Using Perspective Schemata to Model the ETL Process

Valéria M. Pequeno, João Carlos G. M. Pires

Abstract—Data Warehouses (DWs) are repositories which contain the unified history of an enterprise for decision support. The data must be Extracted from information sources, Transformed and integrated to be Loaded (ETL) into the DW, using ETL tools. These tools focus on data movement, where the models are only used as a means to this aim. Under a conceptual viewpoint, the authors want to innovate the ETL process in two ways: 1) to make clear compatibility between models in a declarative fashion, using correspondence assertions and 2) to identify the instances of different sources that represent the same entity in the real-world.

This paper presents the overview of the proposed framework to model the ETL process, which is based on the use of a reference model and perspective schemata. This approach provides the designer with a better understanding of the semantic associated with the ETL process.

Keywords—conceptual data model, correspondence assertions, data warehouse, data integration, ETL process, object relational database.

I. Introduction

NOWADAYS enterprises have an increasing need to take decisions as fast and accurate as possible. The competition is very high and who holds reliable and strategic information in the right moment has the advantage. One way to achieve such a goal is by making use of data warehousing, which provides processes and technologies to build integrated data repositories called Data Warehouses (DWs). The data in those repositories represents the unified history of an enterprise at a suitable level of detail to be useful for analysis [1]. The data warehouse will act as a source for a wide range of applications such as OLAP (On-Line Analytical Processing), data mining and other advanced analysis techniques. A simplified architecture of a DW system is presented in Fig. 1.

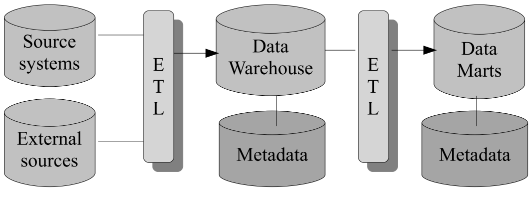

Fig. 1. Simplified data warehouse architecture.

In architecture presented in Fig. 1, the data must be Extracted from different information sources, Transformed and integrated to be Loaded (ETL) into the DW. DW data is then delivered to Data Marts (DM), probably with some more changes. DMs are subsets of DW data designed to serve specific demands of a particular group of users. Moreover, either a DW or a DM should be created and accessed through metadata that provides detailed documentation for data in the DW system, such as applied transformations and origin of data.

The ETL process is the major stage in any DW system. It deals with the complexity inherent in the multiplicity of autonomous and heterogeneous data sources, such as where the data is and which data represents the same concept. Although important, the ETL tasks are made in an ad hoc fashion, most of them by different automatic (or semi-automatic) tools while others are manual. Some of these tools were developed in the academic world (e.g. ARKTOS tool and IBIS tool), and others in the market (e.g. Sunopsis and BO) – see [2] for a survey. However, they either do not deal with conceptual modelling or they use ad hoc formalism.

Nowadays, these problems are the hardest to deal with because DWs are growing rapidly in two ways. Firstly, DW increases in data volume from 20 to 100 terabytes or even pedabytes [3]. Secondly, DW expands data models complexity (both in sources and in DW), which implies the rise of the difficulty of managing and understanding these models [4], [5]. Although there are specialized tools with graphical interface to do the mapping between the source information and the DW system, they are mostly procedural. It means that the knowledge of procedures and policies are hidden in diverse codes, which are totally dependent on experts and technicians. These tools focus strongly on data movement, as the models are only used as a means to this aim.

In an ETL process for building a DW system, it is not concerned just with mapping between schemata, but also with an effective data integration that expresses a unified view of the enterprise. In this context, it is crucial to have a conceptual reference model [6], [3], [7]. A Reference Model (RM) is an abstract framework that provides a common semantic that can be used to guide the development of other models and help with data consistency [3].

There are, mainly, two approaches to build a DW, the *top-down* and the *bottom-up*. In the top-down approach [1] a DW is first built followed by DMs. On the other hand, the bottom-up approach [8] starts with the DMs to get the DW. Nevertheless, in spite of the seeming differences, “both approaches collect data from source systems into a single data store (named *data warehouse* in the top-down approach and *staging area* in the bottom-up one), from which DMs are populated [9].”

V.M. Pequeno is with CENTRIA, Departamento de Informtica, Faculdade de Ciências e Tecnologia, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal (e-mail:vmp@fct.unl.pt).

J.C.G.M. Pires is with CENTRIA, Departamento de Informtica, Faculdade de Ciências e Tecnologia, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal (e-mail:jmp@di.fct.unl.pt).

In addition, both strategies need to keep historical information of *component entities* and *classification entities*, as well as keep information about events that occur in the business (stored in *transaction entities* [10]).[1] Both approaches, earlier or later, store this information on a multidimensional form; moreover, in the top-down approach, the data integration is done using a normalized model with extentions to keep historical information. The ETL process is present in both approaches, although the execution of it differs between the two strategies. In this research, both approaches are equally applicable.

In order to consider the growing complexity of source and DW data models, it is proposed to take a declarative approach, which is based on making explicit the relationship between data sources and data warehouse taking into account the Reference Model, independently of the ETL process involved. Furthermore, the ETL process itself can use this information.

II. Overview

The present proposal offers a way to express the existing data models (source, DW, DM, RM) and the relationship between them. The approach is based on:

1) *Schema language* (L_S), which is used to describe the actual data models (source, DW, DM, RM). The formal framework focuses on an object-relational paradigm, which includes definitions adopted by the main concepts of object and relational models as they are widely accepted in literature – cf. [11], [12].
2) *Perspective schema language* (L_{PS}) is used to describe *perspective schemata*. A perspective schema is a special kind of schema that describes a data model (part or whole) (*target schema*) in terms of other data models (*base schemata*). L_{PS} mainly extends L_S with two components: Correspondence Assertions (CAs) and Matching Functions (MFs). Correspondence Assertions formally specify the relationship between schema components. Matching functions indicate when two data entities represent the same instance of the real world. L_{PS} includes data transformations, such as names conversion and data types conversion. When the target schema is described in the scope of a perspective schema, instead of just referring an existing schema, the perspective schema is called *view schema*.

Fig. 2 illustrates the basic components of the proposed architecture and their relationships. The schemata **RM**, **DW**, **DM**, $\mathbf{S}_1$,...,$\mathbf{S}_n$, $\mathbf{S'}_1$, $\mathbf{S'}_2$ represent, respectively, the reference model, the data warehouse, a data mart, the source schemata $\mathbf{S}_1$,...,$\mathbf{S}_n$, the view schemata $\mathbf{S'}_1$ (a viewpoint of schema $\mathbf{S}_1$), and $\mathbf{S'}_2$ (an integrated viewpoint of schemata $\mathbf{S}_2$ and $\mathbf{S}_3$). The relationship between the RM and the other schemata is shown through the perspective schemata $\mathbf{P}_{s'1|RM}$,..., $\mathbf{P}_{s'n|RM}$,$\mathbf{P}_{RM|DW}$ (denoted by the solid arrows). Once the perspective schemata $\mathbf{P}_{s'1|RM}$,..., $\mathbf{P}_{s'n|RM}$,$\mathbf{P}_{RM|DW}$ is declared, a new perspective schema ($\mathbf{P}_{s'1,s'2,...,s'n|DW}$) between the DW schema and the source schemata (designed by a dotted arrow) can be deduced. This inferred perspective schema shows the direct relationship between the DW and its source information, and can be used to automatically materialize the ETL process. All schemata (including the perspective ones) are stored in a metadata repository.

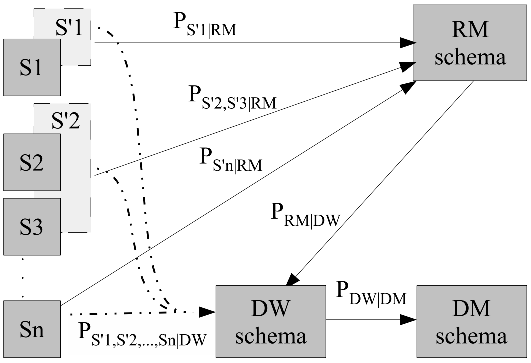

Fig. 2. Proposed architecture.

The remainder of this paper is laid out as follows. Sections III and IV illustrate, through examples, the proposal discussed in this paper: Section III shows the languages to describe schemata and perspective schemata, and Section IV presents the process of inference. Section V briefly reviews related work on conceptual models for DWs and for ETL. The paper ends with Section VI, which points out the new features of the approach presented here and in ongoing or planned future work on this topic.

III. Schema Languages

This Section introduces the languages used to describe the schemata and perspective schemata.

In the remainder of the paper, consider a simple sales scenario comprising a data source $\mathbf{S}_1$, a reference model **RM**, and a data warehouse **DW**. The schemata are shown in Fig. 3. $\mathbf{S}_1$ and **RM** include information about sales of products, being that in ITEM.**unitprice**$_{\mathrm{S}_1}$ the values are stored in dollars while in SALE_ITEM.**unitprice**$_{\mathrm{RM}}$ they are stored in euros. **DW** contains historical information about customers and products, as well as summarized information regarding sales. The relation **DW**.SALE_ITEM stores the amount (**prodsold_amt**$_{\mathrm{DW}}$) and the quantity total (**prodsold_qty**$_{\mathrm{DW}}$) of each product (**prod_id_sk**$_{\mathrm{DW}}$) sold per customer (**cust_id_sk**$_{\mathrm{DW}}$) and per sale date (**date_id_sk**$_{\mathrm{DW}}$). The historical data is kept using three properties: **start_date**, **end_date**, and **current_flag**. **Start_date** and **end_date** indicate the historical range of when each tuple was current. **Start_date** stores the date when the tuple was inserted into the relation. **End_date** stores the date when the tuple is no longer current, or the null value in case it is still current. **Current_flag** store true or false indicating if the tuple is the current one or not. The properties **prod_id_sk**$_{\mathrm{DW}}$, **cust_id_sk**$_{\mathrm{DW}}$, and **date_id_sk**$_{\mathrm{DW}}$ are surrogate keys automatically generated by the system. Though, it is not shown in Fig. 3, all relations have keys and some of them have foreign keys.

“Component entities define the details or ’components’ of each business transaction” while “classification entities represent hierarchies embedded in the data model, which may be collapsed into component entities [...] [10].”

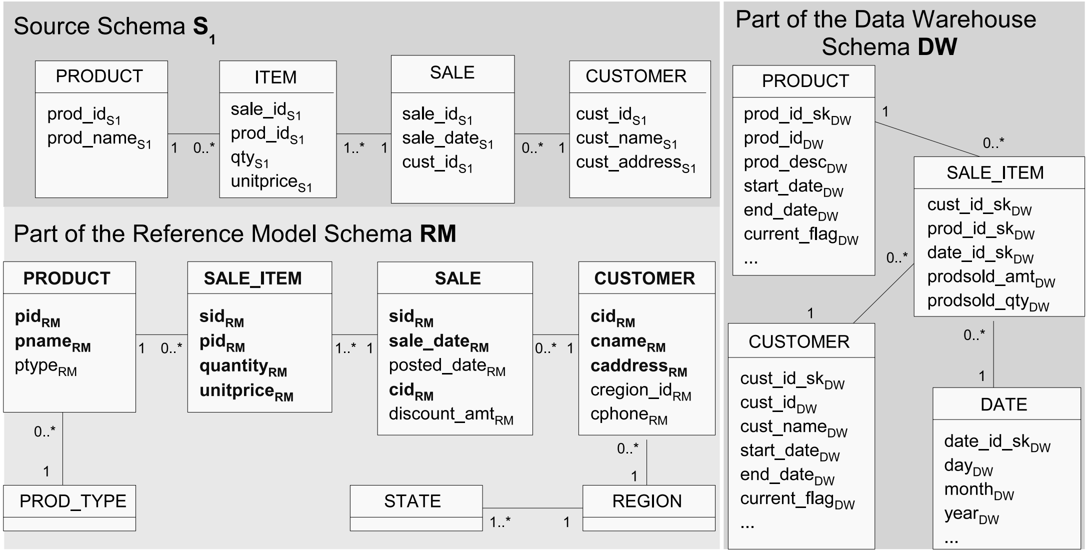

Fig. 3. Motivating example.

A. Basic Schema Language

The schema language L_S is used to describe all actual schemata (source, DW, RM, and DM) of a DW system. It is general, flexible and expressive enough to allow the projection of traditional and non-traditional applications, and so enable the manipulation of complex data – cf. [4], [13], [5].

This language includes definitions adopted by the main concepts of object and relational models as they are widely accepted in literature. Thus, L_S includes the notions of type, typed value, object, tuple, property (in the relational literature, it is named *attribute* [14].), class, relation, signature method, method, key, foreign key, schema, state of a schema, and path expression. All these concepts are formally defined in [15] and are not explained here. However, they are close to the normal definition. Instead of the formal definition, we present an example to clarify the notation of L_S:

Example 1: The relation PRODUCT of the schema **S**$_1$ presented in Fig. 3 has the following declaration in L_S:

(PRODUCT, {**prod_id**$_{S1}$:integer, **prod_name**$_{S1}$:string})

which is formed by (<relation name>,<relation type>).

B. Perspective Schema Language

The language L_{PS} is used to define perspective schemata. A perspective schema describes a data model, part or whole (*target schema*), in terms of other data models (*base schemata*). The language L_{PS} includes new concepts beyond those in language L_S:

- to express the subset of the target schema that will be necessary in the perspective schema;
- to determine when two objects/tuples are distinct representations of the same object in the real-world (named the *instance matching* problem);
- to establish the semantic correspondence between schemata's components.

The target schema may have much more information than it is required to represent in a perspective schema, namely when the target schema is the Reference Model. Hence, it is required to clearly indicate which elements of the target schema are in the scope of the perspective schema. This is done in L_{PS} using 'require' declarations. For instance, consider the perspective schema $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ between the schemata **RM** (the target schema) and **S**$_1$ (the base schema) both as presented in Fig. 3. For this perspective schema, the following relations from **RM** are needed:

- require(PRODUCT,{**pid**$_{RM}$,**pname**$_{RM}$})
- require(SALE_ITEM, {**sid**$_{RM}$, **pid**$_{RM}$, **quantity**$_{RM}$, **unitprice**$_{RM}$})
- require(SALE, {**sid**$_{RM}$, **sale_date**$_{RM}$, **cid**$_{RM}$})
- require(CUSTOMER, {**cid**$_{RM}$, **cname**$_{RM}$, **caddress**$_{RM}$})

Note that, for instance, the properties: **ptype**$_{RM}$ from **RM**.PRODUCT, **posted_date**$_{RM}$ and **discount_amt**$_{RM}$ from **RM**.SALE, and **cregion_id**$_{RM}$ and **cphone**$_{RM}$ from **RM**.CUSTOMER are not shown as being required.

1) Matching Functions: From a conceptual viewpoint, it is essential to provide a way to identify instances of different schemata that represent the same entity in the real-world. The proposal here is to use matching functions, which can include various techniques for matching instances, including some of those used in data cleaning, such as lookup tables, user-defined functions, heuristics and past matching. These functions, as occur in [16], define a 1:1 correspondence between the objects/tuples in families of corresponding classes/relations. In particular, this work is based on the following matching function signature:

$$\mathbf{match} : ((\mathbf{S}_1 \left[\mathrm{R}_1 \right], \tau_1) \times (\mathbf{S}_2 \left[\mathrm{R}_2 \right], \tau_2)) \rightarrow \mathrm{Boolean}, \qquad (1)$$

being $\mathbf{S}_i$ schema names, R_i class/relation names, and τ_i the data type of the instances of R_i, for i $\in$ {1,2}. When both arguments are instanced, **match** verifies whether two instances

match(($\mathbf{S}_1$ [PRODUCT],τ_1)×(**RM**[PRODUCT],τ_2))→Boolean
match((**RM**[PRODUCT],τ_2)×(**DW**[PRODUCT (**current_flag** = true)],τ_3))→Boolean

Fig. 4. Examples of matching function signatures.

are semantically equivalent or not. If only one argument is instanced, e.g. $\mathbf{S}_1.\mathrm{R}_1$, then it obtains the semantically equivalent $\mathbf{S}_2.\mathrm{R}_2$ instance of the given $\mathbf{S}_1.\mathrm{R}_1$ instance, returning true when it is possible, and false when nothing is found or when there is more than one instance to match.

In some scenarios one-to-many correspondence between instances are common, e.g. when historical data is stored in the DW. In this case, a variant of **match** should be used, which has the following form:

$$\mathbf{match} : ((\mathbf{S}_1\left[\mathrm{R}_1\right],\tau_1)\times(\mathbf{S}_2\left[\mathrm{R}_2\left(\mathbf{predicate}\right)\right],\tau_2))\rightarrow \text{Boolean.} \tag{2}$$

predicate is a boolean condition that determines the context in which the instance matching must be applied in $\mathbf{S}_2.\mathrm{R}_2$.

Some examples of matching functions signatures involving schemata of Fig. 3 are presented in Fig. 4. The implementation of the matching functions shall be externally provided, since their code is very close to the application domain.

2) Correspondence Assertions: The semantic correspondence between schemata's components is declared in this proposal through the Correspondence Assertions (CAs), which are used to formally assert the correspondence between schema components in a declarative fashion. CAs are classified in four groups: Property Correspondence Assertion (PCA), Extension Correspondence Assertion (ECA), Summation Correspondence Assertion (SCA), and Aggregation Correspondence Assertion (ACA). Examples of CAs are shown in Fig. 5 and explained in that Section.

Property CAs relate properties of a target schema to the properties of base schemata. They allow for dealing with several kinds of semantic heterogeneity such as: *naming conflict* (for instance synonyms and homonyms properties), *data representation conflict* (that occurs when similar contents are represented by different data types), and *encoding conflict* (that occurs when similar contents are represented by different formats of data or unit of measures). Furthermore, it can declare: i) Boolean conditions where the property's value depends on the validation of one (or several) condition(s); ii) calculations where the value of a property is the result of a calculation involving two or more properties of a same instance.

For example, the PCAs ψ_1 and ψ_3 (see Fig. 5) deal with, respectively, naming conflict and encoding conflict. ψ_1 links the property $\mathbf{pid}_{RM}$ of **RM**.PRODUCT to the property $\mathbf{prod\_id}_{S1}$ of $\mathbf{S}_1$.PRODUCT. ψ_3 assigns $\mathbf{unitprice}_{RM}$ of SALE_ITEM to $\mathbf{unitprice}_{S1}$ of ITEM using the function **dollarTOeuro** to convert currencies from dollars (stored in $\mathbf{unitprice}_{S1}$) to euros (stored in $\mathbf{unitprice}_{RM}$).

The Extension CAs are used to describe which objects/tuples of a base schema should have a corresponding semantically equivalent object/tuple in the target schema. For instance, the relation **RM**.PRODUCT is linked to relation S_1.PRODUCT through the ECA ψ_4 presented in Fig. 5. ψ_4 determines that **RM**.PRODUCT and $\mathbf{S}_1$.PRODUCT are equivalent, i.e., for each tuple of PRODUCT of the schema S_1 there is one semantically equivalent tuple in PRODUCT of the schema **RM**, and vice-versa.

There are five different kinds of ECAs: equivalence, selection, difference, union, and intersection. The ECA of union is not close to union of sets, rather it indicates a relation similar to the natural outer-join of the usual relational models. For instance, consider the view schema S_v with the relation PRODUCT(**code, description, category**) which is related to two relations: PRODUCT in S_1 (in Fig. 3), and PROD(**code, description,category**) in schema S_2 (not presented in any figure) through the ECA ψ_5 shown in Fig. 5. ψ_5 determines that PRODUCT in S_v is the union/join of PRODUCT in S_1 and PROD in S_2, i.e., for each tuple of PRODUCT of the schema S_1 there is one semantically equivalent tuple in PRODUCT of the schema S_v, or for each tuple of PROD of the schema S_2 there is one semantically equivalent tuple in PRODUCT of the schema S_v, and vice-versa.

In an ECA, any relation/class can appear with a selection condition, which determines the subset of instances of the class/relation being considered. This kind of ECA is especially important to the DW because through it the current instances of the DW can be selected and related to the instances of their sources (which usually do not have historical data). For example, consider the ECA ψ_6 presented in Fig. 5. ψ_6 determines that a subset of instances of the relation **DW**.PRODUCT, whose value of the property $\mathbf{current\_flag}_{\mathrm{DW}}$ is true, is the same as those instances of the relation **RM**.PRODUCT.

The Summation CAs are used to describe the summary of a class/relation whose instances are related to the instances of another class/relation by breaking them into logical groups that belong together. There are two kinds of SCAs: *groupby* and *normalize*. Both group instances are based on one or more properties, but *groupby* is used to indicate that some type of aggregate function will be used and *normalize* is used to indicate a normalization process. For example, the instances of the relations **DW**.SALE_ITEM and **RM**.SALE_ITEM are connected through the SCA ψ_7 displayed in Fig. 5. ψ_7 determines that **DW**.SALE_ITEM is the grouping of **RM**.SALE_ITEM based on values of $\mathbf{pid}_{\mathrm{RM}}$, **RM**.SALE_ITEM•$\boldsymbol{FK}_1$•$\mathbf{cid}_{\mathrm{RM}}$, and **RM**.SALE_ITEM • $\boldsymbol{FK}_1$• $\mathbf{sale\_date}_{\mathrm{RM}}$.[2] In other words, there is a tuple in **DW**.SALE_ITEM for each group of tuples in **RM**.SALE_ITEM that have the same value for product ($\mathbf{pid}_{\mathrm{RM}}$), customer (**RM**.SALE_ITEM•$\boldsymbol{FK}_1$•$\mathbf{cid}_{\mathrm{RM}}$) and sale date (**RM**.SALE_ITEM•$\boldsymbol{FK}_1$•$\mathbf{sale\_date}_{\mathrm{RM}}$).

In order to exemplify a SCA of normalization, consider the source schema $\mathbf{S}_2$ (not presented in

[2] **RM**[SALE_ITEM] • $\boldsymbol{FK}_1$ • $\mathbf{cid}_{\mathrm{RM}}$ and **RM**[SALE_ITEM] • $\boldsymbol{FK}_1$ • $\mathbf{sale\_date}_{\mathrm{RM}}$ are path expressions, with $\boldsymbol{FK}_1$ being a foreign key of **RM**.SALE_ITEM that refers to **RM**.SALE. These paths means that there is a link through $\boldsymbol{FK}_1$ from which is obtained, respectively, the customer identity (**RM**.SALE.$\mathbf{cid}_{\mathrm{RM}}$) and the value of sale date (**RM**.SALE.$\mathbf{sale\_date}_{\mathrm{RM}}$).

Property Correspondence Assertions (PCAs):
ψ_1: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ [PRODUCT] • $\mathbf{pid}_{\mathrm{RM}} \rightarrow \mathbf{S}_1$ [PRODUCT] • $\mathbf{prod\_id}_{\mathrm{S}_1}$
ψ_2: $\mathbf{P}_{\mathbf{RM|DW}}$ [PRODUCT] • $\mathbf{prod\_id}_{\mathrm{DW}} \rightarrow \mathbf{RM}$ [PRODUCT] • $\mathbf{pid}_{\mathrm{RM}}$
ψ_3: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ [SALE_ITEM] • $\mathbf{unitprice}_{\mathrm{RM}} \rightarrow$ **dollarTOeuro** $\big(\mathbf{S}_1$ [ITEM] • $\mathbf{unitprice}_{\mathrm{S}_1}\big)$

Extension Correspondence Assertions (ECAs):
ψ_4: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ [PRODUCT] $\rightarrow \mathbf{S}_1$ [PRODUCT]
ψ_5: $\mathbf{S}_v$ [PRODUCT] $\rightarrow \mathbf{S}_1$ [PRODUCT] $⟗$ $\mathbf{S}_2$ [PROD]
ψ_6: $\mathbf{P}_{\mathbf{RM|DW}}$ [PRODUCT ($\mathbf{current\_flag}_{\mathrm{DW}} = \mathrm{True}$)] $\rightarrow \mathbf{RM}$ [PRODUCT]

Summation Correspondence Assertions (SCAs):
ψ_7: $\mathbf{P}_{\mathbf{RM|DW}}$ [SALE_ITEM] ($\mathbf{prod\_id\_sk}_{\mathrm{DW}}, \mathbf{cust\_id\_sk}_{\mathrm{DW}}, \mathbf{date\_id\_sk}_{\mathrm{DW}}$) $\rightarrow$ groupby ($\mathbf{RM}$ [SALE_ITEM] ($\mathbf{pid}_{\mathrm{RM}}$, $\boldsymbol{FK}_1 \bullet \mathbf{cid}_{\mathrm{RM}}, \boldsymbol{FK}_1 \bullet \mathbf{date\_id\_sk}_{\mathrm{RM}}$))
ψ_8: $\mathbf{P}_{\mathbf{S}_2|\mathbf{RM}}$ [PRODUCT] ($\mathbf{pid}_{\mathrm{RM}}$) $\rightarrow$ normalize ($\mathbf{S}_2$ [PRODUCT_SALES] ($\mathbf{product\_number}_{\mathrm{S2}}$))

Aggregation Correspondence Assertions (ACA):
ψ_9: $\mathbf{P}_{\mathbf{RM|DW}}$ [SALE_ITEM] • $\mathbf{prodsold\_qty}_{\mathrm{DW}} \rightarrow \psi_7$, **sum** ($\mathbf{RM}$ [SALE_ITEM] • $\mathbf{quantity}_{\mathrm{RM}}$)
ψ_{10}: $\mathbf{P}_{\mathbf{RM|DW}}$ [SALE_ITEM] • $\mathbf{prodsold\_amt}_{\mathrm{DW}} \rightarrow \psi_7$, **sum** ($\mathbf{RM}$ [SALE_ITEM] • $\mathbf{quantity}_{\mathrm{RM}} \times \mathbf{RM}$ [SALE_ITEM] • $\mathbf{unitprice}_{\mathrm{RM}}$)

Fig. 5. Examples of correspondence assertion.

any figure), which contains a denormalized relation PRODUCT_SALES($\mathbf{product\_number}_{\mathrm{S2}}$, $\mathbf{product}_{\mathrm{S2}}$, $\mathbf{quantity}_{\mathrm{S2}}$, $\mathbf{price}_{\mathrm{S2}}$, $\mathbf{purchase\_order}_{\mathrm{S2}}$) and the schema **RM** presented in Fig. 3. PRODUCT_SALES holds information about sold items in a purchase order as well as information logically related to products themselves, which could be in another relation, occurring in schema **RM**. The SCA ψ_8, displayed in Fig. 5, determines the relationship between PRODUCT_SALES and **RM**.PRODUCT when a normalization process is involved, i.e., it determines that **RM**.PRODUCT is a normalization of $\mathbf{S}_2$.PRODUCT_SALES based on distinct values of property $\mathbf{product\_number}_{\mathrm{S2}}$.

This research also deals with denormalizations, which is defined using *path expressions* (component of the language $\mathrm{L}_{\mathcal{S}}$).

The Aggregation CAs link properties of the target schema to the properties of the base schema when a SCA is used. When the SCA is of groupby, the ACAs may contain aggregation functions. This proposal only deals with aggregate functions supported by most of the queries languages, like SQL-99 [14], i.e. *summation, maximum, minimum, average* and *count*; although more complex aggregation is supported in some object-relational databases – cf. [17].

The ACAs, similar to the PCAs, allow for the description of several kinds of situations; therefore, the aggregate expressions can be more detailed than simple property references. Calculations performed can include, for example, ordinary functions (such as sum or concatenate two or more properties' values before applying the aggregate function), and Boolean conditions (e.g. count all male students whose grades are greater or equal to 10).

Returning to the motivating example, the ACAs ψ_9 and ψ_{10} (displayed in Fig. 5) link the properties of **DW**.SALE_ITEM and **RM**.SALE_ITEM. ψ_9 determines that the value of the property $\mathbf{prodsold\_qty}_{\mathrm{DW}}$ is the summation of the quantity of product sold for each customer for each date, being that the grouping is obtained in ψ_9 by ψ_7. ψ_{10} determines that the value of the property $\mathbf{prodsold\_amt}_{\mathrm{DW}}$ is the amount of product sold for each customer for each date. In ψ_{10} the grouping is obtained by ψ_7, as occurs in ψ_9, and the amount of product sold is calculated using the formula: *price* × *quantity*.

IV. Inferring new perspective schemata

After the perspective schemata has been defined, the next step is to infer a new perspective schema, based on all perspective schemata prior defined (see Fig. 6). This inferred perspective will connect the DW directly to its sources and can be used to automatically materialize the ETL process. For example, consider the perspective schemata $\mathbf{P}_{\mathbf{RM|DW}}$ and $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$; it is possible to infer the perspective schema $\mathbf{P}_{\mathbf{S}_1|\mathbf{DW}}$, which directly connects the DW to the source $\mathbf{S}_1$. $\mathbf{P}_{\mathbf{S}_1|\mathbf{DW}}$ will have the same classes, relations, keys, and foreign keys as defined in original $\mathbf{P}_{\mathbf{RM|DW}}$; and new CAs and new match function signatures will be created.

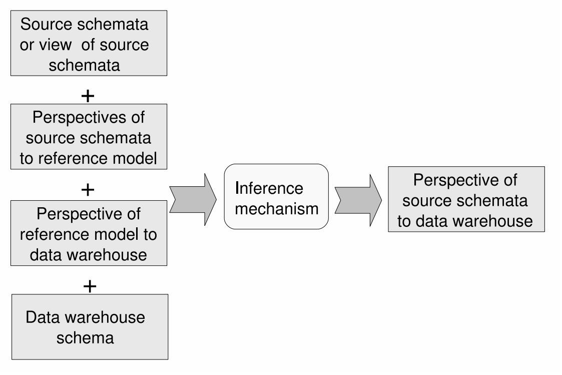
Source schemata or view of source schemata
+
Perspectives of source schemata to reference model
+
Perspective of reference model to data warehouse
+
Data warehouse schema
Inference mechanism
Perspective of source schemata to data warehouse

Fig. 6. Sketch of the creation of an inferred perspective schema.

In this proposal, the deduction of new perspective schemata are done automatically by the mechanism of inference, which is a rule-based rewriting system. It is formed by a set of rules having the general form:

$$\textbf{Rule:}\frac{A \Rightarrow B}{C} \qquad (\text{read A is rewritten in B if C is valid}), \qquad (3)$$

being C one or more conditions that should be satisfied.

The inference rules are recursive, since in C can exists references to other inference rules. There are 39 rules, which can be divided in rules for rewriting CAs (RR-CAs), rules for rewriting matching functions (RR-MFs), and rules for rewriting components that are presents in CAs or in matching

(a) CAs of the perspective $\mathbf{P_{S1}}$ | **DW**:

$\psi'_1 : \mathbf{P}_{\mathbf{S}_1|\mathbf{DW}}\,[\text{PRODUCT}] \bullet \mathbf{prod\_id}_{\text{DW}} \rightarrow \mathbf{S}_1\,[\text{PRODUCT}] \bullet \mathbf{prod\_id}_{\text{S}_1}$

$\psi'_2 : \mathbf{P}_{\mathbf{S}_1|\mathbf{DW}}\,[\text{PRODUCT}\,(\mathbf{current\_flag}_{\text{DW}} = \text{True})] \rightarrow \mathbf{S}_1\,[\text{PRODUCT}]$

(b) Process of inference of $\psi'1$:

$$\begin{array}{l} \psi_2\text{: } \mathbf{P}_{\mathbf{RM}|\mathbf{DW}}\,[\text{PRODUCT}] \bullet \mathbf{prod\_id}_{\text{DW}} \rightarrow \mathbf{RM}\,[\text{PRODUCT}] \bullet \mathbf{pid}_{\text{RM}} \Rightarrow \\ \psi'_1 : \mathbf{P}_{\mathbf{S}_1|\mathbf{DW}}\,[\text{PRODUCT}] \bullet \mathbf{prod\_id}_{\text{DW}} \rightarrow \mathbf{S}_1\,[\text{PRODUCT}] \bullet \mathbf{prod\_id}_{\text{S}_1} \\ \hline \psi_1\text{: } \mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}\,[\text{PRODUCT}] \bullet \mathbf{pid}_{\text{RM}} \rightarrow \mathbf{S}_1\,[\text{PRODUCT}] \bullet \mathbf{prod\_id}_{\text{S}_1} \end{array}$$

Fig. 7. Examples of inferred correspondence assertions and a CA-rewriting rule.

function signatures. Based on these rules, the system will generate all possible CAs to the new perspective as well as all match function signatures.

```
procedure INFER($P_T, P_1, ..., P_n, P_I$)
    for each $CA$ in $P_T.caList$ do
        find all new $CAs$ that are rewritten from $CA$ using
        some CA-rewriting rule;
        add $CAs$ to $P_I.caList$;
    end for
    for each $MF$ in $P_T.mfList$ do
        find all new $MFs$ that are rewritten from $MF$
        using some MF-rewriting rule;
        add $MFs$ to $P_I.mfList$
    end for
    for each $E$ in $classList/relationList/keyList$ do
        create a require declaration to $P_I$;
        add it, appropriately, to $P_I.classList/$
        $P_I.relationList/P_I.keyList$
    end for
end procedure
```

Fig. 8. The pseudo-code to the inference mechanism.

A pseudo-code with the essence of the process to generate a new perspective is shown in Fig. 8. In Fig. 8 P_T is a perspective schema from the reference model to the data warehouse; P_j, 1 $\neq$ j $\neq$ n, are perspective schemata from source schemata to the reference model; and P_I is the inferred perspective schema from source schemata to the data warehouse. All elements of the perspective schemata are grouped in lists: $classList$, $relationList$, $keyList$, $caList$, and $mfList$. The three first lists hold require declarations of, respectively, classes, relations, and keys and foreign keys. $caList$ contains correspondence assertion declarations, and $mfList$ has match function signatures.

The inference mechanism can be illustrated more clearly through the running example. For each CA, originally defined to $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$ (relating elements of the **DW** to elements of the **RM**) will exist a new CA in $\mathbf{P}_{\mathbf{S}_1|\mathbf{DW}}$. Fig. 5 only presents some CAs of $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ and of $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$. Based on these CAs, two new CAs can be inferred: ψ'_1 and ψ'_2 shown in Fig. 7(a). The process to deduce ψ'_1 is presented in Fig. 7(b).

For each match function signature originally defined to $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$ will exists a new match function signature in $\mathbf{P}_{\mathbf{S}_1|\mathbf{DW}}$. Fig. 4 only presents one match function signature of $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$. Based on this match function signature and on the ECAs presented in Fig. 5, a new match function signature can be created, shown in Fig. 9.

The inference mechanism has been developed as part of a proof-of-concept prototype using a Prolog language. Beside this module, the prototype consist of more five modules, such as the *schema manager*, and the *ISCO translator*. The *schema manager* module is employed by the designer to manage the schemata (in language L_S) as well as the perspective schemata (in language L_{PS}). The *ISCO translator* performs the mapping between schemata written in L_S or L_{PS} languages to schemata. defined in a language programming called Information Systems COnstruction language (ISCO) [18]. ISCO is based on a contextual constraint logic programming that allows the construction of information systems. It can define (object) relational schemata, represent data, and transparently access data from various heterogeneous sources in a uniform way, like a mediator system [19]. Thus, it is possible to access data from information sources using the perspective schema in ISCO. Furthermore, once the perspective schema from source schemata to the global schema has been inferred, as well as the new match functions have been implemented, it can be translated to ISCO language and so the data of the global schema can be queried.

Another advantage of the proposed work is that a particular kind of schema evolution in a DW is transparent (when changes occur in the source schemata or new ones are added), since it is enough to automatically generate a new perspective schema from the source schemata to the data warehouse one.

V. Related Work

This Section addresses related work in the fields of conceptual modelling for data warehousing and ETL.

Available literature quotes that the conceptual models for DW have focused on technical aspects such as multidimensional data models (e.g. [20], [21], [4], [22], [23], [24]) as well as the materialized view definition and maintenance (e.g. [25]). In particular, the most conceptual multidimensional models are extensions to the Entity-Relationship model (e.g. [26], [27], [28], [29]) or extensions to UML (e.g. [30], [31], [32]). There are only some works involving conceptual models based on non-multidimensional aspects [33], [25], [6].

There are various approaches in existence [34], [6], [35] for dealing with the ETL activities in a conceptual setting. So far, the authors of this paper are not aware of any research that precisely deals with mappings (structural and instance) between the sources and the DW, and with the problem of semantic heterogeneity in a whole conceptual level.

$$\begin{array}{l} \mathbf{match}((\mathbf{RM}[\textsc{product}],\tau_2)\times(\mathbf{DW}[\textsc{product}\ (\mathbf{current\_flag} = \text{true})],\tau_3))\rightarrow\text{Boolean} \Rightarrow \\ \mathbf{match}((\mathbf{S}_1\ [\textsc{product}],\tau_2)\times(\mathbf{DW}[\textsc{product}\ (\mathbf{current\_flag} = \text{true})],\tau_3))\rightarrow\text{Boolean} \\ \hline \psi_4\text{: } \mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}\ [\textsc{product}] \rightarrow \mathbf{S}_1\ [\textsc{product}] \end{array}$$

Fig. 9. Example of a rule for rewriting a match function signature.

Reference in [6] presented a framework adopted in the Data Warehouse Quality project. Similar to this study, their proposal includes a reference model (cited as "enterprise model") designed using an Enriched Entity-Relationship (EER) model. However, unlike the authors' research, all their schemata, including the DW schema, were formed by relational structures, which were defined as views over the reference model. Their proposal provided the user with various levels of abstraction: conceptual, logical, and physical. In their conceptual level, they introduce the notion of intermodel assertions that precisely capture the structure of an EER schema or allow for the specifying of the relationship between diverse schemata. However, any transformation (e.g. restructuring of schema and values) or mapping of instances is deferred for the logical level. In addition, they did not deal with complex data, integrity constraints, and path expressions, as this research does.

Reference in [34] proposed a conceptual model for dealing with inter-relationships of attributes and concepts and with the ETL activities in the early stages of a DW project. Their research included a rich graphical notation and dealt with several kinds of transformation presented in the usual ETL process, such as surrogate key transformation, checks for null values, primary key violations, aggregate values and data conversion. However, they did not mention the matching of instances, neither did their work have any mechanisms to verify if the conceptual model was legal or not.

Reference in [35] focuses on an ontology-based approach to determine the mapping between attributes from the source schemata and the DW schema, as well as to identify the ETL transformations required for correctly moving data from source information to the DW. Their ontology, based on a common vocabulary as well as a set of data annotations (both provided by the designer), allows formal and explicit description of the semantic of the sources and the DW schemata. However, their strategy requires a deep knowledge of all schemata involved in the DW system, in what is usually not an usual task. It is dispensable in the proposal presented by the authors of the current paper as each schema (source or DW) needs to be related only to the reference model one. Additionally, in [35] there is nothing about the matching of instances and their ETL operations being a subset of transformations treated by us.

VI. Conclusions and Future Work

This paper proposes a declarative approach to make explicit the relationship between data sources and the Data Warehouse (DW), not only at a structural level, but also at an instance level, independently of the ETL process involved. The approach considers the use of a Reference Model (RM) and can be divided in three steps:

1) to describe all source schemata, the DW schema, a Data Mart (DM) schema, and the (RM) schema, using the schema language L_S.
2) to describe perspective schemata to relate the schemata declared in step 1, using the perspective schema language L_{PS}.
3) to infer a new perspective schema from source schemata to the DW schema based on all perspective schemata defined in the step 2.

It has been shown how the actual (DW, DM, RM, sources) and perspective schemata can be described in the proposed language and how new perspective can be deduced. This process was illustrated with some examples. An advantage to this approach is that it provides designers/users with a better understanding of the semantic associated with the ETL process. Moreover, the designers can describe the DW without concerns about where the sources are or how they are stored. It is possible because all schemata in the DW system are related to the RM through the perspective schemata, and based on the latter, the direct relationship between the DW and its sources is automatically generated. Moreover, the schema evolution, when the source schemata changes or new ones are added, becomes transparent to the DW, due to use of the reference model and of the inference process.

A prototype Prolog-based has been developed to allow the description of schemata and perspective schemata in the proposed language as well as to infer new perspective schemata based on other ones. The matching functions can be implemented using Prolog itself or external functions. In addition, the prototype include translators from the proposed language to the ISCO one. ISCO [18] allows access to heterogeneous data sources and to perform arbitrary computations. Thus, user-queries can be done, in a transparent way, to access the information sources, like occurs in mediator systems [19].

A mediator strategy in a data warehouse context is not appropriate. It is due to particular nature of the DW, which uses huge data volume and require that user-queries can be answered quickly and efficiently. Thus, for future work, investigations will be made into how the perspective schemata can be used to automate the materialization of the ETL process. Another important direction for future work is the development of a graphical user-friend interface to declare the schemata in the proposed language, and thus, to hide some syntax details.

References

[1] W. H. Inmon, *Building the data warehouse*, 4th ed. Wiley Publishing, 2005.

[2] R. F. Raminhos, "ETL state of the art," New University of Lisbon, Tech. Rep., June 2007, unplished.

[3] C. Imhoff, N. Galemmo, and J. G. Geiger, *Mastering Data Warehouse Design - Relational and Dimensional Techniques*. Wiley Publishing, 2003.

[4] J. M. Pérez, R. Berlanga, M. J. Aramburu, and T. B. Pedersen, “A relevance-extended multi-dimensional model for a data warehouse contextualized with documents,” in *DOLAP’05: Proc. of the 8th ACM Intl. Workshop on Data Warehousing and OLAP*. USA: ACM, 2005, pp. 19–28.

[5] R. Matias and J. Moura-Pires, “Revisiting the olap interaction to cope with spatial data and spatial data analysis,” in *ICEIS 2007 - Proc. of the 9th Intl. Conf. on Enterprise Information Systems*, J. Cardoso, J. Cordeiro, and J. Filipe, Eds., vol. DISI, 2007, pp. 157–163.

[6] D. Calvanese, L. Dragone, D. Nardi, R. Rosati, and S. M. Trisolini, “Enterprise modeling and data warehousing in TELECOM ITALIA,” *Inf. Syst.*, vol. 31, no. 1, pp. 1–32, 2006.

[7] R. Knackstedt and K. Klose, “Configurative reference model-based development of data warehouse systems,” *Idea group publishing*, vol. Managing Modern Organizations through Information Technology, pp. 32–39, 2005.

[8] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, and B. Becker, *The Data Warehouse Lifecycle Tookit*, 2nd ed. Wiley Publishing, 2008.

[9] W. Eckerson, “Four ways to build a data warehouse,” *What works*, vol. 15, 2003. [Online]. Available: http://www.tdwi.org/research/display.aspx?id=6699.

[10] D. L. Moody, “From enterprise models to dimensional models: A methodology for data warehouse and data mart design,” in *Proc. of the Intl. Workshop on Design and Management of Data Warehouses*, 2000.

[11] E. F. Codd, “A relational model of data for large shared data banks,” in *Communications of the ACM*, 1970, pp. 377–387.

[12] R. G. Cattell and D. Barry, Eds., *The Object Database Standard ODMG 3.0*. Morgan Kaufmann Publishers, 2000.

[13] T. B. Pedersen, “Warehousing the world: a few remaining challenges,” in *DOLAP’07: Proc. of the ACM 10th intl. workshop on data warehousing and OLAP*. USA: ACM, 2007, pp. 101–102.

[14] R. Elmasri and S. B. Navathe, *Fundamentals of database systems*, 5th ed. Pearson Education, 2006.

[15] V. M. Pequeno and J. C. G. M. Pires, “A formal object-relational data warehouse model,” New University of Lisbon, Tech. Rep., November 2007.

[16] G. Zhou, R. Hull, and R. King, “Generating data integration mediators that use materialization,” *J. Intell. Inf. Syst.*, vol. 6(2/3), pp. 199–221, May 1996.

[17] IBM, *DB2 version 9.1 for z/OS - SQL reference*, 6th ed. IBM Corporation, December 2008.

[18] S. Abreu and V. Nogueira, “Using a logic programming language with persistence and contexts,” in *Declarative Programming for Knowledge Management, 16th intl. conf. on applications of declarative programming and knowledge management, INAP 2005, Japan. Revised Selected Papers.*, ser. Lecture Notes in Computer Science, O. Takata, M. Umeda, I. Nagasawa, N. Tamura, A. Wolf, and G. Schrader, Eds., vol. 4369. Springer, 2006, pp. 38–47.

[19] G. Wiederhold, “Mediators in the architecture of future information systems,” in *IEEE Computer*, vol. 25(3), 1992, pp. 38–49.

[20] D. Dori, R. Feldman, and A. Sturm, “From conceptual models to schemata: An object-process-based data warehouse construction method,” *Inf. Syst.*, vol. 33, no. 6, pp. 567–593, 2008.

[21] E. Malinowski and E. Zimányi, “A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models,” *Data knowl. eng.*, vol. 64, no. 1, pp. 101–133, 2008.

[22] M. Golfarelli, V. Maniezzo, and S. Rizzi, “Materialization of fragmented views in multidimensional databases,” *Data Knowl. Eng.*, vol. 49, no. 3, pp. 325–351, 2004.

[23] B. Husemann, J. Lechtenborger, and G. Vossen, “Conceptual data warehouse modeling,” in *Design and Management of Data Warehouses*, 2000, p. 6.

[24] S. Rizzi, “Conceptual modeling solutions for the data warehouse,” *In Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications*, vol. Information Science Reference, pp. 208–227, 2008.

[25] R. Wrembel, “On a formal model of an object-oriented database with views supporting data materialisation,” in *Proc. of the Conf. on Advances in Databases and Information Systems*, 1999, pp. 109–116.

[26] E. Franconi and A. Kamble, “A data warehouse conceptual data model,” *Proc. of the Int. Conf. on Scientific and Statistical Database Management*, vol. 00, pp. 435–436, 2004.

[27] A. S. Kamble, “A conceptual model for multidimensional data,” in *APCCM’08: Proc. of the 15th on Asia-Pacific Conf. on Conceptual Modelling*. Australia: Australian Computer Society, Inc., 2008, pp. 29–38.

[28] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter, “Extending the E/R model for the multidimensional paradigm,” in *Proc. of the Workshops on Data Warehousing and Data Mining*, 1999, pp. 105–116.

[29] N. Tryfona, F. Busborg, and J. G. B. Christiansen, “starER: a conceptual model for data warehouse design,” in *DOLAP ’99: Proc. of the 2nd ACM Intl. Workshop on Data warehousing and OLAP*. USA: ACM, 1999, pp. 3–8.

[30] S. Luján-Mora, J. Trujillo, and I.-Y. Song, “A UML profile for multidimensional modelling in data warehouses,” *Data Knowl. Eng.*, vol. 59, no. 3, pp. 725–769, 2005.

[31] T. B. Nguyen, A. M. Tjoa, and R. Wagner, “An object oriented multidimensional data model for OLAP,” in *Web-Age Inf. Management*, 2000, pp. 69–82.

[32] J. Trujillo, M. Palomar, and J. Gomez, “Applying object-oriented conceptual modeling techniques to the design of multidimensional databases and OLAP applications,” *WAIM’00. Lecture Notes in Computer Science (LNCS)*, vol. 1846, pp. 83–94, 2000.

[33] F. Ravat and O. Teste, “A temporal object-oriented data warehouse model,” in *Proc. of the Int. Workshop on Database and Expert Systems Applications*, 2000, pp. 583–592.

[34] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, “Conceptual modeling for ETL processes,” in *DOLAP’02: Proc. of the 5th ACM Intl. Workshop on Data Warehousing and OLAP*. USA: ACM, 2002, pp. 14–21.

[35] D. Skoutas and A. Simitsis, “Designing ETL processes using semantic web technologies,” in *DOLAP’06: Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*. USA: ACM, 2006, pp. 67–74.