USING PERSPECTIVE SCHEMATA TO MODEL THE ETL PROCESS

Valéria Magalhães Pequeno João Carlos Gomes Moura Pires

[1]Departamento de Informática, Centro de Inteligência Artificial
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Quinta da Torre 2829 516, Caparica, Portugal

keywords: ETL process, conceptual data model, data warehouse, data integration, correspondence assertions, object-relational database.

Abstract

Data Warehouses (DWs) are repositories which contain the unified history of an enterprise at a suitable level of detail for decision support. The data must be Extracted from heterogeneous information sources, Transformed and integrated to be Loaded (ETL) into the DW, using ETL tools. These tools focus strongly on data movement, being that the models only are used as a way for this aim. Under a conceptual viewpoint, we 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 our formal language that deals with matching of both structures and instances. This approach has the advantage of giving designers/users a better understanding of the semantic associated with the ETL process.

1. INTRODUCTION

Nowadays enterprises have a growing 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 the organisation at a suitable level of detail to be useful for analysis [1]. A simplified architecture of a DW system is presented in Figure 1. In this architecture, 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.

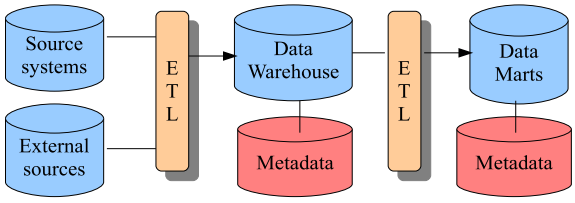

Figure 1: Simplified data warehouse architecture.

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) specialised tools while others are manual. Some 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.

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 specialised 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 way for 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 express a unified view of the enterprise. In that 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[1], from which DMs are populated" [9]. However, in this research there is no distinction made between one or the other.

In order to consider the growing complexity of source and DW data models, it is proposed 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.

2. OVERVIEW

The present proposal offers a way to express the existing data models (source, DW, DM, RM) and the relationship between them. In order to do this two languages are suggested:

1. *Schema language* (L_S), which is used to describe the actual data models (source, DW, DM, RM). This 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. [10, 11]. Thus, L_S includes the notions of type, typed value, object, tuple, property[2], class, relation, signature method, method, key, foreign key, schema, state of a schema, and path expression. All these concepts were formally defined in [13] and will not be explained here, since they are close to the normal

[1] named *data warehouse* in the top-down approach and *staging area* in the bottom-up one.

[2] In the relational literature, properties are normally named *attributes* [12].

definition.

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*.

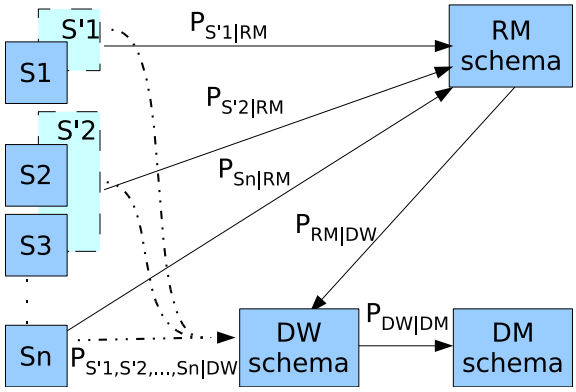

Figure 2: Proposed architecture.

Figure 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 materialise the ETL process. All schemata (including the perspective ones) are stored in a metadata repository.

The aim of this paper is to introduce formally the language L_{PS}, its structure and integrity constraints, as well as illustrate, through examples, the mechanism of inference to deduce new perspectives.

The remainder of this paper is laid out as follows. Sections 3 and 4 illustrate the proposal discussed in this paper: Section 3 shows the language to describe perspective schemata, and Section 4 presents the process of inference. Section 5 briefly reviews related work on conceptual models for DWs and for ETL. The paper ends with Section 6, which points out the new features of the approach presented here and in ongoing or planned future work on this topic.

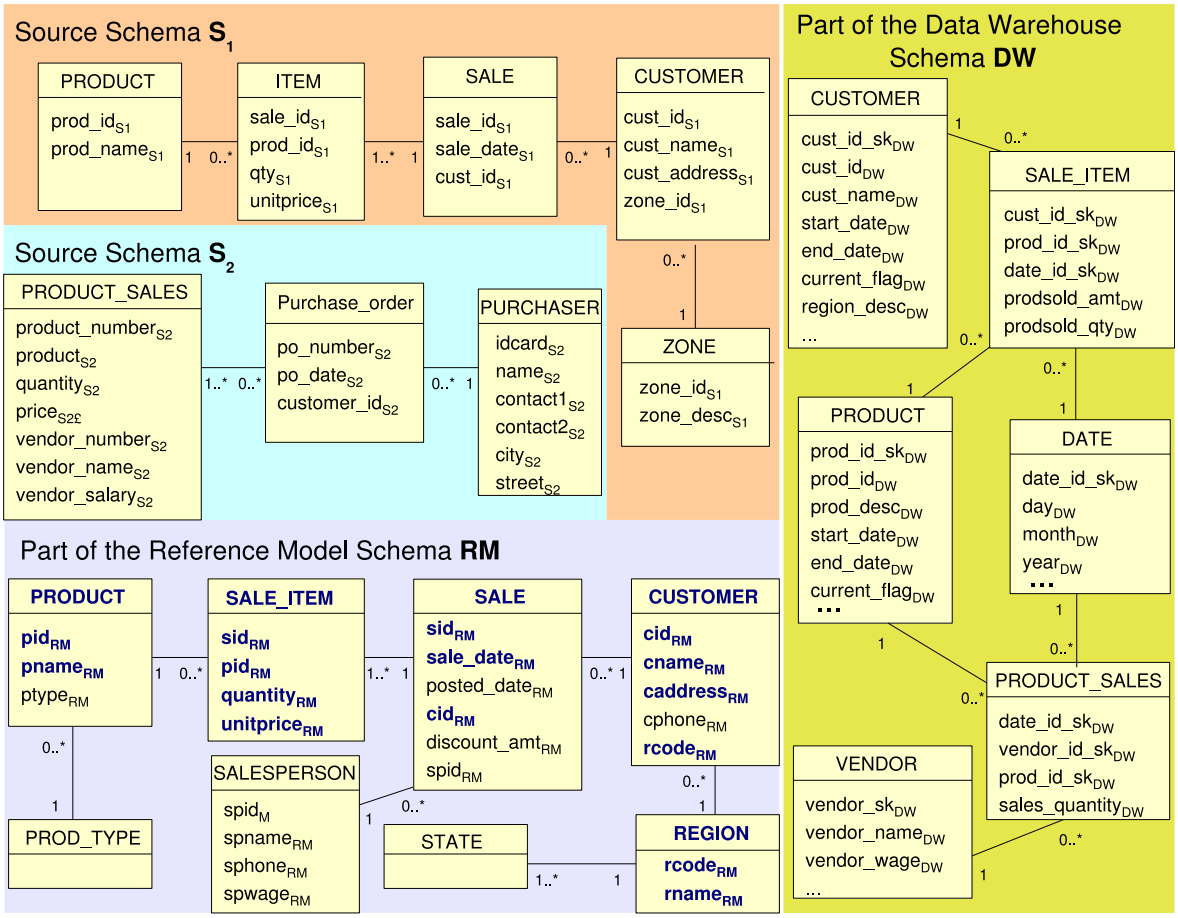

Figure 3: Motivating example.

3. PERSPECTIVE SCHEMA LANGUAGE

In the remainder of the paper, considers a simple sales scenario comprising two data sources $\mathbf{S}_1$ and $\mathbf{S}_2$, a reference model **RM**, and a data warehouse **DW**. The schemata are shown in Figure 3. $\mathbf{S}_1$, $\mathbf{S}_2$, and **RM** include information about sales of products, being that in ITEM.$\mathbf{unitprice}_{S_1}$ the values are stored in dollars while in SALE_ITEM.$\mathbf{unitprice}_{RM}$ they are stored in euros. PRODUCT_SALES holds information about sold items in a purchase order as well as informations logically related to products themselves and salespeople. **DW** contains, besides historical information about customers, products and vendors, summarised information regarding sales. The relation **DW**.SALE_ITEM stores the amount ($\mathbf{prodsold\_amt}_{DW}$) and the quantity total ($\mathbf{prodsold\_qty}_{DW}$) of each product ($\mathbf{prod\_id\_sk}_{DW}$) sold per customer ($\mathbf{cust\_id\_sk}_{DW}$) and per sale date ($\mathbf{date\_id\_sk}_{DW}$). The relation **DW**.PRODUCT_SALES stores the total ($\mathbf{sales\_quantity}_{DW}$) of products sold ($\mathbf{prod\_id\_sk}_{DW}$) per employee ($\mathbf{vendor\_id\_sk}_{DW}$) and sale date ($\mathbf{date\_id\_sk}_{DW}$). 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 data when the tuple was inserted into the relation. **End_date** stores the date when the tuple becomes no longer current, or the null value when the tuple has been recently updated. **Current_flag** store true or false indicating if the tuple is the current one or not. The properties $\mathbf{prod\_id\_sk}_{DW}$, $\mathbf{cust\_id\_sk}_{DW}$, $\mathbf{vendor\_id\_sk}_{DW}$, and $\mathbf{date\_id\_sk}_{DW}$ are surrogate keys automatically generated by the system. In addition, it is not shown in Figure 3, but all relations have keys and some of them have foreign keys.

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. It is done in L_{PS} using 'require' declarations, which may be of one of two forms:

$$\text{require}\,(< \text{name} >)\,, \text{or} \tag{1}$$

$$\text{require}\,(< \text{name} >, < \text{properties} >) \tag{2}$$

being that <name> in item (1) can be the name of a key or of a foreing key, while in item (2) <name> can be the name of a class or of a relation. <properties> is a set of property names belonging the class or relation indicated in <name>.

For instance, consider the perspective schema $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ between the schemata **RM** (the target schema) and $\mathbf{S}_1$ (the base schema) both presented in Figure 3. For this perspective schema, the following relations from **RM** are needed:

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

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

3.1. Matching Functions

In a data integration environment, there is the need to combine information from a same or different sources. This involves comparing the instances from the sources and attempt to determine when two instances of differet schemata refer to the same real-world entity. Here it is called the *instance matching* problem. Variants of this problem are known as *field matching* [14], *merge/purge* problem [15], *object matching* [16, 17], *record linkage* [18], *data matching* [19], *tuple matching* [19], *object fusion* [20], among others.

The management of matching instance is inevitable for controlling of redundancy and turning possible the data integration. Thus, it plays a central role in many information contexts, including data warehousing. Unfortunately, instance matching usually is expensive to compute due to the complex

structure of the schemata and the character of the data. So, the identification of matching instances can require complex tests to identify the equivalence between instances. "The inference that two data items represent the same domain entity may depend upon considerable statistical, logical, and empirical knowledge of the task domain" [15].

The instance matching problem has received much attention in the database, AI, KDD, and WWW communities, mainly covering complex scenaries, in order to improve the quality of data. Thus, they consider to use techniques of identification of instances between multiple sources for removing duplicates and for correcting format inconsistencies in the data (e.g. [21, 22, 23, 24, 25, 26, 27, 28]). Here data in study already is filtered and cleaned.

There are many works in the literature to address the instance matching problem in context here desired. Most systems assume that a universal key is available for performing the instance matching, or at least suppose that the objects (or tuples) share the same set of properties. In this case, they match objects (or tuples) by comparing property similarity between the shared properties. For instance, suppose that in Figure 3 the properties **idcard**$_{S2}$ (of relation $\mathbf{S}_2$.CUSTOMER) and **cust_id**$_{S1}$ (of relation $\mathbf{S}_1$.CUSTOMER) have the same content (the identity card number of each customer) and are primary key of its respective relation. Thus, it is trivial to make the instance matching, being enough to compare the value of these properties. The works in [15, 29, 14, 30], for instance, use this approach. Even though this approach is largely used, is not easy to implement it due to, among other things, the semantic heterogeneity that exists between the several schemata and the heterogeneity of data types.

Another trivial manner to deal with instance matching uses look-up tables. In this technique the types of the classes (or relations) can be disjoints, but there is a look-up table that hold matching information that determines the matching between the objects (or tuples). For example, a look-up table can be provided to determine the equivalence of the various products from the two relations $\mathbf{S}_1$.PRODUCT and $\mathbf{S}_2$.PRODUCT_SALES. This approach requires that the equivalence between the instances are identified and managed, and is also subject to faults since usually the table is not automatically updated. Look-up tables are used, for instance, in [17, 31, 32].

References [17, 31, 32] also consider the problem of instance matching in contexts where universal keys are not available. For instance, [17] deals with user-defined boolean functions to determine the equivalence between the objects. This function receives some input arguments and returns true or false. For instance, two people, a customer and an employee, are the same person if their names are the "closeness" the same. A matching function can be specified to determine the "closeness" of the names, what takes two names as arguments and returns a boolean value.

References [16, 18] exploit external informations, like for example: constraints, negative keys, and past matching, to maximise matching accuracy. It means that they focus on maximising the number of correct matches while minimising the number of false positive. For example, negative keys are informations about objects (or tuples) that intuitively help in deciding if two objects do not match. "They can be stated and implemented as constraints" [31]. For instance, "no person has more than one social security number". It implies that if it is found two people with same (or closeness) name but with different social security number, then they are distinct people.

Some solutions concerning the instance matching has been developed based on rules (e.g. [15]) others in learning (e.g. [29, 30, 33]). Also there are solutions focus on specific contexts, as to match strings [14, 34], or focus on more complex domains, as in [35, 36].

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 [17], 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:

$$\textbf{match} : ((\mathbf{S}_1 [\mathrm{R}_1], \tau_1) \times (\mathbf{S}_2 [\mathrm{R}_2], \{\tau_2\})) \rightarrow \text{Boolean} \quad (3)$$

being $\mathbf{S}_i$ schema names, R_i class/relation names in $\mathbf{S}_i$, 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 are semantically equivalent or not. If only the first argument is instanced, i.e. $\mathbf{S}_1.\mathrm{R}_1$, 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 to guarantee the one-to-one correspondence between instances. In this case **match** has the following syntax signature:

$$\textbf{match} : ((\mathbf{S}_1 [\mathrm{R}_1], \tau_1) \times (\mathbf{S}_2 [\mathrm{R}_2 (\textbf{predicate})], \{\tau_2\})) \rightarrow \text{Boolean} \quad (4)$$

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 function signatures involving schemata of Figure 3 are presented in Figure 4.

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

Figure 4: Examples of matching function signatures.

The implementation of the matching functions shall be externally provided, since their code are very close to the application domain. However, in order to make easer the implementation of a simple prototype, a new variants of **match** is introduced in L_{PS}:

$$\textbf{match} : \left(\left(\mathbf{S}_1 [\mathrm{R}_1], \tau_1, \{\boldsymbol{p}'_1 : \tau'_1, ..., \boldsymbol{p}'_n : \tau'_n\}\right) \times \left(\mathbf{S}_2 [\mathrm{R}_2], \{\tau_2\}, \{\boldsymbol{p}''_1 : \tau''_1, ..., \boldsymbol{p}''_n : \tau''_n\}\right)\right) \rightarrow \text{Boolean} \quad (5)$$

being that $\boldsymbol{p}'_i{:}\tau'_i \in \textbf{\textit{type}}(\mathrm{R}_1)$; and $\boldsymbol{p}''_i{:}\tau''_i \in \textbf{\textit{type}}(\mathrm{R}_2)$, $1 \leq \mathrm{i} \leq \mathrm{n}$.[3] This variant of the matching function is automatically generated by the system and indicates that the matching is done by simple attribute comparison, i.e. each property $\boldsymbol{p}'_i$ of R_1 will be compared with the property $\boldsymbol{p}''_i$ of R_2, for $1 \leq \mathrm{i} \leq \mathrm{n}$.

The matching functions are locally declared in each perspective schema, even thinking that probably must exist many perspective schemata in a same environment, which can need to declare matching functions involving the same classes/relations. It is done in order to guarantee the logical independence between the perspective schemata, that is each perspective schema must have all declarations that are necessary to create it.

[3] ***type***() is a function defined in language L_S that returns the structural type of a relation/class.

3.2. Correspondence Assertions

Another difficulty in a data integration environment, besides of the instance matching problem, is establish the correspondence semantic between schemata's components. This problem can be hard to deal with, since in that context usually are involved multiple, autonomous and heterogeneous information sources. Thus, even after all source schemata are drawn to a common language (in this proposal using the language L_S), semantic heterogeneity (cf. [37]) still should be treated. This semantic heterogeneity can be of several kinds, such as (cf. [38]):

- *Naming conflict*, for example synonyms or homonyms properties. Synonyms occurs when the properties have different names, but the same content. For instance, the name of a person can be represented by **name** in a class, and by **customer_name** in another. Homonyms occurs when the properties have the same name, but different contents. For instance, the property **id** can represent the identity card of a person in a relation, and the identifier of a product in another.
- *Data representation conflict*. It occurs when similar contents are represented by different data types. For instance, an address can be represented by a string containing the street and the number, or by a structural type formed by two properties: **street** and **number**.
- *Encoding conflict*. It occurs when similar contents are represented by different format of data or unit of measures and there is one-to-one mapping between the values of property domains. For instance, the salary of an employee can have the value in US dollars in a property and in euros in another one.
- *Data Precision conflict*. It occurs when similar content are represented by different precision. It is not the same case that before item, because here usually there are a many-to-one mapping from the domain of the precise property to the domain of the coarse property, and can there is not a precise way to transform the information. For instance, a grade of a subject can be represented by the values from 1 to 10 in a property, and by the values $\{$A,B,C,D,E$\}$ in another one.

Here, the correspondence semantic between schemata's components is formally declared in a declarative fashion through the Correspondence Assertions (CAs). There are several kinds of correspondence assertions, which depend on the involved elements and the nature of the correspondence, i.e., of the imposed constraint. These assertions express the knowledge *this element is related with this another element*, and therefore they impose constraints in the permissible states of the schema. The permissible states of the schema are those that satisfy the structure and the constraints of the schema. In this work, the relationship between the target schema and the base schemata can be specified by the following four kinds of correspondence assertions:

- Property Correspondence Assertion (PCA),
- Extension Correspondence Assertion (ECA),
- Summation Correspondence Assertion (SCA), and
- Aggregation Correspondence Assertion (ACA).

Some examples of CAs are shown in Figure 5 and explained in that Section.

In some CAs are presenting Boolean conditions (or selection predicates), which are used to restrict the number of instances in a class/relation or to manage the value of a property. In the current research, it is used the notation defined in the following text to specify selection predicates.

Property Correspondence Assertions:

ψ_1: $\mathbf{P_{S_1|RM}}$ [PRODUCT] • $\mathbf{pid}_{\mathrm{RM}} \rightarrow \mathbf{S}_1$ [PRODUCT] • $\mathbf{prod\_id}_{\mathrm{S_1}}$

ψ_2: $\mathbf{P_{RM|DW}}$ [PRODUCT] • $\mathbf{prod\_id}_{\mathrm{DW}} \rightarrow \mathbf{RM}$ [PRODUCT] • $\mathbf{pid}_{\mathrm{RM}}$

ψ_3: $\mathbf{P_{S_1|RM}}$ [SALE_ITEM] • $\mathbf{unitprice}_{\mathrm{RM}} \rightarrow \mathbf{dollarTOeuro}\ (\mathbf{S}_1$ [ITEM] • $\mathbf{unitprice}_{\mathrm{S_1}})$

Extension Correspondence Assertions:

ψ_9: $\mathbf{P_{S_1|RM}}$ [PRODUCT] $\rightarrow \mathbf{S}_1$ [PRODUCT]

ψ_{10}: $\mathbf{S}_v$ [PRODUCT] $\rightarrow \mathbf{S}_1$ [PRODUCT] $\sqsupset\!\bowtie\!\sqsubset \mathbf{S}_3$ [PROD]

ψ_{11}: $\mathbf{P_{RM|DW}}$ [PRODUCT ($\mathbf{current\_flag}_{\mathrm{DW}} =$ True)] $\rightarrow \mathbf{RM}$ [PRODUCT]

Summation Correspondence Assertions:

ψ_{14}: $\mathbf{P_{RM|DW}}$ [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_{15}: $\mathbf{P_{S_2|RM}}$ [PRODUCT] ($\mathbf{pid}_{\mathrm{RM}}$) $\rightarrow$ normalise ($\mathbf{S}_2$ [PRODUCT_SALES] ($\mathbf{product\_number}_{\mathrm{S2}}$))

Aggregation Correspondence Assertions:

ψ_{16}: $\mathbf{P_{RM|DW}}$ [SALE_ITEM] • $\mathbf{prodsold\_qty}_{\mathrm{DW}} \rightarrow \psi_{14}$, **sum** ($\mathbf{RM}$ [SALE_ITEM] • $\mathbf{quantity}_{\mathrm{RM}}$)

ψ_{17}: $\mathbf{P_{RM|DW}}$ [SALE_ITEM] • $\mathbf{prodsold\_amt}_{\mathrm{DW}} \rightarrow \psi_{14}$, **sum** ($\mathbf{RM}$ [SALE_ITEM] • $\mathbf{quantity}_{\mathrm{RM}} \times \mathbf{RM}$ [SALE_ITEM] • $\mathbf{unitprice}_{\mathrm{RM}}$)

Figure 5: Examples of correspondence assertion.

Definition 1 (*Predicate) Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{P}$ a set of property names, $\mathcal{W}$ a set of typed values, and $\mathcal{L}$ a set of schema names. Let also $\mathrm{S}' \in \mathcal{L}$, $\mathrm{C}' \in \{\mathcal{C}, \mathcal{R}\}$, and $\boldsymbol{w} \in \mathcal{W}$. The predicate* ***pred*** *is a boolean condition defined by the following grammar:*

$$
\begin{array}{lcl}
\boldsymbol{pred} & ::= & \boldsymbol{A}\ \ \boldsymbol{op}\ \ \boldsymbol{B}\ \ | \\
 & & \boldsymbol{A}\ \ \boldsymbol{op}\ \ \boldsymbol{B}\ \ and\ \ \boldsymbol{pred}\ \ | \\
 & & \boldsymbol{A}\ \ \boldsymbol{op}\ \ \boldsymbol{B}\ \ or\ \ \boldsymbol{pred} \\
 & & \\
\boldsymbol{A} & ::= & \mathrm{S}'\,[\mathrm{C}'] \bullet \boldsymbol{p}'_i\ \ | \\
 & & \varphi(\boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_n)\ \ | \\
 & & \mathrm{S}'\,[\mathrm{C}'] \bullet \boldsymbol{p}'_i\{\boldsymbol{p}''_t\} \\
 & & \\
\boldsymbol{B} & ::= & \boldsymbol{w}\ \ | \\
 & & \boldsymbol{A} \\
 & & \\
\boldsymbol{op} & ::= & >\ \ |\ \ <\ \ |\ \ \geq\ \ |\ \ \leq\ \ |\ \ =\ \ |\ \ \neq
\end{array}
$$

being that:

- ***A*** *and* ***B*** *are the parameters of the predicate* ***pred****,*
- ***op*** *is an operand,*
- φ *is a function with n $\geq$ 1 arguments that returns a value,*
- $\boldsymbol{p}'_i \in$ ***props***(C') *or* $\boldsymbol{p}'_i$ *is a path of* C'*, for i $\geq$ 1. The notation* $\boldsymbol{p}'_i\{\boldsymbol{p}''_t\}$ *means that* $\boldsymbol{p}'_i$ *is of the structural type and* $\boldsymbol{p}''_t$*, for t $\geq$ 1, is one of the property names belonging to the type of* $\boldsymbol{p}'_i$*.*[4]

[4] ***props***() is a function defined in language L_S that returns the set of property names of a class/relation; and path of C' is a short form to path expression of class/relation C'(path expressions also are defined in L_S).

φ can be a power function to perform: a) the mapping of domain types (for instance, changing the value from a type to another, or changing different format of data or unit of measure), b) arithmetic calculations, c) operations on strings (as concatenation), for citing only some.

Example 1

Below it is shown some examples of boolean conditions that are predicates and others that are not, based on schemata presented in Figure 3.

Predicates:
a) RM[REGION]•**rcode**$_{RM} \geq 100$ and RM[REGION]•**rcode**$_{RM} \leq 500$
b) S_1[CUSTOMER]•**zone_id**$_{S1} \neq 1000$
c) convertDollarToEuro(S_1[ITEM]•**unitprice**$_{S1}$) > 1000

Non-Predicates:
d) S_1[CUSTOMER]•**cust_id**$_{S1}$ = S_2[CUSTOMER]•**id**$_{S2}$
e) $1000 \leq S_2$[PRODUCT_SALES]•**vendor_salary**$_{S2}$
f) 'John' $\neq$ 'Mary'
g) not(S_1[ZONE]•**zone_id**$_{S1}$) ≥ 100

Note that the items d), e), f), and g) are not legal Boolean conditions in language L_{PS} since the properties being evaluated should belong to the same relation/class in a same schema, the first parameter of the predicate cannot be a value, and 'not' is not a valid operand in L_{PS}.

3.2.1. Property Correspondence Assertions - PCAs

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, data representation conflict, and encoding conflict. Furthermore, we can declare: i) Boolean conditions in that the property's value depends on the validation of a (or several) condition(s); ii) calculations in that the value of a property is the result of a calculation involving two or more properties of a same instance. A PCA is formally defined as follows:

Definition 2 (*Property Correspondence Assertion) Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}$ be a set of schema names. Let also* $\boldsymbol{S} \in \mathcal{L}$, C $\in \{\mathcal{C}, \mathcal{R}\}$, *and* ***pred***, ***A*** *and* ***B*** *be, respectively, a predicate and the parameters of* ***pred*** *such as defined in Definition 1. A property correspondence assertion of* C *is one rule defined over $\mathcal{C}$, $\mathcal{R}$, $\mathcal{A}$, and $\mathcal{L}$ having one of the following forms:*

$$\psi: \quad \boldsymbol{S}[\mathrm{C}] \bullet \boldsymbol{p} \quad \rightarrow \quad \boldsymbol{A} \;\mid\; (\boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_n) \;\mid\; (\boldsymbol{B}_1, \boldsymbol{pred}_1)\,;(\boldsymbol{B}_2, \boldsymbol{pred}_2)\,;...;\left(\boldsymbol{B}_{m-1}, \boldsymbol{pred}_{m-1}\right);\boldsymbol{B}_m$$

or

$$\psi: \quad \boldsymbol{S}[\mathrm{C}] \bullet \boldsymbol{p}\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\} \quad \rightarrow \quad (\boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_n)\,.$$

Being that:

- *ψ is a correspondence assertion name in $\mathcal{A}$;*
- $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{C})$. □

The mean of each rule in Definition 2 is as follows:

- If ψ : $\mathbf{S}$ [C] • $\boldsymbol{p}$ → $\mathbf{A}$ then may exists at least three situations, depending of the value of $\mathbf{A}$:
  1. $\mathbf{A} = \mathbf{S}'$ [C′] • $\boldsymbol{p}'$. In this case, $\mathbf{S}$ [C] •$\boldsymbol{p}$ ≡ $\mathbf{S}'$ [C′] •$\boldsymbol{p}'$ (read $\boldsymbol{p}$ is semantically equivalent to $\boldsymbol{p}'$). $\boldsymbol{p}$ and $\boldsymbol{p}'$ have compatible domain.
  2. $\mathbf{A} = \varphi(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n)$, n ≥ 1. In this case, $\mathbf{S}$[C] •$\boldsymbol{p}$ ◄ $\varphi(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n)$ (read $\boldsymbol{p}$ is derived from $\varphi(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n)$). $\boldsymbol{p}$ is a singlevalued property whose value is calculated, or, when i = 1, $\boldsymbol{p}$ has different domain or diverse type from presented in $\mathbf{A}_1$.
  3. $\mathbf{A}$ = $\mathbf{S}'$ [C′] • $\boldsymbol{p}'\{\boldsymbol{p}''_t\}$. In this case, $\mathbf{S}$[C] •$\boldsymbol{p}$ ≡ $\mathbf{S}'$ [C′] • $\boldsymbol{p}'\{\boldsymbol{p}''_t\}$ (read $\boldsymbol{p}$ is semantically equivalent to $\boldsymbol{p}''_t$ of $\boldsymbol{p}'$. $\boldsymbol{p}$ and $\boldsymbol{p}''_t$ have compatible domains.
- If ψ : $\mathbf{S}$ [C] • $\boldsymbol{p}$ → $(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n)$ then $\mathbf{S}$[C] •$\boldsymbol{p}$ ≡ $(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n)$ (read $\boldsymbol{p}$ is set equivalent to $\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_n$). In this case, $\boldsymbol{p}$ is a multivalued property (a set, a list, or a array of a given type τ), and $\mathbf{A}_i$, for 1 ≤ i ≤ n, is of the type τ.
- If ψ: $\mathbf{S}$[C] • $\boldsymbol{p}\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\}$ → ($\mathbf{A}_1$, $\mathbf{A}_2$, ..., $\mathbf{A}_n$) then $\mathbf{S}$[C] • $\boldsymbol{p}\{\boldsymbol{p}_1, ..., \boldsymbol{p}_n\}$ ≡ ($\mathbf{A}_1$, $\mathbf{A}_2$, ..., $\mathbf{A}_n$) (read $\boldsymbol{p}$ is tuple equivalent to $\mathbf{A}_1$, $\mathbf{A}_2$, ..., $\mathbf{A}_n$). In this case, $\boldsymbol{p}$ is a singlevalued property of the structural type: $\{\boldsymbol{p}_1{:}\tau_1, \boldsymbol{p}_2{:}\tau_2, ..., \boldsymbol{p}_n{:}\tau_n\}$ and each $\mathbf{A}_i$ has the type τ_i, for 1 ≤ i ≤ n.
- If ψ : $\mathbf{S}$ [C] • $\boldsymbol{p}$ → $(\mathbf{B}_1, \mathbf{pred}_1)$; $(\mathbf{B}_2, \mathbf{pred}_2)$; ... $\left(\mathbf{B}_{m-1}, \mathbf{pred}_{m-1}\right)$; $\mathbf{B}_m$ then $\mathbf{S}$ [C] •$\boldsymbol{p}$ can have one of many values $\mathbf{A}_1$, $\mathbf{A}_2$, ..., $\mathbf{A}_m$ depending of conditions presented in each predicate $\mathbf{pred}_i$, for 1 ≤ i ≤ m-1. The value in $\boldsymbol{p}$ will be the first $\mathbf{B}_i$ whose **pred** is true, or $\mathbf{B}_m$ if every **pred** is false.

Example 2

The Figure 6 shows some examples of PCAs relating schemata presented in Figure 3. For declaring these CAs, some suppositions were done, namely:

- *For ψ_4, the type of property* ***caddress***$_{RM}$ *of relation* ***RM***.CUSTOMER *is a string and the type of property* ***cust_address***$_{S1}$ *of relation* $\mathbf{S}_1$.CUSTOMER *is the structural type:* {***city****:string,* ***street****:string*}.
- *For ψ_5, the type of property* ***caddress***$_{RM}$ *pass to be the structural type:* {***city****:string,* ***street****:string*} *and the type of the properties* ***city***$_{S2}$ *and* ***street***$_{S2}$*, both of relation* ***S2***.CUSTOMER*, are string.*
- *For ψ_6 the type of property* ***phone***$_{RM}$ *of relation* ***RM***.CUSTOMER *is a set of strings and the the type of the properties* ***contact1***$_{S2}$ *and* ***contact2***$_{S2}$*, both of relation* ***S2***.CUSTOMER*, are string.*
- *For ψ_7, all products sold in '31/12/2008' have their date changed to '01/01/2009').*

The PCAs ψ_1 and ψ_2 deal with naming conflict. ψ_1 links the property ***pid***$_{RM}$ *of* ***RM***.PRODUCT *to the property* ***prod_id***$_{S1}$ *of* $\mathbf{S}_1$.PRODUCT.

The PCA ψ_3 deals with encoding conflict. It assigns ***unitprice***$_{RM}$ *of* SALE_ITEM *to* ***unitprice***$_{S1}$ *of* ITEM *using the function* ***dollarTOeuro*** *to convert currencies from dollars (stored in* ***unitprice***$_{S1}$*) to euros (stored in* ***unitprice***$_{RM}$*).*

The PCAs ψ_4, ψ_5, and ψ_6 deal with data representation conflict. ψ_4 links the property ***caddress***$_{RM}$ *to property* ***cust_address***$_{S1}$ *changing the type of the latter for the type of the first using the function*

transformAddress. ψ_5 *assigns the properties* ***city*** *and* ***street*** *of the property* $\textbf{\textit{caddress}}_{RM}$ *to, respectively, the properties* $\textbf{\textit{city}}_{S2}$ *and* $\textbf{\textit{street}}_{S2}$*. Here is not necessary to use a function, since each property in the structural type has a correspondent property of the same type in the base schema.* ψ_6 *links the property* $\textbf{\textit{cphone}}_{RM}$ *to the properties* $\textbf{\textit{contact1}}_{S2}$ *and* $\textbf{\textit{contact2}}_{S2}$*, indicating that the values of the latters are the (set, list or array) elements of the former.*

The PCA ψ_7 *deals with conditional values. It assigns to property* $\textbf{\textit{sale\_date}}_{RM}$ *to property* $\textbf{\textit{sale\_date}}_{S1}$*, but if the value of* $\textbf{\textit{sale\_date}}_{S1}$ *is iqual to '31/12/2008', then the value of* $\textbf{\textit{sale\_date}}_{RM}$ *is the constant '01/01/2009'.*

The PCA ψ_8 *deals with the denormalization. It assings the property* $\textbf{\textit{region\_desc}}_{DW}$ *to the path* ***RM***[CUSTOMER]•FK1•$\textbf{\textit{rname}}_{RM}$*, being that* FK1 *is a foreign key of* ***RM***.CUSTOMER *that refers to* ***RM***.REGION.

It is important to note that it is possible exist more than one PCA specified for a same property in a class/relation. For instance, the relation **RM**.CUSTOMER, presented in Figure 3, can have two PCAs to the property $\textbf{cname}_{\text{RM}}$: one relating $\textbf{cname}_{\text{RM}}$ to property $\textbf{cust\_name}_{\text{S1}}$ (ψ_1': $\textbf{P}_{\textbf{S}_1|\textbf{RM}}$ [CUSTOMER] • $\textbf{cname}_{\text{RM}} \rightarrow \textbf{S}_1$ [CUSTOMER] • $\textbf{cust\_name}_{\text{S}_1}$), and other relating $\textbf{cname}_{\text{RM}}$ to property $\textbf{name}_{\text{S2}}$ (ψ_2': $\textbf{P}_{\textbf{S}_2|\textbf{RM}}$ [CUSTOMER]•$\textbf{cname}_{\text{RM}} \rightarrow \textbf{S}_2$ [CUSTOMER]•$\textbf{name}_{\text{S}_2}$). This means that the properties $\textbf{cname}_{\text{RM}}$, $\textbf{cust\_name}_{\text{S1}}$, and $\textbf{name}_{\text{S2}}$ are *synonym properties* and then, they must have the same value.

3.2.2. Extension Correspondence Assertions – ECAs

The Extension Correspondence Assertions (ECAs) are used to describe the relationship that exists between the instances of the target schema and the instances of the base schemata. For instance, considering the schemata displayed in Figure 3, the relation **RM**.PRODUCT is linked to relation $\textbf{S}_1$.PRODUCT through the ECA ψ_9 presented in Figure 7. ψ_9 determines that **RM**.PRODUCT and $\textbf{S}_1$.PRODUCT are equivalent, i.e., for each tuple of PRODUCT of the schema $\textbf{S}_1$ 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 ECAs are used to define which objects/tuples of a base schema should have a corresponding seman-

ψ_1: $\textbf{P}_{\textbf{S}_1|\textbf{RM}}$ [PRODUCT] • $\textbf{pid}_{\text{RM}} \rightarrow \textbf{S}_1$ [PRODUCT] • $\textbf{prod\_id}_{\text{S}_1}$
ψ_2: $\textbf{P}_{\textbf{RM}|\textbf{DW}}$ [PRODUCT] • $\textbf{prod\_id}_{\text{DW}} \rightarrow \textbf{RM}$ [PRODUCT] • $\textbf{pid}_{\text{RM}}$
ψ_3: $\textbf{P}_{\textbf{S}_1|\textbf{RM}}$ [SALE_ITEM] • $\textbf{unitprice}_{\text{RM}} \rightarrow \textbf{dollarTOeuro}\,(\textbf{S}_1$ [ITEM] • $\textbf{unitprice}_{\text{S}_1})$
ψ_4: $\textbf{P}_{\textbf{S}_1|\textbf{RM}}$ [CUSTOMER] • $\textbf{caddress}_{\text{RM}} \rightarrow \textbf{transformAddress}\,(\textbf{S}_1$ [CUSTOMER] • $\textbf{cust\_address}_{\text{S}_1}\{\textbf{city}\}$, $\textbf{S}_1$ [CUSTOMER] • $\textbf{cust\_address}_{\text{S}_1}\{\textbf{street}\})$
ψ_5: $\textbf{P}_{\textbf{S}_2|\textbf{RM}}$ [CUSTOMER] • $\textbf{caddress}_{\text{RM}}\{\textbf{city}, \textbf{street}\} \rightarrow (\textbf{S}_2$ [CUSTOMER] • $\textbf{city}_{\text{S}_2}, \textbf{S}_2$ [CUSTOMER] • $\textbf{street}_{\text{S}_2})$
ψ_6: $\textbf{P}_{\textbf{S}_2|\textbf{RM}}$ [CUSTOMER] • $\textbf{cphone}_{\text{RM}} \rightarrow (\textbf{S}_2$ [CUSTOMER] • $\textbf{contact1}_{\text{S}_2}, \textbf{S}_2$ [CUSTOMER] • $\textbf{contact2}_{\text{S}_2})$
ψ_7: $\textbf{P}_{\textbf{S}_1|\textbf{RM}}$ [SALE] • $\textbf{sale\_date}_{\text{RM}} \rightarrow (\textbf{S}_1$ [SALE] • $\textbf{sale\_date}_{\text{S}_1}, \textbf{S}_1$ [SALE] • $\textbf{sale\_date}_{\text{S}_1} \neq' 31/12/2008')$; $'01/01/2009'$
ψ_8: $\textbf{P}_{\textbf{RM}|\textbf{DW}}$ [CUSTOMER] • $\textbf{region\_desc}_{\text{DW}} \rightarrow \textbf{RM}$ [CUSTOMER] • FK1 • $\textbf{rname}_{\text{RM}}$

Figure 6: Examples of property correspondence assertion.

ψ_9: $\mathbf{P}_{\mathbf{S}_1\|\mathbf{RM}}$ [PRODUCT] $\rightarrow$ $\mathbf{S}_1$ [PRODUCT]	(ECA of equivalence)
ψ_{10}: $\mathbf{S}_v$ [PRODUCT] $\rightarrow$ $\mathbf{S}_1$ [PRODUCT] $\exists\bowtie\sqsubset$ $\mathbf{S}_3$ [PROD]	(ECA of union)
ψ_{11}: $\mathbf{P}_{\mathbf{RM}\|\mathbf{DW}}$ [PRODUCT (**current_flag**$_{\text{DW}}$ = True)] $\rightarrow$ **RM** [PRODUCT]	(ECA of selection)
ψ_{12}: $\mathbf{S}'_v$ [CUSTOMER_NY] $\rightarrow$ $\mathbf{S}_1$ [CUSTOMER] $-$ $\mathbf{S}_2$ [CUSTOMER]	(ECA of difference)
ψ_{13}: $\mathbf{S}'_v$ [SHARED_CUSTOMER] $\rightarrow$ $\mathbf{S}_1$ [CUSTOMER] $\cap$ $\mathbf{S}_2$ [CUSTOMER]	(ECA of intersection)

Figure 7: Examples of extension correspondence assertion.

tically equivalent object/tuple in a target schema, being that, in case of classes, it can taken account the class hierarchy or not. A ECA is formally defined as follows:

Definition 3 (*Extension Correspondence Assertion) Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}$ be a set of schema names. Let also* C *and* C_i $\in \{\mathcal{C}, \mathcal{R}\}$, S *and* $\mathrm{S}_i \in \mathcal{L}$, *for* $1 \leq i \leq n$, ***pred*** *be a predicate as defined in Definition 1, and* $\boldsymbol{E}$ *be an expression defined by the following grammar:*

$$\begin{array}{rcl} \boldsymbol{E} & ::= & \mathrm{S}_i\,[\mathrm{C}_i] \quad | \\ & & \mathrm{S}_i\,[\mathrm{C}_i\,(\boldsymbol{pred})] \end{array}$$

An extension correspondence assertion of C_1 *is a rule defined over $\mathcal{C}$, $\mathcal{R}$, $\mathcal{A}$, and $\mathcal{L}$ having one of the following forms:*

$$\begin{array}{lcl} \psi: \ \mathrm{S}\,[\mathrm{C}] & \rightarrow & \boldsymbol{E}_1 \quad | \\ & & \boldsymbol{E}_1 - \boldsymbol{E}_2 \quad | \\ & & \boldsymbol{E}_1 \cap \boldsymbol{E}_2 \cap \cdots \cap \boldsymbol{E}_n \quad | \\ & & \boldsymbol{E}_1 \ \exists\bowtie\sqsubset\ \boldsymbol{E}_2 \ \exists\bowtie\sqsubset\ \ldots \ \exists\bowtie\sqsubset\ \boldsymbol{E}_n \\ \textit{or} & & \\ \psi: \ \mathrm{S}\,[\mathrm{C}\,(\boldsymbol{pred})] & \rightarrow & \boldsymbol{E}_1 \quad | \\ & & \boldsymbol{E}_1 - \boldsymbol{E}_2 \quad | \\ & & \boldsymbol{E}_1 \cap \boldsymbol{E}_2 \cap \cdots \cap \boldsymbol{E}_n \quad | \\ & & \boldsymbol{E}_1 \ \exists\bowtie\sqsubset\ \boldsymbol{E}_2 \ \exists\bowtie\sqsubset\ \ldots \ \exists\bowtie\sqsubset\ \boldsymbol{E}_n. \end{array}$$

Being that ψ is a correspondence assertion name in $\mathcal{A}$. □

The mean of each rule in Definition 3 is as follows:

- If ψ : $\mathrm{S}\,[\mathrm{C}] \rightarrow \mathrm{S}_1\,[\mathrm{C}_1]$ then $\mathrm{S}[\mathrm{C}] \equiv \mathrm{S}_1[\mathrm{C}_1]$ (read C is semantically equivalent to C_1) (**ECA of equivalence**).
- If ψ : $\mathrm{S}\,[\mathrm{C}] \rightarrow \mathrm{S}_1\,[\mathrm{C}_1\,(\mathbf{pred})]$ then $\mathrm{S}[\mathrm{C}] \equiv \mathrm{S}_1[\mathrm{C}_1\,(\mathbf{pred})]$ (read C is semantically equivalent to C_1(**pred**)) (**ECA of selection**).
- If $\psi : \mathrm{S}\,[\mathrm{C}] \rightarrow \mathrm{S}_1\,[\mathrm{C}_1] - \mathrm{S}_2\,[\mathrm{C}_2]$ then $\mathrm{S}[\mathrm{C}] \equiv \mathrm{S}_1[\mathrm{C}_1] - \mathrm{S}_2[\mathrm{C}_2]$ (read C is semantically equivalent to $\mathrm{C}_1 - \mathrm{C}_2$) (**ECA of difference**).
- If $\psi : \mathrm{S}\,[\mathrm{C}] \rightarrow \mathrm{S}_1\,[\mathrm{C}_1] \ \exists\bowtie\sqsubset\ \mathrm{S}_2\,[\mathrm{C}_2] \ \exists\bowtie\sqsubset\ \ldots \ \exists\bowtie\sqsubset\ \mathrm{S}_n\,[\mathrm{C}_n]$ then $\mathrm{S}[\mathrm{C}] \equiv \overset{n}{\underset{i=1}{\exists\bowtie\sqsubset}} \mathrm{S}_i[\mathrm{C}_i]$ (read C is semantically equivalent to union of C_i) (**ECA of union**).
- If ψ : $\mathrm{S}\,[\mathrm{C}] \rightarrow \mathrm{S}_1\,[\mathrm{C}_1] \cap \mathrm{S}_2\,[\mathrm{C}_2] \cap \cdots \cap \mathrm{S}_n\,[\mathrm{C}_n]$ then $\mathrm{S}[\mathrm{C}] \equiv \overset{n}{\underset{i=1}{\cap}} \mathrm{S}_i[\mathrm{C}_i]$ (read C is semantically equivalent to intersection of C_i) (**ECA of intersection**).

Mutatis mutandis when changing S[C] by S[C(**pred**)].

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 a view schema S_v with the relation PRODUCT(**code, description, category**) which is related to two relations: PRODUCT in S_1 (in Figure 3), and PROD(**code, description,category**) in schema S_3 (not presented in any Figure) through the ECA ψ_{10} shown in Figure 7. ψ_{10} determines that PRODUCT in S_v is the union/join of PRODUCT in S_1 and PROD in S_3, 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_3 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 condition of selection, which determines the subset of instances of the class/relation that is 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_{11} presented in Figure 7. ψ_{11} determines that a subset of instances of the relation **DW**.PRODUCT, whose value of the property **current_flag**$_{\text{DW}}$ is true, is the same as those instances of the relation **RM**.PRODUCT.

Example 3

Below it is presented some more examples of extension correspondence assertions.

Suppose that the schemata $\boldsymbol{S}_1$ *and* $\boldsymbol{S}_2$ *hold sales of distinct shops located, respectively, in New York and in Spring Valley, and that there is an integrated view schema* $\boldsymbol{S}'_v$ *from these two base schema.* $\boldsymbol{S}'_v$ *contains two relations:* CUSTOMER_NY *and* SHARED_CUSTOMER. CUSTOMER_NY *holds the customers that only go shoping in New York shop while* SHARED_CUSTOMER *stores information about customers that go shoping in both shops. The relationship between these relations and ones of the base schemata are described through the ECAs* ψ_{12} *and* ψ_{13} *presented in Figure 7.* ψ_{12} *specifies that* CUSTOMER_NY *contains the instances of the relation* $\boldsymbol{S}_1$.CUSTOMER *that are not instances of the relation* $\boldsymbol{S}_2$.CUSTOMER. ψ_{13} *specifies that the relation* SHARED_CUSTOMER *contains only the instances that are common to the relations* $\boldsymbol{S}_1$.CUSTOMER *and* $\boldsymbol{S}_2$.CUSTOMER.

3.2.3. Summation Correspondence Assertion – SCA

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 *normalise*. 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 *normalise* is used to indicate a normalisation process. For example, the instances of the relations **DW**.SALE_ITEM and **RM**.SALE_ITEM are connected through the SCA ψ_{14} displayed in Figure 8. ψ_{14} determines that **DW**.SALE_ITEM is the grouping of **RM**.SALE_ITEM based on values of **pid**$_{\text{RM}}$, **RM**.SALE_ITEM•$\boldsymbol{FK}_1$•**cid**$_{\text{RM}}$, and **RM**.SALE_ITEM • $\boldsymbol{FK}_1$• **sale_date**$_{\text{RM}}$.[5] In other words, there is a tu-

[5] **RM**[SALE_ITEM] • $\boldsymbol{FK}_1$ • **cid**$_{\text{RM}}$ and **RM**[SALE_ITEM] • $\boldsymbol{FK}_1$ • **sale_date**$_{\text{RM}}$ 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.**cid**$_{\text{RM}}$) and the value of sale date (**RM**.SALE.**sale_date**$_{\text{RM}}$).

ple 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$\bullet \boldsymbol{FK}_1 \bullet \mathbf{cid}_{\mathrm{RM}}$) and sale date (**RM**.SALE_ITEM$\bullet \boldsymbol{FK}_1 \bullet \mathbf{sale_date}_{\mathrm{RM}}$). A formal definition of SCA is as follows:

Definition 4 (*Summation Correspondence Assertion)*
Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{P}$ a set of property names, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}$ be a set of schema names. Let also C *and* $\mathrm{C}' \in \{\mathcal{C}, \mathcal{R}\}$, $\boldsymbol{p}_i \in \boldsymbol{props}(\mathrm{C})$ *or* $\boldsymbol{p}_i$ *is a path of* C *(for $1 \leq i \leq m$),* $\boldsymbol{p'}_i \in \boldsymbol{props}(\mathrm{C}')$ *or* $\boldsymbol{p'}_i$ *is a path of* C' *(for $1 \leq i \leq m$),* S *and* $\mathrm{S}' \in \mathcal{L}$*, and* $\boldsymbol{A}$ *and* $\boldsymbol{E}$ *are expressions as defined in Definition 3. A Summation correspondence assertion of* C *is a rule defined over $\mathcal{C}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{A}$, and $\mathcal{L}$ having one of the following forms:*

$$\psi: \quad \mathrm{S}\,[\mathrm{C}]\,(\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m) \quad \rightarrow \quad \begin{array}{l} \mathit{groupby}\,(\boldsymbol{E}\,(\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_m)) \quad | \\ \mathit{normalize}\,(\boldsymbol{E}\,(\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_m))\,. \end{array}$$

Being that:

- *ψ is a correspondence assertion name in $\mathcal{A}$.*
- *groupby indicates that some kind of aggregate function is involved in the relationship between* C *and* C'*.*
- *normalize indicates that occurs a normalisation process between* C *and* C'*.*
- *$\boldsymbol{p}_i$ and $\boldsymbol{p'}_i$, for $1 \leq i \leq m$, are the aggregate properties,i.e. are the properties that are the base for the grouping.* □

ψ_{14}: $\mathbf{P}_{\mathbf{RM|DW}}$ [SALE_ITEM] ($\mathbf{prod\_id\_sk}_{\mathrm{DW}}$, $\mathbf{cust\_id\_sk}_{\mathrm{DW}}$, $\mathbf{date\_id\_sk}_{\mathrm{DW}}$) → groupby (**RM** [SALE_ITEM] ($\mathbf{pid}_{\mathrm{RM}}$, $\boldsymbol{FK}_1 \bullet \mathbf{cid}_{\mathrm{RM}}$, $\boldsymbol{FK}_1 \bullet \mathbf{date\_id}_{\mathrm{RM}}$))
ψ_{15}: $\mathbf{P}_{\mathbf{S_2|RM}}$ [PRODUCT] ($\mathbf{pid}_{\mathrm{RM}}$) → normalise ($\mathbf{S}_2$ [PRODUCT_SALES] ($\mathbf{product\_number}_{\mathrm{S2}}$))

Figure 8: Examples of summation correspondence assertion.

Example 4

Below it is an example of a SCA of normalisation.

Consider the denormalised relation PRODUCT_SALES *of source schema $\boldsymbol{S}_2$ and the schema* ***RM****, both presented in Figure 3. The SCA ψ_{15}, displayed in Figure 8, determines the relationship between* PRODUCT_SALES *and* ***RM***.PRODUCT *when a normalisation process is involved, i.e., it determines that* ***RM***.PRODUCT *is a normalisation of* $\boldsymbol{S}_2$.PRODUCT_SALES *based on distinct values of property* $\boldsymbol{product\_number}_{S2}$*.*

This research also deals with denormalisations, which is defined using *path expressions* (component of the language L_S). Denormalisation was illustrated in Example 2.

3.2.4. Aggregation Correspondence Assertions – ACAs

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 [12], i.e. *summation, maximum, minimum, average* and *count*; although more complex aggregation are supported in some object-relational databases – cf. [39].

The ACAs, similar to the PCAs, allow the description of several kinds of situations; therefore, the aggregate expressions can be more elaborate 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). A formal definiton of ACA is as follows:

Definition 5 (*Aggregation Correspondence Assertion) Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{P}$ a set of property names, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}$ be a set of schema names. Let also* $\mathrm{S} \in \mathcal{L}$, $\mathrm{C} \in \{\mathcal{C}, \mathcal{R}\}$, $\boldsymbol{p'}_i \in \boldsymbol{props}(\mathrm{C'})$ *or* $\boldsymbol{p'}_i$ *is a path of* $\mathrm{C'}$ *(for* $1 \leq i \leq m$*),* ***pred*** *be a predicate, and* ***A*** *and* ***B*** *be parameters of* ***pred*** *(the last three are as defined in Definition 1). An aggregation correspondence assertion of* C *is one rule defined over $\mathcal{C}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{A}$, and $\mathcal{L}$ having one of the following forms:*

$$\psi : \boldsymbol{S}[\mathrm{C}] \bullet \boldsymbol{p} \rightarrow \psi', \gamma(\boldsymbol{A}) \quad | \tag{6}$$

$$\psi', \gamma(\boldsymbol{A}, \boldsymbol{pred}) \tag{7}$$

$$\psi', \boldsymbol{A} \quad | \tag{8}$$

$$\psi', (\boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_n) \quad | \tag{9}$$

$$\psi', (\boldsymbol{B}_1, \boldsymbol{pred}_1); (\boldsymbol{B}_2, \boldsymbol{pred}_2); ... \left(\boldsymbol{B}_{m-1}, \boldsymbol{pred}_{m-1}\right); \boldsymbol{B}_m \quad | \tag{10}$$

or

$$\psi : \boldsymbol{S}[\mathrm{C}] \bullet \boldsymbol{p}\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\} \rightarrow \psi', (\boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_n) \tag{11}$$

Being that:

- ψ *is a correspondence assertion name in* $\mathcal{A}$*;*
- ψ' *is a summation correspondence assertion name in* $\mathcal{A}$*, which correspond to a SCA of* groupby *in expressions* (6) *and* (7) *and a SCA of* normalize *in the remainder cases.*
- γ *is one of the functions sum, max, min, avg, and count, which are used to, respectively, retrieve the total, the maximum value, the minimum value, the average of a set of values, and for counting the occurrence of a given value in a property.* □

ψ_{16}: $\mathbf{P}_{\mathbf{RM|DW}}$ [SALE_ITEM] • $\mathbf{prodsold\_qty}_{\mathrm{DW}} \rightarrow \psi_{14}$, **sum** (**RM** [SALE_ITEM] • $\mathbf{quantity}_{\mathrm{RM}}$)
ψ_{17}: $\mathbf{P}_{\mathbf{RM|DW}}$ [SALE_ITEM] • $\mathbf{prodsold\_amt}_{\mathrm{DW}} \rightarrow \psi_{14}$, **sum** (**RM** [SALE_ITEM] • $\mathbf{quantity}_{\mathrm{RM}} \times$ **RM** [SALE_ITEM] • $\mathbf{unitprice}_{\mathrm{RM}}$)
ψ_{18}: $\mathbf{P}_{\mathbf{S_2|RM}}$ [PRODUCT] • $\mathbf{pname}_{\mathrm{RM}} \rightarrow \psi_{15}$, $\mathbf{S}_2$ [PRODUCT_SALES] • $\mathbf{product}_{\mathrm{S2}}$

Figure 9: Examples of aggregation correspondence assertion.

Example 5

Returning to the motivating example, the ACAs ψ_{16} and ψ_{17} (displayed in Figure 9) link the properties of ***DW***.SALE_ITEM *and* ***RM***.SALE_ITEM. *ψ_{16} determines that the value of the property* $\boldsymbol{prodsold\_qty}_{DW}$ *is the summation of the quantity of product sold for each customer for each date, being that*

the grouping is obtained in ψ_{16} by ψ_{14}. ψ_{17} determines that the value of the property ***prodsold_amt***$_{DW}$ *is the amount of product sold for each customer for each date. In ψ_{17} the grouping is obtained by ψ_{14}, as occurs in ψ_{16}, and the amount of product sold is calculated using the formula: price $\times$ quantity. The ACA ψ_{18}, also shown in Figure 9, assigns the property* ***pname***$_{RM}$ *(of* ***RM***.PRODUCT*) to property* ***product***$_{S2}$ *of* ***S***$_2$.PRODUCT_SALES*), indicating, by ψ_{15}, that it is functionally dependent of property* ***product_number***$_{S2}$.

3.3. Perspective Schema

Having introduced the require declarations, matching functions, and correspondence assertions, we can define a perspective schema as follows:

Definition 6 (*Perspective schema) Let $\mathcal{P}$ be a set of property names, $\mathcal{K}$ a set of key names, $\mathcal{R}$ a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{M}$ a set of method names, $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}_p$ a set of perspective schema names. A perspective schema* ***PS*** *defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, and $\mathcal{K}$ has one of two forms: $\left(\mathrm{S}, \hat{\mathcal{L}}_b, \left(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\right), \hat{\mathcal{A}}, \hat{\mathcal{F}}\right)$ or $\left(\mathrm{S}, \hat{\mathcal{L}}_b, \left(\mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}}\right), \hat{\mathcal{A}}, \hat{\mathcal{F}}\right)$ such that:*

- S *is a schema name in $\mathcal{L}_p$ (the name of the perspective schema* ***PS****);*
- *$\hat{\mathcal{L}}_b$ is a set of schemata, named* ***base schemata****, whose schema names belongs to $\mathcal{L}$;*
- *$\hat{\mathcal{L}}_t$ is the* ***target schema*** *whose schema names belongs to $\mathcal{L}$;*
- *$\mathcal{H}$ is a class hierarchy as defined in language L_S;*
- *$\hat{\mathcal{R}}$ is a set of relation declarations as defined in language L_S;*
- *$\hat{\mathcal{K}}$ is a set of key declarations and foreign key declarations as defined in language L_S;*
- *$\hat{\mathcal{A}}$ is a set of correspondence assertions whose names belong to $\mathcal{A}$;*
- *$\hat{\mathcal{F}}$ is a set of matching function declarations;*
- *$\widehat{\mathcal{R}q}$ is a set of require declarations.* □

The correspondence assertions of a perspective schema relate the target schema's components to the base schemata's components. 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*.

Due to the structure of a perspective schema is strongly dependent of concepts of other schemata, it is need to verify if the perspective schema is well done under a sintatic viewpoint, i.e., if it is a valid perspective schema. The following text shows some definitions to help in this task.

Definition 7 (*Valid require declarations) Let a perspective schema* ***PS*** = $\left(\mathrm{S}, \hat{\mathcal{L}}_b, \left(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\right)\right.$ *defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, and $\mathcal{K}$. A require declaration is a valid require declaration iff:*

- *If the require declaration is an expression of the form:* require(C,{$\boldsymbol{p}_1$,$\boldsymbol{p}_2$,...,$\boldsymbol{p}_n$}) then:
  - C is a class name or a relation name of, respectively, a classe declaration or a relation declaration declared in $\hat{\mathcal{L}}_t$;
  - $\boldsymbol{p}_i \in$ ***props***(C) and for all i,j, if i $\neq$ j then $\boldsymbol{p}_i \neq \boldsymbol{p}_j$, $1 \leq$ i $\leq$ n;
  - If there is another require declaration of the form: require(C′,{$\boldsymbol{p'}_1$, $\boldsymbol{p'}_2$, ..., $\boldsymbol{p'}_m$}) such that C = C′, then m=n and $\boldsymbol{p}_i = \boldsymbol{p'}_i$, for $1 \leq$ i $\leq$ n.
- If the require declaration is an expression of the form: require(***K***) then:

- ***K*** is a key name of a key declaration (***K***,C,**K**(C)) declared in $\hat{\mathcal{L}}_t$, such that there is a require declaration require(C,{$\boldsymbol{p}_1$,$\boldsymbol{p}_2$,...,$\boldsymbol{p}_n$}) in $\widehat{\mathcal{R}q}$ and **K**(C) $\subseteq$ {$\boldsymbol{p}_1$, $\boldsymbol{p}_2$, ..., $\boldsymbol{p}_n$};
- ***K*** is a key name of a foreign key declaration (***FK***,C,**FK**(C),C′,**K**(C′)) declared in $\hat{\mathcal{L}}_t$, such that there are the require declarations require(C,{$\boldsymbol{p}_1$, $\boldsymbol{p}_2$, ..., $\boldsymbol{p}_n$}) and require(C′,{$\boldsymbol{p'}_1$, $\boldsymbol{p'}_2$, ..., $\boldsymbol{p'}_n$}) in $\widehat{\mathcal{R}q}$, **FK**(C) $\subseteq$ {$\boldsymbol{p}_1$, $\boldsymbol{p}_2$, ..., $\boldsymbol{p}_n$} and **K**(C′) $\subseteq$ {$\boldsymbol{p'}_1$,$\boldsymbol{p'}_2$,...,$\boldsymbol{p'}_n$};
- If there is another require declaration of the form: require(***K***') then ***K*** $\neq$ ***K***'. □

Definition 8 (*Valid perspective schema) Let a perspective schema* ***PS*** = $\left(\mathrm{S}, \hat{\mathcal{L}}_b, \left(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\right), \hat{\mathcal{A}}, \hat{\mathcal{F}}\right)$ *or* ***PS*** = $\left(\mathrm{S}, \hat{\mathcal{L}}_b, \left(\mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}}\right), \hat{\mathcal{A}}, \hat{\mathcal{F}}\right)$ *defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, and $\mathcal{K}$. Let also* C *be the name of some class declaration in $\mathcal{H}$, or of some relation declaration in $\hat{\mathcal{R}}$, or of a class or a relation in some require declararation in $\widehat{\mathcal{R}q}$. A Perspective schema is a valid perspective schema iff:*

1. *All require declarations in $\widehat{\mathcal{R}q}$ are valid require declarations;*
2. *There is at least one key declaration in $\hat{\mathcal{K}}$ for each relation declaration in $\hat{\mathcal{R}}$;*
3. *In $\hat{\mathcal{A}}$ should exist at least one extension correspondence assertion of* C *or one summation correspondence assertion of* C.
4. *In $\hat{\mathcal{A}}$, if there is more than one extension correspondence assertion of* C *or summation correspondence assertion of* C, *then they connect* C *to distinct class names or relation names.*
5. *For each ψ and $\psi' \in \hat{\mathcal{A}}$ such that ψ is an ACA and ψ' is a SCA, ψ and ψ' refer to same class name(s) or relation name(s) of a same schema.*
6. *For each* $\boldsymbol{p}$:$\tau \in$ ***type***(C)*:*
   (a) *when there is a ECA ψ of* C *in $\hat{\mathcal{A}}$, then should exist at least a property correspondence assertion of* C *in $\hat{\mathcal{A}}$ that refers to* $\boldsymbol{p}$*; which relates* $\boldsymbol{p}$ *to some property or path belonging to a class or relation specified in ψ; or*
   (b) *when there is a SCA ψ of* C *in $\hat{\mathcal{A}}$: i) if the SCA is "*groupby*" then should exist at least an aggregation correspondence assertion of* groupby *(items* (6) *and* (7) *in Definition 5) of* C *in $\hat{\mathcal{A}}$ that refers to* $\boldsymbol{p}$*; or ii) if the SCA is "*normalize*" then should exist at least an aggregation correspondence assertion of "*normalize*" (items* (8) *to* (11) *in Definition 5) of* C *in $\hat{\mathcal{A}}$ that refers to* $\boldsymbol{p}$*; or*
   (c) $\boldsymbol{p} \in$ ***K***(C) $\subseteq$ ***props***(C)*, such that there is one of the following elements: a key declaration* (***K***, C, ***K***(C)) $\in \hat{\mathcal{K}}$*; or a require declaration* require(***K***) $\in \widehat{\mathcal{R}q}$ with ***K*** being the name of a key declaration of C; and the values of ***K*** are automatically generated by the system.
7. *In $\hat{\mathcal{A}}$, if there is more than one property correspondence assertion of* C *or aggregation correspondence assertion of* C *to a same* $\boldsymbol{p} \in$ ***props***(C)*, then they connect* $\boldsymbol{p}$ *to properties or paths of distinct class names or relation names;*
8. *$\hat{\mathcal{F}}$ is a nonempty set of matching function declarations built according to following rules:*
   (a) *For each PCA or ACA of* C *connecting* $\boldsymbol{p} \in$ ***props***(C) *to* $\boldsymbol{p}' \in$ ***props***(C_1)*, such that* $\boldsymbol{p}$ *is a property with reference type:* $\natural$C′ *and* $\boldsymbol{p}'$ *is a property or a reference path with reference type:* $\natural\mathrm{C}'_1$*, with* C′ *and* C'_1 *being semantically equivalents, then there is a matching function in $\hat{\mathcal{F}}$ to match objects in* C'_1 *to objects in* C′*. The same applies to the properties* $\boldsymbol{p}$ *and* $\boldsymbol{p}'$ *being collections of reference types or structural types with at least one component of the reference type.*

(b) *For each PCA or ACA of* C *connecting* $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{C})$ *to* $\boldsymbol{p}_1 \in \boldsymbol{props}(\mathrm{C}_1)$, *such that there are the foreign key declarations* FK= $\big(\boldsymbol{F}$,C, $\{\boldsymbol{p}\}$, C′, $\{\boldsymbol{p}'\}\big)$ *and* $\mathrm{FK}_1 = \big(\boldsymbol{F}_1$,$\mathrm{C}_1$, $\{\boldsymbol{p}_1\}$, C_2, $\{\boldsymbol{p}_2\}\big)$, *and there is not any PCA or ACA connecting* $\boldsymbol{p}'$ *to* $\boldsymbol{p}_2$; *then there is a matching function in* $\hat{\mathcal{F}}$ *to match instances of* C′ *to instances of* C_2.

(c) *For each ECA of equivalence or selection relating* C *to* C_1, *there is a matching function declaration in* $\hat{\mathcal{F}}$ *to match instances of* C_1 *with instances of* C.

(d) *For each ECA of difference relating* C *to* C_1 *and* C_2, *there are at least two matching function declarations in* $\hat{\mathcal{F}}$: *one to match instances of* C_1 *with instances of* C_2, *and other to match instances of* C_2 *with instances of* C.

(e) *For each ECA of union relating* C *to* C_1, C_2, ..., C_n, *there are at least* n *matching function declarations in* $\hat{\mathcal{F}}$ *to match instances of* C_i *with instances of* C, *for* $1 \leq i \leq n$.

(f) *For each ECA of intersection relating* C *to* C_1, C_2, ..., C_n, *there are at least* $2n-1$ *matching function declarations in* $\hat{\mathcal{F}}$: n *to match instances of* C_i *with instances of* C, *and* $n-1$ *to match instances of* C_i *with instances of* C_{i+1}, *for* $1 \leq i \leq n$. □

Hereafter we will work with only valid perspective schemata.

3.4. Instantiations

The instances of a perspective schema include typed values, objects, and tuples, just like occurs in a schema. Here, however, the object identities are defined as a function of the object identities from which the (new) objects are originated. Therewith, we can explicitly express the notion of the link between the target schema and the base schemata. This idea is not original and follows the style of Lacroix et al. in [40]. An object identity in a perspective schema is defined by the following grammar:

$$\begin{array}{lll} \mathbf{o} & ::= & (\textbf{base}, \mathrm{C}) \quad | \\ & & \mathbf{o}' \\ \\ \textbf{base} & ::= & (\mathbf{o}_i, \mathrm{C}_i) \quad | \\ & & (\mathbf{o}_i, \mathrm{C}_i) \wedge \textbf{base} \end{array}$$

Being that:

- C is the name of the class in a perspective schema, which the object, whose identity is **o**, belongs.
- $\mathbf{o}'$ is an object identity automatically-generated by the system[6]. It is done when there is not one-to-one correspondence between the objects of the target schema and the base schemata.

[6]Remember that an object automatically generated in our proposal is composed from the symbol ♯ followed by a positive number.

- C_i is the name of a class or relation in a base schema that is related to C through some ECA or SCA.
- $\mathbf{o}_i$ is an object identity with one of two connotations:
  1. It is the object identity of the object belonging to C_i from which the object of C is derived; or
  2. It is an automatically-generated object identity that, in turn, is assigned, in some way, to the tuple belonging to C_i from which the object of C is derived[7].
- "$\wedge$" is symmetric, i.e. the words $\textbf{base}_1 \wedge \textbf{base}_2$ and $\textbf{base}_2 \wedge \textbf{base}_1$ denote the same object identity.

The use of this approach allows us to represent an object as it behaves in a given class. Thus, a same object can belong to different classes, have diverse values and behaviours, and is still uniquely identified by its object identity. In the literature it is known by *multiple instantiation* (cf. [41]).

Example 6

Suppose that there is a perspective schema S_v*, which is an integrated view of schemata* S_1 *and* S_2 *shown in Figure 3.* S_v *contains the class* CUSTOMER_{Sv} *(with properties* ***id, name, contact,region****) which holds customers from classes* CUSTOMER *(from schema* S_1*) and* PURCHASE *(from schema* S_2*). The instances of* CUSTOMER_{Sv} *are, e.g., as follows:*

$(((\ \sharp 10, \text{CUSTOMER}\ ), \text{CUSTOMER}_{Sv}\ ), \{$***id****: 234,****name****:'John',****contact****:nil,****region****:12**$\}\ )$,
$(((\ \sharp 1, \text{PURCHASE}\ ), \text{CUSTOMER}_{Sv}\ ), \{$***id****:123,****name****:'Ann',****contact****:214567890,****region****:nil**$,\}\ )$
$(((\ \sharp 31, \text{CUSTOMER}\ ) \wedge (\sharp 99, \text{PURCHASE}), \text{CUSTOMER}_{Sv}\ ), \{$***id****:333,****name****:'Pat',****contact****:9823,****region****:67**$\}\ )$.

The definition of an instance or state of a perspective schema is the same, mutatis mutandis, to definition of an instance of a schema in language L_S. This means that, given a perspective schema **PS**, $\boldsymbol{D}(\textbf{PS}) = (\hat{\mathcal{O}}, \mathcal{W}_t)$ is an instance of **PS**, being $\hat{\mathcal{O}}$ a set of object identities and $\mathcal{W}_t$ a set of tuples such that each tuple in $\mathcal{W}_t$ is of the type of either some relation declaration in $\hat{\mathcal{R}}$ or some require declaration in $\widehat{\mathcal{R}q}$. However, the validity of a state of a perspective schema involves more tasks than the validity of a state of a schema described in L_S. Usually, a state of a schema is valid if the reference typed values, key declarations, and foreign key declarations defined in the schema are valid reference typed values, valid key declarations, and valid foreign key declarations. In a state of a perspective schema, it also must occurs, but we also have to assure the validity of the correspondence assertions specified in the perspective schema. The following text shows definitions of valid correspondence assertions and of a valid perspective schema. In order to understand better these definitions, it is necessary remember three concepts defined in language L_S, as follows:

- ***dom***() is a function that determines the set of all possible values of a given structure in a given state (instance) of the schema.
- $\boldsymbol{D}_b$(C) (read base class extent of a class C) is the set of object identities in $\hat{\mathcal{O}}$ that are assigned to a given class declaration (whose name is C) in a given state of the schema.

[7] It should be done in order to guarantee that the object identity automatically-generated by the system always refers to the same tuple in the source.

- $\boldsymbol{D}(\mathrm{C})$ (read class/relation extent of a given class/relation) is the set of object identities (or tuples) in $\hat{\mathcal{O}}$ (or in $\mathcal{W}_t$) that are assigned to a given class declaration (or relation declaration) in a given instance of the schema, taking into account the class hierarchy (when applied).

Definition 9 (*Valid PCAs) Let* $\boldsymbol{PS} = \big(\mathrm{S}, \hat{\mathcal{L}}_b, \big(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\big), \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *or* $\boldsymbol{PS} = \big(\mathrm{S}, \hat{\mathcal{L}}_b, \big(\mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}}\big), \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *be a perspective schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, *and* $\mathcal{K}$, $\boldsymbol{D}\big(\boldsymbol{PS}\big)$ *an instance of* $\boldsymbol{PS}$, *and* $\boldsymbol{D}\big(\boldsymbol{S}'\big)$ *an instance of a schema* $\boldsymbol{S}'$ *in* $\hat{\mathcal{L}}_b$. *Let also* C *be the name of a class or relation belonging to* $\widehat{\mathcal{R}q}$, $\mathcal{H}$, *or* $\hat{\mathcal{R}}$, $\psi \in \mathcal{A}$ *be a PCA of* C, C_i *be names of classes or relations defined in schema* $\boldsymbol{S}'$ *and presenting in* ψ. ψ *is valid in a state* $\boldsymbol{D}\big(\boldsymbol{PS}\big)$ *iff for all object identities* $\big($*tuples*$\big)$ $\boldsymbol{o}$ *and* $\boldsymbol{o}'$ *such that* $\boldsymbol{o} \in \boldsymbol{D}\big(\mathrm{C}\big)$ *and* $\boldsymbol{o}' \in \boldsymbol{D}\big(\mathrm{C}_i\big)$, *if* $\boldsymbol{o}$ *is semantically equivalent to* $\boldsymbol{o}'$ $\big($*represented by* $\boldsymbol{o} \equiv \boldsymbol{o}'\big)$, *then:*

1. *If* $\psi : \boldsymbol{PS}[\mathrm{C}] \bullet \boldsymbol{p} \rightarrow \boldsymbol{A}$:
   (a) *If* $\boldsymbol{A} = \boldsymbol{S}'[\mathrm{C}_i] \bullet \boldsymbol{p}'$ *then* $\boldsymbol{o} \bullet \boldsymbol{p} = \boldsymbol{o}' \bullet \boldsymbol{p}'$;
   (b) *If* $\boldsymbol{A} = \varphi(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n)$ *then* $\boldsymbol{o} \bullet \boldsymbol{p} = \boldsymbol{o}' \bullet \boldsymbol{dom}(\varphi(\boldsymbol{dom}(\boldsymbol{A}_1), \ldots, \boldsymbol{dom}(\boldsymbol{A}_n)))$.
   (c) *If* $\boldsymbol{A} = \boldsymbol{S}'[\mathrm{C}_i] \bullet \boldsymbol{p}'\{\boldsymbol{p}''_t\}$ *then* $\boldsymbol{o} \bullet \boldsymbol{p} = \boldsymbol{o}' \bullet \boldsymbol{p}' \bullet \boldsymbol{p}''$;
2. *If* $\psi : \boldsymbol{PS}[\mathrm{C}] \bullet \boldsymbol{p} \rightarrow (\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n)$ *then* $\boldsymbol{o} \bullet \boldsymbol{p} = \boldsymbol{o}' \bullet \boldsymbol{dom}(\boldsymbol{A}_1) \cup \cdots \cup \boldsymbol{o}' \bullet \boldsymbol{dom}(\boldsymbol{A}_n)$. *Specifically if each* $\boldsymbol{A}_i = \boldsymbol{S}'[\mathrm{C}_i] \bullet \boldsymbol{p}'_i$ *then* $\boldsymbol{o} \bullet \boldsymbol{p} = \boldsymbol{o}' \bullet \boldsymbol{p}'_1 \cup \cdots \cup \boldsymbol{o}' \bullet \boldsymbol{p}'_n$.
3. *If* $\psi : \boldsymbol{PS}[\mathrm{C}] \bullet \boldsymbol{p} \rightarrow (\boldsymbol{B}_1, \boldsymbol{pred}_1); \ldots, (\boldsymbol{B}_{n-1}, \boldsymbol{pred}_{n-1}); \boldsymbol{B}_n$:
   (a) *If* $\boldsymbol{pred}_i$, $1 \leq i \leq$ *n-1, is the first predicate that is truthful, then* $\boldsymbol{o} \bullet \boldsymbol{p} = \boldsymbol{o}' \bullet \boldsymbol{dom}(\boldsymbol{B}_i)$;
   (b) *If all* $\boldsymbol{pred}_i$, $1 \leq i \leq$ *n-1, is false then* $\boldsymbol{o} \bullet \boldsymbol{p} = \boldsymbol{o}' \bullet \boldsymbol{dom}(\boldsymbol{B}_n)$.
4. *If* $\psi : \boldsymbol{PS}[\mathrm{C}] \bullet \boldsymbol{p}\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\} \rightarrow (\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n)$ *then each* $\boldsymbol{o} \bullet \boldsymbol{p} \bullet \boldsymbol{p}_i = \boldsymbol{o}' \bullet \boldsymbol{dom}(\boldsymbol{A}_i)$, $1 \leq i \leq n$. *Specifically if each* $\boldsymbol{A}_i = \boldsymbol{S}'[\mathrm{C}_i] \bullet \boldsymbol{p}'_i$ *then each* $\boldsymbol{o} \bullet \boldsymbol{p} \bullet \boldsymbol{p}_i = \boldsymbol{o}' \bullet \boldsymbol{p}'_i$. □

Definition 10 (*Valid ECAs) Let* $\boldsymbol{PS} = \big(\mathrm{S}, \hat{\mathcal{L}}_b, \big(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\big), \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *or* $\boldsymbol{PS} = \big(\mathrm{S}, \hat{\mathcal{L}}_b, \big(\mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}}\big), \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *be a perspective schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, *and* $\mathcal{K}$, *and* $\boldsymbol{D}\big(\boldsymbol{PS}\big)$ *an instance of* $\boldsymbol{PS}$, *and* $\boldsymbol{D}\big(\boldsymbol{S}_i\big)$ *instances of schemata* $\boldsymbol{S}_i$ *in* $\hat{\mathcal{L}}_b$. *Let also* C *be the name of a class or relation belonging to* $\widehat{\mathcal{R}q}$, $\mathcal{H}$, *or* $\hat{\mathcal{R}}$, $\psi \in \mathcal{A}$ *be a ECA of* C, C_i *be names of classes or relations defined in schemata* $\boldsymbol{S}_i$ *and presenting in* ψ. ψ *is valid in a state* $\boldsymbol{D}\big(\boldsymbol{PS}\big)$ *iff it satisfy the following constraints:*

1. *If* ψ *is an ECA of equivalence* $\big(\psi : \boldsymbol{PS}[\mathrm{C}] \rightarrow \mathrm{S}_1[\mathrm{C}_1]\big)$, *then for all object identity* $\big($*or tuple*$\big)$ $\boldsymbol{o} \in \boldsymbol{D}\big(\mathrm{C}\big)$ *in perspective schema* $\boldsymbol{PS}$, *there is an object identity* $\big($*or tuple*$\big)$ $\boldsymbol{o}_1 \in \boldsymbol{D}\big(\mathrm{C}_1\big)$ *in schema* S_1 *such that* $\boldsymbol{o}$ *and* $\boldsymbol{o}_1$ *are semantically equivalents* $\big($*i.e.* $\boldsymbol{o} \equiv \boldsymbol{o}_1\big)$, *and vice-versa;*
2. *If* ψ *is an ECA of selection* $\big(\psi : \boldsymbol{PS}[\mathrm{C}] \rightarrow \mathrm{S}_1[\mathrm{C}_1(\boldsymbol{pred})]\big)$, *then for all object identity (or tuple)* $\boldsymbol{o} \in \boldsymbol{D}\big(\mathrm{C}\big)$ *in perspective schema* $\boldsymbol{PS}$, *there is an object identity (or tuple)* $\boldsymbol{o}_1 \in \boldsymbol{D}\big(\mathrm{C}_1\big)$ *in schema* S_1 *whose the predicate "*$\boldsymbol{pred}$*" is true and such that* $\boldsymbol{o} \equiv \boldsymbol{o}_1$, *and vice-versa;*
3. *If* ψ *is an ECA of difference* $\big(\psi : \boldsymbol{PS}[\mathrm{C}] \rightarrow \mathrm{S}_1[\mathrm{C}_1] - \mathrm{S}_2[\mathrm{C}_2]\big)$, *then for all object identity (or tuple)* $\boldsymbol{o} \in \boldsymbol{D}\big(\mathrm{C}\big)$ *in perspective schema* $\boldsymbol{PS}$, *there is an object identity (or tuple)* $\boldsymbol{o}_1 \in \boldsymbol{D}\big(\mathrm{C}_1\big)$ *in schema* S_1 *such that* $\boldsymbol{o} \equiv \boldsymbol{o}_1$, *and there is not one object identity (or tuple)* $\boldsymbol{o}_2 \in \boldsymbol{D}\big(\mathrm{C}_2\big)$ *in schema* S_2 *such that* $\boldsymbol{o} \equiv \boldsymbol{o}_2$, *and vice-versa;*
4. *If* ψ *is an ECA of union* $\big(\psi : \boldsymbol{PS}[\mathrm{C}] \rightarrow \mathrm{S}_1[\mathrm{C}_1] \rtimes\!\!\sqsubset \ldots \rtimes\!\!\sqsubset \mathrm{S}_n[\mathrm{C}_n]\big)$, *then for all object identity (or tuple)* $\boldsymbol{o} \in \boldsymbol{D}\big(\mathrm{C}\big)$ *in perspective schema* $\boldsymbol{PS}$, *there is an object identity (or tuple)* $\boldsymbol{o}_1 \in \boldsymbol{D}\big(\mathrm{C}_1\big)$ *in schema* S_1 *such that* $\boldsymbol{o} \equiv \boldsymbol{o}_1$, *or there is an object identity (or tuple)* $\boldsymbol{o}_i \in \boldsymbol{D}\big(\mathrm{C}_i\big)$ *in schema* S_i *such that* $\boldsymbol{o} \equiv \boldsymbol{o}_i$, *or there is an object identity (or tuple)* $\boldsymbol{o}_n \in \boldsymbol{D}\big(\mathrm{C}_n\big)$ *in schema* S_n *such that* $\boldsymbol{o} \equiv \boldsymbol{o}_n$, *for* $1 \leq i \leq n$, *and vice-versa;*

5. *If ψ is an ECA of intersection $\big(\psi : \mathbf{PS}[\mathrm{C}] \rightarrow \mathrm{S}_1[\mathrm{C}_1] \cap \mathrm{S}_2[\mathrm{C}_2] \cap \cdots \cap \mathrm{S}_n[\mathrm{C}_n]\big)$, then for all object identity (or tuple) $\boldsymbol{o} \in \boldsymbol{D}(\mathrm{C})$ in perspective schema **PS**, there is an object identity (or tuple) $\boldsymbol{o}_1 \in \boldsymbol{D}(\mathrm{C}_1)$ in schema S_1 such that $\boldsymbol{o} \equiv \boldsymbol{o}_1$, and there is an object identity (or tuple) $\boldsymbol{o}_i \in \boldsymbol{D}(\mathrm{C}_i)$ in schema S_i such that $\boldsymbol{o} \equiv \boldsymbol{o}_i$, and there is an object identity (or tuple) $\boldsymbol{o}_n \in \boldsymbol{D}(\mathrm{C}_n)$ in schema S_n such that $\boldsymbol{o} \equiv \boldsymbol{o}_n$, for $1 \leq i \leq n$, and vice-versa.*
6. *If ψ is an ECA of form $\big(\psi : \mathbf{PS}\big[\mathrm{C}(\mathbf{pred})\big] \rightarrow \mathrm{S}_1[\mathrm{C}_1]$, then for all object identity (or tuple) $\boldsymbol{o} \in \boldsymbol{D}(\mathrm{C})$ in perspective schema **PS**, whose **pred** is true, there is an object identity (or tuple) $\boldsymbol{o}_1 \in \boldsymbol{D}(\mathrm{C}_1)$ in schema S_1 such that $\boldsymbol{o} \equiv \boldsymbol{o}_1$, and vice-versa; mutatis mutandis for the remainder kinds of ECA that have a predicate of selection in the class/relation of the perspective schema.* □

Definition 11 (*Valid SCA) Let* $\mathbf{PS} = \big(\mathrm{S}, \hat{\mathcal{L}}_b, \big(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\big), \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *or* $\mathbf{PS} = \big(\mathrm{S}, \hat{\mathcal{L}}_b, \big(\mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}}\big), \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *be a perspective schema defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, and $\mathcal{K}$, and $\boldsymbol{D}(\mathbf{PS})$ an instance of **PS**, and $\boldsymbol{D}(\boldsymbol{S}_i)$ instances of schemata $\boldsymbol{S}_i$ in $\hat{\mathcal{L}}_b$. Let also* C *be the name of a class or relation belonging to $\widehat{\mathcal{R}q}$, $\mathcal{H}$, or $\hat{\mathcal{R}}$, $\psi \in \mathcal{A}$ be a SCA of* C, C_i *be names of classes or relations defined in schemata $\boldsymbol{S}_i$ and presenting in ψ. ψ is valid in a state $\boldsymbol{D}(\mathbf{PS})$ iff it satisfy the following constraints:*

1. *If ψ is a SCA of the form $\mathrm{S}[\mathrm{C}](\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n) \rightarrow$ **groupby**$(\mathrm{S}'[\mathrm{C}'](\boldsymbol{p}'_1, \boldsymbol{p}'_2, \ldots, \boldsymbol{p}'_n)$, then for all object identity (or tuple) $\boldsymbol{o} \in \boldsymbol{D}(\mathrm{C})$ in perspective schema **PS** there is a set of object identities (or tuples) $\boldsymbol{o}'_i \in \boldsymbol{D}(\mathrm{C}')$ in schema S', $i \geq 1$, such that $\boldsymbol{o}.\boldsymbol{p}_j = \boldsymbol{o}'_i.\boldsymbol{p}'_j$, for $1 \leq j \leq n$. We say that $\boldsymbol{o}$ is derived of $\boldsymbol{o}'_1, \boldsymbol{o}'_2, \ldots, \boldsymbol{o}'_i$, $i \geq 1$. Mutatis mutandis for $\mathrm{S}[\mathrm{C}](\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n) \rightarrow$ **normalize**$(\mathrm{S}'[\mathrm{C}'](\boldsymbol{p}'_1, \boldsymbol{p}'_2, \ldots, \boldsymbol{p}'_n)$.*
2. *If ψ is a SCA of the form $\mathrm{S}[\mathrm{C}](\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n) \rightarrow$ **groupby**$(\mathrm{S}'[\mathrm{C}'(\mathbf{pred})](\boldsymbol{p}'_1, \boldsymbol{p}'_2, \ldots, \boldsymbol{p}'_n)$, then for all object identity (or tuple) $\boldsymbol{o} \in \boldsymbol{D}(\mathrm{C})$ in perspective schema **PS** there is a set of object identities (or tuples) $\boldsymbol{o}'_i \in \boldsymbol{D}(\mathrm{C}')$ in schema S', $i \geq 1$, whose the predicate "**pred**" is true and such that $\boldsymbol{o}.\boldsymbol{p}_j = \boldsymbol{o}'_i.\boldsymbol{p}'_j$, for $1 \leq j \leq n$. We say that $\boldsymbol{o}$ is derived of $\boldsymbol{o}'_1, \boldsymbol{o}'_2, \ldots, \boldsymbol{o}'_i$, $i \geq 1$. Mutatis mutandis for $\mathrm{S}[\mathrm{C}](\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n) \rightarrow$ **normalize**$(\mathrm{S}'[\mathrm{C}'(\mathbf{pred})](\boldsymbol{p}'_1, \boldsymbol{p}'_2, \ldots, \boldsymbol{p}'_n)$.* □

Definition 12 (*Valid ACA) Let* $\mathbf{PS} = \big(\mathrm{S}, \hat{\mathcal{L}}_b, \big(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\big), \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *or* $\mathbf{PS} = \big(\mathrm{S}, \hat{\mathcal{L}}_b, \big(\mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}}\big), \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *be a perspective schema defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, and $\mathcal{K}$, $\boldsymbol{D}(\mathbf{PS})$ an instance of **PS**, $\boldsymbol{D}(\mathrm{S}')$ an instance of a schema S' in $\hat{\mathcal{L}}_b$, and **A** an expression as defined in Definition 1. Let also* C *be the name of a class or relation belonging to $\widehat{\mathcal{R}q}$, $\mathcal{H}$, or $\hat{\mathcal{R}}$, ψ and $\psi' \in \mathcal{A}$ be, respectively, an ACA of* C *and a SCA of* C, C' *be names of classes or relations defined in schema S' and presenting in ψ. ψ is valid in a state $\boldsymbol{D}(\mathbf{PS})$ iff for all object identities (tuples) $\boldsymbol{o}$ and $\boldsymbol{o}'_i$ such that $\boldsymbol{o} \in \boldsymbol{D}(\mathrm{C})$ and $\boldsymbol{o}'_i \in \boldsymbol{D}(\mathrm{C}_i)$, $1 \leq i \leq n$, if $\boldsymbol{o}$ is derived of $\boldsymbol{o}'_i$, $1 \leq i \leq n$, then:*

1. *If ψ is of form $\mathrm{S}[\mathrm{C}]\bullet\boldsymbol{p} \rightarrow \psi', \gamma(\mathbf{A})$ such that:*
   - *γ is the function **count** then $\boldsymbol{o}\bullet\boldsymbol{p} = \boldsymbol{w}$, being $\boldsymbol{w}$ the total number of distinct values of $\boldsymbol{o}'_i.\mathbf{dom}(\mathbf{A})$, $1 \leq i \leq n$. Specifically, when $\mathbf{A} = \mathrm{S}'[\mathrm{C}']\bullet\boldsymbol{p}'$, then $\boldsymbol{w}$ is the total number of distinct values of $\boldsymbol{o}'_i.\boldsymbol{p}'$.*
   - *γ is the function **sum** then $\boldsymbol{o}\bullet\boldsymbol{p} = \boldsymbol{o}'_1.\mathbf{dom}(\mathbf{A}) + \boldsymbol{o}'_2.\mathbf{dom}(\mathbf{A}) + \ldots + \boldsymbol{o}'_n.\mathbf{dom}(\mathbf{A})$. Specifically, when $\mathbf{A} = \mathrm{S}'[\mathrm{C}']\bullet\boldsymbol{p}'$, then $\boldsymbol{o}.\boldsymbol{p} = \boldsymbol{o}'_1.\boldsymbol{p}' + \boldsymbol{o}'_2.\boldsymbol{p}' + \ldots + \boldsymbol{o}'_n.\boldsymbol{p}'$.*
   - *γ is the function **min** then $\boldsymbol{o}\bullet\boldsymbol{p} = \boldsymbol{w}$, being $\boldsymbol{w}$ the least value of all $\boldsymbol{o}'_i.\mathbf{dom}(\mathrm{C}')$, $1 \leq i \leq n$. Specifically, when $\mathbf{A} = \mathrm{S}'[\mathrm{C}']\bullet\boldsymbol{p}'$, then $\boldsymbol{w}$ is the least value of all $\boldsymbol{o}'_i.\boldsymbol{p}'$, $1 \leq i \leq n$.*

- γ *is the function* ***max*** *then* $\boldsymbol{o}\bullet\boldsymbol{p}=\boldsymbol{w}$, *being* $\boldsymbol{w}$ *the greast value of all* $\boldsymbol{o'}_i.\boldsymbol{dom}(\mathrm{C'})$, $1 \leq i \leq n$. *Specifically, when* $\boldsymbol{A}=\mathrm{S'[C']}\bullet\boldsymbol{p'}$, *then* $\boldsymbol{w}$ *is the least value of all* $\boldsymbol{o'}_i.\boldsymbol{p'}$, $1 \leq i \leq n$.
- γ *is the function* ***avg*** *then* $\boldsymbol{o}\bullet\boldsymbol{p}=\frac{\boldsymbol{o'}_1.\boldsymbol{dom}(\boldsymbol{A})+\boldsymbol{o'}_2.\boldsymbol{dom}(\boldsymbol{A})+\ldots+\boldsymbol{o'}_n.\boldsymbol{dom}(\boldsymbol{A})}{n}$. *Specifically, when* $\boldsymbol{A}=\mathrm{S'[C']}\bullet\boldsymbol{p'}$, *then* $\boldsymbol{o}.\boldsymbol{p}=\frac{\boldsymbol{o'}_1.\boldsymbol{p'}+\boldsymbol{o'}_2.\boldsymbol{p'}+\ldots+\boldsymbol{o'}_n.\boldsymbol{p'}}{n}$.

2. *If* ψ *is of form* $\mathrm{S[C]}\bullet\boldsymbol{p}\rightarrow\psi',\gamma(\boldsymbol{A},\boldsymbol{pred})$ *such that:*
   - γ *is the function* ***count*** *then* $\boldsymbol{o}\bullet\boldsymbol{p}=\boldsymbol{w}$, *being* $\boldsymbol{w}$ *the total number of distinct values of* $\boldsymbol{o'}_i.\boldsymbol{dom}(\boldsymbol{A})$, $1 \leq i \leq m$, $m \leq n$, *whose the predicate "****pred****" is true. Specifically, when* $\boldsymbol{A}=\mathrm{S'[C']}\bullet\boldsymbol{p'}$, *then* $\boldsymbol{w}$ *is the total number of distinct values of* $\boldsymbol{o'}_i.\boldsymbol{p'}$, $1 \leq i \leq m$, $m \leq n$, *whose* ***pred*** *is true.*
   - γ *is the function* ***sum*** *and m is the number of object identities* $\boldsymbol{o'}_i$ *whose* ***pred*** *is true, then* $\boldsymbol{o}\bullet\boldsymbol{p}=\boldsymbol{o'}_1.\boldsymbol{dom}(\boldsymbol{A})+\boldsymbol{o'}_2.\boldsymbol{dom}(\boldsymbol{A})+\ldots+\boldsymbol{o'}_m.\boldsymbol{dom}(\boldsymbol{A})$, *such that* $m \leq n$. *Specifically, when* $\boldsymbol{A}=\mathrm{S'[C']}\bullet\boldsymbol{p'}$, *then* $\boldsymbol{o}.\boldsymbol{p}=\boldsymbol{o'}_1.\boldsymbol{p'}+\boldsymbol{o'}_2.\boldsymbol{p'}+\ldots+\boldsymbol{o'}_m.\boldsymbol{p'}$.
   - γ *is the function* ***min*** *and m is the number of object identities* $\boldsymbol{o'}_i$ *whose* ***pred*** *is true, then* $\boldsymbol{o}\bullet\boldsymbol{p}=\boldsymbol{w}$, *being* $\boldsymbol{w}$ *the least value of all* $\boldsymbol{o'}_i.\boldsymbol{dom}(\mathrm{C'})$, $1 \leq i \leq m$, $m \leq n$. *Specifically, when* $\boldsymbol{A}=\mathrm{S'[C']}\bullet\boldsymbol{p'}$, *then* $\boldsymbol{w}$ *is the least value of all* $\boldsymbol{o'}_i.\boldsymbol{p'}$, $1 \leq i \leq m$.
   - γ *is the function* ***max*** *and m is the number of object identities* $\boldsymbol{o'}_i$ *whose* ***pred*** *is true, then* $\boldsymbol{o}\bullet\boldsymbol{p}=\boldsymbol{w}$, *being* $\boldsymbol{w}$ *the greast value of all* $\boldsymbol{o'}_i.\boldsymbol{dom}(\mathrm{C'})$, $1 \leq i \leq m$, $m \leq n$. *Specifically, when* $\boldsymbol{A}=\mathrm{S'[C']}\bullet\boldsymbol{p'}$, *then* $\boldsymbol{w}$ *is the least value of all* $\boldsymbol{o'}_i.\boldsymbol{p'}$, $1 \leq i \leq m$.
   - γ *is the function* ***avg*** *m is the number of object identities* $\boldsymbol{o'}_i$ *whose* ***pred*** *is true, then* $\boldsymbol{o}\bullet\boldsymbol{p}=\frac{\boldsymbol{o'}_1.\boldsymbol{dom}(\boldsymbol{A})+\boldsymbol{o'}_2.\boldsymbol{dom}(\boldsymbol{A})+\ldots+\boldsymbol{o'}_m.\boldsymbol{dom}(\boldsymbol{A})}{m}$. *Specifically, when* $\boldsymbol{A}=\mathrm{S'[C']}\bullet\boldsymbol{p'}$, *then* $\boldsymbol{o}.\boldsymbol{p}=\frac{\boldsymbol{o'}_1.\boldsymbol{p'}+\boldsymbol{o'}_2.\boldsymbol{p'}+\ldots+\boldsymbol{o'}_m.\boldsymbol{p'}}{m}$.
3. *If* ψ *is of form* $\mathrm{S[C]}\bullet\boldsymbol{p}\rightarrow\psi',\boldsymbol{A}$, *then* $\boldsymbol{o}.\boldsymbol{p}=\boldsymbol{o'}_1.\boldsymbol{dom}(\boldsymbol{A})=\boldsymbol{o'}_2.\boldsymbol{dom}(\boldsymbol{A})=\ldots=\boldsymbol{o'}_n.\boldsymbol{dom}(\boldsymbol{A})$. *Specifically, when* $\boldsymbol{A}=\mathrm{S'[C']}\bullet\boldsymbol{p'}$, *then* $\boldsymbol{o}.\boldsymbol{p}=\boldsymbol{o'}_i.\boldsymbol{p'}$, $1 \leq i \leq n$.
4. *If* ψ *is of form* $\mathrm{S[C]}\bullet\boldsymbol{p}\rightarrow\psi',(\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_m)$ *then* $\boldsymbol{o}.\boldsymbol{p}=\boldsymbol{o'}_i.\boldsymbol{dom}(\boldsymbol{A}_1)\cup\boldsymbol{o'}_i.\boldsymbol{dom}(\boldsymbol{A}_2)\cup\ldots\boldsymbol{o'}_i.\boldsymbol{dom}(\boldsymbol{A}_m)$. *Specifically, when* $\boldsymbol{A}_j=\mathrm{S'[C']}\bullet\boldsymbol{p'}_j$, *then* $\boldsymbol{o}.\boldsymbol{p}=\boldsymbol{o'}_i.\boldsymbol{p'}_j$, $1 \leq j \leq m$, $1 \leq i \leq n$.
5. *If* ψ *is of form* $\mathrm{S[C]}\bullet\boldsymbol{p}\rightarrow\psi'$, $(\boldsymbol{B}_1,\boldsymbol{pred}_1);\ldots,(\boldsymbol{B}_{m-1},\boldsymbol{pred}_{m-1});\boldsymbol{B}_m$ *then:*
   (a) *If* $\boldsymbol{pred}_j$, $1 \leq j \leq$ *m-1, is the first predicate that is truthful, then* $\boldsymbol{o}\bullet\boldsymbol{p}=\boldsymbol{o'}_i\bullet\boldsymbol{dom}(\boldsymbol{B}_j)$, $1 \leq i \leq n$;
   (b) *If all* $\boldsymbol{pred}_j$, $1 \leq j \leq$ *m-1, is false then* $\boldsymbol{o}\bullet\boldsymbol{p}=\boldsymbol{o'}_i\bullet\boldsymbol{dom}(\boldsymbol{B}_m)$, $1 \leq i \leq n$. □

Definition 13 (*Valid Instance of a View Schema) Let* $\boldsymbol{PS}=(\mathrm{S},\hat{\mathcal{L}}_b,(\hat{\mathcal{L}}_t,\widehat{\mathcal{R}q}),\hat{\mathcal{A}},\hat{\mathcal{F}})$ *or* $\boldsymbol{PS}=(\mathrm{S},\hat{\mathcal{L}}_b,(\mathcal{H},\hat{\mathcal{R}},\hat{\mathcal{K}}),\hat{\mathcal{A}},\hat{\mathcal{F}})$ *be a a perspective schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, *and* $\mathcal{K}$, *and* $\boldsymbol{D}(\boldsymbol{PS})=(\hat{\mathcal{O}},\mathcal{W}_t)$ *be an instance of* $\boldsymbol{PS}$. *We say that an instance* $\boldsymbol{D}(\boldsymbol{PS})$ *is valid iff:*

- *All reference typed values in* $\boldsymbol{D}(\boldsymbol{PS})$ *are valid reference typed values;*
- *All key declarations in* $\boldsymbol{D}(\boldsymbol{PS})$ *are valid key declarations;*
- *All foreign key declarations in* $\boldsymbol{D}(\boldsymbol{PS})$ *are valid foreign key declarations.*
- *All Extension Correspondence Assertions in* $\boldsymbol{D}(\boldsymbol{PS})$ *are valid Extending Correspondence Assertions.*
- *All Property Correspondence Assertions in* $\boldsymbol{D}(\boldsymbol{PS})$ *are valid Property Correspondence Assertions.*
- *All Summation Correspondence Assertions in* $\boldsymbol{D}(\boldsymbol{PS})$ *are valid Summation Correspondence Assertions.*

- *All Aggregation Correspondence Assertions in* $\boldsymbol{D}(\boldsymbol{PS})$ *are valid Aggregation Correspondence Assertions.* □

Hereafter, we will only consider valid instances of perspective schemata.

4. INFERENCE MECHANISM

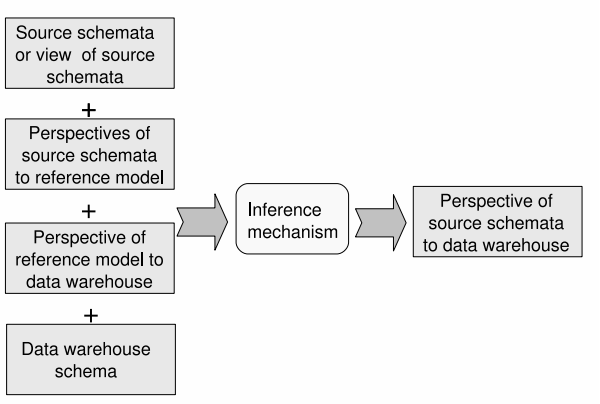

Figure 10: Sketch of the creation of an inferred perspective schema.

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. 10). 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 $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$ 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 $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$; and new CAs and new match function signatures will be created.

(a) CAs of the perspective $\mathbf{P}_{\mathbf{S1}} \mid \mathbf{DW}$:

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

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

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

ψ_2: $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}\,[\text{PRODUCT}] \bullet \mathbf{prod\_id}_{\text{DW}} \rightarrow \mathbf{RM}\,[\text{PRODUCT}] \bullet \mathbf{pid}_{\text{RM}} \Rightarrow$

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

ψ_1: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}\,[\text{PRODUCT}] \bullet \mathbf{pid}_{\text{RM}} \rightarrow \mathbf{S}_1\,[\text{PRODUCT}] \bullet \mathbf{prod\_id}_{\text{S}_1}$

Figure 11: Examples of inferred correspondence assertions and a CA-rewriting rule.

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:

$$\textbf{Rule:}\frac{A \Rightarrow B}{C} \quad \text{(read A is rewritten in B if C is valid)}\,, \tag{12}$$

being *C* one or more conditions that should be satisfied.

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

```
1: procedure INFER(P_T, P_1, ..., P_n, P_I)
2:     for each CA in P_T.caList do
3:         find all new CAs that are rewritten from CA using
4:         some CA-rewriting rule;
5:         add CAs to P_I.caList;
6:     end for
7:     for each MF in P_T.mfList do
8:         find all new MFs that are rewritten from MF
9:         using some MF-rewriting rule;
10:        add MFs to P_I.mfList
11:    end for
12:    for each E in classList/relationList/keyList do
13:        create a require declaration to P_I;
14:        add it, appropriately, to P_I.classList/
15:        P_I.relationList/P_I.keyList
16:    end for
17: end procedure
```

Figure 12: 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. 12. In Fig. 12 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 $\mathbf{P_{RM|DW}}$ (relating elements of the **DW** to elements of the **RM**) will exist a new CA in $\mathbf{P_{S_1|DW}}$. Fig. 5 only presents some CAs of $\mathbf{P_{S_1|RM}}$ and of $\mathbf{P_{RM|DW}}$. Based on these CAs, two new CAs can be inferred: ψ'_1 and ψ'_2 shown in Fig. 11(a). The process to deduce ψ'_1 is presented in Fig. 11(b).

match((**RM**[PRODUCT],τ_2)×(**DW**[PRODUCT (**current_flag** = true)],τ_3))→Boolean ⇒
match((**S**$_1$ [PRODUCT],τ_2)×(**DW**[PRODUCT (**current_flag** = true)],τ_3))→Boolean

---

ψ_4: $\mathbf{P_{S_1|RM}}$ [PRODUCT] → **S**$_1$ [PRODUCT]

Figure 13: Example of a rule for rewriting a match function signature.

For each match function signature originally defined to $\mathbf{P_{RM|DW}}$ will exists a new match function

signature in $\mathbf{P_{S_1|DW}}$. Fig. 4 only presents one match function signature of $\mathbf{P_{RM|DW}}$. 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. 13.

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 s 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) [42]. 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 [43]. 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 DW 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 DW 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.

5. 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. [44, 45, 4, 46, 47, 48]) as well as the materialised view definition and maintenance (e.g. [49]). In particular, the most conceptual multidimensional models are extensions to the Entity-Relationship model (e.g. [50, 51, 52, 53]) or extensions to UML (e.g. [54, 55, 56]). There are only some works involving conceptual models based on non-multidimensional aspects [57, 49, 6].

There are many approaches in existence [58, 6, 59] for dealing with the ETL activities in a conceptual setting. Although, 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.

Calvanese et al. in [6] presented a framework adopted in the Data Warehouse Quality project. Similar to this study, their proposal include 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.

Vassiliadis et al. in [58] proposed a conceptual model for dealing with interrelationships 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.

Skoutas and Simitsis in [59] 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), allowed 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. In our work it is dispensable as each schema (source or DW) needs to be related only to the reference model one. Additionally, in [59] there is nothing about the matching of instances and their ETL operations being a subset of transformations treated by us.

6. 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 our 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 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.

Actually, a prototype Prolog-based is being developed to allow the description of schemata and perspective schemata in our 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 will include translators from our language to the ISCO one. ISCO [42] allows access to heterogeneous data sources and to perform arbitrary computations. Thus, this research can be tested in a context nearer to reality.

For future work, investigations will be made into how the perspective schemata can be used to automate the materialisation of the ETL process. Another important direction for future work is the development of a graphical user-friend interface to declare the schemata in our language, and thus, to hide some

syntax details.

References

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

[2] Ricardo Fortuna Raminhos, "ETL state of the art", Tech. Rep., New University of Lisbon, June 2007.

[3] Claudia Imhoff, Nicholas Galemmo, and Jonathan G. Geiger, *Mastering Data Warehouse Design - Relational and Dimensional Techniques*, Wiley Publishing, 2003.

[4] Juan Manuel Pérez, Rafael Berlanga, María José Aramburu, and Torben Bach Pedersen, "A relevance-extended multi-dimensional model for a data warehouse contextualized with documents", in *DOLAP'05: Proceedings of the 8th ACM international workshop on Data warehousing and OLAP*, USA, 2005, pp. 19–28, ACM.

[5] Rosa Matias and Jo ao Carlos Gomes Moura Pires, "Revisiting the olap interaction to cope with spatial data and spatial data analysis", in *ICEIS 2007 - Proceedings of the Ninth International Conference on Enterprise Information Systems*, Jorge Cardoso, José Cordeiro, and Joaquim Filipe, Eds., 2007, vol. DISI, pp. 157–163.

[6] Diego Calvanese, Luigi Dragone, Daniele Nardi, Riccardo Rosati, and Stefano M. Trisolini, "Enterprise modeling and data warehousing in TELECOM ITALIA", *Inf. Syst.*, vol. 31, no. 1, pp. 1–32, 2006.

[7] Ralf Knackstedt and Karsten 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] Ralph Kimball, Margy Ross, Warren Thornthwaite, Joy Mundy, and Bob Becker, *The Data Warehouse Lifecycle Tookit*, Wiley Publishing, 2nd edition, 2008.

[9] Wayne Eckerson, "Four ways to build a data warehouse", *What works*, vol. 15, 2003.

[10] Edgar F. Codd, "A relational model of data for large shared data banks", in *Communications of the ACM*, 1970, pp. 377–387.

[11] Rick G.G. Cattell et al., *The Object Database Standard ODMG 3.0*, Morgan Kaufmann Publishers, 2000.

[12] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of database systems*, Pearson Education, 5th edition, 2006.

[13] Valéria Magalh aes Pequeno and Jo ao Carlos Gomes Moura Pires, "A formal object-relational data warehouse model", Tech. Rep., New University of Lisbon, November 2007.

[14] Alvaro E. Monge and Charles Elkan, "The field matching problem: Algorithms and applications", in *Knowledge Discovery and Data Mining*, 1996, pp. 267–270.

[15] Mauricio A. Hernandez and Salvatore J. Stolfo, "The merge/purge problem for large databases", in *SIGMOD Conference*, 1995, pp. 127–138.

[16] AnHai Doan et al., "Object matching for information integration: a profiler-based approach", in *Proceedings of IJCAI-03 Workshop on Information Integration on the Web*, Mexico, Aug. 2003.

[17] G. Zhou et al., "Generating data integration mediators that use materialization", *Journal of Intelligent Information Systems*, vol. 6(2/3), pp. 199–221, May 1996.

[18] Martin Michalowski, Snehal Thakkar, and Craig A. Knoblock, "Exploiting secondary sources for automatic object consolidation", in *Proceedings of the KDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation*, 2003, pp. 34–36.

[19] AnHai Doan and Alon Y. Halevy, "Semantic integration research in the database community: A brief survey", American Association for artificial intelligence, 2004.

[20] Yannis Papakonstantinou et al., "Object fusion in mediator systems", in *International Conference on Very Large Databases*, India, 1996.

[21] M. Ganesh, J. Sirvastava, and T. Richardsons, "Mining entity-identification rules for database integration", in *In Proceedings of the Second International Conference on Data Mining and Knowledge Discovery*, 1996, pp. 291–294.

[22] Dana Angluin, "Queries and concept learning", *Mach. Learn.*, vol. 2, no. 4, pp. 319–342, 1988.

[23] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock, "Retrieving and integrating data from multiple information sources", *International Journal on Intelligent and Cooperative Information Systems*, vol. 2(2), pp. 127–158, 1993.

[24] Michael Allen Bickel, "Automatic correction to misspelled names: A fourth-generation language approach", *Commun. ACM*, vol. 30, no. 3, pp. 224–228, 1987.

[25] K.D. Bollacker, S. Lawrence, and C.L. Giles, "Citeseer: An autonomous web agent for automatic retrieval and identication of interesting publications", in *In Proc. of 2nd Int. Conf. on Autonomous Agents*, USA, 1998.

[26] Mikhail Bilenko and Raymond J. Mooney, "Learning to combine trained distance metrics for duplicate detection in databases", Tech. Rep. Technical Report AI 02-296, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX, February 2002.

[27] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava, "Text joins for data cleansing and integration in an rdbms", in *In Proceedings of the 19th IEEE International Conference on Data Engineering - ICDE 2003- (poster paper)*, 2003.

[28] S. Sarawagi and A. Bhamidipaty, "Interactive deduplication using active learning", in *In The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, 2002.

[29] Sheila Tejada, Craig A. Knoblock, and Steven Minton, "Learning domain-independent string transformation weights for high accuracy object identification", in *In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, USA, 2002, pp. 350–359, ACM.

[30] William W. Cohen and Jacob Richman, "Learning to match and cluster large high-dimensional data sets for data integration", in *In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, USA, 2002, pp. 475–480, ACM.

[31] S. Widjojo et al., "A specificational approach to merging persistent object bases: the fourth international workshop on persistent object systems", in *Implementing Persistent Object Bases*, Al Dearle, Gail Sahw, and Stanley Zdonik, Eds. Morgan Kaufmann, Dec. 1990.

[32] W. Kent et al., "Object identification in multi-database systems", in *Interoperable Database Systems (DS-5) (A-25)*, D. Hsiao, E. Neuhold, and R. Sacks-Davis, Eds. Elsevier Science Publishers B. V., North-Holland, 1993.

[33] Sheila Tejada, *Learning object identification rules for information integration*, PhD thesis, Faculty of the graduate school. University of Southern California, USA, August 2002.

[34] Jemy A. Hylton, "Identifying and merging related bibliographic records", Master's thesis, Massachusetts Institute of Technology - Department of Electrical Engineering and Comouter Science, 1996.

[35] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos", in *Proceedings of the International Conference on Computer Vision*, October 2003, vol. 2, pp. 1470–1477.

[36] Ricardo da Silva Torres, *Ambiente de gerenciamento de imagens e dados espaciais para o desenvolvimento de aplicaes em biodiversidade*, PhD thesis, Universidade Estadual de Campinas - Instituto de Computao, 2004.

[37] R. Hull, "Managing semantic heteroneity in databases: a theoretical perspective", in *Proceedings of the 16th SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Tucson, Arizona, May 1997, pp. 51–61.

[38] Amit P. Sheth and Vipul Kashyap, "So far (schematically) yet so near (semantically)", in *Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*, Amsterdam, The Netherlands, The Netherlands, 1993, pp. 283–312, North-Holland Publishing Co.

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

[40] Zoé Lacroix, Claude Delobel, and Philippe Brèche, "Object views constructed with an object algebra", in *13ème Journées Bases de Données Avancées (Informal Proceedings)*, France, 1997.

[41] P. Donini and S. Monties, "Qualified Inheritance in Spatio-Temporal Databases", in *International Archives ofPhotogrammetry and Remote Sensing, Vol XXXIII,Part B*, 2000.

[42] Salvador Abreu and Vitor Nogueira, "Using a Logic Programming Language with Persistence and Contexts", in *Declarative Programming for Knowledge Management, 16th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2005, Fukuoka, Japan, October 22-24, 2005. Revised Selected Papers.*, Osamu Takata, Masanobu Umeda, Isao Nagasawa, Naoyuki Tamura, Armin Wolf, and Gunnar Schrader, Eds. 2006, vol. 4369 of *Lecture Notes in Computer Science*, pp. 38–47, Springer.

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

[44] Dov Dori, Roman Feldman, and Arnon Sturm, "From conceptual models to schemata: An object-process-based data warehouse construction method", *Inf. Syst.*, vol. 33, no. 6, pp. 567–593, 2008.

[45] 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.

[46] Matteo Golfarelli, Vittorio Maniezzo, and Stefano Rizzi, "Materialization of fragmented views in multidimensional databases", *Data Knowl. Eng.*, vol. 49, no. 3, pp. 325–351, 2004.

[47] Bodo Husemann, Jens Lechtenborger, and Gottfried Vossen, "Conceptual data warehouse modeling", in *Design and Management of Data Warehouses*, 2000, p. 6.

[48] 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.

[49] Robert 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.

[50] Enrico Franconi and Anand Kamble, "A data warehouse conceptual data model", *Proc. of the Int. Conf. on Scientific and Statistical Database Management*, vol. 00, pp. 435–436, 2004.

[51] Anand S. Kamble, "A conceptual model for multidimensional data", in *APCCM'08: Proceedings of the fifth on Asia-Pacific conference on conceptual modelling*, Australia, 2008, pp. 29–38, Australian Computer Society, Inc.

[52] Carsten Sapia, Markus Blaschka, Gabriele Höfling, and Barbara 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.

[53] Nectaria Tryfona, Frank Busborg, and Jens G. Borch Christiansen, "starER: a conceptual model for data warehouse design", in *DOLAP '99: Proceedings of the 2nd ACM international workshop on Data warehousing and OLAP*, USA, 1999, pp. 3–8, ACM.

[54] Sergio Luján-Mora, Juan Trujillo, and Il-Yeol Song, "A UML profile for multidimensional modelling in data warehouses", *Data Knowl. Eng.*, vol. 59, no. 3, pp. 725–769, 2005.

[55] T. B. Nguyen, A. Min Tjoa, and Roland Wagner, "An object oriented multidimensional data model for OLAP", in *Web-Age Information Management*, 2000, pp. 69–82.

[56] Juan Trujillo, Manuel Palomar, and Jaime Gmez, "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.

[57] Franck Ravat and Olivier Teste, "A temporal object-oriented data warehouse model", in *Proc. of the Int. Workshop on Database and Expert Systems Applications*, 2000, pp. 583–592.

[58] Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos, "Conceptual modeling for ETL processes", in *DOLAP'02: Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, USA, 2002, pp. 14–21, ACM.

[59] Dimitrios Skoutas and Alkis Simitsis, "Designing ETL processes using semantic web technologies", in *DOLAP'06: Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, USA, 2006, pp. 67–74, ACM.