **MD** a sentencias **SQL**, donde, además, se analiza cómo resolverlos o minimizar su impacto. En [VM01] proponen un modelo **TMD** que contempla actualización de dimensiones y un lenguaje de consulta, *TOLAP*, que permite soportar actualizaciones de dimensiones y evolución del esquema **MD** a un alto nivel de abstracción.

6.5.1. Consultas de Agrupamiento

El propósito, tanto de los sistemas **OLAP** como también de las ***Statistical Data Base*** **(SDB)** es proveer un marco para la realización de operaciones de sumarización sobre bases de datos **MD**. Por lo tanto, un aspecto importante a considerar en el diseño de un **DW** está referido a la aplicación de la operación "suma" en los atributos de hecho. La violación de esta propiedad puede conducir a conclusiones y decisiones erróneas.

Conocer cuáles son los atributos no sumarizables se torna imprescindible, ya que esta operación no puede aplicarse a ellos (pero sí otras, por ejemplo promedios, máximos o mínimos) debido a que las reglas semánticas pueden no ser bien conocidas y el resultado, si bien puede ser establecido (es decir, se puede obtener un valor), éste no será correcto. Por ejemplo, el valor cantidad de ítems vendidos por mes en un supermercado es un atributo sumarizable, esto es, se pueden sumar esos valores y determinar, mensualmente, cuantos ítems se vendieron. Por otro lado, el atributo cantidad de ítems que permanecen sin vender en la góndola del supermercado no es sumarizable, ya que esa suma puede repetir ítems de meses anteriores y, por lo tanto, el valor resultante carece de sentido práctico [LS97].

En el análisis, tanto en las **SDB** como en **DSS**, es útil la agregación de instancias de hecho a diferentes niveles de abstracción (*roll up*), por lo tanto, la mayoría de los atributos de hecho deberían ser aditivos, esto significa que el operador suma puede ser usado para agrupar valores de atributos a lo largo de todas las jerarquías. Un atributo se denomina semi-aditivo si no es aditivo en una o más dimensiones y no-aditivo, si no lo es en ninguna dimensión [GMR98a]. En [LAW98], se proponen dos formas normales multidimensionales que establecen restricciones para la sumarización de atributos y restricciones al modelo **MD**. En [LS97] se establecen condiciones de sumarización en **SDB** y sistemas **OLAP.**

6.5.1.1. Consultas en el Modelo Multidimensional

Analizaremos, a continuación, las operaciones más importantes sobre el modelo **MD** establecidas en el apartado anterior. Utilizaremos para ejemplificar el sistema presentado al inicio del capítulo: "*La empresa fabrica y centraliza sus operaciones comerciales en Buenos Aires, pero sus clientes están distribuidos geográficamente en todo el país. Realiza ventas de productos a partir de pedidos de clientes que están ubicados en distintas provincias, registra las transacciones realizadas, las fechas y cantidades vendidas*".

El modelo **MD** de la Figura 6.1, ofrece una primera aproximación del sistema ejemplo. En el mismo estableceremos los siguientes niveles de jerarquía (no explicitadas en el gráfico). En la dimensión producto: "*producto → tipo de producto → departamento → región*". En la dimensión fecha: *fecha → mes → trimestre → año* y en la dimensión cliente: "*cliente → categoría*"

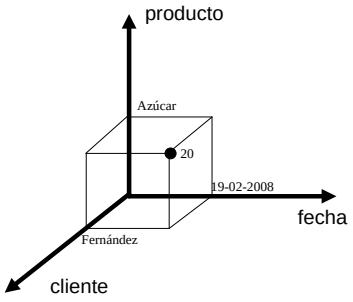


Figura 6.1. Modelo Multidimensional

6.5.1.1.1. Roll-Up

La operación *roll-up*, permite reducir el nivel de detalle en el que se consultan los datos, realizando operaciones de agregación (sumas, promedios, etc.) a través de los niveles de jerarquías de las dimensiones. Calcula las medidas en función de agrupamientos, es decir, realiza el re-cálculo de la medida de acuerdo a los ajustes de escala.

Por ejemplo, en la Tabla 6.1 observamos una representación tabular de las ventas por producto, cliente y fecha, donde adicionamos una columna que explicita el nivel de jerarquía "*tipo de producto*". De acuerdo a los valores de la tabla, las ventas totales por producto son: (azúcar, 60); (arroz, 30); (leche, 10) y (yogurt, 12). (Tabla 6.3)

Tipo de Producto	Producto	Cliente	Fecha	Cantidad
no perecedero	Azúcar	Fernández	19-02-2008	20
no perecedero	Azúcar	Rodríguez	19-02-2008	40
no perecedero	Arroz	Fernández	20-02-2008	30
lácteos	Leche	Pérez	20-02-2008	10
lácteos	yogurt	Pérez	23/12/2008	12

Tabla 6.1. Representación Tabular de Datos

Si ahora agrupamos por "*tipo de producto*", las ventas pueden expresarse, mediante la operación *roll-up*, como (productos no perecedero, 90) y (productos lácteos, 22). (Tabla 6.2)

Tipo de Producto	Cantidad
no perecedero	90
lácteos	22

Tabla 6.2. Resultado de la Operación *Roll-Up*

6.5.1.1.2. Drill-Down

Esta operación es la inversa de la operación *roll-up*, es decir permite aumentar el detalle al que se consultan los datos, al ir a un nivel más bajo dentro de la jerarquía.

Siguiendo con el ejemplo anterior, si tuviéramos las ventas por "tipo de producto" y quisiéramos desagregarlas por producto, el resultado ahora sería: (azúcar, 60); (arroz, 30); (leche, 10) y (yogurt, 12) (Tabla 6.3).

Producto	Cantidad
Azúcar	60
Arroz	30
Leche	10
yogurt	12

Tabla 6.3. Resultado de la Operación d*rill-down*

6.5.1.1.3. Slice y Dice[28]

Estos operadores permiten reducir el conjunto de datos consultados por medio de operaciones de proyección y selección de los datos, basándose en los atributos de las dimensiones y hechos. La operación *slice*, consiste en restringir los valores de una o más dimensiones a un valor o a un rango de valores.

Por ejemplo, en la Tabla 6.1, si fijásemos la dimensión cliente en un valor (por ejemplo, Fernández), reducimos el número de dimensiones consideradas; la representación resultante se muestra en la tabla 6.4.

Producto	Fecha	Cantidad
Azúcar	19-02-2008	20
Arroz	20-02-2008	30

Tabla 6.4. Resultado de la Operación *Slice*

La operación *dice*, establece restricciones en los datos del hecho.

Siguiendo con el ejemplo (Tabla 6.1), si ahora restringiéramos la consulta para los productos cuyas ventas fueran superiores a 15 unidades, la representación resultante se muestra en la Tabla 6.5.

Producto	Cliente	Fecha	Cantidad
Azúcar	Fernández	19-02-2008	20
Azúcar	Rodríguez	19-02-2008	40
Arroz	Fernández	20-02-2008	30

Tabla 6.5. Resultado de la Operación Dice

6.6. Consultas Multidimensionales

Las operaciones **MD** descritas anteriormente pueden implementarse sobre un esquema *estrella*, utilizando sentencias **SQL**, mediante la generalización de la siguiente consulta adaptada de [Kim96].

```
SELECT nivel-ID1,…, nivel-IDn , Funcion(H.medida)
FROM Hecho H, Dim1 d1,…, Dimn dn
WHERE H.pk1 = D1.ID AND,…, H.pkn = Dn.ID
AND d1.attr =  valor1 AND
GROUP BY nivel-ID1,…, nivel-IDn
HAVING Funcion(H.medida)  oper valor2
ORDER BY nivel-ID1,…, nivel-IDn
```

[28] Existen ciertas diferencias en la definición de estos operadores, por lo tanto, nos remitiremos a las establecidas en este capítulo.

La cláusula *FROM* contiene la tabla hecho y las tablas dimensiones, estas tablas están vinculadas mediante la cláusula *WHERE* que también define las condiciones sobre las columnas de las tablas dimensión. La cláusula *GROUP BY* muestra los identificadores de los niveles sobre los cuales agrupamos los datos, esos mismos identificadores aparecen en la cláusula *SELECT*, previos a la función de agregado. La cláusula *HAVING* establece restricciones sobre los valores de las medidas. Por último, la cláusula *ORDER BY* explicita el orden de salida de la consulta.

6.6.1. Implementación de Consultas Multidimensionales en SQL

Proponiendo un esquema estrella que represente la estructura de datos de la Tabla 6.1 (Figura 6.2):

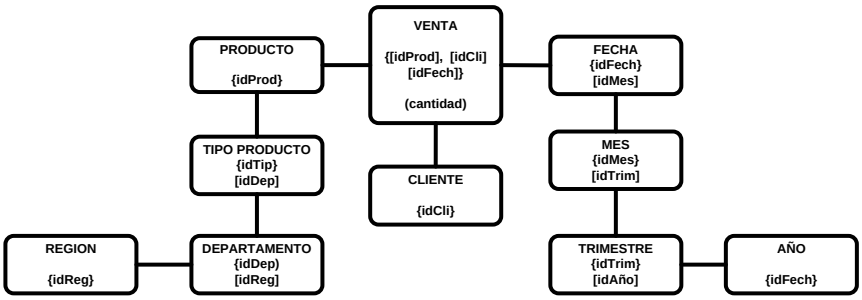


Figura 6.2. Esquema Estrella

Las tablas resultantes en una implementación pseudorelacional serán:

PRODUCTO(idProd, idTip(TIPO-PRODUCTO), producto, …)
FECHA(idFech, idMes(MES),fecha, …)
CLIENTE(idCli, cliente, …)
VENTA(idProd(PRODUCTO), idFech(FECHA), idCli(CLIENTE), cantidad)
TIPO-PRODUCTO(idTip, idDep(DEPARTAMENTO), tipo-de-producto, …)
DEPARTAMENTO(idDep, idReg(REGION), …)
REGION(idReg,…)
MES(idMes, idTri(TRIMESTRE), …)
TRIMESTRE(idTri, idAño(AÑO), …)
AÑO(idAño, …)

Presentamos, a continuación, utilizando el sistema ejemplo mostrado anteriormente, un conjunto de consultas **SQL** que permitirán implementar los operadores **MD** *roll-up, drill-down, slice* y *dice*.

Por ejemplo, utilizando sentencias **SQL**, la consulta "determinar la cantidad ventas, por tipo de producto", usando el ejemplo del esquema anterior, será

```
SELECT tipo-de-producto, SUM(cantidad)
FROM VENTA v, PRODUCTO p, TIPO-PRODUCTO tp
WHERE v.idProd = p.idProd
AND p.idProd = tp.Tip
GROUP BY tipo-de-producto;
```

Si precisáramos realizar la consulta con mayor nivel de detalle (*drill-down*), por ejemplo, "determinar la cantidad ventas, por producto", la sentencia **SQL** será:

```
SELECT producto, SUM(cantidad)
FROM VENTA v, PRODUCTO p, TIPO-PRODUCTO tp
```

```
WHERE v.idProd = p.idProd
AND p.idProd = tp.Tip
GROUP BY producto;
```

Si precisáramos un menor nivel de detalle, estaríamos realizando la operación *roll-up* (volveríamos a la consulta anterior).

Por otro lado, la consulta "determinar la cantidad de ventas de productos por cliente, por producto y por fecha" para el cliente "Fernández" (Tabla 6.1), que, en términos **MD** corresponde a la operación *slice*, queda expresada en sentencias **SQL** de la siguiente manera:

```
SELECT cliente, producto, fecha, SUM(cantidad)
FROM VENTA v, PRODUCTO p, CLIENTE c, FECHA f
WHERE v.idProd = p.idProd
AND v.idProd = c.idCli
AND v.idProd = f.idFech
AND p.cli = "Fernández"
GROUP BY cliente, producto, fecha
ORDER BY cliente, producto, fecha;
```

Por último, La consulta "determinar la cantidad de ventas de productos por cliente, producto y por fecha cuyas ventas sean superior a 20 unidades", que en términos **MD**, corresponde a la operación *dice*, queda expresada en sentencias **SQL**, de la siguiente manera:

```
SELECT cliente, producto, fecha, SUM(cantidad)
FROM VENTA v, PRODUCTO p, CLIENTE c, FECHA f
WHERE v.idProd = p.idProd
AND v.idProd = c.idCli
AND v.idProd = f.idFech
GROUP BY cliente, producto, fecha
HAVING SUM(cantidad) > 20
ORDER BY cliente, producto, fecha
```

6.7. Consultas Temporales

Muchas aplicaciones precisan registrar la evolución de los datos que varían con el tiempo; por ejemplo, las historias clínicas en hospitales, el registro temporal del mantenimiento de equipos, el periodo de validez de leyes, etc. Los actuales **DBMS** y **SQL** en particular proveen poco soporte para administrar datos variables en el tiempo, solo proveen tipos de datos estándar para codificar datos o marcas de tiempo.

A los comienzos de los años ´90 los investigadores en temas vinculados a **TDB** realizaron un considerable esfuerzo dirigido a consensuar un modelo temporal relacional y su correspondiente lenguaje de consulta [CMP07]. Ese esfuerzo concluyo en la propuesta TSQL2 [Sno95]; este lenguaje de consultas para **TDB**, derivado de **SQL**, soporta tanto tiempo válido como tiempo de transacción. Los aspectos temporales también han sido considerados en el desarrollo de las más reciente versión **ISO** del estancar **SQL**, el nuevo componente denominado **SQL**/Temporal. Los componentes temporales extienden el **RM** básico a partir de la incorporación del tipo de dato *PERIOD* y de dos dimensiones temporales *TT* y *VT*, además de la posibilidad de realizar consultas temporales [TBJS99]. **SQL**3 es totalmente compatible con las versiones anteriores de **SQL**, de modo tal que la migración de versiones no temporales es

simple y eficiente y el código existente puede ser reutilizado sin intervenciones adicionales [CMP07].

Otras propuestas más recientes han surgido, en [CKZ03] se presentó el diseño e implementación de un prototipo que soporta extensiones temporales de **SQL.** En [Zim06] se presentó un **TDB** y muestra cómo realizar *joins* temporales y proyecciones temporales en un **SQL** estándar. En [AS08] propone un modelo relacional temporal basado en cronos para **TDB** donde utiliza tuplas marcadas temporalmente.

6.7.1. Implementación de Consultas Temporales en SQL

El nivel de complejidad y variedad de las consultas temporales no permite un análisis simplificado como el realizado en el apartado anterior. Por tal motivo, retomaremos el conjunto de consultas características que propusimos en el capítulo 3 y lo ejemplificaremos para detallarlo. El modelo temporal propuesto (**HDB**) en el capítulo 3, permite registrar el tiempo de vida (*lifespan*) de las entidades y el tiempo válido (*valid time*) tanto de atributos como de interrelaciones.

Utilizaremos, para ejemplificar las consultas, el modelo conceptual (capítulo 3, Figura 3.7), donde consideramos aspectos temporales, en particular, la entidad temporal *CLIENTE-T*, el atributo temporal representada mediante la entidad *PRECIO-T* y la interrelación temporal simbolizada mediante la entidad *LOCALIDAD-T* (Figura 6.3).

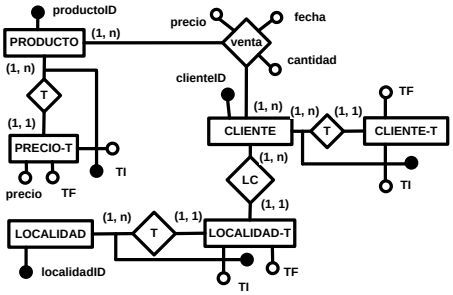


Figura 6.3. Modelo Conceptual Temporal

Su implementación pseudorelacional se detalla a continuación:

```
VENTA(productoID(PRODUCTO), clienteID(CLIENTE), fecha, cantidad,
precio)
PRODUCTO(productoID, …)
CLIENTE(clienteID,…)
CLIENTE-T({clienteID(CLIENTE),TI}, TF)
LOCALIDAD(localidadID,…)
PRECIO-T(productoID(PRODUCTO), TI, TF, precio)
LOCALIDAD-T({clienteID(CLIENTE-T), TI}, TF,
localidadID(LOCALIDAD))
```

Resumimos a continuación un conjunto de consultas que podrán realizarse sobre la estructura de almacenamiento temporal: para las entidades temporales, se podrá obtener su tiempo de existencia (tiempo de vida); para los atributos temporales, la determinación de en qué fechas se modificó su valor, además de determinar sus diferentes valores en diversos momentos (tiempo válido) y, por último, para las interrelaciones temporales, se podrá establecer en qué fechas se modificó el vinculo entre dos entidades y, para un instante particular, cual fue el vínculo entre ellas (tiempo válido).

Detallamos a continuación, mediante sentencias **SQL**, la resolución de las consultas descritas anteriormente.

6.7.1.1. Entidades Temporales

Para las entidades temporales se podrá obtener su tiempo de vida, esto es, los diversos rangos temporales en el cual esa entidad estuvo activa en el universo de discurso.

Por ejemplo, la consulta ¿En qué intervalos un cliente en particular estuvo activo?, se podrá resolver con la siguiente sentencia **SQL.**

```
PARAMETERS p text;
SELECT c.nombre, ct.ti, ct.tf
FROM CLIENTE AS c, CLIENTE-T AS ct
WHERE c.clienteid = ct.clienteid
AND c.clienteid = p
ORDER BY c.nombre, ct.ti, ct.tf;
```

6.7.1.2. Atributos Temporales

Para los atributos temporales se podrá obtener, a) en qué momentos se modificó su valor y cuál fue el mismo.

Por ejemplo, la consulta ¿En qué fechas varió y cuál fue el precio de un producto determinado?, se podrá resolver con la siguiente sentencia **SQL.**

```
PARAMETERS p text;
SELECT pt.precio, pt.ti, pt.tf
FROM PRODUCTO AS p , PRECIO-T AS pt
WHERE p.productoid = pt.productoid
AND p.productoid = p
ORDER BY pt.precio, pt.ti, pt.tf;
```

y, b) para una fecha determinada, cuál fue el valor de ese atributo.

Por ejemplo, la consulta ¿Cuál fue el precio de un producto particular en una fecha determinada? se podrá resolver con la siguiente sentencia **SQL.**

```
PARAMETERS t datetime, p text;
SELECT pt.precio
FROM PRODUCTO AS p, PRECIO-T AS pt
WHERE p.productoid = pt.productoid
AND p.productoid = p
AND ti <= t
AND tf >= t;
```

6.7.1.3. Interrelaciones Temporales

Para las interrelaciones temporales se podrá determinar, a) en qué fechas se modificó el vínculo entre dos entidades.

Por ejemplo, la consulta ¿En qué fechas se mudo un cliente en particular y a dónde?, se podrá resolver con la siguiente sentencia **SQL.**

```
PARAMETERS p text;
SELECT l.nombre, ti, tf
FROM LOCALIDAD AS l, LOCALIDAD-T AS lt, CLIENTE AS c
```

```
WHERE l.localidadid = lt.localidadid
AND lt.clienteid = c.clienteid
AND c.clienteid = p
ORDER BY l.nombre, ti, tf;
```

y, b) para un instante en particular, cuál fue el vínculo entre ellas.

Por ejemplo, la consulta ¿Cuál fue la ubicación de un cliente en particular en una fecha determinada?, se podrá resolver con la siguiente sentencia **SQL**:

```
PARAMETERS t datetime, p text;
SELECT l.nombre
FROM LOCALIDAD AS l, LOCALIDAD-T AS lt, CLIENTE AS c
WHERE l.localidadid = lt.localidadid
AND lt.clienteid = c.clienteid
AND c.clienteid = p
AND TI <= t
AND TF >= t;
```

6.8. Resumen del Capítulo

El objetivo de este capítulo fue presentar, primeramente, un conjunto de estructuras de almacenamientos (ya establecidas en la comunidad informática) junto con sus limitaciones a la hora de resolver las necesidades de información del usuario, Luego, con el objetivo de resolver aquellas y a partir del modelo temporal y **MD** propuestos, se detallaron las principales consultas temporales y de toma de decisión que pueden realizarse sobre dichos modelos. Por último, se describió cómo implementar las consultas temporales y **MD** mediante sentencias **SQL**.

Capítulo 7

Consultas en un Data Warehouse Histórico

7.1. Introducción

Las herramientas **OLAP** permiten la agrupación de medidas mediante la elección de una o más dimensiones y, por medio de una visión **MD,** posibilitan analizar los distintos niveles de las jerarquías en cada dimensión de una manera lógica que permite al usuario una aproximación más intuitiva. Por otro lado, las **TDB**, permiten recuperar información histórica. La forma tradicional de acceder a una base de datos ha sido vía **SQL**; debido a la complejidad en la formulación de consultas en **SQL** se han propuesta diversos enfoques para hacer más accesible a un espectro mayor de usuarios mediante el uso de lenguajes gráficos.

En este capítulo se presentará una interface gráfica de consultas, derivada del modelo **TMD** y describiremos en detalle, de manera informal, cómo derivar, primeramente la interface gráfica y, luego, cómo obtener las consultas **SQL** a partir de marcas sobre dicha interface.

7.2. Consultas en un Data Warehouse Histórico

Una característica distintiva de las herramientas **OLAP** es su acento sobre la agrupación de medidas mediante la elección de una o más dimensiones; la visión **MD** posibilita ver a los distintos niveles de jerarquías en cada dimensión de una manera lógica que permite al usuario una aproximación más intuitiva. Por otro lado, las **TDB**, permiten recuperar información histórica. La forma tradicional de acceder a una base de datos ha sido mediante consultas **SQL**, un lenguaje diseñado específicamente para crear, organizar y consultar bases de datos. Debido a la complejidad en la formulación de consultas en **SQL** se han propuesta diversos enfoques para hacerlas más accesible a un espectro mayor de usuarios [FKSS06]; el uso de lenguajes gráficos, comparado con las

expresiones algebraicas facilita, al usuario final, la especificación de consultas [RTTZ08].

La mayoría de los lenguajes gráficos temporales son extensiones de lenguajes de consulta como **SQL**, ellos están dirigidos a usuarios especializados con conocimiento de construcciones temporales. La mayoría de las **TDB** carecen de interfaces centradas en el usuario. Un ambiente visual centrado en el usuario debería incluir herramientas para el modelado de la **TBD**, la formulación de consultas, diferentes metáforas de visualización en inserción y actualización de datos temporales [KG95].

En comparación con la escritura de expresiones algebraicas para la realización de consultas sobre estructuras de almacenamiento, el uso de lenguajes de consultas gráficos facilita su especificación para el usuario final. Contrariamente al software comercial que provee diferentes vistas de los elementos **MD**, un lenguaje gráfico debería operar sobre una vista gráfica explícita del esquema **MD** conceptual; de la misma forma, las consultas **OLAP** beberían ser expresadas sobre la representación grafica en forma incremental [RTTZ08].

7.2.1. Ejemplo Motivador

El objetivo es plantear una simplificación en la búsqueda de información mediante consultas sobre el **HDW,** donde el usuario utilice un entorno gráfico sin considerar detalles de implementación.

A continuación, presentaremos un ejemplo que nos permitirá mostrar las características más relevantes de la propuesta.

Consideremos la siguiente consulta *¨Determinar la **cantidad** de **productos** vendidos por **provincia** y por **mes**¨,* podrá resolverse en forma automática, mediante marcas en un gráfico, donde los nodos representarán (cada uno con un icono diferente) tanto los niveles de jerarquía como las medidas involucradas. Este enfoque simplificará sobremanera la tarea del usuario; éste solo deberá centrarse en el análisis de los datos de dicha consulta y no en la forma en que ésta deberá resolverse.

> En la Figura 7.1, puede observarse el gráfico en donde el nodo M-cantidad expresa la medida y los nodos N-producto, N-provincia y N-mes (marcados en ese orden) detallan los niveles de la jerarquía utilizados en la consulta.

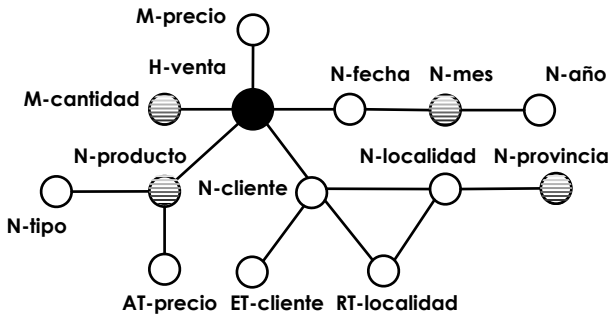


Figura 7.1. Ejemplo de Consulta Gráfica de Toma de Decisión

Del mismo modo, si el usuario precisara realizar consultas temporales sobre el **HDW**, marcando sobre el gráfico los nodos temporales, obtendría, en forma automática, la sentencia **SQL** necesaria para resolver consultas vinculadas con el tiempo de vida de entidades o el tiempo válido de atributos e interrelaciones.

Por ejemplo, "Determinar los diferentes valores que tuvo el **precio** de un **producto** dado a través del tiempo", la consulta se resolvería marcando, simplemente, el atributo temporal AT-precio en el gráfico[29] (Figura 7.2).

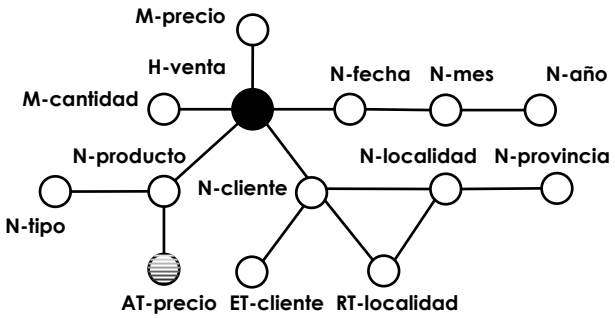


Figura 7.2. Ejemplo de Consulta Gráfica Temporal

En ambos casos, el resultado de aplicar estas consultas en una herramienta automatizada, será la obtención de las sentencias **SQL** que la implementarán en un **RDBMS**.

7.2.2. Visión General del Proceso de Transformación Informal

El método de transformación, que se inicia con un modelo **ER** que describe el esquema de datos fuente de la **ODB**, y continúa hasta la obtención de un **HDW** implementado en un **RDBMS,** mediante un conjunto de tablas en un **RM** expresadas en sentencias **SQL**, fue desarrollado en el capítulo 5.

En este capítulo, detallaremos las transformaciones #7 y #8 (Figura 5.2, capítulo 5). La primera, permitirá obtener un ***Query Graphic (QG)*** a partir de un **TMD**, la segunda, permitirá obtener (un conjunto de) transformaciones que se derivarán del **QG** que nos dará como resultado, sentencias de consulta **SQL** sobre la estructura de almacenamiento.

7.3. Diseño de una Interface Gráfica de Consultas

Detallaremos, a continuación, los pasos a seguir para la construcción del **QG** mediante la transformación informal a partir del **TMD.** En el Anexo IV, describiremos formalmente, en el marco de **MDD,** todas las transformaciones.

7.3.1. Grafo de Consultas

El **QG** permitirá establecer, para un usuario del **HDW**, consultas sobre la estructura de almacenamiento que, posteriormente, serán traducidas a sentencias **SQL**, en forma automática. El objetivo del **QG** es simplificar visualmente el **TMD**, con la intensión de que el usuario, mediante marcas en el grafo y el suministro de parámetros en el momento de realizar la consulta, pueda obtener los resultados requeridos en forma automática.

El **QG** se derivará del **TMD** y el conjunto de consultas posibles se establecerán mediante la identificación de patrones comunes a partir de la tipificación de las mismas.

El **QG** (Figura 7.3) estará compuesto por un nodo raíz que representa al hecho principal y un conjunto de nodos vinculados, cada uno de los cuales

[29] Además de ingresar el valor que identifique el atributo considerado.

representa a los distintos niveles de jerarquías, medidas, atributos temporales, entidades temporales e interrelaciones temporales.

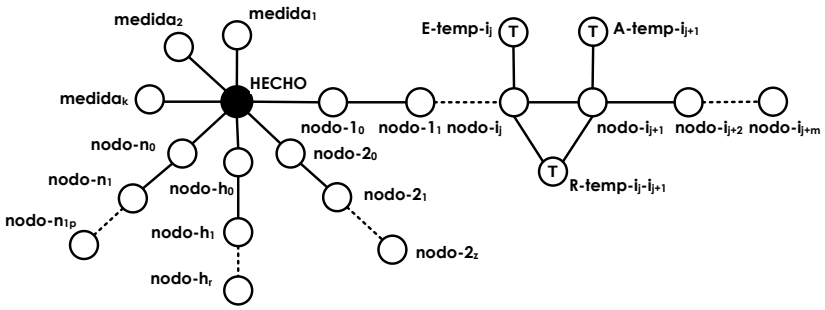


Figura 7.3. Grafo de Consultas

Para que el usuario pueda identificar claramente los componentes del gráfico en vista a realizar las consultas, cada nodo, dependiendo de lo que representará, tendrá un prefijo[30] que lo caracterizará y que estará asociado al nombre del componente del **TMD** del cual proviniere, esto es: ¨N-¨, nivel de jerarquía, ¨M-¨, medida, ¨AT-¨; atributo temporal; ¨ET-¨, entidad temporal y ¨RT-¨, interrelación temporal. Con el mismo criterio, el nodo raíz tendrá el prefijo ¨H-¨, y representará al hecho. Esto facilitará la visualización para el marcado, en el **QG,** por parte del usuario y le permitirá representar gráficamente las consultas requeridas.

Los nodo, con excepción de los que describen medidas, representarán a las tablas homónimas (sin prefijos) del **RM.**

7.3.2. Transformación del Modelo Multidimensional Temporal al Grafo de Consulta

Para la transformación del modelo **TMD** al **QG** utilizaremos el esquema general del **TMD** (Figura 7.4), presentado en el capítulo 5; en el modelo **TMD**, por razones de simplicidad, no estarán representados en forma explícita los atributos no dimensión; ellos serán considerados en la transformación formal (Anexo IV).

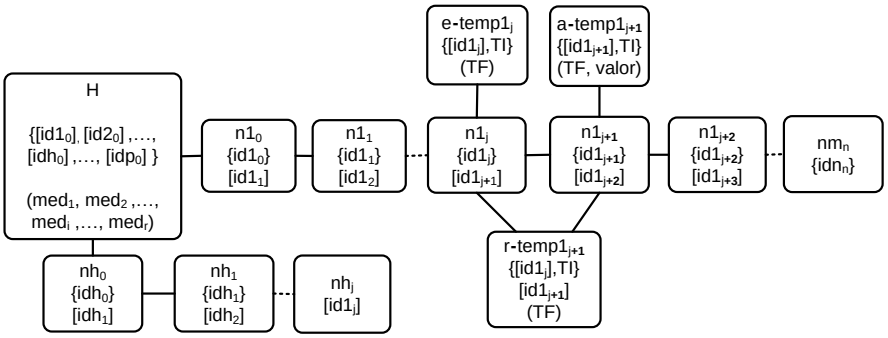


Figura 7.4 Esquema General del TMD

La transformación del **TMD** al **QG** se describirá a continuación.

7.3.2.1. Nodo Raíz (Hecho)

El nodo raíz representará al *hecho* del modelo **TMD**, el nombre del nodo estará compuesto por el prefijo ¨H-¨ vinculado al nombre del hecho (Figura 7.5).

[30] Esta es una simplificación; gráficamente, la diferenciación podría realzarse mediante diferentes colores o formas que identifiquen a cada componente.

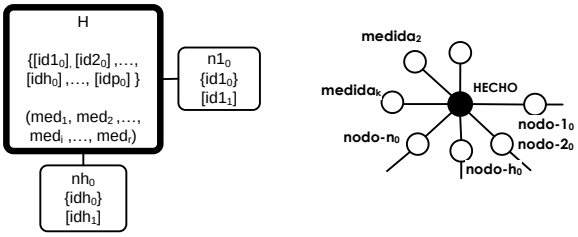


Figura 7.5. Transformación del Nodo *hecho*

Por ejemplo, el hecho *VENTA* (Figura 7.6, izquierda) se transformará en el nodo *hecho H-VENTA* (Figura 7.6, derecha).

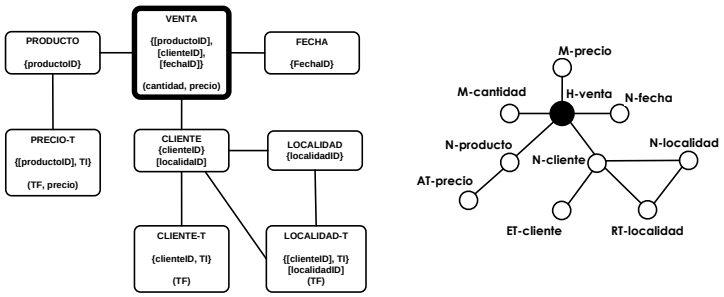


Figura 7.6. Transformación del *hecho* en Nodo *hecho*

7.3.2.2. Medidas

Todos las medidas del hecho se trasformarán en nodos anexos a nodo *hecho*, el nombre del nodo estará compuesto por el prefijo ¨M-¨ asociado al nombre de la medida (Figura 7.7).

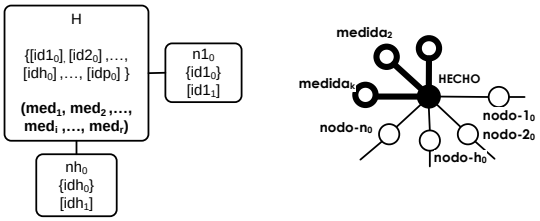


Figura 7.7. Transformación de Medidas

Por ejemplo, las medidas cantidad y precio (Figura 7.8, izquierda) se transformarán en los nodos *M-cantidad y M-precio* (Figura 7.8, derecha).

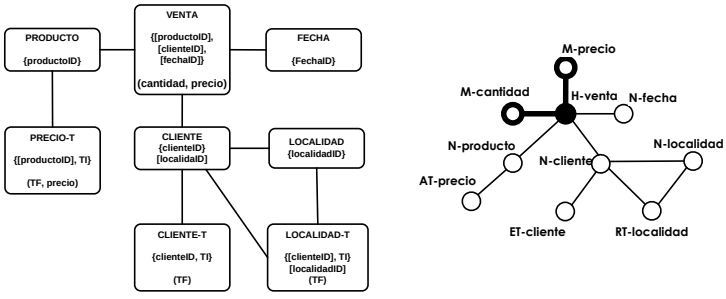


Figura 7.8. Transformación de Medidas en Nodos

7.3.2.3. Nivel *Hoja*

Los niveles *hoja* en la jerarquía se trasformarán en nodos anexos al nodo *HECHO*, el nombre del nodo estará compuesto por el prefijo ¨N-¨ asociado al nombre de la nivel (Figura 7.9).

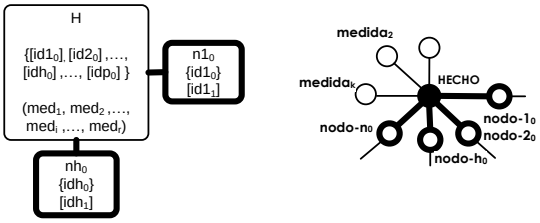


Figura 7.9. Transformación de Niveles

Por ejemplo, los niveles hoja *PRODUCTO, CLIENTE y FECHA* (Figura 7.10, izquierda) se transformarán en los nodos *N-PRODUCTO y N-CLIENTE y N-FECHA* (Figura 7.10, derecha)

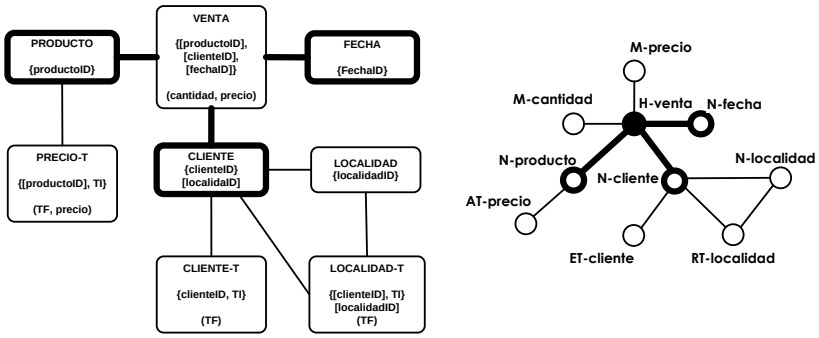


Figura 7.10. Transformación de Niveles *Hoja* en Nodos

7.3.2.4. Niveles en la Jerarquía

Los niveles en la jerarquía se trasformarán en nodos anexos a los niveles *hoja* respectivos, el nombre del nodo estará compuesto por el prefijo ¨N-¨ asociado al nombre del nivel (Figura 7.11).

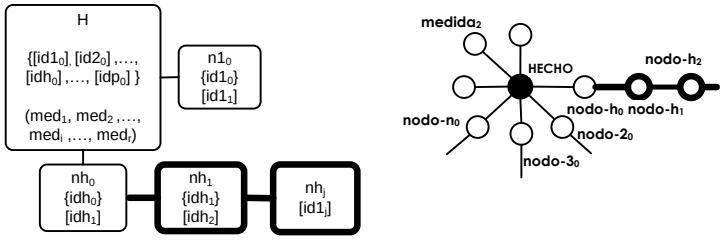


Figura 7.11. Transformación de Niveles de la Jerarquía

Por ejemplo, el nivel de jerarquía *LOCALIDAD* (Figura 7.12, izquierda) se transformará en el nodo *N-LOCALIDAD* vinculado el nodo *N-CLIENTE* (Figura 7.12, derecha).

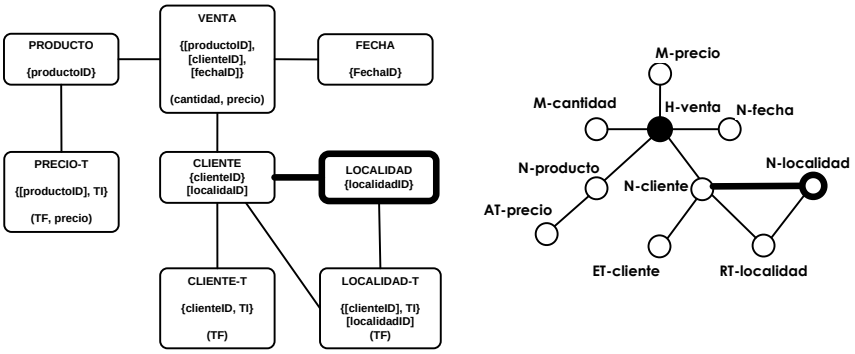


Figura 7.12. Transformación de Niveles de la Jerarquía en Nodos

7.3.2.5. Niveles Temporales

Los niveles de jerarquías temporales (entidades, atributos e interrelaciones) se transformarán en nodos vinculados, respectivamente, a los nodos involucrados; el nombre del nodo estará compuesto por el prefijo ¨AT-, ET-, RT-¨ asociado, respectivamente, al nombre del nivel temporal atributo, entidad o interrelación (Figura 7.13).

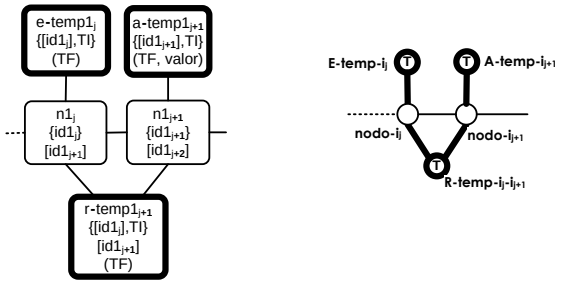


Figura 7.13. Transformación de Niveles de la Jerarquía Temporales

Por ejemplo, los niveles de jerarquía temporal *PRECIO-T, CLIENTE-T y LOCALIDAD-T* (Figura 7.14, izquierda) se transformarán en los nodos temporales *AT-PRECIO, ET-CLIENTE y RT-LOCALIDAD*, respectivamente (Figura 7.14, derecha).

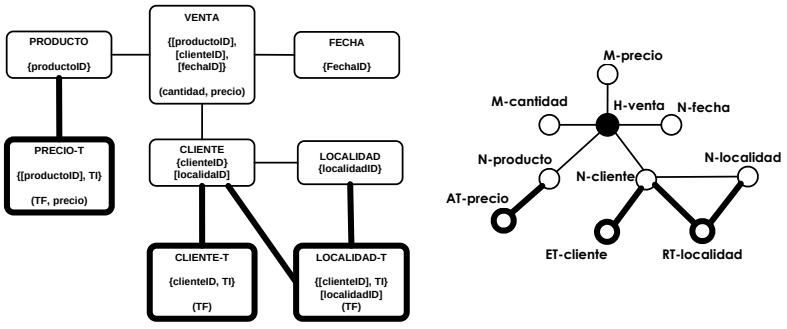


Figura 7.14. Transformación de Niveles de la Jerarquía Temporales en Nodos

7.4. Consultas en un Data Warehouse Histórico

Las consultas clasificadas, en el capítulo 6, como pertenecientes al **tipo D** y definidas como características de un **TDSS**, correspondían a la conjunción de

consultas de **tipo B** y **tipo C**, esto es, consultas en un **TDB** (en particular, un **HDB**) y consultas en un **DSS**, todas ellas realizadas sobre una misma estructura de almacenamiento.

Establecer un patrón para cada tipo de consulta, permitirá resolverlas en forma automática mediante la definición del tipo de consulta y uno o más parámetros que la especialicen (utilizando *parameters*) mediante una sentencia **SQL**.

Las consultas temporales sobre el **HDB** considerarán a entidades, atributos e interrelaciones temporales; las consultas de toma de decisión, contemplarán *roll-up, drill-down* y *slice and dice*.

Utilizaremos para la descripción de la transformación de consultas el **QG** (Figura 7.3) y el esquema del **TMD** (Figura 7.4) y, para la ejemplificación, el **TMD** y **RM** (Figura 7.15) utilizados en el capítulo 5.

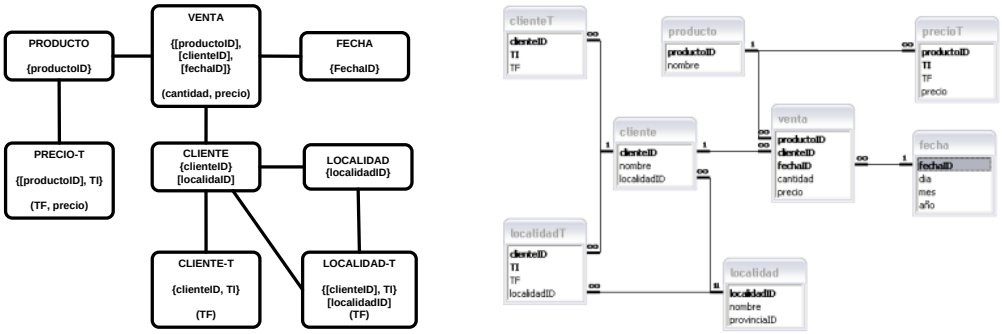


Figura 7.15. Modelo Multidimensional Temporal (izquierda) y Relacional (derecha)

7.4.1. Consultas Temporales

Las principales consultas que podrán realizarse en el **HDB** (capítulo 3) que forma parte del **HDW** (capítulo 5), consideran el tiempo de vida de entidades y el tiempo válido de atributos e interrelaciones. Si bien esto no invalida la posibilidad de realizar otro tipo de consultas sobre la estructura de almacenamiento, sí representan las más significativas del modelo propuesto.

Detallaremos, a continuación, cado uno de los tipos de consultas y, a partir marcas en el **QG**, su transformación directa a sentencias **SQL** usando parámetros**;** esta opción permitirá definir una consulta independiente de los distintos valores que vaya tomando a partir de su uso y será especializada mediante valores determinados de los parámetros por el **DBMS** en el momento de la ejecución de la misma.

7.4.1.1. Consultas Sobre Entidades Temporales

Para las entidades temporales se podrá obtener, mediante consultas, su tiempo de vida, esto es, los diversos rangos temporales en el cual esa entidad estuvo activa en el universo de discurso. Esta consulta se resolverá, mediante una sentencia **SQL**, obteniendo el intervalo temporal o rango de valores [TI, TF) de la entidad temporal.

Generalizando este criterio, podemos establecer un patrón de consulta sobre el **QG** para entidades temporales (Figura 7.16), donde el *nodo-h*$_j$*id* representa al identificador del nodo asociado al nodo temporal; *E-temp-h*$_j$*id*$_\mathbf{p}$, al identificador parcial del nodo temporal que hace referencia al nodo asociado;

nodo-$h_j$atr, al atributo descriptivo del nodo asociado[31], y TI y TF representan los valores extremos del intervalo temporal [TI, TF).

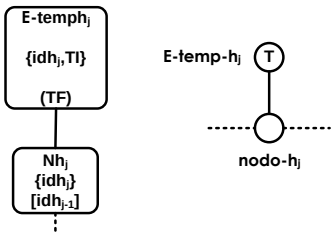


Figura 7.16. Consulta Genérica Sobre Entidades Temporales

Observamos que la única variable en la consulta es el identificador del nodo asociado a la entidad temporal, de modo tal que marcando en el **QG** el nodo temporal (que representa a la entidad temporal) y suministrando, como parámetro, el valor del identificador del nodo asociado, la consulta puede derivarse a partir de la información específica del **QG** más la obtenida en el **TMD** del cual fue derivado el grafo.

```
PARAMETERS p tipodato;
SELECT nodo-hjatr, TI, TF
FROM nodo-hj, E-temp-hj
WHERE nodo-hjid = E-temp-hjidp
AND nodo-hjid = p
ORDER BY nodo-hjatr, TI, TF;
```

Por ejemplo (Figura 7.17), la consulta ¿En qué intervalos un cliente en particular estuvo activo?, puede resolverse marcando en el **QG** el nodo temporal (ET-cliente).

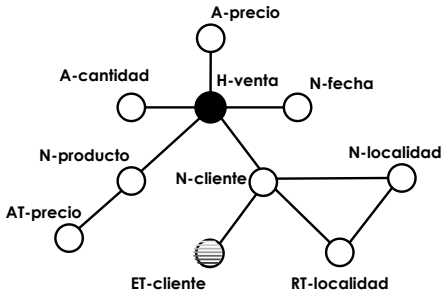


Figura 7.17. Consulta Sobre una Entidad Temporal

Suministrando en la consulta, como parámetro (p), el valor del identificador del nodo asociado (N-cliente[32]), la sentencia **SQL** que lo resuelve es:

```
PARAMETERS p text;
SELECT c.nombre, ct.ti, ct.tf
FROM cliente AS c, clientet AS ct
WHERE c.clienteid = ct.clienteid
AND c.clienteid = p
ORDER BY c.nombre, ct.ti, ct.tf;
```

[31] Los atributos descriptivos (atributos no dimensión) no estarán explícitamente representados en el **TMD**, con el objetivo de no atiborrar el gráfico.
[32] N-cliente, representa al nombre de la tabla cliente

7.4.1.2. Consultas Sobre Atributos Temporales

En las consultas sobre atributos temporales se podrá obtener a) en qué momentos se modificó su valor y cuál fue el mismo y, b) para una fecha determinada, cuál fue el valor de ese atributo.

En el caso a) la única variable en la consulta es el identificador del nodo asociado al nodo temporal, de modo tal que marcando en el **QG** el nodo temporal además de brindar, como parámetro, el valor del identificador del nodo asociado, la consulta quedará resuelta.

Generalizando este criterio, podemos establecer un patrón de consulta sobre el **QG** para determinar los diferentes valores de atributos temporales (Figura 7.18), donde el *nodo-h$_j$id* representa al identificador del nodo asociado al nodo temporal; *A-temp-h$_j$id$_p$*, al identificador parcial del nodo temporal que hace referencia al nodo asociado; *A-temp-h$_j$valor, al atributo variante*.

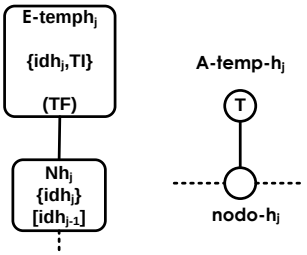


Figura 7.18. Consulta Genérica Sobre Modificación de Atributos Temporales

La consulta se resuelve suministrando el parámetro *p,* el valor especifico que identifica al nodo asociado al nodo temporal; TI y TF representan los valores extremos del intervalo temporal [TI, TF).

```
PARAMETERS p tipodato;
SELECT A-temp-hjvalor, TI, TF
FROM nodo-hj, A-temp-hj
WHERE nodo-hjid = A-temp-hjidp
AND nodo-hjid = p
ORDER BY A-temp-hjvalor, TI, TF;
```

Por ejemplo (Figura 7.19) la consulta ¿En qué fechas varió y cuál fue el precio de un producto determinado?, puede resolverse marcando en el **QG** el nodo temporal (AT-precio) y suministrando, como parámetro de la consulta, el valor del identificador de nodo asociado (N-producto).

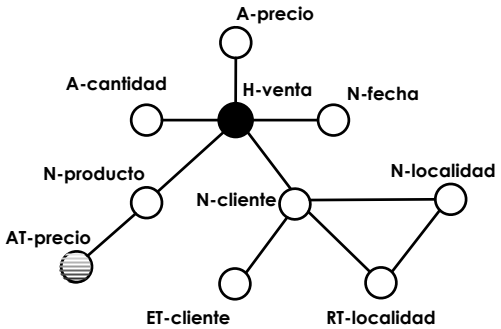


Figura 7.19. Consulta Sobre Modificación de Atributos Temporales

Suministrando en la consulta, el parámetro p, la sentencia **SQL** que lo resuelve es:

```
PARAMETERS p text;
SELECT pt.precio, pt.ti, pt.tf
FROM producto AS p , preciot AS pt
WHERE p.productoid = pt.productoid
AND p.productoid = p
ORDER BY pt.precio, pt.tI, pt.tf;
```

En el caso b) observamos que las variables en la consulta son el identificador del nodo asociado al nodo temporal y la fecha específica, de modo tal que marcando en el **QG** el nodo temporal además de brindar como parámetro el valor del identificador del nodo asociado y la fecha, la consulta quedará resuelta.

Generalizando este criterio, podemos establecer un patrón de consulta sobre el **QG** para determinar el valor de un atributo en un momento determinados (Figura 7.20), donde *nodo-h$_j$id* representa al identificador del nodo asociado al nodo temporal; *A-temp-h$_j$id$_p$*, al identificador parcial del nodo temporal que hace referencia al nodo asociado; *A-temp-h$_j$valor, al atributo variante;* el parámetro *p,* al valor especifico que identifica al nodo asociado al nodo temporal *y t*, al valor temporal específico.

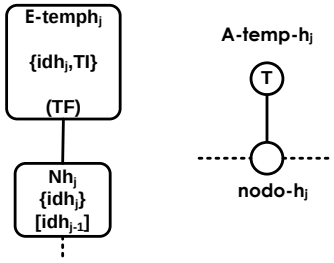


Figura 7.20. Consulta Genérica Sobre Valores Atributos Temporales

La consulta se resuelve suministrando la fecha específica de la consulta y el parámetro *p,* valor especifico que identifica al nodo asociado al nodo temporal; TI y TF representan los valores extremos del intervalo temporal [TI, TF).

```
PARAMETERS t datetime, p tipodato;
SELECT A-temp-hj.valor
FROM nodo-hj, A-temp-hj
WHERE nodo-hjid = A-temp-hjidp
AND nodo-hjid = p
AND TI <= t
AND TI >= t;
```

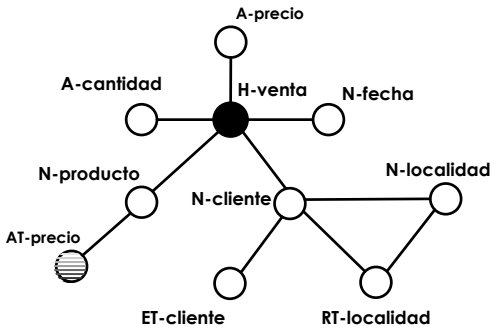


Figura 7.21. Consulta Sobre Valores Atributos Temporales

Por ejemplo (Figura 7.21), la consulta ¿Cuál fue el precio de un producto particular en una fecha determinada? se resolverá, mediante una

sentencia **SQL**, obteniendo el valor del atributo en esa fecha, marcando en el **QG** el nodo temporal (AT-precio) y suministrando, como parámetros, el valor que representa la fecha (t), además del valor del identificador del nodo asociado (N-producto).

Suministrando en la consulta, la fecha t y el parámetro p, la sentencia **SQL** que lo resuelve es:

```
PARAMETERS t datetime, p text;
SELECT pt.precio
FROM producto AS p, preciot AS pt
WHERE p.productoid  = pt.productoid
AND p.productoid = p
AND ti <= t
AND tf >= t;
```

7.4.1.3. Consultas Sobre Interrelaciones Temporales

Cuando consideramos consultas sobre interrelaciones temporales, se podrá determinar, a) en qué fechas se modificó el vínculo entre dos entidades y, b) para un instante en particular, cuál fue el vínculo entre ellas.

En el caso a) la única variable en la consulta es el identificador del *nodo-h$_j$* vinculado a la interrelación temporal, de modo tal que marcando en el **QG**, el nodo temporal que representa a la interrelación temporal, además de suministrar, cuando se realice la consulta, el valor del identificador del *nodo-h$_j$*, ésta quedará resuelta.

Generalizando este criterio, podemos establecer un patrón de consulta sobre el **QG** para interrelaciones temporales (Figura 7.22), donde *nodo-h$_j$id* y *nodo-h$_{j+1}$id* representan a los identificadores de los nodos involucrados; *R-temp-h$_j$-h$_{j+1}$id$_p$*, al identificador parcial del nodo temporal que hace referencia al *nodo-h$_j$*; *R-temp-h$_j$-h$_{j+1}$ref al atributo que hace referencia al nodo-h$_{j+1}$*; *nodo-h$_{j+1}$atr*, al atributo descriptivo del *nodo-h$_j$*; *p,* al valor específico que identifica al *nodo-h$_j$* y TI y TF representan los valores extremos del intervalo temporal [TI, TF).

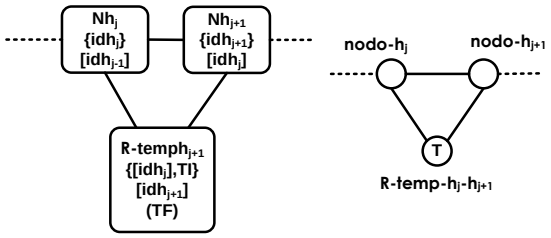


Figura 7.22. Consulta Genérica Sobre Vinculo Entre Entidades

La consulta se resuelve suministrando el parámetro *p,* valor especifico que identifica al nodo asociado al nodo temporal; TI y TF representan los valores extremos del intervalo temporal [TI, TF).

```
PARAMETERS p tipodato;
SELECT nodo-hj+1atr, TI, TF
FROM nodo-hj, nodo-hj+1, R-temp-hj-hj+1
WHERE nodo-hjid = R-temp-hj-hj+1idp
AND nodo-hj+1id = R-temp-hj-hj+1ref
AND nodo-hj = p
ORDER BY nodo-hj+1atr, TI, TF;
```

Por ejemplo (Figura 7.23), la consulta ¿En qué fechas se mudo un cliente en particular y a dónde?, puede resolverse marcando en el **QG** el nodo

temporal (RT-localidad) y suministrando, como parámetro, el valor del identificador del nodo asociado (N-cliente).

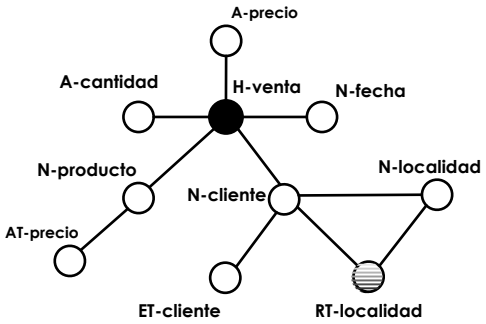


Figura 7.23. Consulta Sobre Vinculo Entre Entidades

La consulta se resuelve suministrando el parámetro *p,* valor especifico que identifica al nodo asociado al nodo temporal; TI y TF representan los valores extremos del intervalo temporal [TI, TF).

```
PARAMETERS p text;
SELECT l.nombre, ti, tf
FROM localidad AS l, localidadt AS lt, cliente AS c
WHERE l.localidadid = lt.localidadid
AND lt.clienteid = c.clienteid
AND c.clienteid = p
ORDER BY l.nombre, ti, tf
```

En el caso b) las variables son el identificador del *nodo-h$_j$* vinculado al nodo temporal y la fecha específica de consulta, de modo tal que marcando en el **QG**, el nodo temporal y suministrando el valor del identificador del *nodo-h$_j$* y la fecha, la consulta quedará resuelta.

Generalizando este criterio, podemos establecer un patrón de consulta sobre el **QG** para interrelaciones temporales (Figura 7.24), donde *nodo-h$_j$id* y *nodo-h$_{j+1}$id* representan a los identificadores de los nodos involucrados, *R-temp-h$_j$-h$_{j+1}$id$_p$*, al identificador parcial del nodo temporal que hace referencia al *nodo-h$_j$*; *R-temp-h$_j$-h$_{j+1}$ref al atributo que hace referencia al nodo-h$_{j+1}$*; *nodo-h$_{j+1}$atr*, al atributo descriptivo del *nodo-h$_j$*; *t*, al valor temporal específico y, *p,* al valor que identifica al *nodo-h$_j$.*

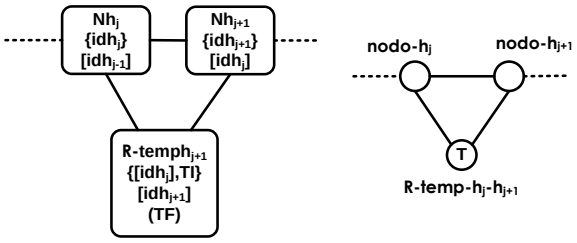


Figura 7.24. Consulta Genérica Sobre el Valor del Vínculo Entre Entidades

La consulta se resuelve suministrando la fecha específica y el parámetro *p,* valor especifico que identifica al nodo asociado al nodo temporal; TI y TF representan los valores extremos del intervalo temporal [TI, TF).

```
PARAMETERS p tipodato, t datetime;
SELECT nodo-hj+1atr
FROM nodo-hj, nodo-hj+1, R-temp-hj-hj+1
```

```
WHERE nodo-hjid = R-temp-hj-hj+1idp
AND nodo-hj+1id = R-temp-hj-hj+1ref
AND nodo-hj = p
AND TI <= t
AND TF >= t;
```

Por ejemplo (Figura 7.25), la consulta ¿Cuál fue la ubicación de un cliente en particular en una fecha determinada?, puede resolverse marcando en el **QG** el nodo temporal (RT-localidad) y suministrando, como parámetros, el valor del identificador del nodo asociado (N-cliente) y la fecha de consulta (t).

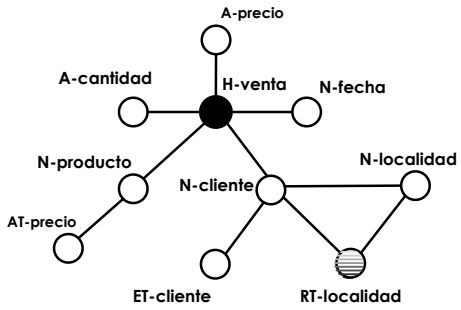


Figura 7.25. Consulta Sobre el Valor del Vínculo Entre Entidades

La consulta se resuelve suministrando la fecha específica y el parámetro *p,* valor especifico que identifica al nodo asociado al nodo temporal; TI y TF representan los valores extremos del intervalo temporal [TI, TF).

```
PARAMETERS t datetime, p text;
SELECT l.nombre
FROM localidad AS l, localidadt AS lt, cliente AS c
WHERE l.localidadid = lt.localidadid
AND lt.clienteid = c.clienteid
AND c.clienteid = p
AND TI <= t
AND TF >= t;
```

7.5. Consultas de Toma de Decisión

Las consultas típicas en un **DSS** (*roll- up, drill-down, y slice and dice*) se resolverán en el **TMD** marcando en el **QG** los nodos correspondientes a los niveles de jerarquía necesarios en un orden determinado y las medidas que se consideren como parámetros de la función de agrupación. Éstas luego serán generadas, en forma automática, mediante sentencias **SQL.**

Describiremos, a continuación, y en grado creciente de complejidad, consultas que involucran distintos dimensiones y niveles de jerarquía.

7.5.1. Consultas que Involucra un Nivel de Jerarquía *Hoja*

La consulta más simple vincula un nivel de jerarquía hoja y una función de agregado; marcando el nodo y la medida, y suministrando, además, la función de agrupamiento, la consulta podrá derivarse y expresarse en una sentencia **SQL** a partir de la información específica del **QG** más la obtenida en el **TMD** del cual fue derivado el grafo.

A continuación (Figura 7.26) se detallan el **TMD** y el **QG** marcado, donde *hecho.idh₀* representa a la clave parcial que hace referencia al nivel *hoja* y *nodo-h₀id*, al identificador del nivel *hoja*.

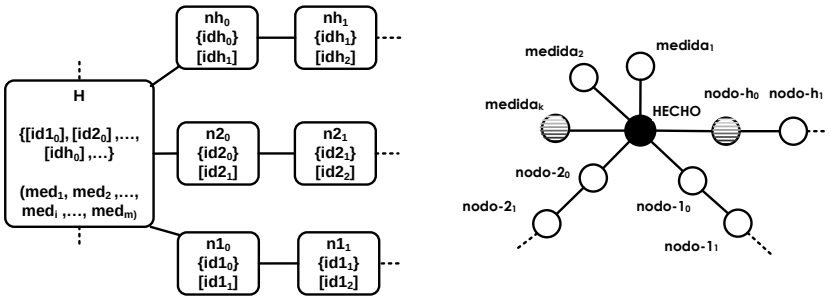


Figura 7.26. Consulta Genérica Sobre un Nivel Hoja

La sentencia **SQL** que resuelve la consulta es:

```
SELECT nodo-h0id, fun(medidak)
FROM hecho, nodo-h0,
WHERE hecho.idh0 = nodo-h0id
GROUP BY nodo-h0id
```

Por ejemplo (Figura 7.27), la consulta "Calcular la suma total de las ventas realizadas por la empresa", puede resolverse marcando en el **QG** el nodo N-producto) y la medida (M-precio), además de suministrar la función de agrupación requerida[33].

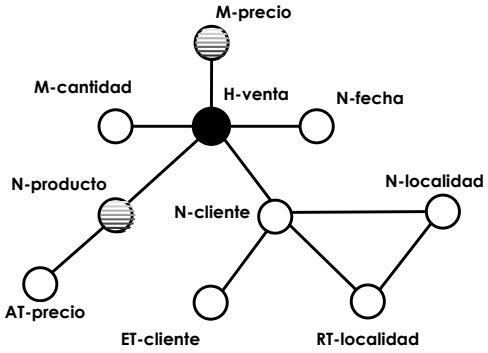


Figura 7.27. Consulta Sobre un Nivel Hoja

La sentencia **SQL** que la resolverá será:

```
SELECT p.productoid,  sum(v.precio)
FROM producto AS p, venta AS v
WHERE v.productoid = p.productoid
GROUP BY p.productoid;
```

7.5.2. Consultas que Involucra Dos Niveles de Jerarquía Hoja

Si la consulta vincula a dos niveles de jerarquía y a una función de agregado (Figura 7.28), marcando los niveles de jerarquía en el orden que determine el criterio principal y secundario de agrupamiento y, además, la medida, la

[33] Por simplicidad, en los ejemplos utilizaremos la función de agrupamiento SUM(), sin que esto reste generalidad al método.

consulta puede derivarse y expresarse en sentencias **SQL**, donde hechoid1$_0$ y hechoid2$_0$ representan las claves parciales que hacen referencia, respectivamente, a los niveles *hoja* y, *nodo-1$_0$id y nodo-2$_0$id*, respectivamente, a los identificadores de los niveles *hoja*.

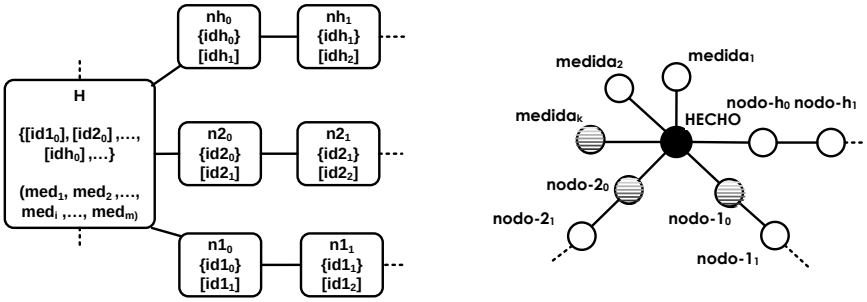


Figura 7.28. Consulta Genérica Sobre Dos Niveles Hoja

La sentencia **SQL** que la resolverá será:

```
SELECT nodo-1₀id, nodo-2₀id fun(medidaₖ)
FROM hecho, nodo-1₀, nodo-2₀
WHERE hechoid1₀ = nodo-1₀id
AND hechoid2₀ = nodo-2₀id
GROUP BY nodo-1₀id, nodo-2₀id
ORDER BY nodo-1₀id, nodo-2₀id
```

Por ejemplo (Figura 7.29), la consulta ¨Calcular la suma total de ventas realizadas por producto y cliente¨, puede resolverse marcando en el **QG** los nodos N-producto y N-cliente, en ese orden y la medida M-precio. La sentencia **SQL**, será:

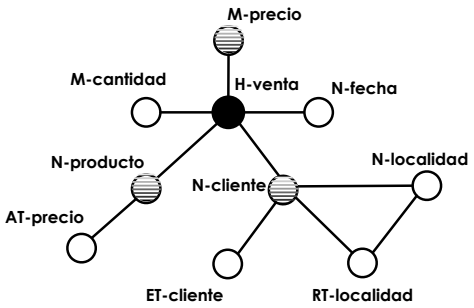


Figura 7.29. Consulta Sobre Dos Niveles Hoja

La sentencia **SQL** que la resolverá será:

```
SELECT p.productoid, c.clienteid, sum(v.precio)
FROM producto AS p, venta AS v, cliente c
WHERE v.productoid = p.productoid
AND v.clienteid = c.clienteid
ORDER BY p.productoid, c.clienteid
GROUP BY p.productoid, c.clienteid;
```

7.5.3. Consultas que Involucra Tres Niveles de Jerarquía Hoja

Por último, si ahora la consulta vincula a tres niveles de jerarquía y una función de agregado (Figura 7.30), marcando los niveles de jerarquía en el orden que determine el criterio principal y el o los criterios secundarios de agrupamiento y,

además, la medida, la consulta puede derivarse y expresarse en sentencias **SQL,** utilizando los mismos criterios establecidos anteriormente, de la siguiente forma:

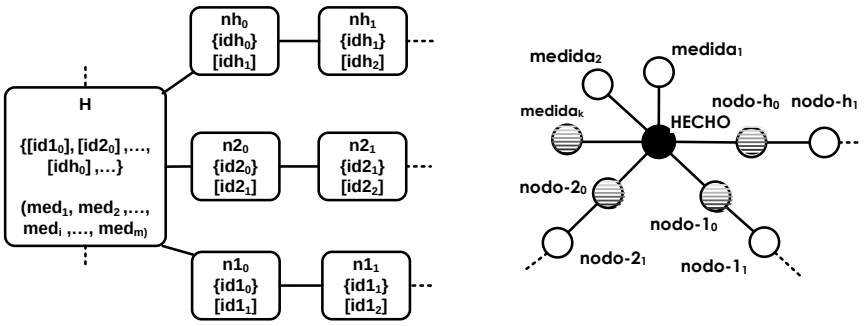


Figura 7.30. Consulta Genérica Sobre Tres Niveles Hoja

La sentencia **SQL** que la resolverá será:

```
SELECT nodo-1₀id, nodo-2₀id, nodo-h₀id, fun(medidaₖ)
FROM hecho, nodo-1₀,nodo-2₀, nodo-h₀
WHERE hecho.id1₀ = nodo-1₀id
AND hecho.id2₀ = nodo-2₀id
AND hecho.idh₀ = nodo-h₀id
GROUP BY nodo-1₀id, nodo-2₀id, nodo-h₀id
ORDER BY nodo-1₀id, nodo-2₀id, nodo-h₀id;
```

Por ejemplo (Figura 7.31), la consulta ¨Calcular la suma total de ventas realizadas por producto, por cliente y por fecha.

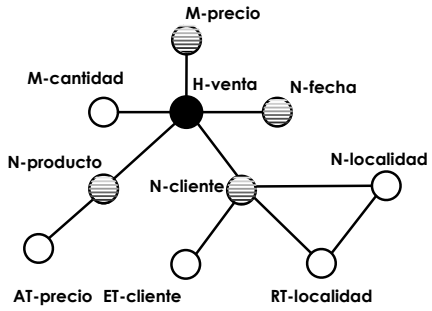


Figura 7.31. Consulta Sobre Tres Niveles Hoja

La sentencia **SQL** que la resolverá será:

```
SELECT p.productoid, c.clienteid, f.fechaid, sum(v.precio)
FROM producto AS p, venta AS v, fecha AS f, cliente as c
WHERE v.productoid = p.productoid
AND v.clienteid = c.clienteid
AND v.fechaid = f.fechaid
GROUP BY p.productoid, c.clienteid, f.fechaid
ORDER BY p.productoid, c.clienteid, f.fechaid;
```

7.5.4. Consultas que Involucran Distintos Niveles de Agrupamiento

Las consultas realizadas hasta aquí corresponden al mínimo nivel de agregación, si ahora quisiéramos considerar distintos niveles de agrupamiento, podemos generalizar el criterio utilizado hasta ahora. Por ejemplo, si consideramos una consulta con un primer nivel de agregación, la sentencia **SQL** será (Figura 7.32).

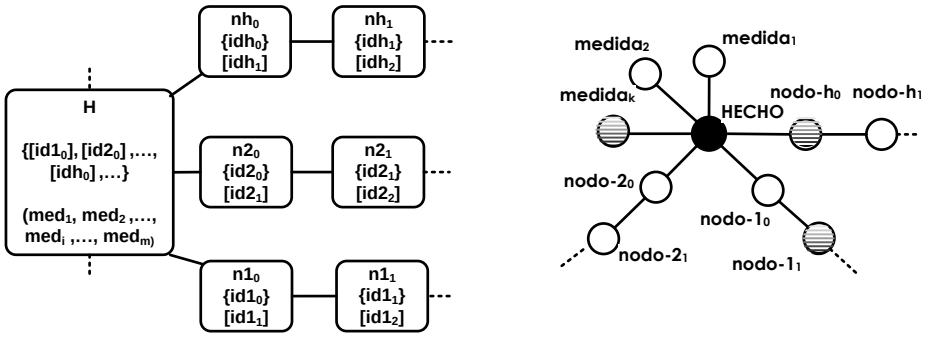


Figura 7.32. Consulta Genérica Sobre Distintos Niveles de Agrupamiento

La sentencia **SQL** que la resolverá será:

```
SELECT nodo-1₁id,  nodo-h₀id, fun(medidaₖ)
FROM hecho, nodo-1₀, nodo-1₁, nodo-h₀
WHERE hecho.id1₀ = nodo-1₀id
AND hecho.idh₀ = nodo-h₀id
AND nodo-1₀id = nodo-1₁id
GROUP BY nodo-1₁id, nodo-h₀id
ORDER BY nodo-1₁id, nodo-h₀id;
```

Por ejemplo (Figura 7.33), la consulta ¨Calcular la suma total de la cantidad de ventas realizadas por producto y por localidad¨, la sentencia **SQL**, será.

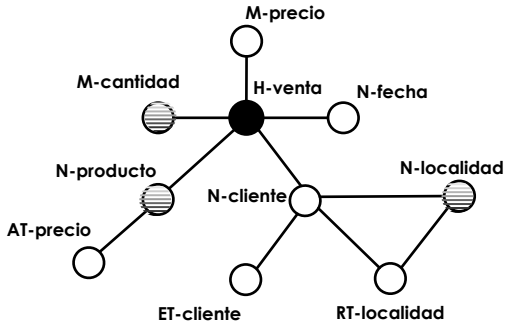


Figura 7.33. Consulta Sobre Distintos Niveles de Agrupamiento

La sentencia **SQL** que la resolverá será:

```
SELECT l.localidadid, p.productoid, sum(v.cantidad)
FROM producto AS p, venta AS v, cliente AS c, localidad AS l
WHERE v.productoid = p.productoid
AND v.clienteid=c.clienteid
AND l.localidadid = c.localidadid
GROUP BY l.localidadid, p.productoid
ORDER BY l.localidadid, p.productoid;
```

7.5.5. Consultas que Admiten Restricciones en la Dimensión Fecha

Todas las consultas realizadas hasta ahora admiten, además, establecer ciertas restricciones sobre las dimensiones de modo tal que se consideren para el análisis un subconjunto de esos valores.

Para mantener la simplicidad en la transformación de las consultas **SQL**, estableceremos ciertas restricciones: consideraremos un máximo de tres dimensiones de análisis al mismo tiempo y solo estableceremos restricciones sobre la dimensión fecha; además de la posibilidad de limitar en el rango de los

valores observados. Además, cuando se elijan más de una medida, se considerará que éstas se relacionarán dentro de la función de agrupamiento (sum()) mediante el operador de multiplicación.

Por ejemplo (Figura 7.34), la consulta ¨Calcular la suma de las ventas por cliente, por producto, y por fechas en un intervalo determinado, para valores de ventas dentro de un rango establecido.

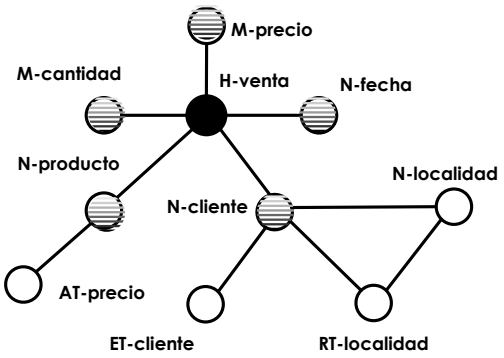


Figura 7.34. Consultas que Admiten Restricciones en la Dimensión Fecha

La sentencia **SQL** que lo resolverá, utilizando parámetros, será:

```
PARAMETERS [fecha inicial] datatime, [fecha final] datatime,
[desde valor] real, [hasta valor] real;
SELECT c.clienteid, p.productoid, f.fechaid,
sum(v.precio * v.cantidad) as ventas
FROM producto AS p, venta AS v, fecha AS f, cliente AS c
WHERE v.productoid = p.productoid
AND v.clienteid = c.clienteid
AND v.fechaid = f.fechaid
AND v.fechaid >= [fecha inicial]
AND v.fechaid <= [fecha final]
GROUP BY c.clienteid, p.productoid, f.fechaid
HAVING sum(v.precio * v.cantidad) >= [desde valor]
AND sum(v.precio * v.cantidad) <= [hasta valor]
ORDER BY c.clienteid, p.productoid, f.fechaid;
```

7.6. Resumen del Capítulo

El objetivo de este capítulo fue presentar una interface gráfica, derivada del modelo **TMD**, que permite realizar, mediante marcas en el mismo, consultas temporales y de toma de decisión. Primeramente, se mostró informalmente la transformación, paso a paso, del modelo **TMD** al **QG**: luego, se detallaron los patrones de consultas temporales y de toma de decisión y, a continuación, se describió cómo, mediante marcas en el **QG**, se pueden obtener las transformaciones que permiten expresar las consultas en sentencias **SQL.**

Capítulo 8

Arquitectura de Software Dirigida por Modelos

8.1. Introducción

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. [PGP09]

Actualmente la construcción de software se enfrenta a continuos cambios en las tecnologías de implementación, lo que implica realizar importantes esfuerzos tanto en el diseño de la aplicación, para integrar las diferentes tecnologías que lo componen, como en el mantenimiento, para adaptar el sistema tanto a los continuos cambios en los requisitos, como a las tecnologías que lo implementan.

En enfoque **MDD**, plantea una arquitectura para el desarrollo de aplicaciones informáticas cuyo objetivo es proporcionar una solución para los cambios de negocio y de tecnología, tal que permita construir aplicaciones independientes de su posterior implementación. Este enfoque representa un nuevo paradigma en donde se utilizan modelos del sistema, a distinto nivel de abstracción, para guiar todo el proceso de desarrollo. La idea clave subyacente es que, si se trabaja con modelos, se obtendrán importantes beneficios tanto en la productividad, la portabilidad, la interoperatividad, el mantenimiento y la documentación [KWB03].

En este capítulo presentaremos, primeramente, una visión general del enfoque de la arquitectura de software dirigida por modelos. Luego, se describirán los diferentes niveles de madurez de los modelos. Posteriormente, se detallarán los diferentes modelos usados y los tipos básicos de transformaciones entre ellos. Luego, se detallarán las ventajas de la arquitectura de software dirigida por modelos sobre el proceso tradicional de desarrollo de software; posteriormente, se especificarán las diferentes propuestas del enfoque dirigida

por modelos. Por último, se detallarán los distintos grados y métodos de transformación de modelos.

8.2. Visión General del Enfoque MDD

MDD se sintetiza como la combinación de tres ideas complementarias [BBR04]: a) *la representación directa*, ya que el foco del desarrollo de una aplicación se desplaza del dominio de la tecnología hacia las ideas y conceptos del dominio del problema, de este modo se reduce la distancia semántica entre el dominio del problema y su representación**,** b) *la automatización*, porque promueve el uso de herramientas automatizadas en aquellos procesos que no dependan del ingenio humano, de este modo se incrementa la velocidad de desarrollo del software y se reducen los errores debidos a causas humanas, y c) *los estándares abiertos*, debido a que éstos no solo ayudan a eliminar la diversidad innecesaria sino que, además, alientan a producir, a menor costo, herramientas tanto de propósito general como especializadas.

Podemos dividir, concisamente, el proceso **MDD** en tres fases; en la primera se construye un ***Platform Independent Model* (PIM)**, éste es un modelo de alto nivel del sistema que se desarrolla independientemente de cualquier tecnología; luego, se transforma el modelo anterior a uno o más **Platform *Specific Model* (PSM)**, éstos modelos son de más bajo nivel que el **PIM** y describen al sistema de acuerdo con una tecnología de implementación determinada; por último, se genera un ***Implementation Model* (IM)**, esto es, código fuente a partir de cada **PSM**. La división entre **PIM** y **PSM** está vinculada al concepto de plataforma, el cual, al no estar expresamente definido, tampoco permite delimitar claramente la línea divisoria entre **PIM** y **PSM**.

MDD también presenta un ***Computation Independent Model* (CIM)** que describe al sistema dentro de su ambiente y muestra lo que se espera de él sin exhibir detalles de cómo será construido.

El beneficio principal del enfoque **MDD** es que una vez que se ha desarrollado cada **PIM**, se puede derivar, automáticamente, el resto de los modelos aplicando las correspondientes transformaciones en forma vertical.

En **MDD** se describen tres opciones para la transformación de modelos: a) que ésta sea totalmente manual y sea realizada por los desarrolladores de la aplicación; b) que sea necesaria la participación de desarrolladores en la transformación, pero que éstos estén guiados por herramientas automatizadas y, por último, c) que sean generadas automáticamente a partir de los modelos del sistema, sin intervención humana.

En el desarrollo de nuestra propuesta de un **HDW** utilizaremos la segunda opción. El proceso de desarrollo estará automatizado en su mayor parte, pero requerirá de ciertas consideraciones de diseño que deberán ser tomadas por el desarrollador de la aplicación aunque, no obstante, estarán guiadas por la herramienta de desarrollo.

8.3. Los Modelos en el Contexto de MDD

Un modelo es una representación de una parte de la función, estructura y/o comportamiento de una aplicación o sistema. Una representación es formal cuando está basada en un lenguaje que tienen una forma (sintaxis) y un significado (semántica) bien definidos e, incluso, puedan aplicárseles reglas de análisis, inferencia o pruebas. La sintaxis puede ser gráfica o textual [QVT]

MDD enfatiza el hecho de que los modelos son el foco central en el desarrollo de software. Por lo tanto, es necesaria una definición que sea lo suficientemente general para abarcar varios tipos diferentes de modelos pero, que al mismo tiempo, sea lo suficientemente específica para permitirnos definir transformaciones automáticas de un modelo a otro. En el ámbito científico un modelo puede ser tanto un objeto matemático, un gráfico o un objeto físico. El modelo se focaliza sobre el mundo real: identificando, clasificando y abstrayendo los elementos que constituyen el problema y organizándolos en una estructura formal [PGP09].

El modelo de un sistema es una descripción o especificación del mismo y de su entorno para un propósito determinado. Los modelos, frecuentemente, se representan en una combinación de gráficos y texto, la forma de descripción textual puede ser mediante un lenguaje de modelado o, simplemente, a través del lenguaje natural. **MDD** es un enfoque para el desarrollo de sistemas en donde los modelos juegan un papel de suma importancia. El proceso de desarrollo se denomina conducido por modelos (*Model Driven*) debido a que se utilizan los modelos para dirigir los procesos de comprensión, diseño, construcción, despliegue, operación, mantenimiento y modificación de sistemas. **MDD**, por lo tanto, prescribe un conjunto de modelos que deben ser utilizados, cómo éstos deben ser construidos y los diferentes tipos de relaciones entre ellos [MDA].

El modelo de un problema es esencial para describirlo y entenderlo, independientemente de cualquier posible sistema informático que se use para su automatización. El modelo constituye la base fundamental de información sobre la que interactúan los expertos en el dominio del problema y los desarrolladores de software. Por lo tanto es de fundamental importancia que exprese la esencia del problema en forma clara y precisa.

Las cualidades que esperamos encontrar en los modelos están expresadas en [PGP09]. Un modelo deber ser:

- *Comprensible*. Debe ser expresado en un lenguaje que resulte accesible (es decir entendible y manejable) para todos sus usuarios.
- *Preciso*. Debe ser una fiel representación del objeto o sistema modelado. Para que esto sea posible, el lenguaje de modelado debe poseer una semántica precisa que permita la interpretación unívoca de los modelos. La falta de precisión semántica es un problema que no solamente atañe al lenguaje natural, sino que también abarca a algunos lenguajes gráficos de modelado que se utilizan actualmente.
- *Consistente*. No debe contener información contradictoria. Dado que un sistema es representado a través de diferentes sub-modelos relacionados, debería ser posible especificar precisamente cual es la relación existente entre ellos, de manera que sea posible garantizar la consistencia del modelo como un todo.
- *Suficientemente completo*. Debe documentar todos los requerimientos necesarios. Dado que en general, no es posible lograr un modelo completo desde el inicio del proceso, es importante poder incrementar el modelo. Es decir, comenzar con un modelo incompleto y expandirlo a medida que se obtiene más información acerca del dominio del problema y/o de su solución.
- *Modificable*. Debido a la naturaleza cambiante de los sistemas actuales, es necesario contar con modelos flexibles; es decir, que puedan ser fácilmente adaptados para reflejar las modificaciones en el dominio del problema.
- *Reusable*. El modelo de un sistema, además de describir el problema, también debe proveer las bases para el reuso de conceptos y

construcciones que se presentan en forma recurrente en una amplia gama de problemas.

- *Permitir validación y verificación*. El análisis de la corrección del sistema de software debe realizarse en dos puntos. En primer lugar el modelo en sí debe ser analizado para asegurar que cumple con las expectativas del usuario. Este tipo de análisis generalmente se denomina 'validación del modelo'; luego, asumiendo que el modelo es correcto, puede usarse como referencia para analizar la corrección de la implementación del sistema, esto se conoce como 'verificación del software'. Ambos tipos de análisis son necesarios para garantizar la corrección de un sistema de software.

8.4. Niveles de Madurez de los Modelos

En [WK03] se presenta el concepto de niveles de madurez de modelos ***Maturity Model Level* (MML)**, mediante el cual se puede descubrir en qué estado de madurez se encuentra un grupo de desarrollo de software respecto al uso de modelos. Estos son:

- **MML** *0, sin especificar*. En este nivel no se utilizan modelos; la especificación del software se conserva en la mente de los desarrolladores.
- **MML** *1, especificación textual*. La especificación del software en este nivel está escrita en uno o más documentos, pero éstos están escritos en lenguaje natural.
- **MML** *2, texto con diagramas*. En este nivel, la especificación del software está escrita en uno o más documentos en lenguaje natural y, además, están acompañados con varios diagramas de alto nivel que permiten explicar la arquitectura global.
- **MML** *3, modelos con texto*. La especificación del software en este nivel está escrita en uno o más modelos, además de descripciones textuales en lenguaje natural para explicar el trasfondo y la motivación de los modelos.
- **MML** *4, modelos precisos*. La especificación del software en este nivel está escrita en uno o más modelos; además, el texto en lenguaje natural se utiliza para explicar el trasfondo y la motivación de los modelos. Los modelos son precisos y alcanzan a tener una relación directa con el código.
- **MML** *5, solo modelos*. En este nivel los modelos son lo suficientemente precisos y detallados como para permitir una completa generación de código. El lenguaje de modelado equivale a un lenguaje de programación de alto nivel. Observamos que el objetivo principal de **MDD** está íntimamente vinculado este nivel.

8.5. Los Diferentes Modelos de MDD

Un punto de vista de un sistema es una técnica de abstracción que utiliza un conjunto de conceptos arquitectónicos y reglas de estructuración para focalizar una parte dentro del sistema. El concepto de "*abstracción*" se utiliza para representar el proceso de supresión selectiva de detalles menos significativos para establecer un modelo simplificado. Una vista de un sistema es una representación del sistema desde un punto de vista determinado.

Una plataforma, por otro lado, es un conjunto de subsistemas y de tecnologías que proveen un conjunto coherente de funcionalidad a través de interfaces y patrones especificados que cualquier aplicación puede usar, sin tener en cuenta los detalles de cómo esa funcionalidad será provista por la plataforma que la implementa. En **MDD**, el término plataforma es utilizado para referirse a detalles tecnológicos o ingenieriles que son irrelevantes al momento de describir la funcionalidad de un componente software [MDA].

La independencia de plataforma es una cualidad que un modelo puede exhibir y que expresa que es independiente de cualquier tipo particular de implementación. De todos modos, como cualquier otra cualidad, la independencia de plataforma es un tema de grado.

El estándar **OMG** define los términos **PIM** y **PSM**, estableciendo una clara distinción entre ambos conceptos. De todos modos, un modelo siempre tiene partes de ambos ya que es difícil establecer una tajante distinción entre ellos. Lo único que se puede afirmar es que un modelo es más (o menos) específico de una plataforma que otro; por lo tanto, estos términos son relativos [KWB03]

Detallaremos, a continuación, los diferentes modelos que presenta **MDD,** el **CIM**, **PIM**, **PSM** y, por último, el **IM**

8.5.1. Modelo Computacionalmente Independiente

El **CIM** es una vista del sistema que lo describe sin considerar aspectos computacionales, esto es, no muestra detalles de la estructura del sistema. Al **CIM**, habitualmente se lo denomina modelo del dominio, y utiliza en su especificación un vocabulario común al profesional vinculado a la aplicación, donde asume que éste es el usuario primario. El modelo del dominio no necesariamente hace referencia al software del sistema adoptado en la empresa, de todos modos, parte de la lógica del negocio está soportada por un sistema software [KWB03]. Este modelo juega un rol importante en la disminución de la brecha entre los expertos del dominio de la aplicación y los diseñadores de los artefactos que satisfarán los requerimientos de dicho dominio.

El **CIM** describe las situaciones en las cuales el sistema será usado, este modelo puede ocultar mucha o toda la información vinculada con los sistemas de procesamiento de datos automáticos. Por otro lado, describe al sistema en el ambiente en el cual operará, de este modo permitirá la representación exacta de lo que se espera que el sistema realice. No solo es útil para la comprensión del problema, sino también como una fuente de vocabularios compartidos para el uso en otros modelos.

En una especificación **MDD**, los requerimientos **CIM** deberían ser fáciles de rastrear desde los **PIM** hasta los **PSM** que lo implementan y viceversa. El **CIM**, como modelo independiente de la plataforma, se utiliza para describir un sistema, por lo tanto, algunas partes del **CIM** puede ser implementados en software, pero este modelo, por si mismo, permanece independiente del software que lo implementará. No es posible la derivación automática de un **CIM** a **PIM**, debido a que las decisiones de qué partes deberán ser implementadas son siempre decisiones humanas. Por lo tanto, por cada sistema que implementará parte del **CIM**, deberá desarrollarse, primeramente, un **PIM** vinculado al mismo [KWB03].

8.5.2. Modelo Independiente de la Plataforma

Un **PIM** es una vista del sistema que no contempla aspectos vinculados con la plataforma que se usará para su implementación. Este modelo exhibe un grado específico de independencia debido a que puede ser utilizado adecuadamente para un conjunto de plataformas de igual tipo.

El uso del **PIM** permite validar más fácilmente la corrección del modelo que describe, ya que carece de detalles de implementación irrelevantes en etapas tempranas de desarrollo. Este modelo facilita la producción de implementaciones en diversas plataformas conforme a una misma descripción esencial y precisa de la estructura y comportamiento del sistema. Además, la integración e interoperabilidad entre sistemas puede ser definida más claramente en términos de plataformas independientes.

Por ejemplo, en la Figura 8.1 se muestra un **PIM** que representa a un modelo de datos temporal mediante la representación de un modelo **ER.** En la Figura 8.2 se describe mediante otro **PIM**, un grafo de atributos (ver capítulo 5). En ambos modelos se omiten detalles de cómo el sistema será implementado, no obstante tienen la información suficiente para poder realizarlo.

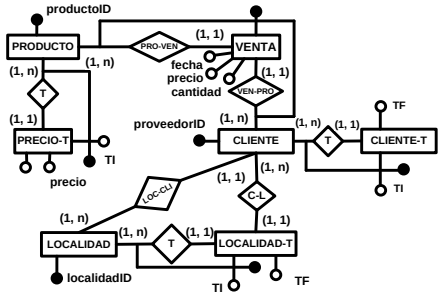


Figura 8.1. Modelo de Datos Temporal (PIM)

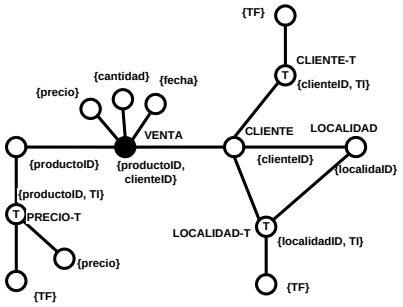


Figura 8.2. Modelo de Grafo de Atributos (PIM)

8.5.3. Modelo Específico de la Plataforma

Un **PSM** es una vista del sistema desde la perspectiva de una plataforma específica; esto es, posee detalles específicos de la plataforma en la que será implementado. Este modelo combina las especificaciones descritas en el **PIM** con los detalles que describen cómo el sistema usa un particular tipo de plataforma. El **PSM** es un modelo del sistema de más bajo nivel que el **PIM**, mucho más cercano al código; este modelo ofrece un conjunto de conceptos técnicos que representan los diferentes partes que hacen a la plataforma y a los servicios provistos por ella.

Por ejemplo, en la Figura 8.3 se muestra una representación de un modelo **TMD**, pero con detalles vinculados al **RM** (**PSM**), donde podemos observar, en forma indirecta, nombre de tablas, claves primarias, claves foráneas, restricciones de integridad referencial, etcétera.

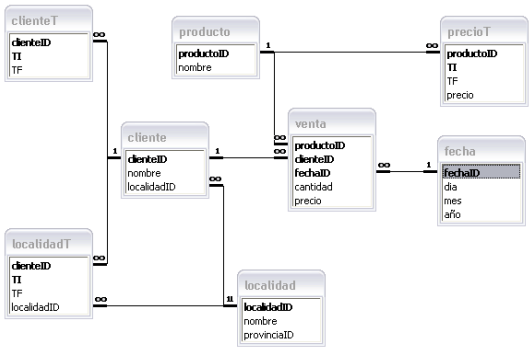


Figura 8.3. Modelo de Datos Multidimensional Temporal (PSM)

8.5.4. Modelo de Implementación

El paso final en el desarrollo es la transformación de cada **PSM** a código fuente, esto es, al **IM**. Debido a que el **PSM** tiene una relación cercana a la tecnología utilizada en la implementación, esta última transformación es, relativamente, simple.

Por ejemplo, en la Figura 8.4 se observa el código fuente (**IM**) expresado en sentencias **SQL** que permite implementar en un **RDBMS**, la tabla *VENTA* del modelo **TMD** de la Figura 8.3.

```
CREATE TABLE VENTA (productoID text, clienteID text,
                    fechaID date, cantidad numeric,
                    precio numeric)
       PRIMARY KEY (productoID, clienteID, fechaID)
       FOREIGN KEY (productoID) REFERENCES PRODUCTO,
       FOREIGN KEY (clienteID)  REFERENCES CLIENTE,
       FOREIGN KEY (fechaID)    REFERENCES FECHA,
                 )
```

Figura 8.4. Código Fuente (IM)

8.6. Transformación de Modelos

MDD detalla el rol, en el proceso de desarrollo, de cada uno de los modelos **PIM**, **PSM** y del **IM**. Una herramienta de transformación toma un **PIM** y lo transforma en uno (o más) **PSM** y de éste a un **IM**. Por lo tanto la transformación es fundamental en este proceso.

Las herramientas de transformación definen cómo un modelo será transformado; esto se denomina definición de transformación. Hay una diferencia entre la transformación en sí misma, que es el proceso de generar un nuevo modelo a partir de otro, y la definición de transformación. La herramienta de transformación utiliza la misma definición de transformación para cualquier modelo de entrada. Para permitir esta generalidad, la herramienta relaciona elementos del lenguaje fuente con elementos del lenguaje destino. En general, podemos decir que una definición de transformación consiste en una colección de reglas de transformación.

Una transformación es la generación automática de un modelo destino a partir de un modelo fuente de acuerdo a una definición de transformación. Una definición de transformación es un conjunto de reglas de transformación que, en conjunto, definen cómo un modelo fuente será transformado a un modelo destino; por último, una regla de transformación es una descripción de cómo

uno o más elementos en un lenguaje fuente pueden ser transformados en uno o más elementos en el lenguaje destino [KWB03].

Una de las características claves del enfoque **MDD** es la noción de *mapping*. Un *mapping* es un conjunto de reglas y técnicas utilizadas para modificar un modelo con el objetivo de obtener otro. Los *mappings* son utilizados para realizar diferentes tipos de transformaciones [MDA]:

8.6.1. Transformación PIM a PIM

La transformación **PIM** a **PIM** se utiliza cuando los modelos precisan ser ampliados, filtrados o especializados durante el desarrollo, sin la necesidad de información dependiente de la plataforma sobre la cual podrá ser posteriormente implementada. Un caso particular es la transformación del modelo de análisis al de diseño. Este tipo de transformación está relacionado con el concepto de refinamiento [MDA].

Por ejemplo, en la Figura 8.5 se observa una transformación de **PIM** a **PIM**, donde, a partir de un modelo de datos expresado mediante un modelo **ER** (Figura 8.5, izquierda) se obtiene un **AG** (Figura 8.5, derecha); en ninguno de los dos modelos se observan características de una implementación particular.

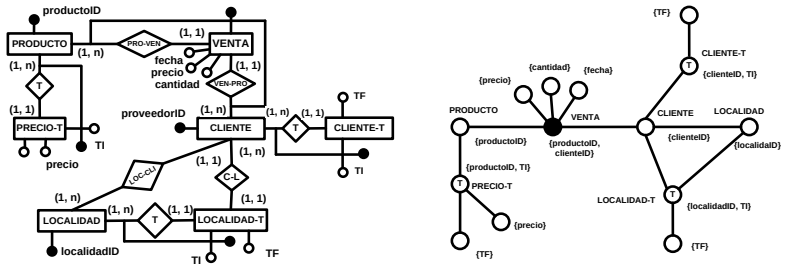


Figura 8.5. Transformación PIM a PIM

8.6.2. Transformación PIM a PSM

Este tipo de transformación se utiliza cuando el **PIM** está lo suficientemente refinado como para poder ser transformado a una plataforma con una tecnología subyacente especifica.

Por ejemplo, en la Figura 8.6 se observa una transformación de **PIM** a **PSM**, donde, a partir de un **AG** (Figura 8.6, izquierda) se obtiene un modelo **TMD** (Figura 8.6, derecha); en este último observamos características del modelo de implementación (**RM**).

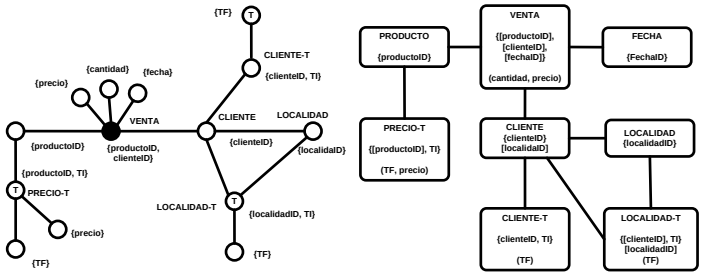


Figura 8.6. Transformación PIM a PSM

8.6.3. Transformación PSM a PSM

Estas transformaciones son necesarias para la realización y despliegue de componentes. Este tipo de transformaciones está relacionada con el refinamiento de modelos dependientes de la plataforma.

Por ejemplo, en la Figura 8.7 se observa una transformación de **PSM** a **PSM**, donde, a partir de un modelo **TMD** (Figura 8.7, izquierda) se obtiene un **RM** (Figura 8.7, derecha).

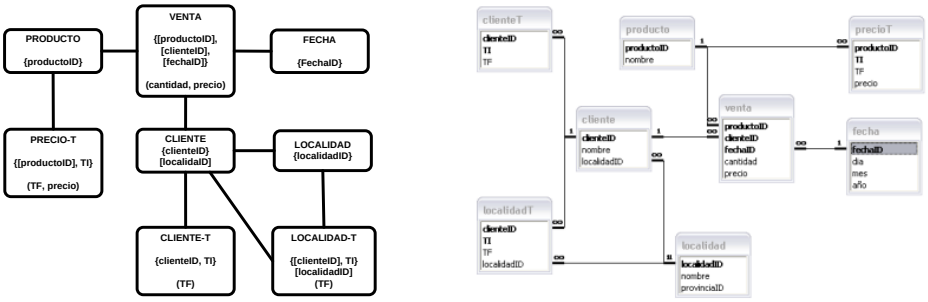


Figura 8.7. Transformación PSM a PSM

8.6.4. Transformación PSM a PIM

Esta transformación se utiliza para obtener abstracciones de modelos, a partir de implementaciones existentes en una tecnología particular, representada en un **PIM**.

8.6.5. Puentes de Comunicación

Los esquemas de transformación presentados anteriormente pueden complementarse con una visión más general (y más realista) donde varios **PSMs** derivados del mismo **PIM** (donde cada **PSM** representa una parte del sistema) se comunican entre si mediante puentes de comunicación, tanto a nivel de **PSM** como a nivel de código. [KWB03]

8.7. El Proceso de Desarrollo en MDD

El ciclo de vida tradicional de desarrollo de software incluye las etapas de requisitos, análisis, diseño, codificación, prueba e implementación. El proceso **MDD** incluye, conceptualmente, las mismas actividades; incluso las etapas de requisitos, prueba e implementación no presentan diferencias. El cambio principal lo encontramos en las actividades de análisis, diseño y codificación (Figura 8.8).

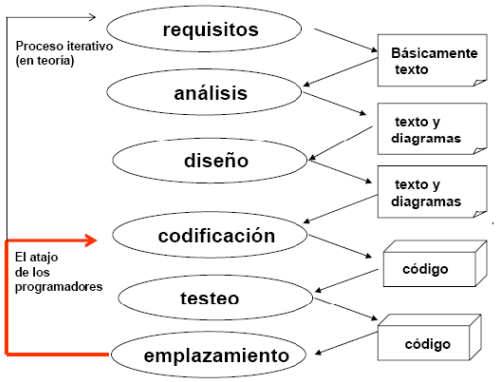


Figura 8.8. Ciclo de Vida Tradicional [KWB03]

Mediante el enfoque **MDD**, el análisis será realizado por un grupo de personas que desarrollarán el **PIM** a partir de las necesidades del usuario y la funcionalidad del sistema. El diseño, ahora, será desarrollado por un grupo diferente de personas, con conocimiento sobre distintas plataformas y arquitecturas, que realizarán las transformaciones del **PIM** a uno o más **PSMs**. Por último, la generación de código (**IM**), se realizará mediante herramientas especializadas que lo generará en forma automática o semiautomática (Figura 8.9).

Surge, ahora, un nuevo tipo de desarrolladores, aquellos que escriben las definiciones de las transformaciones que serán utilizadas en las herramientas de **MDD** y que permitirán las transformaciones de **PIM** a **PSM** y de este al **IM.**

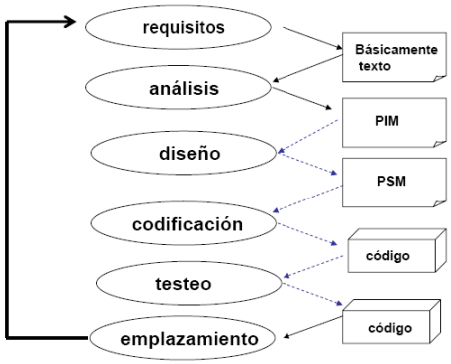


Figura 8.9. Ciclo de Vida de Desarrollo de Software Dirigido por Modelos [KWB03]

8.7.1. Beneficios de MDD

El enfoque **MDD** brinda beneficios tanto en productividad, portabilidad, interoperatividad y mantenimiento y documentación [KWB03].

8.7.1.1. Productividad

En **MDD** el foco está en el desarrollo de un **PIM**. Los **PSMs** necesarios son generados mediante una transformación desde un **PIM**. Ya que los desarrolladores necesitan enfocarse solo en modelos sin considerar aspectos de implementación, pueden trabajar independientemente de los detalles de las plataformas tecnológicas específicas. Esos detalles técnicos serán agregados automáticamente por la transformación del **PIM** al **PSM**. Por tanto los desarrolladores del **PIM** tendrán menos trabajo que realizar, ya que los detalles

específicos de la plataforma no necesitan ser diseñados, sino que ya están en la definición de la transformación.

Por otro lado, pensando en el código, habrá mucho menos que escribir ya que una gran cantidad de ese código será generado a partir del **PIM**. Además, los desarrolladores pueden enfocarse en el **PIM** en vez de hacerlo en el código. Esto hace que el sistema desarrollado se acerque más a las necesidades del usuario final, el cual tendrá mejor funcionalidad en menor tiempo. Por último, habrá una mejora en la implementación de modificaciones ya que éstas se realizarán en el **PIM**; por lo tanto, introducir modificaciones sobre los modelos será mucho más rápido y seguro que hacerlo sobre el código final.

Estas mejoras en la productividad se pueden alcanzar con el uso de herramientas que automaticen completamente la generación de un **PSM** a partir de un **PIM**.

8.7.1.2. Portabilidad

En **MDD**, el mismo **PIM** puede ser automáticamente transformado en varios **PSMs** para diferentes plataformas. Además, todo lo que se especifica en el nivel de un **PIM** es completamente portable, solo depende de las herramientas de transformación disponibles. Para las plataformas más populares, un gran número de herramientas ya se encuentran disponibles. Para las plataformas menos utilizadas, puede emplearse una herramienta que soporte *plug-ins* para definición de transformaciones y, entonces, escribir uno mismo la definición de la transformación. Para tecnologías nuevas y plataformas a futuro, la industria del software necesitará entregar la transformación correspondiente a tiempo. Esto permitirá desarrollar rápidamente sistemas nuevos con la nueva tecnología, basados en nuestros **PIMs** ya existentes.

8.7.1.3. Interoperabilidad

Se pueden generar muchos **PSMs** a partir del mismo **PIM**, y éstos pueden estar relacionados. En **MDD** estas relaciones se llaman *bridges* o puentes. Los **PIMs** se transformarán en múltiples **PSMs**, los cuales frecuentemente están relacionados.

Por ejemplo, se podría transformar el mismo **PIM** a dos plataformas distintas modeladas con dos **PSMs**. Para cada elemento del primer **PSM** se puede conocer desde qué elemento del **PIM** fue generado. Desde el **PIM** sabemos qué elementos se generan en el segundo **PSM**. Por lo tanto, se puede deducir cómo se relacionan los elementos del primer **PSM** con los elementos del segundo **PSM**. Ya que conocemos todos los detalles específicos de ambas plataformas (de los dos **PSMs**) podemos generar el *bridge* entre los dos **PSMs**.

8.7.1.4. Mantenimiento y Documentación

Con **MDD** los desarrolladores pueden enfocarse solamente en el **PIM**, el cual tiene un nivel más abstracto que el código. El **PIM** es utilizado, entonces, para generar **PSMs** que, posteriormente, serán la base para generar código. Por lo tanto, el modelo será una exacta representación del código. Así el **PIM** cumplirá la función de documentación de alto nivel necesaria en cualquier sistema de software. La gran diferencia con el proceso tradicional de desarrollo de software es que el **PIM** no es abandonado luego de ser escrito. Los cambios a ser realizados sobre el sistema deberán ser hechos cambiando el **PIM** y regenerando el **PSM** y el código.

Actualmente en la práctica, muchos cambios se realizan en el **PSM** y el código se regenera desde allí. Las buenas herramientas, sin embargo, permitirán mantener la relación entre **PIM** y **PSM**, aun cuando se modifique el **PSM**. Estos

cambios se reflejarán en el **PIM**, y la documentación de alto nivel quedará consistente con el código real.

8.8. Propuestas Concretas para MDD

Dos de las propuestas concretas más conocidas y utilizadas en el ámbito de **MDD** son, por un lado ***Model Driven Architecture* (MDA)** [MDA] desarrollada por el ***Object Management Group* (OMG)** [OMG] y por otro lado el ***Domain-specific modeling* (DSM)** acompañado por los ***Domain-Specific Language* (DSL)**. Ambas iniciativas guardan naturalmente una fuerte conexión con los conceptos básicos de **MDD**. Específicamente, **MDA** tiende a enfocarse en lenguajes de modelado basados en estándares del **OMG**, mientras que **DSM** utiliza otras notaciones no estandarizadas para definir sus modelos [PGP09].

8.8.1. Arquitectura de Software Dirigida por Modelos

MDA es una de las iniciativas más conocida y extendida dentro del ámbito de **MDD**. **MDA** es un concepto que fue promovido por el **OMG** en noviembre de 2000, con el objetivo de afrontar los desafíos de integración de las aplicaciones y los continuos cambios tecnológicos.

MDA es una arquitectura que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos, siguiendo el proceso **MDD**. En **MDA**, la funcionalidad del sistema es definida en primer lugar como un **PIM** a través de un lenguaje específico para el dominio del que se trate. En este punto aparece además un tipo de modelo existente en **MDA**, no mencionado por **MDD**: el ***Platform Definition Model* (PDM)**. Éste especifica el metamodelo de la plataforma destino. Entonces, Dado un **PDM** correspondiente a CORBA, .NET, Web, etc., el modelo **PIM** puede traducirse a uno o más **PSMs** para la implementación correspondiente, usando diferentes lenguajes específicos del dominio, o lenguajes de propósito general como JAVA, C#, Python, etc.

MDA está relacionada con múltiples estándares, tales como el **UML**, **Meta-Object Facility (MOF)**, **XML Metadata Interchange (XMI)**, **Enterprise Distributed Object Computing (EDOC)**, **Software Process Engineering Metamodel (SPEM)** y **Common Warehouse Metamodel (CWM)**.

Cumpliendo con las directivas del **OMG**, las dos principales motivaciones de **MDA** son la interoperabilidad (independencia de los fabricantes a través de estandarizaciones) y la portabilidad (independencia de la plataforma) de los sistemas de software; las mismas motivaciones que llevaron al desarrollo de CORBA. Además, **OMG** postula como objetivo de **MDA** separar el diseño del sistema tanto de la arquitectura como de las tecnologías de construcción, facilitando así que el diseño y la arquitectura puedan ser alterados independientemente. El diseño alberga los requisitos funcionales (casos de uso, por ejemplo) mientras que la arquitectura proporciona la infraestructura a través de la cual se hacen efectivos los requisitos no funcionales como la escalabilidad, fiabilidad o rendimiento. **MDA** se asegura de que el modelo independiente de la plataforma (**PIM**), el cual representa un diseño conceptual que plasma los requisitos funcionales, sobreviva a los cambios que se produzcan en las tecnologías de fabricación y en las arquitecturas de software. Por supuesto, la noción de transformación de modelos en **MDA** es central. La traducción entre modelos se realiza normalmente utilizando herramientas automatizadas, es decir herramientas de transformación de modelos que soportan **MDA**. Algunas de ellas permiten al usuario definir sus propias transformaciones. [PGP09].

8.8.1.1. MDA y Perfiles UML

Cuando no se desea modificar la semántica de **UML**, sino sólo particularizar algunos de sus conceptos, no se necesita definir un nuevo lenguaje utilizando **MOF**. **UML** incluye un mecanismo de extensión en el propio lenguaje que permite definir lenguajes de modelado que son derivados de **UML**. Un perfil define una serie de mecanismos para extender y adaptar las metaclases de un metamodelo cualquiera a las necesidades concretas de una plataforma o de un dominio de aplicación.

UML es un lenguaje de propósito general que proporciona tanto flexibilidad como expresividad para modelar sistemas. No obstante, en ciertas aplicaciones es conveniente contar con un lenguaje más específico para representar los conceptos de ciertos dominios particulares. **OMG** define dos posibilidades cuando es necesario definir lenguajes específicos de un dominio particular. La primera es crear un nuevo lenguaje (por ejemplo, **CWM**), la otra es extender el **UML**.

Las razones para extender **UML** son varias: disponer de una terminología y vocabulario propios de un dominio de aplicación o de una plataforma de implementación concreta, definir una sintaxis para construcciones que no cuentan con una notación propia, definir una nueva notación para símbolos ya existentes, más acorde con el dominio de la aplicación objetivo, añadir cierta semántica que no aparece determinada de forma precisa en el metamodelo, añadir restricciones a las existentes en el metamodelo, restringiendo su forma de utilización, añadir información que puede ser útil a la hora de transformar el modelo a otros modelos, o a **IM**.

Un perfil **UML** es un mecanismo que proporciona **UML** para extender su sintaxis y semántica con el objetivo de expresar conceptos específicos de un dominio particular. A tal fin, define un conjunto de *estereotipos, restricciones y valores etiquetados.*

- *Estereotipos:* permiten crear nuevos tipos de elementos de modelado basados en elementos ya existentes en el metamodelo **UML**; estos extienden la semántica del metamodelo.
- *Restricciones:* impone condiciones a los nuevos elementos de modelado que han sido estereotipados.
- *Valores etiquetados:* un valor etiquetado es *un* metaatributo adicional que se asocia a una metaclase del metamodelo extendido por el perfil; permite agregar nueva información en la especificación del elemento.

En resumen, un perfil **UML** permite disponer de una terminología y vocabulario propios de un domino de aplicación o de una plataforma especifica; definir nuevas notaciones para símbolos preexistentes más acorde con el dominio que se está modelando; además en el marco **MDA**, permite añadir información para la transformación entre modelos o de éstos a código.

8.8.2. Grados y Métodos de Transformación de Modelos en MDA

Las transformaciones pueden utilizar diferentes combinaciones de transformaciones, tanto automáticas como manuales. Existen diferentes posibilidades para añadir, en un modelo, la información necesaria para transformar un **PIM** en uno o más **PSM**.

Cuatro diferentes enfoques pueden describirse para ilustrar el rango de posibilidades: transformación manual, transformación a **PIM** usando perfiles,

transformación usando patrones y marcados y, por último, transformación automática [MDA].

8.8.2.1. Transformación Manual

En la transformación manual, para obtener un **PSM** a partir de un **PIM**, deben tomarse decisiones de diseño que deberán efectuarse durante el proceso de desarrollo, en un diseño en particular, de modo de cumplir los requerimientos en la implementación. Este enfoque es útil en la medida que permite realizar las decisiones en el contexto de un diseño específico. Si bien este enfoque no tiene grandes diferencias de cómo se realizaron por años los buenos diseños de software, el enfoque **MDA** añade dos características importantes: la explícita distinción entre los **PIM** y los **PSM,** producto de la transformación y el registro de dicho transformación.

8.8.2.2. Transformación Usando Perfiles UML

En la transformación usando perfiles **UML**, un **PIM** puede ser ampliado usando un perfil independiente de la plataforma. Este modelo puede ser, luego, transformado a un **PSM** usando un segundo perfil **UML** específico de la plataforma. El mecanismo de extensión de **UML** puede incluir la especificación de operaciones, por lo tanto las reglas de transformación pueden ser especificadas usando operaciones y, de ese modo, permitir la especificación de transformaciones mediante perfiles **UML.**

8.8.2.3. Transformación Usando Patrones y Marcas

En la transformación usando patrones y marcas, los patrones pueden ser usados en la especificación del *mapping*, éstos incluyen un patrón y marcas que corresponden a algún elemento del patrón. En transformaciones de instancias de modelos, las marcas específicas se usan para preparar un **PIM** marcado. Los elementos marcados del **PIM** son transformados de acuerdo al patrón para producir el **PSM**. Se pueden combinar varios patrones para producir un nuevo patrón. Nuevas marcas pueden entonces ser especificadas para usar con el nuevo patrón.

En la transformación de tipos de modelos, las reglas especificarán que todos los elementos en el **PIM** los cuales concuerdan con un particular patrón, serán transformados en instancias de otro patrón en el **PSM.**

8.8.2.4. Transformación Automática

Existen contextos en donde un **PIM** puede proveer toda la información necesaria para la implementación y, por lo tanto, no hay necesidad de adicionar marcas o usar perfiles adicionales para poder generar un **IM**. En estos casos es posible, para un desarrollador de aplicaciones, construir un **PIM** que será completo en cuanto a la clasificación, estructura, invariantes, pre y post condiciones. El desarrollador puede entonces especificar el comportamiento requerido directamente en el modelo mediante un lenguaje de acción (*action Language*), de modo tal que el **PIM** contendrá toda la información necesaria para producir código ejecutable.

En este contexto, el desarrollador no precisa conocer detalles del **PSM** ni es necesario agregar información en el **PIM** ya que la información está disponible en la herramienta de transformación que transforma directamente a programa ejecutable.

8.8.3. Modelado Específico del Dominio

La iniciativa **DSM** propone un enfoque donde se crean modelos para un dominio, utilizando un lenguaje focalizado y especializado para dicho dominio. Estos lenguajes, denominados **DSLs**, permiten especificar la solución usando directamente conceptos del dominio del problema. Los productos finales son luego generados desde estas especificaciones de alto nivel. Esta automatización es posible si ambos, el lenguaje y los generadores, se ajustan a los requisitos de un único dominio.

Definimos como dominio a un área de interés para un esfuerzo de desarrollo en particular. En la práctica, cada solución **DSM** se enfoca en dominios pequeños porque el foco reductor habilita mejores posibilidades para su automatización y estos dominios pequeños son más fáciles de definir. Usualmente, las soluciones en **DSM** son usadas en relación a un producto particular, una línea de producción, un ambiente específico, o una plataforma.

El desafío de los desarrolladores y empresas se centra en la definición de los lenguajes de modelado, la creación de los generadores de código y la implementación de *Framework* específicos del dominio, elementos claves de una solución **DSM**. Estos elementos no se encuentran demasiado distantes de los elementos de modelado de **MDD**. Actualmente puede observarse que las discrepancias entre **DSM** y **MDD** han comenzado a disminuir. Podemos comparar el uso de modelos así como la construcción de la infraestructura respectiva en **DSM** y en **MDD**. En general, **DSM** usa los conceptos dominio, modelo, metamodelo, meta-metamodelo como **MDD**, sin mayores cambios y propone la automatización en el ciclo de vida del software [PGP09].

8.9. Resumen del Capítulo

El objetivo de este capítulo fue presentar, primeramente, una visión general del enfoque de la arquitectura de software dirigida por modelos. Luego, se describieron los diferentes niveles de madurez de los modelos. Posteriormente, se detallaron los diferentes modelos usados y los tipos básicos de transformaciones entre ellos. Luego, se describen las ventajas de la arquitectura de software dirigida por modelos sobre el proceso tradicional de desarrollo de software; posteriormente, se especificaron las diferentes propuestas del enfoque dirigida por modelos. Por último, se detallaron los distintos grados y métodos de transformación de modelos.

Capítulo 9

Lenguajes para la Transformación de Modelos

9.1. Introducción

Una transformación entre modelos toma, como entrada, un modelo y produce, como salida, otro modelo; esto puede interpretarse como un programa de computadora. Por lo tanto, las transformaciones podrían describirse (es decir, implementarse) utilizando cualquier lenguaje de programación. Sin embargo, para simplificar la tarea de codificar transformaciones se han desarrollado lenguajes de más alto nivel (o específicos del dominio de las transformaciones) tanto para transformaciones modelo a modelo (M2M) como transformaciones modelo a texto M2Text) [PGP09].

En este capítulo se analizarán, primeramente, los principales mecanismos existentes para la definición de transformaciones modelo a modelo; a continuación se introducirá el lenguaje ***Querry View Transformation (QVT)***, un estándar para transformaciones especificado por el **OMG**; luego, se analizarán algunos de los principales requisitos para que un lenguaje de transformaciones M2M sea práctico y usable; a continuación, se presentarán las principales características de las transformaciones modelo a texto, en particular, se introducirá el lenguaje estándar MOF2Text para transformaciones M2Text. Por último, se detallarán las principales herramientas, tanto de transformación M2M, como de transformación M2Text.

9.2. Visión General del Proceso de Transformación

El proceso de desarrollo de sistemas mediante el enfoque **MDD** muestra el rol de varios modelos: **PIM**, **PSM** y **IM**. Una herramienta que soporte **MDD**, toma un **PIM** como entrada y lo transforma en un (o varios) **PSM**. La misma herramienta u otra, tomará el **PSM** y lo transformará a código. Estas transformaciones son esenciales en el proceso de desarrollo de **MDD**. En la Figura 9.1 se muestra la herramienta de transformación como una caja negra, ésta toma un modelo de entrada y produce otro modelo como salida [PGP09].

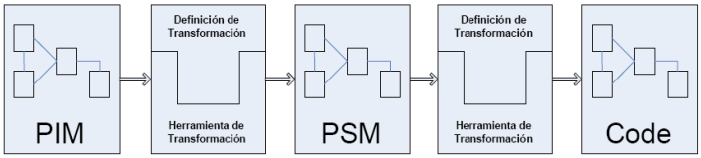


Figura 9.1. Transformaciones Dentro de las Herramientas de Transformación [PGP09]

Si analizáramos la herramienta de transformación y mirásemos dentro, podríamos ver qué elementos están involucrados en la ejecución de la transformación. En algún lugar dentro de la herramienta hay una definición que describe cómo se debe transformar el modelo de entrada para producir el modelo destino. Esta es la definición de la transformación. La Figura 9.1 muestra la estructura de la herramienta de transformación. Notemos que hay una diferencia entre la transformación misma, que es el proceso de generar un nuevo modelo a partir de otro modelo, y la definición de la transformación.

Para especificar la transformación, (que será aplicada muchas veces, independientemente del modelo fuente al que será aplicada) se relacionan construcciones de un lenguaje fuente con construcciones de un lenguaje destino. Se podría, por ejemplo, definir una transformación que relacionara elementos de **UML** a elementos Java, la cual describiría cómo los elementos Java pueden ser generados a partir de cualquier modelo **UML**. Esta situación se muestra en la Figura 9.2. En general, se puede decir que una definición de transformación consiste en una colección de reglas, las cuales son especificaciones no ambiguas de las formas en que un modelo (o parte de él) puede ser usado para crear otro modelo (o parte de él).

Figura 9.2. Definición de Transformaciones entre Lenguajes [PGP09].

Una transformación entre modelos puede verse como un programa de computadora que toma un modelo como entrada y produce un modelo como salida. Por lo tanto, como hemos comentado, las transformaciones podrían describirse (es decir, implementarse) utilizando cualquier lenguaje de programación. Sin embargo, para simplificar la tarea de codificar transformaciones se han desarrollado lenguajes de más alto nivel (o específicos del dominio de las transformaciones) para tal fin, tales como ***ATLAS Transformation Language* (ATL)** [ATL] y **QVT** [QVT].

Las siguientes definiciones fueron extraídas de [KWB03]:

- Una transformación es la generación automática de un modelo destino desde un modelo fuente, de acuerdo a una definición de transformación.
- Una definición de transformación es un conjunto de reglas de transformación que, juntas, describen cómo un modelo en el lenguaje fuente puede ser transformado en un modelo en el lenguaje destino.
- Una regla de transformación es una descripción de como una o más construcciones en el lenguaje fuente pueden ser transformadas en una o más construcciones en el lenguaje destino.

Analizaremos los principales mecanismos existentes para la definición de transformaciones modelo a modelo. Discutiremos sobre sus usos, presentaremos los requisitos necesarios que los lenguajes de estas características deberían satisfacer y finalmente introduciremos **QVT**, el estándar para transformaciones especificado por el **OMG**.

9.3. Transformaciones Modelo a Modelo

En la categoría de transformaciones modelo a modelo (M2M) podemos distinguir varios mecanismos para definirlas. En [CH06] se propone una posible taxonomía para estos mecanismos de transformación. Cada mecanismo específico puede corresponder a más de una clase en la taxonomía. Las siguientes son las principales propuestas a considerar:

9.3.1. Manipulación Directa

En la manipulación directa, los usuarios tienen que implementar tanto las reglas de transformación como de organización, así cómo manejar trazas y otras facilidades desde cero, en un lenguaje de programación de propósito general como Java.

9.3.2. Propuesta Operacional

La propuesta operacional es similar a la manipulación directa pero ofrece soporte más orientado a transformación de modelos. Una solución típica en esta categoría es extender la sintaxis de un lenguaje de modelado mediante el agregado de facilidades para expresar las transformaciones.

Por ejemplo, podemos extender un lenguaje de consultas como ***Object Constraint Language* (OCL)**, con constructores imperativos. Esta versión de **OCL** ampliado y ejecutable, resulta en un sistema de programación orientado a objetos propiamente dicho. Ejemplos de sistemas en esta categoría son el lenguaje *Operational Mappings* de **QVT**, MTL [MTL] y Kermeta [Kermeta]. Algunos de estos lenguajes brindan otras facilidades específicas, tales como mecanismos para *rastreo de transformaciones*.

9.3.3. Propuesta Relacional

Esta categoría agrupa propuestas en las cuales el concepto principal está representado por las relaciones matemáticas. En general, las propuestas relacionales pueden verse como una forma de resolución de restricciones. Ejemplos de propuestas relacionales son el lenguaje Relations de **QVT**, MTF [MTF], *Kent Model Transformation Language* [AK02] y Tefkat [LS05], entre otros.

Básicamente, una transformación se especifica a través de la definición de restricciones sobre los elementos del modelo fuente y el modelo destino. Sin bien son declarativas, estas transformaciones relacionales pueden tener semántica ejecutable implementada con programación lógica, usando *matching* basado en unificación, *backtracking*, etc. Las propuestas relacionales soportan naturalmente reglas multi-direccionales.

9.3.4. Propuesta Basada en Transformaciones de Grafos

Esta categoría reúne a las propuestas de transformación de modelos basadas en la teoría de transformaciones de grafos. En particular, este tipo de transformaciones actúan sobre grafos tipados, etiquetados y con atributos, que

pueden pensarse como representaciones formales de modelos de clase simplificados. Las principales propuestas en esta categoría incluyen, entre otros, a AGG [Tae03], AToM3 [AToM3], VIATRA [CHM+02] y MOLA [MOLA]. AGG y AToM3 son sistemas que implementan directamente la propuesta teórica para grafos con atributos y también las transformaciones sobre estos grafos.

Las reglas de transformación son unidireccionales e *in-place*. Cada regla de transformación de grafo consta de dos patrones, el derecho y el izquierdo. La aplicación de una regla implica localizar un patrón izquierdo en el modelo y reemplazarlo por el correspondiente patrón derecho.

9.3.5. Propuesta Híbrida

Las propuestas híbridas combinan técnicas diferentes de las categorías anteriores. Las distintas propuestas pueden ser combinadas como componentes separados o, en un modo granularmente más fino, a nivel de reglas individuales.

QVT es una propuesta de naturaleza híbrida, con tres componentes separadas, llamadas *Relations*, *Operational mappings*, y *Core*. Ejemplos de combinación a nivel fino son **ATL** y YATL [Pat04].

9.4. El Estándar para Transformaciones de Modelos

La convocatoria **QVT** *RFP*, para la definición de un lenguaje capaz de expresar consultas, vistas y transformaciones sobre modelos en el contexto de la arquitectura de metamodelos **MOF** 2.0 derivó en una propuesta estándar para **QVT**.

Los requerimientos impuestos por la **OMG** para un lenguaje estándar para transformaciones, justifican el acrónimo **QVT**. *Query*: el lenguaje debe facilitar consultas *ad-hoc* para selección y filtrado de elementos de modelos. Generalmente se seleccionan elementos de modelos que son la fuente de entrada de una transformación. *View*: el lenguaje debe permitir la creación de vistas de metamodelos **MOF**, sobre los que se define la transformación. *Transformation*: el lenguaje debe permitir la definición de transformaciones, a través de relaciones entre un metamodelo **MOF** fuente F, y un metamodelo **MOF** destino D; las transformaciones son usadas para generar un modelo destino, instancia que conforma con el metamodelo D, desde un modelo fuente, instancia que conforma con el metamodelo F.

La especificación de **QVT** tiene una naturaleza híbrida, relacional (o declarativa) y operacional (o imperativa). Comprende tres diferentes lenguajes M2M: dos lenguajes declarativos llamados *Relations* y *Core*, y un tercer lenguaje, de naturaleza imperativa, llamado *Operational Mappings*. La naturaleza híbrida fue introducida para cubrir diferentes tipos de usuarios con diferentes necesidades, requisitos y hábitos.

La especificación de **QVT** define tres paquetes principales, uno por cada lenguaje definido: *QVTCore*, *QVTRelation* y *QVTOperational*. El paquete *QVTBase* define estructuras comunes para transformaciones. El paquete *QVTRelation* usa expresiones de patrones *template* definidas en el paquete *QVTTemplateExp*. El paquete *QVTOperational* extiende al *QVTRelation*, dado que usa el mismo *framework* para trazas. Usa también las expresiones imperativas definidas en el paquete *ImperativeOCL*. Todos los paquetes **QVT** dependen del paquete *EssentialOCL* de **OCL** 2.0 [OCL], y a través de él también dependen de **Essential Meta-Object Facility (EMOF)** (*EssentialMOF*) [MOF].

9.4.1. QVT Declarativo

La parte declarativa de **QVT** está dividida en una arquitectura de dos niveles. Las capas son: a) un lenguaje *Relations,* amigable para el usuario, que soporta *pattern matching* complejo de objetos y la creación de *template* para objetos. Las trazas entre elementos del modelo involucrados en una transformación se crean implícitamente. Soporta propagación de cambios, ya que provee un mecanismo para identificar elementos del modelo destino y b) un lenguaje *Core* definido usando extensiones minimales de **EMOF** y **OCL**. Las trazas no son automáticamente generadas, se definen explícitamente como modelos **MOF**, y pueden crearse y borrarse como cualquier otro objeto. El lenguaje Core no soporta *pattern matching* para los elementos de modelos. Esta propuesta absolutamente mínima lleva a que el Core sea el "*assembler*" de los lenguajes de transformación.

9.4.1.1. Lenguaje *Relations*

El lenguaje *Relations* es una especificación declarativa de las relaciones entre modelos **MOF**. En este lenguaje, una transformación entre modelos se especifica como un conjunto de relaciones que deben establecerse para que la transformación sea exitosa. Los modelos tienen nombre, y los elementos que contienen deben ser de tipos correspondientes al metamodelo que referencian. Por ejemplo:

```
transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS)
{
       …
}
```

En esta declaración llamada "umlRdbms," hay dos modelos tipados: "uml" y "rdbms". El modelo llamado "uml" declara al paquete *SimpleUML* como su metamodelo, de la misma forma, el modelo "rdbms" declara al paquete *SimpleRDBMS* como su metamodelo. Una transformación puede ser invocada para *chequear* consistencia entre dos modelos o para modificar un modelo *forzando* consistencia. Cuando se fuerza la consistencia, se elige el modelo destino; éste puede estar vacío o contener elementos a ser relacionados por la transformación. Los elementos que no existan serán creados para forzar el cumplimiento de la relación.

Por ejemplo, en la relación *PackageToSchema*, el dominio para el modelo "uml" está marcado como *checkonly* y el dominio para el modelo "rdbms" está marcado *enforce*. Estas marcas habilitan la modificación (creación/borrado) de elementos en el modelo "rdbms", pero no en el modelo "uml", el cual puede ser solamente leído pero no modificado.

```
relation PackageToSchema
/* transforma cada paquete UML a un  esquema relacional */
{
       checkonly domain uml p: Package {name = pn}
       enforce domain rdbms s: Schema {name = pn}
}
```

9.4.2. Relaciones, Dominios y *Pattern Matching*

Las relaciones en una transformación declaran restricciones que deben satisfacer los elementos de los modelos. Una relación se define por dos o más dominios y un par de predicados *when* y *where*, que deben cumplirse entre los elementos de los modelos. Un dominio es una variable con tipo que puede tener

su correspondencia en un modelo de un tipo dado. Un dominio tiene un patrón, que se puede ver como un conjunto de variables y un conjunto de restricciones que los elementos del modelo asocian a aquellas variables que lo satisfacen (*pattern matching*). Un patrón del dominio es un *template* para los objetos y sus propiedades que deben ser inicializadas, modificadas, o creadas en el modelo para satisfacer la relación.

Por ejemplo, en la relación *PackageToSchema* se declaran dos dominios que harán correspondencia entre elementos de los modelos "uml" y "rdbms" respectivamente. Cada dominio especifica un patrón simple: un paquete con un nombre, y un esquema con un nombre, en ambos la propiedad "*name*" asociada a la misma variable "pn" implica que deben tener el mismo valor. A su vez, en la relación *ClassToTable*, se declaran también dos dominios. Cada dominio especifica patrones más complejos que en la otra relación: por un lado una clase con un espacio de nombres, de tipo persistente y con un nombre. Por otro lado, el otro dominio especifica una tabla con su esquema, un nombre y una columna cuyo nombre se formará en base al nombre de la clase **UML** y define a esta columna como clave primaria de la tabla. Por la propiedad "*name*" asociada a la misma variable "cn", la clase y la tabla deben tener el mismo nombre. Además la cláusula *when* en la relación indica que, previamente a su aplicación, debe satisfacerse la relación *PackageToSchema*, mientras que la cláusula *where* indica que siempre que la relación *ClassToTable* se establezca, la relación *AttributeToColumn* debe establecerse también.

```
relation PackageToSchema{
/* map each package to a schema */

        domain uml p: Package {name = pn}
        domain rdbms s: Schema {name = pn}
        }

        relation ClassToTable
        /* map each persistent class to a table */
        {

        domain uml c: Class { namespace = p: Package {},
               kind = 'Persistent', name = cn
               }

        domain rdbms t: Table { schema = s: Schema {},
               name = cn, column = cl: Column {
               name = cn + '_tid', type = 'NUMBER'},
               primaryKey = k:PrimaryKey { name = cn + '_pk',
               column = cl }
               }

        when { PackageToSchema(p, s);}
        where { AttributeToColumn(c, t); }
}
```

9.4.3. Relaciones *Top Level*

Una transformación puede definir dos tipos de relaciones: *top Level* y no *top Level*. La ejecución de una transformación requiere que se puedan aplicar todas sus relaciones *top Level*, mientras que las no *top Level* se deben cumplir solamente cuando son invocadas, directamente o a través de una cláusula *where* de otra relación:

```
transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {
        top relation PackageToSchema {…}
        top relation ClassToTable {…}
```

```
        relation AttributeToColumn {…}
}
```

Una relación *top Level* tiene la palabra *top* antecediéndola para distinguirla sintácticamente. En el ejemplo ya citado, *PackageToSchema* y *ClassToTable* son relaciones *top Level*, mientras que *AttributeToColumn* es una relación no *top Level*, que será invocada por alguna de las otras para su ejecución. Una expresión *object template* provee una *plantilla* para crear un objeto en el modelo destino. Cuando para un patrón válido en el dominio de entrada no existe el correspondiente elemento en la salida, entonces la expresión *object template* es usada como *plantilla* para crear objetos en el modelo destino.

Por ejemplo, cuando *ClassToTable* se ejecuta con el modelo destino "rdbms", la siguiente expresión *object template* sirve como *plantilla* para crear objetos en el modelo destino "rdbms":

```
t:Table {
        schema = s: Schema {},
                name = cn,
                column = cl: Column {name = cn+'_tid', type =
        'NUMBER'},
                primaryKey = k : PrimaryKey {name=cn + '_pk',
                column = cl}
}
```

El *template* asociado con *Table* especifica que un objeto de tipo Tabla debe ser creado con las propiedades "schema", "name", "column" y "primaryKey" con los valores especificados en la expresión *template*. Similarmente, los *templates* asociados con *Column y PrimaryKey,* especifican cómo sus respectivos objetos deben ser creados.

9.5. Definición Formal de las Transformaciones Usando QVT

Detallamos, a continuación, un ejemplo de transformación entre modelos utilizando el lenguaje estándar **QVT**. La transformación "*Modelo de Datos a Modelo de Datos Temporal*" fue inicialmente presentada, de manera informal, en el capítulo 5.

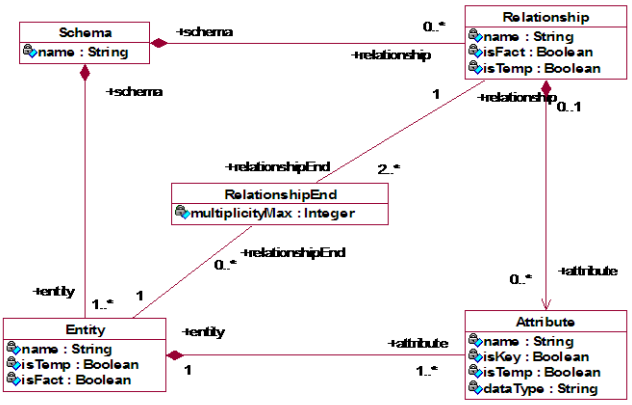


Figura 9.3. Metamodelo de Datos

Presentamos los metamodelos utilizados y, en particular, la transformación **`TemporalAttributeToTemporalEntity{}`**; primeramente mediante una *relation* y, luego, gráficamente mediante la representación diagramático. En la Figura 9.3

se muestra el metamodelo de datos y en la Figura 9.4 el metamodelo de datos temporal.

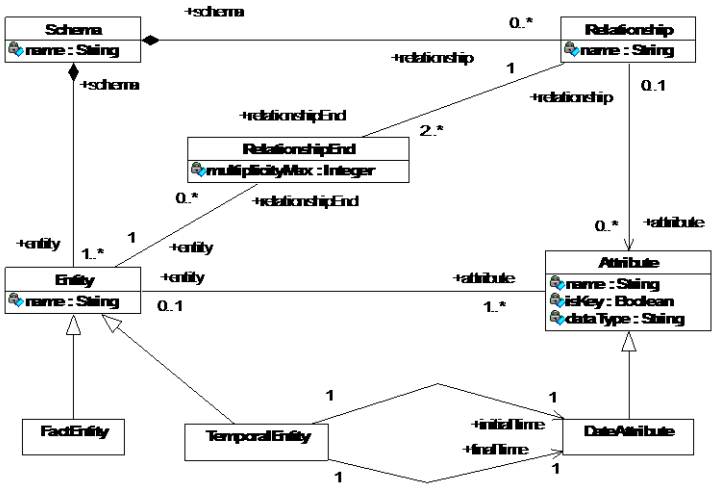


Figura 9.4. Metamodelo de Datos Temporal

A continuación, se detalla la transformación **`transformation dbToTdb`** y, en particular, la *relation* **`TemporalAttributeToTemporalEntity{}`**:

```
transformation dbToTdb (db: Model, tdb: TemporalModel){
       key Entity (name, schema);
       key Attribute(name, entity);

/* …
relation TemporalAttributeToTemporalEntity{};
}
/* …

/*
* Transformación de un modelo conceptual de datos a un modelo
* conceptual de datos temporal
*/

transformation dbToTdb (db:Model, tdb:TemporalModel)
{
       key Entity (name, schema);
       key Attribute(name, entity);

top relation SchemaToSchema
{
n: String;

checkonly domain db p: Schema(name=n);
enforce domain tdb s: Schema(name=n);

/*
 * Los atributos temporales se transformarán como
* entidades temporales
*/

relation TemporalAttributeToTemporalEntity
{
ent, t, id    : String;
n             : Integer;

checkonly domain db e: Entity{
       schema = e: Schema{},
       name = ent,
```

```
            isRoot = 'false' ,
            attribute = k: Attribute{
                  name = id,
                  isKey = ´true´,
                  dataType = dt: Datatype{}
                  },
            attribute = a: Attribute{
                  isKey = 'false',
                  isTemporal = 'true',
                  name = t,
                  dataType = dt: Datatype{}
      };

      enforced domain tdb te: TemporalEntity{
            schema = s: Schema{},
            name = t +'-T',
            isRoot = 'false' ,
            attribute = a: Attribute{
                  isKey = 'false',
                  isTemporal = 'false',
                  name = t,
                  dataType = dt: Datatype{}
                  },
            finalTime = ft: DateAttribute{
                  isKey = 'false',
                  isTemporal = 'false',
                  name = 'finalTime',
                  dataType = dt: Datatype{
                        name = 'Date'}
                  },
            attribute = k: Attribute{
                  isKey = 'true',
                  name = id,
                  dataType = dt: Datatype{},
                  },
            initialTime = it: DateAttribute{
                  isKey = 'true',
                  isTemporal = 'false',
                  name = 'initialTime',
                  dataType = dt: Datatype{
                        name = 'Date'}
            relationshipEnd = rse1: RelationshipEnd{
                  multiplicity = 1,
                  relationship = r: Relationship{
                        isTemporal= 'true',
                        isRoot = 'false' ,
                        name = 'T'}
                  },
                  relationshipEnd = rse2: RelationshipEnd{
                        multiplicity = n},
                  entity = e: Entity{}
                        };
      }
}
```

En la Figura 9-5 se detalla la descripción diagramático, que muestra la misma información pero en forma gráfica, de la *relation* **TemporalAttributeToTemporalEntity()**.

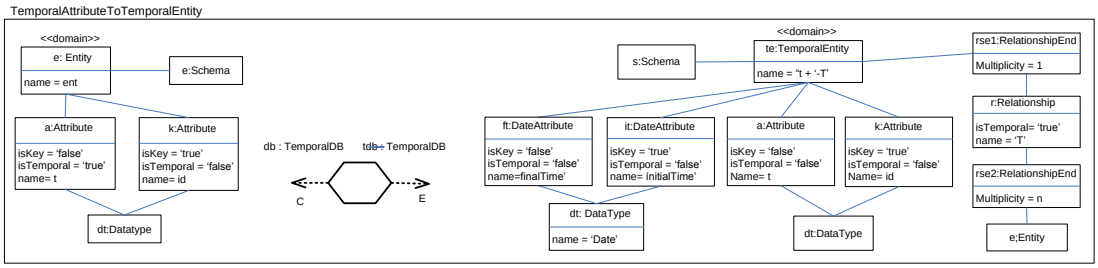


Figura 9.5. Representación Diagramática

9.6. Transformaciones Modelo a Texto

Respecto a la distinción entre transformaciones modelo a modelo (M2M) con transformaciones modelo a texto (M2T), podemos destacar que la principal diferencia es que, mientras la transformación modelo a modelo crea su modelo destino como una instancia de un metamodelo específico, el destino de una transformación modelo a texto es simplemente un documento en formato textual, generalmente de tipo *String*.

9.6.1. Características de los Lenguajes Modelo a Texto

No sólo dentro de la propuesta **MDD**, sino también en áreas tradicionales de la ingeniería de software como programación y definición de lenguajes, contar con lenguajes para transformaciones modelo a texto nos permite [PGP09]:

- **Elevar el nivel de abstracción**: dado que los sistemas se tornan más complejos, está comprobado, en la historia de los lenguajes de programación, que elevar el nivel de la abstracción tiene gran utilidad (por ejemplo el pasaje de lenguajes *Assembler* a lenguajes de alto nivel como *Cobol*).
- **Automatizar el proceso de desarrollo de software en sus últimas etapas**: en particular decrementar el tiempo de desarrollo, incrementar la calidad del software y poner el foco en la parte creativa del software.
- **Generar automáticamente nuevos artefactos desde nuestros modelos**. Por ejemplo: obtener código Java, EJB, JSP, C#, o bien *Scripts* de **SQL**, texto **HTML**, casos de test y documentación de modelos.

Existen varias alternativas que podríamos considerar para alcanzar estos objetivos, por ejemplo: lenguajes de programación de propósito general (como Java), lenguajes de *Template/scripting* (por ejemplo XSLT, Velocity Template Language, Eclipse Java Emitter Templates - JET), Lenguajes de transformación de modelos (por ejemplo **ATL**), lenguajes propietarios basados en **UML** y propuestas basadas en **DSL**, entre otras.

Sin embargo, estas alternativas tienen ciertas desventajas. Por un lado, los lenguajes de propósito general como los de *scripting* no permiten definir los metamodelos fuente para transformaciones (lo mismo ocurre para transformaciones M2M, no obstante, en Java esto puede hacerse usando **EMF** [EMF]). Por otra parte, los lenguajes para transformación modelo a modelo, como **ATL**, están basados en metamodelos pero no están diseñados pensando en la generación de texto. Podría lograrse, pero en forma más complicada, definiendo también el metamodelo de salida, lo cual no es necesario para este tipo de transformación.

Con respecto a las herramientas para **UML**, la transformación de los modelos a código está implementada de manera fija dentro de la herramienta lo cual la torna poco adaptable. Por su parte, los **DSLs** proveen la flexibilidad de las herramientas basadas en metamodelo, pero deben adaptar la generación de código para cada lenguaje específico del dominio.

Estas razones son algunas de las más importantes que justifican por qué es conveniente definir lenguajes específicos para soportar transformaciones M2T.

9.6.2. Requisitos de un Lenguaje Modelo a Texto

Siguiendo un proceso similar al que precedió a la aparición del estándar para transformaciones modelo a modelo **QVT**, el **OMG** publicó un *Request for Proposal* (*RFP*) para transformaciones modelo a texto en Abril de 2004 [OMG04]. Este *RFP* definía requisitos específicos que las propuestas enviadas debían satisfacer. Algunos requisitos eran obligatorios y otros opcionales; los listamos a continuación:

9.6.2.1. Requisitos Obligatorios

- La generación de texto a partir de modelos **MOF** 2.0
- El reuso (si es posible) de especificaciones **OMG** ya existentes, en particular **QVT**
- Las transformaciones deberán estar definidas en el metanivel del modelo fuente
- Soporte para conversión a *String* de los datos del modelo.
- Manipulación de *strings*
- Combinación de los datos del modelo con texto de salida
- Soporte para transformaciones complejas
- Varios modelos **MOF** de entrada (múltiples modelos fuente)

9.6.2.2. Requisitos Opcionales:

- Ingeniería *round-trip* (de ida y vuelta)
- Detección/protección de cambios manuales, para permitir re-generación
- Rastreo de elementos desde texto, si es necesario para poder soportar los dos últimos requisitos.

Como propuesta inicial a este *RFP* M2Text del **OMG**, surge el lenguaje MOFScript (**MOF** Model to Text Transformation Language) [Old06] (ver Anexo III), que cuenta con una herramienta del mismo nombre. MOFScript participó en el proceso de estandarización dentro de **OMG**. El proceso de combinación con otras propuestas produjo como resultante final al estándar llamado MOF2Text. El objetivo del **OMG** al diseñar el estándar MOF2Text fue dar soporte a la mayoría de los requisitos listados arriba.

9.7. El Estándar Para Expresar Transformaciones Modelo a Texto

El lenguaje Mof2Text es el estándar especificado por el **OMG** para definir transformaciones modelo a texto. Presentaremos sólo una descripción general de sus elementos; en el documento de especificación del **OMG** [Mof2Text], se puede ver en detalle la descripción de su sintaxis concreta y su sintaxis abstracta (metamodelo).

9.7.1. Descripción General del Lenguaje

De la misma manera que el estándar **QVT** fue desarrollado para cubrir las necesidades para transformaciones M2M, el estándar Mof2Text fue creado para poder transformar un modelo a diversos artefactos textuales tales como código, especificaciones de implementación, informes, documentos, etc. Esencialmente, este estándar está pensado para transformar un modelo en una representación textual plana.

En Mof2Text una transformación es considerada una plantilla donde el texto que se genera desde los modelos se especifica como un conjunto de bloques de texto parametrizados con elementos de modelos. Estos elementos son extraídos de los modelos que van a completar los *placeholders* en la transformación. Los *placeholders* son expresiones especificadas a través de *queries*, mecanismos útiles para seleccionar y extraer valores desde los modelos. Estos valores son luego convertidos en fragmentos de texto usando un lenguaje de expresiones extendido con una librería de manipulación de *Strings*.

Por ejemplo, la siguiente especificación de transformación genera una definición Java para una clase **UML**.

```
[template public classToJava(c : Class)]
class [c.name/]
{
	// Constructor
	[c.name/]()
	{
	}
}
[/template]
```

Si aplicamos esta transformación a la clase "Empleado" instancia de la metaclase Entidad del metamodelo de datos (Figura 9.6), se genera el siguiente texto:

```
class Empleado
{
	// Constructor
	Empleado ()
	{
	}
}
```

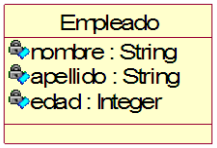


Figura 9.6. Clase Empleado

Como puede verse, el texto generado tiene una estructura similar a la especificación, preservando la indentación, los espacios en blanco, etc.

Una transformación puede invocar otras transformaciones. La invocación a una transformación es equivalente a colocar *in situ* el texto producido por la transformación invocada.

En el siguiente fragmento de Mof2Text, la transformación *classToJava* invoca a la transformación *attributeToJava* para cada atributo de la clase y agrega ';' como separador entre los fragmentos de texto producidos por cada invocación a *attributeToJava*:

```
[template public classToJava(c : Class)]
class [c.name/]
{
```

```
        // Attribute declarations
        [attributeToJava(c.attribute)/]

        // Constructor
        [c.name/]()
        {
        }
}
[/template]

[template public attributeToJava(a : Attribute)]
[a.type.name/] [a.name/];
[/template]
```

Para la clase Empleado, se genera el siguiente texto:

```
class Empleado
{
        // Attribute declarations
        String Nombre;
        String Apellido;
        int edad;

        // Constructor
        Empleado ()
        {
        }
}
```

En vez de definir dos transformaciones separadas, una transformación puede iterar sobre una colección usando el bloque *for*. El uso de este bloque mejora la legibilidad.

Por ejemplo la transformación *classToJava,* mostrada anteriormente, puede usar el bloque *for* de la siguiente manera:

```
[template public classToJava(c : Class)]
class [c.name/]
{
        // Attribute declarations
        [for(a : Attribute | c.attribute)]
        [a.type.name/] [a.name/];
        [/for]
        // Constructor
        [c.name/]()
        {
        }
}
[/template]
```

De forma similar puede utilizarse el bloque *if* como sentencia condicional, con la condición entre paréntesis.

Por ejemplo, dentro de un bloque *for* que itera sobre atributos de una clase **UML** y los convierte a atributos en una clase C++, si el tipo del atributo es complejo, al nombre del tipo se le agrega un '*':

```
[for(a : Attribute | c.attribute)]
[a.type.name/][if(isComplexType (a.type)]*[/if][a.name/];
[/for]
```

Una transformación puede tener una guarda o condición escrita en **OCL**, que decida cuando puede ser invocada.

Por ejemplo, la siguiente transformación *classToJava* es invocada solamente si la clase es concreta:

```
[template public classToJava(c : Class) (c.isAbstract = false)]
class [c.name/]
{
       // Attribute declarations
       [attributeToJava(c.attribute)/]
       // Constructor
       [c.name/]()
       {
       }
}
[/template]
```

Las navegaciones complejas sobre el modelo pueden ser especificadas a través de *queries*.

El siguiente ejemplo muestra el uso de un *query* llamado *allOperations()* que retorna, para una clase dada como parámetro, el conjunto de operaciones de la clase y de todas sus superclases abstractas:

```
[query public allOperations(c: Class) : Set ( Operation ) =
       c.operation-> union( c.superClass->
       select(sc|sc.isAbstract=true)-> iterate(ac : Class;
       os:Set(Operation) = Set{}| os->
       union(allOperations(ac)))) /]

[template public classToJava(c : Class) (c.isAbstract = false)]
class [c.name/]
{
// Attribute declarations
[attributeToJava(c.attribute)/]
// Constructor
[c.name/]()
{
}
[operationToJava(allOperations(c))/]
}

[/template]

[template public operationToJava(o : Operation)]
[o.type.name/] [o.name/] ([for(p:Parameter | o.parameter)
separator(',')] [p.type/] [p.name/] [/for]);
[/template]
```

El bloque *let* puede utilizarse para chequear si un elemento es de un cierto tipo y en ese caso, declarar una variable de ese tipo, que luego sea usada en la transformación:

```
[template public classToJava(c : Class)]
[let ac : AssociationClass = c ]
class [c.name/]
{
    // Attribute declarations
    [attributeToJava(c.attribute)/]
    // Constructor
    [c.name/]()
    {
    }

    // Association class methods
    [for (t:Type | ac.endType)]
    Attach_[t.name/]([t.name/] p[t.name/])
```

```
    {
    // Code for the method here
    }
    [/for]
}
[/let]
[/template]
```

En esta transformación, el bloque *let* verifica si el argumento real es de tipo *AssocationClass*. Si es así, declara una variable ac de tipo *AssociationClass* que puede ser usada dentro del bloque *let*. Si la verificación falla, el bloque *let* no produce ningún texto.

Las transformaciones pueden componerse para lograr transformaciones complejas. Pueden construirse transformaciones de mayor tamaño estructuradas en módulos con partes públicas y privadas.

Un *Module* puede tener dependencias '*import*' con otros módulos. Esto le permite invocar *Templates* y *Queries* exportados por los módulos importados.

Una transformación puede comenzar su ejecución invocando directamente a una transformación pública. No hay noción explícita de transformación *main*. Las transformaciones pueden sobre-escribirse. Un módulo puede extender a otro a través del mecanismo de herencia. El lenguaje solamente soporta herencia simple.

9.7.2. Texto Explícito vs. Código Explícito

En la mayoría de los casos, el estilo de especificación de las transformaciones es intuitivo; la lógica de producción de texto se especifica en una forma delimitada por corchetes ('[' y ']'). Sin embargo, puede haber casos donde esta forma de escritura delimitada por corchetes y anidada, puede tornarse compleja. En este caso es más intuitivo especificar la estructura de la transformación sin usar delimitadores especiales. Esto se logra indicando el modo de escritura, es decir, *text-explicit* o *code-explicit;* text-explicit es el modo por defecto. La sintaxis es similar a las anotaciones Java: @text-explicit o @code-explicit.

Por ejemplo:

```
@text-explicit
[template public classToJava(c : Class)]
class [c.name/]
{
// Constructor
[c.name/]()
{
}
}
[/template]

@code-explicit
template public classToJava(c : Class)
'class 'c.name '
{
// Constructor
'c.name'()
{
}
}'
/template
```

En la forma *code-explicit*, en vez de la plantilla, se visualiza el texto de salida. Los bloques se cierran usando una barra seguida de la palabra clave del bloque (por ejemplo, /template).

9.7.3. Rastreo de Elementos Desde Texto (Traceability)

La generación de código basado en modelos es una de las principales aplicaciones de las transformaciones M2T. Mof2Text provee soporte para rastrear elementos de modelo desde partes de texto. Un bloque *Trace* relaciona texto producido en un bloque con un conjunto de elementos de modelo provistos como parámetros. El texto a ser rastreado debe estar delimitado por palabras clave especiales. Además, las partes de texto pueden marcarse como protegidas. Tales partes se preservan y no se sobrescriben por subsecuentes transformaciones M2T. El bloque *trace* puede incluir información de la transformación original. Las partes de texto deben poder relacionarse sin ambigüedad con un conjunto de elementos y deben tener una identificación única.

En el ejemplo siguiente, el bloque *trace* identifica el texto a ser rastreado relacionando el texto generado con el elemento de modelo 'c' de tipo *Class*. El bloque protegido identifica la parte de texto que debe preservarse durante subsecuentes transformaciones. Se producen delimitadores en el texto de salida para identificar claramente la parte protegida. Dado que los delimitadores son específicos del lenguaje destino, no están definidos en este estándar. Las herramientas de implementación serán las responsables de producir los delimitadores correctos para el lenguaje destino elegido:

```
[template public classToJava(c : Class)]
[trace(c.id()+ '_definition') ]
class [c.name/]
{
// Constructor
[c.name/]()
{
[protected('user_code')]
; user code
[/protected]
}
}
[/trace]
[/template]
```

9.7.4. Archivos de Salida

El bloque *file* especifica el archivo donde se envía el texto generado. Este bloque tiene tres parámetros: una url que denota el nombre del archivo, un *booleano* que indica cuando el archivo se abre en modo *append* (en este caso el parámetro tiene el valor *true*) o no (valor *false*), y un *id* único y opcional, generalmente derivado desde los identificadores de los elementos del modelo, que también se utilizan para las trazas. Por ejemplo, una herramienta de transformación puede usar el id para buscar un archivo que fue generado en una sesión previa o cuando el nombre del archivo fue modificado por el desarrollador de la transformación. El modo de apertura por defecto es *overwrite*.

El siguiente ejemplo especifica que la clase Java se guarda en un archivo cuya url se forma con 'file:\\' concatenado con el nombre de la clase seguido por '.java'. El archivo se abre en modo 'overwrite' (2do parámetro en *false*) y se le asigna un id, que es el id de la clase concatenado con 'impl'.

```
[template public classToJava(c : Class)]
[file ('file:\\'+c.name+'.java', false, c.id + 'impl')]
class [c.name/]
{
// Constructor
```

```
[c.name/]()
{
}
}
[/file]
[/template]
```

9.8. Herramientas de Soporte para Transformaciones de Modelos

Existen diversos enfoques que contribuyen brindando soluciones para implementar transformaciones de modelos: lenguajes, *frameworks*, motores, compiladores, etc. En su mayoría estas soluciones no son compatibles con la especificación del lenguaje estándar para transformaciones QVT. Esto se debe a que la versión 1.0 de QVT fue publicada en abril del año 2008 por lo cual las soluciones en base a dicha especificación se encuentran en un estado inicial.
Describimos, a continuación, las características de las herramientas más populares que soportan al proceso de definición de transformaciones de modelo a modelo y de modelo a texto [PGP09].

9.8.1. Herramientas de Transformación de Modelo a Modelo

Presentamos algunas de las herramientas más difundidas que permiten implementar transformaciones entre modelos:

- **ATL** (Atlas Transformation Language) [ATL] consiste en un lenguaje de transformación de modelos y un *toolkit* creados por ATLAS Group (INRIA y LINA) que forma parte del proyecto **GMT** de Eclipse. En él se plantea un lenguaje híbrido de transformaciones declarativas y operacionales de modelos que si bien es del estilo de **QVT** no se ajusta a las especificaciones ya que fue construido en respuesta a un *Request-For-Proposal* lanzado por la **OMG** para transformaciones de modelos. Aunque la sintaxis de **ATL** es muy similar a la de **QVT**, no es interoperable con este último. Tiene herramientas que realizan una compilación del código fuente a *bytecodes* para una máquina virtual que implementa comandos específicos para la ejecución de transformaciones. **ATL** forma parte del *framework* de gestión de modelos AMMA que se encuentra integrado en Eclipse y **EMF**. **ATL** posee un algoritmo de ejecución preciso y determinista. El *toolkit* es de código abierto. (Más detalles en Anexo II)

- **ModelMorf** [ModelMorf] es un motor de transformación desarrollado por Tata Consultancy Servicies. Funciona bajo línea de comandos para ejecutar transformaciones de modelos basados en la especificación **QVT** declarativo. Cumple parcialmente con la especificación de **QVT** de la **OMG** pero es propietaria, no es de código abierto.

- **Medini QVT** [Medini] es una herramienta de código abierto, construida por ikv++ *technologies* sobre las bases del proyecto OSLO [Oslo]. *Open Source Libraries* for **OCL** (OSLO) es un proyecto de la Universidad de Kent que provee una implementación para las clases del metamodelo de **OCL**. Medini incluye un conjunto de herramientas construidas para el diseño y ejecución de transformaciones declarativas de modelos. Define una estructura de clases que extienden a las librerías de OSLO para modelar transformaciones **QVT**. Pero además, Medini implementa un motor capaz de ejecutar especificaciones escritas en **QVT** declarativo.

- **SmartQVT** [SmartQVT] es un proyecto que nació en el departamento de I+D de France Telecom. Es un compilador de código abierto para transformaciones de modelos escritas en **QVT** operacional. En esencia, toma como entrada una transformación escrita en lenguaje **QVT** operacional y obtiene a partir de ella, una clase Java que implementa el comportamiento descrito en lenguaje **QVT**. Esta última, como toda clase Java, puede ser integrada en cualquier aplicación y ejecutada desde cualquier máquina virtual. Para realizar su trabajo, la herramienta implementa un *parser* y un compilador El *parser* lee código **QVT** y obtiene a partir de él, su metamodelo. El compilador toma el metamodelo y escribe el texto del código fuente de la clase Java.

- **Together Architecture/QVT** [Together] es un producto de Borland que integra la IDE de Java. Tiene sus orígenes en la integración de JBuilder con la herramienta de modelado UML. Together es un conjunto de plugins para la plataforma Eclipse. Permite a los arquitectos de software trabajar focalizados en el diseño del sistema de manera gráfica; luego la herramienta se encarga de los procesos de análisis, de diseño y de generar modelos específicos de la plataforma a partir de modelos independientes de la plataforma. El proceso de transformación de modelo a modelo se especifica usando **QVT** operacional y la transformación de modelos a texto se define directamente en Java (con JET). La herramienta provee un editor con resaltado y sugerencias al escribir el código; soporta la sintaxis de **OCL** y ofrece soporte para *debugging* y rastreo automático de las transformaciones.

- **Viatra** [CHM+ 02] (acrónimo de "*VIsual Automated model TRAnsformations*") es una herramienta para la transformación de modelos que actualmente forma parte del *framework* VIATRA2, implementado en lenguaje Java y se encuentra integrado en Eclipse. Provee un lenguaje textual para describir modelos y metamodelos, y transformaciones llamados VTML y VTCL respectivamente. La naturaleza del lenguaje es declarativa y está basada en técnicas de descripción de patrones, sin embargo es posible utilizar secciones de código imperativo. Se apoya en métodos formales como la transformación de grafos (GT) y la máquina de estados abstractos (ASM) para ser capaz de manipular modelos y realizar tareas de verificación, validación y seguridad, así como una temprana evaluación de características no funcionales como fiabilidad, disponibilidad y productividad del sistema bajo diseño. Como puntos débiles podemos resaltar que Viatra no se basa en los estándares **MOF** ni **QVT**. No obstante, pretende soportarlos en un futuro mediante mecanismos de importación y exportación integrados en el framework.

- **Tefkat** [LS05] es un lenguaje de transformación de modelos y un motor para la transformación de modelos. El lenguaje está basado en F-logic y la teoría de programas estratificados de la lógica. Tefkat fue uno de los subproyectos del proyecto Pegamento, desarrollado en *Distributed Systems Technology Centre* (DSTC) de Australia. El motor está implementado como un *plugin* de Eclipse y usa **EMF** para manejar los modelos basados en **MOF**, **UML** 2 y esquemas **XML.** Tefkat define un mapeo entre un conjunto de metamodelos origen en un conjunto de metamodelos destino. Una transformación en Tefkat consiste de reglas, patrones y plantillas. Las reglas contienen un término origen y un término destino. Los patrones son términos origen compuestos agrupados bajo un nombre, y los plantillas son términos destino compuestos agrupados bajo

un nombre. Estos elementos están basados en la F-logic y la programación pura de la lógica, sin embargo, la ausencia de símbolos de función significa una reducción importante en la complejidad. Tefkat define un lenguaje propio con una sintaxis concreta parecida a **SQL**, especialmente diseñado para escribir transformaciones reusables y escalables, usando un conceptos del dominio de alto nivel más que operar directamente con una sintaxis **XML**. El lenguaje Tefkat está definido en términos de **EMOF** (v2.0), y esta implementado en términos de Ecore. El lenguaje es muy parecido al lenguaje *Relations* de **QVT**.

- **Epsilon** [Epsilon] es una plataforma desarrollada como un conjunto de *plug-ins* (editores, asistentes, pantallas de configuración, etc.) sobre Eclipse. Presenta el lenguaje metamodelo independiente *Epsilon Object Language* que se basa en **OCL**. Puede ser utilizado como lenguaje de gestión de modelos o como infraestructura a extender con nuevos lenguajes específicos de dominio. Tres son los lenguajes definidos en la actualidad: *Epsilon Comparison Language* (ECL), *Epsilon Merging Language* (EML), *Epsilon Transformation Language* (ETL), para comparación, composición y transformación de modelos respectivamente. Se da soporte completo al estándar **MOF** mediante modelos **EMF** y documentos **XML** a través de JDOM.
  La finalidad pretendida con la extensión del lenguaje **OCL** es dar soporte al acceso de múltiple modelos, ofrecer constructores de programación convencional adicionales (agrupación y secuencia de sentencias), permitir modificación de modelos, proveer depuración e informe de errores, así como conseguir una mayor uniformidad en la invocación. Soporta mecanismos de herencia y rastreo e introduce otros para comprobación automática del resultado en composición y transformación de modelos.

- **AToM3** [Atom3] (acrónimo de "*A Tool for Multi-formalism and Meta-Modeling*") es una herramienta para modelado en muchos paradigmas bajo el desarrollo de MSDL (*Modelling Simulation and Design Lab*) en la escuela de ciencias de la computación de la universidad de McGill.
  Las dos tareas principales de AToM3 son metamodelado y transformación de modelos. El metamodelado se refiere a la descripción o modelado de diferentes clases de formalismos usados para modelar sistemas (aunque se enfoca en formalismos de simulación y sistemas dinámicos, las capacidades de AToM3 no se restringen a solo esos). Transformación de modelos se refiere al proceso automático de convertir, traducir o modificar un modelo en un formalismo dado en otro modelo que puede o no estar descrito en el mismo formalismo. En AToM3 los formalismos y modelos están descriptos como grafos, para realizar una descripción precisa y operativa de las transformaciones entre modelos. Desde una meta especificación (en un formalismo de entidad relación), AToM3 genera una herramienta para manipular visualmente (crear y editar) modelos descriptos en el formalismo especificado. Las transformaciones de modelos están ejecutadas por la reescritura de grafos. Las transformaciones pueden entonces ser expresadas declarativamente como modelos basados en grafos. Algunos de los metamodelos actualmente disponibles son: Entidad/Relación, GPSS, autómatas finitos determinísticos, autómatas finitos no determinísticos, redes de Petri y diagramas de flujo de datos. Las transformaciones de modelos típicas incluyen la simplificación del modelo, generación de código, generación de simuladores ejecutables basados en la semántica

operacional de los formalismos así como transformaciones que preservan el comportamiento entre modelos en diferentes formalismos.

- **MOLA** [Mola] El proyecto Mola (acrónimo de "MOdel transformation LAnguage") consiste de un lenguaje de transformación de modelos y de una herramienta para la definición y ejecución de transformaciones. El objetivo del proyecto Mola es proveer un lenguaje gráfico para definir las transformaciones entre modelos que sea simple y fácilmente entendible. El lenguaje para la definición de la transformación Mola es un lenguaje gráfico, basado en conceptos tradicionales como pattern matching y reglas que definen como los elementos deben ser transformados. El orden en el cual se aplican las reglas es el orden tradicional de los constructores de programación (secuencia, loop y branching). Los procedimientos Mola definen la parte ejecutable de la transformación. La unidad principal ejecutable es la regla que contiene un pattern y acciones. Un procedimiento está construido con reglas usando constructores de la programación estructural tradicional, es decir, loops, branchings y llamadas a procedimientos, todos definidos de una manera grafica. La parte ejecutable es similar a los diagramas de actividad de **UML**. Mola usa una manera simple para definir metamodelos: Diagramas de clases UML, los cuales consisten solo en un subconjunto de los elementos de UML, es decir, clases, asociaciones, generalizaciones y enumerativos. Solamente soporta herencia simple. Actualmente el metamodelo completo (el metamodelo fuente y el metamodelo destino) deben estar en el mismo diagrama de clases. Adicionalmente se le agregan asociaciones para mapear los elementos del metamodelo fuente en el destino.

- **Kermeta** [Kermeta]. El lenguaje Kermeta fue desarrollado por un equipo de investigación de IRISA (investigadores de INRIA, CNRS, INSA y la Universidad Rennes). El nombre Kermeta es una abreviación para "*Kernel Metamodeling*" y refleja el hecho que el lenguaje fue pensado como una parte fundamental para el metamodelado. La herramienta que ejecuta las transformaciones Kermeta está desarrollada en Eclipse, bajo licencia EPL. Kermeta es un lenguaje de modelado y de programación. Su metamodelo conforma el estándar **EMOF**. Fue diseñado para escribir modelos, para escribir transformaciones entre modelos y para escribir restricciones sobre estos modelos y ejecutarlos. El objetivo de esta propuesta es brindar un nivel de abstracción adicional sobre el nivel de objetos, y de esta manera ver un sistema dado como un conjunto de conceptos (e instancias de conceptos) que forman un todo coherente que se puede llamar modelo. Kermeta ofrece todos los conceptos de **EMOF** usados para la especificación de un modelo. Un concepto real de modelo, más precisamente de tipo de modelo, y una sintaxis concreta que encaja bien con la notación de modelo y metamodelo. Kermeta ofrece dos posibilidades para escribir un metamodelo: escribir el metamodelo con Omondo, e importarlo o escribir el metamodelo en Kermeta y traducirlo a un archivo ecore usando la función "kermeta2ecore".

9.8.2. Herramientas de Transformación de Modelo a Texto

En esta sección describimos las principales herramientas que facilitan la transformación de modelos a texto. [[PGP09].

- **M2T la herramienta para MOF2Text:** El proyecto M2T de la comunidad Eclipse está trabajando en la implementación de una herramienta de código abierto que soporte al lenguaje estándar MOF2Text [MOF2Text]. Está implementación se realizará en dos etapas. En la primera etapa, prevista para 2009, se entregará una versión compatible con las características básicas (el '*core*') del lenguaje Mof2Text, mientras que la segunda versión soportará también sus características avanzadas.

- **MOFScript** [Old06] La herramienta MOFScript [Old06] permite la transformación de cualquier modelo **MOF** a texto. Por ejemplo, permite la generación de código Java, EJB, JSP, C#, **SQL** Scripts, **HTML** o documentación a partir de los modelos. La herramienta está desarrollada como un *plugin* de Eclipse, el cual soporta el *parseo*, chequeo y ejecución de scripts escritos en MOFScript. El lenguaje de transformación MOFScript es un lenguaje que fue enviado al pedido de propuestas de lenguajes de transformación de modelo a texto lanzado por el **OMG** [OMG04], pero que no resultó seleccionado. MOFScript está basado en **QVT**, es un refinamiento del lenguaje operacional de **QVT**. Es un lenguaje textual, basado en objetos y usa **OCL** para la navegación de los elementos del metamodelo de entrada. Además, presenta algunas características avanzadas, como la jerarquía de transformaciones y mecanismos de rastreo. (Más detalles en Anexo III)

9.9. Resumen del Capítulo

El objetivo de este capítulo fue, primeramente, plantear una visión general del proceso de transformación. Luego, analizar los principales mecanismos existentes para la definición de transformaciones modelo a modelo. A continuación se introdujo el lenguaje **QVT**, un estándar para transformaciones especificado por el **OMG**. Luego, se analizaron algunos de los principales requisitos para que un lenguaje de transformaciones M2M sea práctico y usable. A continuación, se presentaron las principales características de las transformaciones modelo a texto, en particular, se introdujo el lenguaje estándar MOF2Text para transformaciones M2Text. Por último, se detallaron las principales herramientas, tanto de transformación M2M, como de transformación M2Text.

Capítulo 10

Metamodelado

10.1. Introducción

Un metamodelo es un mecanismo que permite definir, formalmente, lenguajes de modelado. Por lo tanto, un metamodelo de un lenguaje (gráfico o textual) es una definición precisa de sus elementos mediante conceptos y reglas de cierto metalenguaje necesaria para describir modelos en ese lenguaje.

Con respecto a la definición sintáctica de un nuevo lenguaje, existen dos opciones: una de ellas es crear el lenguaje definiendo su sintaxis y creando nuevas metaclases que lo implementen. Otra opción es implementarlo mediante el mecanismo estándar de extensión de **UML** (mediante un *profile*) que consiste en definir estereotipos que extienden metaclases ya existentes en las especificaciones de OMG para representar los nuevos conceptos del lenguaje. En nuestro trabajo hemos elegido la primera opción

En el capítulo 5 hemos descrito el proceso informal de transformación del modelo de datos a tablas relacionales expresadas en sentencias **SQL** que permiten implementar un **HDW**. En el capítulo 7 se expresó, de manera informal, el proceso de transformación de un **TMD** a un **QG** que permite realizar, en forma automática, consultas sobre la estructura de almacenamiento mediante sentencias **SQL**; además de un conjunto de consultas temporales y de toma de decisión expresadas en forma genérica. En este capítulo se describirá, primeramente, el concepto de metamodelo y la arquitectura de cuatro capas definida por **OMG**. Luego, se describirán todos los metamodelos usados en las transformaciones y, para cada uno de ellos, un conjunto de restricciones **OCL** asociados a los mismos. Las transformaciones en **ATL** completas se desarrollarán en el anexo IV.

10.2. Metamodelos y Meta Object Facility

Un metamodelo es un mecanismo que permite definir, formalmente, lenguajes de modelado. Por lo tanto, un metamodelo de un lenguaje (gráfico o textual) es una definición precisa de sus elementos mediante conceptos y reglas de cierto metalenguaje necesaria para describir modelos en ese lenguaje. El ***Meta Object Facility* (MOF)** [MOF] define un lenguaje común y abstracto para definir lenguajes

de modelado y la forma de acceder e intercambiar modelos expresados en dichos lenguajes.

Existen dos razones fundamentales para el uso de metamodelos en el marco **MDD**; primero, la necesidad de un mecanismo para definir lenguajes de modelado que no sean ambiguos, de modo tal que una herramienta de transformación pueda leer, escribir y comprender los modelos. Por lo tanto, en **MDD**, los modelos se definen mediante metamodelos. Segundo, las reglas de transformación que describen la definición de dicha transformación y detalla cómo un modelo, en un lenguaje fuente, puede ser transformado en un modelo, en un lenguaje destino, utilizan los metamodelos fuente y destino para definir la transformación de modo tal que éstas sean definidas en general y no para una aplicación en particular.

10.2.1. Mecanismos para Definir la Sintaxis de un Lenguaje de Modelado

La sintaxis de los lenguajes se definía, hace algunos años, casi exclusivamente usando ***Backus Naur Form* (BNF)**. Este formalismo es una meta sintaxis usada para expresar gramáticas libres de contexto, es decir, una manera formal de describir cuales son las palabras básicas del lenguaje y cuales secuencias de palabras forman una expresión correcta dentro del mismo [PGP09].

Una especificación en **BNF** es un sistema de reglas de la derivación, escrito como:

```
<símbolo> ::= <expresión con símbolos>
```

Donde <símbolo> es un *no-terminal*, y la expresión consiste en secuencias de símbolos separadas por la barra vertical, '|', indicando una opción, cada una de las cuales es una posible substitución para el símbolo a la izquierda. Los símbolos que nunca aparecen en un lado izquierdo son *terminales*.

El **BNF** se utiliza extensamente como notación para definir la sintaxis (o gramática) de los lenguajes de programación.

Por ejemplo, las siguientes expresiones **BNF** definen la sintaxis de un lenguaje de programación simple:

```
<Programa> ::= "Begin" <Comando> "end"
<Comando> ::= <Asignacion>|<Loop>|<Decision>|<Comando>";"<Comando>
<Asignacion> ::= variableName ":=" <Expresion>
<Loop> ::= "while" <Expresión> "do" <Comando> "end"
<Decision>::= "if"<Expresión>"then"<Comando>"else"<Comando>"endif"
<Expresión> ::= ...
```

El siguiente código es una instancia válida de esta definición:

```
Begin
  If y = 0
    Then  result := 0
    else  result := x;
    while (y > 1) do
       result := result + x;
       y := y - 1
    end
  endif
end
```

Este método es útil para lenguajes textuales, pero dado que los lenguajes de modelado, en general, no están basados en texto, sino en gráficos, es conveniente recurrir a un mecanismo diferente para definirlos. Las principales

diferencias entre los lenguajes basados en texto y los lenguajes basados en gráficos son las siguientes [PGP09]:

10.2.2. Contenedor vs. Referencia

En los lenguajes textuales una expresión puede formar parte de otra expresión, que, a su vez, puede estar contenida en otra expresión mayor. Esta relación de contenedor da origen a un árbol de expresiones. En el caso de los lenguajes gráficos, en lugar de un árbol se origina un grafo de expresiones ya que una sub-expresión puede ser referenciada desde dos o más expresiones diferentes.

10.2.3. Sintaxis Concreta vs. Sintaxis Abstracta.

En los lenguajes textuales la sintaxis concreta coincide (casi) exactamente con la sintaxis abstracta mientras que en los lenguajes gráficos se presenta una marcada diferencia entre ambas.

10.2.4. Ausencia de una Jerarquía Clara en la Estructura del Lenguaje.

Los lenguajes textuales en general se aprehenden leyéndolos de arriba hacia abajo y de izquierda a derecha, en cambio los lenguajes gráficos suelen asimilarse de manera diferente dependiendo de su sintaxis concreta (por ejemplo, comenzamos prestando atención al diagrama más grande y/o colorido). Esto influye en la jerarquía de la sintaxis abstracta, ocasionando que no siempre exista un orden entre las categorías sintácticas.

Usando un lenguaje de modelado, podemos crear modelos; un modelo especifica qué elementos pueden existir en un sistema. Si se define la clase Cliente en un modelo, se pueden tener instancias de Cliente como José, Pedro, etc. Por otro lado, la definición de un lenguaje de modelado establece qué elementos pueden existir en un modelo. Por ejemplo, el lenguaje **UML** establece que dentro de un modelo se pueden usar los conceptos Clase, Atributo, Asociación, Paquete, etc. Debido a esta similitud, se puede describir un lenguaje por medio de un modelo, usualmente llamado "*metamodelo*". El metamodelo de un lenguaje describe qué elementos pueden ser usados en el lenguaje y cómo pueden ser conectados.

Como un metamodelo es también un modelo, el metamodelo en sí mismo debe estar escrito en un lenguaje bien definido. Este lenguaje se llama metalenguaje. Desde este punto de vista, **BNF** es un metalenguaje.

El metamodelo describe la sintaxis abstracta del lenguaje. Esta sintaxis es la base para el procesamiento automatizado (basado en herramientas) de los modelos. Por otra parte, la sintaxis concreta es definida mediante otros mecanismos y no es relevante para las herramientas de transformación de modelos. La sintaxis concreta es la interface para el modelador e influye fuertemente en el grado de legibilidad de los modelos.

Como consecuencia de este desacoplamiento, el metamodelo y la sintaxis concreta de un lenguaje pueden mantener una relación 1: n, es decir que la misma sintaxis abstracta (definida por el metamodelo) puede ser visualizada a través de diferentes sintaxis concretas. Incluso un mismo lenguaje puede tener una sintaxis concreta gráfica y otra textual.

10.3. Arquitectura de Cuatro Capas

OMG define una arquitectura en la que existen cuatro niveles, o capas, denominadas M3, M2, M1 y M0 y que se corresponden, respectivamente, con un

meta-metamodelo, metamodelo, modelo y, por último, instancias del usuario en tiempo de ejecución.

La responsabilidad primaria de la capa de meta-metamodelo (capa M3) es definir el lenguaje para especificar un metamodelo. Un metamodelo (capa M2) es una instancia de un meta-metamodelo donde cada elemento del metamodelo es una instancia de un elemento del meta-metamodelo. El objetivo primario de la capa del metamodelo es definir un lenguaje para especificar modelos.

Un *modelo* (capa M1) es una instancia de un metamodelo. La responsabilidad de esta capa es definir un lenguaje que describa los dominios semánticos, es decir, para permitirles a los usuarios modelar una variedad de dominios diferentes, como procesos, requerimientos, etc.

Por último, la capa de instancias (nivel M0), representa las instancias de los elementos del modelo en tiempo de ejecución; en este nivel están las instancias "reales" del sistema.

> Por ejemplo, en la Figura 10.1 puede verse un modelo de 4 capas donde se observa la metametaclase *Class*, las metaclases *Table* y *Column*, la clase *Cliente* y, por último, una instancia particular de la clase *Cliente*.

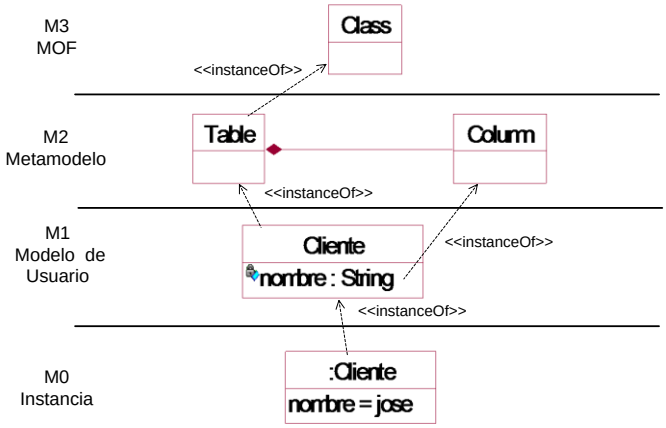


Figura 10.1. Arquitectura de Cuatro Capas

La importancia del metamodelo en **MDD** reside en que, por un lado, el metamodelado permite definir lenguajes de modelado, de modo tal que su definición no sea ambigua y, por otro lado, las reglas de transformación usadas para transformar un modelo en lenguaje A en otro modelo en un lenguaje B, usan los metamodelos de los lenguajes A y B para definir las transformaciones.

10.3.1. Meta Object Facility

OMG propuso el estándar **MOF,** que extiende a **UML** para que éste sea aplicado en el modelado de diferentes sistemas de información. **MOF** es un ejemplo de un meta-metamodelo orientado a objetos por naturaleza (capa M3, en la arquitectura de 4 capas, Figura 10.1). Define un lenguaje común y abstracto para definir lenguajes de modelado. Además, proporciona un modelo común para los metamodelos de **CWM** [CWM], cuyo propósito es permitir un fácil intercambio de metadata entre herramientas de **DW** y repositorios de metadata en ambientes heterogéneos distribuidos, **UML**, lenguaje gráfico, estandarizado, para la visualización, especificación, construcción y documentación de sistemas software y **XMI** [XMI] que define un formato de intercambio basado en **XML** [XML] para modelo en la capa M1 y M2.

MOF utiliza cinco construcciones básicas para definir un lenguaje de modelado: *Clases*, para definir tipos de elementos en un lenguaje de modelado; *Generalización*, define herencia entre clases; *Atributos*, para definir propiedades de elementos de modelado; *Asociaciones*, para definir relaciones entre clases y *Operaciones*, que permite definir operaciones en el ámbito de las clases.

Actualmente, la definición de **MOF** está separada en dos partes fundamentales, EMOF (*Essential MOF*) y CMOF (*Complete MOF*), y se espera que en el futuro se agregue SMOF (*Semantic **MOF***). Ambos paquetes importan los elementos de un paquete en común, del cual utilizan los constructores básicos y lo extienden con los elementos necesarios para definir metamodelos simples, en el caso de EMOF y metamodelos más sofisticados, en el caso de CMOF.

10.3.2. MOF vs. BNF

Si bien la meta sintaxis **BNF** y el meta lenguaje **MOF** son formalismos creados con el objetivo de definir lenguajes textuales y lenguajes gráficos respectivamente, En [WK05] han demostrado que ambos formalismos poseen igual poder expresivo, siendo posible la transformación bi-direccional de sentencias cuya sintaxis fue expresada en **BNF** y sus correspondientes modelos cuya sintaxis fue expresada en **MOF**.

10.3.3. MOF vs. UML

Actualmente la sintaxis de **UML** está definida usando **MOF**, sin embargo **UML** fue creado antes que **MOF**. Inicialmente **UML** no estaba formalmente definido, su sintaxis sólo estaba descrita informalmente a través de ejemplos. **MOF** surgió posteriormente, con el objetivo de proveer un marco formal para la definición de **UML** y otros lenguajes gráficos.

La sintaxis concreta de **MOF** coincide con la de **UML**, lo cual resulta algo confuso para los principiantes en el uso de estos lenguajes. Además **MOF** contiene algunos elementos también presentes en **UML**, por ejemplo, ambos lenguajes tienen un elemento llamado Class. A pesar de que los elementos tienen el mismo nombre y superficialmente describen al mismo concepto, no son idénticos y no deben ser confundidos [PGP09].

10.4. Metamodelos Usados en las Transformaciones

A continuación, presentaremos los metamodelos utilizados para las transformaciones del modelo de datos al **HDW**: el metamodelo de datos (Figura 10.2), metamodelo de datos temporal (Figura 10.3), el metamodelo del grafo de atributos (Figura 10.4), el metamodelo multidimensional temporal (Figura 10.5), el metamodelo relacional (Figura 10.6) y, por último, el metamodelo del grafo de consultas (Figura 10.7).

10.4.1. Metamodelo de Datos

El metamodelo de datos (Figura 10.2) será utilizado para realizar la transformación horizontal (**PIM** a **PIM**), del modelo de datos básico, al modelo de datos temporal (Figura 10.3). Este proceso permitirá transformar las entidades, los atributos y las interrelaciones temporales, en el modelo fuente, en entidades temporales en el modelo destino y, además, las interrelaciones "hecho", en entidades.

El metamodelo de la Figura 10.2 indica que: un `Schema` está compuesto por una o más `Entity` y cero o más `Relationship`; Las `Entity` se vinculan entre

sí mediante `Relationship` y, para determinar la multiplicidad y el tipo de interrelación, se usa la metaclase `RelationshipEnd`. Tanto las entidades como las interrelaciones pueden tener uno o más `Attribute`. El atributo identificador de las entidades está formado por la unión de los atributos identificadores (`IsKey = true`). El metaatributo `isTemp`, permite establecer la característica temporal en `Entity`, `Relationship` y `Attribute`. El metaatributo `isFact` en las interrelaciones, permite determinar a ellas como el hecho principal del **DW.**

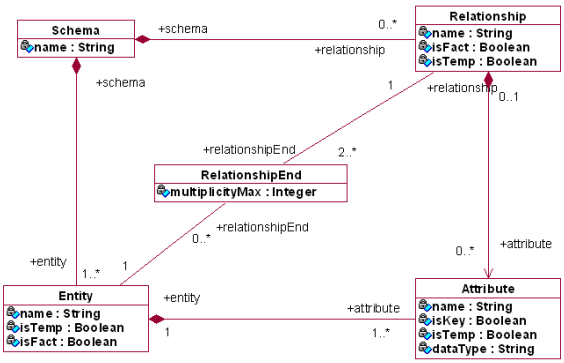


Figura 10.2. Metamodelo de Datos

10.4.1.1. Restricciones en el Metamodelo de Datos

Para mejorar la descripción de los metamodelos son importantes las restricciones, mediante un conjunto de sentencias **OCL**, que permitan establecer un correcto estado del sistema y que garantice que los esquemas estén correctamente representados. Sin estas restricciones podrían ocurrir estados indeseables en el sistema [GL03]

Estableceremos un conjunto reglas de buena formación sobre el metamodelo (Figura 10.2) usando restricciones escritas en **OCL.**

- Dos entidades o interrelaciones que pertenecen a un mismo modelo de datos, no pueden tener el mismo nombre.

```
context Schema inv uniqueNameEntity:
entity -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)

context Schema inv uniqueNameRelationship:
relationship-> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- Los nombres de los atributos e interrelaciones que pertenezcan a un mismo modelo de datos no pueden repetirse.

```
context entity inv uniqueNameAttributeEntity:
attribute -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)

context Relationship inv uniqueNameAttributeRelationship:
relationship -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- Todas las entidades deben tener un atributo clave (`isKey= true`)

```
context Entity inv entityKeyNotEmpty:
attribute -> select(a|a.isKey) -> notEmpty
```

- Las interrelaciones no tienen atributos clave (`isKey= false`)

```
context Relationship inv entityKeyEmpty:
attribute -> select(a|a.isKey) -> Empty
```

- Los atributos temporales de las entidades no pueden ser atributos claves

```
context Entity inv temporalAttrOrKey:
if (attribute -> select(a|a.isKey) -> notEmpty)
then (attribute -> select(a|a.isTemp) -> Empty)
```

- Las interrelaciones que se transformen a hechos (`isFact= true`) no podrán ser temporales.

```
context Schema inv relationshipToFact:
if (relationship -> select (a|a.isFact) -> notEmpty)
then (relationship -> select(a|a.isTemp) -> Empty)
```

- Las entidades que se transformen a hechos (`isFact= true`) no podrán ser temporales.

```
context Schema inv entityToFact:
if (entity -> select (a|a.isFact) -> notEmpty)
then (entity -> select(a|a.isTemp) -> Empty)
```

- En el modelo puedo elegir una entidad o una interrelación como hecho, no ambas

```
context Schema inv entityXorRelatinonshiptoFact:
entity -> select (a|a.isFact) -> notEmpty) xor
relationship -> select(a|a.isFact) -> notEmpty
```

10.4.2. Metamodelo de Datos Temporal

El metamodelo de datos temporal (Figura 10.3) será utilizado conjuntamente con el metamodelo del **AG** (Figura 10.4) para realizar la transformación horizontal (**PIM** a **PIM**), del modelo de datos temporal al **AG.**

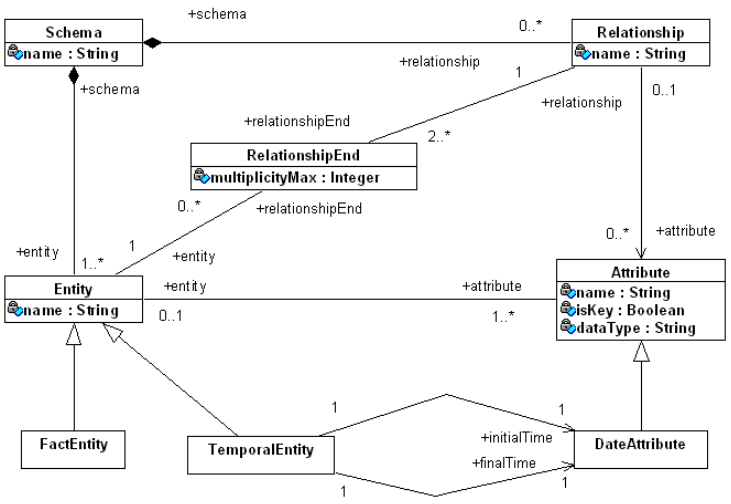


Figura 10.3. Metamodelo de Datos Temporal

El metamodelo de la Figura 10.3 indica que: un `Schema` está compuesto por una o más `Entity` y una cero o más `Relationship`; las entidades pueden representar el hecho principal del **DW** (`FactEntity`) o a las entidades temporales (`TemporalEntity`); tanto las entidades como las interrelaciones pueden tener cero o más `Attribute`, este último puede especializarse para representar el intervalo temporal de las entidades temporales (`DateEntity`). El atributo identificador de las entidades está formado por la unión de los atributos identificadores (`IsKey = true`), Las `Entity` se vinculan entre sí mediante `Relationship` y, para determinar la multiplicidad y el tipo de interrelación, se utiliza una `RelationshipEnd`.

10.4.2.1. Restricciones en el Metamodelo de Datos Temporal

Estableceremos un conjunto reglas de buena formación sobre el metamodelo (Figura 10.3) usando restricciones escritas en **OCL.**

- Dos entidades o interrelaciones que pertenecen a un mismo modelo de datos, no pueden tener el mismo nombre

```
context Schema inv uniqueNameEntity:
entity -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)

context Schema inv uniqueNameRelationship:
relationship-> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- Los nombres de los atributos e interrelaciones que pertenezcan a un mismo modelo de datos no pueden repetirse.

```
context entity inv uniqueNameAttributeEntity:
attribute -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)

context Relationship inv uniqueNameAttributeRelationship:
relationship -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- Todas las entidades deben tener un atributo clave (`isKey= true`)

```
context Entity inv entityKeyNotEmpty:
attribute -> select(a|a.isKey) -> notEmpty
```

- Las interrelaciones no tienen atributos clave (`isKey= false`)

```
context Relationship inv entityKeyEmpty:
attribute -> select(a|a.isKey) -> Empty
```

- Los entidades temporales (`TemporalEntity`) no pueden tener atributos claves

```
context TemporalEntity inv temporalEntityKeyEmpty:
attribute -> select(a|a.isKey) -> Empty
```

10.4.3. Metamodelo del Grafo de Atributos

El metamodelo del **AG** (Figura 10.4) será utilizado, conjuntamente con el metamodelo de datos temporal (Figura 10.3), para la transformación horizontal (**PIM** a **PIM**) del modelo de datos temporal adaptado al **AG**.

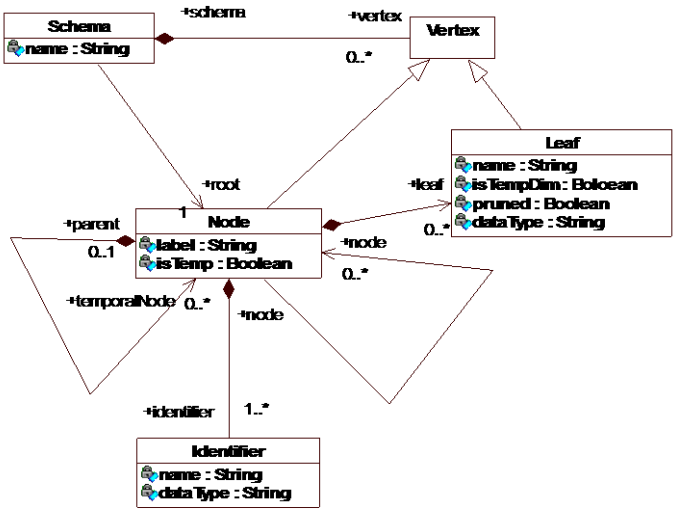


Figura 10.4. Metamodelo del Grafo de Atributos

El metamodelo de la Figura 10.4 indica que: un `Schema` está compuesto por `Vertex`, éstos se clasifican en `Leaf` y `Node`, los `Leaf` corresponden a atributos que se derivaron del modelo de datos y que no son identificadores y pueden ser o no temporales; por el contrario, los `Node`, corresponden a atributos identificadores. En el `Schema` uno de los nodos es raíz (`root`). Los `Node` están compuestos por cero o más `Leaf`. Las `Leaf` tienen un nombre (`name`) que describe al atributo que identifica, los `Node` tienen una etiqueta (`Label`) que se corresponde con el nombre de la entidad de la cual proviene, además, los `Node` tienen identificadores (`Identifier`) que se corresponden con los identificadores de la entidad que representa el `Node`. Los `Node` pueden vincularse entre sí o con `Leaf`.

10.4.3.1. Restricciones en el Metamodelo del Grafo de Atributos

Estableceremos un conjunto reglas de buena formación sobre el metamodelo (Figura 10.4) usando restricciones escritas en **OCL.**

- Dentro de un mismo esquema, los nodos no pueden tener el mismo nombre.

```
context Schema inv uniqueNameNode:
node -> forAll (e1,e2 | e1.label = e2.label
implies e1 = e2)
```

- Dentro de un mismo esquema, las hojas no pueden tener el mismo nombre.

```
context Schema inv uniqueNameLeaf:
leaf -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- El nodo raíz no puede ser temporal

```
context Schema inv rootNotTemporal:
root.isTemp = false
```

- La hoja elegida como dimensión temporal (`isTempDim = true`), no podrá ser una medida de hecho principal

```
context Node inv isTempDimNotMeasure:
leaf -> select(m | m.isTempDim = false)
```

10.4.4. Metamodelo Multidimensional Temporal

El metamodelo **TMD** (Figura 10.5) será utilizado, conjuntamente con el metamodelo de **AG** (Figura 10.4) para la transformación horizontal (**PIM** a **PIM**) del **AG** al modelo **TMD.**

El metamodelo de la Figura 10.5 indica que: el `Schema` está formado por un `Fact`. Un `Fact` se vincula con una o más `Dimension` y éstas con cero o más `Hierarchy`; éstas últimas se clasifican en temporales (`TempHierarchy`) o estrictas (`StrictHierarchy`) estas últimas pueden relacionarse entre sí. Una `TempHierarchy` está compuesta por cero o dos `StrictHierarchy`. Las `Fact`, `Dimension` y `Hierarchy` se generalizan en un elemento de modelado multidimensional (`MultidimModelElement`) que está compuesto por atributos (`Attribute`), identificadores (`Identifier`) y referencias (`Reference`). El hecho puede tener cero o más medidas (`Measure`).

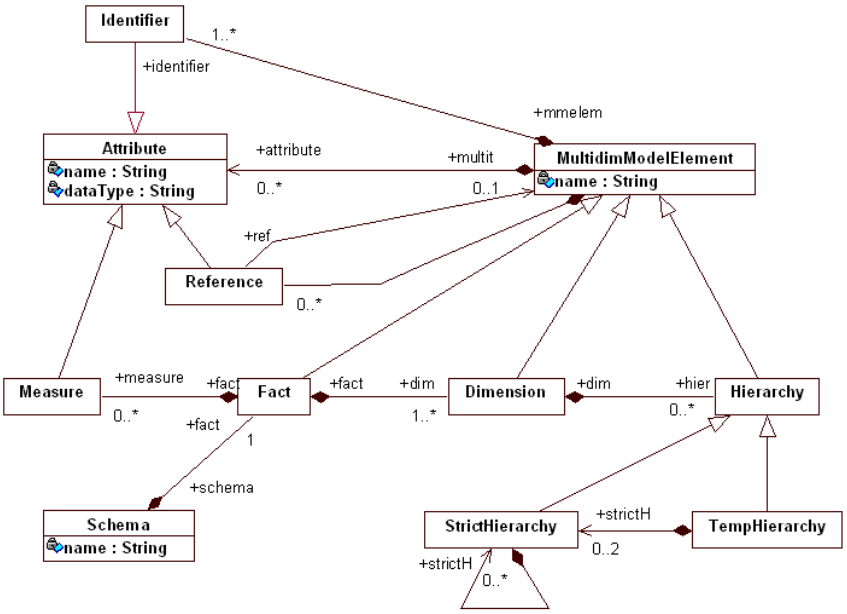


Figura 10.5. Metamodelo Multidimensional Temporal

10.4.4.1. Restricciones en el Metamodelo Multidimensional Temporal

Estableceremos un conjunto reglas de buena formación sobre el metamodelo (Figura 10.5) usando restricciones escritas en **OCL**

- Las dimensiones, en un mismo esquema, no pueden tener el mismo nombre.

```
context Fact inv uniqueNameDimension:
dim -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- Las jerarquías, en de un mismo esquema, no pueden tener el mismo nombre.

```
context Dimension inv uniqueNameHierarchy:
hier -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- Las medidas de los hechos, en de un mismo esquema, no pueden tener el mismo nombre.

```
context fact inv uniqueNameMeasure:
measure -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

10.4.5. Metamodelo Relacional

El metamodelo relacional (Figura 10.6) será utilizado conjuntamente con el metamodelo multidimensional temporal (Figura 10.5), en la transformación **PIM** a **PSM** del modelo **TMD** a tablas relacionales.

El metamodelo de la Figura 10.6 indica que: el `Schema` está compuesto por una o más `Table` que tienen `Column`. Las `Column` pueden ser, a su vez, claves foráneas (`ForeignKey`) o claves primarias (`Key`). La `Table`, tiene una `Key` (que puede ser compuesta) y cero o más `ForeignKey`, cada `ForeignKey` hace referencia a una `Table`.

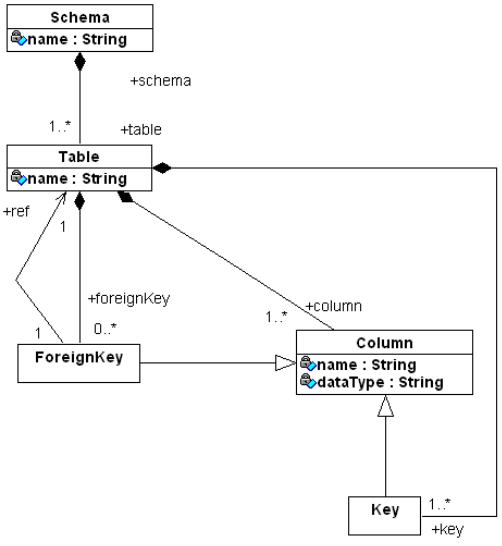


Figura 10.6. Metamodelo Relacional

10.4.5.1. Restricciones en el Metamodelo Relacional

Estableceremos un conjunto reglas de buena formación sobre el metamodelo (Figura 10.6) usando restricciones escritas en **OCL**

- Las diferentes tablas, en el esquema, no pueden tener el mismo nombre.

```
context Schema inv uniqueNameTable:
table -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- Los nombres de los atributos, dentro de la misma tabla, no pueden tener el mismo nombre.

```
context Table inv uniqueNameAttributeTable:
column -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

10.4.6. Metamodelo de Grafo de Consulta

El metamodelo de **QG** (Figura 10.7) será utilizado conjuntamente con el metamodelo **TMD** (Figura 10.5) para la transformación al **GC,** de cual se derivaran las consultas en **SQL.**

El metamodelo de la Figura 10.7 indica que: el `Schema` está compuesto por `Vertex` que puede ser o una `Measure` o un `Node`, este último representará a una tabla en el modelo relacional. Un `Node` está compuesto por `Attribute`, `Key` y `ForeignKey`. Los `Node` se especializan en `Fact`, que representan al hecho principal del **DW** temporal, en `Level`, que se corresponden con los distintos niveles de la jerarquía y en `Temporal`, que representan los nodos temporales; estos últimos se especializan, a su vez, en entidades (`TempEnt`), atributos (`TempAtt`) o interrelaciones (`TempRel`) temporales; el metaatributo orden (`order`) de `Vertex`, permite establecer, por parte del usuario, el orden en la elección de los niveles de jerarquía en la construcción de la sentencia **SQL.**

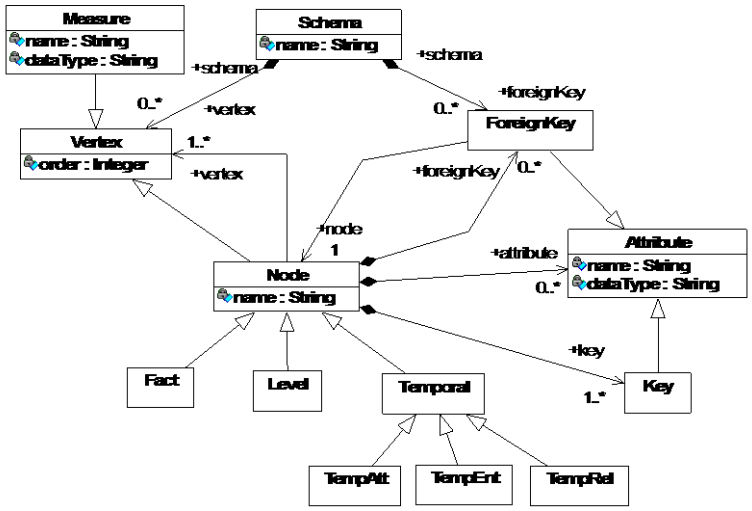


Figura 10.7. Metamodelo del Grafo de Consulta

10.4.6.1. Restricciones en el Metamodelo del Grafo de Consultas

Estableceremos un conjunto reglas de buena formación sobre el metamodelo (Figura 10.7) usando restricciones escritas en **OCL**

- Los diferentes vértices, en el mismo esquema no pueden tener el mismo nombre.

```
context Schema inv uniqueNameVertex:
```

```
vertex -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

- Los diferentes atributos, en el mismo nodo no pueden tener el mismo nombre.

```
context Node inv uniqueNameAttibute:
attribute -> forAll (e1,e2 | e1.name = e2.name
implies e1 = e2)
```

10.5. Visión General del Proceso Completo

Los metamodelos mostrados anteriormente, serán utilizados en las transformaciones formales para el diseño del **HDW** y del **QG.** Detallamos en la Figura 10.8 el proceso formal de las transformaciones en forma completa.

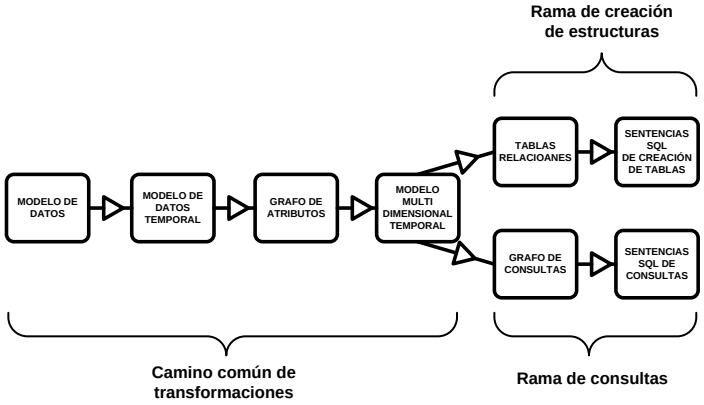


Figura 10.8. Visión Esquemática de las Transformaciones

El proceso está compuesto por tres partes: un camino común de transformación que comienza en el modelo de datos y culmina en el **TMD** y dos ramas, a partir de éste, una (rama superior) que permite crear el conjunto de tablas relacionales que implementan el **HDW** y la otra (rama inferior), que concluye en el **QG** y que permite realizar, en forma automática, consultas sobre la estructura **TMD** mediante sentencias **SQL.**

10.6. Resumen del Capítulo

El objetivo de este capítulo fue mostrar, primeramente, una visión general del concepto de metamodelo. Luego, se detalló la arquitectura de 4 capas propuesta por **OMG**. A continuación, se describieron los metamodelos usados en las transformaciones: el metamodelo de datos, el metamodelo de datos temporal, el metamodelo del grafo de atributos, el metamodelo multidimensional temporal, el metamodelo relacional y, por último, el metamodelo del grafo de consultas. Por último, para cada uno de ellos, se establecieron un conjunto de restricciones **OCL** asociados a los mismos.

Capítulo 11

Prototipo de Implementación

11.1. Introducción

En este capítulo, utilizando el prototipo desarrollado en ECLIPSE que implementa el método de diseño del **HDW** desarrollado en la tesis, mostraremos, mediante el ejemplo presentado en los capítulos precedentes, cómo utilizarlo detallando, paso a paso, las acciones necesarias para producir su implementación en un **RBBMS**.

Tal como lo planteamos en el desarrollo de la tesis, esta herramienta tiene dos usuarios diferenciados; uno, el desarrollador de la aplicación que, comenzando con el modelo de datos fuente, generará, primero el **HDW** y, luego, el **QG**; el otro, el usuario de la aplicación propiamente dicho, él realizará consultas sobre el **QG** marcando los iconos respectivos en función de sus necesidades de información.

El código completo de esta herramienta puede bajarse de:

http://www.lifia.info.unlp.edu.ar/eclipse/pages/tesis_neil.htm

11.2. Creación de un Nuevo Proyecto

El proceso comienza con la creación de un nuevo proyecto, ésta actividad la realiza el desarrollador de la aplicación, los pasos a seguir son:

1. Creamos un nuevo proyecto vacío.
2. Ingresamos el nombre del proyecto.
3. Para crear un modelo de datos utilizando el editor gráfico, creamos un nuevo archivo.
4. Seleccionamos un diagrama MD
    a. Especificamos el nombre del diagrama, que debe tener extensión, en el ejemplo, "md_diagram"
    b. Seleccionamos el nombre del modelo asociado al diagrama, debe tener extensión "md"
    c. En la figura 11.1, mostramos la pantalla donde se observa el diagrama del proyecto recién creado.

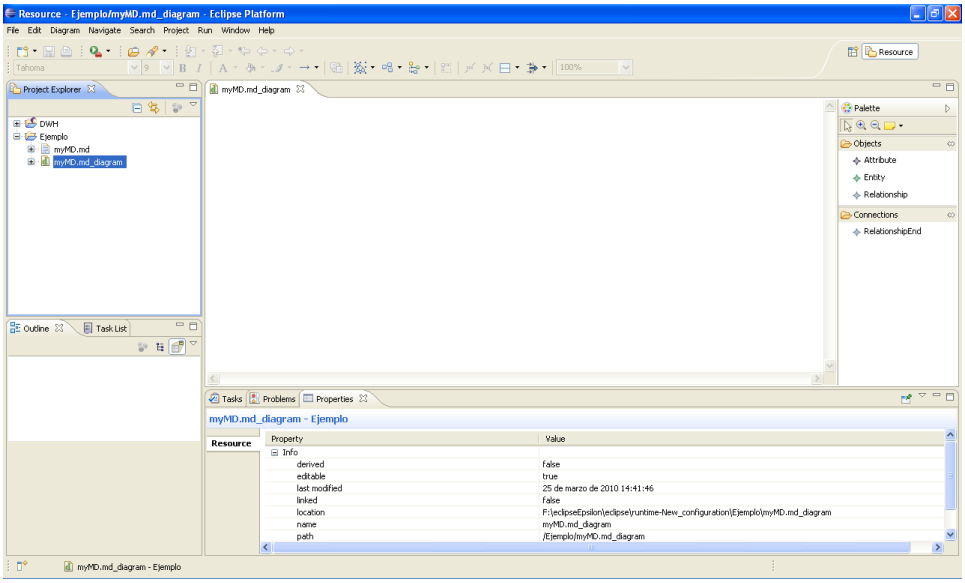


Figura 11.1. Diagrama del Proyecto Recién Creado

11.3 Creación del Modelo de Datos

El proceso comienzo con la creación de un modelo de datos que el desarrollador de la aplicación utilizará como modelo fuente para la creación del **HDW**. Los pasos a seguir para la creación del modelo de datos (**PIM**) son los siguientes:

1. Utilizando la paleta ubicada en la izquierda de la pantalla, se crean las entidades e interrelaciones del modelo.
2. Una vez seleccionado un ícono de la paleta, hacemos clic en el área en blanco para crear entidades o interrelaciones.

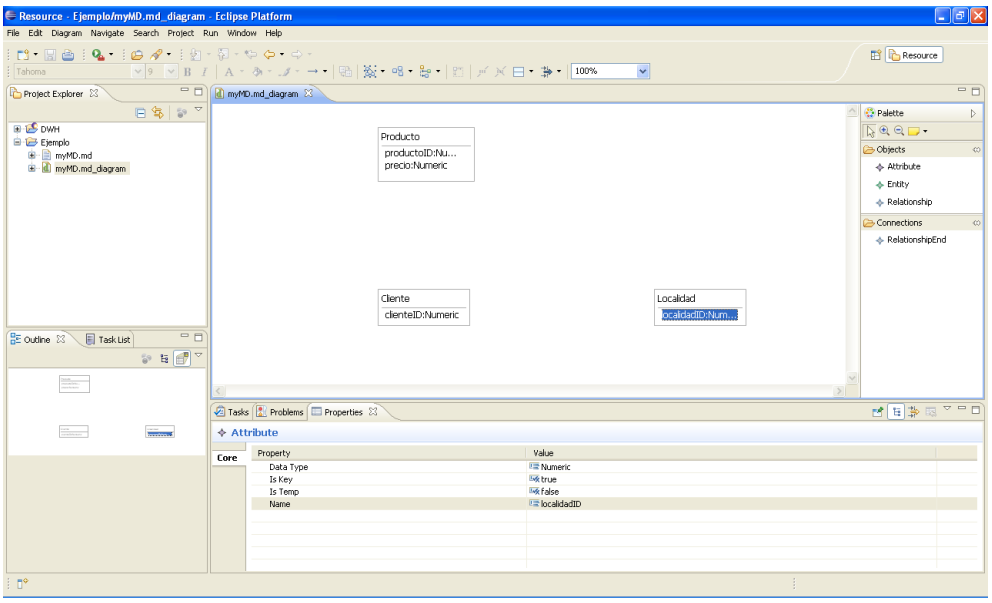


Figura 11.2. Entidad Producto, Cliente y *Localidad* con sus Atributos

3. Al crear un elemento, en la solapa de propiedades se mostrarán los valores de cada una.
4. Para crear un atributo, los seleccionamos de la paleta y hacemos *clic* dentro de la entidad o interrelación a la cual se lo desee agregar.
5. En la figura 11.2, mostramos la pantalla donde se observa la entidad *Producto, Localidad y Cliente* con sus atributos.
5. Para crear una interrelación, seleccionamos el icono correspondiente de la paleta y hacemos *clic* sobre el área en blanco.
6. En la figura 11.3, mostramos la pantalla donde se observa el agregado de la interrelación LOC-CLI.

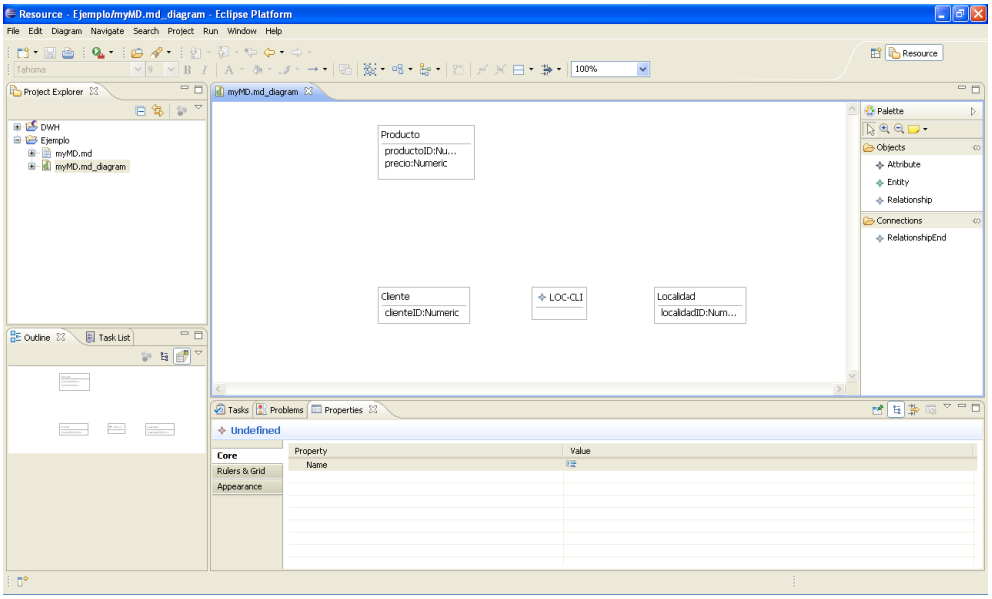


Figura 11.3. *Entidad Producto y Cliente* y *Localidad* con sus Atributos

7. Para crear los *relationEnds*, seleccionamos el icono correspondiente en la paleta y hacemos *clic* entre la interrelación y la entidad que se desee asociar.

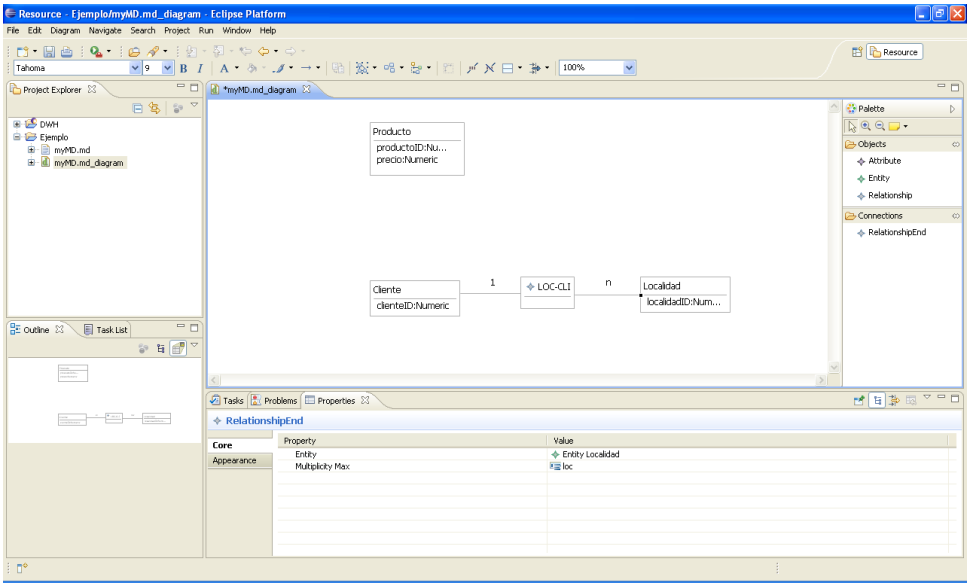


Figura 11.4. *relationEnd* entre *Cliente*, *LOC-CLI* y Localidad

8. En la figura 11.4, mostramos la pantalla donde se observa *relationEnds* entre *Cliente*, *LOC-CLI* y *Localidad*.
9. En la figura 11.5, mostramos la pantalla donde se agregó la interrelación *Venta*, con sus atributos, asociada a las entidades *Producto* y *Cliente*.

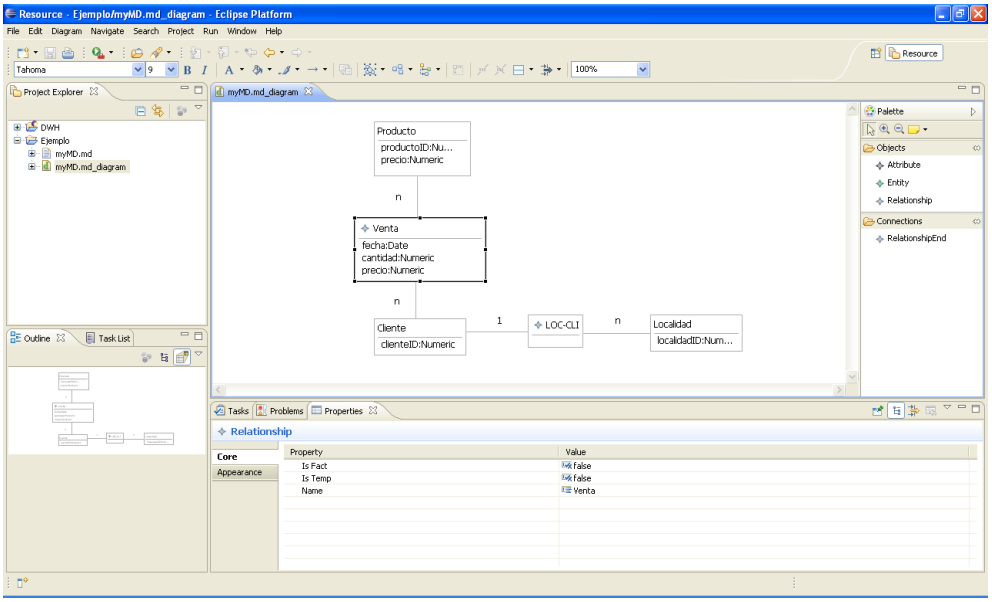


Figura 11.5. Interrelación *Venta*, con atributos y asociada a *Producto* y *Cliente*.

11.3.1. Creación del Modelo de Datos Temporal

Una vez creado el modelo de datos, el paso siguiente es establecer cuáles componentes del modelo se necesitan preservar en el tiempo. Éstos pueden ser atributos, entidades e interrelaciones temporales.

1. Para marcar un atributo como temporal, deberemos cambiarle el valor de la propiedad *IsTemp* a *True*.

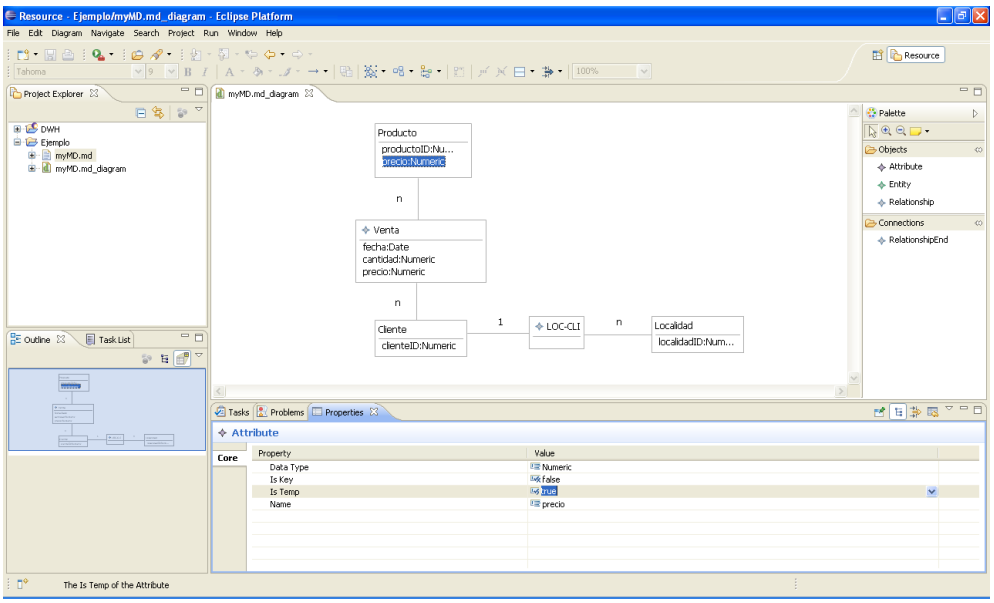


Figura 11.6. Transformación de un Atributo en temporal

2. En la figura 11.6, mostramos la pantalla donde se observa cómo transformar un atributo en temporal
3. Para transformar una entidad o interrelación en temporal, debemos cambiarle el valor de la propiedad *IsTemp* a *true*.
4. En la figura 11.7, mostramos la pantalla donde se observa la interrelación LOC-CLI como temporal.

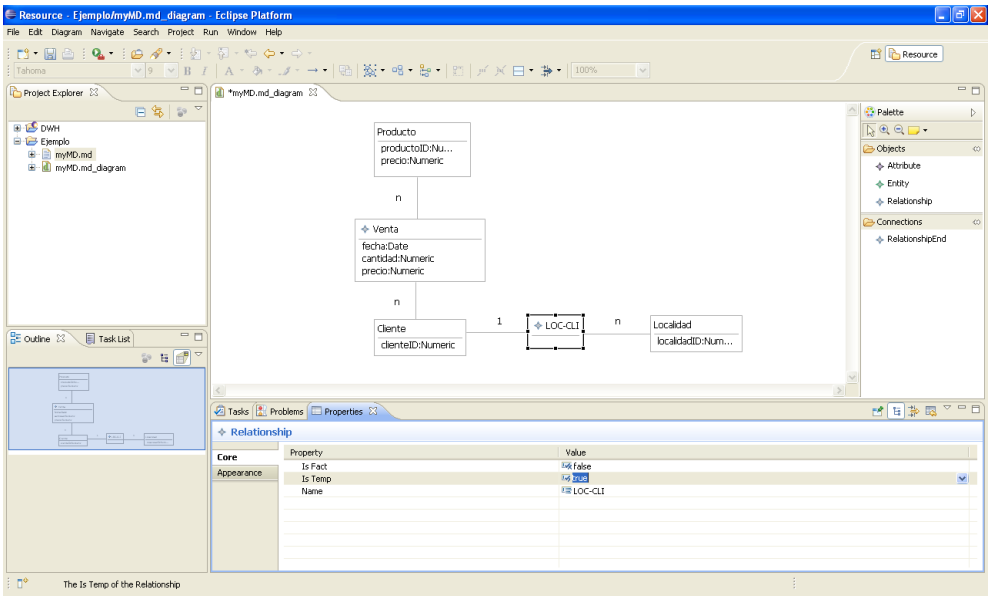


Figura 11.7. Transformación de la interrelación *LOC-CLI* en temporal

11.3.2. Marcado del Hecho Principal

Una vez creado el modelo de datos temporal, el paso siguiente es determinar cuál será el hecho principal del **HDW**.

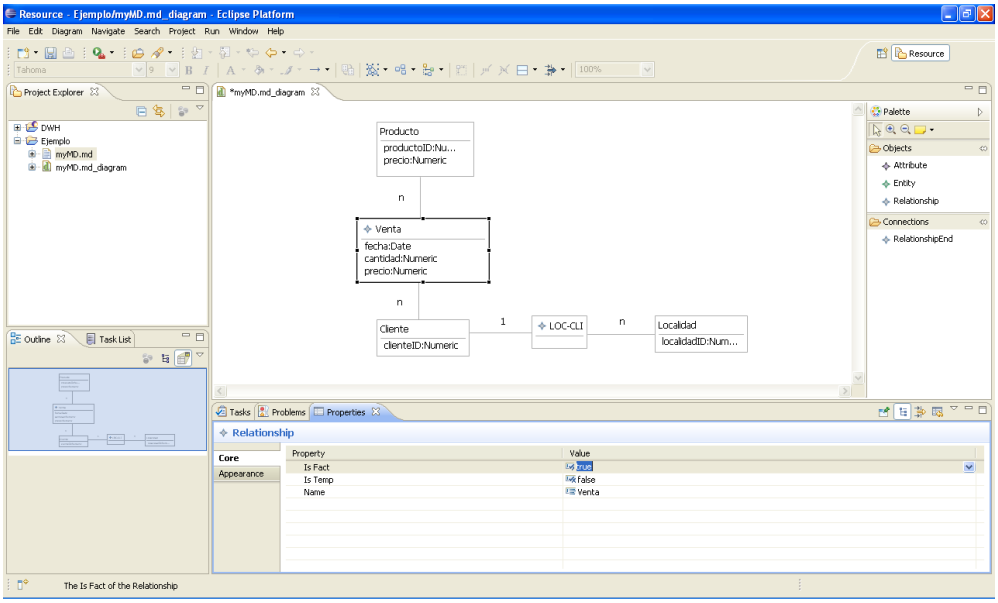


Figura 11.8. Transformación de una Interrelación en hecho principal

1. Para establecer una entidad o interrelación como hecho principal, deberemos cambiarle a *true* la propiedad *isFact*.

2. En la figura 11.8, mostramos la pantalla donde se observa cómo transformar la interrelación *Venta* en el hecho principal.

11.3.3. Restricciones en el Modelo

La herramienta implementa un conjunto de las restricciones impuestas a los metamodelos (ver capítulo 10). Por ejemplo, las entidades deben tener clave primaria y el hecho principal no puede ser marcado como temporal.

1. Para chequear que el modelo cumpla con las verificaciones, hacemos *clic* en *Edit* y luego en *Validate*.
2. En la figura 11.9, mostramos la pantalla donde se observan los errores encontrados, tanto en el modelo como en la solapa *problems*.

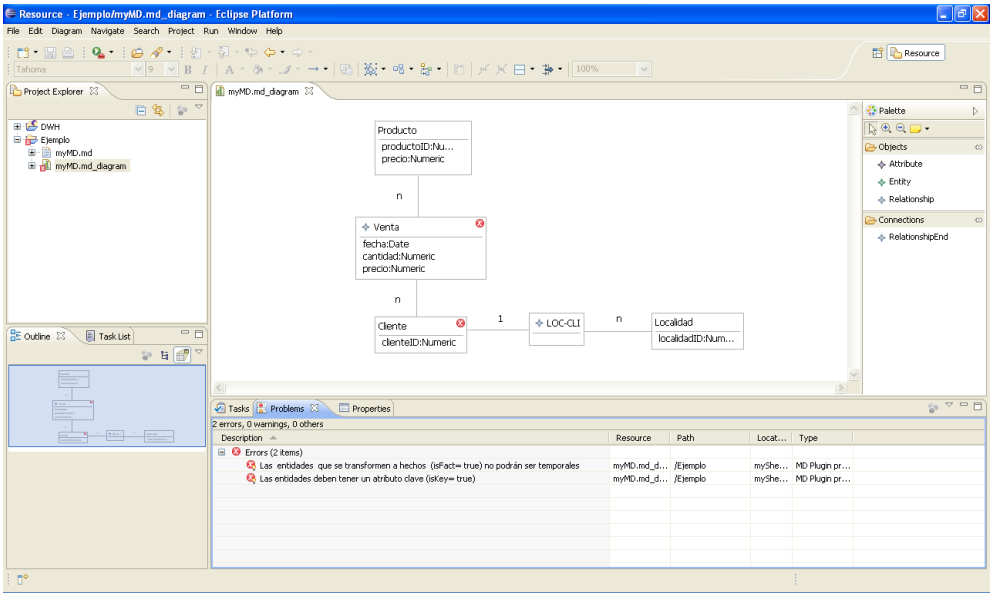


Figura 11.9. Validación de errores en el modelo

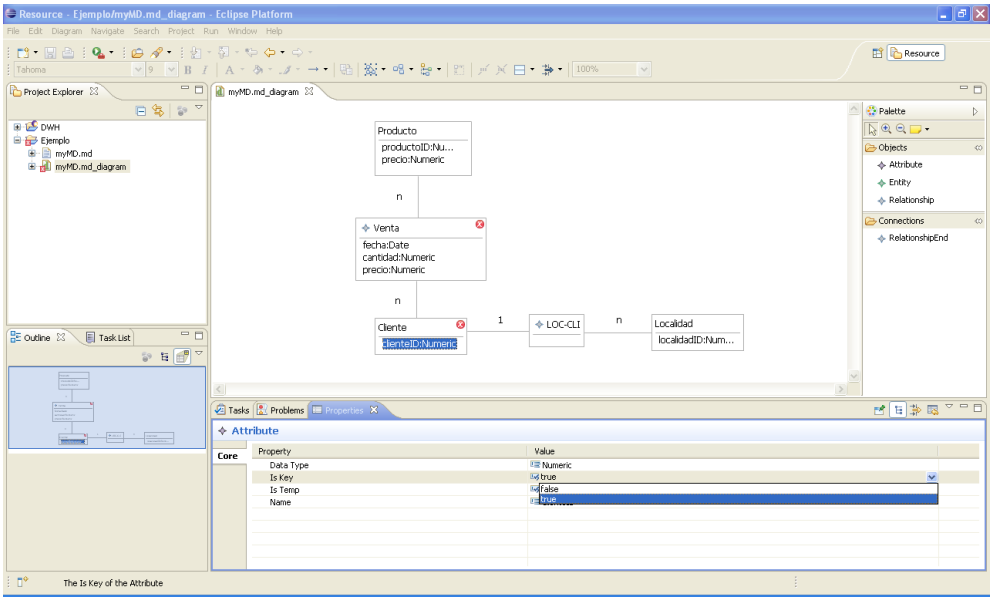


Figura 11.10. Entidad *Cliente* con la propiedad *isKey* en *true*.

3. En la figura 11.10, mostramos cómo se establece el atributo clave primaria en la entidad *Cliente* con la propiedad *isKey* en *true*.
4. En la figura 11.11, mostramos cómo se establece el atributo *isTemp* de la interrelación *Venta* en *false*.

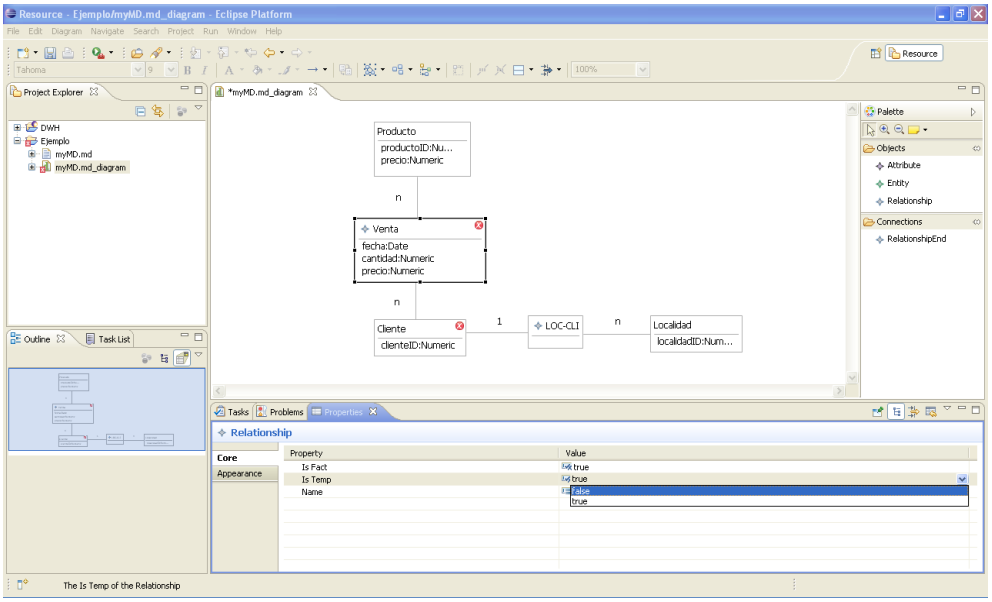


Figura 11.11. Atributo *isTemp* de la interrelación *Venta* en *false*.

5. En la figura 11.12, se observa el diagrama sin errores de validación.

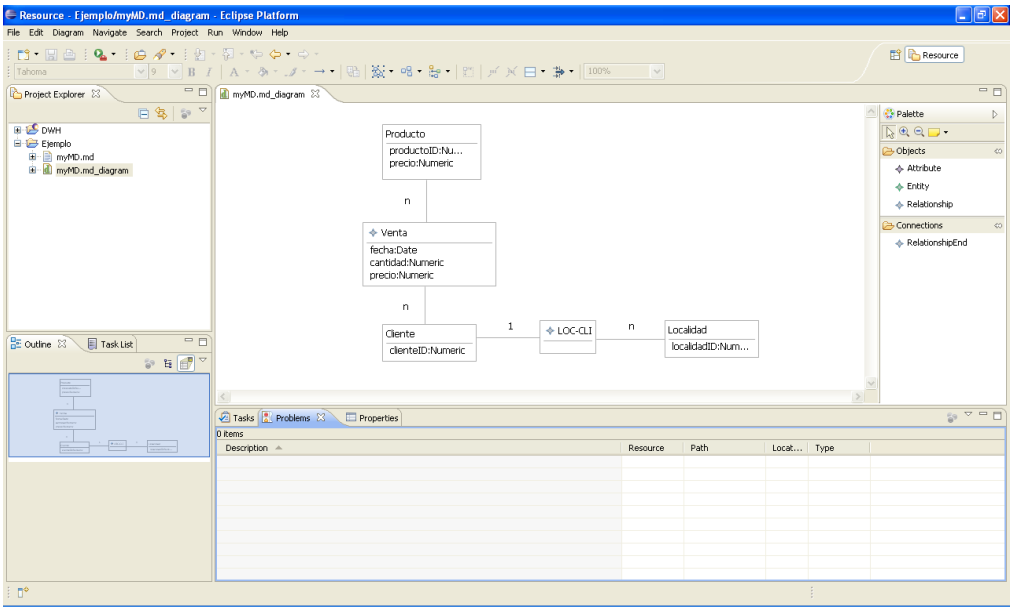


Figura 11.12. Diagrama sin errores de validación.

11.4. Transformación al Modelo de Datos Temporal

Una vez terminado (y validado) el modelo de datos, el paso siguiente es generar, automáticamente, el modelo de datos temporal.

1. Para obtener el modelo de datos temporal a partir del modelo de datos creado anteriormente, debemos ejecutar la transformación denominada MD2MDT.atl.
2. En la figura 11.13, mostramos cómo se ejecuta la transformación **ATL** que generará el modelo de datos temporal.

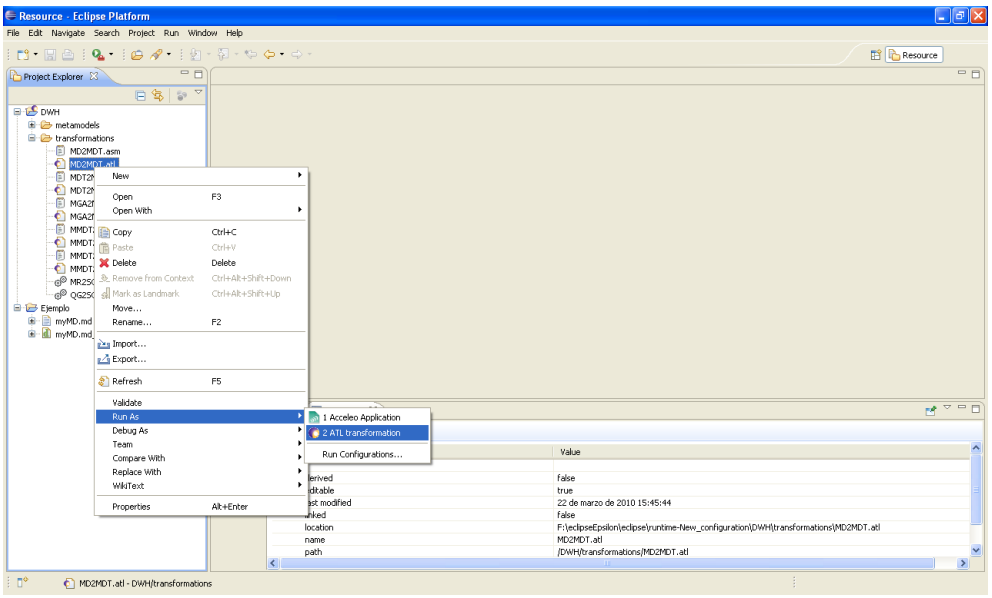


Figura 11.13. Ejecución de la transformación ATL

3. Debemos especificar la ubicación de los metamodelos, del modelo de datos que deseamos transformar y el nombre del diagrama de datos temporal a crear.

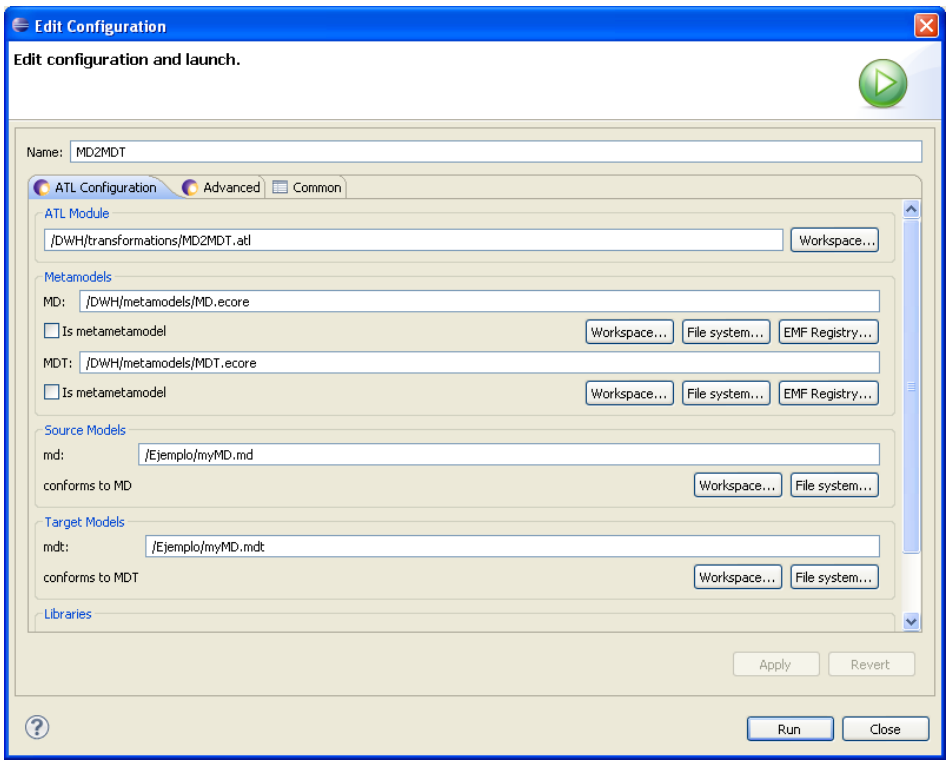


Figura 11.14. Configuración para realizar la transformación

4. En la figura 11.14, mostramos la pantalla donde se observa la configuración necesaria para realizar la transformación.
5. La transformación crea el archivo correspondiente, con extensión "mdt"
6. No visualizamos el modelo temporal, ya que el mismo no será modificado, se utilizará como modelo fuente para la creación del **AG**.

11.5. Transformación al Grafo de Atributos

El paso siguiente en el proceso de transformación, es la creación del **AG**. Éste grafo se utilizará para establecer, por parte del diseñador de la aplicación, las características que tendrá el **HDW**.

1. Para obtener el **AG**, deberá ejecutarse la transformación denominada MDT2MGA.atl.
2. La transformación creará un archivo con extensión "mga"
3. En la figura 11.15, mostramos cómo se ejecuta la transformación **ATL** que permite la generación del **AG**.

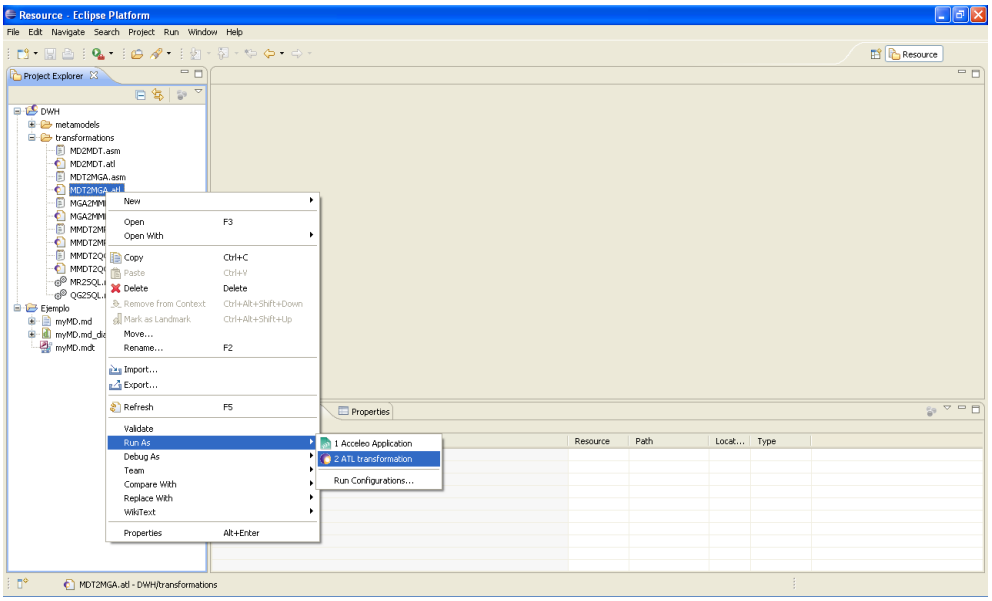


Figura 11.15. Ejecución de la transformación ATL

4. En el **AG** creado se pueden marcar los atributos que no se deseen que aparezca en el **HDW** y, en el hecho principal, se puede marcar, además, un atributo de tipo fecha como dimensión temporal, en el caso de no contar con una jerarquía temporal.
5. En la figura 11.16, se muestra el diagrama del **AG** generado.

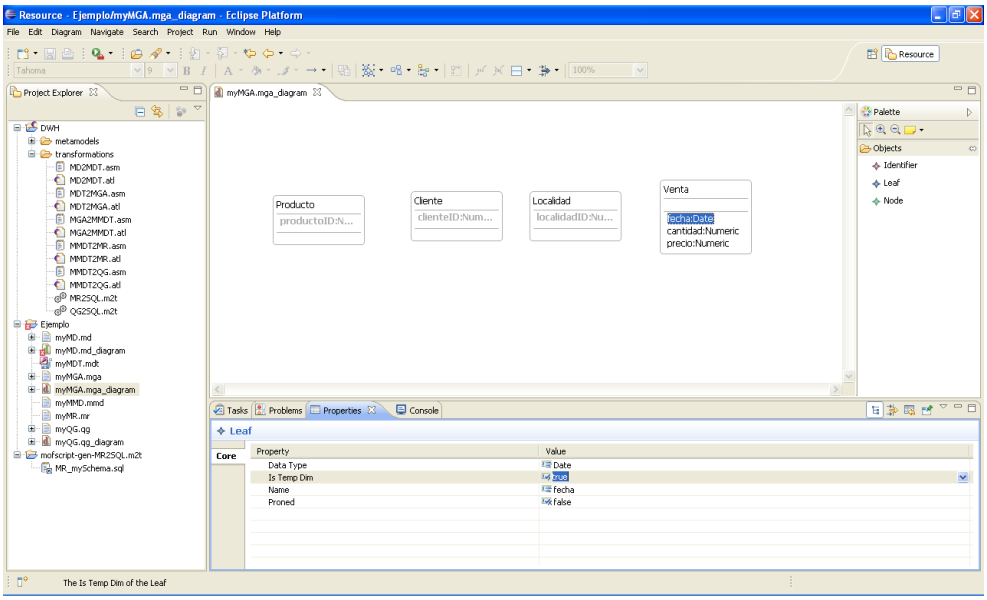


Figura 11.16. Diagrama del AG generado y modificado

11.6. Transformación al Modelo Multidimensional Temporal

Una vez modificado, si amerita, el **AG**, el paso siguiente es la creación del **HDW**, éste modelo tampoco tendrá visibilidad ya que no se precisan realizar modificaciones sobre el mismo.

1. Para obtener el modelo de datos **TMD**, ejecutamos la transformación MGA2MMDT.atl
2. En la figura 11.17, mostramos cómo se ejecuta la transformación **ATL** para la obtención del **HDW**.

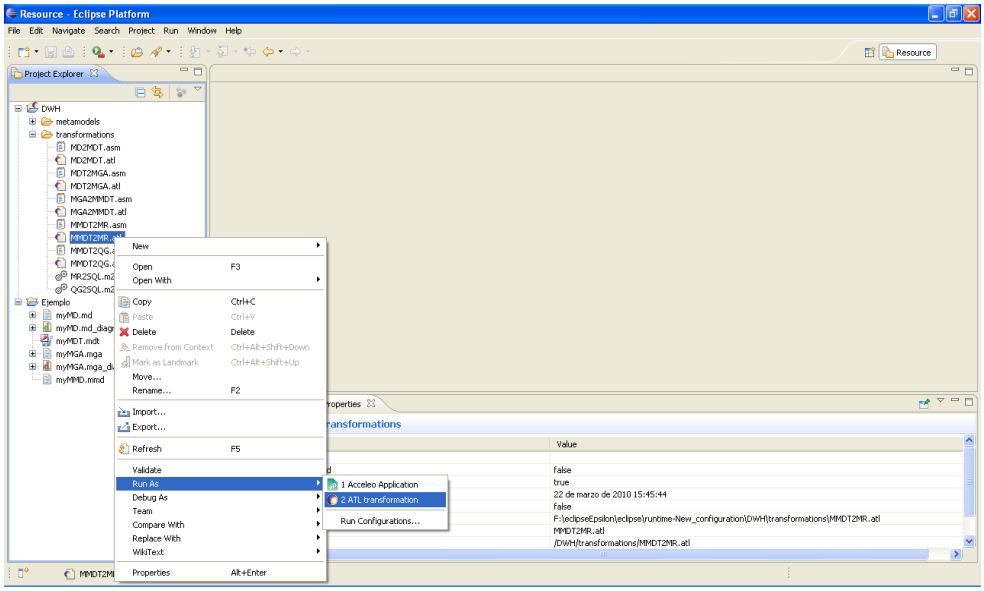


Figura 11.17. Ejecución de la transformación ATL

11.7. Transformación al Modelo Relacional

Una vez obtenido el **HDW**, el paso siguiente es la transformación al modelo relacional que nos permitirá, en la próxima transformación, obtener las sentencias **SQL** que implementarán el **HDW** en un **RDBMS**.

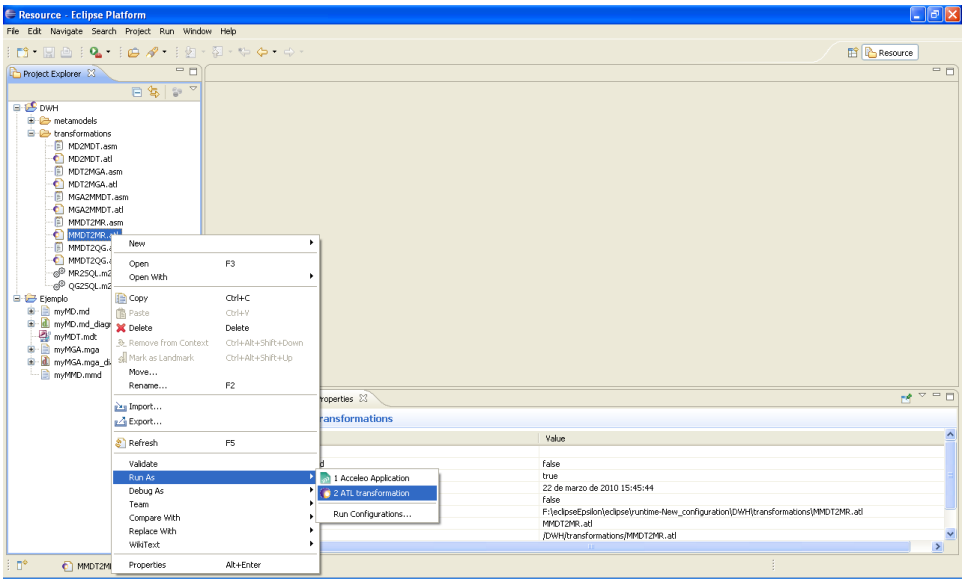


Figura 11.18. Ejecución de la transformación ATL

1. Para obtener el modelo relacional, a partir del **HDW**, ejecutamos la transformación MMDT2MR.atl
2. En la figura 11.18, mostramos cómo se ejecuta la transformación **ATL** para obtener el modelo relacional.
3. Éste modelo tampoco precisa tener visibilidad, ya que no necesita ser modificado.

11.8. Transformación a Sentencias SQL

Hasta este momento, las transformaciones han sido de modelo a modelo, usando **ATL**; la próxima transformación, donde usaremos MOFScript, es de modelo a texto.

1. Para obtener el código **SQL** en texto plano para la creación del modelo relación, ejecutamos la transformación MR2SQL.
2. En la figura 11.19, mostramos cómo se ejecuta la transformación MOFScript que nos permitirá obtener código **SQL**.

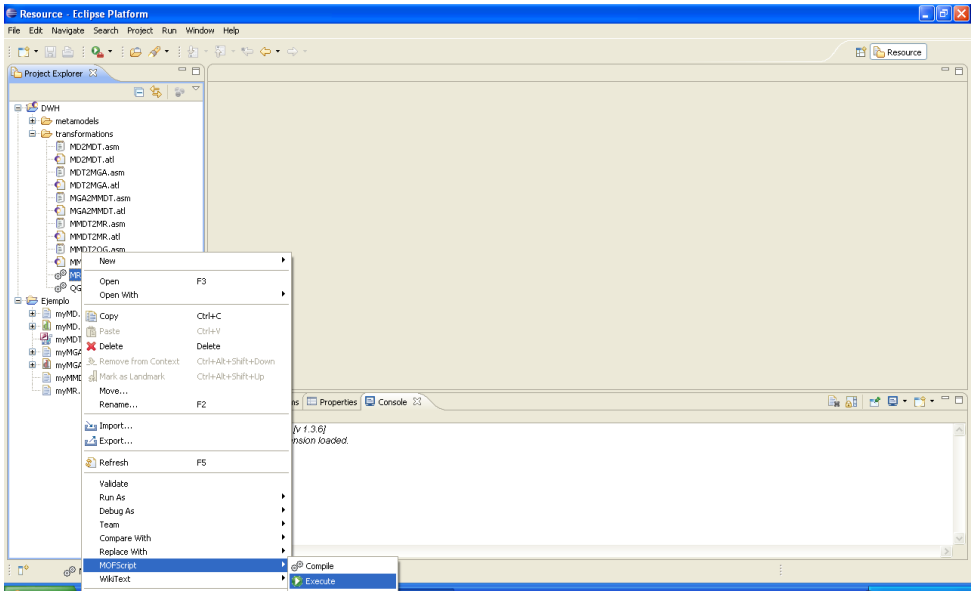


Figura 11.19. Ejecución de la transformación MOFScript

3. En la figura 11.20, se visualiza el código **SQL** para la creación del modelo relacional obtenido correspondiente al **HDW.**

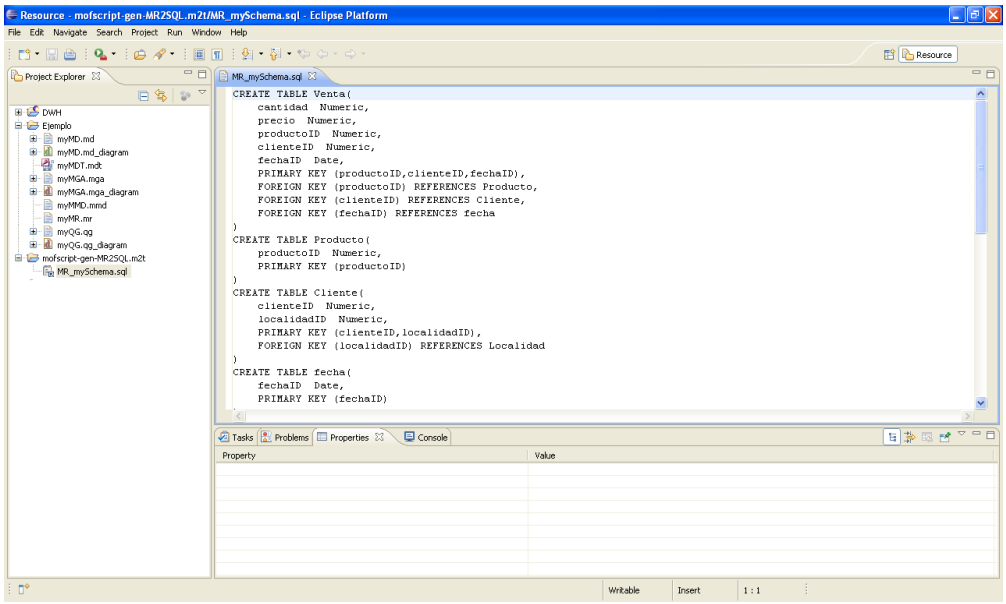


Figura 11.20. Código SQL generado

11.9. Transformación al Grafo de Consultas

Una vez generado el modelo del **HDW**, el paso siguiente es la creación del **QG**, el mismo permitirá realizar consultas automáticas sobre el almacenamiento **TMD**.

1. Para obtener el diagrama del **QG**, que se generará a partir del modelo **TMD**, seleccionamos en la pantalla la opción *Initializa diagram file*.
2. Especificamos el nombre del diagrama, la extensión debe ser "qg_diagram".
3. En la figura 11.21, se muestra e **QC** generado.

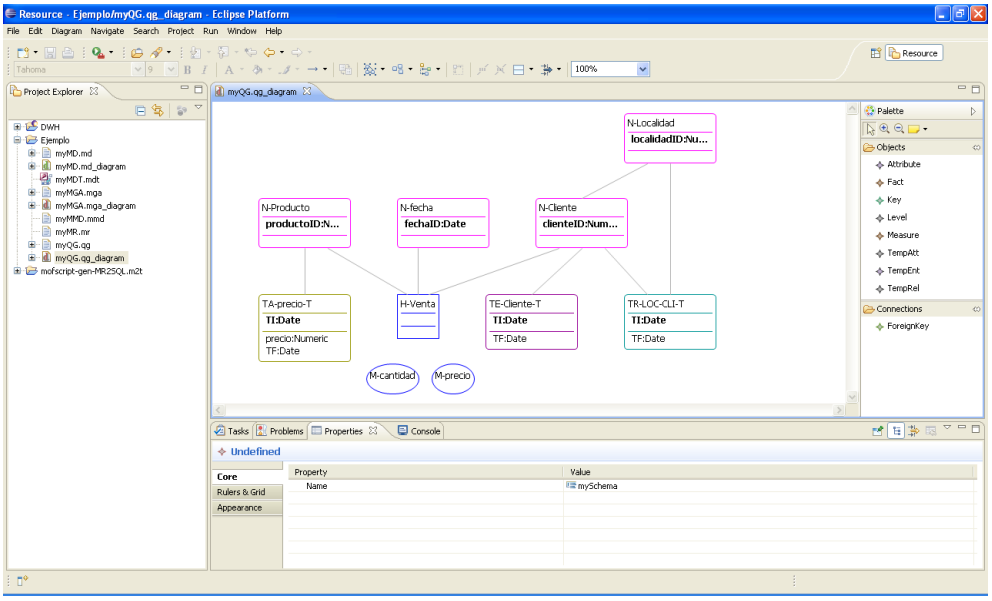


Figura 11.21. Generación del **QG**

11.9.1. Marcado del Gafo de Consulta

Una vez generado el **QG**, el paso siguiente es el marcado del mismo para la realización específica de consultas por parte del usuario de la aplicación.

Es importante aclarar nuevamente, que las actividades anteriores son desarrolladas por el diseñador de la aplicación, en cambio, el marcado del **QG**, para la obtención de información, lo realiza el usuario final de la aplicación.

El **QG** admite consultas **MD** y temporales.

1. Seleccionando los elementos del diagrama, en la solapa de propiedades, pueden marcarse los elementos del **QG** que precisan consultarse y, en el caso de consultas **MD**, el orden de ordenamiento en la función de agregado utilizada.
2. En la figura 11.22, se muestra e **QG** que está siendo marcado para la realización de la consulta.

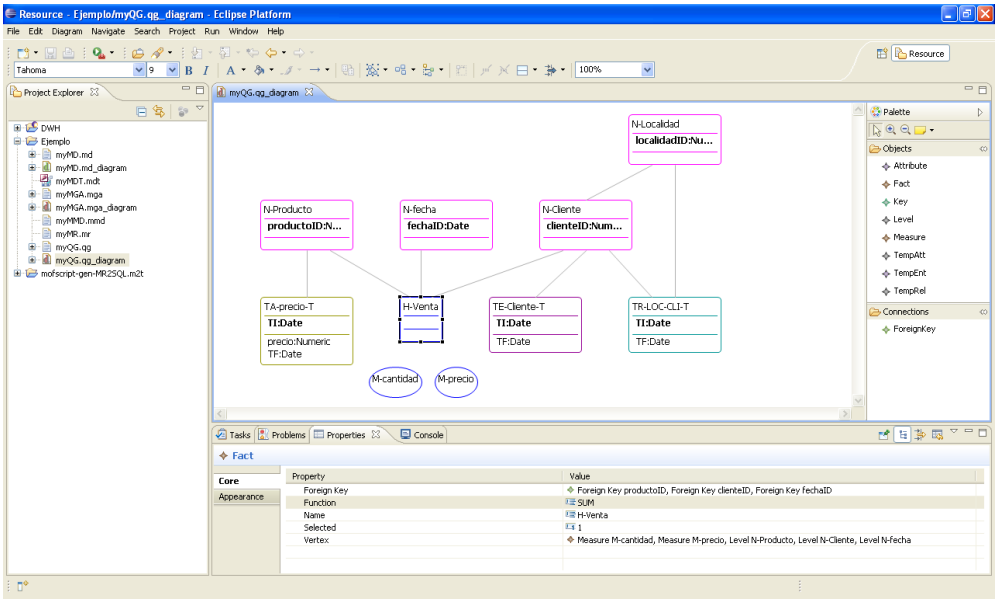


Figura 11.22. Marcado del Gafo de Consulta para consulta MD

11.9.2. Transformación a Sentencias SQL

Una vez marcado el **QG**, el paso siguiente es la obtención de las sentencias **SQL** que resuelven las consultas planteadas por el usuario. Estas transformaciones las realizamos mediante MOFScript.

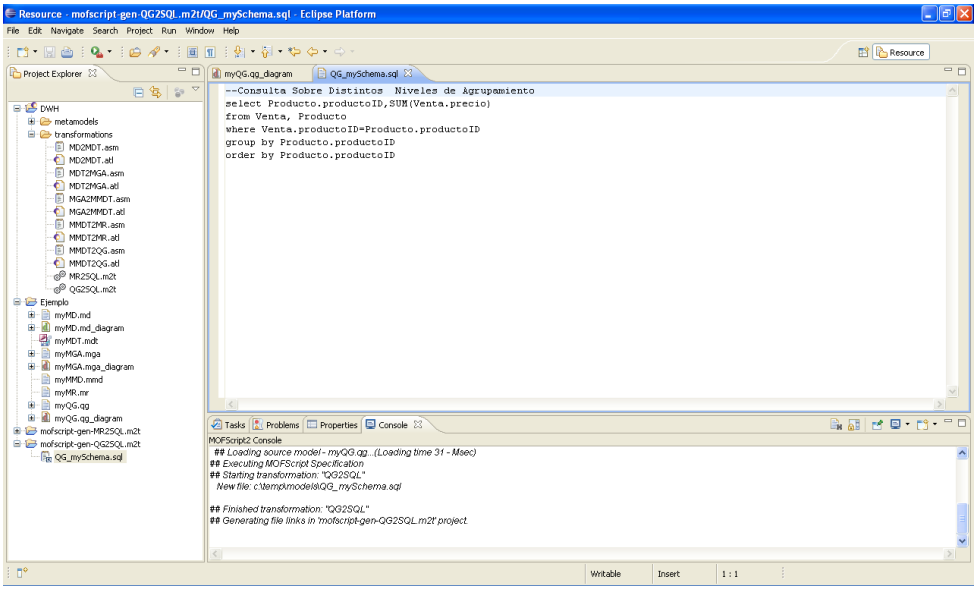


Figura 11.23. Código SQL generado

1. Para obtener el código **SQL** en texto plano para las consultas realizadas, ejecutamos la transformación QG2SQL.
2. Seleccionamos el nombre y ubicación del archivo .sql que se generará.
3. En la figura 11.23, se muestra el código **SQL** en texto plano de la consulta **MD** realizada.
4. En la figura 11.24, se muestra la realización de una consulta temporal sobre el **QG** (el hecho principal no está seleccionado).

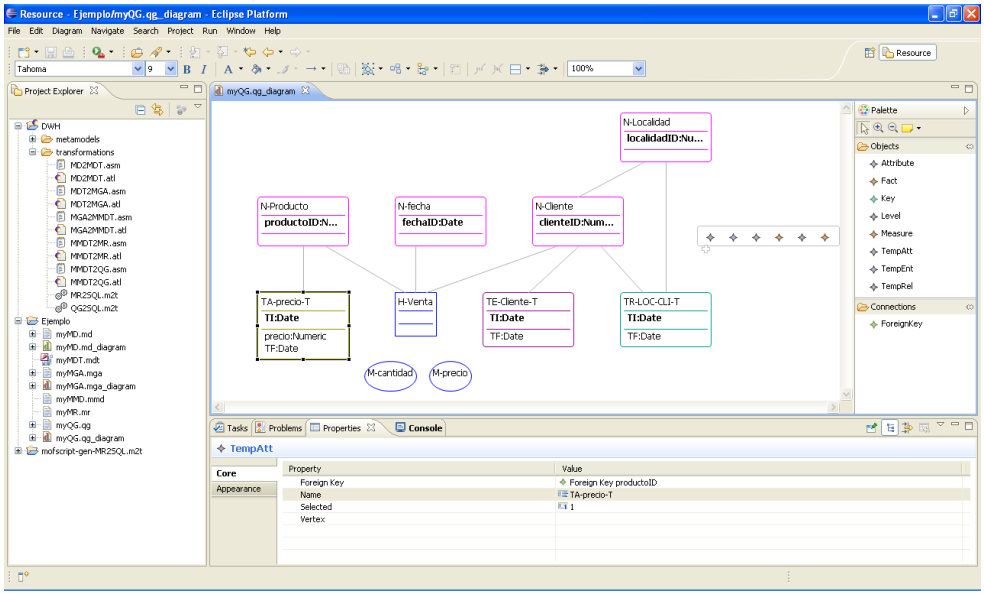


Figura 11.24. Marcado del Gafo de Consulta para consulta Temporal

5. En la figura 11.25, se muestra el código **SQL** en texto plano de la consulta temporal realizada.

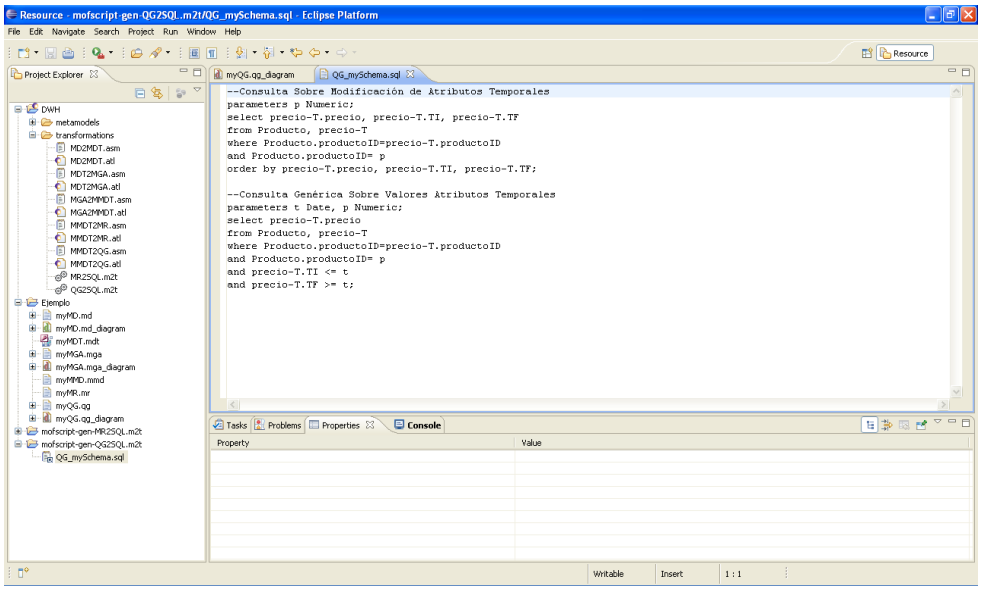


Figura 11.25. Código SQL generado

11.10. Resumen del Capítulo

El objetivo de este capítulo fue mostrar el prototipo desarrollado en ECLIPSE que permite implementar el **HDW**. Mediante el uso del ejemplo desarrollado en los capítulos precedentes detallamos, para a paso, cómo generar los modelos y las transformaciones necesarias, tanto para la obtención del modelo **TMD** como así también la realización de las consultas automáticas sobre el mismo.

Capítulo 12

Trabajos Relacionados

12.1. Introducción

En los últimos años han aparecido diversos trabajos vinculados al uso del enfoque **MDD** para el diseño de sistemas informáticos; en particular, se presentaron propuestas que utilizan **MDA** para el diseño de diferentes tipos de estructuras de almacenamiento, tales como **TDB**, **DW** y **DW** espaciales.

En este capítulo describiremos, primeramente, trabajos vinculados al uso de este nuevo paradigma de desarrollo en el diseño de estructuras de almacenamiento; a continuación, detallaremos trabajos de investigación relacionados con el uso de consultas gráficas sobre estructuras de datos. Posteriormente, estableceremos las diferencias de nuestra propuesta respecto de los trabajos relacionados. Por último, presentaremos los principales trabajos que hemos desarrollado vinculados a la temática propuesta.

12.2. MDD en el Diseño de Estructuras de Almacenamiento

Presentaremos, a continuación trabajos de investigación relacionados con el uso de **MDD** (en particular **MDA**) vinculados al diseño de estructuras de almacenamiento

- En [MT09] presentan un *framework* para el desarrollo de un modelo **MD** híbrido mediante el uso de una representación conceptual que permite derivar automáticamente un modelo lógico. El *framework* de modelado propuesto está alineado con en enfoque **MDA**: primero se especifican los requerimientos de información mediante un **CIM**; luego, se describe cómo derivar un **PIM** inicial para el modelo **MD** a partir del **CIM**. Posteriormente, se corrobora el **PIM** con información que proviene de las fuentes de datos que pueblan en **DW**. Usan **CWM** para construir diversos **PSM** para diferentes tecnologías de **DB**. Por último, establecen formalmente relaciones **QVT** entre modelos y transformaciones Mof2Text para obtener el código correspondiente.

- En [GT08] presentan un enfoque **MDA** para el diseño de **DW** espaciales. La propuesta incluye una extensión espacial del modelo **MD**, donde introducen una descripción geométrica de algunos elementos. Luego, se definen formalmente un conjunto de reglas de transformación usando el lenguaje **QVT** para lograr una representación lógica en forma automática. Por último, implementan la propuesta en una herramienta basada en Eclipse. En el trabajo presentado, utilizan un perfil espacio **MD** como **PIM** y el paquete relacional **CWM** como **PSM.**

- En [NP08] proponen un enfoque **MDA** para la transformación de un modelo de datos temporal a un esquema relacional. Presentan un modelo de datos que describe atributos e interrelaciones temporales y un conjunto de transformaciones informales que permiten convertir, primeramente, un modelo de datos estándar a un modelo temporal y luego, de éste, a un modelo relacional. Luego definen, en el marco **MDA**, los metamodelos de datos temporal y relacional y, mediante el estándar **QVT**, desarrollan las transformaciones de **PIM** a **PIM** y de **PIM** a **PSM** que permiten obtener una implementación, en un modelo lógico relacional, de una **HDB**.

- En [MOT07] se presenta una aproximación basada en ingeniería inversa dirigida por modelos para el desarrollo de **DW**. El enfoque consiste, primero, en analizar la fuentes de datos, transformando su representación lógica en un **PSM**; luego, el **PSM** se marca con conceptos **MD**, y finalmente, se obtiene un **PIM** a partir del **PSM** marcado que representa el modelo conceptual **MD.** En el trabajo propuesto, se definieron un conjunto de relaciones **QVT**, tanto para marcar el **PSM** como para obtener el **PIM** de manera automática.

- En [VVCM07] se presenta el desarrollo de una **ORDB** en el marco de **MDA**. Se definen transformaciones para generar el esquema de la base de datos (**PSM**), partiendo del esquema conceptual de datos (**PIM**) representado mediante diagramas de clases **UML**. El enfoque propone la creación de dos **PSM**; uno, para representar el esquema de la base de datos en el estándar **SQL**:2003, el otro, para un producto concreto: Oracle10g. Además, se presentan los metamodelos y los perfiles **UML** necesarios y la definición de reglas para la transformación del **PIM** en los **PSM**, primeramente en lenguaje informal y, formalmente, mediante gramática de grafos.

- En [TFP07a] proponen una extensión del paquete relacional **CWM** para representar, a nivel lógico, todos los requisitos de seguridad y auditoria capturados durante la fase de modelado conceptual del **DW**. La propuesta, alineada con **MDA,** permite considerar aspectos de seguridad en todas las fases de diseño del **DW**, desde la construcción del **PIM,** con la propuesta del modelado conceptual basado en **UML**, como en su correspondiente representación a nivel lógico.

- En [TFP07b], [TFP07c], [Sol+07] y [STBF09] presentan, con enfoques similares, un conjunto de transformaciones **MDA** mediante el estándar **QVT** para transformar un modelo conceptual **MD** seguro en un esquema lógico relacional seguro. Proponen un conjunto de definiciones de **PIM** y **PSM** seguros que permiten definir una

arquitectura **MDA MD** segura. Proponen, como principal ventaja de su propuesta, la creación de un solo **PIM** seguro y, de manera automática, la obtención del **RM** con su correspondiente código para dicha plataforma relacional. Esto, arguyen, permite ahorrar tiempo y esfuerzo para los desarrolladores. En particular, en [Bla+09] desarrollan, usando transformaciones **QVT**, un **DW** seguro mediante la generación de código **MD** seguro en una herramienta **OLAP** especifica: *SQL Server Analysis Services.*

- En [NP07] proponen, en el marco **MDA**, un conjunto de transformaciones para derivar un **TMD** a partir de un modelo de datos con marcas temporales. Se presenta una metodología semi-automática para generar un esquema relacional de un **TDW** a partir de un modelo de datos temporal; primero, se presenta un algoritmo recursivo que permite crear un **AG** a partir de un modelo de datos; luego, se establece informalmente la transformación del **AG** al modelo **MD** y de éste al esquema relacional; a continuación, se presentan los metamodelos del modelo de datos temporales, del **AG** del modelo **MD** y del relacional. Finalmente, se presentan las transformaciones formales utilizando sentencias **OCL**.

- En [MTL06] se presenta un enfoque para asegurar la corrección de un **DW** conceptual a partir de los datos fuentes que lo pueblan. Primero, se obtiene un esquema conceptual **MD** del **DW** a partir de los requerimientos de los usuarios. Luego, se verifica y se fuerza su corrección a partir de los datos fuentes, usando un conjunto de relaciones **QVT** basado en formas normales **MD.** Además, proponen el uso de formas normales **MD** para asegurar que el modelo conceptual **MD** satisfaga otras propiedades deseables, tales como: fidelidad, completitud, no redundancia, y sumarización sensible al contexto. La propuesta permite integrar las relaciones **QVT**, dentro del enfoque **MDA**, para el desarrollo de **DW.**

- En [ZC06] se presenta una metodología semiautomática para el diseño conceptual de un **DW**. La propuesta está estructurada en tres fases. En la primera, se consolidan los requisitos de los usuarios usando entrevistas y lluvias de ideas y se extraen un conjunto de de esquemas **MD** a partir del esquema conceptual de la **ODB**; argumentan que esa actividad, por su naturaleza, es un proceso difícil e interpersonal y por lo tanto difícil de automatizar. En contraste, las fases dos (identificar y elegir los requisitos de usuario) y tres (seleccionar y refinar los esquemas **MD**), como parten de la información estructurada y formal (el esquema), pueden ser automatizadas. Se utiliza el enfoque **MDA** para el proceso de desarrollo y el lenguaje **ATL** para implementar las reglas de transformación.

- En [MTSP05] presentan un *Framework,* orientado a **MDA,** para el desarrollo de un **DW**. El marco propuesto tiene como objetivo el diseño completo de un sistema de **DW,** donde se alinean cada una de las etapas de desarrollo con los diferentes puntos de vista de **MDA.** En el trabajo se presenta el MD²A, un enfoque que aplica **MDA** para el desarrollo de un **DW**; se definen un **MD PIM**, un **MD PSM** y las correspondientes transformaciones usando el lenguaje **QVT**. El **PIM** está modelado usando perfiles **UML** y, el **PSM,** usando el paquete relacional **CWM.**

12.3. Consultas Gráficas

Presentaremos, a continuación, los principales trabajos de investigación vinculados al diseño de consultas gráficas.

- En [AYW08] se desarrolla un prototipo para la visualización de la ejecución de consultas en un **DB** Oracle. Plantean que el proceso subyacente que permite la ejecución de consultas en una base de daros es fundamental para comprender el funcionamiento de un **DBMS** y que esos procesos son complejos y pueden ser difíciles de explicar e ilustrar y proponer. Para solucionar ese problema, proponen un sistema de simulación de consultas basados en Java que permite a los estudiantes visualizar los pasos involucrados en el proceso **DML** de consultas. El objetivo es permitir a los estudiantes, en forma interactiva, examinar y comprender el proceso de consultas

- En [RTTZ08] se presenta un modelo conceptual **MD** como una constelación de hechos y dimensiones compuestas por multijerarquías, proponen un algebra orientada al usuario para consultas complejas y un lenguaje gráfico basado en ella para facilitar la especificación de consultas **MD**. Este modelo soporta un álgebra de consulta que define un núcleo mínimo de operadores, que producen tablas **MD** para mostrar los datos analizados. El algebra, orientado al usuario, soporta análisis complejos a través de operadores de búsqueda avanzada y operadores binarios. También provee un lenguaje gráfico, basado en esta álgebra, que facilita la especificación de consultas **MD**. Estas manipulaciones gráficas se expresan mediante una constelación de hechos que permiten producir tablas **MD.**

- En [MS06] se presenta un *framework* para el modelado de jerarquías dimensionales complejas y una transformación a una estructura de navegación basada en esquemas para una interface visual **OLAP**. Proponen una clasificación de comportamientos dimensionales: jerarquías no sumarizable, jerarquías desbalanceadas o no estrictas, esquemas dimensionales heterogéneos, etc. El esquema de transformación está planteado en dos fases, primero, fuerzan la sumarización a jerarquías simples y homogéneas y, segundo, reorganizan el esquema jerárquico complejo en un conjunto adecuado de subdimensiones que evitan los problemas iniciales.

- En [FKSS06] se presenta una interface integrada para consultas y visualización de conjuntos de resultados para búsqueda y descubrimiento de patrones temporales. El trabajo propone un lenguaje visual de consultas temporales y una visualización navegable de conjuntos de datos temporales. La interface presentada ofrece búsquedas *ad hoc* y permite descubrir patrones temporales en un conjunto datos multivariables. La herramienta gráfica facilita, además, definir por parte del usuario patrones de consulta.

- En [ACDS02] presentan un sistema de consultas basados en iconos que permiten la interacción de usuarios noveles con una base de datos relacional. El objetivo es ayudar a los usuarios no expertos en el aprendizaje y comprensión del modelo de datos relacional y de un lenguaje de consulta textual, tal como **SQL**, a través de la utilización

de una metáfora icónica. El trabajo presenta, también, los resultados del uso la herramienta visual con estudiantes para evaluar la efectividad de la propuesta.

- En [Rei02] se describe un sistema que permite al usuario definir visualizaciones del software en forma rápida y efectiva. El sistema utiliza un lenguaje de consultas visuales sobre una variedad de fuentes de datos para permitirle al usuario especificar qué información es relevante para la comprensión de las tareas y para correlacionar la información. Esto le provee, luego, mecanismos que le permiten al usuario seleccionar y personalizar una apropiada visualización de los datos.

- En [ON01] se presenta un lenguaje de consultas que permite la abstracción conceptual de consultas en base de datos. La propuesta hace uso de la riqueza de los modelos de datos semánticos para facilitar la formulación de consultas en base de datos relacionales. El lenguaje propuesto, denominado CQL (Conceptual Query Language), exige una mínima demanda cognitiva a los usuarios finales. Desarrollan un *framework* que utiliza la semántica relacional de los modelos de datos para hacer transparente la complejidad técnica de los lenguajes de consultas de base de datos. En la propuesta, usan las relaciones semánticas para construir automáticamente un grafo de consultas y un pseudo código en lenguaje natural para generar el código **SQL**.

- En [TSH01] presentan una interface gráfica para explorar grandes bases de datos **MD**, denominada Polaris. Como característica distintiva, la propuesta incluye una interface para la construcción de especificaciones visuales y la posibilidad de generar un conjunto de consultas relacionales a partir de dichas especificaciones.

- En [KG95] se introducen un conjunto de construcciones visuales que permiten al usuario construir consultas en un formato modular basado en una extensión temporal de un modelo **ER**. En dicho trabajo presentan un conjunto de construcciones temporales visuales que permiten al usuario construir consultas en forma modular.

12.4. Trabajos Relacionados vs. Nuestra Propuesta

Detallaremos, a continuación, una caracterización de los trabajos relacionados vinculados al enfoque de desarrollo utilizado, al diseño de las estructuras de almacenamiento y a las consultas gráficas; luego, estableceremos las principales diferencias de nuestra propuesta respecto de aquellas.

12.4.1. Utilización del Enfoque MDD

Los trabajos relacionados al diseño de estructuras de datos que utilizan el enfoque **MDD** presentados por otros autores, plantean los siguientes objetivos: mejorar la productividad en el desarrollo de un **DW,** en el marco **MDA**, ([MT09], [MOT07], [MTL06], [ZC06], [MTSP05]); utilizar el enfoque **MDA** en el diseño de **DW** espaciales ([GT08]); considerar aspectos de seguridad en el **DW** ([STFP07a], [STFP07b], [STFP07c], [Sol+07], [STBF09]) e implementarlos en una herramienta

OLAP especifica ([Bla+09]) o, por último, utilizar el enfoque **MDA** para el desarrollo de un **ORDB** ([VVCM07]).

El enfoque utilizado en todos los casos por los trabajos presentados es en el marco de **MDA**, esto implica el uso de estándares asociados propuestos por la **OMG**, (**UML** y *profiles*, **OCL**, **XMI**, **CWM**, **QVT**).

En nuestra propuesta, el enfoque utilizado es **MDD**, en particular **DSM**, creamos modelos específicos del dominio utilizando un lenguaje focalizado y especifico para cada uno de ellos (**DSL**); en particular, no utilizamos **CWM** sino metamodelos más simples, instancias de **MOF**, para los modelos de datos, el modelo **MD** y el **RM;** además, diseñamos metamodelos específicos para los modelos utilizados en el proceso: para la construcción del **AG**, el metamodelo **AG** y para la construcción del **QG**, el metamodelo **QG.** No usamos **UML** ni *profiles* para el diseño de **PIM** del modelo de datos fuente, ya que consideramos que el modelo **ER** es más expresivo para el modelado de datos. Aunque sí utilizamos **OCL** para establecer restricciones sobre los metamodelos propuestos.

Respecto de las estructuras de almacenamiento, se diferencia de los trabajos referenciados, principalmente, en el modelo **MD** propuesto: el **HDW** representa una nueva estructura de datos que combina e integra, en un solo modelo, un **DW** y un **HDB**; este modelo incluye, además del hecho principal de análisis, estructuras temporales vinculadas a los niveles de las jerarquías dimensionales que posibiliten registrar los datos y recuperar la información que variase en el tiempo.

12.4.2. Consultas Gráficas Automatizadas

Respecto a las consultas gráficas, los trabajos relacionados pueden clasificarse en: aquellos vinculados con consultas a base de datos **MD** ([RTTZ08], [MS06], [TSH01]); a los lenguajes relacionadas con consultas en **TDB** ([FKSS06]); a los lenguajes vinculados a consultas en **BD** ([Rei02], [ON01], [KG95]); por último, aquellos vinculados al diseño de interfaces visuales utilizados para el aprendizaje, tanto en la creación de consultas ([ACDS02]), como para las comprensión del proceso subyacente del **DBMS** ([AYW08])

Dentro de esta clasificación, nuestra propuesta es, en parte, una conjunción de las dos primeras: la interface gráfica presentada le permite al usuario final realizar consultas **MD** e históricas en forma automática.

No obstante, lo más significativo que diferencia nuestra propuesta del resto es el diseño de un entorno gráfico, derivado automáticamente del **HDW,** que permite la generación automática de sentencias **SQL** para realizar consultas **MD** e históricas**.** Hasta donde hemos visto, no hemos encontrado investigaciones que utilicen el enfoque **MDD,** ni para la derivación de un entorno gráfico de consultas, ni así tampoco para la creación de sentencias **SQL,** en forma automática, sobre una estructura **TMD**.

12.5. Publicaciones Vinculadas a la Tesis

A continuación, se listan los principales trabajos publicados cuya temática está vinculada al de los objetivos plateados en la tesis:

- Carlos Neil, Jerónimo Irazábal, Marcelo De Vincenzi, Claudia Pons. **Graphical Query Mechanism for Historical DW within MDD**. XXIX Conferencia Internacional de la Sociedad Chilena de Ciencia de la Computación (IEEE Press). Chile 2010

- Neil, Carlos, Pons Claudia. **Aplicando QVT en la Transformación de un Modelo de Datos Temporal.** Jornadas Chilenas de Computación. Punta Arenas. Chile. 2008

- Neil Carlos, Baez Martín, Pons Claudia. **Usando ATL en la Transformación de Modelos Multidimensionales Temporales**. XIII Congreso Argentino de Ciencias de la Computación. Corrientes y Resistencia, Argentina. 2007.

- Neil Carlos, Pons Claudia. **Aplicando MDA al Diseño de un Data Warehouse Temporal.** VII Jornada Iberoamericana de Ingeniería de Software e Ingeniería del Conocimiento. Lima, Perú. 2007.

- Neil Carlos, Pons Claudia. **Diseño Conceptual de un Data Warehouse Temporal en el Contexto de MDA**. XII Congreso Argentino de Ciencias de la Computación. CACIC. San Luis. Argentina. 2006.

- Neil Carlos, Ale Juan. **A Conceptual Design for Temporal Data Warehouse**. 31° JAIIO. Santa Fe. Simposio Argentino de Ingeniería de Software. 2002.

12.6. Resumen del Capítulo

El objetivo de este capítulo fue presentar, primeramente, los principales trabajos de investigación vinculados al uso del enfoque **MDD** en el diseño de estructuras de almacenamiento; a continuación, se detallaron diferentes trabajos de investigación relacionados con el uso de consultas gráficas sobre estructuras de datos. Posteriormente, establecimos las diferencias de nuestra propuesta respecto de los trabajos relacionados. Al final, presentamos los principales trabajos que hemos desarrollado vinculados a la temática propuesta.

Capítulo 13

Conclusiones

13.1. Introducción

En este último capítulo, presentaremos primero un resumen de la tesis presentada, luego, las contribuciones principales y, por último, los trabajos futuros. En la sección *resumen*, sintetizaremos la propuesta de la tesis respecto del diseño de **HDW** en el contexto de **MDD,** mostraremos las diferencias entre nuestro planteo respecto de los trabajos vinculados; en la siguiente sección, *contribuciones principales*, detallaremos los aportes más significativo de nuestra propuesta; finalmente, en la sección *trabajos futuros*, presentaremos distintas líneas de investigación que permitirán continuar con el trabajo desarrollado en la tesis.

13.2. Resumen

La propuesta principal de la tesis es la creación de un modelo y un método para el diseño automático de un **HDW**, esto es, una nueva estructura de almacenamiento de datos que combina e integra en un solo modelo, un **DW** y un **HDB**; este modelo **TMD** incluye, además del hecho principal de análisis, estructuras temporales vinculadas a los niveles de las jerarquías dimensionales que posibiliten registrar los datos y recuperar la información que variase en el tiempo.

Utilizando el paradigma **MDD,** el **HDW** se genera a partir de un método de diseño que, tomando como fuente un modelo de datos conceptual expresado en un modelo **ER** y mediante sucesivas transformaciones, permite obtener una implementación lógica en un **RDBMS.**

El trabajo incluye, además, el desarrollo de un entorno gráfico derivado automáticamente del **HDW,** mediante el enfoque **MDD,** que le permite realizar al usuario final, sobre una interface gráfica e intuitiva, consultas sobre la estructura **TMD**; este entorno gráfico genera automáticamente sentencias **SQL** que permite realizar, sobre el **HDW,** tanto las consultas características de un **DW** como las típicas de un **HDB.**

Por último, creamos un prototipo, basada en tecnología ECLIPSE**,** que implementa el método de diseño del **HDW,** la interface gráfica de consultas y la realización de sentencias **SQL.**

Nuestra propuesta se diferencia de otros trabajos vinculados a la temática propuesta en la tesis en varios aspectos. Primero, la estructura de almacenamiento propuesta (**HDW**) es novedosa respeto de otros trabajos, donde el enfoque principal está centrado en **DWs**, **DWs** espaciales y **ORDBs**; segundo, el enfoque utilizado es **MDD**, en particular **DSM**, nuestra propuesta utiliza **DSL**, a diferencia del enfoque **MDA,** donde promueven el uso de estándares **OMG**; tampoco usamos **UML** ni *profiles* para el diseño del **PIM** del modelo de datos fuente; esto simplifica el trabajo del diseñador, ya que solamente debe aprender una notación sencilla, con escasa cantidad de elementos y enfocada en su dominio de experticia. Otra diferencia significativo, respecto de trabajo anteriores, es que, hasta donde hemos visto, no hemos encontrado investigaciones que utilicen el enfoque **MDD,** ni para la derivación de un entorno gráfico de consultas, ni así tampoco para la creación de sentencias **SQL,** en forma automática, sobre una estructura **TMD.**

Respecto de la aplicabilidad del paradigma **MDD**, es importante destacar que la mayoría de los trabajos presentados están vinculados, tal como hemos descrito en este capítulo, con estructuras de almacenamiento. Sin duda la principal causa es la relativa simplicidad que representa la transformación de estructuras estáticas a diferencia de la parte dinámica, esto es, los aspectos funcionales de los sistemas informáticos ya que éstos resultan más complejos de capturar en su esencia. Respecto a este último punto, nuestra propuesta considera aspectos vinculados con el comportamiento, en cierta forma, al permitir derivar consultas sobre la base de datos.

13.3. Contribuciones Principales

Como principales contribuciones de nuestro trabajo, destacamos la creación de un nuevo modelo de datos temporal (**HDB**) simplificado que permite registrar la variación de los valores de atributos, entidades e interrelaciones que se modifiquen en el tiempo; utilizando este modelo, presentamos una nueva estructura de almacenamiento de datos (**HDW**), que combina e integra en un solo modelo, un **DW** con el **HDB.**

A partir de los modelos propuestos, presentamos un método de diseño que, comenzando con un modelo de datos conceptual y mediante sucesivas transformaciones, permite obtener una representación lógica de un **HDW** en un **RDBMS** y lo implementamos, mediante el enfoque **MDD.**

Por otro lado, vinculado a la recuperación de información por parte del usuario final, presentamos un entorno gráfico derivado automáticamente del **HDW,** en el marco **MDD,** para la realización de consultas sobre la estructura **TMD** que permite la generación automática de sentencias **SQL** para realizar, tanto las consultas características de un **DW,** como las típicas de un **HDB.**

Por último, creamos un prototipo, basada en tecnología ECLIPSE**,** que implementa el método de diseño del **HDW,** la interface gráfica de consultas y la realización de sentencias **SQL** y que permitió, mediante su uso, corroborar parcialmente la propuesta presentada.

13.4. Trabajos Futuros

A partir de la tesis presentada, se abre un abanico de posibles líneas de investigación asociadas que no fueron consideradas en el desarrollo de la tesis pero que ameritan ser tenidas en cuenta en futuros trabajos.

Detallaremos, a continuación, los temas que no hemos considerados y cuya solución implica una línea de investigación a desarrollar:

- **Creación de un DLS para transformaciones de DW:** en este trabajo hemos utilizado el lenguaje **ATL** para definir las transformaciones entre los modelos propuestos. **ATL** es un lenguaje específico del dominio de las transformaciones, es decir, provee construcciones sintácticas focalizadas en la definición de transformaciones entre modelos. Adicionalmente estos lenguajes de transformación (LT) pueden admitir un grado más de especialización, es decir, podríamos definir lenguajes de transformación específicos de dominio (LTED). En nuestro caso, podríamos definir un LT específico para transformaciones de **DW**. Contar con un lenguaje específico versus un lenguaje más general, como **ATL**, facilitaría notablemente la definición y la reutilización de las transformaciones.

- **Verificación formal de las propiedades de las transformaciones:** Se espera que cada transformación realice solamente modificaciones sintácticas sobre los modelos, pero respetando su semántica. La verificación de esta propiedad (semantic preserving transformations) de las transformaciones es un tema muy complejo; sin embargo, al restringirlo al dominio específico de los **DW** sería posible obtener mejores resultados.

- **Creación de una interface de consulta eficiente:** la interface gráfica presentada en el método de diseño y posteriormente implementada en el prototipo desarrollado en ECLIPSE es, sin duda, *naïf*. El uso de interfaces gráficas implica consideraciones sobre tipo de iconos a utilizar, diferentes colores, distribución de los iconos en la pantalla, etc. que no han sido considerados en el actual trabajo y que deberán considerarse en ulteriores investigaciones.

- **Considerar requisitos del usuario en el diseño del HDW:** nuestra trabajo utiliza como modelo fuente, para el proceso de transformaciones, un modelo de datos conceptual, expresado en un **ER**, del cual consideramos que representa los requisitos de información de los usuarios, al menos en lo referente a la aplicación transaccional. No hemos considerado cómo evaluar y plasmar en el diseño del **HDW** los requisitos de información del usuario en aspectos **MD** y temporales. En parte esto es así ya que las transformaciones comienzan por el **PIM**, sin considerar el **CIM**, en forma explícita. La transformación de **CIM** a **PIM**, es un ámbito no muy desarrollado, que abre una línea de investigación a considerar en futuros trabajos.

- **Ampliación del modelo de datos:** el modelo de datos temporal presentado utilizado es el modelo **ER** estándar, donde solo se incluyen las construcciones básicas y, mediante ellas, la captura, en forma implícita, de los aspectos temporales. No hemos considerado inicialmente en nuestra modelo de datos interrelaciones de grado > 2 y tampoco extendimos su semántica con conceptos tales como generalizaciones, agregaciones y construcciones temporales. Estos aspectos ameritan ser

evaluados y considerados en futuras ampliaciones del modelo de datos utilizado.

- **Restricciones de Integridad Temporal**: el modelo, tal cual está planteado, no contempla restricciones con respecto a las actualizaciones, pudiendo darse el caso de solapamientos temporales. El establecimiento de restricciones con respecto a la inserción, borrado y modificación de valores temporales debería impedir posibles inconsistencias en la base de datos. Además, como se planteo en el capítulo 3, es necesario considerar el tiempo válido de las entidades temporales como subconjunto de los tiempos validos de atributos, entidades e interrelaciones involucradas. Estos temas constituyen un ámbito de investigación a considerar.

- **Derivación automática del proceso ETL:** otro aspecto que no hemos considerado en nuestro diseño del **HDW** es cómo realizar el proceso de **ETL**. Este proceso es importante porque es el encargado de integrar datos de diferentes fuentes heterogéneas. Para la construcción del modelo conceptual partimos de un modelo **ER** que representa, a una base de datos no histórica. La ampliación a un modelo temporal no implica, conceptualmente, mayores inconvenientes. La carga de datos históricos sí requerirá considerar estrategias que permitan el poblado del **HDW** a partir de datos provenientes de copias de resguardo almacenados en distintos soportes y formatos. Una línea de investigación a considerar es, en el contexto de **MDD**, la generación automática de código de procesos **ETL.**

- **Integración de la herramienta de transformación con DBMS's**: el prototipo presentado culmina su proceso de transformación con la generación, por un lado, de sentencias **SQL** para la creación de estructuras de almacenamiento y, por otro lado, en sentencias **SQL** que resuelven las consultas sobre el **HDW;** ambos tipos de sentencias no son ejecutadas directamente sino que, el texto generado, debe ser ejecutado posteriormente en un **DBMS.** Por lo tanto, otra línea de investigación a seguir es **I**a integración directa de la herramienta de transformación desarrollada en ECLIPSE con un **DBMS.**

- **Uso de un PSM Objeto Relacional:** hemos utilizado el **RM**, en particular el estándar **SQL**92 para el desarrollo del **PSM.** Las bases de datos **ORDB** (el estándar **SQL**2003) presentan construcciones tales como los tipos abstractos de datos definidos por el usuario, que permitirían una más simple representación el modelo temporal propuesto, permitiendo desarrollar una línea de investigación asociada.

13.5. Resumen del Capítulo

El objetivo de este capítulo fue presentar, primero, una síntesis de la tesis respecto del diseño de **HDW** en el contexto de **MDD,** donde mostramos las diferencias entre nuestro planteo respecto de los trabajos vinculados. Luego, detallamos los aportes más significativos de nuestra tesis. Finalmente, presentamos distintas líneas de investigación que permitirán continuar con el trabajo presentado.

Anexo I

ECLIPSE

I.1. Introducción

Eclipse es, principalmente, una comunidad que fomenta el código abierto. Los proyectos en los que trabaja se enfocan principalmente en construir una plataforma de desarrollo abierta. Esta plataforma puede ejecutarse en múltiples sistemas operativos, lo que lo convierte en una multi-plataforma. Eclipse está compuesto por *frameworks* extensibles y otras herramientas que facilitan construir y administrar software. Permite que los usuarios colaboren para mejorar la plataforma existente y extiendan su funcionalidad a través de la creación de *plugins*. Fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas *VisualAge*. Inicialmente fue un entorno de desarrollo para Java, escrito en Java, pero luego fue extendiendo el soporte a otros lenguajes de programación.

Eclipse ganó popularidad debido a que su plataforma proporciona el código fuente y esta transparencia generó confianza en los usuarios. Hace algunos años se creó la Fundación Eclipse, una organización independiente sin fines de lucro que fomenta una comunidad de código abierto. La misma trabaja sobre un conjunto de productos complementarios, capacidades y servicios que respalda y se encarga del desarrollo continuo de la plataforma. La comunidad Eclipse trabaja actualmente en 60 proyectos.

En este anexo solamente pondremos foco en el proyecto dedicado a promover tecnologías de desarrollo basadas en modelos llamado Eclipse *Modeling Project* (http://www.eclipse.org/modeling/). Este proyecto está compuesto por otros subproyectos relacionados. Los subproyectos que forman Eclipse Modeling Project, son:

- *Abstract Syntax development (EMF)*
- *Concrete Syntax development (GMF)*
- *Model Transformation (QVT, JET, MOF2Text)*
- *Standards Implementations (UML2, OCL, OMD, XSD)*
- *Technology & Research (GMT, MDDi, Query, Transaction, etc)*

Describiremos los dos primeros, los cuales ya son proyectos maduros y cuyas implementaciones han ido evolucionando en los últimos años. La primera

versión de **EMF** se lanzó en el año 2003 y la primera versión de **Graphical Modeling Framework** (**GMF)** en el año 2006. [PGP09]. Para más información http://www.eclipse.org/

I.2. Eclipse Modeling Framework (EMF)

El proyecto **EMF** es un *framework* para modelado, que permite la generación automática de código para construir herramientas y otras aplicaciones a partir de modelos de datos estructurados. La información referida a este proyecto, así como también la descarga del *plugin* pueden encontrarse en [EMF]. **EMF** comenzó como una implementación del metalenguaje **MOF**. En los últimos años se utilizó para implementar una gran cantidad de herramientas lo que permitió mejorar la eficiencia del código generado. Actualmente el uso de **EMF** para desarrollar herramientas está muy extendido. Se usa, por ejemplo, para implementar **XML,** *Schema Infoset Model* (XSD), Servicio de *Data Objects* (SDO), UML2, y *Web Tools Platform* (WTP) para los proyectos Eclipse. Además **EMF** se utiliza en productos comerciales, como Omondo, EclipseUML, IBM *Rational* y productos *WebSphere*.

EMF permite usar un modelo como el punto de partida para la generación de código, e, iterativamente, refinar el modelo y regenerar el código, hasta obtener el código requerido. Aunque también prevé la posibilidad de que el programador necesite modificar ese código, es decir, se contempla la posibilidad de que el usuario edite las clases generadas, para agregar o editar métodos y variables de instancia. Siempre se puede regenerar desde el modelo cuando se necesite, y las partes agregadas serán preservadas durante la regeneración.

I.2.1. Descripción

En Eclipse, la generación de código es posible a partir de la especificación de un modelo. De esta manera, permite que el desarrollador se concentre en el modelo y delegue en el *framework* los detalles de la implementación.

El código generado incluye clases Java para manipular instancias de ese modelo como así también clases adaptadoras para visualizar y editar las propiedades de las instancias desde la vista "propiedades" de Eclipse. Además provee un editor básico en forma de árbol para crear instancias del modelo. Y por último, incluye un conjunto de casos de prueba para permitir verificar propiedades. En la figura AI.1 y AI.2 pueden verse dos pantallas.

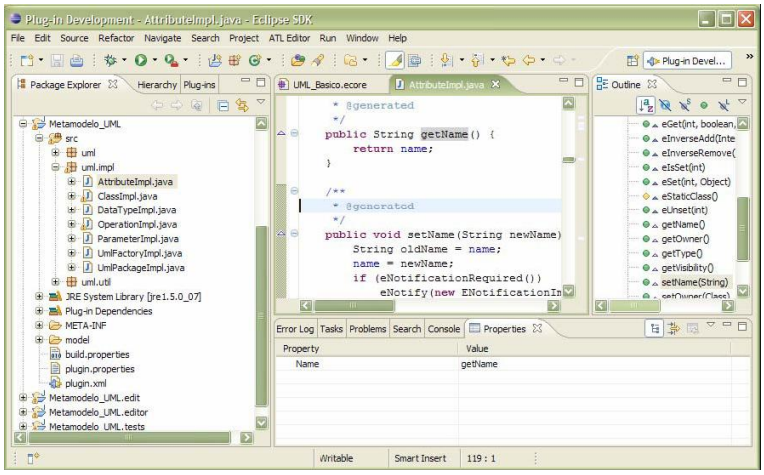


Figura AI.1. Pantalla de Eclipse (parte 1)

La primera muestra los cuatro *plugins* generados con **EMF**. Dentro de los *plugins* pueden verse los paquetes y las clases correspondientes. La segunda, muestra el editor en forma de árbol. El código generado por **EMF** es eficiente, correcto y fácilmente modificable. El mismo provee un mecanismo de notificación de cambios de los elementos, una implementación propia de operaciones reflexivas y persistencia de instancias del modelo. Además provee un soporte básico para rehacer y deshacer las acciones realizadas. Por último, establece un soporte para interoperabilidad con otras herramientas en el *framework* de Eclipse.

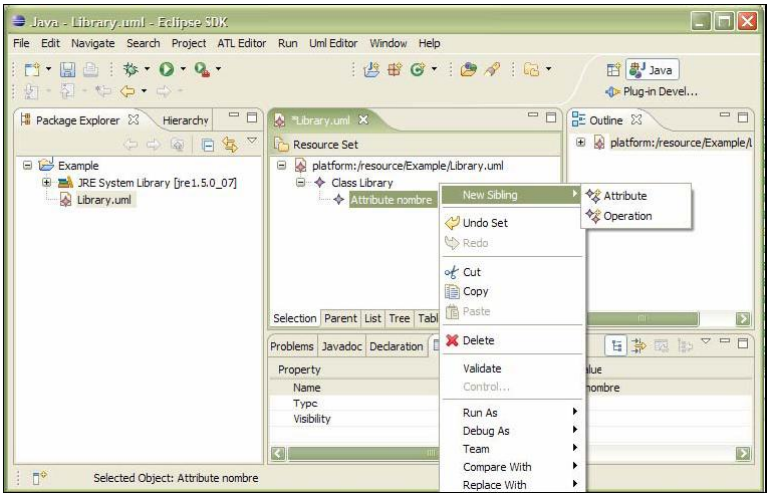


Figura AI.2. Pantalla de Eclipse (parte 2)

I.2.2. Metametamodelo

En **EMF** los modelos se especifican usando un metametamodelo llamado *Ecore*. Esta es una implementación de **Essential Meta-Object Facility** (**EMOF**). *Ecore* en sí, es un modelo **EMF** y su propio metamodelo. Existen algunas diferencias entre *Ecore* y **EMOF**, pero aun así, **EMF** puede leer y escribir serializaciones de **EMOF** haciendo posible un intercambio de datos entre herramientas. La figura AI.3 muestra las clases más importantes del meta metamodelo *Ecore*.

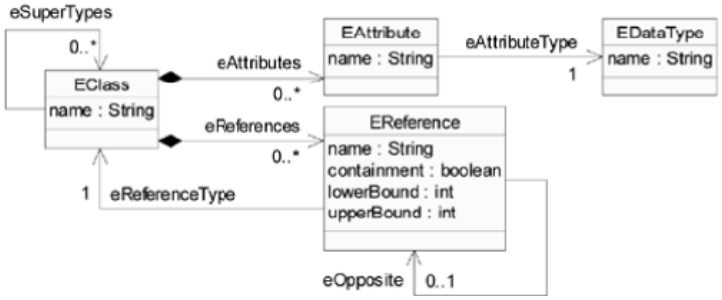


Figura AI.3. Parte del meta metamodelo *Ecore*

Ecore respeta las metaclases definidas por **EMOF**: todas las metaclases mantienen el nombre del elemento que implementan y agregan como prefijo la letra "E", indicando que pertenecen al metamodelo *Ecore*. Por ejemplo, la metaclase *EClass* implementa la metaclase *Class* de **EMOF**.

La principal diferencia está en el tratamiento de las relaciones entre las clases. **MOF** tiene a la asociación como concepto primario, definiéndola como una relación binaria entre clases y tiene finales de asociaciones, con la

propiedad de navegabilidad. En cambio, *Ecore* define solamente *EReferences*, como un rol de una asociación, sin finales de asociación ni *Association* como metaclases. Dos *EReferences* pueden definirse como opuestas para establecer una relación navegable para ambos sentidos. Existen ventajas y desventajas para esta implementación. Como ventaja puede verse que las relaciones simétricas, como por ejemplo "esposoDe", implementadas con *Association*, son difíciles de mantener ya que debe hacerse consistentemente. En cambio con *Ecore*, al ser sólo una referencia, ella misma es su opuesto, es decir, se captura mejor la semántica de las asociaciones simétricas, y no es necesario mantener la consistencia en el otro sentido. Los modelos son entonces, instancias del metamodelo *Ecore*. Estos modelos son guardados en formato **XMI**, que es la forma canónica para especificar un modelo.

I.2.3. Pasos Para Generar Código a Partir de un Modelo

Para generar el código a partir de la definición de un modelo deben seguirse los siguientes pasos:

I.2.3.1. Definición del Metamodelo

Un metamodelo puede especificarse con un editor gráfico para metamodelos *Ecore*, o de cualquiera de las siguientes formas: como un documento **XML**, como un diagrama de clases **UML** o como interfaces de Java con Anotaciones. **EMF** provee asistentes para interpretar ese metamodelo y convertirlo en un modelo **EMF**, es decir, en una instancia del metametamodelo *Ecore*, que es el metamodelo usado por **EMF** (Figura AI.4).

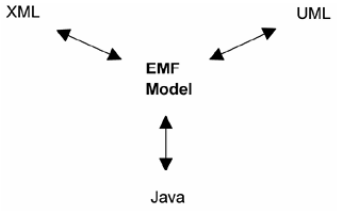


Figura AI.4. Obtención de un modelo EMF

La ventaja de partir de un modelo **UML** es que es un lenguaje conocido, por lo tanto no se requiere entrenamiento para usarlo, y que, además, las herramientas de **UML** que suelen ser más amigables y proveen una mayor funcionalidad. Sólo hay que tener en cuenta que la herramienta permita exportar un modelo *core* que pueda ser usado como entrada para el *framework* **EMF**.

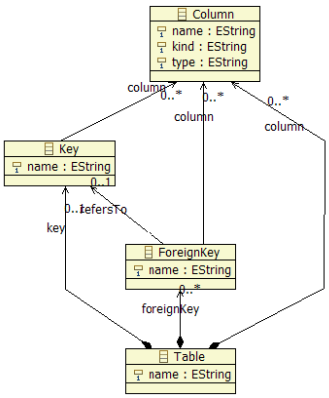


Figura AI.5. Metamodelo del Lenguaje Relacional

La opción más reciente para la especificación de un metamodelo es utilizar el editor gráfico para *Ecore* que provee **GMF**. En la figura AI.5 puede verse el metamodelo del lenguaje Relacional, hecho con esta herramienta.

I.2.3.2. El Código Generado

A partir de un modelo representado como instancias de *Ecore*, **EMF** puede generar el código para ese modelo. Además, provee un menú contextual con cinco opciones:

- *Generate Model Code*
- *Generate Edit Code*
- *Generate Editor Code*
- *Generate Test Code*
- *Generate All*

EMF permite generar cuatro *plugins*. Con la primera opción se genera un *plugin* que contiene el código Java de la implementación del modelo, es decir, un conjunto de clases Java que permiten crear instancias de ese modelo, hacer consultas, actualizar, persistir, validar y controlar los cambios producidos en esas instancias. Por cada clase del modelo, se genera dos elementos en Java: una interface y la clase que la implementa. Todas las interfaces generadas extienden directa o indirectamente a la interface *EObject*; ésta interface es el equivalente de **EMF** a *JAVA.lang.Object*, es decir, la base de todos los objetos en **EMF**. *EObject* y su correspondiente implementación *EObjectImpl* proveen la base para participar en los mecanismos de notificación y persistencia. Con la segunda opción se genera un *plugin* con las clases necesarias por el editor. Contiene un conjunto de clases adaptadoras que permitirán visualizar y editar propiedades de las instancias en la vista "propiedades" de Eclipse. Estas clases permiten tener una vista estructurada y permiten la edición de los objetos de modelo a través de comandos. Con la tercera opción se genera un editor para el modelo. Este *plugin* define, además, la implementación de un asistente para la creación de instancias del modelo. Finalmente. Con la cuarta opción se generan casos de prueba, que son esqueletos para verificar propiedades de los elementos. La última opción permite generar todo el código anteriormente mencionado en un solo paso.

En resumen, el código generado es limpio, simple, y eficiente. La idea es que el código que se genera sea lo más parecido posible al que el usuario hubiera escrito, si lo hubiera hecho a mano; pero, por ser generado, se puede tener la confianza de que es correcto. **EMF** establece un soporte para interoperabilidad con otras herramientas en el *framework* de Eclipse ya que genera código base para el desarrollo de editores basados en Eclipse.

El generador de código de **EMF** produce archivos que pretenden que sean una combinación entre las partes generadas y las partes modificadas por el programador. Se espera que el usuario edite las clases generadas, para agregar o editar métodos, variables de instancia. Siempre se puede regenerar desde el modelo cuando se necesite, y las partes agregadas serán preservadas durante la regeneración. **EMF** usa los marcadores *@generated* en los comentarios *JAVAdoc* de las interfaces, clases y métodos generados para identificar las partes generadas. Cualquier método que no tenga ese marcador se mantendrá sin cambios luego de una regeneración de código. Si hay un método en una clase que está en conflicto con un método generado, la versión existente tendrá prioridad.

I.2.3.3. Anatomía del Editor Básico Generado

En la figura AI.6 puede verse el editor generado por **EMF**. A la derecha, se encuentra la vista *outline*, que muestra el contenido del modelo que se está editando con una vista de árbol. Cuando se selecciona un elemento en el *outline*, se muestra también seleccionado en la primera página del editor, que tiene una forma de árbol similar.

Independientemente de la vista donde se seleccione el elemento, al seleccionarlo se muestran sus propiedades en la vista de propiedades; ésta vista permite editar los atributos del elemento y las referencias a otros elementos del modelo (permite seleccionar de una lista de posibles). Se pueden arrastrar con el *mouse* los elementos para moverlos, cortarlos, copiarlos y pegarlos, borrarlos, y estas acciones soportan deshacer y rehacer. Las otras páginas del editor, *Parent, List, Tree, Table, TableTree* permiten otras visualizaciones de los elementos, como en forma de tabla y en forma de lista.

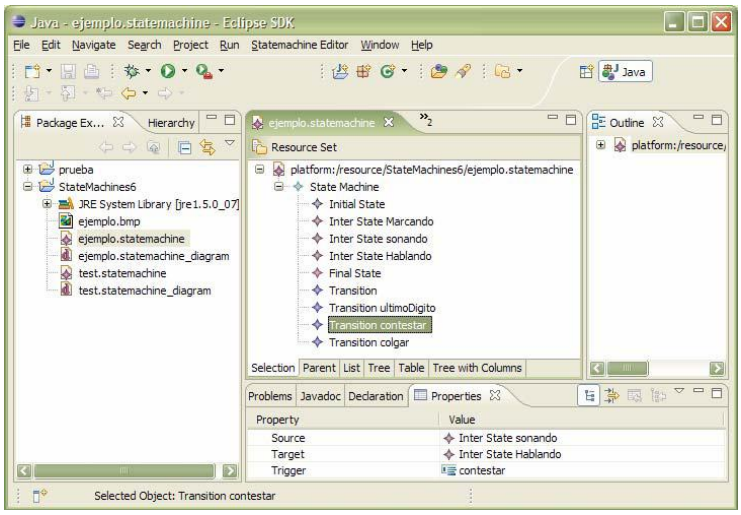


Figura AI.6. Editor Generado con EMF

I.3. Graphical Modeling Framework

Graphical Modeling Framework (GMF) [GMF] es un *framework* de código abierto que permite construir editores gráficos, también desarrollado para el entorno Eclipse. Está basado en los *plugins* **EMF** y **Graphical Editing Framework (GEF)**. Algunos ejemplos de editores generados con **GMF** son los editores **UML**, de *Ecore*, de procesos de negocio y de flujo.

I.3.1. Descripción

Los editores gráficos generados con **GMF** están completamente integrados a Eclipse y comparten las mismas características con otros editores, como vista *overview*, la posibilidad de exportar el diagrama como imagen, cambiar el color y la fuente de los elementos del diagrama, hacer *zoom* animado del diagrama, imprimirlo. Estas funcionalidades están disponibles desde la barra de herramientas, como en cualquier otro editor.

En la figura AI.7 pueden verse los principales componentes y modelos usados durante el desarrollo basado en **GMF**. El primer paso es la definición del metamodelo del dominio (archivo con extensión *.ecore*), donde se define el metamodelo en términos del metametamodelo *Ecore*. A partir de allí, se derivan

otros modelos necesarios para la generación del editor: el modelo de definición gráfica específica las figuras que pueden ser dibujadas en el editor y el modelo de definición de *tooling* contiene información de los elementos en la paleta del editor, los menús, etc. Una vez definidos estos modelos, habrá que combinar sus elementos. Esto se hace a través de un modelo que los relaciona, es decir, relaciona los elementos del modelo de dominio con representaciones del modelo gráfico y elementos del modelo de *tooling*. Estas relaciones definen el modelo de *mapping* (archivo con extensión *gmfmap*). Por último, una vez definidos todos los modelos, y relacionados a través del archivo *mapping*, **GMF** provee un generador de modelo que permite definir los detalles de implementación anteriores a la fase de generación de código (archivo *gmfgen*). La generación de código producirá un *plugin* editor que interrelaciona la notación con el modelo de dominio. También provee persistencia y sincronización de ambos.

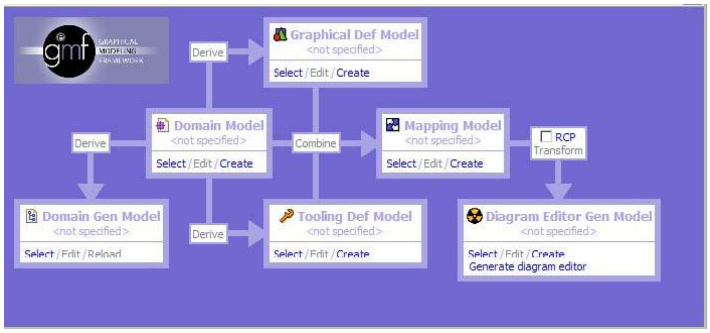


Figura AI.7. Componentes y Modelos en GMF

I.3.2. Pasos para Definir un Editor Gráfico

En esta sección se detallan los modelos necesarios para generar el código del editor gráfico. Cada uno de los modelos se define en un archivo separado, con formato **XMI**. **GMF** provee un editor para hacer más amigable cada una de estas definiciones.

I.3.2.1. Modelo de Dominio

Se debe especificar el metamodelo que se quiere instanciar usando el editor y generar el código necesario para manipularlo, usando **EMF**. No hay necesidad de crear el editor ni los casos de *test*.

I.3.2.2. Modelo de Definición Gráfica

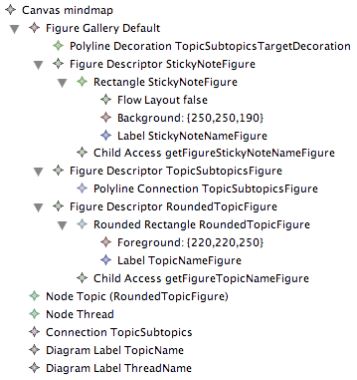


Figura AI.8. Editor del Modelo de Definición Gráfica

El modelo de definición gráfica se usa para definir figuras, nodos, *links*, compartimientos y demás elementos que se mostrarán en el diagrama. Este modelo está definido en un archivo con extensión "*.gmfgraph*". En la figura AI.8 puede verse el editor.

I.3.2.3. Definición de Herramientas

El modelo de definición de herramienta se usa para especificar la paleta, las herramientas de creación, las acciones que se desencadenan detrás de un elemento de la paleta, las imágenes de los botones, etc. En la Figura AI.9 puede verse el editor para la creación de este modelo.

Figura AI.9 Editor del Modelo de Definición de Herramientas

1.3.3. Definición de las Relaciones Entre los Elementos

El modelo de definición de relaciones entre elemento afecta a los tres modelos: el modelo de dominio, la definición gráfica y la definición de herramientas. Es un modelo clave para **GMF** y a partir de este modelo se obtiene el modelo generador, que es el que permite la generación de código. En la Figura AI.10 puede verse el editor para este modelo. En este modelo se especifica que elemento del metamodelo va a ser instanciado con que herramienta o con que entrada de la paleta, y que vista le corresponde en el editor.

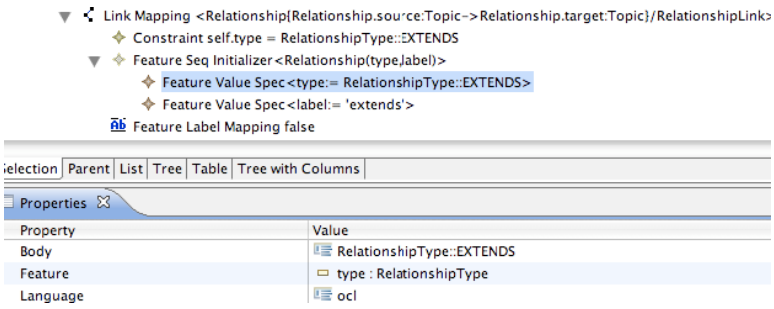


Figura AI.10. Editor del Modelo de Definición de Relaciones

I.3.4. Generación de Código Para el Editor Gráfico

Ya definidos los elementos gráficos y de mapeo, se puede comenzar a generar el código necesario. Para llevar a cabo la generación de código, primero es necesario crear el modelo generador para poder *setear* las propiedades propias de la generación de código. Este archivo es muy similar al archivo *genmodel* definido en **EMF**. Este paso se hace por medio de un menú contextual, sobre el archivo de mapeo, con la opción *'Create generator model...'* El modelo generador permite configurar los últimos detalles antes de la generación de código, como por ejemplo, las extensiones del diagrama y del dominio, que por defecto es el nombre del metamodelo del lenguaje, o cambiar el nombre del paquete para el *plugin* que se quiere generar. También permite configurar si el diagrama se guardará o no junto con la información del dominio en un único archivo.

I.3.5. Anatomía del Editor Gráfico

En la Figura AI.11 puede verse el editor gráfico cuyo código es completamente generado por **GMF**. A la izquierda se encuentra el explorador de paquetes, que muestra los recursos que se encuentran en cada proyecto. Por defecto, cada modelo creado por el editor está formado de dos archivos: un archivo conteniendo las instancias del metamodelo y otro conteniendo la información gráfica. Ambos archivos deben mantenerse sincronizados para tener un modelo consistente. Abajo a la izquierda se encuentra la vista *outline*, que es una vista con escala reducida que permite ver la totalidad del diagrama. En la parte inferior de la pantalla se ve la vista de propiedades, desde la cual se editan las propiedades de los elementos. La parte central de la pantalla muestra el editor, con su paleta de elementos. Como se dijo, este editor está integrado al entorno Eclipse, ya que muchas de las funcionalidades del editor están disponibles por medio de menús contextuales, y también desde la barra de herramientas de eclipse, como por ejemplo, el *zoom*, tipos de fuente, etc.

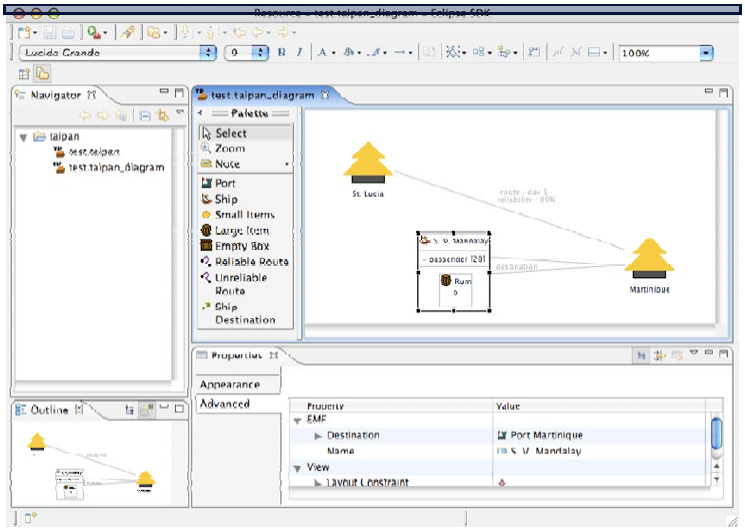


Figura AI.11. Anatomía del Editor Gráfico

I.4. Resumen del Anexo

El objetivo de este anexo fue describir, primeramente, las características principales del **EMF,** el *framework* de Eclipse para modelado, que permite la generación automática de código para construir herramientas y otras aplicaciones a partir de modelos de datos estructurados; luego detallar el **GMF,** un *framework* de código abierto, completamente integrados a Eclipse, que permite construir editores gráficos.

Anexo II

ATLAS Transformation Language (ATL)

II.1. Introducción

ATL es un lenguaje de programación hibrido (imperativo y declarativo); si bien el estilo declarativo es el recomendado, ya que permite simplificar los *mappings* entre el modelo fuente y el destino, **ATL** también provee construcciones imperativas para facilitar las especificaciones de *mappings* que son difíciles de especificar declarativamente. Una transformación en **ATL** está compuesta por reglas que definen cómo elementos del modelo fuente son relacionados y navegados para crear e inicializar los elementos del modelo destino.

Desarrollado sobre la plataforma Eclipse, provee un conjunto de herramientas de desarrollo estándar con el objetivo de facilitar las transformaciones **ATL.**

II.2. Visión General del ATL

El presente anexo estará dedicado a la descripción del lenguaje de transformación utilizado en el contexto del presente trabajo. Para una descripción completa recurrir al manual de referencia oficial de dicho lenguaje. http://www.eclipse.org/m2m/atl/

II.2.1. Modulo ATL

Un módulo **ATL** se corresponde a una transformación modelo a modelo. Los módulos permiten, al desarrollador, especificar la forma de producir un conjunto de modelos destinos a partir de un conjunto de modelos fuentes. Ambos modelos (fuente y destino) de un modulo **ATL** deben conformar con sus respectivos metamodelos.

II.2.1.1. Estructura de un Modulo ATL

Un módulo **ATL** define una transformación modelo a modelo (M2M) y está compuesto por los siguientes elementos:

- Una sección *header* que define algunos atributos vinculados al modulo de transformación.
- Una sección *import* (opcional) que permite importar librerías **ATL** existentes.
- Un conjunto de *helpers* que pueden ser entendidos como el equivalente **ATL** a los métodos java.
- Un conjunto de *rules* que definen la forma en que los modelos destino son generados a partir de los modelos fuente.

II.2.1.1.1. Sección *Header*

La sección *header* define el nombre del módulo de transformación y el nombre de las variables correspondientes a los modelos fuente y destino. A continuación se muestra la sintaxis de la sección *header*

```
module module_name;
create output_models [from|refines] input_models;
```

Luego de la palabra reservada `module`, el *module_name* describe el nombre del modulo. La declaración del modelo destino, *output_models*, se ubica luego de la palabra reservada `create`. La declaración del modelo fuente se detalla después la palabra reservada `from` (en modo normal) o *refines* (en el caso de una transformación de refinamiento). Ambos modelos (fuente y destino) deben conformar con sus metamodelos respectivos.

II.2.1.1.2. Sección *Import*

La opcional sección `import` permite declarar cuáles librerías **ATL** serán importadas. La declaración tiene la siguiente sintaxis:

```
uses extensionless_library_file_name;
```

Es posible declarar distintas librerías utilizando, sucesivamente, instrucciones `uses`.

II.2.2. Helpers

Un *helper* **ATL** puede ser visto como el equivalente **ATL** a los métodos Java. Permite definir código reutilizable que puede ser llamado en diferentes puntos en una transformación **ATL.** Un *helper* **ATL** se define mediante los siguientes elementos:

- Un nombre que corresponde al nombre del método.
- Un tipo contextual que define el contexto en el que el atributo está definido.
- Un tipo de valor de retorno. En **ATL**, cada *helper* debe tener un valor de retorno.
- Una expresión **ATL** que representa el código del *helper* **ATL.**
- Un conjunto opcional de parámetros identificados por el par (parámetro, tipo)

Por ejemplo, un *helper* con parámetros que retorna el valor máximo de dos enteros, se expresa de la siguiente manera:

```
helper context Integer def : max(x : Integer) : Integer = ...;
```

También es posible declarar un *helper* que no contenga parámetros. Por ejemplo:

```
helper context Integer def : double() : Integer = self * 2;
```

II.2.3. Rules

En **ATL**, existen dos tipos diferentes de reglas que corresponden a los dos modos diferentes de programación que provee **ATL** (declarativa e imperativa): los *matched rules* (la forma declarativa) y los *called rules* (la forma imperativa).

II.2.3.1. Matched Rules

Las *matched rules* constituyen el núcleo de una transformación declarativa de **ATL**, éstas permiten especificar 1) para cuáles tipos de elementos fuentes, los elementos destinos son generados y 2) la forma en que los elementos destino son inicializados.

Una matched *rules* comienza con la palabra reservada *rule*, que está compuesta por dos secciones obligatorias (los *patterns* origen y destino) y dos optativas (las variables locales y la sección imperativa).

El *pattern* fuente se define después de la palabra reservada *from*, permite especificar las variables de los elementos del modelo que corresponden a los tipos de los elementos fuente que la regla vincula.

El *pattern* destino se define después de la palabra reservada *to*, su objetivo es especificar los elementos que serán generados y cómo esos elementos generados son inicializados.

Cuando se definen variables locales, éstas se declaran después de la palabra reservada *using*. La sección imperativa opcional se detalla después de la palabra reservada *do*, ésta hace posible especificar el código imperativo que será ejecutado después de la inicialización de los elementos destinos generados mediante la *rule*. El siguiente código muestra un ejemplo de *matched rules:*

```
rule Author {
       from
              a : MMAuthor!Author
       to
              p : MMPerson!Person (
              name <- a.name,
              surname <- a.surname
              )
}
```

II.2.3.2. Called Rules

Las *called rules* permiten a los desarrolladores utilizar programación imperativa. Estas *rules* pueden ser vistas como un tipo de *helper*: tienen que ser explícitamente llamadas para ser ejecutadas y pueden aceptar parámetros. Sin embargo, a diferencia de los *helpers*, las *called rules* pueden generar elemento del modelo destino como las *marches rules*. Las *called rules* tiene que ser llamadas desde una sección de código imperativo, desde una *marches rules* o de otra *called rules*.

Al igual que las *matched rules*, las *called rules* se introducen después de la palabra reservada `rule` y también pueden tener una sección de variables locales. Sin embargo, como las *called rules* no tienen que vincular elementos del modelo fuente, no incluyen un *pattern* fuente. El *pattern* destino se introduce después de la palabra reservada `to`. El siguiente código muestra un ejemplo *called rules:*

```
rule NewPerson (na: String, s_na: String) {
       to
              p : MMPerson!Person (
              name <- na
              )
       do {
              p.surname <- s_na
       }
}
```

II.3. El lenguaje ATL

En esta sección se describirán los distintos tipos de datos, colecciones y operaciones y las características principales del lenguaje **ATL.**

II.3.1. Tipos de Datos

El esquema de los tipos de datos de **ATL** está basado en **OCL. ATL** considera seis principales clases de tipos de datos: los tipos de datos primitivos, los tipos de datos de colecciones, las tuplas, los mapeos, las enumeraciones y el tipo elemento de modelo (Figura AII.1).

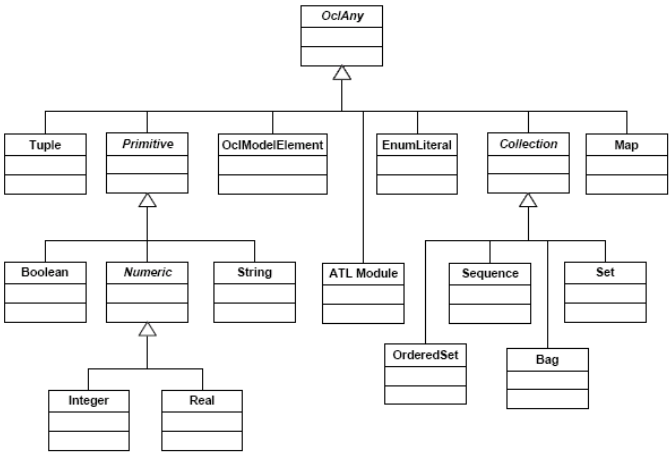


Figura AII. 1. Metamodelo de Tipos de Datos ATL

II.3.2. Operaciones Comunes a Todos los Tipos de Datos

Esta sección describe el conjunto de operaciones que son comunes a todos los tipos de datos existentes. La sintaxis usada para llamar una operación de una variable **ATL** es la clásica notación "punto": `self.operacion nombre(parámetros)`.

ATL provee, actualmente, soporte para las siguientes operaciones **OCL** predefinidas:

- `allInstances()` retorna todas las instancias existentes del tipo de datos del elemento sobre el cual se ejecuta dicha operación.
- `allInstancesFrom(metamodelo:String)` retorna todas las instancias existentes en el metamodelo identificado por el parámetro del tipo de datos del elemento sobre el cual se ejecuta dicha operación.
- operadores de comparación: `=, <>;`
- *`oclIsUndefined()`* retorna un valor booleano indicando cuando *`self`* está indefinido;
- *`oclIsKindOf(t : oclType)`* retorna un valor booleano indicando cuando *self* es una instancia del tipo t o de un subtipo de t;
- *`oclIsTypeOf(t : oclType)`* retorna un valor booleano indicando cuando *self* es una instancia del tipo t.

Las operaciones *`oclIsNew()`* y *`oclAsType()`* definidas en **OCL** aún no están soportadas en **ATL**. Sin embargo, **ATL** implementa las siguientes operaciones adicionales: *`toString()`*, *`oclType()`*, *`asSequence()`*, *`asSet()`*, *`asBag()`*, *`output(s : String)`*, *`debug(s : String)`*, *`refSetValue(name : String, val : oclAny)`*, *`refGetValue(name : String)`*, *`refImmediateComposite()`*.

II.3.3. El Tipo de Dato ATL Module

El tipo de dato **ATL** *Module* es específico del lenguaje. El mismo involucra a una unidad (modulo o consulta) y existe una única instancia de dicho tipo de datos, permitiendo de esta manera al programador acceder a la misma a través de la variable *`thisModule`*. A través de dicha variable es posible acceder a las funciones auxiliares (helpers) y a los atributos declarados en el contexto del módulo.

Este tipo de datos también provee la operación *`resolveTemp`*. Mediante dicha operación es posible apuntar, desde una regla, a cualquier elemento de modelo de salida (*target*) que será generado a partir de un dado elemento por una regla con *matching*.

II.3.3.1. Tipos de Datos Primitivos

OCL define cuatro básicos tipos de datos primitivos:

- *`Boolean`*, con los posibles valores *`true`* o *`false`*
- *`Integer`*, asociado a los valores numéricos (1, -5, 2, 34, 26524, ...);
- *`Real`*, asociado a los valores numéricos con punto flotante (1.5, 3.14, ...);
- *`String`*, representa a una cadena de caracteres.

Las operaciones sobre el tipo de datos booleano son las siguientes:

- Operadores lógicos: *`and`*, *`or`*, *`xor`*, *`not`*.
- *`implies(b : Boolean)`* retorna *false* si *`self`* es `true` y b es *`false`*, en cualquier otro caso retorna *`true`*.

Las operaciones sobre el tipo de datos *`String`* definidas en **OCL** son las siguientes:

- *size(), concat(s : String), substring(lower : Integer, upper : Integer), toInteger() y toReal().*

Las operaciones adicionales soportadas en **ATL** para el tipo de datos *String* son:

- operadores de comparación: <, >, =>, =<

- *concat(), toUpper(), toLower(), toSequence(), trim(), startsWith(s : String), endsWith(s : String), indexOf(s : String), lastIndexOf(s : String), split(regex : String), replaceAll(c1 : String, c2 : String), regexReplaceAll(regex : String, replacement : String).*

Y dos funciones que permiten escribir en archivos o en la consola, principalmente con propósito de ser útiles para depurar las transformaciones:

- writeTo(fileName : String) y println().

Las operaciones sobre el tipo de datos numérico definidas en **OCL** son los siguientes:

- operadores de comparación: <, >, =>, =<
- operadores binarios: *, +, -, /, *div(), max(), min(), mod()* sólo para *integer*, *floor()* y round() sólo para *real*
- operadores unarios: *abs().*

ATL provee, adicionalmente, las siguientes operaciones sobre los tipos de datos numéricos:

- *cos(), sin(), tan(), acos(), asin()*
- *toDegrees(), toRadians()*
- *exp(), log(), sqrt().*

II.3.4. Colecciones

Las clases de colecciones poseen las siguientes características:

- *Set* es una colección sin elementos duplicados y no tiene orden
- *OrderedSet* es una colección sin elementos duplicados y sus elementos se encuentran ordenados
- *Bag* es una colección donde se permiten los elementos duplicados y no tiene orden
- *Sequence* es una colección donde se permiten los elementos duplicados y sus elementos se encuentran ordenados.

II.3.4.1. Operaciones sobre Colecciones

Las siguientes operaciones definidas en **OCL** son compartidas por todas las clases de colecciones soportadas:

- *size(), includes(o:oclAny), excludes(o:oclAny), count(o: oclAny), includesAll(c:Collection), excludesAll(c: Collection), isEmpty(), notEmpty(), sum().*

Nota: La operación *product()* aún no se encuentra implementada en **ATL**.

Las siguientes operaciones se encuentran en la implementación actual de **ATL**:

- *asBag(), asSequence(), asSet().*

La clase *Sequence* soporta las siguientes operaciones específicas:

- *union(c : Collection), flatten(), append(o : oclAny), prepend(o : oclAny), insertAt(n : Integer, o : oclAny), subSequence(lower : Integer, upper : Integer), at(n : Integer), indexOf(o: oclAny), first(), last(), including(o : oclAny), excluding(o: oclAny).*

La clase *Set* soporta las siguientes operaciones específicas:

- *union(c:Collection), intersection(c:Collection), operator – (s : Set), including(o : oclAny), excluding(o : oclAny), symetricDifference(s : Set).*

La clase *OrderedSet* soporta las siguientes operaciones específicas:

- *append(o : oclAny), prepend(o : oclAny), insertAt(n : Integer, o : oclAny), subOrderedSet (lower : Integer, upper : Integer), at(n : Integer), indexOf(o : oclAny), first(), last(), union(c : Collection), flatten(), including(o : oclAny), excluding(o : oclAny).*

Nota: La clase *Bag* actualmente no cuenta con sus operaciones específicas implementadas.

II.3.4.2. Iteración Sobre Colecciones

Actualmente **ATL** soporta las siguientes expresiones de iteración:

- *exists(body), forAll(body), isUnique(body), any(body), one(body), collect(body), select(body), reject(body), sortedBy(body).*

II.3.5. Tipos de Datos Elementos de Modelo

En **ATL**, las variables correspondientes a elementos de modelos son referenciadas mediante *metamodel!class*, donde *class* apunta al *model element* (clase) del metamodelo referenciado. Cada elemento de modelo posee atributos y referencias a otros elementos de modelo. Es posible obtener una colección con dichos atributos y referencias mediante el atributo *self.feature*.

En **ATL**, la única manera de generar *model elements* es mediante la aplicación de reglas. La manera de inicializar un nuevo *model element* consiste en inicializar sus diferentes características (atributos y referencias). Tales asignaciones son realizadas mediante enlaces desde los elementos fuente de la regla de transformación.

II.3.6. Expresiones Declarativas

Existen dos tipos de expresiones declarativas: La expresión *if* provee alternativas, mientras que la expresión *let* permite definir e inicializar nuevas variables en el contexto de otra expresión.

Una expresión *if* se expresa mediante la estructura *if-then-else-endif*. La sintaxis es la siguiente:

```
If condition
        then
                exp1
        else
                exp2
endif
```

La *condition* debe ser una expresión booleana. De acuerdo a la evaluación de la expresión booleana if retornara el valor correspondiente a *exp1*, en el caso en que sea evaluada *true*, o *exp2* en el caso contrario.

La expresión **OCL** *let* permite la definición de variables. La sintaxis es la siguiente.

```
let var_name : var_type = var_init-exp in exp
```

El identificador *var_name* corresponde al nombre de la variable declarada, *var_type* identifica al tipo de la variable declarada. Una variable declarada mediante la expresión let debe inicializarse con var_init-exp. La inicialización de la expresión puede ser cualquiera de los tipos de expresiones **OCL** disponibles, incluyendo expresiones *let* anidadas. Por último, la palabra reservada *in* introduce la expresión en donde la nueva variable declarada puede ser usada.

II.3.7. ATL Helpers

ATL permite definir métodos en los distintos tipos de unidades (módulos, consultas y librerías), dando la posibilidad de modularizar el código de transformación. Existen dos tipos de *helpers*, los funcionales y los atributos. Ambos deben definirse en el contexto de un tipo de datos, sin embargo, los funcionales pueden ser parametrizados.

Un *helper* se define mediante el siguiente esquema:

```
helper [context context_type]? def :
helper_name(parameters) : return_type = exp;
```

El cuerpo del *helper* se corresponde a una expresión **OCL**, y puede ser de cualquiera de los tipos de expresiones soportadas. Un *helper* no funcional, posee el siguiente esquema:

```
helper [context context]? def :
attribute_name : return_type = exp;
```

En ambos casos, si no se especifica un contexto, el *helper* se asociará al módulo donde se encuentra definido.

II.3.8. Reglas de Transformación ATL

En el ámbito del leguaje de transformación **ATL**, la generación de elementos del modelo destino se realiza únicamente mediante reglas de transformación.

Existen dos tipos de reglas de transformación: *matched rules* y *called rules*. Una matched rules permite vincular algún elemento del modelo fuente y generar, a partir de éste, un conjunto de elementos del modelo destino. Un *called rules* tiene que ser invocado desde un bloque imperativo **ATL** para que sea ejecutado.

II.3.9. Código Imperativo ATL

Es posible escribir código imperativo en el contexto de las reglas de transformación. La implementación actual incluye tres tipos de sentencias: la asignación (<-), el condicional (*if*) y la iteración (*for*). Dado que ninguna de dichas sentencias tiene un valor de retorno, no pueden ser empleadas en el contexto de sentencias declarativas.

La instrucción de asignación permite asignar valores a los atributos que son definidos en el contexto de un modulo **ATL** o a elementos del modelo destino. La sintaxis es la siguiente:

```
target <- exp;
```

La expresión asignada (*exp*) puede ser cualquier cualquiera de las expresiones soportadas por **ATL.**

La instrucción *if* permite definir un tratamiento alternativo a las sentencias imperativas. La sintaxis es la siguiente:

```
if(condition) {
        statements1
}
[else {
        statements2
}]?
```

Cada sentencia *if* define una condición que debe ser una expresión **OCL** que debe devolver un valor booleano. La sentencia `if` puede también incluir una sección *then* que contiene la secuencia de sentencias que serán ejecutadas cuando la expresión condicional sea evaluada *true*. También incluye una sección opcional *else* que contiene la secuencia de sentencias que serán ejecutadas cuando la expresión condicional sea evaluada *false*.

La sentencia *for* permite definir código imperativo en forma iterativa. La sintaxis es la siguiente:

```
for(iterator in collection) {
        statements
}
```

La sentencia *for* define una variable de iteración (*iterator*) que iterará sobre los diferentes de la colección (*collection*) referenciada.

Actualmente, no es posible declarar variables en el contexto de bloques imperativos. Las variables que pueden ser empleadas en dichos bloques son:

- Elementos del modelo origen y destino declarados en la *matched rule* local
- Elementos del modelo destino declarados en la *matched rule* local
- Variables localmente declaradas en la regla
- Atributos declarados en el contexto del módulo **ATL**

II.3.10. Matched Rules

Mediante las *matched rules* se especifica la manera en que los elementos del modelo destino son generados a partir de los elementos del modelo origen. Con este propósito, una *matched rule* permite especificar: 1) los elementos del modelo original que serán relacionados, 2) el número y tipo de los elementos a generar, y 3) la forma que dichos elementos son inicializados a partir de los elementos fuentes vinculados.

La especificación de una *matched rule* debe respetar la siguiente sintaxis:

```
rule rule_name {
        from
                in_var : in_type [(
                        condition
                )]?
        [using {
                var1 : var_type1 = init_exp1;
                ...
                varn : var_typen = init_expn;
        }]?
        to
                out_var1 : out_type1 (
                        bindings1
                ),
                out_var2 : distinct out_type2 foreach(e in collection)(
                        bindings2
                ),
                ...
                out_varn : out_typen (
                        bindingsn
                )
        [do {
                statements
        }]?
}
```

Las reglas son identificadas por su nombre (*rule_name*). Una *matched rule* está compuesta por dos partes obligatorias (*from* y *to*) y dos partes opcionales (*using* y *do*).

- *from* se corresponde al elemento del modelo origen que es enlazado
- *using permite declarar variables* locales
- *to* se corresponde con los elementos a generar, puede ser simple o tener un patrón iterativo (para generar más de un elemento)
- *do permite especificar un bloque de código imperativo*.

II.3.11. Called Rules

Además de las *matched rules,* **ATL** define un tipo adicional de reglas que permiten generar elementos del modelo destino a partir de código imperativo. Excepto por las *entrypoint de las c*alled rules, éstas deben ser explícitamente invocadas desde bloques imperativos **ATL.**

La especificación de las *called rule* debe respectar la siguiente sintaxis:

```
[entrypoint]? rule rule_name(parameters){
        [using {
                var1 : var_type1 = init_exp1;
                ...
```

```
                var_n : var_type_n = init_exp_n;
        }]?
        [to
                out_var_1 : out_type_1 (
                        bindings_1
                ),
                out_var_2 : distinct out_type_2 foreach(e in collection)(
                        bindings_2
                ),
                ...
                out_var_n : out_type_n (
                        bindings_n
                )]?
        [do {
                statements
        }]?
}
```

Las reglas son identificadas por su nombre (*rule_name*). Una regla con *entrypoint* es invocada automáticamente al comienzo del proceso de transformación, una vez terminado el proceso de inicialización del módulo.

Una *called rule* puede aceptar parámetros, y se encuentran compuestas por tres secciones opcionales: `using`, `to` y `do` con la misma semántica que para las *matched rule*.

II.4. Resumen del Anexo

El objetivo de este anexo fue se detallar las construcciones más importantes del lenguaje de transformación **ATL**, lenguaje que permite, al desarrollador, especificar la forma de producir un conjunto de modelos destinos a partir de un conjunto de modelos fuentes. El lenguaje **ATL** es utilizado para describir las transformaciones.

Anexo III

MOFScript

III.1. Introducción

MOFScript es una herramienta para la transformación de modelo a texto (M2Text), esto es, asiste en el proceso de desarrollo de la generación, tanto de código de implementación como de la documentación, a partir de modelos. Posee pocas construcciones y es fácil de usar y comprender y tiene un estilo similar a los lenguajes de *script*. Es un lenguaje que está influido por **MOF QVT**, en particular, MOFScript especializa a **QVT.**

El presente anexo está dedicado a la descripción del lenguaje de transformación MOFScript utilizado en el contexto del presente trabajo. Para una descripción completa, recurrir al manual de referencia oficial de dicho lenguaje de donde se obtuvo esta información. http://www.eclipse.org/gmt/mofscript/

III.2. Características Principales de MOFScript

El lenguaje está construido en base a un conjunto de reglas que son llamadas explícitamente. Una transformación consiste en un conjunto de reglas/métodos. Las transformaciones y las reglas no pueden ser anidadas, pero si las estructuras de control (`if` y `loops`)

La herramienta MOFScript está basada en **EMF** y **ECORE.** MOFScript es un *plug-in* de Eclipse, pero también puede ejecutarse como una aplicación Java independiente.

Esta sección describe el lenguaje MOFScript y sus diversas construcciones.

III.2.1. Texttransformation

Un `Texttransformation` define el nombre del módulo. Puede ser elegido cualquier nombre, independiente del nombre de archivo. (Cualquiera de las palabras clave `texttransformation` o `textmodule` se puede utilizar.) Este define el metamodelo de entrada en función de un parámetro. Por ejemplo:

```
texttransformation testAnnotations (in
```

```
uml:"http://www.eclipse.org/uml2/1.0.0/UML"))
```

Un *texttransformation* puede tener varios parámetros de modelo de entrada. Estos deben ser separados por coma. Por ejemplo:

```
texttransformation TransformationWithSeveralMetaModels (in
uml:"http://www.eclipse.org/uml2/1.0.0/UML", in
ec:"http://www.eclipse.org/emf/2002/Ecore") { }
```

III.2.2. Importación

Un módulo de transformación puede importar otras transformaciones tales como las librarías. Por ejemplo:

```
import aSimpleName ("std/stdLib2.m2t")
import "std/stdLib2.m2t"
```

Las transformaciones importadas son analizas por el *parser*, buscando primero en el directorio actual, y luego en cualquier directorio especificado por la ruta de importación.

III.2.3. Reglas de Punto de Acceso

Las normas de punto de acceso definen dónde comienza la ejecución de la transformación. Es similar a un *main* de Java. Puede tener un contexto (en el ejemplo `uml.Model`), que define qué tipo de elemento del metamodelo será el punto de partida para la ejecución. Su cuerpo contiene las instrucciones. Por ejemplo:

```
uml.Model::main () {
        self.ownedMember->forEach(p:uml.Package)
        {
                p.mapPackage()
        }
}
```

Un punto de acceso puede tener un tipo contextual con varias instancias. En ese caso, el punto de acceso será ejecutado para cada instancia del tipo. Un ejemplo se muestra a continuación, donde el tipo contextual el punto de acceso es *uml.Class*. Por ejemplo:

```
uml.Class::main () {
        'class: ' self.name
}
```

Las reglas pueden ser especificadas sin un tipo contextual, así también las reglas de punto de acceso. Un punto de acceso sin ningún tipo contextual se ejecutará una vez. Éste se especifica mediante la palabra clave *module* o sin ninguna palabra clave. Para recuperar el modelo de entrada usando este enfoque, debe ser utilizado el parámetro de modelo para la transformación, junto con la operación *objectsOfType* para recuperar los objetos del modelo. Por ejemplo:

```
module::main () {
        uml.objectsOfType(uml.Package)
}
```

O

```
main () {
	uml.objectsOfType(uml.Package)
}
```

La operación *objectsOfType* puede aplicarse a cualquier elemento del modelo para recuperar un conjunto de objetos de un determinado tipo.

III.2.4. Reglas

Las reglas son, básicamente, similares a las funciones. Ellas pueden tener un tipo contextual, que es un tipo de metamodelo. También pueden tener un tipo de retorno, que puede ser un tipo integrado (*built-in)* o un tipo de modelo. El cuerpo de la regla contiene un conjunto de sentencias. Por ejemplo:

```
uml.Package::mapPackage () {
	self.ownedMember->forEach(c:uml.Class)
		c.mapClass()
}

uml.Class::mapClass(){
	file (package_dir + self.name + ext)
	self.classPackage()
	self.standardClassImport ()
	self.standardClassHeaderComment ()
	‘
	public class ‘ self.name ‘ extends Serializable { ‘
	self.classConstructor()
	‘
	/*
		* Attributes
		*/
	‘
	self.ownedAttribute->forEach(p : uml.Property) {
		p.classPrivateAttribute()
		}
	newline(2)
	‘}’
}
```

III.2.5. Valores de Retorno

Una regla también puede devolver un valor, éste puede ser reutilizado en las expresiones en otras reglas. Para devolver un valor, se utiliza la sentencia *result*. Por ejemplo:

```
uml.Package::getFullName (): String {
	if (self.owner != null)
		result = self.owner.getFullName() + "."
	else if (self.ownerPackage != null)
		result = self.ownerPackage.getFullName() + "."

	result += self.name.toLower().replace(" ", "_");
}
```

Un valor de retorno también puede ser dada por la sentencia *return*, que de inmediato finaliza la ejecución de la regla y regresa con el valor dado (si hubiere). Por ejemplo:

```
uml.Package::getFullName (): String {
	result self.owner.getFullName();
}
```

III.2.6. Parámetros

Una regla puede tener cualquier número de parámetros. Un parámetro puede ser un tipo integrado (*built-in)* o un tipo de metamodelo. Por ejemplo:

```
uml.Model::testParameters2 (s1:String, i1:Integer) {
       stdout.println("testParameters2: " + s1 + ", " + i1)
}

       uml.Model::testParameters3 (s1:String, r2:Real, b1:Boolean,
package:uml.Package) {
       stdout.println("Package:" + package.name)
       stdout.println ("testParameters3: " + s1 + ", " + r2 + ", "
       + b1 + " " + package.name)
       }
```

Una regla puede prescindir del tipo contextual. Esto se realiza usando la palabra reservada *module* u omitiendo el tipo contextual.

III.2.7. Propiedades y Variables

Las propiedades y las variables pueden ser definidas en forma global o local dentro de una regla o un bloque (por ejemplo el bloque iterador). Una propiedad es una constante que es asignada a un valor de la declaración. El tipo de una propiedad puede ser cualquiera de los tipos definidos, un tipo de modelo, o puede carecer de tipo, en este último caso el tipo será determinado por el valor asignado.

Una variable puede cambiar su valor durante el tiempo de ejecución de la asignación. Una variable puede ser del tipo de cualquiera de los definidos. También puede ser definido sin tipo en la declaración. Su tipo será determinado por el valor asignado. Si no se asigna ningún tipo, éste se convertirá en un *String*. Por ejemplo:

```
property packageName:String = "org.mypackage"
var myInteger = 7
```

III.2.8. Tipos Integrados

A continuación, se detallan los tipos integrados de MOFScript:

- *String*: representa valores de texto .
- *Integer*: representa a enteros;
- *Real*: representa a reales;
- *Boolean*: representa a boleanos;
- *Hashtable*: representa a una *Hashtable*;
- *List*: representa a una *List*:
- *Object*: representa a cualquier tipo

III.2.9. Archivos

La instrucción *file* declara el dispositivo de salida para el texto. Utiliza la palabra reservada *file*. Usa como parámetros el nombre de archivo y la extensión. También puede incluir la ruta absoluta o relativa de la salida. Por ejemplo:

```
file (c.name + ".java")
file ("c:\tmp\" + c.name + ".java")
file f2 ("test.java")
```

```
f2.println ("\t\t output to file f2")
```

Cuando un archivo se declara en una regla de transformación, éste será el destino de la salida previsto en las reglas invocadas, a menos que estas reglas declaren su propio archivo de salida. El archivo de referencia declarado, sin embargo, no es visible en las reglas invocadas.

III.2.10. Instrucciones de Impresión

Las instrucciones de impresión proveen el dispositivo de salida, éste puede ser un archivo o una salida estándar. Por ejemplo:

```
println ("public class" + c.name);
```

Si no se declara ningún archivo, se utiliza la salida estándar. Si la salida estándar debe ser forzada, la sentencia de impresión debe tener el prefijo *stdout*. Por ejemplo:

```
stdout.println ("public class" + c.name);
```

También están definidas funciones de impresión para gestionar los espacios en blanco más fácilmente: *newline* (o *nl*), *tab*, o *space*, seguido de un número entero opcional. También son legales, dentro de los literales de *String*, los caracteres de escape (\ n \ t). Por ejemplo:

```
print ("This is a standard print statement " + aVar.name)
newline (10)
tab(4) ' More escaped output \n\n '
println (" /** Documentation output */ ");
```

III.2.11. Salida de Escape

Los *Escaped Output* proporciona una manera diferente y, en algunos casos, más sencilla de dar salida a un dispositivo. Los *Escaped Output* trabajan igual que la mayoría de los lenguajes de *script*, tales como Java *script*, con caracteres de inicio y fin. Básicamente, es una declaración de impresión que puede subsumir múltiples líneas y se pueden combinar con todas las expresiones que se evalúan como *string*. Utiliza los caracteres ‘’ para empezar y terminar una escape. Todos los espacios en blanco se copian en el dispositivo de salida. Por ejemplo:

```
`
public class ` c.name ` extends Serializable {
`
```

También es posible utilizar, (por razones de compatibilidad histórica) los caracteres '<%' y '%>'. Por ejemplo:

```
<%
public class %> c.name <% extends Serializable {
%>
```

III.2.12. Iteradores

Los Iteradores se utilizan principalmente para iterar sobre colecciones de elementos del modelo de origen. La instrucción `forEach` define un iterador sobre una colección de elementos, tales como elementos del modelo, *list* o *hashtable*, o `integer`/`string`. Una instrucción `forEach` puede ser restringido por una restricción de tipo `(`**colección**`-> foreach (c: SomeType))`, donde el

tipo puede ser un tipo del metamodelo o un tipo integrado. Si no se utiliza una restricción de tipo, la instrucción se aplica a todos los elementos de la colección. Una instrucción `foreach` también puede tener una guarda (una restricción adicional) que, básicamente, puede ser cualquier de los tipos de expresión booleana. Una restricción es descrita mediante un símbolo de barra vertical ('|') (**colección**-> **foreach** (a: **String** | a.**size** () = 2). Por ejemplo:

```
-- se aplica a todos los objetos en la colección de tipo operación

c.ownedOperation->forEach(o:uml.Operation) {

-- instrucciones.
}

-- se aplica a todos los objetos en la colección de tipo operación
-- que tienen un nombre que empieza con 'a'

c.ownedOperation->forEach(o:uml.Operation |
o.name.startsWith("a")) {

/* instrucciones */

}
// se aplica a todos los objetos en la colección que
// con más de un parámetro y tipo de retorno

c.ownedOperation->forEach(o:uml.Operation |
o.ownedParameter.size() > 0 and o.returnResult.size() > 0) {

/* instrucciones */

}
```

III.2.12.1. Iteradores sobre Variables *List* y *Hashtable*

Los iteradores también pueden definirse para variables *list*/*hashtable*, como se ilustra a continuación:

```
var list1:List
list1.add("E1")
list1.add("E2")
list1.add(4)
list1->forEach(e){
       stdout.println (e)
}
```

III.2.12.2. Iteradores sobre String

Los Iteradores `String` definen bucles sobre el contenido de caracteres de un *String*. Por ejemplo:

```
var myVar: String = "Jon Oldevik"
myVar->forEach(c)
       stdout.print (c + " ")
```

III.2.12.3. Iteradores sobre *Integers*

Los Iteradores *integer* definen bucles basado en el tamaño del entero. Por ejemplo, `integer '3'` producirá un bucle que se ejecutará 3 veces.

```
property aNumber:Integer = 3
```

```
aNumber->forEach(n)
        stdout.print(" " + n)
```

III.2.12.4. Iteradores sobre *String* y Literales *Integer*

Los iteradores también pueden ser definidos usando *String* o literales *integer*. Estos funcionan de la misma manera que los iteradores basados en variables/operadores *String* y *integer*. Por ejemplo:

```
"MODELWare, the MDA(tm) project"-> forEach(s)
stdout.print ("" + s) 5->forEach(n)
stdout.println (" " + n)
```

III.2.13. Instrucciones Condicionales

Las instrucciones condicionales son las instrucciones estándar *if*. Pueden estar definidos con una simple rama `if` seguida de un conjunto de ramas *else-if*. Los argumentos de las ramas *else-if* son expresiones boolenas. Las instrucciones condicionales toman como argumento una expresión lógica. Por ejemplo:

```
if (c.hasStereoType ("entity")) {
        // instrucciones
} else if (c.hasStereoType("service")) {
        // instrucciones
} else {
        // instrucciones
}
if (c.ownedOperations.size() > 0 and c.name.startsWith("C")) {
        // instrucciones
} else {
        // instrucciones
}
```

III.2.14. Instrucciones `While`

La instrucción `while` trabaja de la misma manera que lo hace en el lenguaje java. La palabra reservada `while` es seguida por una restricción que puede ser cualquier tipo booleano. Por ejemplo:

```
var i : Integer = 10
while (i > 0){
// instrucciones
i -=1
}
```

III.2.15. Expresiones *Select*

Una expresión *select* realiza una consulta sobre una colección del modelo (o una variable colección) y devuelve una lista con el resultado de la consulta. La sintaxis de *select* es similar a la de *forEach*. Por ejemplo:

```
var xList:List = self.eClassifiers->select(c:ecore.EClass)
'Number of classes: ' xList.size()
xList->forEach(clazz:ecore.EClass) {
'\n \t Class: ' clazz.name
}

var yList:List = self.eClassifiers->select(c:ecore.EClass |
c.name.startsWith("MOF"))
```

III.2.16. Expresiones Lógicas

Las expresiones lógicas son expresiones que se evalúan como verdaderas o falsas y se utilizan en las instrucciones de iteración y condicionales. Por ejemplo:

```
self.ownedAttribute->forEach(p : uml.Property |
p.association != null){
// instrucciones
}

if (self.name = ”Car” or self.name = ”Person”) {
}
```

III.3. Resumen del Anexo

El objetivo de este anexo fue detallar las construcciones más importantes del lenguaje de transformación, modelo a texto, MOFScript, herramienta que asiste en el proceso de desarrollo de software, tanto en la generación de código de implementación, como de documentación, a partir de modelos.

Anexo IV

Transformaciones en ATL y MOFScript

IV.1. Introducción

En este anexo detallaremos todas las transformaciones que inicialmente fueron descritas de manera informal en el capítulo 5 (Diseño de un Data Warehouse Histórico) y en el capítulo 7 (Consultas en un Data Warehouse Histórico). Las transformaciones M2M, serán descritas en **ATL** y las transformaciones M2Text, en MOFScript.

IV.2. Transformaciones M2M Descritas en ATL

A continuación, se detallarán las transformaciones M2M, utilizaremos las transformaciones desarrolladas en el capítulo 5 y los metamodelos descritos en el capítulo 10.

IV.2.1. Transformación del Modelo de Datos al Modelo de Datos Temporal

```
module MD2MDT;

create mdt : MDT from md : MD;

       ------------------------------------------------
       -- retorna si la entidad está marcada como temporal
       ------------------------------------------------
       helper context MD!Entity def:
       isTemporalAnnotated(): Boolean = self.isTemp;

       ------------------------------------------------
       -- retorna si la entidad está marcada como hecho
       ------------------------------------------------
       helper context MD!Entity def:
       isFactAnnotated(): Boolean = self.isFact;

       ------------------------------------------------------------
       -- retorna si la entidad está marcada como temporal o hecho
```

```
	-------------------------------------------------------------
	helper context MD!Entity def:
	isAnnotated(): Boolean =
	self.isTemporalAnnotated() or self.isFactAnnotated();

	-------------------------------------------------
	-- retorna si el atributo está marcado como temporal
	-------------------------------------------------
	helper context MD!Attribute def:
	isTemporalAnnotated(): Boolean = self.isTemp;

	-------------------------------------------------------
	-- retorna si la interrelación está marcada como temporal
	-------------------------------------------------------
	helper context MD!Relationship def:
	isTemporalAnnotated(): Boolean = self.isTemp;

	-------------------------------------------------------
	-- retorna si la interrelación está marcada como hecho
	-------------------------------------------------------
	helper context MD!Relationship def:
	isFactAnnotated(): Boolean = self.isFact;

	-----------------------------------------------------------------
	-- retorna si la interrelación está marcada como temporal o hecho
	-----------------------------------------------------------------
	helper context MD!Relationship def:
	isAnnotated(): Boolean =
	self.isTemporalAnnotated() or self.isFactAnnotated();

	-----------------------------------------------------------------
	-- retorna todos los atributos que no están marcado como temporal
	-----------------------------------------------------------------
	helper context MD!Entity def:
	nonAnnotatedAttributes(): Boolean =
	self.attribute->select(attribute|not
	attribute.isTemporalAnnotated());


-------------------------------------------------------
-- la instancia de Schema es transformada a una instancia
-- de Schema en el modelo de datos temporal
-------------------------------------------------------
rule Schema2Schema {
	from
		schema : MD!Schema
	to
		schemaT : MDT!Schema (
		name <- schema.name,
		entity <- schema.entity
    )
}

-------------------------------------------------------
-- la Entity marcada como hecho se transforma FactEntity
-------------------------------------------------------
rule FactAnnotatedEntity2FactEntity {
	from
		entity : MD!Entity (entity.isFactAnnotated())
	to
		entityT : MDT!FactEntity (
		name <- entity.name,
		attribute <- entity.nonAnnotatedAttributes()->
			collect(a                                                    |
thisModule.NonAnnotatedAttribute2Attribute(a))                    )
}
```

```
-------------------------------------------------------------------
-- la Entity no marcada como hecho se transforma en Entity; si es
-- temporal, se crea una TemporalEntity y se la vincula con la Entity
-------------------------------------------------------------------
rule Entity2Entity {
	from
		entity : MD!Entity (not entity.isFactAnnotated())
	to
		entityT : MDT!Entity (
		name <- entity.name,
		attribute <- entity.nonAnnotatedAttributes()->
		collect(a | thisModule.NonAnnotatedAttribute2Attribute(a))
	)

	do {
		if entity.isTemporalAnnotated()
		then thisModule.createTemporalEntity(entity.name + '-T',

	Sequence{thisModule.CreateRelationshipEnd('n',entityT)},
			Sequence{})
		else false
		endif;
  }
}

-------------------------------------------------------------------
-- cuando es llamada, transforma un atributo del modelo de datos al
-- modelo de datos temporal sin modificaciones
-------------------------------------------------------------------
lazy rule NonAnnotatedAttribute2Attribute {
	from
		attribute : MD!Attribute
	to
		attributeT : MDT!Attribute (
		name <- attribute.name,
		isKey <- attribute.isKey,
		dataType <- attribute.dataType,
		entity <- attribute.entity
    )
}

-------------------------------------------------------------------
-- El Attribute marcado como temporal se transforma en
-- TemporalEntity y se lo vincula a la Entity de la cual proviene
-------------------------------------------------------------------
rule AnnnotatedAttribute2Attribute {
	from
		attribute : MD!Attribute (attribute.isTemporalAnnotated())
	to
		attributeT : MDT!Attribute (
		name <- attribute.name,
		isKey <- attribute.isKey,
		dataType <- attribute.dataType
    )

	do {
		thisModule.createTemporalEntity(attribute.name + '-T',

	Sequence{thisModule.CreateRelationshipEnd('n',attribute.entity)},
		Sequence{attributeT});
  }
}

-------------------------------------------------------------------
-- La Relationship no marcado como hecho se transforma en Relationship;
-- si es temporal, se crea una TemporalEntity y se la vincula a
-- las Entity que formaban parte de la Relationship, se mantiene la
```

```
-- Relationship original
-------------------------------------------------------------------------
rule NonFactRelationship2Relationship {
	from
		relationship : MD!Relationship (
		not relationship.isFactAnnotated())
	to
		relationshipT : MDT!Relationship(
		name <- relationship.name,
		schema <- relationship.schema,
		attribute <- relationship.attribute->

	collect(a|thisModule.NonAnnotatedAttribute2Attribute(a)),
			relationshipEnd <- relationship.relationshipEnd->

	collect(re|thisModule.RelationshipEnd2RelationshipEnd(re))
    )

	do {
		if (relationship.isTemporalAnnotated())
		then  thisModule.createTemporalEntity(relationship.name  +  '-
T',
		relationship.relationshipEnd->

	collect(re|thisModule.RelationshipEnd2RelationshipEnd(re)),
			relationship.relationshipEnd->
				collect(re | thisModule.CreateRelationshipEnd(
			'n', re.entity)),
			Sequence{})
		else false
		endif;
  }
}

-------------------------------------------------------------------
-- La Relationship marcada como hecho se transforma en FactEntity
-------------------------------------------------------------------
rule FactAnnotatedRelationship2Entity {
	from
		relationship : MD!Relationship
		(relationship.isFactAnnotated())
	to
		entity : MDT!FactEntity(
		name <- relationship.name,
		schema <- relationship.schema,
		attribute <- relationship.attribute ->
			collect(a                                                |
thisModule.NonAnnotatedAttribute2Attribute(a))
    )

	do {
		for (relationshipEnd in relationship.relationshipEnd) {
		thisModule.CreateRelationship(relationship.name + '-' +
		relationshipEnd.entity.name,
		Sequence{thisModule.CreateRelationshipEnd('1', entity),
		thisModule.CreateRelationshipEnd('n',
		relationshipEnd.entity)});
	 }
  }
}

-------------------------------------------------------
-- copia las instancias de RelationshipEnd al modelo de
-- datos temporal sin realizar cambios
-------------------------------------------------------
lazy rule RelationshipEnd2RelationshipEnd {
	from
```

```
                relationshipEnd : MD!RelationshipEnd
        to
                relationshipEndT : MDT!RelationshipEnd (
                entity <- relationshipEnd.entity,
                multiplicityMax <- relationshipEnd.multiplicityMax
    )
}

-------------------------------------------------------------
-- cuando es llamada, crea una TemporalEntity con sus propiedades
-------------------------------------------------------------
rule createTemporalEntity(temporalEntityName : String,
                relationshipEnds : Sequence(MDT!RelationshipEnd),
                attributes : Sequence(MD!Attribute)) {
       to
                temporalEntity : MDT!TemporalEntity (
                name <- temporalEntityName,
                schema <- relationshipEnds.first().entity.schema,
                attribute <- attributes,
                initialTime <- initialTimeAttribute,
                finalTime <- finalTimeAttribute
    ),
                initialTimeAttribute : MDT!DateAttribute (
                name <- 'TI',
                isKey <- true,
                dataType <- 'Date',
                entity <- temporalEntity
       ),
                finalTimeAttribute : MDT!DateAttribute (
                name <- 'TF',
                isKey <- false,
                dataType <- 'Date',
                entity <- temporalEntity
       )

       do {
                for (relationshipEnd in relationshipEnds) {
                thisModule.CreateRelationship('T',

                Sequence{thisModule.CreateRelationshipEnd('1',temporalEntity
                ),
                thisModule.CreateRelationshipEnd('n',relationshipEnd.entity)
                });
                if (relationshipEnd.multiplicityMax = '1')
                then thisModule.CreateRelationship('T',
                       Sequence{thisModule.CreateRelationshipEnd(
                       'n',temporalEntity), relationshipEnd})
                else thisModule.CreateRelationship('T',
                       Sequence{thisModule.CreateRelationshipEnd(
                       '1', temporalEntity),
                       relationshipEnd})
                endif;
        }
  }
}

-----------------------------------------------------
-- cuando es llamada, crea y retorna una Relationship
-----------------------------------------------------
rule CreateRelationship(relName : String,
                relationshipEnds : Sequence(MDT!RelationshipEnd)) {
       to
                relationship : MDT!Relationship (
                name <- relName,
                       relationshipEnd <- relationshipEnds,
                schema <- relationshipEnds.first().entity.schema
       )
```

```
        do {
                relationship;
    }
}

-------------------------------------------------------
-- cuando es llamada, crea y retorna una RelationshipEnd
-------------------------------------------------------
rule CreateRelationshipEnd(multiplicity:String,
                refEntity:MDT!Entity) {
        to
                relationshipEnd : MDT!RelationshipEnd (
                multiplicityMax <- multiplicity,
                entity <- refEntity
        )
        do {
                relationshipEnd;
    }
}
```

IV.2.2. Transformación del Modelo de Datos Temporal al Grafo de Atributos

```
module MDT2MGA;
create mga : MGA from mdt : MDT;

        ---------------------------------------------
        -- retorna los atributos marcados como clave
        ---------------------------------------------
        helper context MDT!Entity def:
        keyAttributes(): Sequence(MDT!Attribute) =
        self.attribute -> select(a | a.isKey);

        ------------------------------------------------
        -- retorna los atributos no marcados como clave
        ------------------------------------------------
        helper context MDT!Entity def:
        nonKeyAttributes(): Sequence(MDT!Attribute) =
        self.attribute -> select(a | not a.isKey);

-------------------------------------------------------------------
-- copia la instancia del Schema, estableciendo como root el hecho
-------------------------------------------------------------------
rule Schema2Schema {
         from
                schema : MDT!Schema
         to
                schemaT : MGA!Schema (
                name <- schema.name,
                root <- MDT!FactEntity.allInstances().first()
     )
}

-------------------------------------------------------------------
-- las entidades se transforman en nodos (Node), se establecen las
-- conexiones con los identificadores, hojas y nodos relacionados
-------------------------------------------------------------------
rule Entity2Node {
        from
                entity : MDT!Entity
        using {
                relEntities : Sequence(MDT!Entity) =
                MDT!RelationshipEnd.allInstances()->
                select(re | re.entity = entity and re.multiplicityMax='1') -
>               collect(re | re.relationship)->
                collect(r | r.relationshipEnd)-> flatten() ->
                select(re |re.entity <> entity and re.multiplicityMax='n')->
```

```
                collect(re | re.entity);
                parentEntity:MDT!Entity =
                if entity.oclIsTypeOf(MDT!TemporalEntity)
                then relEntities -> first()
                else OclUndefined
                endif;
  }
        to
                node : MGA!Node(
                label <- entity.name,
                schema <- MGA!Schema.allInstances().first(),
                isTemporal <- entity.oclIsTypeOf(MDT!TemporalEntity),
                identifier <- entity.keyAttributes()->
                        collect(a | thisModule.Attribute2Identifier(a)),
                leaf <- entity.nonKeyAttributes()->
                        collect(a|thisModule.Attribute2Leaf(a)),
                node <- relEntities ->
                        select(e | not e.oclIsTypeOf(MDT!TemporalEntity)),
                parent <- parentEntity
    )
}

---------------------------------
-- Transforma el atributo en hoja
---------------------------------
lazy rule Attribute2Leaf {
        from
                attribute : MDT!Attribute
        to
                leaf : MGA!Leaf (
                        name <- attribute.name,
                        dataType <- attribute.dataType
          )
}
-----------------------------------------
-- Transforma el atributo en identificador
-----------------------------------------
lazy rule Attribute2Identifier {
        from
                attribute : MDT!Attribute
        to
                identifier : MGA!Identifier (
                name <- attribute.name,
                dataType <- attribute.dataType
          )
}
```

IV.2.3. Transformación del Grafo de Atributos al Modelo MD Temporal

```
module MGA2MMDT;
create mmdt : MMDT from mga : MGA;

        ------------------------------------------------------------
        -- Retorna la instancia del esquema del grafo de atributos
        ------------------------------------------------------------
        helper def:
        mgaSchema(): MGA!Schema = MGA!Schema.allInstances()-> first();

        ------------------------------------------------------------
        -- Dado un nodo, retorna el conjunto de nodos que vinculados
        ------------------------------------------------------------
        helper context MGA!Node def:
        referencedByNodes() : Sequence(MGA!Node) =
        MGA!Node.allInstances()-> select(n | n <> self and
        n.node -> includes(self));
```

```
------------------------------------------------
-- copia la instancia del Schema, manteniendo una
-- referencia directa al hecho
------------------------------------------------
rule Schema2Schema {
	from
		schema : MGA!Schema
	to
		schemaT : MMDT!Schema (
		name <- schema.name,
		fact <- schema.root
    )
}

-----------------------------------------
-- la root del grafo se transforma en el fact
-----------------------------------------
rule Root2Fact {
	from
		root : MGA!Node (root = thisModule.mgaSchema().root)
	using {
		tmpDim : Sequence(MMDT!Dimension) =
		root.leaf -> select(l| l.isTemporal)->
		collect(l| thisModule.Leaf2Dimension(l));
		tmpDimIDS : Sequence(MMDT!Identifier) =
		tmpDim -> collect(td| td.identifier)-> flatten();
  }
	to
		fact : MMDT!Fact (
		name <- root.label,
		identifier <- root.identifier->
			collect(id | thisModule.Identifier2Identifier(id)),

		ref <- root.node -> collect(n| n.identifier)-> flatten()->
			collect(id | thisModule.Identifier2Reference(id))
			-> union(tmpDimIDS->
			collect(id | thisModule.MMIdentifier2Reference(id))),
		measure <- root.leaf->
			select(l | not l.pruned and not l.isTemporal)->
			collect(l | thisModule.Leaf2Measure(l)),
				dim <- root.node-> union(tmpDim)
    )
}

-------------------------------------------------
-- Se crea una referencia al elemento relacionado a
-- partir de su identificador
-------------------------------------------------
lazy rule MMIdentifier2Reference {
	from
		id : MMDT!Identifier
	to
		r: MMDT!Reference (
		name <- id.name,
		dataType <- id.dataType,
		multit <- id.mmelem
	)
}


-----------------------------------------
-- Se crea una referencia al nodo relacionado
-- a partir de su identificador
-----------------------------------------
lazy rule Identifier2Reference {
	from
		id : MGA!Identifier
```

```
	to
		r: MMDT!Reference (
		name <- id.name,
		dataType <- id.dataType,
		multit <- id.node
	)
}


-------------------------------------------------
-- Copia los identificadores sin realizar cambios
-------------------------------------------------
lazy rule Identifier2Identifier {
	from
		id : MGA!Identifier
	to
		id1 : MMDT!Identifier (
		name <- id.name,
		dataType <- id.dataType
	)
}

-----------------------------------------------------------
-- Las hojas relacionadas al hecho se transforman en medidas
-----------------------------------------------------------
lazy rule Leaf2Measure {
	from
		l : MGA!Leaf
	to
		m : MMDT!Measure (
		name <- l.name,
		dataType <- l.dataType
	)
}

-------------------------------------------------------------
-- Las hojas temporales del hecho se transforman en dimensión
-------------------------------------------------------------
lazy rule Leaf2Dimension {
	from
		l : MGA!Leaf
	to
		m : MMDT!Dimension (
		name <- l.name,
		identifier <- Sequence{id} ),
			id: MMDT!Identifier (
		name <- l.name + 'ID',
		dataType <- l.dataType
	)
}


---------------------------------
-- Transforma un nodo en dimensión
---------------------------------
rule Node2Dimension {
	from
		node : MGA!Node (
		thisModule.mgaSchema().root.node->includes(node))
	to
		dim : MMDT!Dimension (
		name <- node.label,
		identifier <- node.identifier ->
			collect(id | thisModule.Identifier2Identifier(id)),

		ref <- node.node -> collect(n | n.identifier ->
			collect(id | thisModule.Identifier2Reference(id)))->
```

```
                flatten(),
            attribute <- node.leaf ->
                collect(l | thisModule.Leaf2Attribute(l)),
            hier <- node.node -> union(node.temporalNode)
    )
}


---------------------------------
-- Transforma una hoja en atributo
---------------------------------
lazy rule Leaf2Attribute {
      from
            leaf : MGA!Leaf
       to
            attribute : MMDT!Attribute (
            name <- leaf.name,
            dataType <- leaf.dataType
    )
}


-----------------------------------------------
-- Transforma un nodo en una jerarquía estricta
-----------------------------------------------
rule Node2StrictHierarchy {
      from
            node : MGA!Node (not node.isTemporal and
            not thisModule.mgaSchema().root.node ->
            includes(node) and node <> thisModule.mgaSchema().root)
      to
            tmpHier : MMDT!StrictHierarchy (
            name <- node.label,
            identifier <- node.identifier ->
                collect(id | thisModule.Identifier2Identifier(id)),

            ref <- node.node -> collect(n | n.identifier->
                collect(id | thisModule.Identifier2Reference(id)))->
                flatten(),
            attribute <- node.leaf->collect(l|
            thisModule.Leaf2Attribute(l)),
            strictH <- node.node->excluding(node.parent)->
                select(n  |n.referencedByNodes()->
                size() = 1)-> union(node.temporalNode)
    )
}

-----------------------------------------------
-- Transforma un nodo en una jerarquía temporal
-----------------------------------------------
rule Node2TempHierarchy {
      from
            node : MGA!Node (node.isTemporal and not
            thisModule.mgaSchema().root.node ->
            includes(node)  and node <> thisModule.mgaSchema().root)
      to
            tmpHier : MMDT!TempHierarchy (
            name <- node.label,
            identifier <- node.identifier ->
                collect(id | thisModule.Identifier2Identifier(id)),
            ref <- node.node -> collect(n | n.identifier ->
                collect(id | thisModule.Identifier2Reference(id)))->
                flatten(),
            attribute <- node.leaf -> collect(l|
                thisModule.Leaf2Attribute(l)),
            strictH  <- node.node->excluding(node.parent)->
                select(n | n.referencedByNodes()-> size() = 1)->
```

```
                    union(node.temporalNode)
    )
}
```

IV.2.4. Transformación del Modelo MD Temporal al Modelo Relacional

```
module MMDT2MR;
create mr : MR from mmdt : MMDT;

       ----------------------------------------
       -- Retorna la única instancia de Schema
       ----------------------------------------
       helper def:
       schema(): MMDT!Schema = MMDT!Schema.allInstances()-> first();

----------------------------------------
-- Copia el esquema sin realizar cambios
----------------------------------------
rule Schema2Schema {
       from
              schema : MMDT!Schema
       to
              schemaR : MR!Schema (
              name <- schema.name
    )
}

-------------------------------------
-- El hecho se transforma en una tabla
-------------------------------------
rule Fact2Table {
       from
              f : MMDT!Fact
       to
              t: MR!Table (
              name <- f.name,
              schema <- thisModule.schema()
       )
}

-------------------------------------------------
-- Las medidas del hecho se transforman en columnas
-- asociadas a la tabla del hecho
-------------------------------------------------
rule Measure2Column {
       from
              m : MMDT!Measure
       to
              c : MR!Column (
              name <- m.name,
              dataType <- m.dataType,
              table <- m.fact
    )
}

-----------------------------------------
-- Las dimensiones son transformadas en tablas
-----------------------------------------
rule Dimension2Table {
       from
              d: MMDT!Dimension
       to
              t: MR!Table (
              name <- d.name,
              schema <- thisModule.schema()
    )
```

```
}

-------------------------------------------------
-- Las jerarquías estrictas se transforman en tablas
-------------------------------------------------
rule StrictHierarchy2Table {
	from
		th: MMDT!StrictHierarchy
	to
		t: MR!Table (
		name <- th.name,
		schema <- thisModule.schema()
    )
}

-------------------------------------------------
-- Las jerarquías temporales se transforman en tablas
-------------------------------------------------
rule TempHierarchy2Table {
	from
		th: MMDT!TempHierarchy
	to
		t: MR!Table (
		name <- th.name,
		schema <- thisModule.schema()
    )
}

-----------------------------------------------
-- Los identificadores se transforman en claves
-----------------------------------------------
rule Identifier2Key {
	from
		id : MMDT!Identifier
	to
		key : MR!Key (
		name <- id.name,
		dataType <- id.dataType,
		table <- id.mmelem
    )
}

-----------------------------------------------------------
-- Los atributos que no sean identificadores, medidas ni
-- referencias se transforman en columnas
-----------------------------------------------------------
rule Attribute2Column {
	from
		at : MMDT!Attribute (not at.oclIsTypeOf(MMDT!Identifier)
		and not at.oclIsTypeOf(MMDT!Measure)
		and not at.oclIsTypeOf(MMDT!Reference))
	to
		col : MR!Column (
		name <- at.name,
		dataType <- at.dataType,
		table <- at.multit
    )
}

-------------------------------------------------
-- Las referencias se transforman en claves externas
-------------------------------------------------
rule Reference2ForeignKey {
	from
		r : MMDT!Reference
	to
		fkey : MR!ForeignKey (
```

```
            name <- r.name,
            dataType <- r.dataType,
            node <- r.multit,
            schema <- thisModule.schema(),
            parentFK <- r.mmelem
    )
}
```

IV.2.5. Transformación del Modelo MD Temporal al Grafo de Consultas

```
module MMDT2QG;
create qg : QG from mmdt : MMDT;

-------------------------------------------------------------
- Se crea el elemento root del grafo de consultas a
- partir del element root del modelo multidimensional
-------------------------------------------------------------

helper def: schema(): MMDT!Schema = MMDT!Schema.allInstances()->first();

rule Schema2Schema {
  from
    schema : MMDT!Schema
  to
    schemaQ : QG!Schema (
      name <- schema.name
    )
}

-------------------------------------------------------------------------
- El hecho principal también tiene representación distinguida
- y es creado a partir del hecho principal del modelo multidimensional
-------------------------------------------------------------------------

rule Fact2Fact {
  from
       f: MMDT!Fact
  to
       fq: QG!Fact(
             name <- 'H-' + f.name,
             schema <- thisModule.schema(),
             vertex <- f.measure->union(f.dim),
             attribute <- f.attribute,
             key <- f.identifier,
             foreignKey <- f.ref
       )
}
-------------------------------------------------------------
- Las medidas del modelo multidimensional son medidas
- también en el grafo de consultas
-------------------------------------------------------------

rule Measure2Measure {
  from
    m : MMDT!Measure
  to
    mq : QG!Measure (
       name <- 'M-' + m.name,
             dataType <- m.dataType,
             schema <- thisModule.schema()
    )
}
-------------------------------------------------------------
- Las dimensiones se mapean a niveles del grafo de consultas
-------------------------------------------------------------
```

```
rule Dimension2Level {
  from
    d: MMDT!Dimension
  to
    l: QG!Level (
       name <- 'N-' + d.name,
              schema <- thisModule.schema(),
              vertex <- d.hier,
              attribute <- d.attribute,
              key <- d.identifier,
              foreignKey <- d.ref
    )
}
-----------------------------------------------------
- Las jerarquías estrictas se mapean a niveles del grafo
- de consultas
-----------------------------------------------------

rule StrictHierarchy2Level {
  from
    th: MMDT!StrictHierarchy
  to
    l: QG!Level (
       name <- 'N-' + th.name,
              schema <- thisModule.schema(),
              attribute <- th.attribute,
              key <- th.identifier,
              foreignKey <- th.ref
    )
}
-------------------------------------------------------------
- Se crea una entidad temporal para cada jerarquía temporal
- con una referencia y atributo
-------------------------------------------------------------

rule TempHierarchy2TempEnt {
  from
    th: MMDT!TempHierarchy (th.ref->size()=1 and th.attribute->size()=1)
  to
    t: QG!TempEnt (
       name <- 'TE-' + th.name,
              schema <- thisModule.schema(),
              attribute <- th.attribute,
              key <- th.identifier,
              foreignKey <- th.ref
    )
}
-------------------------------------------------------------
- Se crea un atributo temporal para cada jerarquía temporal
- con dos atributos y una referencia
-------------------------------------------------------------

rule TempHierarchy2TempAtt {
  from
    th: MMDT!TempHierarchy (th.ref->size()=1 and th.attribute->size()=2)
  to
    t: QG!TempAtt (
       name <- 'TA-' + th.name,
              schema <- thisModule.schema(),
              attribute <- th.attribute,
              key <- th.identifier,
              foreignKey <- th.ref
    )
}
-------------------------------------------------------------
- Se crea una relación temporal para cada jerarquía temporal
```

```
- con más de una referencia
-------------------------------------------------------------

rule TempHierarchy2TempRel {
  from
    th: MMDT!TempHierarchy (th.ref->size()>1)
  to
    t: QG!TempRel (
       name <- 'TR-' + th.name,
              schema <- thisModule.schema(),
              attribute <- th.attribute,
              key <- th.identifier,
              foreignKey <- th.ref
    )
}
-------------------------------------------------------------------
- Los identificadores se mapean a claves en el grafo de consultas
-------------------------------------------------------------------

rule Identifier2Key {
  from
    id : MMDT!Identifier
  to
    key : QG!Key (
       name <- id.name,
              dataType <- id.dataType
    )
}
-----------------------------------------------------------------
- Por cada atributo que no sea ni identificador, ni medida
- ni referencia, se crea un atributo en el grafo de consultas
-----------------------------------------------------------------

rule Attribute2Attribute {
  from
    at : MMDT!Attribute (not at.oclIsTypeOf(MMDT!Identifier) and
                                  not at.oclIsTypeOf(MMDT!Measure) and
                                         not
at.oclIsTypeOf(MMDT!Reference))
  to
    atq : QG!Attribute (
       name <- at.name,
              dataType <- at.dataType
    )
}
-----------------------------------------------------------
- Las referencias se mapean a claves externas en el grafo
- de consultas
-----------------------------------------------------------

rule Reference2ForeignKey {
  from
    r : MMDT!Reference
  to
    fkey : QG!ForeignKey (
       name <- r.name,
              dataType <- r.dataType,
              node <- r.multit,
              schema <- thisModule.schema(),
              parentFK <- r.mmelem
    )
}
```

IV.3. Transformaciones M2Text Descritas en MOFScript

A continuación se detallarán las transformaciones M2Text, utilizaremos las transformaciones desarrolladas en e capítulo 7 y los metamodelos descritos en el capítulo 10.

IV.3.1. Transformación del Modelo Relacional a Sentencias SQL

```
texttransformation MR2SQL (in mr:"mr") {

-----------------------------------------------------------------------
-- Se crea el archivo con  extensión  sql con el mismo nombre que
-- el esquema.  Por cada tabla del esquema se generan las sentencias
-- para la creación de la misma, con sus claves y columnas
-----------------------------------------------------------------------

  mr.Schema::main () {
    file('/models/MR_'+self.name + '.sql')

    self.table->forEach(t:mr.Table) {
      'CREATE TABLE ' t.name '(\n'
      t.column->forEach(c:mr.Column) {
        '\t'c.mapColumn()',\n'
      }
      t.key->forEach(k:mr.Key) {
        '\t'k.mapKey()',\n'
      }
      '\t''PRIMARY KEY ('
        t.key->forEach(k:mr.Key) {
        k.name
        if (t.key.last()!=k) {','}
      }
      ')'
      if (not t.foreignKey.isEmpty()) {','}
      '\n'
      t.foreignKey->forEach(fk:mr.ForeignKey) {
        '\t'
        fk.mapForeignKey()
        if (t.foreignKey.last()!=fk) {', '}
        '\n'
      }
      ')\n'
    }
  }

-----------------------------------------------------
-- Retorna la sentencia que declara la clave externa
-----------------------------------------------------
  mr.ForeignKey::mapForeignKey() {
      'FOREIGN KEY ('self.name') REFERENCES ' self.ref.name
  }

---------------------------------------
-- Retorna el nombre y tipo de la clave
---------------------------------------
  mr.Key::mapKey() {
      self.name'  'self.dataType
  }

-----------------------------------------
-- Retorna el nombre y tipo de la columna
-----------------------------------------
  mr.Column::mapColumn() {
      self.name'  'self.dataType
```

```
  }

}
```

IV.3.2. Transformación del Grafo de Consultas a Sentencias SQL

```
texttransformation QG2SQL (in qg:"qg") {

-----------------------------------------------------------------
-- Se crea el archivo con  extensión  sql con el mismo nombre que
-- el esquema antecedido por "QG_".
-- Si el hecho principal no está marcado la consulta es temporal
-----------------------------------------------------------------

  qg.Schema::main () {
    file('/models/QG_'+self.name + '.sql')

    property fact:qg.Fact = qg.objectsOfType(qg.Fact).first()
    if (fact.selected >=1 ) {
      fact.mapAsNonTemporalQuery()
    }  else {
       qg.objectsOfType(qg.Temporal)->forEach(vt:qg.Temporal |
       vt.selected>=1){
         vt.mapAsTemporalQuery()
       }
    }
  }

-------------------------------------------------------------------
-- La consulta no temporal es generada a partir del hecho principal
-- Se genera la Consulta Sobre Distintos Niveles de Agrupamiento.
-- Se tiene en cuenta el orden establecido sobre cada nodo del grafo
-------------------------------------------------------------------

  qg.Fact::mapAsNonTemporalQuery() {

    property fact:qg.Fact = qg.objectsOfType(qg.Fact).first()
    var c : Integer = 0
    var order : Hashtable
    var orderedLevels : List

    qg.objectsOfType(qg.Level)->forEach(l:qg.Level | l.selected >=1 ) {
      order.put(l.selected,l)
    }

    order.keys().sort()->forEach(k:Integer) {
      orderedLevels.add(order.get(k))
    }

    '--Consulta Sobre Distintos  Niveles de Agrupamiento\n'
    'select '
    orderedLevels->forEach(l:qg.Level) {
      l.name.subStringAfter('-') '.' l.key.first().name ','
    }
    self.vertex->forEach(m:qg.Measure | m.selected >=1) {
    self.function '(' self.name.subStringAfter('-') '.'
m.name.subStringAfter('-') ')'
    }
    '\n'
    'from '
    fact.name.subStringAfter('-')
    qg.objectsOfType(qg.Level)->forEach(l:qg.Level | l.selected>=1 or
l.isSelected()) {
      ', ' l.name.subStringAfter('-')
    }
    '\n'
```

```
    'where '
    fact.foreignKey->forEach(f:qg.ForeignKey | f.node.selected>=1 or
f.node.isSelected()) {
      if (c>0) {'and '} fact.name.substringAfter('-')'.'f.name '='
f.node.name.substringAfter('-')'.'f.name '\n'
      c=c+1
    }
    qg.objectsOfType(qg.Level)->forEach(l:qg.Level | l.isSelected()) {
      'and ' l.chain()
    }
    '\n'
    'group by '
    c = 0
    orderedLevels->forEach(l:qg.Level) {
      if (c>0) {', '} l.name.subStringAfter('-') '.' l.key.first().name
      c=c+1
    }
    '\n'
    'order by '
    c = 0
    orderedLevels->forEach(l:qg.Level) {
      if (c>0) {', '} l.name.subStringAfter('-') '.' l.key.first().name
      c=c+1
    }
  }

------------------------------------------------------------
-- Retorna los nodos seleccionados que estén directa o
-- indirectamente relacionados, junto con los nodos intermedios
------------------------------------------------------------
  qg.Node::chain():String {
    var ch : String = ''

    self.foreignKey->forEach(f:qg.ForeignKey) {
        if (f.node.selected >=1 ) {
          ch = '' self.name.substringAfter('-') '.'
self.foreignKey.first().name '=' f.node.name.substringAfter('-')'.'
f.name
        } else {
          if (f.node.isSelected()) {
                  ch = '' self.name.substringAfter('-
      ')'.'self.foreignKey.first().name '='
      f.node.name.substringAfter('-')'.' f.name  ' and ' f.node.chain()
          }
        }
      }
      return ch;
  }

------------------------------------------------------
-- Retorna si el nodo está marcado o uno de los nodos
-- relacionados está marcado
------------------------------------------------------
  qg.Node::isSelected():Boolean {
      self.foreignKey->forEach(f:qg.ForeignKey) {
        if (f.node.selected>=1 or f.node.isSelected()) {
          return true
        }
      }
      return false
  }

------------------------------------------------------
-- Si las consultas son temporales, genera la consulta
-- correspondiente sobre la entidad temporal
------------------------------------------------------
  qg.TempEnt::mapAsTemporalQuery() {
```

```
     property p : qg.ForeignKey = self.foreignKey.first()
     property entity : qg.Node = p.node
        '--Consulta Sobre  una Entidad Temporal\n'
        'parameters p ' p.dataType ';\n'
     'select ' entity.name.subStringAfter('-') '.' p.name ', '
               self.name.subStringAfter('-')'.TI, '
               self.name.subStringAfter('-')'.TF' '\n'
     'from ' entity.name.subStringAfter('-') ', '
self.name.subStringAfter('-') '\n'
        'where ' entity.name.subStringAfter('-') '.' p.name '='
self.name.subStringAfter('-') '.' p.name '\n'
     'and ' entity.name.subStringAfter('-') '.' p.name '= p' '\n'
     'order by ' entity.name.subStringAfter('-') '.' p.name ', '
                 self.name.subStringAfter('-')'.TI, '
self.name.subStringAfter('-')'.TF;'
  }

-------------------------------------------------------
-- Si las consultas son temporales, genera las consultas
-- correspondientes sobre el atributo temporal
-------------------------------------------------------
  qg.TempAtt::mapAsTemporalQuery() {
        property p : qg.ForeignKey = self.foreignKey.first()
     property entity : qg.Node = p.node
     property selfName : String = self.name.subStringAfter('-');

     '\n\n'

     '--Consulta Sobre Modificación de Atributos Temporales' '\n'
        'parameters p ' p.dataType ';\n'
     'select ' selfName '.' self.attribute.first().name ', '
selfName'.TI, ' selfName'.TF' '\n'
     'from ' entity.name.subStringAfter('-') ', ' selfName '\n'
        'where ' entity.name.subStringAfter('-') '.' p.name '=' selfName
'.' p.name '\n'
     'and ' entity.name.subStringAfter('-') '.' p.name '= p' '\n'
     'order by ' selfName '.' self.attribute.first().name ', '
selfName'.TI, ' selfName'.TF;'
     '\n\n'

     '--Consulta Genérica Sobre Valores Atributos Temporales' '\n'
     'parameters t Date, p ' p.dataType ';\n'
     'select ' selfName '.' self.attribute.first().name '\n'
     'from ' entity.name.subStringAfter('-') ', ' selfName '\n'
        'where ' entity.name.subStringAfter('-') '.' p.name '=' selfName
'.' p.name '\n'
     'and ' entity.name.subStringAfter('-') '.' p.name '= p' '\n'
     'and ' selfName'.TI <= t' '\n'
     'and ' selfName'.TF >= t' ';'
  }

-------------------------------------------------
-- Si la consulta es temporal, genera las consultas
-- sobre la relación temporal
-------------------------------------------------
  qg.TempRel::mapAsTemporalQuery() {
        property p : qg.ForeignKey = self.foreignKey.first()
        property q : qg.ForeignKey = self.foreignKey.last()
        property entity : qg.Node = p.node
         property entityq : qg.Node = q.node
         property selfName : String = self.name.subStringAfter('-');

     '\n\n'

     '--Consulta Sobre Vinculo Entre Entidades' '\n'
        'parameters p ' p.dataType ';\n'
```

```
     'select ' entityq.name.subStringAfter('-') '.' q.name ', '
selfName'.TI, ' selfName'.TF' '\n'
     'from ' entityq.name.subStringAfter('-') ', '
entity.name.subStringAfter('-') ', ' selfName '\n'
        'where ' entity.name.subStringAfter('-') '.' p.name '=' selfName
'.' p.name '\n'
        'and ' entityq.name.subStringAfter('-') '.' q.name '=' selfName
'.' q.name '\n'
     'and ' entity.name.subStringAfter('-') '.' p.name '= p' '\n'
     'order by ' entityq.name.subStringAfter('-') '.' q.name ', '
selfName'.TI, ' selfName'.TF;'
     '\n\n'
     '--Consulta Genérica Sobre el Valor del Vínculo Entre Entidades'
'\n'
     'parameters t Date, p ' p.dataType ';\n'
     'select ' entityq.name.subStringAfter('-') '.' q.name '\n'
     'from ' entityq.name.subStringAfter('-') ', '
entity.name.subStringAfter('-') ', ' selfName '\n'
        'where ' entity.name.subStringAfter('-') '.' p.name '=' selfName
'.' p.name '\n'
        'and ' entityq.name.subStringAfter('-') '.' q.name '=' selfName
'.' q.name '\n'
     'and ' entity.name.subStringAfter('-') '.' p.name '= p' '\n'
     'and ' selfName'.TI <= t' '\n'
     'and ' selfName'.TF >= t' ';'
  }

}
```

IV.4. Resumen del Anexo

El objetivo de este anexo fue detallar todas las transformaciones que inicialmente fueron descritas de manera informal en el capítulo 5 (Diseño de un Data Warehouse Histórico) y en el capítulo 7 (Consultas en un Data Warehouse Histórico). Las transformaciones M2M, fueron descritas en **ATL** y las transformaciones M2Text, en MOFScript.

Anexo V

Corroboración Empírica de la Propuesta

V.1. Introducción

La computación es considerada, en general, como una disciplina científica que pone énfasis en tres perspectivas diferentes: la matemática, presente en el desarrollo de formalismos, teorías y algoritmos; la ingeniería, vinculada con el objetivo (propio de cualquier rama de la ingeniería) de hacer las cosas mejores, más rápido, más pequeñas y más económicas y, por último, la ciencia, que puede definirse como la actividad de desarrollar teorías generales y predictivas que permiten describir y explican fenómenos observados y donde esas teorías, además, son evaluadas y puestas a prueba [Den05].

Las disciplinas científicas más consolidadas presentan elaboradas explicaciones sobre sus estrategias de investigación; en ellas, están presentes las nociones de falsación, teorías, leyes, paradigmas y programas de investigación. En cambio, las investigaciones en ingeniería de software raramente describen, en forma explícita, sus paradigmas de investigación y sus estándares de modo tal que permitan juzgar la calidad de sus resultados [Sha02].

En particular, Snodgrass [Sno07] destaca que, si bien la investigación en el campo de las bases de datos presenta importantes desarrollos matemáticos e ingenieriles, la perspectiva científica ha sido escasamente desarrollada. En el mismo trabajo destaca que, en más de 10.000 papers sobre base de datos, estudiados entre los años 1975 y 2000, en solo 37 de ellos se mencionaba la frase ¨testeo de hipótesis¨ y, de ellos, menos de una docena aplicaba realmente ese método.

Debido a que existe una creciente comprensión en la comunidad informática de que los estudios empíricos son necesarios para mejorar los procesos, los métodos y las herramientas para el desarrollo y mantenimiento de software [Bas96], un área emergente en la ingeniería de software, la **Empirical Software Engineering (ESE),** si bien en un escalón un poco más abajo en las pretensiones de cientificidad, tiene como objetivo enfrentar esta falencia. Esta disciplina centra su actividad en experimentos en sistemas de software (productos, procesos y recursos) y propone un riguroso enfoque experimental sobre ellos. En particular, esta rama de la ingeniería de software está interesada

en el diseño de experimentos, en la recolección de datos y en la elaboración de leyes y teorías a partir de ellos.

Este anexo tiene como objetivo, en línea con lo expresado anteriormente referido a la necesidad de evaluar las propuestas de desarrollos en ingeniería de software, corroborar empíricamente el método y los modelos presentados en esta tesis y considerar el impacto de los mismos en un ambiente los más cercano posible al que en la práctica real será utilizado.

Dadas las limitaciones de espacio y considerando que el tema escapa al ámbito específico de la propuesta de tesis, el presente anexo propone un acercamiento que amerita un desarrollo ulterior más profundo.

V.2. Investigación Cualitativa

Si bien los métodos de investigación pueden clasificarse de diversos modos, una tipificación ampliamente aceptada es la que los divide en cuantitativos y cualitativos. Los primeros son especialmente apropiados para el estudio de fenómenos u objetos naturales. Por otro lado, el estudio de fenómenos culturales y sociales requiere de otro tipo de métodos, que no se basen en experimentos ni teorías formales, sino en entrevistas, cuestionarios, documentos, impresiones y reacciones observadas por el investigador. Este segundo grupo cae en el ámbito de los métodos cualitativos [Mye02]. El enfoque cualitativo es inductivo y tiene la particularidad de no ser lineal, sino iterativo y recurrente, las supuestas etapas son acciones para adentrarse más en el problema a investigar, donde la tarea de recolección y análisis es permanente [HFC06].

La investigación enfocada a la construcción de nuevos objetos (procesos, modelos, métodos, técnicas, etc.) son de naturaleza ingenieril, en el sentido de que su objeto de estudio es la construcción de nuevas herramientas (métodos, modelos, etc.) para la construcción de software. Este tipo de investigación está vinculado a la implantación y uso de estos nuevos productos; los problemas plateados en este ámbito requieren del estudio de factores sociales y culturales; pretenden responder a preguntas tales como: ¿cuáles son los factores por los que un determinado proceso de software no es aceptado en la empresa? ó ¿por qué una herramienta de desarrollo de software tiene más o menos aceptación que otra? Este tipo de problemas no pueden ser abordados únicamente mediante los tradicionalmente llamados "métodos científicos", es decir, mediante métodos puramente cuantitativos; son problemas que deben ser abordados mediante métodos cualitativos [Mar02].

V.2.1. Experimentos Controlados

La investigación de tipo cualitativa puede desarrollarse mediante un experimento controlado, donde una hipótesis testeable de una o más variables independientes es manipulada para medir su efecto sobre una o más variables dependientes. El experimento controlado permite determinar en términos precisos cómo las variables están relacionadas y, específicamente, si existe una relación causa-efecto entre ellas. Cada combinación de valores de la variable independiente es considerada como ámbito de estudio diferente. La forma más simple de realizar un experimento controlado es mediante la sola representación de los dos niveles de una variable independiente (por ejemplo, el uso de una herramienta vs. no usarla) [ESSD07].

Los experimentos controlados en ingeniería de software frecuentemente involucran a estudiantes que resuelven tareas en papel y lápiz; la principal razón es que estos son más accesibles, más fáciles para organizar y, generalmente, sin costo. Sin embargo, esta postura muchas veces es criticada por falta de

realismo, y proponen, por consiguiente, que los experimentos deberían realizarse sobre tareas reales en sobre sistemas reales por profesionales que utilizan sus tecnologías de desarrollo en sus ambientes habituales de trabajo [Sjø+02].

V.2.2. Técnicas Utilizadas de Recolección de Datos

Para el análisis cualitativo, al igual que para el cuantitativo, la recolección de datos es fundamental, solamente que su propósito no es medir variables para realizar inferencias o análisis estadísticos. Lo que se busca en un estudio cualitativo es obtener datos (que se convertirán en información) de personas, contextos o situaciones. Al tratarse de seres humanos, los datos que interesan son conceptos, percepciones, creencias, emociones, etc. [HFC06]

Las entrevistas y cuestionarios, ambas herramientas de recolección de datos, están centradas en preguntas sobre temas puntuales. En los cuestionarios, las preguntas pueden ser de dos tipos: abiertas o cerradas. Las primeras no delimitan de antemano las alternativas de respuesta, sirven para profundizar una opinión o los motivos de un comportamiento, requiere de un mayor tiempo de procesamiento. Las preguntas de tipo cerradas contienen categorías o alternativas de repuestas que han sido delimitadas, son más fáciles de codificar y de preparar para su análisis pero limitan las respuestas de la muestra y no describen con exactitud lo que las personas tienen en su mente. Una entrevista, por otro lado, es la obtención de información a través de una conversación de naturaleza profesional. En la entrevista, el instrumento de evaluación es el cuestionario, pero con la diferencia que es el observador quien realiza la pregunta y éstas no siempre son fijas [CL04].

En el enfoque cualitativo, al no interesar tanto la posibilidad de generalizar los resultados, las muestras no probabilísticas o dirigidas constituyen una opción adecuada en la elección del grupo de estudio [HFB06].

V.3. Trabajo de Investigación

El trabajo de investigación realizado para evaluar empíricamente la propuesta presentada en la tesis fue de tipo cualitativa, se desarrolló mediante un experimento controlado, a través de cuestionarios con preguntas de tipo abierta. El trabajo de investigación se realizó con estudiantes de la Maestría en Tecnología Informática de la facultad de Tecnología Informática de la Universidad Abierta Interamericana.

V.3.1. Objetivos

Los objetivos perseguidos en esta investigación están vinculados a los tres aportes principales de la tesis y pueden resumir en los siguientes ítems:

- Evaluación del método de diseño de **HDW.**
- Evaluación de la estructura de almacenamiento**.**
- Evaluación del la interface gráfica.

V.3.2. Hipótesis de Trabajo

Las propuestas presentadas en la tesis parten inicialmente de tres hipótesis que, aunque no planteadas en forma explícita, pueden ser consideradas como tales y, por lo tanto, permitirán ser corroboradas empíricamente, a saber: Hipótesis 1, el método de diseño aumenta la productividad del desarrollador de aplicaciones informáticas; Hipótesis 2, la estructura de almacenamiento

propuesta permite mejorar el proceso de toma de decisión; Hipótesis 3, el uso de interfaces gráficas mejora la productividad del usuario tomador de decisión.

Plantemos, por consiguiente, tres hipótesis del tipo causa-efecto, donde utilizaremos, para cada una de ellas, una variable dependiente, **VD**, (la variable a explicar, el objeto de la investigación) y una variable independiente, **VI**, (la variable explicativa, el factor susceptible a explicar la variable dependiente).

- Hipótesis 1: el método propuesto (**VI**) aumenta la productividad del desarrollador de aplicaciones informáticas (**VD**).

- Hipótesis 2: la estructura de almacenamiento propuesta (**VI**) permite mejorar el proceso de toma de decisión (**VD**)

- Hipótesis 3: el uso de interfaces gráficas (**VI**) mejora la productividad del usuario tomador de decisión (**VD**).

V.3.3. Grupo de Estudio

El grupo de estudio estuvo formado por seis estudiantes de la Maestría en Tecnología Informática, por lo tanto se los asumió con características homogéneas. El grupo de estudio, por simplicidad, asumirá el rol de desarrollador de aplicaciones para evaluar las hipótesis 1) y 2) y de usuario final y para la hipótesis 3).

V.3.4. Desarrollo del Trabajo de Investigación

El grupo de estudio fue instruido, en tres instancias diferentes, sobre las características del método de diseño (hipótesis 1), la estructura de almacenamiento (hipótesis 2) y la interface de consulta (hipótesis 3). La explicación fue realizada evitando hacer comentarios sobre los beneficios y limitaciones de la propuesta. Además, se les solicito a los alumnos describir lo más detalladamente posible sus respuestas.

En el primer caso (método de diseño) se realizo primeramente una explicación del método propuesto y de las características del prototipo, luego cada participante, en su estación de trabajo, utilizó la herramienta en el diseño de un **HDW** a partir de un modelo de datos fuente dado (**ER**). Finalizada la experiencia, se le entregó un cuestionario con preguntas.

En el segundo caso (estructura de almacenamiento) se explicó la estructura de almacenamiento que integra en un solo modelo un **DW** y un **HDB**. Finalizada la experiencia, se le entrego un cuestionario con preguntas.

Por último, en el tercer caso (interface de consulta), se les explicó las características de la interface y su uso y, posteriormente, se les propuso al grupo diferentes tipos de consultas, tanto temporales como de toma de decisión, y se les instó a resolverlas primero en forma manual mediante sentencias **SQL** y, luego, mediante la interface gráfica. Finalizada la experiencia, se le entregó un cuestionario con preguntas.

V.3.4.1. Cuestionarios Para Evaluación de las Hipótesis

Se realizaran tres tipos diferentes de baterías de preguntas con respuestas abiertas, cada una ellas tenía como objetivo determinar en qué medida las hipótesis planteadas eran evaluadas por las personas. Se consideraron los tres temas planteados en las hipótesis:

- El método de diseño

- La estructura de almacenamiento
- La interface gráfica

V.3.4.2. El Método de Diseño

Las siguientes preguntas estuvieron dirigidas a evaluar el impacto del uso de la herramienta de diseño a partir de las consideraciones expresadas por el grupo de trabajo en el uso del prototipo.

1. Para el diseño de una estructura de almacenamiento es importante determinar inicialmente los requerimientos del usuario. El método propuesto comienza utilizando el modelo de la fuente de datos de la aplicación, en particular un **ER**. ¿considera que permite un diseño efectivo de un **HDW**? Evalúe los beneficios e inconvenientes de este enfoque

2. El método propone el marcado de atributos, entidades e interrelaciones temporales y del hecho principal de análisis, a partir del cual se genera el proceso de transformación. Analice la propuesta y evalúela en cuanto a lo intuitivo del método y su eficacia en la determinación de las construcciones temporales.

3. El paso siguiente en el diseño es el marcado del **AG** para la determinación de las dimensiones, medidas y jerarquías temporales y no temporales. Analice la propuesta y evalúela en cuanto a lo intuitivo del método y su eficacia en la determinación de las características distintivas de un **HDW.**

V.3.4.3. La Estructura de Almacenamiento

Las siguientes preguntas estuvieron dirigidas a evaluar el beneficio de una estructura de almacenamiento que integra, en un solo modelo, un **DW** y un **HDB** a partir de la explicación de la misma y de las diferentes consultas temporales y de toma de decisión que se pueden realizar sobre ella.

1. En un **DW** el tiempo está implícito en la estructura de almacenamiento, pero hace referencia al momento en que se realizó una transacción, no detalla como variaron los datos vinculados con ella. Evalúe el modelo integrado por las dos estructuras de almacenamiento.

2. El modelo de datos permite realizar tanto consultas temporales como de toma de decisión ¿Considera que la estructura de datos facilita las consultas y amplia el rango de las mismas? Evalúe ventajas y desventajas del modelo.

3. El modelo admite, además de las típicas consultas de toma de decisión, determinar cómo varían temporalmente atributos, entidades e interrelaciones. Analice los tipos de consultas genéricas que resuelve y detalle sus ventajas y desventajas

V.3.4.4. La interface Gráfica

Las siguientes preguntas estuvieron dirigidas a evaluar el beneficio de la interface gráfica de consultas que permite, mediante marcas, resolver automáticamente

consultas temporales y de toma de decisión a partir del uso del prototipo mediante la realización de consultas sobre la estructura **TMD**.

1. La forma tradicional de obtener información sobre una estructura de almacenamiento es a través de consultas mediante sentencias **SQL**. Las mismas, en general son iterativas y muchas veces complejas para usuarios no técnicos. Evalúe la interface de consultas a partir de su uso.

2. ¿Considera que la interface gráfica constituye una alternativa intuitiva en la realización de consultas? Evalúe ventajas y desventajas.

3. La interface gráfica permite resolver las consultas mediante marcas en el grafo. Evalúe ventajas y desventajas del marcado en el grafo.

V.3.5. Datos Obtenidos

A partir de las preguntas planteadas se describen las respuestas, en forma literal, dadas por los alumnos evaluados. Cada cuadro corresponde a la respuesta de cada alumno evaluado en el orden en que fueron presentadas las preguntas. Dividimos el reporte en tres partes: El método de diseño, la estructura de almacenamiento y la interface gráfica.

Ficha técnica:
Fecha: 19 de junio de 2010
Hora: 12:30
Lugar: facultad de TI, UAI, San Juan 960. Cuidad Autónoma de Buenos Aires
Participantes: 6 (seis) alumnos de la maestría en TI de la UAI

V.3.5.1. El Método de Diseño

A continuación se detallan las respuestas a las preguntas planteadas sobre el método de diseño:

1. Si. Aparenta ser una forma práctica y rápida de generar el HDW.
2. El método es muy intuitivo y eficaz.
3. Resulta fácil marcar en el grafo y determinar así las propiedades que queremos o no conservar.

1. Considero que si permite un diseño efectivo de un HDW. No estoy en condiciones de evaluar beneficios e inconvenientes porque no tengo en mente otros modelos para lograr esto.
2. Es intuitivo. La eficacia no puede evaluarse con el nivel de prueba realizado, como tampoco su robustez. Suena prometedor
3. Es bastante intuitivo. Es positivo el poder eliminar atributos innecesarios para las consultas en los grafos.

1. El **ER** me permite obtener claridad en la definición de los requerimientos del usuario, ya que nos permite definir entidades y relaciones que intervienen en el modelo
2. Es muy importante establecer atributos, entidades e interrelaciones temporales ya que podemos determinar las fluctuaciones y obtener mediciones de las mismas que afectan al desenvolvimiento del negocio.
3. Es muy importante a través del grafo identificar dimensiones, medidas y jerarquías a través de mis conocimientos y mi experiencia podré determinar con más exactitud.

1. Creo que el **ER** permite un diseño efectivo. Ahora bien, no en muchas empresas tienen un DER actualizado como para tomar como punto de partida para aplicar la transformación. Comenzar con una ingeniería inversa a partir de las tablas para obtener el **ER** creo que sería necesaria.
2. Me parece intuitivo.
3. Me parece intuitivo.

1. Es efectivo ya que permite tener claramente identificadas las entidades y relaciones necesarias para el conocimiento del modelo.
2. Es eficaz en la determinación de las construcciones temporales ya que permite obtener consultas sobre variaciones en el tiempo
3. Permite en el grafo podar los atributos que no son importantes y de esa manera hacer un modelo más eficaz.

1. Aparece como adecuado en virtud de que el modelo de datos refleja la esencia del negocio y, por lo tanto, el personal con capacidad técnica apropiada está en condiciones de sugerir un modelo de inicio.
2. Aparece lo suficientemente simple y en la simple práctica llevada a cabo parece ser eficaz.
3. Ídem la respuesta anterior

V.3.5.2. La Estructura de Almacenamiento

A continuación se detallan las respuestas a las preguntas planteadas sobre la estructura de almacenamiento:

1. El modelo integrado permite efectuar consultas complejas sobre los datos y su relación temporal.
2. La estructura de datos facilita las consultas y amplia el rango de las mismas.
3. El modelo posibilita una amplia gama de posibilidades en cuanto a las consultas que pueden realizarse y las interrelaciones que permite.

1. Obviamente permite obtener muchísima información más, sobre todo en lo relativo a la toma de decisiones; claro está que también es otro nivel de recursos y complejidad y supongo que de velocidad de acceso.
2. Si, facilita la estructura de datos y amplia el rango, y sobre todo hacer combinaciones, por lo que cuando hay muchas variables de acceso a la información ha de potenciarse la utilizada alternativa que ofrece este desarrollo. Para describir las desventajas debería compararlos con otras estructuras de datos, cosa que no me fue posible de evaluar durante esta experiencia, aunque estimo que una es que es poco práctica para consultas estáticas acotadas.
3. Parece que permite realizar una gran cantidad de consultas típicas del código **SQL**

1. Es muy importante tener definidas las dos estructuras de almacenamiento integradas en un modelo ya que me permiten obtener las oscilaciones o variaciones de los datos a través de atributos temporales (consultas)
2. Sí, me parece óptimo el modelo integrado para la realización de consultas temporales o de toma de decisión ya que me permite ver la estructura del negocio.
3. Sí, me permiten obtener consultas con el efecto del tiempo que es muy importante en las decisiones del negocio para futuros lineamientos. Para ello, es necesario tener bien claro mi modelo donde establezco atributos, relaciones, jerarquías y dimensiones

1. Me parece una buena idea para hacer análisis más completos.
2. Creo que esta estructura facilita el armado de las consultas, y la posibilidad de que nuevas consultas sean realizadas por personas sin conocimientos de **SQL**.
3. Las consultas genéricas que resuelve son las más importantes. No estoy seguro si puede resolver consultas complejas.

1. El grafo de consulta integra en un solo modelo la estructura transaccional y los datos históricos.
2. Permite crear la consulta **SQL** rápidamente seleccionando el orden de consulta.
3. Resuelve consultas temporales históricas acerca de atributos que hayamos seleccionados como temporales, además resuelve consultas de toma decisión habiendo marcando previamente las entidades para dicha consulta y se selecciona qué tipo de función y luego se define el orden de consulta que generara un **SQL**

1. Obviamente resulta más completa al permitir aumentar la complejidad de las consultas en relación a cambios temporales
2. Sin duda potencia las consultas en virtud de complementarlas con la variación de los atributos
3. La posibilidad de incluir cambios temporales de los atributos puede permitir comprender determinados comportamientos de las entidades

V.3.5.3. La Interface Gráfica

A continuación se detallan las respuestas a las preguntas planteadas sobre la interface gráfica:

1. La interface es muy buena y facilita notablemente la realización de consultas, sin tener que escribirlas en **SQL**, ni manejar ese lenguaje.
2. Sí, la interface constituye una excelente alternativa para la realización de consultas por el usuario en aspectos de toma de decisiones.
3. Es muy simple de utilizar y proporciona buenas funcionalidades a la hora de plantear las consultas.

1. Es muy buena idea pero todavía la interface está en un estado rudimentario: todos tuvimos que preguntar cómo usarla y un usuario no capacitado no podría hacer aún nada con ellas. Se dijo que está en estado experimental y así es.
2. Por supuesto; me remito al comentario anterior, es muy buena idea pero todavía no salió de ese estado.
3. Cuando sean "marcas" posiblemente sea muy bueno, ya que hoy por hoy hay que revisar el estado de todos los grafos para hacer cualquier consulta: no hay nada en los gráficos que indiquen su selección ni orden.

1. La forma tradicional es tediosa en cuanto que necesito conocer el lenguaje (sintaxis) para poder establecer consultas en tiempo y forma, limitando su uso si la persona no es informática.
2. Me facilita la identificación de los elementos en base a mis conocimientos en el desenvolvimiento del manejo del negocio.
3. Las marcas en el grafo me permiten definir con claridad las consultas permitiendo identificar atributos temporales o no, ordenamientos y demás elementos que intervienen en la ejecución de la consulta para luego concluir con la definición de la consulta en **SQL**.

1. Sobre la interface gráfica, hay que usarla un poco para entender el criterio con el que hay que elegir las entidades y las dimensiones. Una vez entendido esto, que no es complejo, generar las consultas **SQL** es algo bastante sencillo, y puede realizarlo una persona sin conocimientos de **SQL**.
2. Como decía en el punto anterior, lleva unos minutos entender la lógica de la interface.
3. El marcado es sencillo, una vez que se entiende la lógica.

1. Es más simple ya que hay que definir qué atributos serán temporales y las entidades que servirán para toma de decisión.
2. Viendo el modelo y posicionándose en la entidad que se definieron para toma de decisiones es fácil definir cuál es la función que se quiere consultar y sobre que grupos identificando con un número el orden del grupo.
3. Si. Ídem respuesta anterior.

1. Resulta una modalidad más sencilla para usuarios no técnicos, obviamente y en el producto terminado, con una interface gráfica más amigable
2. Si, dentro de la práctica que llevamos a cabo en el aula
3. No encontré desventajas dentro del ámbito de la practica

V.3.6. Conclusión

A partir de la evaluación de las respuestas descritas anteriormente, podemos sacar algunas conclusiones sobre las propuestas presentadas en la tesis.

El objetivo en la evaluación del método de diseño era, primeramente, establecer si comenzar el diseño por el modelo de datos era una opción efectiva. En virtud de las respuestas, este enfoque parece ser apropiado. El

planteo de qué sucede cuando no poseo este insumo inicial y la propuesta de usar ingeniería inversa para obtener el modelo de datos abre un ámbito de investigación futuro, conjuntamente con la posibilidad de evaluar otro tipo de captura de requerimientos, que eventualmente pueda traducirse en un modelo de datos que se adapte al método propuesto.

Respecto de lo intuitivo del marcado del modelo como medio para obtener las construcciones temporales y el hecho principal de análisis, en general fue considerado intuitivo, aunque la idea estaba diezmada debido a las limitaciones del prototipo; no obstante, con alguna práctica, resultaba amigable. Por último, la evaluación del marcado en el **AG** para la determinación de las dimensiones, medidas y jerarquías temporales y no temporales tuvo, en general, respuestas similares al punto anterior.

En la evaluación de la estructura de almacenamiento integrada, se puede apreciar una opinión favorable ya que, en general, es consideraba una propuesta innovadora que facilita las consultas combinadas permitiendo realizar un rango mayor de consultas temporales o de toma de decisión.

Por último, el uso de la interface gráfica resultó, luego de un breve entrenamiento, simple e intuitiva. La construcción de consultas automáticas resultaba una forma sencilla de obtener información para usurarios no técnicos. Si bien el marcado actualmente está implementado mediante modificaciones de propiedades del modelo, en general evalúan que la idea es aplicable y consideran que el marcado es sencillo una vez que se entiende la lógica.

En resumen, a partir de la evaluación realizada y teniendo en cuenta las limitaciones de la misma, consideramos que la propuesta presentada en la tesis constituye una alternativa válida en el diseño de estructuras **TMD**, tanto en la propuesta del método de diseño como en la estructura de almacenamiento y en la interface gráfica de consultas.

V.4. Resumen del Anexo

El objetivo de este anexo fue evaluar empíricamente la propuesta de la tesis a partir del uso, por parte de usuarios reales y en un ambiente controlado, del método de diseño de un **HDW**, la estructura de almacenamiento integrada y la interface gráfica de consultas a partir del uso del prototipo que implementa las ideas principales desarrolladas en la tesis.

Acrónimos

ANSI ***American Nacional Standards Institute***
Organización que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos.

ATL **ATLAS Transformation Language**
Lenguaje de transformación de modelos desarrollado sobre la plataforma ECLIPSE.

BTDB ***BiTemporal Data Base***
Un tipo de base de datos que modela el tiempo válido y el tiempo de transacción.

BNF ***Backus Naur Form***
Meta sintaxis usada para expresar gramáticas libres de contexto.

CIM ***Computation Independent Model***
Modelo que describe al sistema dentro de su ambiente y muestra lo que se espera de él sin exhibir detalles de cómo será construido. Es uno de los modelos de **MDA.**

CWM ***Common Warehouse Metamodel***
Una especificación de **OMG** cuyo propósito es permitir un fácil intercambio de metadata entre herramientas de **DW** y repositorios de metadata en ambientes heterogéneos distribuidos.

DBMS ***Data Base Management System***
Un tipo de software que permite almacenar, modificar y extraer información de una base de datos.

DCL ***Data Control Language***
Sublenguaje de **SQL** que permite controlar el acceso a los datos en una base de datos.

DDL ***Definition Data Language***
Sublenguaje de **SQL** que permite crear, modificar y eliminar estructuras de datos en una base de datos.

DML ***Data Manipulation Language***
Sublenguaje de **SQL** que permite insertar, borrar y recuperar datos en una base de datos.

DSL ***Domain-Specific Language***
Lenguaje de especificación dedicado a un dominio de problema particular.

DSM ***Domain-Specific Modeling***
Iniciativa que propone la creación de modelos para un dominio, en un **DSL** apropiado, que permite especificar directamente la solución usando conceptos del dominio del problema.

DSS ***Decision Support System***
Un sistema computarizado interactivo que recupera y presenta datos, normalmente para fines empresariales. Las aplicaciones **DSS** ayudan a las personas a tomar decisiones basadas en datos recuperados de diversas fuentes.

DW ***Data Warehouse***
Una colección de datos diseñados para ayudar al proceso de toma de decisión. El **DW** contiene una amplia variedad de datos que representan una visión coherente de las condiciones del negocio en un momento determinado.

EMOF ***Essential Meta-Object Facility***.
Una especificación de **MOF** que se usa para la definición de metamodelos simples, usando conceptos simples.

ER ***Entity-Relationship***
Un modelo de datos conceptual que visualiza al mundo real mediante entidades, interrelaciones y atributos.

ERR ***Extended Entity-Relationship***
Un modelo de datos conceptual que incluye toda la semántica de **ER** y construcciones adicionales que permiten modelar conceptos tales como clases, subclases, semántica temporal, etc.

ESE ***Empirical Software Engineering***
Rama de la Ingeniería de Software interesada en el diseño de experimentos en sistemas software, en la recolección de datos de esos experimentos y en la elaboración de leyes y teorías sobre esos datos.

ETL ***Extract, Transform and Load***
Proceso que permite mover datos desde múltiples fuentes, reformatearlos y limpiarlos, y cargarlos en otra base de datos o **DW.**

GEF ***Graphical Editing Framework.***
Un *framework* implementado para la plataforma Eclipse que ayuda en el desarrollo de componentes gráficos.

GMF ***Graphical Modeling Framework***
Un *framework* para modelado gráfico dentro del proyecto Eclipse.

HDB ***Historical Data Base***
Un tipo de base de datos temporal que modela solamente el tiempo válido.

HDW ***Historical Data Warehouse***
Un tipo de **DW** que permite ampliar su funcionalidad admitiendo, además, consultas temporales.

HOLAP ***Hybrid OnLine Analitycal Processing***
Un tipo de **OLAP** que combina características de **ROLAP** y **MOLAP**. Los productos **HOLAP** almacenan la mayoría de los datos en una base de datos relacional y en una base de datos **MD.**

IM ***Implementation Model***
Código fuente derivado de cada **PSM.** Es uno de los modelos de **MDA.**

ISO ***International Standards Organization***
Organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y la electrónica.

MD ***Multidimensional***
Término que se aplica a cualquier sistema que es diseñado para analizar gran cantidad de datos organizados en un espacio n-dimensional.

MDA ***Model Driven Architecture***
Propuesta por **OMG,** es una arquitectura que, siguiendo el enfoque **MDD,** proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos.

MDD ***Model Driven software Development***
Enfoque basado en modelos para el diseño e implementación de sistemas software. Los modelos son considerados los conductores primarios en todos los aspectos del desarrollo de software.

MML ***Maturity Model Level***
Concepto mediante el cual se puede descubrir en qué estado de madurez se encuentra un grupo de desarrollo de software respecto al uso de modelos.

MOF ***Meta Object Facility***
Un estándar de **OMG** usado para definir y manipular un conjunto de metamodelos interoperables y sus respectivos modelos.

NTS ***Non-Temporal Schema***
Esquema de datos que no contempla la posibilidad de registrar explícitamente la variación temporal de datos.

OCL ***Object Constraint Language***
Lenguaje declarativo usado para establecer reglas aplicados a modelos **UML.**

ODB ***Operational Data Base***
Un tipo de **DBMS** que se utilizan en sistemas **OLTP.**

QG ***Query Graph***
Grafo que permite establecer, para un usuario del **HDW**, consultas sobre la estructura de almacenamiento que, posteriormente, serán traducidas a sentencias **SQL**, en forma automática.

MOLAP ***Multidimensional OnLine Analitycal Processing***
Un tipo de **OLAP**, que provee análisis **MD** de datos usando una estructura de cubos para almacenar los datos.

OLAP ***OnLine Analitycal Processing***
Una categoría de herramienta de software que provee análisis de datos. Las herramientas **OLAP** permiten a los usuarios estudiar datos mediante funciones especiales para análisis de los mismos.

OLTP ***OnLine Transaction Processing***
Un sistema que administra aplicaciones transaccionales para entrada, procesamiento y recuperación de datos.

OMG ***Object Management Group***
Consorcio internacional que dirige y provee un marco común para el desarrollo de aplicaciones usando técnicas de programación orientadas a objetos.

OODBMS ***Object-Oriented Data Base Management System***
Un tipo de **DBMS** que incorpora los conceptos más importantes del paradigma de objetos.

ORDBMS ***Object-Relational Data Base Management System***
Un tipo de **DBMS** que incluye conceptos relacionales y orientados a objetos.

PIM ***Platform Independent Model***
Modelo de un alto nivel de abstracción que es independiente de cualquier tecnología de implantación. Es uno de los modelos de **MDA.**

PSM ***Platform Specific Model***
Modelo que describe en detalle cómo un **PIM** es implementado sobre una plataforma o tecnología especifica. Es uno de los modelos de **MDA.**

PDM ***Platform Definition Model***
Especifica el metamodelo utilizado de la plataforma destino. Es uno de los modelos de **MDA.**

QG ***Query Graphic***
Grafo construido automáticamente a partir de un modelo **TMD,** como paso previo a la construcción de consultas **SQL.**

QVT ***Query View Transformation***
Iniciativa de **OMG** cuyo objetivo es proveer un estándar para expresar transformación entre modelos. Las transformaciones se definen en forma precisa en términos de relaciones entre metamodelos basados en **MOF.**

RBDB ***RollBack Data Base***
Un tipo de base de datos que modela solo el tiempo de transacción.

RDBMS ***Relational Data Base Management System***
Un tipo de **DBMS** que soporta el modelo relacional, donde los datos son almacenados como tablas relacionadas.

RM ***Relational Model***
Un modelo de datos que considera la base de datos como un conjunto de relaciones.

ROLAP ***Relational OnLine Analitycal Processing***
Un tipo de sistema **OLAP** que utiliza, para almacenar los datos, un **RDBMS.**

SDB ***Statistical Data Base***
Un tipo de base de datos que se utiliza para propósitos de análisis estadístico.

SQL ***Structured Query Language***
Un lenguaje de consulta, estandarizado, usado para recuperar información en bases de datos. También puede ser utilizado para manipular datos y estructuras de datos.

TBD ***Temporal Data Base***
Un tipo de **DBMS** que permite realizar consultas temporales.

TDSS ***Temporal Decision Support System***
Un tipo particular de **DDS** que considera explícitamente aspectos temporales.

TDW ***Temporal Data Warehouse***
Un tipo de **DW** que registra temporalmente la modificación de dimensiones tanto en el esquema como en las instancias.

TER ***Temporal Entity-Relationship***
Un tipo especializado de **ERR** que modelo aspectos temporales.

TAG ***Temporal Attribute Graph***
Grafo construido automáticamente a partir de un modelo **TER,** como paso previo a la construcción de un **HDW.**

TMD ***Temporal Multidimensional***
Modelo **MD** que considera explícitamente aspectos temporales.

TS ***Temporal Schema***
Esquema de datos que contempla la posibilidad de registrar explícitamente la variación temporal de datos

UML ***Unified Modeling Language***
Un lenguaje gráfico, estandarizado, para la visualización, especificación, construcción y documentación de sistemas software.

XMI ***Extensible Markup Language Metadata Interchange***
Formato de intercambio basado en **XML** para modelo en la capa M1 y M2.

XML ***Extensible Markup Language***
Lenguaje de marcado extensible para el intercambio de datos.

Referencias

[ACDS02] Aversano, L., G. Canfora, A. De Lucia, S. Stefanucci. "Understanding SQL through Iconic Interfaces," *Proceedings 26th Annual International Computer Software and Applications Conference*, 2002, pp. 703-708.

[AF99] Artale, Alessandro and Franconi, Enrico. Reasoning with enhanced Temporal Entity-Relationship Models. Knowledge Representation Meets Databases. Pp.1-5, 1999.

[AGS97] Agrawal, R. Gupta, A. Sarawagi, S. Modeling Multidimensional Databases, Research Report, IBM Almaden Research Center, San Jose, California, 1995. Appeared in Proceeding. ICDE '97.

[AK02] D. H. Akehurst and S. J. H. Kent, ''A Relational Approach to Defining Transformations in a Metamodel,'' Proceedings of the 5th International Conference on the Unified Modeling Language, Dresden, Germany (2002), pp. 243–258.

[AM03a] Abelló, Alberto and Martín, Carme. A Bitemporal Storage Structure for a Corporate Data Warehouse. Short paper in International Conference on Enterprise Information Systems (ICEIS 2003). Angers (France). April, 2003.

[AM03b] Abelló, Alberto and Martín, Carme The Data Warehouse: A Temporal Database. In Proceedings of Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2003). Alacant (Spain), November 2003. Pages 675-684. Campobell S.L., ISBN 84-688-3836-5.

[AS08] Anurag, D. Sen, A.K The Chronon Based Model for Temporal Databases. 13th International Conference on Database systems for advance applications (DASFAA 2008) proceedings, New Delhi, India, March 2008, LNCS 4947, Springer, pp. 461-469.

[ATL] ATLAS Transformation Language) http://www.eclipse.org/m2m/atl/

[Atom3] Atom3 home page. http://atom3.cs.mcgill.ca/index_html

[AYW08] Brett Allenstein , Andrew Yost , Paul Wagner , Joline Morrison, A query simulation system to illustrate database query execution, Proceedings of the 39th SIGCSE technical symposium on Computer science education, March 12-15, 2008, Portland, OR, USA

[Bad04] Antonio Badia, Entity-Relationship modeling revisited, ACM SIGMOD Record, v.33 n.1, March 2004

[Bad04] Basili, V.R. The Role of Experimentation in Software Engineering: Past, Current, and Future, Proceedings of the 18th International

Conference on Software Engineering, Berlin, Germany, March 25-29, pp. 442-449, 1996.

[Bla+09] Blanco, C., Garca-Rodrguez de Guzmn, I., Fernández-Medina, E., Trujillo, J., Piattini, M.: "Applying QVT in order to implement Secure Data Warehouses in SQL Server Analysis Services", Journal of Research and Practice in Information Technology, 41, 2 (2009), 119-138.

[BSW92] Mc Brien P, Seltveit A. Wangler B. An Entity-Relationship Model Extended to Describe Historical Information. CISMOD pp. 244-260 July 1992.

[CCS93] Codd, E.F. Codd, S.B. Salley, C.T.: "Providing OLAP to user-analysts. An IT mandate". Technical Report. E.F. Codd and Associates, 1993.

[CD97] Chaudhuri, Surajit and Dayal, Umesh An Overview of Data Warehousing and OLAP Technology, ACM SIGMOD Record 26(1), March 1997.

[CH06] K. Czarnecki, S. Helsen, Feature – based survey of model transformation approaches, IBM Systems Journal, Vol 45, NO 3, 2006.

[Che76] Chen, Peter, "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems, 1(1)9-36, March 1976.

[CHM+02] Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varró, D.: VIATRA: visual automated transformations for formal verification and validation of UML models. In: Proceedings 17th IEEE international conference on automated software engineering (ASE 2002), pp. 267–270, Edinburgh, UK. (2002).

[CKZ03] Cindy Xinmin Chen, Jiejun Kong, and Carlo Zaniolo. Design and implementation of a temporal extension of SQL. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, ICDE, pages 689–691. IEEE Computer Society, 2003.

[CL04] Cataldi, Zulma, Lage, Fernando. Diseño y organización de tesis. Nueva Librería. Buenos Aires. 2004.

[Cli+97] J. Clifford, C. E. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "Now" in Databases. ACM Trans. on Database Systems , TODS, 22(2), 1997.

[CMP07] Carlo Combi, Angelo Montanari, and Giuseppe Pozzi. The T4SQL Temporal Query Language. ACM CIKM 2007 Sixteenth Conference on Information and Knowledge Management. Pages 193-202, Lisbon, Portugal, Nov. 2007.

[CWM] OMG, Common Warehouse Metamodel Specification. OMG Document formal/2003- 03-02, available at http://www.omg.org/cwm/

[Den05] Denning P. J., "What is Experimental Computer Science?" CACM

23(10):543–544, October, 1980.

[DW02] Date, C. and Darwen, H. Temporal Data and the Relational Model, Morgan Kaufmann Publishers Inc, 2002.

[ECLIPSE] The Eclipse Project. Home Page. http://www.eclipse.org/. Copyright IBM Corp, 2000.

[EC00] Eder, Johann. Concilia, Christian. Evolution of Dimension Data in Temporal Data Warehouses Technical Report, 2000.

[EC02] Eder, Johann. Concilia, Christian. Representing Temporal Data in Non-Temporal OLAP Systems Proceedings of the VLDWH Workshop 2002 (DEXA 2002), Aix-en-Provence, France, September 2-3, 2002, IEEE No PR01668, ISBN 0-7695-1668-8, ISSN 1529-4188, page 817-821.

[EK93] Elmasri, R. and Kouramajian, V. A Temporal Query Language for a Conceptual Model. In N. R. Adam and B. K. Bhargava, editors, Advanced database systems, Volume 759 of Lecture Notes in Computer Science, Chapter 9, pp. 175–195. Berlin, Springer-verlag, 1993.

[EK02] Eder, Johann. Koncilia, Christian. Representing Temporal Data in Non-Temporal OLAP Systems Proceedings of the VLDWH Workshop 2002 (DEXA 2002), Aix-en-Provence, France, September 2-3, 2002, IEEE No PR01668, ISBN 0-7695-1668-8, ISSN 1529-4188, page 817-821.

[EKK02] Eder, Johann. Koncilia, Christian. Kogler, Herbert: Temporal Data Warehousing: Business Cases and SolutionsProceedings of the 4th International Conference on Enterprise Information Systems, Ciudad Real-Spain, 3-6 April 2002, ICEIS Press, Volume 1, page 81-88, ISBN 972-98050-6-7.

[EKM01] Eder, J. Koncilia, C. and Morzy, T.. A Model for a Temporal Data Warehouse. In Proc. of the Int. OESSEO 2001 Conference, Rome, Italy, 2001.

[EKM02] Eder, J. Koncilia, C. Morzy, T. The COMET Metamodel for Temporal Data Warehouses. Proc. Of the 14th Int. Conference on Advanced Information Systems Engineering (CAISE'02). Canada. 2002.

[EMF] Eclipse Modeling Framework http://www.eclipse.org/modeling/emf/

[EN97] Elmasri, Navathe. Sistemas de Base de Datos - Conceptos Fundamentales - Addison- Wesley Iberoamericana, segunda edición. 1997.

[Epsilon] Epsilon Home page http://www.eclipse.org/gmt/epsilon/

[ESSD07] S. Easterbrook, J. Singer, M. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. Guide to Advanced Empirical Software Engineering, 2007.

[EWK93] Elmasri, R. Wuu, G. and Kouramajian, V. A Temporal Model and Query Language for EER Databases. In A. Tansel et al., editor,

Temporal Databases:Theory, Design, and Implementation, Chapter 9, pp. 212–229. Benjamin/Cummings Publishers, Database Systems and Applications series, 1993.

[FKSS06] Fails, J., Karlson, A., Shahamat, L., Shneiderman, B. A Visual Interface for Multivariate Temporal Data: Finding Patterns of Events over Time Proceedings of IEEE Symposium on Visual Analytics Science and Technology (VAST 2006), 167-174. 2006.

[FM96] M. Finger and P. McBrien. On the Semantics of 'Current-Time' in Temporal Databases. In 11th Brazilian Symposium on Databases, pp. 324–337, 1996.

[GCR06] Grant, E. S., Chennamaneni R. Reza, H. Towards analyzing UML class diagram models to object-relational database systems transformations, Proceedings of the 24th IASTED international conference on Database and applications 2006, Innsbruck, Austria, 129 – 134. 2006.

[GHRU97] Gupta, H. Harinarayan, V. Rajaraman, A. and Ullman, J. Index Selection for OLAP. Proceeding ICDE '97. 1997.

[GJ98] Gregersen, H. and Jensen, C. S. Conceptual Modeling of Time-Varying Information. TR-35. September 10, 1998.

[GJ99] Gregersen, H. and Jensen, C. S. Temporal Entity Relationship Models -a Survey. IEEE Trans. on Knowledge and Data Engineering, 11(3):464--497, 1999.

[GL03] M. Gogolla, A. Lindow. Transformations Data Models whit UML. In B. Omelayenko and M. Klein, editors. Knowledge Transformations for the Semantic Web, pages 18-33 IOS Press, Amsterdam. 2003.

[GMF] The Eclipse Graphical Modeling Framework . http://www.eclipse.org/modeling/gmf/

[GMJ98] Gregersen, H. Mark, L. Jensen, C. S. Mapping Temporal ER Diagrams to Relational Schemas. TR-39. December 4, 1998.

[GMR98a] Golfarelli, M. Maio, D. Rizzi, S. Conceptual Design of Data Warehouses from E/R schemes, Proceedings 31st Hawaii International Conference on System Sciences, 1998.

[GMR98b] Golfarelli, M. Maio, D. Rizzi, S. "The Dimensional Fact Model: A Conceptual Model for Data Warehouses"; Int. Journal of Cooperative Information Systems 1998.

[GT08] Glorio O and Trujillo J., "An MDA Approach for the Development of Spatial Data Warehouses," in DaWaK, Turin, Italy, 2008, pp. 23–32.

[HCM03] Holcombe M., Cowling A., Macias F., "Towards an Agile Approach to Empirical Software Engineering" in Proceedings of the ESEIW 2003 Workshop on Empirical Studies in Software Engineering, WSESE 2003, Roman Castles, Italy, Sep 29 2003, pp. 37-48. [Judd 91] Judd C. M.,

[HE00] Hubler, Patrícia Nogueira. Edelweiss Nina: Implementing a Temporal Database on Top of a Conventional Database: Mapping of the Data Model and Data Definition Management. SBBD 2000: 259-272.

[HFC06] Hernández Sampieri R. Fernández-Collado C. Baptista Lucio P. Metodología de la investigación. McGraw Hill. México. 2006.

[HMV99] Hurtado, C., Mendelzon, A. & Vaisman, A., Maintaining Data Cubes Under Dimension Updates, in `Proc. of the 15th Int. Conf. on Data Engineering', pp. 346-355. 1999.

[HRU96] Harinarayan, V., Rajaraman, A. & Ullman, J.D. Implementing data cubes efficiently. ACM SIGMOD Record, 25(2): 205--216. 1996.

[Inm02] Inmon, W. Building the Data Warehouse. John Wiley & Sons, 2002.

[Jen+94] Christian S. Jensen et al. A Consensus Glossary of Temporal Database Concepts. ACM SIGMOD Record, 23(1):52-65, March 1994.

[JS97a] Jensen, C. S. and Snodgrass, R. T. Semantics of Time-Varying Attributes and Their Use for Temporal Database Design. In Fourteenth International Conference on Object-Oriented and Entity Relationship Modeling, pp. 366-377. 1997.

[JS97b] Jensen, C. S. and Snodgrass, R. T. Temporal Data Management. TR-7, June 9, 1997.

[Kermeta] Kermeta home page http://www.kermeta.org/

[KG95] Kouramajian, V. and Gertz, M., A graphical query language for temporal databases. in Proceedings of 14th International Conference on Object-Oriented and Entity Relationship Modeling, (1995), Springer-Verlag, 388-399.

[Kim96] Kimball, R. The Data Warehouse Toolkit. John Willey and Sons, 1996.

[KS97] Krippendorf, Michael and Song, Il-Yeol. The Translation of Star Schema into Entity-Relationship. DEXA, 1997.

[Kra96] Kraft, P. Temporale Kvaliteter i ER Modeller. Hvordan? Working paper 93, The Aarhus School of Business, Department of Information Science, January 1996.

[KWB03] Kleppe, Anneke G. and Warmer Jos, and Bast, Wim. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[LAW98] Lehner, Wolfgang. Albrecht, Jens. Wedekind, Hartmut: Normal Forms for Multidimensional Databases. SSDBM 1998: 63-72.

[LJ09] B. Liu and H. Jagadish. A spreadsheet algebra for a direct data manipulation query interface. In ICDE, 2009.

[LS97] Lenz, Hans-Joachim. Shoshani, Arie: Summarizability in OLAP and Statistical Data Bases. Ninth International Conference on Scientific and Statistical Database Management, Proceedings, August 11-13, 1997, Olympia, Washington, USA.

[LS05] M. Lawley and J. Steel, ''Practical Declarative Model Transformation with Tefkat, ''Proceedings of Model Transformations in Practice Workshop, MoDELS Conference, Montego Bay, Jamaica (2005),

[Mar02] Marcos, E. (2002) Investigación en Ingeniería del Software vs Desarrollo Software. Actas de 1er Workshop en Métodos de Investigación y Fundamentos Filosóficos en IS y SI. November, pp. 136-149.

[MDA] MDA. Model Driven Architecture. 2004. http://www.omg.org/cgi-bin/doc/formal/03-06-01

[ME99] Moreira, V.P. and Edelweiss, N.. Schema Versioning: Queries to the Generalised Temporal Database System. in In: INTERNATIONAL WORKSHOP ON SPATIO-TEMPORAL DATA MODELS AND LANGUAGES. 1999. Florence, Italy: IEEE.

[Medini] Medini Home page. http://projects.ikv.de/qvt

[ModelMorf] ModelMorf http://www.tcs-trddc.com/ModelMorf/index.htm

[MOF] OMG, MOF Meta Object Facility Specification, OMG Document formal/ http://www.omg.org/spec/MOF/2.0/zzPDF

[Mof2Text] MOF Models to Text Transformation Language. OMG Final AdoptedSpecification. http://www.omg.org/docs/ptc/ January 2008.

[Mola] Mola home page http://mola.mii.lu.lv/

[MOT07] Mazón, J.N., Ortega, E., Trujillo, J.: Ingeniería inversa dirigida por modelos para el diseño de almacenes de datos. In: JISBD. 2007.

[MS06] Svetlana Mansmann and Marc H. Scholl. Extending visual OLAP for handling irregular dimensional hierarchies. In *DaWaK*, pages 95 (105, 2006).

[MSW92] McBrien, P. Seltveit, A. H. and Wangler, B. An Entity-Relationship Model Extended to describe Historical information. In International Conference on Information Systems and Management of Data, pages 244--260, Bangalore, India, July 1992.

[MT02] Medina E., Trujillo J. A Standard for Representing Multidimensional Properties: The Common Warehouse Metamodel (CWM). In proceedings of the 6 East-European Conference on Advances in Databases and Information Systems (ADBIS'02), volume 2435 of Lecture Notes in Computer Science, pages 232-247, Bratislava, Slovakia. September, 2002. Springer-Verlag.

[MT09] Jose-Norberto Mazon and Juan Trujillo. A Hybrid Model Driven Development Framework for the Multidimensional Modeling of Data Warehouses. Jose-Norberto Mazon and Juan Trujillo. SIGMOD Record, June 2009 (Vol. 38, No. 2).

[MTF] Model Transformation Framework (MTF), IBM United Kingdom Laboratories Ltd., IBM alphaWorks (2004), http://www.alphaworks.ibm.com/tech/mtf.

[MTL06] Jose-Norberto Mazón, Juan Trujillo, Jens Lechtenbörger: A Set of QVT Relations to Assure the Correctness of Data Warehouses by Using Multidimensional Normal Forms. ER 2006: 385-398.

[MTSP05] Mazón Jose Norberto, Trujillo Juan, Serrano Manuel, Piattini Mario: Applying MDA to the Development of Data Warehouses. DOLAP 2005: 57-66.

[Mye02] Myers, M. D. Qualitative Research in Information Systems. MIS Quarterly, 21:2, pp 241-242, junio 1997. Recuperado de MISQ Discovery.

[MZ04] Malinowski, E. Zimányi, E: OLAP Hierarchies: A Conceptual Perspective. CAiSE 2004: 477-491.

[MZ05] Malinowski, E. & Zimányi, E. (2005), Hierarchies in a multidimensional model: from conceptual modeling to logical representation. Accepted for publication in Data & Knowledge Engineering. 2005.

[NA02] Neil Carlos, Ale Juan. A Conceptual Design for Temporal Data Warehouse. 31° JAIIO. Santa Fe. Simposio Argentino de Ingeniería de Software. 2002.

[Nar88] Narasimhalu. A Data Model for Object-Oriented Databases With Temporal Attributes and Relationships. Technical report, National University of Singapore, 1988.

[NBP07] Neil C, Baez M, Pons C. Usando ATL en la Transformación de Modelos Multidimensionales Temporales. XIII Congreso Argentino de Ciencias de la Computación. Corrientes y Resistencia, Argentina. 2007.

[NP06] Neil Carlos, Pons Claudia. Diseño Conceptual de un Datawarehouse Temporal en el Contexto de MDA. XII Congreso Argentino de Ciencias de la Computación. CACIC. San Luis. Argentina. 2006.

[NP07] Neil C, Pons C. Aplicando MDA al Diseño de un Datawarehouse Temporal. VII Jornada Iberoamericana de Ingeniería de Software e Ingeniería del Conocimiento. Lima, Perú. 2007.

[NP08] Neil, Carlos, Pons Claudia. Aplicando QVT en la Transformación de un Modelo de Datos Temporal. Jornadas Chilenas de Computación. Punta Arenas. Chile. 2008.

[OCL] OCL 2.0. The Object Constraint Language Specification – for UML 2.0, revised by the OMG, http://www.omg.org, April 2004.

[Old06] Oldevik, Jon. MOFScript User Guide . Version 0.6 (MOFScript v 1.1.11)

[OMG] Object Management Group. http://www.omg.org

[OMG04] MOF Model to Text Transformation Language. Request For Proposal. OMG Document: ad/2004-04-07. (2004).

[ON01] Owei, V. and Navathe, S. B. (2001) Enriching the conceptual basis for query formulation through relationship semantics in databases. *Inf. Syst.*, 26, 445–475.

[Pat04] Patrascoiu, O. ''YATL: Yet Another Transformation Language, ''Proceedings of the 1st European MDA Workshop, Twente, The Netherlands (2004), pp. 83–90.

[PGP09] Claudia Pons, Roxana Giandini, Gabriela Pérez. Desarrollo de Software Dirigido por Modelos. Conceptos teóricos y su aplicación práctica. cantidad de pgs. 300. Editorial: McGraw-Hill Education. 2009.

[PJ98a] Pedersen, T. B. and Jensen, C. S. Multidimensional Data Modeling for Complex Data. 1998. ICDE 1999:336-345.

[QVT] MOF 2.0 Query/View/Transformations - OMG Adopted Specification. March 2005. http://www.omg.org

[RA05] O. Romero and A. Abelló. Improving Automatic SQL Translation for ROLAP Tools. Proc. of JISBD 2005, 284(5):123– 130, 2005.

[RA06] Romero, O., Abelló, A.: Multidimensional design by examples. aWaK 2006. Vol. 4081 of LNCS, 85–94.

[RA08] Oscar Romero, Alberto Abelló: MDBE: Automatic Multidimensional Modeling. ER 2008: 534-535.

[Rei02] Reiss SP (2002) A visual query language for software visualization. In: Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02), pp 80–82.

[RTT08] Ravat, Franck; Teste, Olivier; Tournier, Ronan; Zurfluh, Pilles. Algebraic and Graphic Languages for OLAP Manipulations International Journal of Data Warehousing and Mining, Vol. 4, Issue 1. 2008.

[SBHD98] Sapia, Carsten. Blaschka, Markus. Höfling, Gabi. Dinter, Barbara. Extending the E/R Model for the Multidimensional Paradigm. Proc. International Workshop on Data Warehouse and Data Mining in conjunction with the ER98, Singapore. 1998.

[Sha02] Shaw, M. What makes good research in software engineering? Int. Jour. of Soft. Tools for Tech. Trans., 4(1):1–7, 2002.

[SmartQVT] SmartQVT home page. http://smartqvt.elibel.tm.fr/

[Sno95] Richard T. Snodgrass, editor. The TSQL2 Temporal Query Language. Kluwer, 1995.

[Sno95] Snodgrass, Richard T.: Towards a Science of Temporal Databases. 14th International Symposium on Temporal Representation and Reasoning (TIME'07) 0-7695-2936-8/07.

[Sjø+02] D.I.K. Sjøberg, B. Anda, E. Arisholm, T. Dyba, M. Jørgensen, A. Karahasanovic, E. Koren, and M. Voka´c, "Conducting Realistic Experiments in Software Engineering," Proc. First Int'l Symp. Empirical Software Eng. (ISESE '2002), pp. 17-26, Oct. 2002.

[Sol+07] Soler Emilio, Trujillo Juan, Fernández-Medina Eduardo, Piattini Mario: A set of QVT relations to transform PIM to PSM in the Design of Secure Data Warehouses. ARES 2007: 644-654.

[SQL] SQL. Structured Querry Language. ISO/IEC 9075. http://www.iso.org

[SRME01] Song, I.-Y., Rowen, W., Medsker, C., Ewen, E.: An Analysis of Many-to-Many Relationship Between Facts and Dimension Tables in Dimensional Modeling. In: Proceedings of the Int. Workshop on Design and Management if Data Warehouses (DMDW 2001), Interlaken, 2001, pp. 6-5 - 6-13.

[STBF09] Emilio Soler, Juan Trujillo, Carlos Blanco, Eduardo Fernández-Medina. Designing Secure Data Warehouses by Using MDA and QVT Journal of Universal Computer Science, vol. 15, no. 8 (2009), 1607-1641.

[STFP07a] Soler, E., Trujillo, J., Fernández-Medina, E., Piattini, M.: Una extensión del metamodelo relacional de CWM para representar Almacenes de Datos Seguros a nivel lógico. In: JISBD. (2007).

[STFP07b] Soler E., Trujillo J., Fernández-Medina E., Piattini M.: Aplicación de QVT al Desarrollo de Almacenes de Datos Seguros: Un Caso de Estudio. IDEAS 2007. Isla Margarita (Venezuela).

[STFP07c] Soler, E., Trujillo, J., Fernández-Medina, E., Piattini, M.: Un Conjunto de Transformaciones QVT para el Modelado De Almacenes de Datos Seguros. In: JISBD Workshops (DSDM), (2007).

[Tae03] G. Taentzer, ''AGG: A Graph Transformation Environment for Modeling and Validation of Software,'' Application of Graph Transformations with Industrial Relevance (AGTIVE'03) 3062, pp. 446–453 (2003).

[Tan04] Abdullah Uz Tansel: Temporal Data Modeling and Integrity Constraints in Relational Databases. International Symposium on Computer, Information Sciences ISCIS 2004:459-469.

[Tau91] Tauzovich, B. Toward Temporal Extensions to the Entity-RelationshipModel. In The 10th International Conference on the Entity Relationship Approach, pp. 163–179, October 1991.

[TBJS99] Tryfona, N. Busborg, F. Christiansen, J.G.B. starER: A Conceptual Model for Data Warehouse Design. In ACM Second International

Workshop on Data Warehousing and OLAP (DOLAP'99), pp.3-8, November 1999, Missouri, USA.

[TLW91] Theodoulidis, Charalampos I. Loucopoulos, Pericles and Wangler, Benkt. A conceptual modelling formalism for temporal database applications. Information System, 16(3): 401-416, 1991.

[Together] http://www.borland.com/us/products/together/index.html

[TSH01] D. Tang C. Stolte and P. Hanrahan, "Polaris: A system for query, analysis and visualization of multi-dimensional relational databases," Transactions on Visualization and Computer Graphics, 2001.

[UD03] Urban, S. D. and Dietrich, S. W. 2003. Using UML class diagrams for a comparative analysis of relational, object-oriented, and object-relational database mappings. SIGCSE Bull. 35, 1 (Jan. 2003), 21-25.

[UML] OMG, Unified Modeling Language Specification, Version 2.2, 2009, OMG Document formal/ 2009-02-04, available at. http://www.omg.org/spec/UML/2.2/Infrastructure/PDF

[Wid95] Jennifer Widom, Research Problems in Data Warehousing, International Conference on Information and Knowledge Management '95.

[Vas+05] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. Information Systems, 30(7):492–525, 2005.

[VM01] Vaisman, A. A. and Mendelzon, A.O. (2001). A Temporal Query Language for OLAP: Implementation and a Case Study. In Proc. of the 8th BiennialWorkshop on Data Bases and Programming Languages (DBPL), Rome, Italy.

[VS99] Vassilliadis P, Sellis T. A Survey on Logical Models for OLAP Databases. SIGMOD Record 28(4), pp. 64-69, December 1999.

[VSS02] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for ETL processes. In Proc. DOLAP, pages 14–21, 2002.

[VVCM07] Vara, J. M. Vela, B. Cavero, J. M. Marcos, E. Transformación de Modelos para el Desarrollo de Base de Datos Objeto-Relacionales. IEE latin American transactions vol 5. No4, july 2007.

[WK03] The Object Constraint Language, Second Edition. Jos Warmer and Anneke Kleppe. Addison-Wesley, 2003. ISBN: 0-321-17936.

[WK05] Wimmer, Manuel and Kramler, Gerhard. Bridging Grammarware and Modelware. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. Vol. 3844/2006 Satellite Events at the MoDELS 2005 Conference. pg. 59.

[WSS03] Winter, R. & Strauch, B. (2003) A Method for Demand-driven Information Requirements Analysis in Data Warehousing Projects, Journal of Data Warehousing, 8 (1), 38-47.

[XMI] OMG, MOF Meta Object Facility Specification OMG Document 2007-12-01, available at http://www.omg.org/spec/XMI/2.1.1/PDF

[XML] W3C, Extensible Markup Language http://www.w3.org/TR/2004/REC-xml-2004

[ZC06] Zepeda, L., Celma, M. Aplicando MDA al Diseño Conceptual de Almacenes de Datos. 9º Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS'06). Mar del plata, Argentina, 2006.

[Zim06] Zimányi, Esteban. Temporal aggregates and temporal universal quantification in standard SQL, ACM SIGMOD Record, v.35 n.2, p.16-21, June 2006.

[ZPSP97] Zimanyi, E. Parent, C. Spaccapietra, S. and Pirotte, A. TERC+: A Temporal Conceptual Model. In Proc. Int. Symp. on Digital Media Information Base, November 1997.