A Framework for Multi-Source Information Systems Development

Verónika Peralta

Université de Versailles-St-Quentin-en-Yvelines.
Laboratoire PRISM
Veronika.Peralta@prism.uvsq.fr

Abstract: A Multi-Source Information System (MSIS) is composed of a set of independent data sources and a set of views that define user requirements. Its differences with classical information systems introduced new design activities and motivated the development of new techniques.

In this paper we present a general framework that allows an easy integration of design and maintenance tools through a common metadata platform that centralizes the inter-application data management and the integrity control routines.

We study a particular case of MSIS: a Data Warehouse (DW) and propose a model to represent its metadata from two points of view: the schema representation and the inter-schema relationships that allow to calculate an object from another ones.

Our work includes the development of a prototype that implements the DW metadata model as the core of the design platform.

1. Introduction

The need to access in a uniform way to information that is available in several data sources is every day stronger and generalized. Complex user requirements, in the form of views or queries, need data from possibly heterogeneous and autonomous sources, where data can be represented in different formats.

A Multi-Source Information System (MSIS) is an information system that accesses data from different independent databases in order to solve user requirements over these sources.

Examples of a MSIS are web systems, where data is extracted from web pages, integrated and presented to the user. The wrappers and mediators architecture [Wie92] is commonly used to perform these tasks. OLAP and Data Warehouse systems [Inm96] also extract, transform and integrates information from various, possibly heterogeneous sources and make it available for the strategic analysis of data. Other examples of MSIS are federations [She90] where the key characteristic is the preservation of the source autonomy [Mot02].

A MSIS is composed of independent data sources and views or queries that define user requirements. Figure 1 sketches the general MSIS architecture.

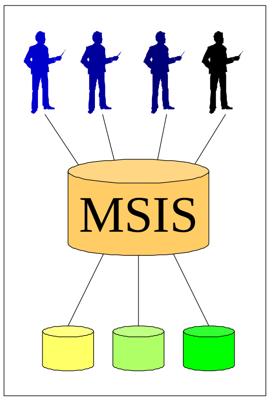


Figure 1 – Multi-Source Information Systems

The main difference between an MSIS and a classical information system resides in its definition and its feeding with data. While the database schema of a classical information system is defined as an integrated data structure, the database structure of a MSIS is defined as a set of (possible independent) views. Moreover, while the feeding of the first is done by the users through their applications, the feeding of the second is automatically done by the system from the data sources. MSIS applications aim only at the use of this data without any direct update [Ked99].

MSIS have introduced new design activities such as the selection of relevant data sources [Yan01] [Ked99], the definition of schema mappings [Yan01] [Vid01], the selection of views to materialize [The97] [The99], the definition and generation of mediators [Wie92] [Vid01], the data integration [Cal99] [Fan97] [Ked99b], the data cleaning [Gal01], the update propagation of source changes [Zhu97] [Mar01] [Vdo01] [Mot02] and the data load and refreshment [Bou99].

Our goal is not to solve all these problems; our general objective is to build a platform to integrate different techniques, process and tools that concern the development of MSIS.

The platform uses a common data model to represent the MSIS metadata (meta-model) in a uniform way, allowing the inter-application data flow and the integrity control and the dynamic incorporation of new tools and applications. Our first version of the platform describes a DW meta-model.

A DW as a particular case of MSIS that stores information oriented to satisfy decision-making requests. DW data is the result of transformations, quality control and integration of data from different source databases. User request are generally complex queries that involve the crossing of great amounts of data, groupings and aggregations.

A DW meta-model has been developed in the context of the DWQ project [Jar97]. This meta-model conceives the DW emphasizing its schemas' representation. We are also interested in representing the operational relations and mappings between schema objects, and how an object is calculated from another ones.

In this work, we extend the DW logical meta-model to represent these calculations and present an algebra to represent the calculation expressions. Explain more

We also present a prototype of the platform that includes the implementation of the DW logical meta-model.

The remaining of this document is organized as follows: Section 2 describes the platform architecture. Section 3 presents the DWQ metadata framework and describes the existing meta-model. Section 4 introduces the calculation notions by means of an example, and section 5 formalizes it. Section 6 extends the meta-model adding the calculation expression metadata. Section 7 describes the prototype and section 8 concludes.

2. DW design and maintenance platform

Our general goal is to develop a DW design and maintenance platform to integrate and communicate different applications and design tools in a uniform way. This platform allows the data flow between applications, the centralized management of common data and the inter-application integrity control. It also allows the dynamic incorporation of new tools and applications when new techniques are developed.

We use a detailed DW meta-model to describe all conceptual, logical and physical aspects of the DW, the different data models, data stores and the data flow between them. A meta-model manager is the heart of our platform. It is responsible of manage and maintain all DW meta-data and make it accessible for the related applications.

The meta-model manager loads and stores its data in a centralized repository. This makes easier the access to common data and their consistency control. Explain more

Different types of applications interoperate with the meta-model manager, for example design and maintenance tools, user applications or integrity control applications. These tools load different meta-objects as input and update them or generates new ones. The tools can have other inputs or outputs, for example other repositories or user guidelines.

Figure 2 sketches the platform architecture. The meta-model manager is in the central position and manages the applications access to metadata.

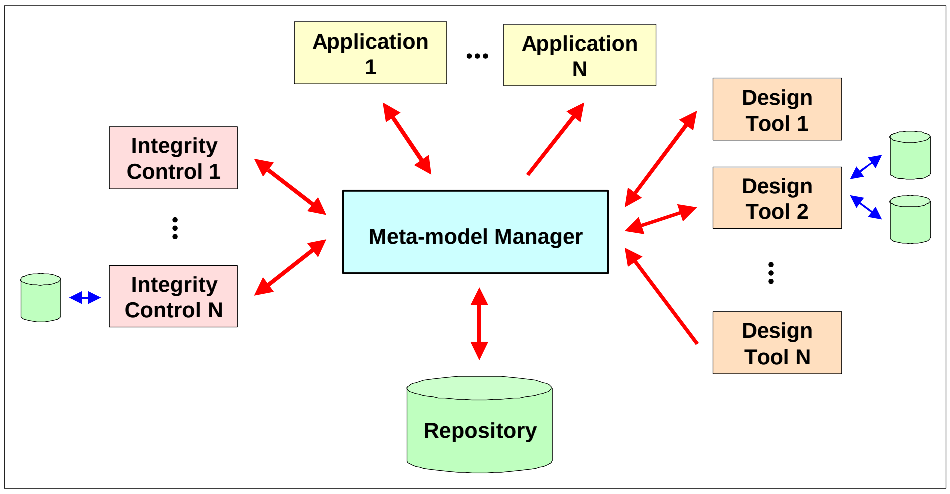


Figure 2 –Platform architecture

3. Existing DW logical meta-model

This work continues the DW meta-model development of the DWQ project [Jar97].

In the following sections we shortly describe the DWQ metadata framework and part of their meta-model. A more detailed description can be found in [Jar99].

3.1. DWQ metadata framework

The DWQ Data Warehouse metadata framework proposes three perspectives to describe a DW system: a *conceptual* enterprise perspective, a *logical* data modeling perspective and a *physical* data flow perspective [Jar99].

The *conceptual perspective* concerns the business model of the information systems of an enterprise. The central role is played by the *enterprise model*, which gives an integrative overview of the conceptual *objects* of the enterprise. It also concerns the different operational models and the client models of each section of the enterprise. Both operational and client models are partial views of the enterprise model.

The *logical perspective* conceives a Data Warehouse from the viewpoint of the actual data models involved, which implements the logical *schemas*. The conceptual models have a logical representation in the various schemas, namely, client or data mart (DM) schemas, enterprise or corporate data warehouse (CDW) schemas and source schemas. The logical perspective also describes the way to calculate data mart objects from enterprise objects and these from source objects.

The *physical perspective* represents the data stores corresponding to each schema and the physical data transport between them.

Each of these perspectives, and their relationships are orthogonally linked to the three traditional layers of data warehousing, namely sources, corporate DW and data marts. Figure 3 shows the framework.

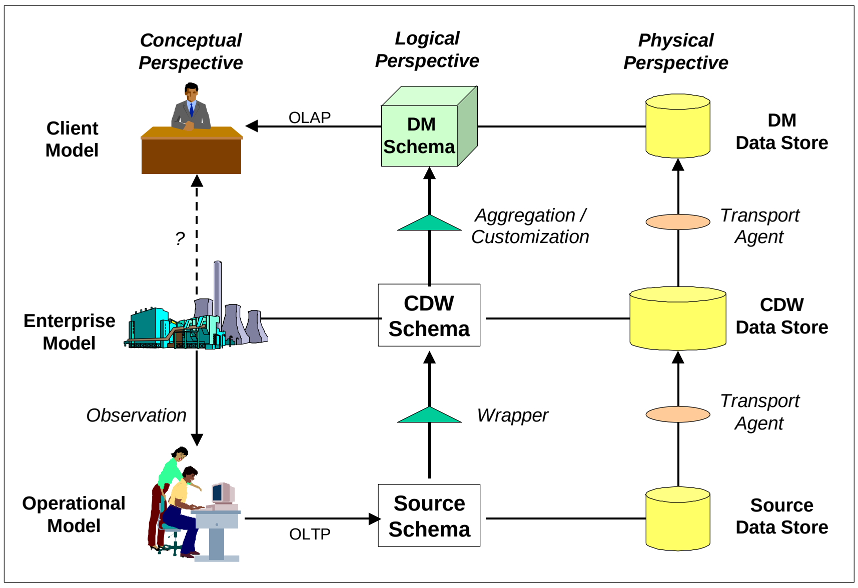


Figure 3 – Data Warehouse Metadata framework

3.2. DW logical meta-model

In this work, we concentrate in the logical perspective, i.e. the DM, CDW and source schemas, and the way of calculating schema objects from another ones.

The existing meta-model describes the properties and components of the different schemas. In section 5 we present an extension to the meta-model that adds information about how to calcule the objects.

Figure 4 shows a first simplified version of the DW logical meta-model.

The DW is composed of three main schema sets: (i) the DM schemas that abstract the client models, (ii) the CDW schema which represent the enterprise model and (iii) the source schemas that corresponds to the operational data models.

Each schema is composed of relations that represent the conceptual objects and the relations are composed of attributes. One relation can reference another one, representing the conceptual link between conceptual objects. We also consider a relation can be part of a history schema.

From the quality point of view, there are constraints and quality factors related to most of the objects.

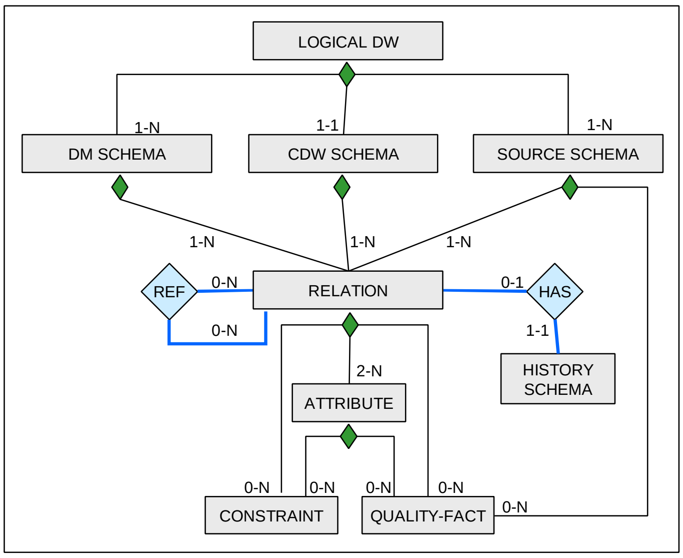


Figure 4 – A simple DW logical meta-model

For a best understanding of the figures, we omit the *constraint*, *quality-fact* and *history-schema* meta-classes and the *ref* meta-relationship in the following sections.

The source, CDW and DM schemas, are all composed of relations. However, the nature of these relations is different. The source schemas are composed of source relations with materialized and possibly heterogeneous data. But the CDW schema and the DM schemas are composed of (possibly not materialized) views, that represent transformations of the source relations data.

To represent both the source relations and the views, we specialize the meta-class *relation* in two meta-classes: *source-relation* and *view* respectively.

The same reasoning can be done for the attributes, resulting in two meta-classes: *view-attribute* and *source-attribute*. Both meta-classes are specializations of the *attribute* meta-class.

The resulting schema is shown in Figure 5.

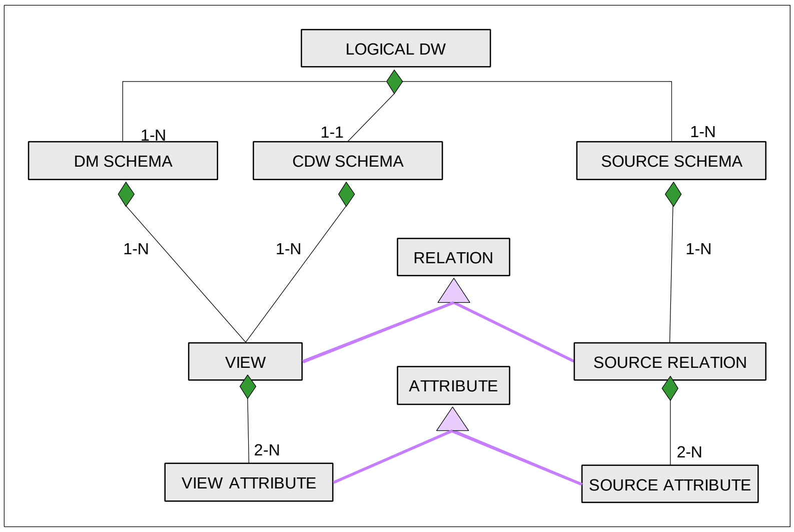


Figure 5 – Relation and attribute specializations

This meta-model emphasizes the schema representation. For example, for a view, it describes its schema, i.e. how it is composed of attributes. But to define a view, its schema it is not sufficient. We also need to represent its query, i.e. the way to calculate its data from the source relations.

The calculation expressions can be represented as operations which inputs are the source relations and their attributes. In the following sections we discuss a representation for the view calculation expressions and we include meta-classes in the model to support it.

4. An example

Figure 6a shows an example source database with information about the employees of the south region of a company and their salaries. There are three relations, namely: *posts*, *salaries* and *extras*. The *posts* relation has descriptive information about employees, the *salaries* relation keep information of the salary amount (in francs) of each employee, and the *extras* relation keep information of the extra-time worked by each employee, including date, time and amount to be paid (also in francs).

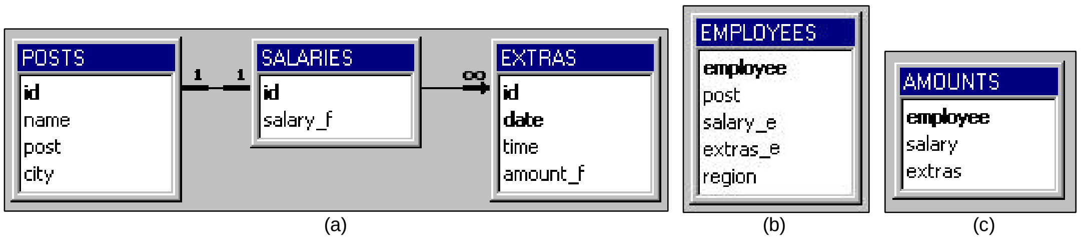


Figure 6 – The employee example: a) source relations, b) a view, c) another source relation

Suppose a user is interested in the view *employees* shown in Figure 6b. The view includes as attributes: the employee (renaming the *id* source attribute), the post, the salary (calculated in euros), the sum of extra-time amount (also in euros) and the region (south). The view can be calculated from source relations *posts*, *salaries* and *extras*, with the following calculation expressions for its attributes:

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”

The arrows indicate that the view attribute is calculated from the expression.

The calculation expressions include different types of calculations, which are discussed in section 5.3.

We also can indicate constraints that the data selected from the sources must achieve. In the example, the user can be interested only in extra-work of current year and in employees who have earn more than 1000 euros due to extra-work.

We can indicate the following constraints:

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

The first two constraints support the user constraints, and the last ones support the join between relations. The constraints are also of different types, which are discussed in section 5.4.

Finally, to define a calculation expression for a view, we must indicate:

- The source relations.
- The calculation expressions for the attributes.
- The constraints.

Suppose there is another source schema for the north region employees of the company, witch includes the *amounts* relation shown in Figure 6c. The relation has information about employees, their salaries and extra-work amount.

We can (partially) calculate the view from the *amounts* relation. And we can also make a combination of the data from both calculation expressions, for example, we can make the union of data, or the intersection, or any user-defined combination operation.

In the following sections, we present a definition for the view calculation expressions, and we include them in the DW logical meta-model.

5. View expressions

Our goal is to represent complex calculation expressions for a view and to give the designer a powerful framework to support the different design activities for MSIS (data cleanning, load and refreshment, inter-schema mapping, etc.).

To represent a calculation expression, we utilize the Nested GPSJ (Generalized Projection, Selection and Join) queries proposed by Golfarelli and Rizzi [Gol00]. These queries allow the representation of complex calculations with sequences of aggregate operators and selection predicates at different granularities.

We also extend the calculation types to give more expressiveness and present a grammar to represent the calculations.

5.1. GPSJ queries

A GPSJ query is a project-select-join query extended with aggregation, grouping and group selection [Aki99]. It uses the generalized projection operator [Gup95] to represent aggregation. The generalized projection has the form $\pi_{G,\, F(A)}$ where G denotes the set of group-by attributes and F denotes a set of aggregate functions over attributes in the attribute set A.

A large class of queries can be expressed as GPSJ queries, in particular, all SELECT-FROM-WHERE-GROUP BY-HAVING queries can be reduced to this form if: (i) the attributes and aggregate functions in the GROUP BY and HAVING clauses appear in the SELECT clause, (ii) no aggregate functions use the DISTINCT keyword and (iii) the WHERE clauses are conjunctive.

Using the Normal Form proposed in [Gup95], a GSPJ query has the general form:

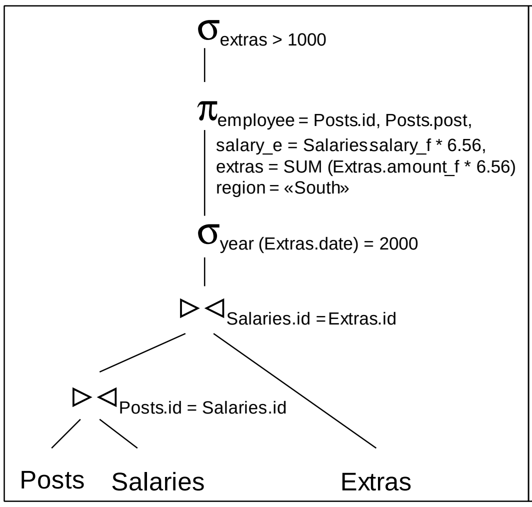
$$\sigma_{CA} (\pi_{S,A} (\sigma_{CS} (R_1 \bowtie_{CR1} \dots \bowtie_{CRn-1} R_n)))$$

where:

- R = {R1, …Rn } is a set of relations.
- E = S ∪ A is a set of expressions formed from attributes of the relations in R. Specifically, S is a set of group by attributes and A is a set of aggregate expressions.
- C = CS ∪ CA ∪ CR_1 … ∪ CR_{n-1} is a set of selection constraints formed from attributes of the relations in R. Specifically, CS is a set of constraints on the relations (before group by), CA is a set of constraints on the projection (after group by) and CR = {CR_1 … ∪ CR_{n-1}} is a set of join predicates between the relations.

As an example consider the calculation expression for view *employees* from the source relations *posts*, *salaries* and *extras*, described in the example of section 4.

We can resolve the calculation expression with the query shown in Figure 7a. For better illustrating the example, we also include the corresponding SQL query in Figure 7b.


σ extras > 1000
π employee = Posts.id, Posts.post, salary_e = Salaries.salary_f * 6.56, extras = SUM (Extras.amount_f * 6.56) region = «South»
σ year (Extras.date) = 2000
▷◁ Salaries.id = Extras.id
▷◁ Posts.id = Salaries.id
Posts
Salaries
Extras

```
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
```

(a) (b)

Figure 7 – A GPSJ query example: a) query graph, b) corresponding SQL query

5.2. Nested GPSJ queries

A GPSJ query only allows the construction of simple aggregates expressions, but doesn't allow other complex constructions as subqueries. The nested GPSJ queries [Gol00] are a mechanism to allow the calculation of a bigger set of queries, which cannot be described only by an aggregation pattern.

Nesting GSPJ queries means using the result from a query as the input for another query.

The general form of the query is the same, but the {R1, …Rn} may be either source relations or nested GPSJ queries.

Nesting GSPJ queries allow the representation of complex calculations in which sequences of aggregate operators may be applied and selection predicates may be applied at different granularities. In [Gol00] the authors discuss the expressiveness of the Nested GPSJ expressions.

A view calculation expression can be defined given:

- A set of relations, that conform the input for the view calculation (from). As explained before, we can use as input both source relations or another view.
- A calculation rule for each attribute of the view, called an attribute expression (select as). The following section describes a grammar for building the attribute expressions.
- A set of selection constraints or predicates on the relations (where). Section 5.4 describes the constraints.
- A set of selection constraints or predicates on the aggregations (having). The constraints are also described in section 5.4.

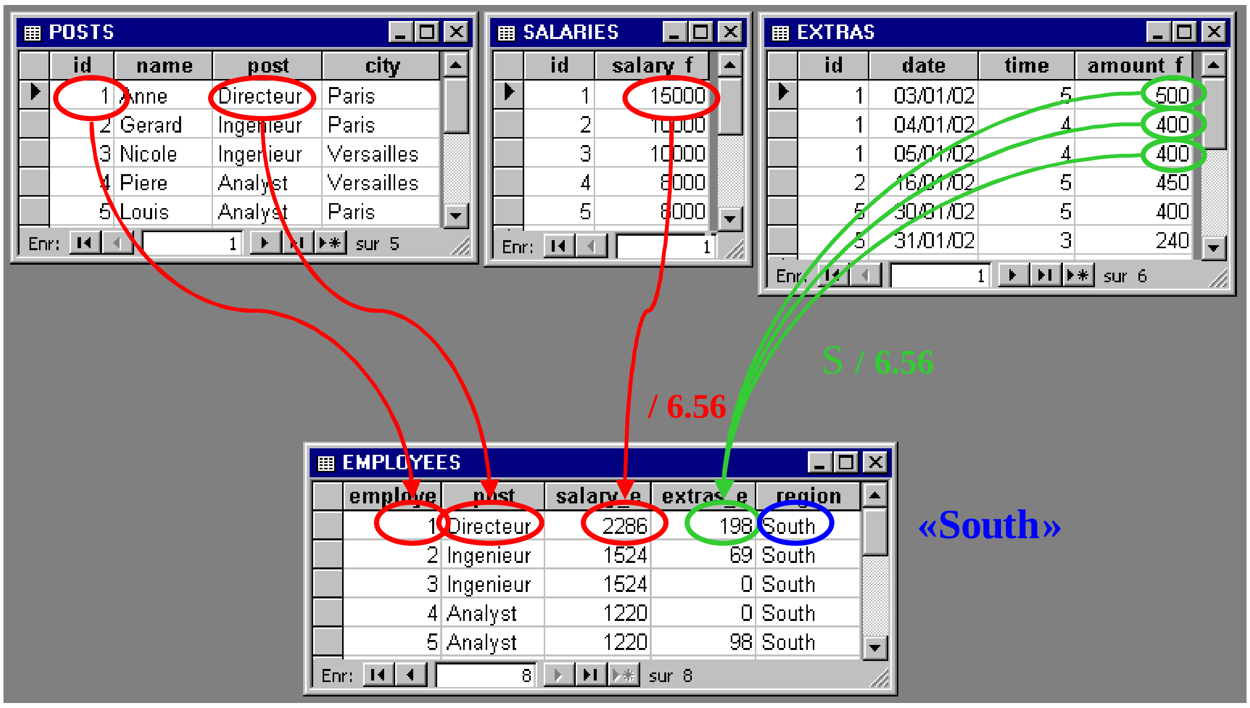
5.3. Attribute expressions

In this section we present a grammar for representing attribute expressions or calculation rules. The grammar must allow different calculation types, which a not-fixed user-function set which eventually can be extended and must allow an easy and fast translation to the GPSJ queries algebra and SQL syntax.

The grammar differences four types of attribute expressions:

- Value expressions: a constant or user-function not related with the source relations. For example the constants: 3, “South”, null, or the user-functions: today() or random-number().
- Simple expressions: an attribute or user-function without aggregations. For example: Posts.id, Salaries.salary_f * 6.56, trim(Posts.post), concat(Posts.id, Posts.name).
- Aggregate expressions: an aggregate function or user-function with aggregates. For example: average(Extras.amount_f), sum(Extras.amount_f * 6.56), sum(Extras.amount_f) / count(Extras.id).
- Composed expressions: a user-function that combines simple and aggregate calculations. For example: sum(Extras.amount_f) / Salaries.salary_f, sum(Extras.amount_f) * 1.25.

Figure 8 illustrates the difference between simple and aggregate expressions. The view attributes *employee*, *post* and *salary_e* are calculated from one value of relations *Posts* and *Salaries*, but the view attribute *extras_e* is calculated as the aggregation of several values in relation *Extras*. Both, *salary_e* and *extras_e* attributes are calculated using a user-function. Figure 8 also shows the use of a value expression: *South* to calculate the *region* attribute, which is not represented in the source.



POSTS

id	name	post	city
1	Anne	Directeur	Paris
2	Gerard	Ingenieur	Paris
3	Nicole	Ingenieur	Versailles
4	Piere	Analyst	Versailles
5	Louis	Analyst	Paris

Enr: 1 sur 5

SALARIES

id	salary_f
1	15000
2	10000
3	10000
4	8000
5	8000

Enr: 1

EXTRAS

id	date	time	amount_f
1	03/01/02	5	500
1	04/01/02	4	400
1	05/01/02	4	400
2	16/01/02	5	450
5	30/01/02	5	400
5	31/01/02	3	240

Enr: 1 sur 6

EMPLOYEES

employe	post	salary_e	extras_e	region
1	Directeur	2286	198	South
2	Ingenieur	1524	69	South
3	Ingenieur	1524	0	South
4	Analyst	1220	0	South
5	Analyst	1220	98	South

Enr: 8 sur 8

Figure 8 – Attribute expressions example

Graphically, we can represent attribute calculation rules as arrows between source and view attributes. Each arrow means that the source attributes is used for the calculation of the view attribute. The different arrow colors correspond to the different calculation types: red for simple expressions, green for aggregate expressions. Composed expressions are represented with both, green and red arrows depending if the source attributes are used in an aggregate function or not. If the calculation has user-functions, a legend is included to indicate the function. For the external values, only the legend is included.

Figure 9 shows the graphical representation of the view expression discussed. It shows three simple expressions, one of them, for the attribute *salary_e* has a legend with the conversion function. It also shows an aggregate expression for the attribute *extras_e* with the corresponding sum function, and a value expression for attribute *region*.

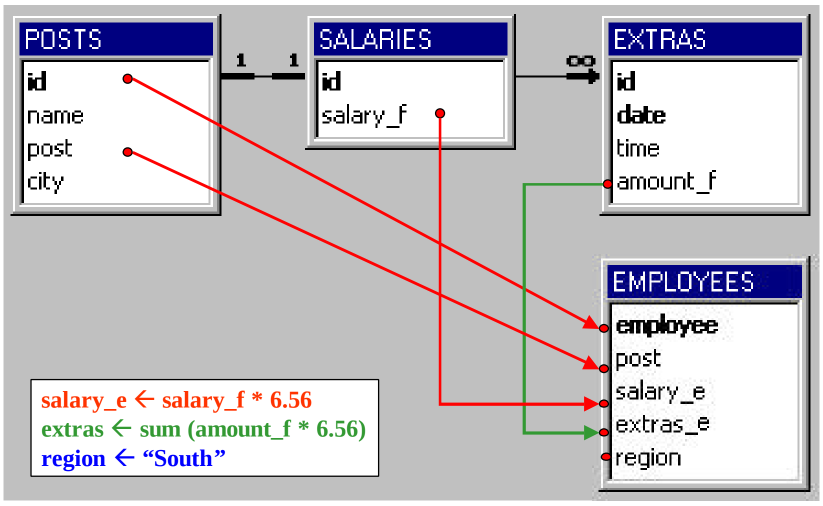


Figure 9 – Graphical representation of attribute expressions

The rest of the section presents the grammar for the attribute expressions. The grammar defines the four types of expressions, by the sets *VExpr*, *SExpr*, *AExpr* and *CExpr*.

Let BasicTypes be the set of basic types, such as *boolean, integer, float, string* and *date*. Let Values be a set of constant values of basic types. Let Attributes be a set of attributes, such that for each A ∈ Attributes, Dom(A) ∈ BasicTypes.

Value Expressions

♦ If E ∈ Values

E ∈ VExpr

Simple Expressions

♦ If A ∈ Attributes

A ∈ SExpr

♦ If $E_1 \ldots E_n$ ∈ SExpr, $n \geq 1$, f ∈ $\text{Dom}(E_1) \times \ldots \times \text{Dom}(E_n)$ T, T ∈ BasicTypes

$f(E_1, \ldots E_n)$ ∈ SExpr

Aggregate Expressions

♦ If E ∈ SExpr, g ∈ Set(Dom(E)) T, T ∈ BasicTypes

g(E) ∈ AExpr

♦ If $E_1 \ldots E_n$ ∈ AExpr, $n \geq 1$, f ∈ $\text{Dom}(E_1) \times \ldots \times \text{Dom}(E_n)$ T, T ∈ BasicTypes

$f(E_1, \ldots E_n)$ ∈ AExpr

Composed Expressions

♦ If E_1 ∈ SExpr, E_2 ∈ AExpr, $E_3 \ldots E_n$ ∈ Sexpr ∪ Aexpr, $n \geq 2$, f ∈ $\text{Dom}(E_1) \times \ldots \times \text{Dom}(E_n)$ T, T ∈ BasicTypes

$f(E_1, \ldots E_n)$ ∈ CExpr

The grammar user-functions are not explicitly defined, and can be restricted to any user-specific subset. The recursive way in which the grammar is defined, allows the definition of properties and functions, such as extract the attributes that compose an expression, or extract the attributes that are not part of an aggregation. These functions are useful for example to write SQL queries.

5.4. Selection constraints

As the attribute expression grammar distinguishes between different types of expressions, we can distinguish two types of selection constraints: simple constraints and aggregates constraints, depending on the use of aggregate expressions in the constraints. For example: constraints 1, 3 and 4 of section 4 example are simple constraints, but constraint 2 is an aggregate one because it uses the sum aggregation function.

Other sub-classifications can be done for adding more semantic to the constraint definition, for example make a difference between join constraints and user constraints.

The rest of the section presents the grammar for selection constraints.

Simple Constraints

♦ If $E_1 \ldots E_n \in \mathrm{SExpr}$, $n \geq 1$, $f \in \mathrm{Dom}(E_1) \times \ldots \times \mathrm{Dom}(E_n)$ boolean
$f(E_1, \ldots E_n) \in \mathrm{SConstr}$

Aggregate Constraints

♦ If $E_1 \in \mathrm{AExpr}$, $E_2 \ldots E_n \in \mathrm{Sexpr} \cup \mathrm{Aexpr}$, $n \geq 1$, $f \in \mathrm{Dom}(E_1) \times \ldots \times \mathrm{Dom}(E_n)$ boolean
$f(E_1, \ldots E_n) \in \mathrm{AConstr}$

As for attribute expressions, the grammar user-functions are not explicitly defined, and can be restricted to any user-specific subset, for example the classical algebraic operators set: $\{=, >, <, \geq, \leq, \neq\}$; and the recursive way in which the grammar is defined, allows the definition of properties and functions.

5.5. View Expressions' Modelization

As we introduce before, a view calculation expression can be defined given:

- A set of relations, that conform the input for the view calculation (from). We can use as input both source relations or another view.
- A calculation expression for each attribute of the view, called an attribute expression (select as). Calculation expressions can be of four types: value expressions, simple expressions, aggregated expressions or composed expressions.
- A set of selection constraints or predicates on the relations (where). These constraints don't use aggregate functions or operators.
- A set of selection constraints or predicates on the aggregations (having). These constraints use one or more aggregate functions.

6. Proposed DW logical meta-model

In this section we add some meta-classes to the DW logical meta-model shown in Figure 5 to represent the view calculation expressions. To better understand the proposal, we first present a simplified meta-model, which does not consider the possibly nesting view expressions. We will generalize the meta-model later.

Figure 10 shows the simplified meta-model.

For representing a view calculation expression we utilize the *view expression* meta-class, which has four components, corresponding to the definition given in section 5.5, with role names: *from*, *select*, *where* and *having*.

The *from* component indicates the set of source relations that are used to calculate the view.

The *select* component indicates the set of attribute calculation expressions used, each one corresponding to a view attribute. The *attribute expression* meta-class establishes a mapping between view and source attributes. The *as* component indicates the view attribute to be calculated, and the *component* component indicates the source attributes that are used to build the expression, following the grammar rules. Only expressions built from attributes of the source relations indicated by the *from* component are allowed.

The *where* and *having* components indicates the simple and aggregated constraints respectively. The *simple constraint* and *aggregate constraint* meta-classes are specializations of the *constraint* meta-class and represent the selection constraint grammar. The constraints are build beginning from the attribute of source relations. The *component* component indicates which attributes are used to build a constraint. Only constraints built from attributes of the source relations indicated by the *from* component are allowed.

A view can have one or more view calculation expressions, but it has to have a way for making the composition of these expressions. The *expression composition* meta-class corresponds to it.

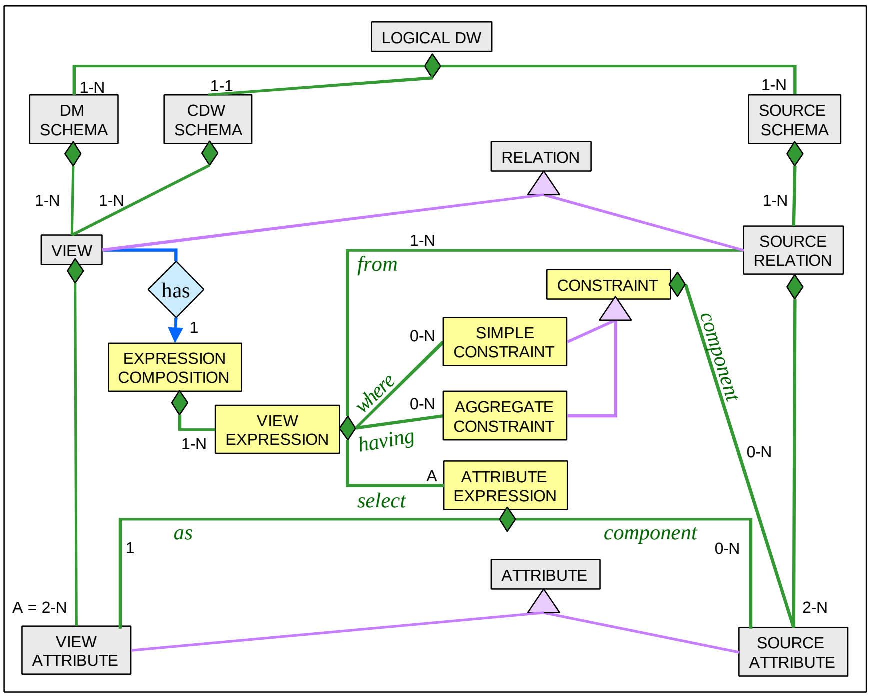


Figure 10 – A simple meta-model

To allow nested view expressions, a view should be calculated from others views. Then, the *from* component must represent both the source relations and the views. For this reason we generalize the *from* component to represent a set of relations. Figure 11 shows such generalization. Abnormal instances, such as a view calculated from itself, or loops in the *from* component must be controlled.

As a view can be calculated from any relation, also the attribute expressions and the constraints can be built from the attributes of these relations. Then, the *as* and *component* components are generalized, too.

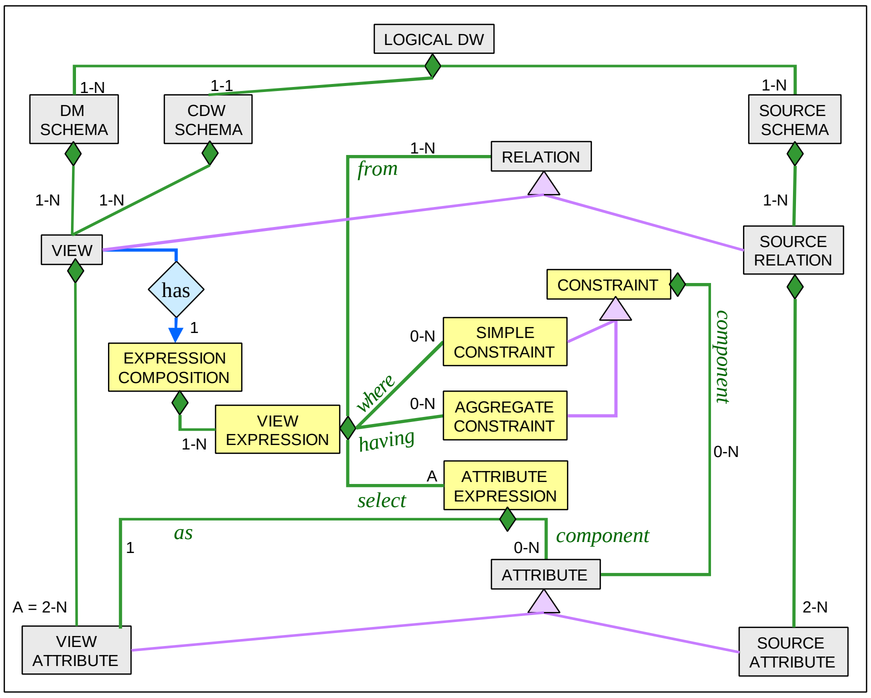


Figure 11 – A generalized meta-model

The *attribute expression* sub-meta-classes represents the four types of expressions defined in the grammar, namely: VExpr, SExpr, AExpr and CExpr. Each one can be extended to represent a specific calculation rule. For example, a simple expression (SExpr) can be either an attribute or a function of other simple expressions. The valid functions as well as their cardinality can be restricted for each application.

Figure 12 shows the class hierarchy for the attribute expressions.

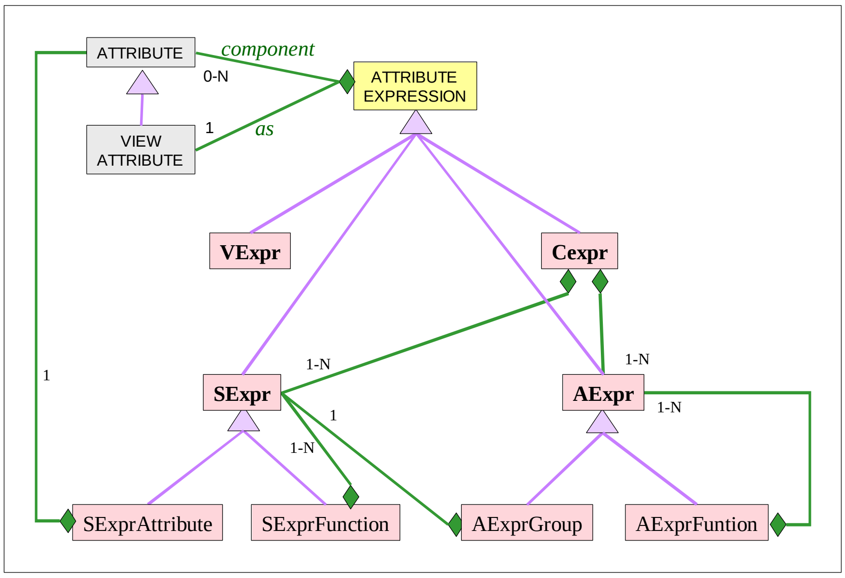


Figure 12 – Attribute expression meta-model

The same kind of hierarchy can be defined for the *constraint* and *expression composition* meta-classes.

7. Prototype

In this work, we develop a prototype of the meta-model manager. The prototype implements the logical meta-model and allows the creation and manipulation of meta-object instances, such as CDWs, sources and DMs and all its properties and components. The prototype also allows the definition of view calculation expressions and their components.

Some applications are currently being developed. They are discussed in section 7.3.

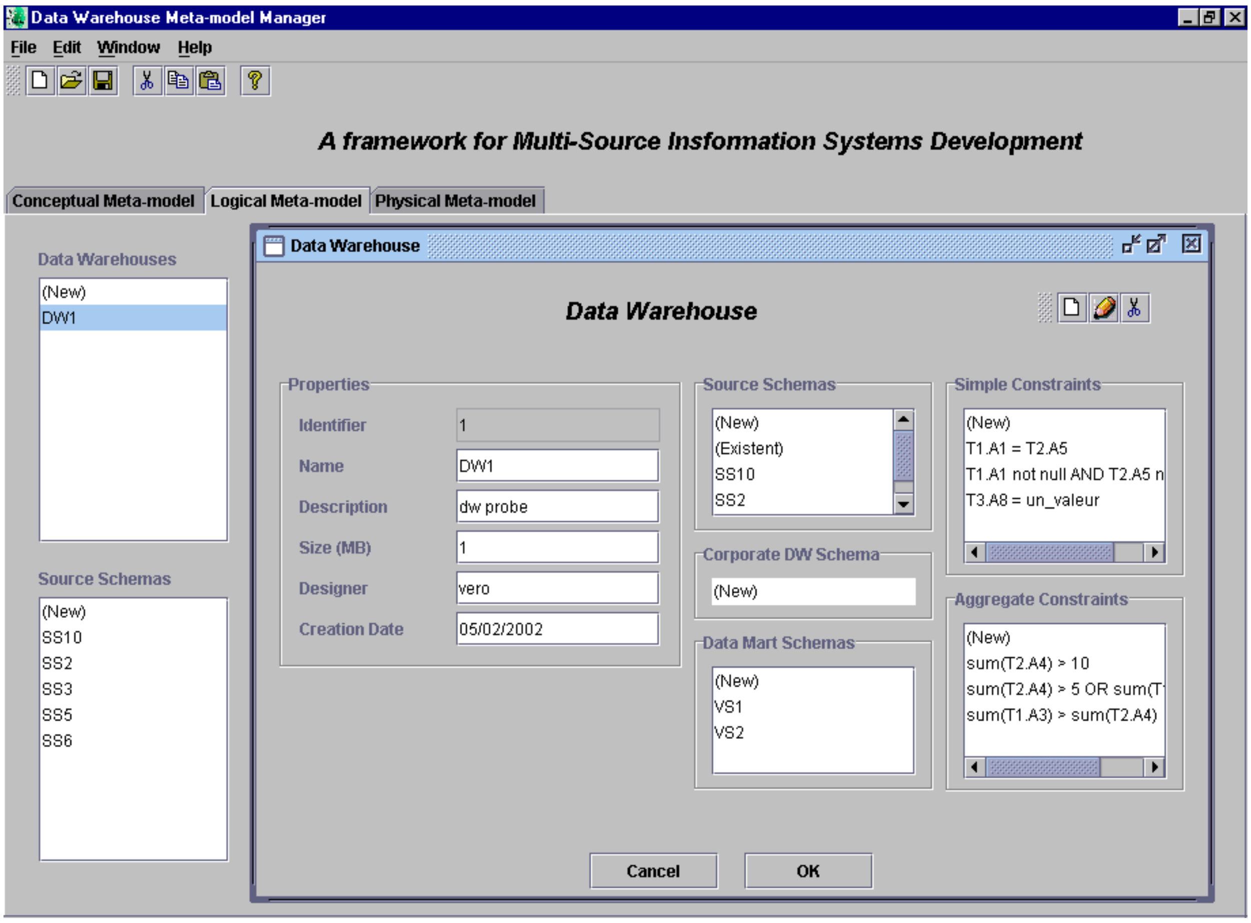
7.1. Description

The data-model manager goal is to manage and maintain all DW meta-data and make it accessible for the platform applications and tools.

By filling forms, the user can create, edit or delete the meta-objects. Each meta-object has a particular form, to easily fill in and update properties, and add or delete components.

Figure 13 shows the form for editing a DW. There'are three main sections: (i) a section to fill in general properties and descriptive information (*Properties*), (ii) a section to administer the DW schemas, namely sources (*Source Schemas*), CDW (*Corporate DW Schema*) and DMs (*Data Mart Schemas*), and (iii) a section to administer the general constraints, including simple and aggregate constraints (*Simple Constraints* and *Aggregate Constraints* respectively).

The forms' aspect preserve the appearance of a DW conceptual definition tool [Bou01].



Data Warehouses
(New)
DW1

Source Schemas
(New)
SS10
SS2
SS3
SS5
SS6

Data Warehouse

Properties

Identifier	1
Name	DW1
Description	dw probe
Size (MB)	1
Designer	vero
Creation Date	05/02/2002

Source Schemas
(New)
(Existent)
SS10
SS2

Corporate DW Schema
(New)

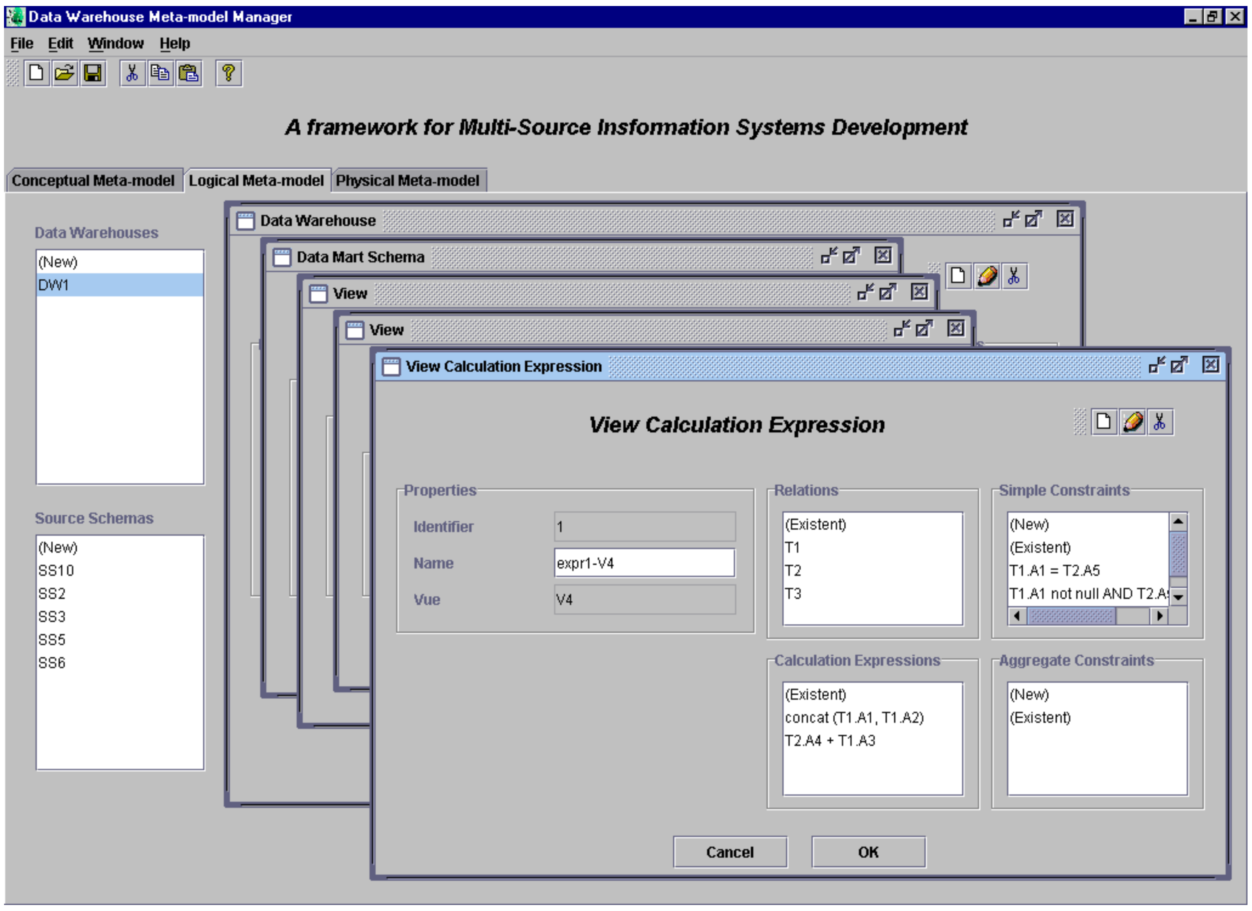
Data Mart Schemas
(New)
VS1
VS2

Simple Constraints
(New)
T1.A1 = T2.A5
T1.A1 not null AND T2.A5 n
T3.A8 = un_valeur

Aggregate Constraints
(New)
sum(T2.A4) > 10
sum(T2.A4) > 5 OR sum(T
sum(T1.A3) > sum(T2.A4)

Figure 13 – Data Warehouse Definition

All meta-objects have associated a similar frame to edit its properties or update its components. The user can easily do a cascade component edition. Figure 14 shows another frame for the edition of view calculation expressions.



Data Warehouses
(New)
DW1

Source Schemas
(New)
SS10
SS2
SS3
SS5
SS6

View Calculation Expression

Properties

Identifier	1
Name	expr1-V4
Vue	V4

Relations
(Existent)
T1
T2
T3

Simple Constraints
(New)
(Existent)
T1.A1 = T2.A5
T1.A1 not null AND T2.A

Calculation Expressions
(Existent)
concat (T1.A1, T1.A2)
T2.A4 + T1.A3

Aggregate Constraints
(New)
(Existent)

Figure 14 – View Calculation Expression Definition

The creation, delete or update of meta-objects is saved in a repository. At the moment, we use an Oracle® database as repository and have developed a small library of database access functions. In near future a repository manager can be integrated [Arz00].

7.2. Development

The prototype design follows object oriented strategies, and we used UML as specification language.

The prototype architecture has three levels, namely: *persistence*, *application* and *presentation*. This division in three levels gives presentation and persistence independence and allows an easy replacement of them. Figure 15 shows the three levels of the prototype architecture.

The *persistence* level assures data is saved in a more persistent media, such as a database, a metadata repository or a file. The actual implementation uses an Oracle® database as repository to assure persistence. The persistence level is divided in two sub-levels. The bottom level (*connection dependent*) is closely related with the repository. It solves the connection problems, and the common operations to insert, delete and update objects. The top level (*connection independent*) assures the independence with the chosen repository, allows an easy replacement of it, and offers a collection of useful access functions.

The *application* level embeds all the application logic. It implements all the meta-model objects and their relationships and provides the basic access funtionalities. This level is also responsible of checking the correct interaction between objects, for example preventing a source deletion if a DW uses it.

The *presentation* level resolves the user interaction with the *application* level. This interaction is carried out by graphical interfaces for creating and updating meta-objects. There is a form window (*frame*) for each application object and a main window (*main*) that includes all frames and dialog windows to ask for parameters (*dialog*).

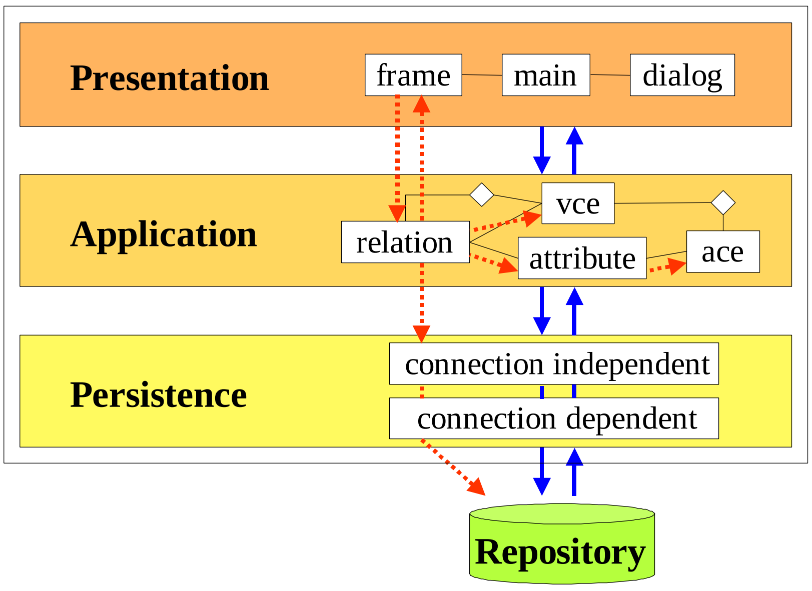


Figure 15 – Prototype architecture

The continuos arrows show the communication between the different levels. It allows an easy substitution of modules.

Each frame executes logic level object methods according to the user requirements. The object methods control the interaction with other objects and call the persistence methods. The persistence methods make the repository updates. This is represented in Figure 15 by the dotted arrows.

The implementation was made in Java, with Java Development Kit version 1.3 as compiler and Borland Jbuilder® version 4 as development platform. Implementation details can be found in [Per02].

7.3. Applications and tools

The present prototype only includes the implementation of the meta-model manager, but several applications and tools are being developed:

Discovering view calculation expressions:

A MSIS is composed of a set of sources and a set of views. A complex problem in MSIS design is how to calculate the view expressions accessing the existing sources. In [Ked99] an algorithm to solve the problem is proposed. It concerns the pertinent source identification, the study of possible calculation expressions from these sources and the generation of the most reasonable ones.

A related design tool can take the views and the existing source schemas as input, and automatically generate the view calculation expressions.

Automatic query rewriting using user profiles:

A user profile that expresses the user preferences, can be modeled like relational views. User requirements can be rewritten by integrating the profile views. These problems are addressed in a Ph.D. thesis [Lah02].

The related design tool use the meta-model to represent the profile views, the user requirements and the resulting query.

DW cleaning processes:

The cleaning and data transformation tasks allow to conform the source data before transferring it to the CDW or using it to calculate other data. Because of source heterogeneity, many problems can arise such as the identity of the objects belonging to different sources, the multiple values of these objects, the errors of data entry, the violation of integrity constraints, the differences in the measuring units (francs, euros, dollars) and value scales (thousands or million francs). These problems require the exploitation of the metadata which describe the semantics of the concepts and the relations used in each data source.

The problem is addressed in the context of a Ph.D. thesis [Sou02].

The associated tool takes the source and CDW schemas as input, and produces the calculation expressions as result. It also produces additional loading processes information.

CDW logical design and evolution:

[Mar00] address the CDW design problem as a sequence of schema transformations to the source schemas. It also considers the source schema evolution problem and how to propagate changes to the CDW. This problem is studied in a wrapper-mediator architecture [Mar01].

There is a tool for automatic CDW design and evolution propagation [Per01]. In near future, the integration of the tool to the platform can be done. To carry out this integration, we must modify the tool input, to read the source schemas information from the meta-model and the tool output, to add the generated CDW schema to the meta-model.

8. Conclusion

In this paper we have addressed the problem of model DW metadata from two points of view: the schema representation and the way of objects are calculated from another ones. We describe a DW logical meta-model that describes schema and calculation aspects and give a general algebra to manipulate calculation expressions.

The DW meta-model is a particular case of a MSIS meta-model. It is the heart of a global platform for MSIS development that supports the interoperation of different applications, process and tools that concern the development of MSIS.

The main results of our work are: (i) the specification of view calculation expressions, (ii) a grammar to represent attribute expressions, (iii) the definition of a DW meta-model that extends the existing one with the calculation expressions, and (iv) a prototype of the platform to support logical DW metadata.

The present prototype only includes the implementation of the DW logical perspective, but the conceptual and physical perspectives are being implemented. The integration of several tools and applications is also preview [Per01] [Lah02] [Sou02].

A more sophisticated repository interface has being developed [Arz00] and the use of a metadata manager is currently being studied. Graphical and web user interfaces are also being developed [Bar01] [Bit01] [Cal01] [Per01].

9. Bibliography

[Ada98] Adamson, C. Venerable, M.: "Data Warehouse Design Solutions". J. Wiley & Sons, Inc.1998.

[Arz00] G. Arzúa, G. Gil, S. Sharoian. "Manejador de Repositorio para Ambiente CASE". Under-graduate Project. Advisor: Raúl Ruggia. InCo, Universidad de la República, Uruguay, 2000.

[Aki99] Akinde, M. Böhlen, M.: "Constructing GPSJ View Graphs". Proc. 1st. Int. Workshop on Design and Management of Data Warehouses (DMDW). Heidelberg, Germany, 1999.

[Bar97] Baralis, E. Paraboschi, S. Teniente, E.: "Materialized view selection in a multidimensional database". Proc. 23rd. Int. Conf on Very Large Data Bases (VLDB). Athens, Greece, 1997.

[Bar01] Barcelo, L. Bartesaghi, M. Bettosini, F. Fagalde, B. Lezue, S. Michelena, L. Miranda, C. Polero, M. Rouiller, M. Saccone, M. Zubia, F.: "Implementación de herramientas CASE que asistan en el Diseño de Data Warehouses (Definición de Mapeos)". Software engineering project. Advisors: Jorge Triñanes, Verónika Peralta, Raúl Ruggia. InCo, Universidad de la República, Uruguay. 2001.

[Bat92] Batini, C. Ceri, S. Navathe, S.: "Conceptual Database Design. An Entity-Relationship Approach". Benjamin/Cummings Publishing. 1992.

[Bit01] Bittencourt, E. Borghi, D. Bravo, A. Camino, L. Giménez, H. González, M. Madera, M. Ravinovich, J. Roldós, G. Romano, X. Serra, F. Viera, M.: "Implementación de herramientas CASE que asistan en el Diseño de Data Warehouses (Interfaz Web)". Software engineering project. Advisor: Andrés Vignaga, Verónika Peralta, Raúl Ruggia. InCo, Universidad de la República, Uruguay. 2001.

[Boe99] 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.

[Bou99] 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.

[Bou00] 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.

[Bou01] Bouzeghoub, M. Lahlou, A. "Le Profiling en Intelligence d'Entreprise". Internal Report. PRISM, Université de Versailles Saint-Quentin-en-Yvelines. Versailles, Frances, 2001.

[Cal99] Calvanese, D. De Giacomo, G. Lenzerini, M. Nardi, D. Rosati, R.: "A Principled Approach to Data Integration and Reconciliation in Data Warehousing". Proc. 1st. Int. Workshop on Design and Management of Data Warehouses (DMDW). Heidelberg, Germany. 1999.

[Cal01] Calegari, D. Copette, D. Fajardo, F. Ferré, J. Kristic, G. López, R. López, P. Pizzorno, E. Sierra, N. Tobler, F. Viña, C.: "Implementación de herramientas CASE que asistan en el Diseño de Data Warehouses (Definición de Lineamientos)". Software engineering project. Advisor: Jorge Triñanes, Verónika Peralta, Raúl Ruggia. InCo, Universidad de la República, Uruguay. 2001.

[Gal01] Galhardas, H. Florescu, D. Shasha, D. Simon, E. Saita, C.: " Declarative Data Cleaning: Language, Model, and Algorithms". Proc. 27th. Int. Conf. on Very Large Data Bases (VLDB). Roma, Italy. 2001.

[Gar00] 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.

[Gol98] Golfarelli, M. Rizzi, S.:"Methodological Framework for Data Warehouse Design". Proc. 1st. Int. Workshop on Data Warehousing and OLAP (DOLAP). Bethesda, USA, 1998.

[Gol00] 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.

[Gra96] 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.

[Gup95] 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.

[Gup97] Gupta, H.: "Selection of Views to Materialize in a Data Warehouse". Proc. Int. Conf. on Database Theory. Delphy, Greece, 1997.

[Inm96] Inmon, W.: "Building the Data Warehouse". John Wiley & Sons, Inc. 1996.

[Inm96a] Inmon, W.: "Building the Operational Data Store". John Wiley & Sons, Inc. 1996.

[Jar97] Jarke, M. Vassiliou, Y.: "Data Warehouse Quality Design: A Review of the DWQ Project". In Proc. MIT Conf. on Information Quality, UK, 1997.

[Jar99] Jarke, M. Lenzerini, M. Vassiliou, Y. Vassiliadis, P.: "Fundamentals of Data Warehouses". Springer-Verlag, 1999.

[Ked99] 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.

[Ked99b] 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.

[Kim96] Kimball, R.:"The Datawarehouse Toolkit". John Wiley & Son, Inc., 1996.

[Lah02] Lahlou, A. "Integration du profil utilisateur dans les requêtes des bases de données". PhD Thesis. PRISM, Université de Versailles Saint-Quentin-en-Yvelines. Versailles, France. On going work.

[Lig99] 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.

[Mar00] Marotta, A.: “Data Warehouse Design and Maintenance through Schema Transformations”. Master Thesis. InCo - Pedeciba, Universidad de la República. Montevideo, Uruguay, 2000.

[Mar01] Marotta, A. Motz, R. Ruggia, R.: “Managing Source Schema Evolution in Web Warehouses”. Proc. 1st. Int. Workshop on Information Integration on the Web (WIIW). Rio de Janeiro, Brazil, 2001.

[Mot02] Motz, R.: “Dynamic Maintenance of an Integrated Schema”. PhD Thesis. IPSI-GMD, Darmstadt University of Technology, Darmstadt, Germany. On going work.

[Per01] Peralta, V.: “Diseño lógico de un Data Warehouses a partir de un Esquema Conceptual Multidimensional”. Master Thesis. InCo - Pedeciba, Universidad de la República. Montevideo, Uruguay, 2001.

[Per02] Peralta, V.: “A Plataform for Data Warehouse Design”. Technical Report. Prism, Université de Versailles Saint-Quentin-en-Yvelines. Versailles, France, 2002.

[She90] Sheth, A. Larson, J.: “Federated database systems and managing distributed, heterogeneous, and autonomous databases”. ACM Computing Surveys, 22(3):183-226, 1990.

[Sil97] Silverston, L. Inmon, W. Graziano, K.: “The Data Model Resource Book”. John Wiley & Sons, Inc. 1997.

[Sou02] Soukane, A. “Generation d’Expressions de requetes de Netoyage de Données”. PhD Thesis. PRISM, Université de Versailles Saint-Quentin-en-Yvelines. Versailles, France. On going work.

[The97] Theodoratos, D. Sellis, T.: “The Data Warehouse Configuration”. Proc. 23rd. Int. Conf. on Very Large Data Bases (VLDB). Athens, Greece, 1997.

[The99] 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.

[Vid01] Vidal, V. Lóscio, B. Salgado, A.: “Using correspondence assertions for specifying the semantics of XML-based mediators”. Proc. 1st. Int. Workshop on Information Integration on the Web (WIIW). Rio de Janeiro, Brazil, 2001.

[Vdo01] Vdovjak, R. Houben, G.: “RDF-based architecture for semantic integration of heterogeneous information sources”. Proc. 1st. Int. Workshop on Information Integration on the Web (WIIW). Rio de Janeiro, Brazil, 2001.

[Wie92] Wiederhold, G.: “Mediators in the architecture of future information systems”. IEEE Computer, 25(3):38-49, 1992.

[Yan97] 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.

[Yan01] Yang, L. Miller, R. Haas, L. Fagin, R.: “Data-Driven Understanding and Refinement of Schema Mappings”. Proc. ACM SIGMOD Conf. Santa Barbara, USA, 2001.

[Zhu97] Zhuge, Y. Garcia-Molina, H. Wiener, J. : “Consistency Algorithms for Multi-Source Warehouse View Maintenance”. Journal of Distributed and Parallel Databases, July 1997.