Hacia la Automatización del Diseño de Data Warehouses

Verónika Peralta
Universidad de la República, Laboratorio CSI
Montevideo, Uruguay, 11.300
vperalta@fing.edu.uy

and

Adriana Marotta
Universidad de la República, Laboratorio CSI
Montevideo, Uruguay, 11.300
adriana@fing.edu.uy

Abstract

A Data Warehouse (DW) is a Database that stores information in order to satisfy decision-making requests. Due to the differences that exist between DWs and operational databases, related to the information they store and to the kind of operations they support, the traditional design techniques for relational databases cannot be applied for relational DWs. This paper addresses the DW logical design problem through a schema transformation approach. We present a DW logical design environment where the DW is built through application of predefined transformations to the source schema, and where we provide a set of rules that embed high-level design strategies and are oriented to the solution of general design problems. These rules determine which transformations must be applied in order to solve each problem, working from information that is provided by the designer. This information is about the DW requirements, their relationship with the source database, and the design criteria the designer wants to apply.

Keywords: Data Warehouse Design, Schema Transformations.

Resumen

Los Data Warehouses (DW) son bases de datos cuyo cometido es dar soporte a la toma de decisiones. Debido a sus diferencias con las bases de datos operacionales, con respecto a la información que almacenan y al tipo de operaciones que se realiza sobre ellas, las técnicas clásicas de diseño de bases de datos relacionales no pueden ser aplicadas para el diseño de DWs relacionales. En este artículo se trata el problema de diseño lógico de DWs con un enfoque de transformación de esquemas. Se presenta un ambiente de diseño lógico de DW donde éste se construye a través de la aplicación de transformaciones predefinidas al esquema fuente, y donde se provee un conjunto de *reglas* que embeben estrategias de diseño de alto nivel, orientadas a encarar globalmente los distintos problemas de diseño. Estas reglas determinan qué transformaciones aplicar para resolver cada problema a partir de información provista por el diseñador. Esta información consiste en los requerimientos del DW, la relación de éstos con las bases de datos fuentes, y los criterios de diseño que desea aplicar.

Palabras claves: Diseño de Data Warehouses, Transformaciones de Esquemas.

1. Introducción

Los Data Warehouses (DW) son bases de datos cuyo cometido es dar soporte a la toma de decisiones. Usualmente la información de estas bases de datos es analizada por el usuario final con una perspectiva multidimensional [11], existiendo herramientas especializadas en facilitar este tipo de análisis (herramientas OLAP). Los DWs tienen características que los distinguen de las bases de datos operacionales, fundamentalmente con respecto a la información que almacenan y con respecto al tipo de operación que se realiza sobre ellos. La información contenida en un DW es resultado de transformaciones de datos provenientes de las bases operacionales. Las operaciones que debe soportar un DW son en general consultas complejas y no operaciones del tipo transaccional (OLTP), ya que el mantenimiento se hace normalmente en forma "batch" y periódicamente.

Debido a las diferencias mencionadas anteriormente las técnicas clásicas de diseño de bases de datos relacionales no pueden ser aplicadas para el diseño de DWs relacionales. Para diseñar este tipo de bases de datos se suele buscar estructuras que optimizan las consultas, teniendo en cuenta el modelo dimensional [12].

En este artículo se trata el problema de diseño lógico de DWs relacionales, trabajando con un enfoque de transformación de esquemas. Algunos de los trabajos existentes en metodologías de diseño de DW basadas en transformaciones [16][3][8] construyen el DW a partir de un esquema conceptual (Entidad-Relación) de la base fuente, y llegan a un modelo dimensional (conceptual o lógico) del DW.

En nuestra propuesta se diseña el esquema lógico del DW mediante transformaciones aplicadas al esquema lógico de la base fuente [14][15]. Con este encare se cubren algunos aspectos que consideramos importantes en cualquier proyecto de construcción de un DW: (a) trazabilidad del diseño, (b) mapeo entre esquema lógico fuente y esquema lógico de DW, y (c) facilidades para diseño de estructuras complejas de DW (como estructuras para manejo de datos históricos, versionamiento de dimensiones, datos calculados, generalización de claves).

El objetivo general de nuestro trabajo es construir un ambiente de diseño semiautomático del esquema lógico de DW. El objetivo de este artículo es presentar un ambiente para el diseño asistido del esquema lógico de DW. Este ambiente se basa en un conjunto de transformaciones de esquema predefinidas [14] [15], las cuales tienen embebidas en su semántica técnicas usuales de diseño de DW. Además de las transformaciones el ambiente provee un conjunto de *reglas* que embeben estrategias de diseño de más alto nivel, orientadas a encarar globalmente los distintos problemas de diseño. Estas reglas determinan qué transformaciones aplicar para resolver cada problema a partir de información provista por el diseñador. Esta información consiste en los requerimientos del DW, la relación de éstos con las bases de datos fuentes, y los criterios de diseño que desea aplicar.

El ambiente permite al diseñador especificar la información de entrada de las reglas mediante tres construcciones: (1) un modelo conceptual multidimensional [4], (2) *mapeos*, los cuales especifican las correspondencias entre los elementos del esquema conceptual y los elementos del esquema fuente, y (3) *lineamientos*, que dan información sobre criterios de diseño relacional de DW.

El uso de reglas de diseño es una técnica clásica en el diseño de bases de datos [20][19][13]. En [20] se proponen reglas para construir un esquema relacional a partir de un esquema entidad relación, cuya aplicación se ordena mediante una secuencia de pasos. El establecimiento de correspondencias o mapeos entre diferentes esquemas también es ampliamente utilizado [19][5][21]. En el contexto de DWs, además de correspondencias de equivalencia se necesita expresar cálculos y funciones sobre los elementos de la base de datos fuente. En [22] presentan una herramienta para deducir y refinar mapeos entre estructuras relacionales, que asocian un valor para un atributo destino calculado a partir de un conjunto de valores de atributos fuentes. Nuestra propuesta incluye mapeos entre objetos conceptuales y estructuras relacionales y maneja diferentes tipos de cálculos en esos mapeos.

La contribución principal de este artículo es la presentación de un ambiente de diseño lógico de DW y la especificación de construcciones que permiten al diseñador: (a) especificar las correspondencias entre el esquema conceptual de DW y el esquema fuente, (b) especificar criterios de diseño lógico a aplicar, y (c) obtener un conjunto de transformaciones a aplicar para resolver los distintos problemas de diseño.

En la sección 2 presentamos el enfoque de transformación de esquemas para diseño de DWs. En la sección 3 proponemos el ambiente de diseño de DWs que incluye los lineamientos de diseño, los mapeos entre el esquema conceptual y el esquema fuente y las reglas. En la sección 4 describimos el escenario de trabajo y en la sección 5 presentamos las conclusiones.

2. Diseño de DW Mediante Transformación de Esquemas

En esta sección comentamos la propuesta para diseño lógico de DW a través de transformación de esquemas, y en particular el conjunto de transformaciones predefinidas. Se puede encontrar una versión extendida en [14] [15].

Se propone un mecanismo donde el esquema de DW es generado por aplicación sucesiva de transformaciones al esquema fuente. Durante el proceso las transformaciones se componen para obtener el esquema objetivo. La Figura 1 muestra la arquitectura básica de la transformación.

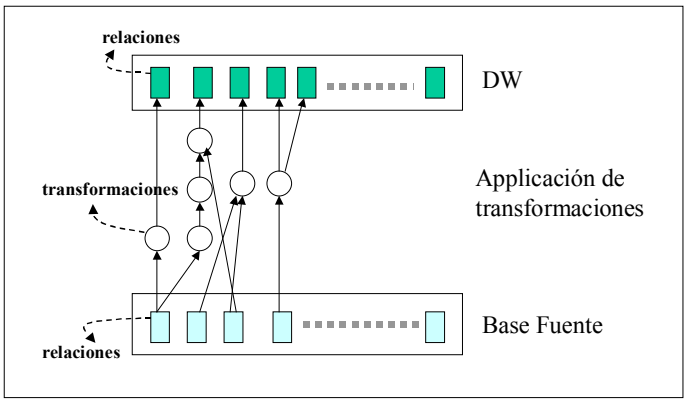

Figura 1 - Arquitectura de la transformación

Se trabaja sobre el Modelo Relacional. A este modelo lo complementamos con una clasificación dimensional de sus elementos (relaciones y atributos) acorde a los conceptos del modelo dimensional definido en [12]. Por ejemplo, tenemos *relaciones de dimensión*, *relaciones de medida (fact table)*, *atributos descriptivos*, *atributos de medida*. Esta clasificación permite a las transformaciones realizar un tratamiento más refinado de las diferentes situaciones de diseño de DW.

También complementamos al modelo con la definición de un conjunto de Invariantes de Esquema. Este es un conjunto de propiedades que un esquema relacional de DW debe satisfacer para ser consistente. Dichas propiedades se refieren a: integridad referencial, jerarquías entre atributos, dependencias entre atributos, etc.

2.1. Las Transformaciones

Las transformaciones propuestas tienen como entrada un sub-esquema y su salida es otro sub-esquema y un esbozo de la transformación de la instancia correspondiente.

Las transformaciones pre-definidas realizan operaciones tales como: partición de una tabla, combinación de tablas, agregado de atributos calculados, y cambios en claves primarias y foráneas. Algunas las agrupamos en familias. Las transformaciones pertenecen a una misma familia cuando resuelven el mismo problema siguiendo distintas alternativas de diseño. Los siguientes son algunos ejemplos que muestran su utilidad.

Muchas relaciones en los sistemas operacionales no mantienen una noción temporal. Por ejemplo, las relaciones de stock suelen tener los datos del stock actual, actualizándolo con cada movimiento de los productos. Sin embargo, en el DW la mayoría de las relaciones necesitan incluir un elemento temporal, de manera que puedan mantener información histórica. Para este propósito existe una transformación llamada *Temporalization* que agrega un atributo de tiempo al conjunto de atributos de una relación.

En los sistemas de producción, usualmente los datos se calculan a partir de otros datos en el momento de las consultas, a pesar de la complejidad de algunas funciones de cálculo, para prevenir cualquier clase de redundancia. Por ejemplo, los precios de productos expresados en dólares son calculados a partir de los precios expresados en otra moneda y una tabla conteniendo el valor del dólar. En un sistema de DW, a veces es conveniente mantener esta clase de datos pre-calculados, por razones de performance. Definimos una familia de transformaciones, cuyo nombre es *DD-Adding*, que agrega a una relación un atributo derivado a partir de otros.

La Figura 2 muestra la lista de transformaciones predefinidas propuestas y la Figura 3 muestra la especificación de una de las transformaciones.

T1	Identity	T8	Hierarchy Roll Up
T2	Data Filter	T9	Aggregate Generation
T3	Temporalization	T10	Data Array Creation
T4	Key Generalization (family)	T11	Partition by Stability (family)
T5	Foreign Key Update	T12	Hierarchy Generation (family)
T6	DD-Adding (family)	T13	Minidimension Break off
T7	Attribute Adding	T14	New Dimension Crossing

Figura 2 – Las transformaciones

T6.3 – DD-Adding N-N

Description:

This transformation adds to a relation an attribute that is derived from an attribute of another relation. In this case the calculation function works over a set of tuples of the other relation. This set is obtained through a join operation between the two relations.

Input:

- source schema : $R_1(A_1,...,A_n)$, $R_2(B_1,...,B_m) \in Rel$
- e(B)/ B ∈ $\{B_1,...,B_m\}$, where e(B) is an aggregate expression over the attribute B
- X / X ⊂ $Att_D(R_2)$, attributes by which we want to group
- C(Y,Z) / Y ⊆ $\{A_1,...,A_n\}$ ∧ Z ⊆ $\{B_1,...,B_m\}$, join condition
- source instance : r_1, r_2

Resulting schema:

$R'_1(A_1,...,A_n,A_{n+1}) \in Rel$ / A_{n+1} represents e(B) in R_2

Generated instance:

- r'_1 = select $A_1,...,A_n$,e(B)
  from R_1 R_2
  where C(Y,Z)
  group by A_1, ..., A_n, X

Figura 3 - Especificación de la transformación T6.3

La entrada de la transformación está compuesta por un subesquema (2 relaciones), una expresión de resumen (agregación) para un atributo, un conjunto de atributos por los que agrupar, una condición de join y la instancia del subesquema. *Rel* denota al conjunto de todas las relaciones y $Att_D(R)$ denota al conjunto de los atributos descriptivos de la relación R.

En la Figura 4 se muestra un ejemplo de la aplicación de la transformación *DD_Adding N-N*. Esta transformación se aplica a las tablas Clientes y Cuentas y da como resultado la tabla Clientes-DW. En la tabla Clientes-DW se agregó el atributo Monto, el cual contendrá para cada cliente la totalización de los montos de todas sus cuentas.

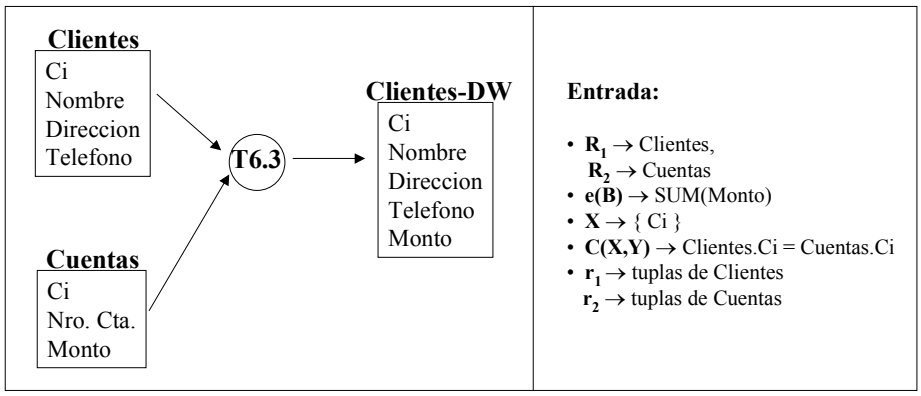

Figura 4 – Aplicación de la transformación DD-Adding N-N

3. Ambiente de Ayuda Para la Aplicación de las Transformaciones

El diseño lógico del DW se realiza aplicando las transformaciones de esquemas, pero la decisión de qué transformaciones aplicar, y en qué orden hacerlo dependerá de los requerimientos, de las bases de datos fuente y de los criterios de diseño del propio diseñador. Los requerimientos son modelados usando un modelo conceptual multidimensional [4], que representa los objetos del negocio en términos de dimensiones y cruzamientos entre ellas. Los diferentes criterios de diseño para el esquema lógico relacional pueden conducir a esquemas más o menos normalizados [16] y fragmentados [7]. Estos criterios de diseño tienen un papel fundamental en la elección de las transformaciones a aplicar. Por otro lado, dado que las transformaciones se aplican al esquema fuente, es necesario vincular a éste con el esquema conceptual, indicando donde se encuentran en las fuentes los diferentes conceptos multidimensionales. En nuestra propuesta permitimos al diseñador especificar esta información mediante un *esquema conceptual*, *lineamientos, y mapeos*. El diseñador es guiado en la elección de las transformaciones a aplicar mediante un conjunto de *reglas*.

3.1. Lineamientos y Mapeos

En esta sección presentamos los lineamientos y los mapeos valiéndonos de un ejemplo para su ilustración.

El ejemplo trata de una empresa que brinda atención telefónica a sus clientes y quiere contabilizar el tiempo invertido en atención de llamadas por mes y por cliente, y totales anuales, tanto por ciudad como por departamento (división política del país) de residencia del cliente.

La Figura 5 muestra un esquema multidimensional para el problema, modelado con CMDM [4]. Se identifican tres dimensiones: *clientes, fechas* y *llamadas* (medida). La dimensión clientes tiene 3 niveles: *departamento, ciudad* y *cliente*; y la dimensión fechas tiene 2 niveles: *año* y *mes*. Para resolver los requerimientos se define la relación dimensional *atención*, que cruza *clientes y fechas*, tomando *llamadas* como medida.

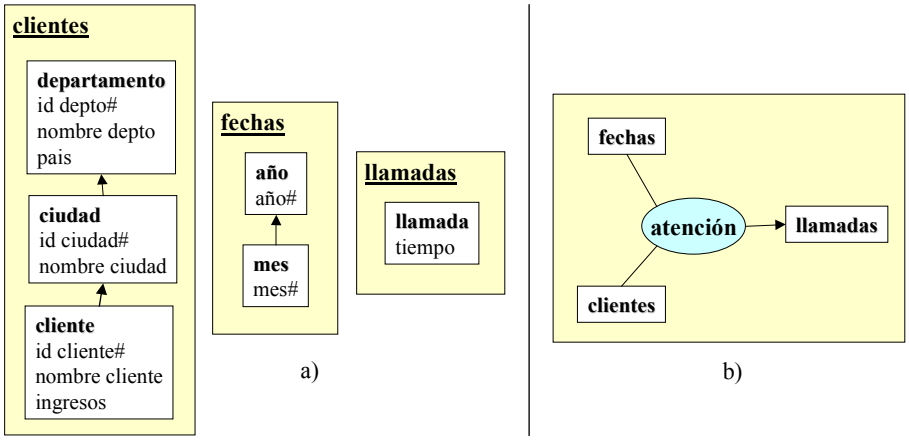

Figura 5 – Esquema conceptual: a) dimensiones, y b) relación dimensional

En a) los cuadros de texto son los niveles de la dimensión; sus nombres aparecen en negrita y debajo aparecen los atributos llamados *ítems*. Los ítems seguidos por un numeral (#) identifican al nivel. Las jerarquías de niveles se representan por flechas entre los niveles. En b) el óvalo central indica el nombre de la relación, las líneas indican las dimensiones que se cruzan y la flecha indica la medida.

Se quiere construir un esquema relacional a partir del esquema conceptual. Se tienen varias alternativas para definir tanto las tablas de dimensión como las tablas de medidas.

Por ejemplo, para la dimensión *clientes*, se puede optar por almacenar cada nivel en una tabla, con los ítems del nivel y la clave del nivel superior en la jerarquía para poder realizar el join (Figura 6a). Otra opción sería mantener una única tabla para la dimensión que contenga todos los ítems (Figura 6b), o seguir estrategias intermedias. El diseñador debe decidir qué niveles almacenar juntos en una misma tabla, basándose en criterios de performance, redundancia y ocupación de disco.

De la misma forma se debe definir qué cruzamientos (llamados cubos [4]) se van a implementar, obteniendo tablas de medidas. El cubo con más detalle para la relación *atención* es por *mes* y por *cliente* (Figura 6c) pero también se puede querer almacenar totales, por ejemplo, por *ciudad* y *año* (Figura 6d). El diseñador debe decidir qué cubos materializar, tratando de lograr un equilibrio entre performance y espacio de almacenamiento.

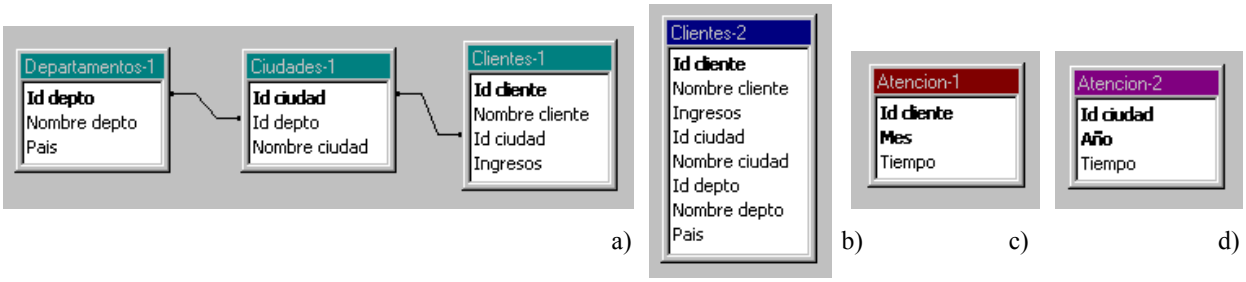

Figura 6 – Diferentes estrategias de generación de estructuras relacionales

Los cuadros de texto representan tablas en el modelo relacional. En la parte superior aparece el nombre de la tabla, y debajo aparecen los atributos. Los atributos en negrita conforman la clave primaria y las líneas entre tablas indican los atributos por los que debe realizarse el join (igualando atributos).

Las tablas de medidas se pueden fragmentar horizontalmente [17] de manera de mantener tablas con menos registros que mejoren la performance de las consultas de los usuarios. Se puede, por ejemplo, tener una tabla con los datos de los últimos dos años y tener los datos históricos en otra tabla. El diseñador debe decidir mediante qué criterios fragmentar horizontalmente las tablas de medidas.

Estos tipos de decisiones que debe tomar el diseñador se especifican mediante los *lineamientos*. Se trata de información que complementa al esquema conceptual con estrategias de diseño. Al definir los lineamientos el diseñador está dando pautas de alto nivel, de cómo debe ser el esquema lógico del DW.

Una vez definidos los lineamientos, se debe relacionar el esquema conceptual con la base de datos fuente. La Figura 7 muestra una posible base de datos de la empresa con tres maestros: de clientes, ciudades y departamentos, una tabla con los sueldos de los clientes y una tabla donde se registran los llamados de los clientes.

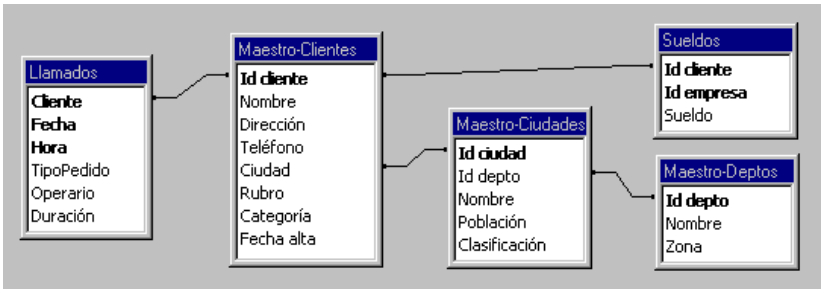

Figura 7 – Base fuente

Dadas dos tablas interesa reflejar cómo se vinculan, es decir, cómo se debe realizar el join entre ellas. A un vínculo de este tipo se le llama *link*. La vinculación entre tablas puede expresarse mediante un grafo, donde los nodos son las tablas y las aristas son los links rotulados con predicados de join. Gráficamente se omiten los rótulos para los predicados de igualdad de atributos (mayoría de los casos) y se dibujan líneas que unen los atributos involucrados. En la Figura 7 puede visualizarse la vinculación entre tablas.

El diseñador debe indicar de qué datos fuentes (tablas y atributos) se tomarán los datos para cargar el DW. Para ello indica como se corresponden los ítems del esquema conceptual con los atributos de la base fuente. Esto se especifica mediante los *mapeos*. Por ejemplo, puede indicar que los ítems del nivel ciudad se mapean a los atributos *Id ciudad* y *Nombre* de la tabla *Maestro-Ciudades*, y los ítems *mes* y *año* se calculan a partir del atributo *Fecha* de la tabla *Llamados*.

En las siguientes secciones se formalizan estos conceptos.

3.1.1. Lineamientos de Diseño

A través de los lineamientos el diseñador define el estilo de diseño para el DW (snowflakes, estrella, alguna estrategia intermedia [16]) e indica requerimientos de performance y almacenamiento.

Existe un conjunto de lineamientos que el diseñador debe especificar. Este contiene lineamientos para *Materialización de Cubos*, *Fragmentación Vertical de Dimensiones* y *Fragmentación Horizontal de Cubos.* A continuación se presenta la especificación de un lineamiento, el conjunto completo puede consultarse en [18].

Fragmentación Vertical de Dimensiones

Mediante este lineamiento, el diseñador puede indicar el grado de normalización que quiere lograr al generar estructuras relacionales para cada dimensión. Por ejemplo, le puede interesar un esquema estrella, es decir, denormalizar todas las dimensiones. Por el contrario, le puede interesar un esquema de snowflakes, normalizando todas las dimensiones [16]. También le puede interesar tratar diferente cada dimensión, indicando para cada una si normaliza, denormaliza o efectúa una estrategia intermedia, indicando en este último caso, qué niveles quedan en una misma tabla.

Para especificar este lineamiento, el diseñador debe indicar para cada dimensión qué niveles desea almacenar juntos. Para ello define por extensión una función que a cada dimensión le hace corresponder una fragmentación de sus niveles. A esa función se le llama SchDFragmentation.

Para que una fragmentación tenga sentido los niveles de cada fragmento deben estar relacionados jerárquicamente. Es decir, si se considera el grafo que representa a la jerarquía de la dimensión, el sub-grafo inducido por los niveles del fragmento debe ser conexo. Si dos niveles de un mismo fragmento no están relacionados, ni directa ni transitivamente, entonces conforman un cruzamiento y se pierde la relación jerárquica de la dimensión.

La fragmentación además debe ser completa, es decir que todos los niveles deben estar en al menos un fragmento para no perder información. Si no se quiere duplicar el almacenamiento de información de cada nivel, los fragmentos deben ser disjuntos, pero no es una exigencia. El diseñador decide cuando duplicar información de

acuerdo a su estrategia de diseño.

En la Definición 1 se define primero un *fragmento* de una dimensión (Fragments(D)), como un sub-conjunto del conjunto de los niveles de la dimensión (D.L), los cuales conforman un sub-grafo conexo de su jerarquía (D.Po es el orden parcial de niveles de la dimensión , conformando sus jerarquías [4]). Luego se define un *conjunto completo de fragmentos* (CompleteFragmentSets (D)) como un conjunto de fragmentos en el que todo nivel de la dimensión (D.L) está en algún fragmento. Por último se define una *fragmentación de dimensiones* (SchDFragmentation) como una función que a cada dimensión le hace corresponder un conjunto completo de fragmentos (SchDimensions es el conjunto de dimensiones del esquema conceptual [18]).

Gráficamente, una fragmentación se representa como una coloración de niveles. Los niveles de un mismo fragmento se recuadran con el mismo color. La Figura 8 muestra la representación gráfica de dos fragmentos para la dimensión clientes. El primer fragmento incluye los niveles *departamento* y *ciudad* (línea corrida), el segundo incluye al nivel *cliente* (línea punteada).

- FRAGMENTS (D) ≡ {F / F ⊆ D.L ∧ ∀A,B ∈ F . (
  <A,B> ∈ D.PO ∨
  <B,A> ∈ D.PO ∨
  ∃ C ∈ F . (<A,C> ∈ D.PO ∧ <B,C> ∈ D.PO)) }
- COMPLETEFRAGMENTSETS (D) ≡ {FS / FS ⊆ FRAGMENTS (D) ∧
  ∀L ∈ D.L . (∃ F ∈ FS . (L ∈ F)) }
- SCHDFRAGMENTATION ∈ {D / D ∈ SCHDIMENSIONS} →
  COMPLETEFRAGMENTSETS(D) }

Definición 1 – SchDFragmentation

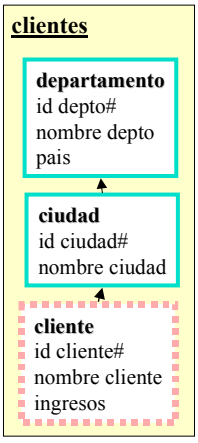

Figura 8 – Fragmentación de la dimensión *clientes*

3.1.2. Mapeos entre el Esquema Conceptual y la Base de Datos Fuente

Los mapeos indican dónde se encuentran en el esquema lógico de la fuente los diferentes elementos del esquema conceptual y los lineamientos. Son funciones que asocian a cada elemento del esquema conceptual una expresión construida en base a las tablas y atributos de la base de datos fuente. Son definidas por el diseñador en forma explícita.

Antes de definir las funciones de mapeo se presenta una caracterización de las expresiones involucradas.

Expresiones de Mapeo

En una función de mapeo, a cada ítem del esquema conceptual le corresponde una expresión construida en base a atributos de las tablas fuentes, a la que se llama expresión de mapeo (*mapexpr*).

Una expresión de mapeo puede ser un atributo de una tabla fuente (DirectME), o un cálculo que involucra varios atributos de una tupla (1calcME), o un resumen o totalización que involucra varios atributos de varias tuplas (NcalcME) o un valor externo a las fuentes como una constante, una marca de tiempo o dígitos de versión (ExternME).

A modo de ejemplo, la Definición 2 presenta la definición de las expresiones de totalización (NcalcME), las restantes definiciones pueden consultarse en [18]. Una expresión de totalización es una tripla que contiene: una tabla fuente: Tab (SchTables es el conjunto de tablas de la fuente), un conjunto de atributos de la tabla que intervienen en el cálculo: Patts (Tab.As son los atributos de la tabla TAB) y una expresión de cálculo: Expr (RollUpExpressions(Patts) es el conjunto de expresiones tipadas de resumen que se pueden construir con los elementos del conjunto Patts [18])

- NCALCME ≡ { <EXPR, TAB, PATTS > / TAB ∈ SCHTABLES
  ∧ PATTS ⊆ TAB.AS ∧ EXPR ∈ ROLLUPEXPRESSIONS(PATTS) }

Definición 2 – NcalcME

El conjunto MAPEXPR es la unión de los distintos tipos de expresiones.

Funciones de Mapeo

Definido MapExpr, se pueden definir funciones de mapeo (Definición 3). Una función de mapeo, definida para un conjunto de ítems, le hace corresponder a cada ítem, una expresión de mapeo, controlando que coincidan los tipos de los ítems (I.Type) y de la expresión (F(I).Expr.Type).

- MAPPINGS(ITS) ≡ { F / F ∈ ITS → MAPEXPRS ∧ ∀ I ∈ ITS. (I.TYPE = F(I).EXPR.TYPE) }

Definición 3 – Mappings

Las funciones de mapeo se utilizan en 2 contextos: para vincular fragmentos de dimensiones a las tablas fuentes (mapeos de fragmentos), y vincular cubos a las tablas fuentes (mapeos de cubos). Esta vinculación se realiza definiendo una función de mapeo que haga corresponder los ítems del fragmento o cubo con las tablas fuentes.

Gráficamente una función de mapeo se representa por flechas de los ítems del esquema conceptual a los atributos de las tablas fuentes. Cuando el mapeo es directo se representa con una línea corrida, cuando es un cálculo se representa con una línea cortada a cada atributo que interviene en el cálculo y se adjunta la definición del cálculo, y cuando es externo no se utilizan líneas pero se adjunta la expresión a la que mapea. En la Figura 9 se muestra una función de mapeo para una dimensión *clientes*, con un único fragmento. El ítem *país* tiene un mapeo externo de tipo constante, el ítem *ingresos* tiene un mapeo calculado en base al atributo *sueldo* de la tabla *sueldos*. Los demás ítems tienen mapeos directos.

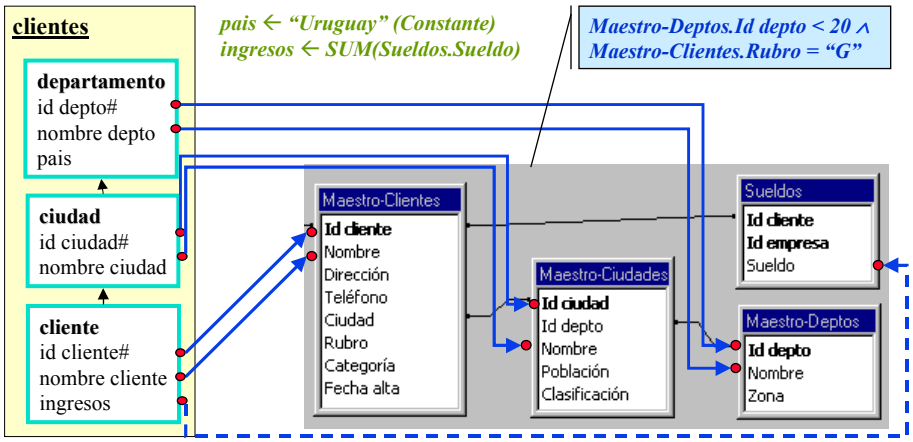

Figura 9 – Representación gráfica de una función de mapeo

Cada función de mapeo representa una vista o expresión SQL que tiene en el cabezal (SELECT) las expresiones de mapeo y obtiene los datos (FROM) de las tablas referenciadas en dichas expresiones, imponiendo los links entre las tablas como condición de join (WHERE) y agrupando por las expresiones que no son de agregación. Las tablas involucradas deben tener links definidos, no tienen sentido los productos cartesianos.

En el ejemplo de la Figura 9, se asocia al fragmento de la dimensión *clientes* la siguiente vista SQL:

```
SELECT  Maestro-Deptos.Id_depto as id_depto,
        Maestro-Deptos.Nombre as nombre_depto,
        "Uruguay" as pais,
        Maestro-Ciudades.Id_ciudad as id_ciudad,
        Maestro-Ciudades.Nombre as nombre_ciudad,
        Maestro-Clientes.Id_cliente as id_cliente,
        Maestro-Clientes.Nombre as nombre_cliente,
        SUM(Sueldos.Sueldo) as ingresos,
FROM Maestro-Deptos, Maestro-Ciudades, Maestro-Clientes, Sueldos
WHERE   Maestro-Clientes.Ciudad = Maestro-Ciudades.Id_ciudad
AND     Maestro-Ciudades.Id_depto = Maestro-Deptos.Id_depto
AND     Maestro-Clientes.Id_cliente = Sueldos.Id_cliente
GROUP BY Maestro-Deptos.Id_depto, Maestro-Deptos.Nombre, Maestro-Ciudades.Id_ciudad,
        Maestro-Ciudades.Nombre, Maestro-Clientes.Id_cliente, Maestro-Clientes.Nombre
```

En la vinculación de un fragmento o un cubo pueden existir condiciones, por ejemplo que los valores de un atributo se encuentren en determinado rango. Estas condiciones pueden deberse a restricciones en el esquema conceptual o a restricciones que deseen aplicarse a las fuentes.

En el ejemplo de la Figura 9, el nivel *departamento* puede tener una restricción indicando que sólo interesan los departamentos con id menor a 20, que pueden ser por ejemplo, los departamentos uruguayos. Esta es una restricción del esquema conceptual. También puede ocurrir que en la tabla *Maestro-Clientes* se almacenen clientes de varios sistemas o secciones, y que para el DW sólo interesen los de un determinado rubro (por ejemplo: G - Gastronomía). Esta es una restricción respecto a las fuentes.

Ambas restricciones deben tenerse en cuenta al establecer los mapeos, e incorporar una condición tipo:

$$Maestro\text{-}Deptos.Id\_depto < 20 \wedge Maestro\text{-}Clientes.Rubro = \text{“}G\text{”}$$

Estas restricciones pueden verse como condiciones adicionales de la vista SQL. Para el ejemplo anterior se agregaría a la expresión:

```
AND Maestro-Deptos.Id_depto < 20
AND Maestro-Clientes.Rubro = "G"
```

Gráficamente las condiciones se representan como llamadas (callouts). La Figura 9 muestra las condiciones anteriores. Formalmente, las condiciones son predicados sobre atributos de las tablas fuentes [18].

Para mapear un fragmento debe definirse una función de mapeo y opcionalmente una condición. Para mapear un cubo, además de una función de mapeo deben definirse otros elementos, como las operaciones de roll-up de las medidas. Omitimos estas definiciones por restricciones de espacio de este artículo; las mismas pueden ser consultadas en [18].

3.2. Reglas

Para construir manualmente y paso a paso un DW el diseñador toma una serie de decisiones, y en cada paso va transformando el esquema relacional del DW. Para ello se basa en características que observa en los lineamientos y mapeos; es decir, cuando se cumple determinada característica o condición, realiza cierta transformación en el DW.

Proponemos un conjunto de reglas de diseño, cuyo objetivo es automatizar esa serie de decisiones, sustituyendo las decisiones del diseñador. Las reglas controlan que se verifiquen una serie de condiciones, y determinan que transformaciones deben aplicarse al esquema.

La Figura 10 muestra una tabla con las reglas y un breve comentario de su utilidad. Las reglas marcadas con * representan familias de reglas orientadas a resolver un mismo problema.

	Regla	Descripción
R1	Join	Combina las dos tablas generando una nueva tabla. Se usa cuando un fragmento o cubo mapea a más de una tabla.
R2	Rename	Renombra los atributos mapeados en forma directa, utilizando los nombres de los ítems. Se usa cuando los nombres de los ítems no coinciden con los de los atributos a los que mapean.
R3	Calculate *	Agrega a una tabla un atributo que materializa un cálculo. Se usa cuando un ítem mapea a una expresión de cálculo.
R4	Extern *	Agrega a la tabla un atributo que materializa un valor externo. Se usa cuando un ítem mapea a una expresión externa.
R5	Group	Elimina de una tabla los atributos no mapeados por ningún ítem, agrupando por los demás y resumiendo (roll-up) las medidas. Se usa para eliminar atributos no mapeados.
R6	Drill Up *	Reduce el nivel de detalle de una tabla de medidas, realizando un drill-up respecto a una dimensión.
R7	Primary Key	Modifica la clave primaria de una tabla. Se usa para hacer coincidir los ítems clave de un fragmento o cubo, con los atributos que los mapean.
R8	Filter	Elimina de una tabla las tuplas que no cumplan una condición. Se usa para imponer condiciones en los mapeos, y para fragmentar horizontalmente un cubo.

Figura 10 – Reglas de diseño

A continuación presentamos a modo de ejemplo, la especificación de la regla 3.2 – Aggregate Calculate.

El objetivo de esta regla es materializar un cálculo. Se utiliza cuando se cumplen las siguientes condiciones: (i) un ítem de un fragmento o cubo se corresponde (por la función de mapeo) con una expresión de mapeo de tipo resumen (NcalcME), y (ii) las expresiones de mapeo de tipo directo o cálculo simple (directME y 1calcME) que se corresponden con los otros ítems, involucran una única tabla.

Esta regla da como resultado la aplicación de la transformación T6.3 – DD-Adding N-N ilustrada en la sección 2.1.

Ejemplo:

La Figura 11a muestra la dimensión *clientes* (con un único fragmento) y su función mapeo a las tablas *Clientes* y *Sueldos*. Al ítem *ingresos* le corresponde una expresión de mapeo de tipo NcalcME.

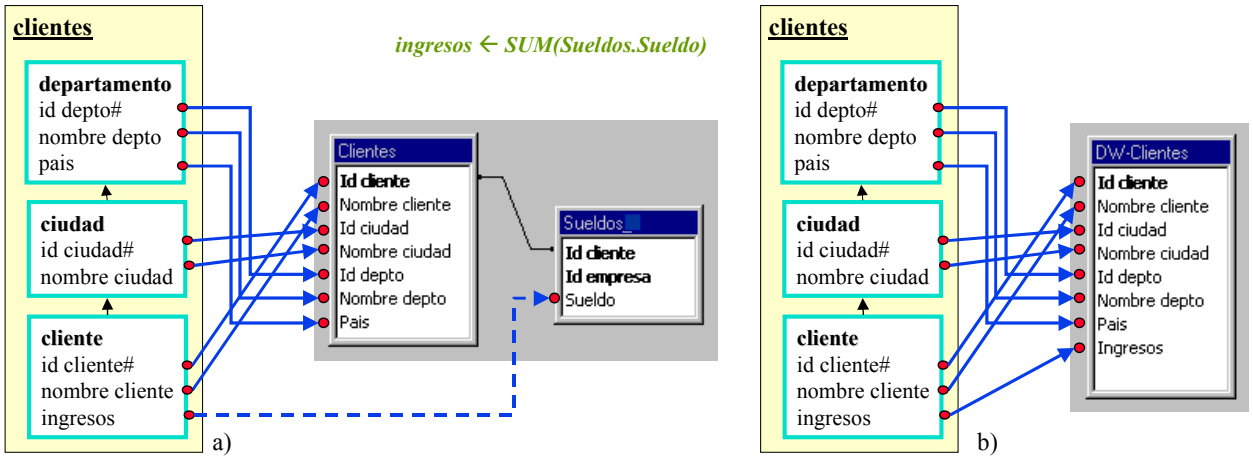

Figura 11 – Función de mapeo del fragmento de la dimensión clientes: a) antes y, b) después de aplicar la regla

Se calcula un nuevo atributo, de nombre *Ingresos,* como *SUM(Sueldos.Sueldo)*. Esto es, se construye una nueva tabla (*DW-Clientes*) agregando dicho atributo a los atributos de la tabla *Clientes*.

La aplicación de la regla actualiza la función de mapeo para que referencie a los atributos de la nueva tabla cuando corresponda. La Figura 11b muestra la función de mapeo actualizada.

Especificación:

La especificación de una regla consta de 5 secciones: *Description, Objects, Input, Conditions* y *Result*. *Description* es una explicación en lenguaje natural del comportamiento de la regla. *Objects* enumera los objetos del esquema conceptual y lineamientos a los que se puede aplicar la regla. *Input* enumera los parámetros de entrada a la regla, estos son: funciones de mapeo de los objetos y tablas del esquema lógico. *Conditions* es un conjunto de predicados que se deben cumplir para poder aplicar la regla. Finalmente, *Result* es la salida de la regla que consta de tablas resultado de aplicar transformaciones a las tablas de entrada, y funciones de mapeo actualizadas para reflejar la aplicación de la regla.

La Figura 12 muestra la especificación de la regla R3.2. Como objetos se tienen un fragmento o cubo (*SchFragments* y *SchCubes* son los conjuntos de todos los fragmentos y cubos del esquema) y uno de sus ítems (la función *ObjectItems* devuelve todos los ítems que lo conforman). Como entrada se tiene la función de mapeo del objeto (el fragmento o cubo) y una tabla de las involucradas en las expresiones de mapeo de sus ítems (devueltas por la función *MapTables*). Se especifican las condiciones ya mencionadas (ítem con mapeo calculado, y una única tabla involucrada en las expresiones de mapeo).

La tabla resultado, es la que se obtiene de aplicar la transformación T6.3 (descrita en la sección 2.1). Cómo parámetros de la transformación se tienen:

- Las tablas T (la tabla de entrada) y F(I).Tab (la tabla involucrada en la expresión de mapeo).
- La función de cálculo para el nuevo atributo: F(I).Expr (el campo expr de la expresión de mapeo). El nombre del atributo coincidirá con el nombre del ítem (I.ItemName).
- El parámetro de atributos adicionales de la transformación no se utiliza.
- Como función de join entre las tablas se usa el predicado de link definido entre las mismas (TabLink (T, F(I).Tab).Cond).
- Las instancias de las tablas: Instance(T) y Instance(F(I).Tab)

El procedimiento *UpdateTabs* define los links para la nueva tabla. Las funciones *ReplaceMap*, *ReplaceTable* y *UpdateMaps* actualizan la función de mapeo como resultado de aplicar la regla. La función de mapeo debe utilizar la nueva tabla en las expresiones de mapeo, y el ítem al que se aplicó la regla debe corresponderse con el nuevo atributo.

RULE R3.2 – AGGREGATE CALCULATE

Description:

Dado un objeto del esquema conceptual más lineamientos, uno de sus ítems, su función de mapeo (con mapeo NcalcME para el ítem) y una tabla que lo mapea, se genera una nueva tabla agregando el atributo calculado.

Objects:

- OS ∈ SchFragments ∪ SchCubes */* un objeto: fragmento o cubo */*
- I ∈ ObjectItems(OS) */* un ítem del objeto */*

Input:

- **Maps:** F ∈ Mappings(ObjectItems(OS)) */* función de mapeo del objeto */*
- **Tables:** T ∈ MapTables(F, ObjectItems(OS)) */* tabla que mapea al objeto */*

Conditions:

- F(I) ∈ NcalcME */* expresión de mapeo de tipo resumen */*
- # MapTables(F, ObjectItems(OS)) = 1 */* el objeto es mapeado por una sóla tabla */*

Result:

- **Tables:**
  - T' = Transformation T6.3 (*/* DD-Adding N-N */*
    {T, F(I).Tab}, */* esquema fuente */*
    I.ItemName = F(I).Expr, */* función de cálculo */*
    {}, */* atributos adicionales de agrupamiento */*
    TabLink (T, F(I).Tab).Cond, */* función de join */*
    {Instance(T), Instance(F(I).Tab)} */* instancias */*
)
  - UpdateTabs (T', {T}) */* con links a T por su clave, y hereda los links de T */*
- **Maps:**
  - F' = ReplaceMap (F, F(I), <I, T', T'•I, F(I).Cond)}) */* ahora con mapeo DirectME*/*
  - F'' = ReplaceTable (F',T, T')
  - UpdateMaps (F'', OS)

Figura 12 – Especificación de una regla

4. Escenario de trabajo propuesto

Las diferentes metodologías de diseño de DWs [9][12][3][16][8] dividen el problema global de diseño en sub-problemas más pequeños y proponen algoritmos (o simplemente secuencias de pasos) para ordenar y atacar esos sub-problemas.

El escenario propuesto consiste en utilizar las reglas para la resolución de esos problemas concretos. En cada paso se aplicarían diferentes reglas, y podría intercalarse la aplicación de alguna transformación aislada. La incorporación de nuevos problemas a resolver puede demandar la definición de nuevas reglas, nuevos lineamientos e incluso nuevas transformaciones.

En este escenario el diseñador de DW trabaja de la siguiente manera. Comienza por definir un esquema conceptual multidimensional (en CMDM). Luego define los lineamientos y los mapeos. A continuación aplica las reglas que le dicen qué transformaciones debe aplicar al esquema fuente, y lo complementa si es necesario, con la aplicación de otras transformaciones.

Este escenario constituye un paso importante hacia la automatización del proceso de diseño. Actualmente se está trabajando en la definición de un algoritmo que engloba los pasos de metodologías existentes y genera automáticamente el esquema lógico del DW eligiendo el orden de aplicación de las reglas.

5. Conclusiones

Este artículo presenta un ambiente de ayuda para el diseño de DWs relacionales. En este ambiente el DW es construido mediante aplicación sucesiva de transformaciones al esquema fuente. Como ayuda en la automatización de las tareas del diseñador, se proveen reglas que dan como resultado la aplicación de las transformaciones adecuadas para los distintos problemas de diseño teniendo en cuenta información de entrada. Cada regla está orientada a resolver un problema particular, cubriendo los problemas más frecuentes en diseño de DWs relacionales. La información de entrada, que debe dar el diseñador, se especifica mediante el esquema conceptual del DW, lineamientos de diseño y mapeos entre dicho esquema conceptual y el esquema de la base fuente.

Con este trabajo apuntamos a obtener un mecanismo semiautomático de diseño lógico de DWs, donde a partir de las especificaciones del diseñador, mencionadas anteriormente, se genere el esquema de DW. Este mecanismo tendría la importante ventaja de permitir aplicar distintos criterios de diseño, por medio de la especificación de los lineamientos. Pero a la vez se podrían definir estrategias por defecto que automaticen la definición de los lineamientos (por ejemplo, una estrategia posible es construir siempre un esquema estrella denormalizando todas las dimensiones). Podrían estudiarse funciones de costos y heurísticas que automaticen la definición de lineamientos de acuerdo a distintos factores: limitaciones de espacio físico, performance en las consultas, afinidad, cantidad de datos.

Se ha desarrollado un prototipo de una herramienta CASE que soporta la generación del esquema relacional del DW a partir del esquema conceptual [18]. El prototipo incluye una interfaz para la definición de lineamientos, una interfaz para la definición de mapeos, y la aplicación automática de las reglas para la generación del esquema lógico.

Referencias

[1] Adamson, C.; Venerable, M.: “Data Warehouse Design Solutions”. J. Wiley & Sons, Inc.1998.

[2] Alcarraz, A.; Ayala, M.; Gatto, P.: “Diseño e implementación de una herramienta para evolución de un Data Warehouse Relacional”. Undergraduate project. InCo, Universidad de la República, Uruguay, 2001.

[3] Ballard, C.; Herreman, D.; Schau, D.; Bell, R.; Kim, E.; Valncic, A.: “Data Modeling Techniques for Data Warehousing”. SG24-2238-00. IBM Red Book. ISBN number 0738402451. 1998.

[4] Carpani, F. Ruggia, R.: “An Integrity Constraints Language for a Conceptual Multidimensional Data Model”. SEKE’01, Argentina, 2001.

[5] Fankhauser, P.: “A Methodology for Knowledge-Based Schema Integration”. PhD-Thesis, Technical University of Vienna, 1997.

[6] Garbusi, P.; Piedrabuena, F.; Vázquez, G.: “Diseño e Implementación de una Herramienta de ayuda en el Diseño de un Data Warehouse Relacional”. Undergraduate project. InCo, Universidad de la República, Uruguay, 2000.

[7] Golfarelli, M. Maio, D. Rizzi, S.:”Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases”. DAWAK’00, UK, 2000.

[8] Golfarelli, M. Rizzi, S.:”Methodological Framework for Data Warehouse Design.", DOLAP’98, USA, 1998.

[9] Inmon, W.: “Building the Data Warehouse”. John Wiley & Sons, Inc. 1996.

[10] Inmon, W.: “Building the Operational Data Store”. John Wiley & Sons, Inc. 1996.

[11] Kenan Technologies:"An Introduction to Multidimensional Databases”. White Paper, Kenan Technologies, 1996.

[12] Kimball, R.:"The Datawarehouse Toolkit". John Wiley & Son, Inc., 1996.

[13] Lerner, B. Habermann, A.: "Beyond Schema Evolution to Database Reorganization”. OOPSLA/ECOOP, 1990.

[14] Marotta, A.: “Data Warehouse Design and Maintenance through Schema Transformations”. Master Thesis. InCo - Pedeciba, Universidad de la República, Uruguay, 2000.

[15] Marotta, A. Ruggia, R.: “Data Warehouse Design: A schema-transformation approach”. Accepted paper in SCCC’2002. Chile. 2002.

[16] Moody, D.; Kortnik, M.: “From Enterprise Models to Dimensionals Models: A Methodology for Data Warehouse and Data Mart Design”. DMDW’00, Sweden, 2000.

[17] Ozsu, M.T. Valduriez, P.: “Principles of Distributed Database Systems”. Prentice-Hall Int. Editors. 1991.

[18] Peralta, V.: “Diseño lógico de Data Warehouses a partir de Esquemas Conceptuales Multidimensionales”. Master Thesis. InCo - Pedeciba, Universidad de la República, Uruguay. 2001.

[19] Spaccapietra, S. Parent, C.: “View integration: A step forward in solving structural conflicts”. TKDE’94, Vol 6, No. 2, 1994.

[20] Teorey, T. Yang, D. Fry, J.: “A logical design methodology for relational databases using: the extended entity-relationship model”, Cornput&Surveys 18,2. June 1986.

[21] Vidal, V. Lóscio, B. Salgado, A.: “Using correspondence assertions for specifying the semantics of XML-based mediators”. WIIW’2001, Brazil, 2001.

[22] Yang, L. Miller, R. Haas, L. Fagin, R.: “Data-Driven Understanding and Refinement of Schema Mappings”. ACM SIGMOD’01, USA, 2001.