Una plataforma basada en Metadata para Cálculo de Vistas en Sistemas de Información Multi-Fuentes

Verónika Peralta

Instituto de Computación - Facultad de Ingeniería - Universidad de la República[1]
Montevideo, Uruguay
vperalta@fing.edu.uy

Resumen: Un Sistema de Información Multi-Fuente (MSIS) se compone de un conjunto de fuentes de datos independientes y un conjunto de vistas o consultas que definen los requerimientos de los usuarios. Sus diferencias con los sistemas de información clásicos introducen nuevas actividades de diseño y motiva el desarrollo de nuevas técnicas.

En este artículo estudiamos un caso particular de un MSIS: un Data Warehouse (DW) y proponemos un meta-modelo para representar su metadata desde dos puntos de vistas: la representación de los esquemas y las relaciones inter-esquema que permiten calcular una vista a partir de los datos fuentes.

El meta-modelo es el centro de una plataforma general para desarrollo de MSIS. La plataforma permite la fácil integración de herramientas de diseño y mantenimiento a través de un modelo de datos común que centraliza el flujo de datos y las rutinas de control de integridad entre las herramientas.

Palabras clave: Bases de datos, Data Warehousing, Metadata.

1. Introducción

La necesidad de acceder a información disponible en varias fuentes de datos de una forma unificada es cada día más fuerte y generalizada. Los requerimientos de información de los usuarios, generalmente en forma de vistas o consultas, necesitan datos de diversas fuentes, posiblemente heterogéneas y autónomas, con datos en diferentes formatos.

Un Sistema de Información Multi-Fuente (Multi-Source Information System: MSIS) es un sistema de información que accede a varias bases de datos independientes para resolver requerimientos de usuarios sobre sus datos. Como ejemplos de MSIS podemos citar los sistemas Web, donde los datos son extraídos de diferentes páginas web, integrados y presentados al usuario. La arquitectura de wrappers y mediadores [28] se utiliza comunmente para realizar dichas tareas. Los sistemas de Data Warehousing y sistemas Olap [13][1] también extraen, transforman e integran información de varias fuentes de datos, posiblemente heterogéneas, y la dejan disponible para análisis estratégico. Otros ejemplos de MSIS son las federaciones [26] que permiten el acceso a múltiples bases de datos cuya característica clave es la preservación de la autonomía de las fuentes [21].

Un MSIS se compone de un conjunto de fuentes de datos independientes y un conjunto de vistas o consultas que definen los requerimientos de los usuarios. La diferencia principal entre un sistema de información tradicional y un MSIS reside en su definición y en la carga de sus datos. Mientras el esquema de datos de un sistema de información tradicional se define como una estructura de datos integrada, la estructura de un MSIS se define como un conjunto de vistas posiblemente independientes. Además, la carga de datos del primero la realizan los usuarios mediante sus aplicaciones, mientras que la carga del segundo se hace automáticamente a partir de las fuentes. Las aplicaciones de un MSIS involucran sólo la presentación y uso de los datos, sin actualizaciones [16].

En este contexto se han propuesto varias técnicas y herramientas que apuntan a resolver diferentes actividades de diseño de MSIS, que incluyen la selección de fuentes de datos relevantes [16], el diseño y generación del sistema

[1] Este trabajo se desarrolló durante una pasantía en el grupo Systèmes d'Informtation del Laboratorio PRISM – Universidad de Versailles, 2002.

[20][24][8][5] y su mantenimiento [4][21]. Estas herramientas en general son independientes y utilizan su propia metadata.

Nuestro principal aporte es la construcción de una plataforma para integrar esas herramientas y el desarrollo de un modelo de datos común para representar de una forma uniforme la metadata del MSIS (meta-modelo). Esta plataforma permite el flujo de datos entre las aplicaciones, el control de integridad y la incorporación de nuevas técnicas y herramientas.

En este artículo proponemos un meta-modelo para un caso particular de un MSIS: un Data Warehouse (DW). Dicho meta-modelo, basado en el desarrollado en el proyecto DWQ [14], define no sólo la estructura de las fuentes y vistas del DW, sino también las relaciones entre ellos, de manera de reflejar las expresiones de cálculo de las vistas a partir de datos de las fuentes. Dichas expresiones de cálculo indican: (i) las relaciones fuentes de dónde se extraerán los datos, (ii) las expresiones de cálculo para cada atributo de la vista, y (iii) las condiciones o restricciones adicionales para la selección de datos de las fuentes.

Resumiendo, este trabajo aporta no sólo la especificación de un meta-modelo para la plataforma de desarrollo de MSIS, sino también la especificación de expresiones de cálculo de vistas y una gramática para representar el cálculo de atributos y condiciones. Además presentamos un prototipo de la plataforma de diseño de MSIS que incluye la implementación de las expresiones de cálculo.

El resto del documento se organiza de la siguiente manera: La sección 2 presenta el marco de metadata de DWQ y describe el meta-modelo existente. La sección 3 introduce las nociones de cálculo de vistas mediante un ejemplo y formaliza dichos conceptos. En la sección 4 se extiende el meta-modelo para incorporar las expresiones de cálculo de vistas. La sección 5 describe la arquitectura general de la plataforma y la sección 6 concluye.

2. Preliminares

Nuestra propuesta toma como base el meta-modelo desarrollado en el contexto del proyecto DWQ [14]. En esta sección describimos brevemente el marco de trabajo del proyecto DWQ y el meta-modelo lógico del DW. Una descripción más detallada puede encontrarse en [15].

2.1. Marco de metadata de DWQ

El marco de metadata del proyecto DWQ propone tres perspectivas para describir un sistema de DW: (i) una perspectiva *conceptual* de la corporación, (ii) una perspectiva *lógica* de los modelos de datos y (iii) una perspectiva *física* del flujo de datos [15].

Cada una de las perspectivas y sus relaciones están ortogonalmente relacionadas con las tres capas tradicionales de un sistema de DW, que son las bases de datos fuentes, el DW corporativo y los data marts. La Figura 1 muestra el marco de trabajo.

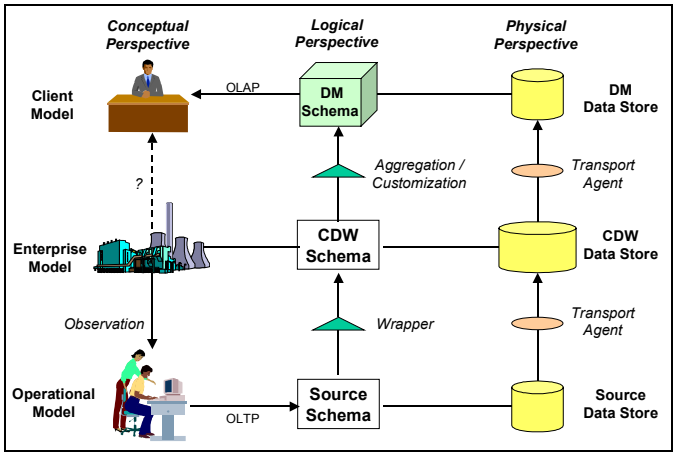


Figura 1 – Marco de metadata para DW

La *perspectiva conceptual* concierne los modelos de negocio de los sistemas de información de la empresa. El rol central corresponde al *modelo corporativo*, el cual brinda una visión corporativa de los *objetos* conceptuales de la corporación. También le conciernen los diferentes modelos *operacionales* y los modelos *clientes* de cada sección de la empresa. Tanto los modelos operacionales como los modelos clientes son vistas parciales del modelo cor-porativo.

La *perspectiva lógica* concibe al DW desde el punto de vista de los modelos de datos con-

cretos, implementados por los *esquemas* lógicos. Los modelos conceptuales tienen una representación lógica en los esquemas clientes o data marts (DM), el esquema de empresa o data warehouse corporativo (CDW) y los esquemas fuentes, respectivamente. La perspectiva lógica también describe la manera de calcular los objetos de cada DM a partir de los objetos del CDW y éstos a partir de los objetos fuentes.

La *perspectiva física* representa los depósitos de datos correspondientes a cada esquema y el flujo de datos entre ellos.

2.2. Meta-modelo lógico de DW

En este trabajo nos concentramos en la perspectiva lógica, es decir, en los esquemas de DMs, CDW y fuentes, y en la manera de calcular los objetos de un esquema en función de los de otro.

El meta-modelo existente describe las propiedades y componentes de los diferentes esquemas. En la sección 3 presentamos una extensión al meta-modelo para incorporar información sobre cómo calcular los objetos.

La Figura 2 muestra una versión simplificada del meta-modelo, donde sólo incluimos los objetos relevantes para nuestro análisis, otros componentes como restricciones, factores de calidad, esquemas históricos, etc. fueron excluidos para facilitar la legibilidad del modelo y comprensión de la propuesta.

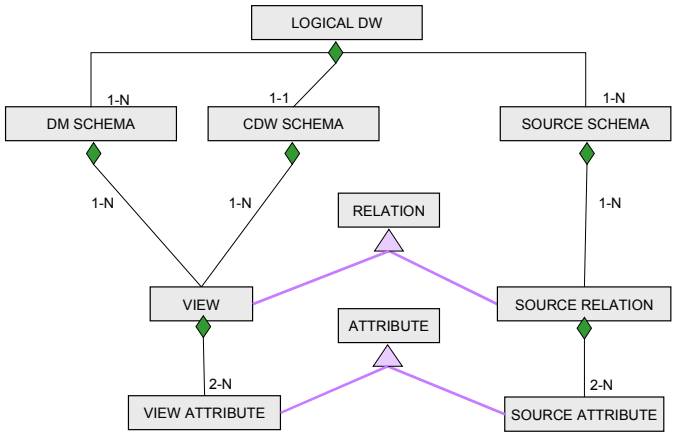


Figura 2 – Simplificación del meta-modelo lógico de un DW (notación UML)

El DW se compone de tres conjuntos de esquemas: (i) los esquemas de DMs que abstraen los modelos clientes, (ii) el esquema del CDW que representa el modelo corporativo y (iii) los esquemas fuentes que corresponden a los modelos operacionales.

Los esquemas están compuestos de relaciones que representan los objetos conceptuales, pero la naturaleza de esas relaciones es diferente. Los esquemas fuentes contienen relaciones fuentes con datos materializados y posiblemente heterogéneos, pero los esquemas de DMs y del CDW contienen vistas, posiblemente no materializadas que representan transformaciones de los datos de las relaciones fuentes. Para representar las relaciones fuentes y las vistas, se especializa la clase *Relation* en dos sub-clases *SourceRelation* y *View* respectivamente.

El mismo razonamiento se puede hacer para los atributos, obteniendo dos sub-clases *ViewAttribute* y *SourceAttribute*, ambas especializaciones de la clase *Attribute*.

Este meta-modelo enfatiza la representación de los esquemas. Por ejemplo, describe el esquema de las vistas, es decir, indica los atributos de los que está compuesta. Pero para definir una vista no es suficiente con dar el esquema se necesita definir la consulta de cálculo, la cual indica cómo calcular los datos a partir de relaciones fuentes.

Dichos cálculos pueden representarse como operaciones, cuya entrada son las relaciones fuentes y sus atributos. En las siguientes secciones discutimos una representación para expresiones de cálculo de vistas y modificamos el meta-modelo para incluir esas expresiones.

3. Cálculo de vistas

Nuestro objetivo es representar expresiones de cálculo complejas para vistas y darle al diseñador una plataforma robusta para soportar las diferentes actividades de desarrollo de un MSIS (limpieza de datos, carga de datos, etc).

A continuación presentamos el problema mediante un ejemplo.

3.1. Un ejemplo

La Figura 3a muestra una base de datos fuente con información sobre los empleados de la región sur de una compañía y sus salarios. Hay tres relaciones: *posts*, *salaries* y *extras*. La relación *posts* tiene información descriptiva de los empleados, la relación *salaries* tiene información de los salarios de cada empleado (en francos) y la relación *extras* tiene información de las horas extras trabajadas por cada empleado, incluyendo fecha, tiempo y monto a pagar (también en francos).

Supongamos que un usuario está interesado en la vista *employees* de la Figura 3b. La vista incluye como atributos: empleado (renombrando el atributo *id* de la fuente), puesto, salario (calculado en euros), la suma de importes de horas extras (también en euros) y la región (South). La vista puede calcularse a partir de las relaciones *posts*, *salaries* y *extras*, con las siguientes expresiones de proyección para sus atributos:

1) EMPLOYEES.employee ← POSTS.id
2) EMPLOYEES.post ← POSTS.post
3) EMPLOYEES.salary_e ←SALARIES.salary_f * 6.56
4) EMPLOYEES.extras_e ←SUM(EXTRAS.amount_f * 6.56)
5) EMPLOYEES.region ← "South"

Las flechas indican que el atributo de la vista (lado izquierdo) se calcula a partir de la expresión (lado derecho). La expresión de cálculo incluye diferentes tipos de cálculos, los cuales se discuten en la sección 3.3.

También podemos indicar condiciones que los datos seleccionados de las fuentes deban cumplir. En el ejemplo, el usuario puede estar interesado en horas extras del año en curso y en los empleados que han ganado más de 1000 euros en horas extras.

Podemos indicar las siguientes condiciones:

1) year (EXTRAS.date) = 2002
2) SUM (EXTRAS.amount_f * 6.56) > 1000
3) POSTS.id = SUPPLEMENTAIRES.id
4) POSTS.id = SALAIRES.id

Las dos primeras condiciones soportan las restricciones del usuario y las dos últimas soportan el join entre las relaciones. Las condiciones también pueden ser de diferentes tipos, los cuales son discutidos en [25].

Resumiendo, para definir una expresión de cálculo para una vista, se debe indicar:

- Las relaciones fuentes.
- Las expresiones de proyección para los atributos.
- Las condiciones.

Supongamos que existe otro esquema fuente para la región norte de la compañía, el cual incluye la relación *amounts* de la Figura 3c. La relación tiene información sobre empleados, sus salarios e importes de horas extras.

Podemos calcular (parcialmente) la vista a partir de la relación *amounts*. También podemos hacer una combinación de los datos obtenidos calculando ambas expresiones, por ejemplo, podemos computar la unión de los datos, o la intersección, o cualquier operación de combinación definida por el usuario.

En las siguientes secciones se presenta la definición de las expresiones de cálculo de vistas y se las incluye en el meta-modelo del DW.

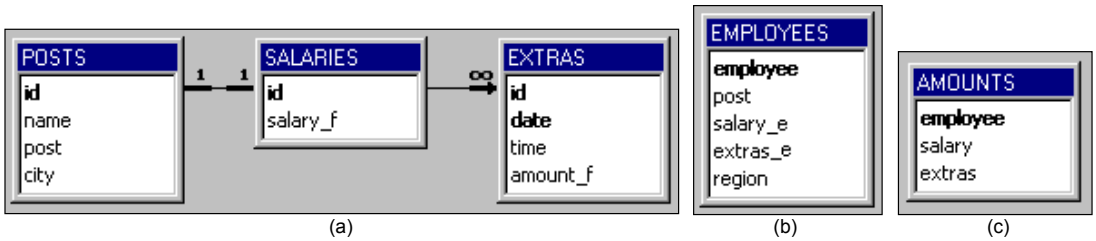


Figura 3 – Un ejemplo: a) base de datos fuente, b) una vista, c) otra relación fuente

3.2. Consultas GPSJ y GPSJ anidadas

Para representar una expresión de cálculo, utilizamos las consultas *GPSJ anidadas* (Generalized Projection, Selection and Join) propuestas en [9].

Una consulta GPSJ es una consulta de proyección-selección-join extendida con totalización, agrupamiento y selección de grupos [2]. Utiliza el operador de proyección generalizada [11] para representar totalizaciones. La proyección generalizada tiene la forma $\pi_{G, F(A)}$ donde G denota un conjunto de atributos para agrupar y F denota un conjunto de funciones de agregación sobre los atributos de A.

Una gran clase de consultas puede ser expresada con consultas GPSJ. En particular, todas las consultas SQL de la forma SELECT-FROM-WHERE-GROUP BY-HAVING pueden reducirse a esta forma si: (i) los atributos y funciones de totalización en las cláusulas GROUP BY y HAVING aparecen en la cláusula SELECT, (ii) las funciones de totalización no usan la cláusula DISTINCT y (iii) las cláusulas WHERE son conjuntivas [2].

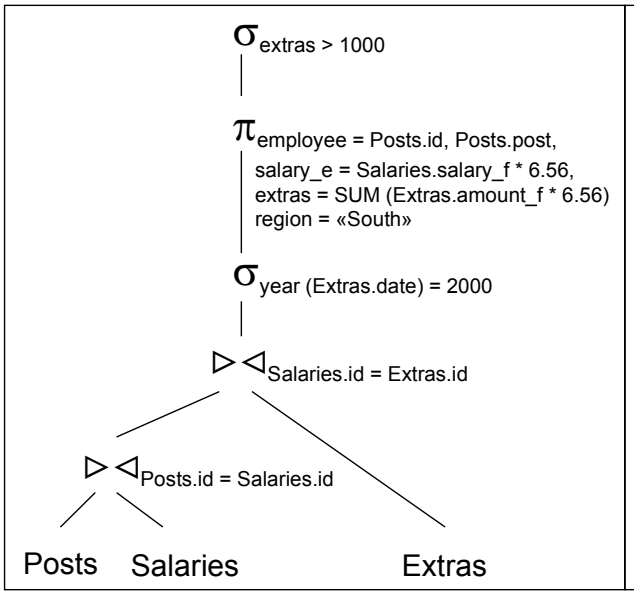
Usando la forma normal propuesta en [11], una consulta GSPJ tiene la siguiente forma:

$$\sigma_{CA} (\pi_{S,A} (\sigma_{CS} (R_1 \bowtie_{CR1} \ldots \bowtie_{CRn-1} R_n)))$$

donde:

- R = {R1, …Rn } es un conjunto de relaciones.
- E = S ∪ A es un conjunto de expresiones formadas a partir de atributos de las relaciones en R. Específicamente, S es un conjunto de atributos para agrupar y A es un conjunto de expresiones de totalización.
- C = CS ∪ CA ∪ CR_1 … ∪ CR_{n-1} es un conjunto de condiciones de selección formadas a partir de atributos de las relaciones en R. Específicamente, CS es un conjunto de condiciones sobre las relaciones (antes de agrupar), CA es un conjunto de condiciones sobre la proyección (después de agrupar) y CR = {CR_1… ∪ CR_{n-1}} es un conjunto de predicados de join entre las relaciones.

Como ejemplo consideremos la expresión de cálculo de la vista *employees* a partir de las relaciones fuentes *posts*, *salaries* y *extras* descritas en la sección anterior. Podemos resolver la expresión de cálculo con la consulta de la Figura 4a. Para ilustrar mejor el ejemplo, adjuntamos la correspondiente consulta SQL en la Figura 4b.



(a)

```
SELECT    Posts.id as employee,
          Posts.post,
          Salaries.salary_f * 6.56 as salary_e,
          sum(Extras.amount_f * 6.56) as extras,
          «South» as region
FROM      Posts,
          Salaries,
          Extras
WHERE     Posts.id = Salaries.id
AND       Posts.id = Extras.id
AND       year (Extras.date) = 2000
GROUP BY  Posts.id,
          Posts.post,
          Salaries.salary_f
HAVING    sum(Extras.amount_f * 6.56) > 1000
```

(b)

Figura 4 – Ejemplo de consulta GPSJ: a) grafo de la consulta, b) consulta SQL correspondiente

Una consulta GPSJ sólo permite la construcción de expresiones de totalización simples, pero no permite construcciones complejas como sub-consultas.

Las consultas *GPSJ anidadas* [9] son un mecanismo para permitir el cálculo de un conjunto más amplio de consultas, que no puede ser descrito sólo con un patrón de totalización. Anidar consultas GSPJ corresponde con utilizar el resultado de una consulta como entrada para otra consulta.

La forma general es la misma, pero considerando que {R1, …Rn} pueden ser relaciones fuentes o consultas GPSJ anidadas.

Las consultas GSPJ anidadas permiten la representación de cálculos complejos en los cuales pueden aplicarse secuencias de operadores y predicados de selección a diferentes granularidades. En [9] se discute la expresividad de las consultas GPSJ anidadas.

Una expresión de cálculo para una vista puede definirse dando:

- Un conjunto de relaciones que conforman la entrada de la expresión de cálculo (*from*). Como explicamos anteriormente, una vista puede calcularse tanto a partir de relaciones fuentes como de otras vistas.
- Una expresión de proyección para cada atributo de la vista (*select as*). La siguiente sección presenta una gramática para construir las expresiones.
- Un conjunto de condiciones o predicados que deben cumplir los datos de las relaciones (*where*).
- Un conjunto de condiciones o predicados que deben cumplir los datos una vez agrupados o resumidos (*having*).

3.3. Expresiones de proyección

En esta sección presentamos una gramática para representar expresiones de proyección. La gramática permite diferentes tipos de cálculos sobre un conjunto no prefijado de funciones de usuario que pueden ser extendidas y permite una traducción fácil y rápida de las consultas GPSJ a consultas en álgebra relacional o SQL.

La gramática propuesta diferencia 4 tipos de expresiones de proyección:

- *Value expressions*: Son constantes o funciones no relacionadas con los datos fuentes. Por ejemplo:
  las constantes: 3, “South”, null, o
  las funciones: today(), randomNumber().
- *Simple expressions*: Son atributos o funciones que no contienen agregaciones. Por ejemplo:
  Posts.id
  trim(Posts.post)
  Salaries.salary_f * 6.56
  concat(Posts.id, Posts.name)
- *Aggregate expressions*: Son agregaciones o funciones sobre ellas. Por ejemplo:
  average(Extras.amount_f)
  sum(Extras.amount_f * 6.56)
  sum(Extras.amount_f) / count(Extras.id)
- *Composed expressions*: Son funciones que combinan expresiones simples y agregaciones. Por ejemplo:
  sum(Extras.amount_f) / Salaries.salary_f
  sum(Extras.amount_f) * 1.25

La Figura 5 ilustra la diferencia entre expresiones simples y agregaciones. Los atributos *employee*, *post* y *salary_e* de la vista *employees* se calculan a partir de un valor de las relaciones *Posts* y *Salaries*, pero el atributo *extras_e* se calcula como agregación de varios valores en la relación *Extras*. Tanto el atributo *salary_e* como *extras_e* se calculan usando funciones de usuarios. La Figura 5 también muestra el uso de expresiones de valores en el cálculo del atributo *region* usando la constante *South*.

Gráficamente, podemos representar las reglas de cálculo de atributos usando flechas que van desde los atributos fuentes a los atributos de la vista. Cada flecha indica que el atributo fuente es usado para el cálculo del atributo de la vista. Los diferentes colores y formatos corresponden a los diferentes tipos de expresiones: las líneas rojas corridas corresponden a expresiones simples, las líneas verdes cortadas corres-

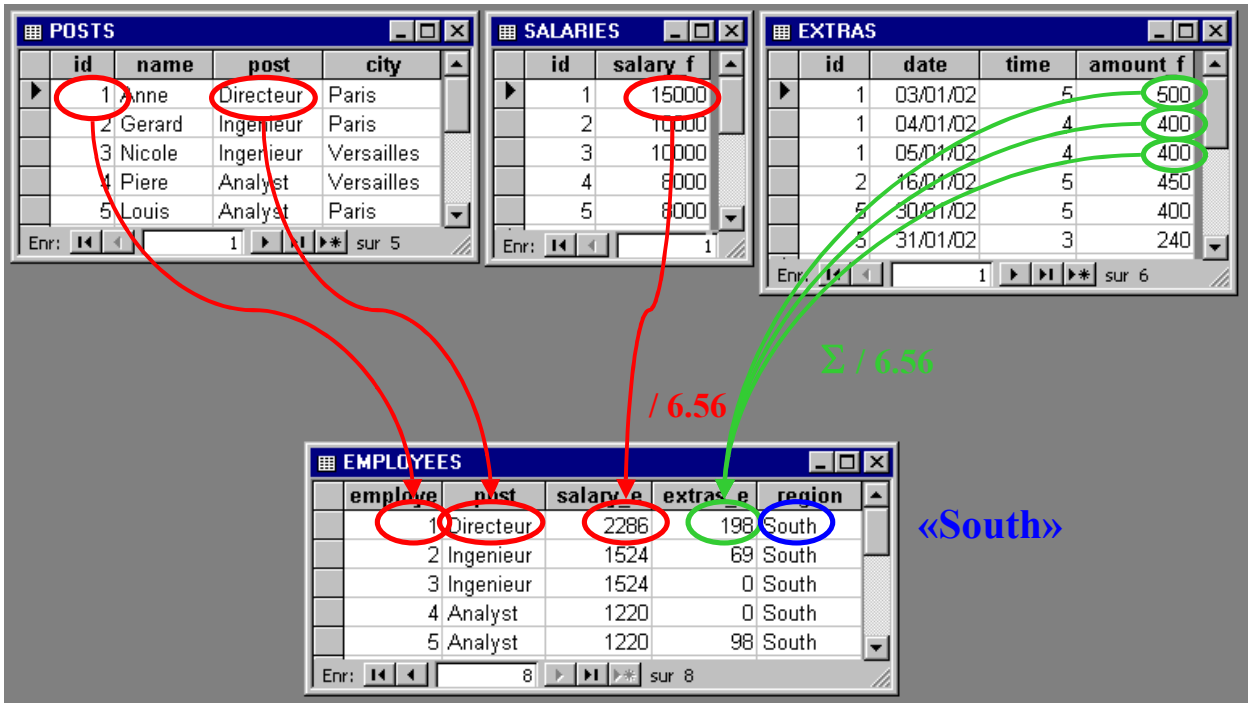


Figura 5 – Ejemplo de expresiones de proyección

ponden a expresiones de agregación. Las expresiones compuestas se representan con líneas verdes o rojas dependiendo de si el atributo se usa o no en una función de agregación. Si el cálculo usa funciones de usuario, se incluye una leyenda indicando la función. Para las expresiones de valores sólo se incluye la leyenda.

La Figura 6 muestra la representación gráfica de las expresiones de proyección del ejemplo.

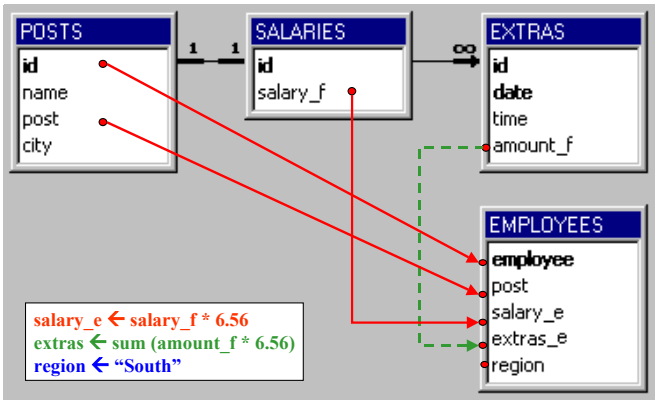


Figura 6 – Representación gráfica de expresiones de proyección

Pueden verse tres expresiones simples, una de ellas, la del atributo *salary_e*, tiene una leyenda con la función de conversión. También puede verse una expresión de agregación para el atributo *extras_e* con la correspondiente función de agregación (sum) y una expresión de valor para el atributo *region*.

La Figura 7 presenta la gramática para las expresiones de proyección. La gramática define los cuatro tipos de expresiones a través de los conjuntos *VExpr*, *SExpr*, *AExpr* y *CExpr*.

Las funciones de usuario de la gramática no se definen en forma explícita y pueden ser restringidas a cualquier subconjunto de funciones. La forma recursiva en la cual se define la gramática permite la definición de propiedades y funciones, tales como extraer los atributos que componen una expresión o extraer los atributos que no son parte de una agregación. Estas propiedades y funciones son útiles, por ejemplo, para escribir consultas SQL.

De la misma forma, se define una gramática para las condiciones de selección, que distingue entre las condiciones con y sin agregados. Omitimos la gramática por cuestiones de espacio [25].

Sea BasicTypes el conjunto de los tipos básicos, como *boolean, integer, float, string* y *date*.

Sea Values el conjunto de los valores constantes de los tipos básicos.

Sea Attributes un conjunto de atributos tal que: $\forall$ A $\in$ Attributes, Dom(A) $\in$ BasicTypes.

Value Expressions

- If E $\in$ Values ➔ E $\in$ VExpr

Simple Expressions

- If A $\in$ Attributes ➔ A $\in$ SExpr
- If $E_1 \ldots E_n \in$ SExpr, n $\geq$ 1, f $\in$ Dom(E_1) x … x Dom(E_n) ➔ T, T $\in$ BasicTypes ➔ f(E_1, …E_n) $\in$ SExpr

Aggregate Expressions

- If E $\in$ SExpr, g $\in$ Set(Dom(E)) ➔ T, T $\in$ BasicTypes ➔ g(E) $\in$ AExpr
- If $E_1 \ldots E_n \in$ AExpr, n $\geq$ 1, f $\in$ Dom(E_1) x … x Dom(E_n) ➔ T, T $\in$ BasicTypes ➔ f(E_1, …E_n) $\in$ AExpr

Composed Expressions

- If $E_1 \in$ SExpr, $E_2 \in$ AExpr, $E_3 \ldots E_n \in$ Sexpr $\cup$ Aexpr, n $\geq$ 2, f $\in$ Dom(E_1)x…x Dom(E_n) ➔ T, T$\in$BasicTypes ➔ f(E_1, …E_n) $\in$ CExpr

Figura 7 - Gramática para expresiones de proyección

4. Meta-modelo propuesto

En esta sección enriquecemos el meta-modelo lógico del DW mostrado en la Figura 2 con algunas clases que representan las expresiones de cálculo de vistas y sus componentes. Para entender mejor la propuesta, presentamos primero una versión simplificada que no considera el anidamiento de vistas y luego lo generalizamos.

La Figura 8 muestra el meta-modelo simplificado.

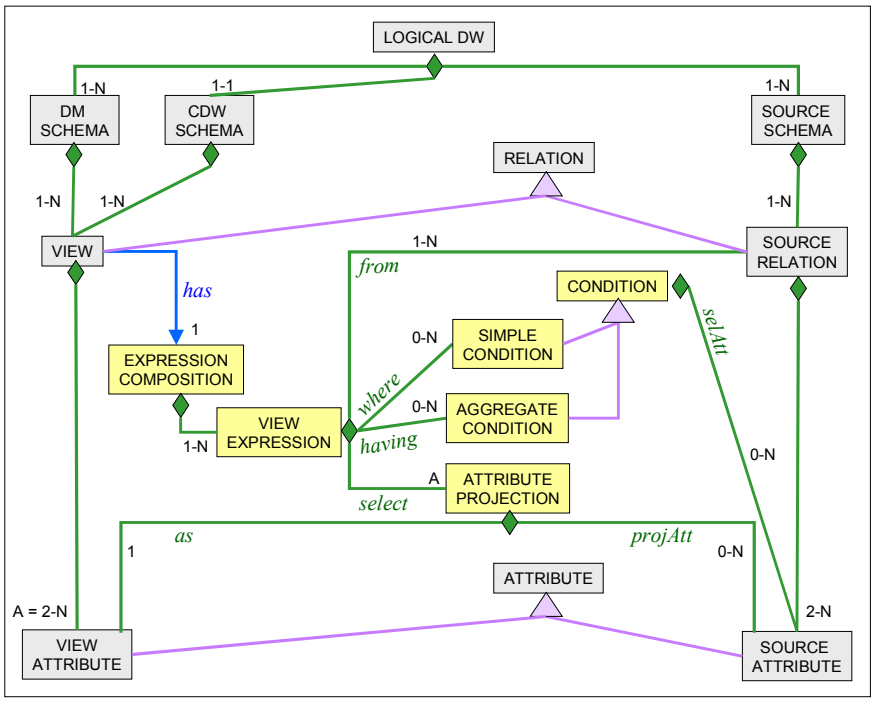


Figura 8 – Un meta-modelo simple

Para representar una vista utilizamos la clase *ViewExpression*, la cual tiene cuatro componentes que se corresponden con la definición dada en la sección 3.2, con roles: *from*, *select*, *where* y *having* respectivamente.

El componente *from* indica el conjunto de relaciones fuentes necesarias para calcular la vista.

El componente *select* indica el conjunto de expresiones de proyección usadas, cada una correspondiendo a un atributo de la vista. La clase *AttributeProjection* establece un mapeo entre un atributo de una vista a ser calculado (*as*) y los atributos fuentes usados para construir la expresión (*projAtt*) siguiendo las reglas de la gramática. Sólo se permite construir expresiones de proyección a partir de los atributos fuentes de las relaciones indicadas en el componente *from*.

Los componentes *where* y *having* indican las condiciones sin y con agregados respectivamente. Las clases *SimpleCondition* y *AggregateCondition* son especializaciones de la clase *Condition* y siguen las reglas de la gramática de condiciones. El rol *selAtt* indica qué atributos fuentes se usan para construir una condición, que deben ser atributos de las relaciones indicadas en el componente *from*.

Una vista puede ser calculada con varias expresiones de cálculo, pero debe existir un mecanismo para componer dichas expresiones al calcular la vista, por ejemplo, realizando la unión de las instancias. La clase *ExpressionComposition* expresa la manera de combinar varias expresiones para calcular la vista.

Para permitir el anidamiento de expresiones de cálculo de vistas, una vista debe poder calcularse tanto a partir de relaciones fuentes como de otras vistas. Por este motivo generalizamos el componente *from* para representar un conjunto de relaciones. La Figura 9 muestra dicha generalización. Se debe controlar la consistencia de las instancias, impidiendo que una vista pueda calcularse a partir de si misma, o que existan ciclos en la dependencia de cálculos (componente *from)*.

Como una vista puede calcularse a partir de cualquier relación, también las expresiones de proyección y las condiciones pueden construirse a partir de atributos de esas relaciones. Entonces, los componentes *projAtt* y *selAtt* se generalizan también.

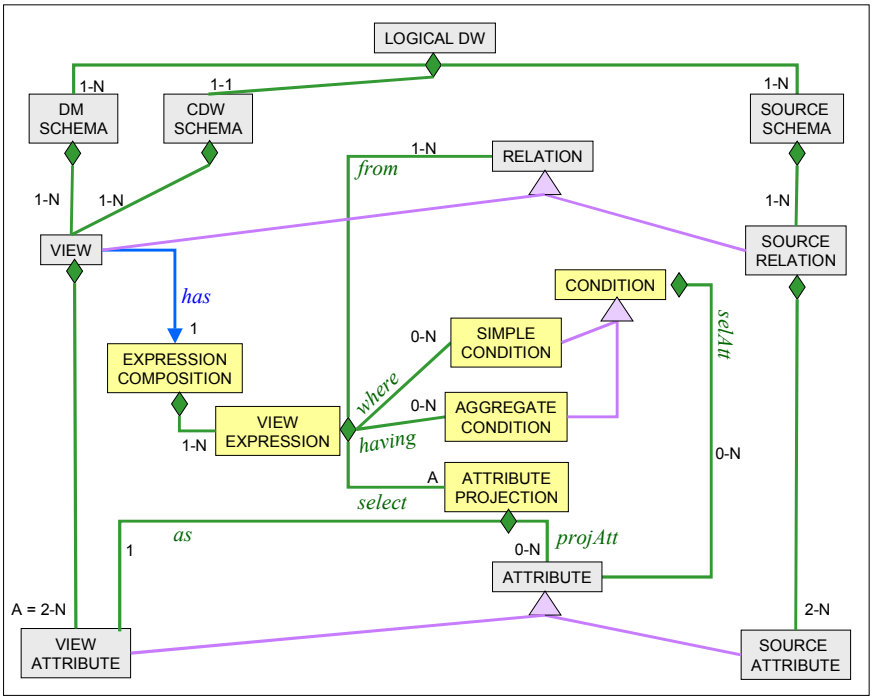


Figura 9 – Un meta-modelo generalizado

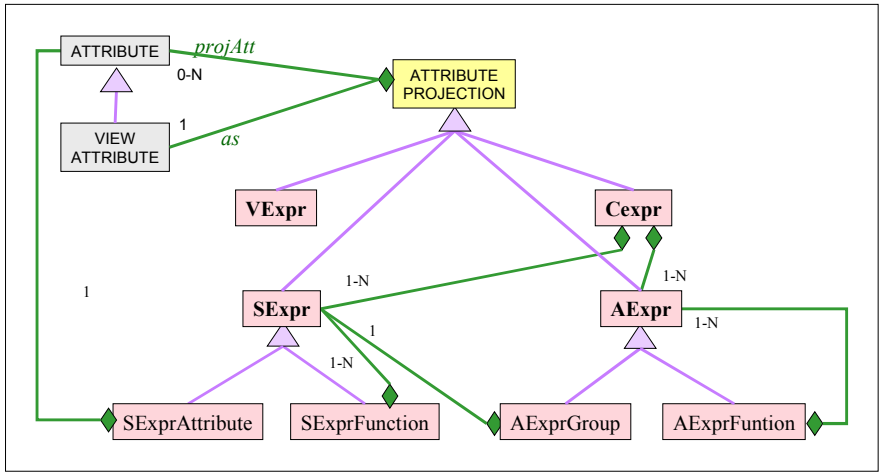


Figura 10 – Expresiones de cálculo de atributos

La Figura 10 muestra la jerarquía de clases para las expresiones de proyección.

Las sub-clases de la clase *AttributeProyection* representan los cuatro tipos de expresiones de proyección definidas en la gramática: VExpr, SExpr, AExpr y CExpr. Cada una puede ser extendida para representar una regla de cálculo específica. Por ejemplo, una expresión simple (SExpr) puede ser un atributo o una función sobre otras expresiones simples. Tanto el conjunto de funciones válidas cómo su cardinalidad pueden ser restringidas por cada aplicación.

El mismo tipo de jerarquías puede definirse para las clases *Condition* y *ExpressionComposition*. Se omiten por restricciones de tamaño del artículo.

5. Plataforma de diseño y mantenimiento de DWs

Nuestro objetivo general es el desarrollo de una plataforma de diseño y mantenimiento de DWs que integra y comunica de una manera uniforme las diferentes aplicaciones y herramientas de diseño [25]. Dicha plataforma permite el flujo de datos entre aplicaciones, el manejo centralizado de datos y el control de integridad entre aplicaciones. También permite la incorporación dinámica de nuevas herramientas y aplicaciones cuando nuevas técnicas son desarrolladas.

La Figura 11 esquematiza la arquitectura de la plataforma. El módulo de gestión del meta-modelo (meta-model manager) es el corazón de la plataforma. Es el responsable de la gestión y mantenimiento de toda la metadata del DW y de mantenerla accesible para las aplicaciones relacionadas.

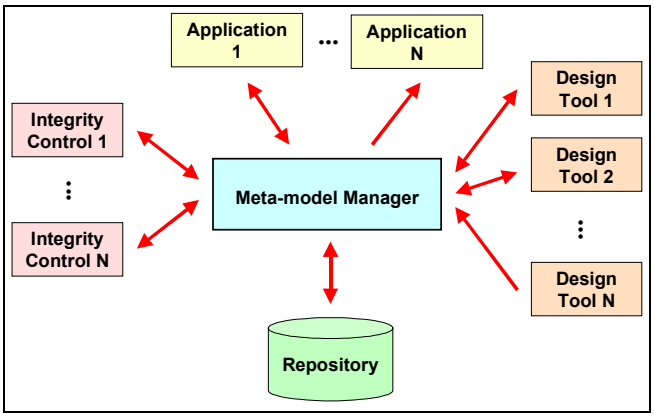


Figura 11 – Arquitectura de la Plataforma

El módulo de gestión del meta-modelo lee y almacena sus datos en un repositorio centralizado. Esto facilita el acceso a datos comunes y el desarrollo de funciones de control de consistencia, ya que las aplicaciones no acceden directamente a los datos.

Diferentes tipos de aplicaciones interoperan con el módulo de gestión del meta-modelo, por ejemplo herramientas de diseño y mantenimiento [8][23][24][16], programas de usuarios [6], aplicaciones de control de integridad [25]. Estas herramientas leen diferentes objetos como entrada y los actualizan o generan

nuevos objetos. Las herramientas pueden tener otras entradas o salidas, por ejemplo otros repositorios o directivas de usuarios.

Como ejemplo de aplicación se puede considerar el descubrimiento automático de expresiones de cálculo para vistas a partir de un conjunto de fuentes [16]. La herramienta que implementa esta funcionalidad toma como entrada un conjunto de vistas y las relaciones fuentes y genera las expresiones de cálculo de las vistas. Otro ejemplo consiste en generar las vistas del CDW transformando los esquemas fuentes a partir de estrategias de alto nivel del diseñador [20][24]. La herramienta desarrollada toma como entrada el esquema fuente y genera el esquema del CDW y las expresiones de cálculo de sus vistas.

Se cuenta con un prototipo de la plataforma desarrollado en Java, con el JDK versión 1.3, que accede a una base de datos Oracle® a través de JDBC [25]. Actualmente se está desarrollando una nueva versión que incorpora un servidor web y manejo de objetos Corba® [7].

6. Conclusiones

En este artículo encaramos el problema de modelar la metadata de un MSIS desde dos puntos de vista: la representación de los esquemas y la manera en que los objetos de dichos esquemas se calculan a partir de otros. En particular, describimos un meta-modelo lógico para un caso particular de un MSIS: un DW. El meta-modelo permite representar no sólo los esquemas del DW, sino también expresiones de cálculo para sus vistas relacionales a partir de relaciones fuentes.

Dicho meta-modelo es el corazón de una plataforma que soporta la interoperabilidad entre diferentes aplicaciones y herramientas que conciernen el desarrollo de MSIS.

Los resultados principales de este trabajo consisten en: (i) la especificación de expresiones de cálculo de vistas, (ii) una gramática para representar expresiones de proyección para los atributos de las vistas, (iii) la definición de un meta-modelo para DW que incorpora las expresiones de cálculo, y (iv) un prototipo de plataforma para soportar la metadata lógica de un DW.

Actualmente se están complementando el prototipo con los meta-modelos conceptual y físico de DW. La integración de varias herramientas y aplicaciones también está prevista. Paralelamente se está desarrollando una interfaz más sofisticada de acceso al repositorio y se está estudiando el uso de un manejador de metadata [7].

Agradecimientos

Agradecemos a los profesores Mokrane Bouzeghoub y Zoubida Kedad del Laboratorio PRISM por su apoyo durante el desarrollo del presente trabajo. Agradecemos también a Regina Motz y Raúl Ruggia por sus valiosas sugerencias para la redacción de este documento.

Referencias

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

[2] Akinde, M. Böhlen, M.: "Constructing GPSJ View Graphs". Proc. 1st. Int. Workshop on Design and Management of Data Warehouses (DMDW). Heidelberg, Germany, 1999.

[3] Boehnlein, M. Ulbrich-vom Ende, A.:"Deriving the Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems". Proc. 2nd. Int. Workshop on Data Warehousing and OLAP (DOLAP). Kansas City, USA, 1999.

[4] Bouzeghoub, M. Fabret, F. Matulovic-Broqué, M. "Modeling Data Warehouse Refreshment Process as a Workflow Application". Proc.1st.Int. Workshop on Design and Management of Data Warehouses (DMDW). Heidelberg, Germany. 1999.

[5] Bouzeghoub, M. Kedad, Z. "A Logical Model for Data Warehouse Design and Evolution". Proc. 2nd. Int. Data Warehousing and Knowledge Discovery (DaWaK). London, UK, 2000.

[6] Bouzeghoub, M. Lahlou, A. "Le Profiling en Intelligence d'Entreprise". Internal Report. PRISM, Université de Versailles Saint-Quentin-en-Yvelines. Versailles, France, 2001.

[7] Delfino, E.: "Integración de herramientas CASE usando Internet, CORBA, y repositorios de metainformación". Accepted paper for Congreso Iberoamericano de Telemática (CITA), 2002.

[8] Garbusi, P. Piedrabuena, F. Vázquez, G.: "Diseño e Implementación de una Herramienta de ayuda en el Diseño de un Data Warehouse Relacional". Undergraduate project. Advisors: Adriana Marotta, Alejandro Gutiérrez. InCo, Universidad de la República. Montevideo, Uruguay, 2000.

[9] Golfarelli, M. Rizzi, S.:"View Materialization for Nested GPSJ Queries". Proc. 2nd. Int. Workshop on Design and Management of Data Warehouses (DMDW). Stockholm, Sweden, 2000.

[10] Gray, J. Bosworth, A. Layman, A. Pirahesh, H.: "Data Cube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals". Proc. 12. Int. Conf on Data Engineering (ICDE). New Orleans, USA, 1996.

[11] Gupta, A. Harinarayan, V. Quass, D.: "Aggregated-query processing in Data Warehousing environments". Proc. 21st. Int. Conf on Very Large Data Bases (VLDB). Zurich, Switzerland, 1995.

[12] Gupta, H.: "Selection of Views to Materialize in a Data Warehouse". Proc. Int. Conf. on Database Theory. Delphy, Greece, 1997.

[13] Inmon, W.: "Building the Data Warehouse". John Wiley & Sons, Inc. 1996.

[14] Jarke, M. Vassiliou, Y.: "Data Warehouse Quality Design: A Review of the DWQ Project". In Proc. MIT Conf. on Information Quality, UK, 1997.

[15] Jarke, M. Lenzerini, M. Vassiliou, Y. Vassiliadis, P.: "Fundamentals of Data Warehouses". Springer-Verlag, 1999.

[16] Kedad, Z. Bouzeghoub, M.: "Discovering View Expressions from a Multi-Source Information System". Proc. 4th. Int. Conf. on Cooperative Information Systems (CoopIS), Edinburgh, Scotland, 1999.

[17] Kedad, Z.: "Techniques d'intégration dans les systèmes d'information multi-source". PhD Thesis. PRISM, Université de Versailles Saint-Quentin-en-Yvelines. Versailles, France. 1999.

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

[19] Ligouditianos, S. Sellis, T. Theodoratos, D. Vassiliou, Y.: "Heuristic Algorithms for Designing a Data Warehouse with SPJ Views". Proc. Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK). Florence, Italy, 1999.

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

[21] Motz, R.: "Dynamic Maintenance of an Integrated Schema". PhD Thesis. IPSI-GMD, Darmstadt University of Technology, Darmstadt, Germany. On going work.

[22] Peralta, V. Garbusi, P. Ruggia, R.: "DWD: Una herramienta para diseño de Data Warehouses basada en transformaciones sobre esquemas". Internal Report. InCo, Universidad de la República, Uruguay, 2000.

[23] Peralta, V. Ruggia, R.: "Implementación de herramientas CASE que asistan en el Diseño de Data Warehouses". Technical Report. InCo, Universidad de la República, Uruguay, 2001.

[24] Peralta, V.: "Diseño lógico de un Data Warehouses a partir de Esquemas Conceptuales Multidimensionales". Master Thesis. InCo - Pedeciba, Universidad de la República. Montevideo, Uruguay, 2001.

[25] Peralta, V. "A Framework for Multi-Source Information Systems Development". Internal Report, Université de Versailles Saint-Quentin-en-Yvelines. Versailles, France, 2002.

[26] Sheth, A. Larson, J.: "Federated database systems and managing distributed, heterogeneous, and autonomous databases". ACM Computing Surveys, 22(3):183-226, 1990.

[27] Theodoratos, D. Ligoudistianos, S. Sellis, T.: "Designing the Global Data Warehouse with SPJ Views". Proc. 11th. Int. Conf. on Advanced Information Systems Engineering (CAISE). Heidelberg, Germany, 1999.

[28] Wiederhold, G.: "Mediators in the architecture of future information systems". IEEE Computer, 25(3):38-49, 1992.

[29] Yang, J. Karlapalem, K. Li, Q.: "Algorithms for materialized view design in data warehousing environment". Proc. 23rd. Int. Conf on Very Large Data Bases (VLDB). Athens, Greece, 1997.

[30] Yang, L. Miller, R. Haas, L. Fagin, R.: "Data-Driven Understanding and Refinement of Schema Mappings". Proc. ACM SIGMOD Conf. Santa Barbara, USA, 2001.

[31] Zhuge, Y. Garcia-Molina, H. Wiener, J. : "Consistency Algorithms for Multi-Source Warehouse View Maintenance". Journal of Distributed and Parallel Databases, July 1997.