Automatic generation of the schema of the Data Mart from OLAP Requirements

Nouha Arfaoui¹ and Jalel Akaichi²

¹BESTMOD- Institut Supérieur de Gestion, 41, Avenue de la liberté, Cité Bouchoucha, Le Bardo 2000, Tunisia

ABSTRACT

The data mart is defined as retail level of the DW. Different approaches and methods have been proposed to design it. In this work, we propose an automatic way to generate the schema of the data mart from the OLAP requirements that are stored as schemas. Our solution consists on merging the schemas that belong to the same domain using schema integration method. This latter compares in the first step the elements of the schemas using matching technique, then it merges the global schema using the mapping technique. The schema matching allows extracting the semantic correspondences and the conflicts that may exist. The schema mapping uses mainly to solve the conflicts to get at the end merged schemas. For each step, we will present the corresponding algorithm.

Keywords: OLAP Requirements Schema, Data Mart, Schema Integration, Schema Matching, Schema Merging, Schema Mapping

1. INTRODUCTION

The data mart “is defined as a flexible set of data, ideally based on the most atomic (granular) data possible to extract from an operational source, and presented in a symmetric (dimensional) model that is most resilient when faced with unexpected user queries”[31]. It is accessed directly by end users, and its data is structured in a way that is easy for users to understand and use [11].

In the literature, different approaches are proposed to design of the data mart [1]: Demand-driven approach where the data mart is designed starting from the user requirement, Supply-driven approach where the data mart is designed starting from the schema of a source operational database. Kimball in [31] suggests the construction of the data mart from the existing data warehouse. In addition to the previous ones, the authors in [2] propose a mixed approach which is based on three steps: 1) a first top-down step that makes possible eliciting and consolidating the user requirements and expectations using Goal/Question/Metric paradigm (they consider that the ideal data marts are derived from user requirements) 2) the second bottom-up step extracts the candidate data marts from the conceptual schema of the information system. 3) The last step compares ideal and candidate data marts.

Concerning the techniques, several ones have been proposed to automate the design of some parts of the data marts. We can mention [22] for the conceptual design, [12] for the logical design, etc. and others are concerned with automating the design of the whole data mart, such as [11] that suggests a method based on an enterprise data model (Entity-Relationship form).

In this work, we look to ensure the construction of the schema of the data mart in an automatically way. We opt for the top-down approach because we will use the data marts to construct our data warehouse and we use as input OLAP requirements that are stored as schemas.

We suppose that the OLAP requirements schemas (ORS) belong to the same domain to facilitate the generation of our schema since the data mart should be constructed for a specific business line or team. The ORSs are different in the structure and semantic. So, and in order to achieve our goal, we propose the use of schema integration technique. This latter is used to find all relationships between the different schemas which will be merged. The integration process is not an easy task and the basic problems are mainly because of the structural and semantic diversities of schemas that will be merged.

In order to use the schema integration 3 problems related to the heterogeneity must be solved. Those problems are described as follow: the first one is related to the data model heterogeneity. In this case, the problem arises when there is no guarantee that the data schemas share a common data model. As example: some local schemas can be modeled using object-oriented data model, and others using Entity Relationship data model. The second point is the structure heterogeneity. This problem exists when equivalent business concepts are modeled using different models in the same data model. As example we can mention the case when the same business concept e. “City” is employed once as dimension and another time as a level. The last point is related to the semantic heterogeneity. This kind of heterogeneity is because of the difference related to the interpretation of real words concepts. There are many forms
related to the semantic heterogeneity such as the homonymous, synonymous, intensions, extensions, etc. A major problem must be solved is related to the extension of words when one word can be used in different context, so it becomes important to add more information to distinguish the different means. As example “Person” and “Student” can implies the same thing if they are defined in the same context.

To overcome the first problem, we specify a common structure interface to the different user, so we are sure that the different schemas have the same data model, and the common structure schema model serves also to solve the second problem since it defines their categories i.e. the user defines if he needs a specific term as a fact, dimension, level, etc. We still have the semantic heterogeneity. This latter will be solved next using different techniques such the schema matching, mapping, etc.

The outline of this work is as follow: in the second section we will present a study of the existing methodologies related to the schema integration. Next, we will give the structure of the schema that we will use in the rest of this work, to move after to define the notion of degree of similarity that is necessary to calculate the similarity of the elements of the schemas. In section 5, we present the different phases of our methodology that is used to merge the schemas. Our methodology starts with the schema comparison that uses the schema matching technique in the beginning to detect the semantic correspondences also the conflicts, then, it uses the schema mapping to solve the exiting conflicts. In the next section, we introduce the algorithms that correspond to different methodology phases and we will finish our work with the conclusion and perspective.

2. STATE OF THE ART

In this section we present the different methodologies used to ensure the schema integration.

There are two strategies behind the creation of a global schema using the schema integration which are “bottom-up” and “top- down”. The use of one of them depends on the existence or not of the global schema. So, in the first strategy, the global schema does not exist, and the integration process involves both the definition of a global schema, as well as the definition of the mappings between the data source schemas and the global schema [7]. In top-down integration setting the global schema exists, and mappings need to be defined between the data source schemas and this global schema [24]

The bottom up strategy is appropriate in the case of schema integration process while the top-down is more suited from the perspective of domain engineering [18]. So, and according to our goal we will adapt bottom up strategy to create our global schema. There are different ways to apply this strategy. In fact, it depends on how we will merge the local schemas i.e. using as input two schemas (binary) or all-at-once (n-ary). The binary can be divided into “ladder”(Figure.1) [6] and “balanced” (Figure.2) [21]. The n-ary is composed by “one-shot” (Figure.3) [32], [29], [5] and “iterative” (Figure.4) [33].

![Figure 1 binary ladder](image1.png) ![Figure 2 binary balanced](image2.png) ![Figure 3 n-ary one-shot](image3.png) ![Figure 4 n-ary iterative](image4.png)

In [7], the proposed methodology is composed by 4 phases: 1) pre-integration, 2) comparison of the schemas, 3) conformation of the schemas, and 4) merging and restructuring of the schemas.

- Pre-integration: the analysis of schemas intervenes to choose of schemas to be integrated, the order of integration, and a possible assignment of preferences to entire schemas or portions of schemas.
- Comparison of the schemas: the schemas are analyzed and compared in order to determine the correspondences among the concepts also to detect the conflicts.
- Conformation of the schemas: this step is applied once conflicts are detected, and it must be treated before merging the schemas. Making the resolution an automatic task is not a good idea; it needs human intervention (interaction between the user and the designer).
- Merging and restructuring of the schemas: the result of this step is a global schema resulting from superimposing the intermediate integrated schema(s).

In [39], the authors propose a methodology composed by 3 phases:

- Resolving conflicts among concepts in the local schema,
- Solving differences among data in existing databases,
- Modifying queries to make them consistent with the global schema. They categorize four schema differences (naming conflicts, scale conflicts, structural conflicts and differences in abstraction) and two data conflicts (mutually inconsistent local databases containing correct or incorrect information).
In [10], the proposed methodology is composed by 3 phases:
- Pre-integration: the schemas in the input are processed in different ways to make them more homogeneous.
- Searching for matches: this level serves to identify the similar elements in the first schemas, and then it details the inter-relationship diagrams.
- Integration: the final step unifies the corresponding types into one integrated schema and produces the translated rules associated between the integrated schema and the initial ones.

To ensure the comparison of the schemas, we propose a new schema integration methodology is proposed. This latter is composed by two phases which are: 1) schemas comparison, and 2) schema fusion.

3. THE SCHEMA STRUCTURE
Since a schema is complex in term of composition, comparing its whole structure at once is not an adequate solution, so we propose its decomposition into set of categories which are: fact, dimension, measure, attribute, parameter, and hierarchy.

- The Fact corresponds to the subject of analysis. It is defined by a tuple (FN, M{}) with FN represents the name of the fact and M{m_1, m_2, m_3, m_4, ...} corresponds to the set of measures related to the fact F.
- The Dimension represents the axis of analysis. It is composed by (DN, A{}, H{h_1, h_2, h_3, h_4, ...}) presents the set of attributes describing the current dimension D, and H{h_1^D, h_2^D, h_3^D, h_4^D, ...} is a set of ordered hierarchies. Each hierarchy has a tuple (HN, P{}) with HN is the name of the current hierarchy and P{p_1, p_2, p_3, p_4, ...} is a set of ordered parameters.

4. DEFINITION
4.1. Degree of similarity (DeSim)
When we calculate the similarity between the elements of the two schemas, we should take into consideration the following points:
- The identical: the case where we use the same elements name in the two schemas.
  \[ DeId(e_1, e_2) = \begin{cases} 1 & \text{if } e_1 \text{ and } e_2 \text{ are identical} \\ 0 & \text{else} \end{cases} \]
- The synonymous: it is the case where we use two different names that have the same meaning.
  \[ DeSy(e_1, e_2) = \begin{cases} 1 & \text{if } e_1 \text{ and } e_2 \text{ are synonymous} \\ 0 & \text{else} \end{cases} \]
- The typos: it is the case where the user makes mistakes when writing the name of the element.
  In this case, we calculate the degree of error. If it is low, we are in the case of typing error. If it is high we are in the case of two different words. In the following we only take into consideration the first case.
  \[ DeTy(e_1, e_2) = \begin{cases} 1 & \text{if } e_1 \text{ and } e_2 \text{ are the same with the existence of typing error} \\ 0 & \text{else} \end{cases} \]
- The post-and pre- fixe: it is the case where we use post-fixes or pre-fixes to design the same thing.
  \[ DePost(e_1, e_2) = \begin{cases} 1 & \text{if one two the elements is the post-fixe of the other, and } 0 & \text{else} \end{cases} \]
  \[ DePre(e_1, e_2) = \begin{cases} 1 & \text{if one of the elements is the pre-fixe of the other, and } 0 & \text{else} \end{cases} \]

Let Sch1 and Sch2 be two schemas belonging to the same domain.
Let Ci be the categories of elements existing in the schema. Ci can be: fact, dimension, measure, attribute, parameter, and hierarchy: \( \forall e_i \in Sch_1, \exists e_j \in Sch_2 \text{ such that } e_i \text{ and } e_j \text{ belong to the same category } Ci. \)

The degree of similarity between \( e_i \) and \( e_j \) (DeSim \( (e_i, e_j) \)) measured by the numeric value in \([0, 1]\) as in (1):

\[
DeSim(e_i, e_j): Sch_1 \times Sch_2 \rightarrow [0, 1] \\
DeSim(e_1, e_2) = [DeId(e_1, e_2) + DeSy(e_1, e_2) + DeTy(e_1, e_2) + DePost(e_1, e_2) or DePre(e_1, e_2)] / 4
\]

4.2. Similarity
Two schemas are considered “Similar” if they have the highest degree of similarity.
Two similar schemas can be totally or partially merged.
- Total Merge (TM) implies the two schemas have the same elements.
  \[ TM(sch_1, sch_2) = \{ \forall x_1 \in sch_1, \exists x_2 \in sch_2 \}, x_1 = x_2 \]
- Partial Merge (PM) implies the two schemas have some elements in common.
  \[ PM(sch_1, sch_2) = \{ \exists x_1 \in sch_1, \exists x_2 \in sch_2 \}, x_1 = x_2 \]
5. SCHEMA INTEGRATION PHASE

In this section we will present our methodology that ensures the construction of the global schema by merging the set of local schemas belonging to the same domain.

The proposed methodology is composed by two steps: the first one consists in comparing the schemas to determine the elements that are semantically related. It serves also to detect the conflicts (if they exist). In the second step, we start by solving the conflicts, and then we merge the schemas.

5.1. Schema Comparison

In the schema comparison step, we propose the use of the schema matching technique to detect the semantic correspondence, as well as the conflicts that may exist in such case.

5.1.1. Schema Matching

The schema matching is considered as one of the basic operations required by the process of data integration [25]. It is used to solve the problem related to the heterogeneity of the data sources by finding semantic correspondence between the elements of the two schemas. This phase takes as input two or many schemas and for our case we will take two schemas to get as output set of mapping rules.

In the literature, it is considered as challenging task for the following reasons [17]:
- Different schemas presenting identical concepts can have different structure also different names
- They can contain similar but non-identical concepts
- They can be expressed using different models.
- They can use similar words to have different meanings.
- Etc.

And because it is tedious, time consuming, error-prone, and expensive process [41], it can fail to capture critical information [27] that will be used next to ensure the schema mapping and schema integration. The information can involve dozens of schemas including their elements. Thus its automation has received great attention [27], and many tools have been developed such as: Autoplex [15], Automatch [16], Clio [19], [30], COMA [14], Cupid [17], Delta [8], DIKE [20], EJX [13], GLUE [4], LSD [3], MOMIS (and ARTEMIS) [37], [34], SemInt [40], SKAT [28], Similarity Flooding (SF) [35], and TranScm [38].

To ensure the effective schema matching tool, we should take into consideration the combination of several techniques such as linguistic matching of names of schemas elements, the comparison of the instance of data having similar structure [new1]. In this level of our work, we need to focus on the first technique, and according to [41], it proceeds in three steps: normalization, categorization and comparison.

- Normalization: the difference of names can be because of the use of abbreviations, acronyms, punctuation, etc. They perform tokenization (i.e. parsing names into tokens based on punctuation, case, etc), expansion (identification of the abbreviation, acronyms, etc). So to take the previous steps into consideration we propose the use domain ontology, lenvenshtein name, etc.
- Categorization: the elements composing the schemas are clusters into categories. In our case we have the following categories: fact, dimension, measures, attributes, hierarchies, parameters. Each element of the schema belongs to a specific category.
- Comparison: a coefficient of linguistic similarity is calculated by comparing the tokens extracted from the names of the elements.

Clustering the elements into categories reduces the number of one-to-one comparison eliminating the unnecessary comparisons (for example: comparing a fact element with a dimension element).

At the end of this phase we will get a table containing set of coefficients calculating the similarity between the elements belonging to the same category.

To compare the schema, we propose the division of the categories of the schemas into two types: the first one includes fact, dimensions, measures and levels and the second type includes the hierarchy. The identification is done, then, object by object in function of its category (fact of sch1 against fact of sch2, dimension of sch1 against dimension of sch2, etc) except for the hierarchy where we have to take into consideration the relationships of the parameters of the hierarchies also their order.

5.1.2. Schema Matching Algorithm

This algorithm serves to extract the matching elements to facilitate their merging next. This is done by calculating the similarity degree between the elements as follow:

We start the comparison with the “fact” if the two facts of the two schemas are equivalent we move to the comparison of the measures. If they are equivalent, we are in the case where the two schemas deal with the same fact information. The resulting schema will be composed by one fact table and a set of measures of one of the two schemas. In the case of
where the measures are different, the fact table will contain the combination of all the existing measures. When the two facts are different, the resulting schema will contain the two fact tables.

Next, we move to compare the “dimensions”. We propose in this level the use of similarity matrix (Figure. 5). The columns contain the names of the dimensions of the first schema and the lines contain the name of the dimensions of the second schema. The cells contain the “DeSim” that corresponds to the degree of similarity between the elements of two schemas.

When two dimensions of two different schemas are equivalent, we compare the “attributes”, then the “parameters” of the hierarchies. If they are equivalent we keep each one of them, else we combine them.

In all of the previous comparison cases, we use similarity matrix as a way to find the closest elements, and for the hierarchies, we should take into consideration the order of the elements.

<table>
<thead>
<tr>
<th></th>
<th>Sch1.D 1</th>
<th>Sch1.D 2</th>
<th>Sch1.D 3</th>
<th>Sch1.D 4</th>
<th>Max(DeSim)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sch2.D 1</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>X = Max (a, b, c, d)</td>
</tr>
<tr>
<td>Sch2.D 2</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>Y = Max (e, f, j, h)</td>
</tr>
<tr>
<td>Sch2.D 3</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>Z = Max (i, j, k, l)</td>
</tr>
</tbody>
</table>

Figure 5 Example of similarity matrix comparing dimensions of two different schemas

5.1.3. The Conflicts Detection

The previous step helps to identify the elements that are semantically related but this is not sufficient to integrate the set of local schemas into global one. What we need now is extracting the schemas conflicts and dealing with them to satisfy the requirements. The author in [18] presents three types of conflicts that occur during the integration phase: extensional, structural and naming conflicts.

- Extensional conflict: it refers to the redundancies among different classes [36]. There are four types of extensional relationship. The authors in [42] present them and they give the solution for each type. Let $E_A$ and $E_B$ two elements extracted from the two schemas and belonging to the same category, and $K_{EA}$, $K_{EB}$
  - Equivalent sets: $E_A = E_B$ (there is no conflict). The two elements present the same instance e.g. “employee”, “worker”, “Staff_Member”.
  - Subset relationships: $E_A \subseteq E_B$: $E_A$ represents a subset of $E_B$ instances at all times. e.g. “Employee”, “Manager”. The solution: $K_{EA}$ inherits from $K_{EB}$
  - Overlapping sets $E_A \cap E_B \neq \emptyset$ and $E_A - E_B \neq \emptyset$ and $E_B - E_A \neq \emptyset$. A and B can share the same instances. e.g. “Employee” and “Client”. The solution: $E_A$ and $E_B$ inherit from the new class $K_{EA\cap EB}$
  - Disjoint sets $E_A \cap B = \emptyset$. e.g. $E_A$ and $E_B$ represent a set of instances but they do not share it at any time. e.g. “EmployeeMS” and “EmployeeDO”. The solution $E_A$ and $E_B$ inherit from the new class $K_{EA\cap EB}$

- Structural conflict: In the context of the schemas of databases, the structure conflict “occurs when related real world concepts are modeled using different constructs in the different schemas” [23].

The authors [23] extract from the literature the following types of structural conflicts that are specified to ER schema.

1. An entity type in one schema is modeled as an attribute of an entity type or a relationship set in another schema.
2. An entity type in one schema is modeled as a relationship set in another schema.
3. A relationship set in one schema is modeled as an attribute of an entity type or a relationship set in another schema.
4. An attribute of a relationship set is modeled as an attribute of an entity type.

In the context of our work, we do not need to focus on this kind of conflict since we will keep each element as it is in the global schema.

- Naming conflict: According to [9], it “refers to the relationship between the object attribute or instance names”. The relationship between the names is commutative i.e. term$_1$ is homonyms of term$_2$ implies also term$_2$ is homonyms of term$_1$.

In this part, we treat homonyms and synonyms. The homonyms occur if one name is used for two or more concepts [26], and the synonyms occur if two or more names are used for the same concept [26], it can exist in any category. It is solved using the generalization [9].
This conflict is determined using different tools such as wordnet, thesaurus, etc. Their specification depends on their context.

5.2. Schema Mapping

Once we detect the conflicts existing between the two schemas, we move to the next step that consists on resolving those conflicts using schema mapping technique. This latter is used to specify the relationships between two types of schemas: the source and the target.

In our case we have two sources (the inputs) “sch1” and “sch2”, and one target (the output) “T”.

\[ M = (sch1; sch2; T; \delta) \]

**Definition:** a schema mapping is a qua-triple \( M = (sch1; sch2; T; \delta) \) such that “sch1” is the first schema, “sch2” is the second schema, “T” is the target schema (the schema resulting from the merging of the two input schemas), and \( \delta \) is a set of formulas over \( <sch1, sch2; T> \).

- An instance of \( M \) is an instance of \( <s_1, s_2; t; \delta> \) over \( <sch1, sch2; T; \delta> \) that has a specific formula in the set \( \delta \).
- Let \( \text{Ins}<M> \) denotes the instances \( <s_1, s_2; t; \delta> \) of \( M \). Each instance has its own formula \( \delta_i \).

The formulas existing in \( \delta \) correspond to one of the following functions:

- **Union:** \( R = \text{union} (e_1, e_2) \) implies that \( R \) is the union of the two elements \( e_1 \) and \( e_2 \). This function can take as input more than two elements but since we propose the use of binary ladder, we need two elements as input. \( R \) contains all the components of \( e_1 \) and all components of \( e_2 \).

- **Intersection:** \( R = \text{intersection} (e_1, e_2) \) implies that \( R \) is the intersection of the two elements \( e_1 \) and \( e_2 \). \( R \) contains the components that exist in \( e_1 \) and \( e_2 \).

- **Disjoint:** \( \text{disjoint}(e_1, e_2) \) \( e_1 \) and \( e_2 \) are disjoint if they no component in common.

The schema mapping is generally done manually and it requires good domain knowledge. Even the applications that have been developed to facilitate this task, they visualize the sources and it is the role of the user to finish this task.

We use the mapping as an intermediate step for merging the schemas sources. In the following, we propose an algorithm that serves to map two schemas. Here the task of user consists on confirming the result or modifying it if it is necessary.

5.2.1. Mapping fact tables and measures

5.2.1.1. Case of two same fact tables

Let two fact tables “Sales” and “SalesFact” (Figure 6). The first fact has as measures “Store-Sales”, “Store-Cost” and “Unit-Cost”. Concerning the second fact, its measures are: “Quantity” and “UnitPrice”.

![Figure 6 Example of two similar fact tables](image)

- **Ins**<sub>1</sub><sub>&lt;</sub><sup>M</sup> : &lt; “Sales”, “SalesFact”; “Sales” or “SalesFact”; \( \delta_1 \) \> With \( \delta_1 \) =Intersection (Sales, SalesFact).
  
  For the Target “T”: two solutions are possible “Sales” or “SalesFact”.

- **Ins**<sub>2</sub><sub>&lt;</sub><sup>M</sup> : &lt; “Store-Sales”, “Quantity”; “Store-Sales” or “Quantity”; \( \delta_2 \) \> With \( \delta_2 \) =Intersection (Store-Sales, Quantity).
  
  For the Target “T” two solutions are possible “Store-Sales” or “Quantity”.

- **Ins**<sub>3</sub><sub>&lt;</sub><sup>M</sup> : &lt; “Unit-Cost”, “UnitPrice”, “Unit-Cost” or “UnitPrice”; \( \delta_3 \) \> With \( \delta_3 \) =Intersection (Unit-Cost, UnitPrice).
  
  For the Target “T” two solutions are possible “Unit-Cost” or “UnitPrice”.

- **Ins**<sub>4</sub><sub>&lt;</sub><sup>M</sup> : &lt; “Store-Cost”; \( \emptyset \); “Store-Cost”; \( \delta_4 \) \> With \( \delta_4 \) =Union (Store-Cost, \( \emptyset \)).
  
  For the Target “T” contain the only solution “Store-Cost”.

The possible targets (Figure 7):
Remark: the same steps are done in the case of using the same word as name of different facts.

5.2.1.2. The case of two different fact tables
Let two fact tables “Sale” and “Purchase” (Figure 8). The first one contains “QuantitySold” measure and the second has “QuantityPurchase” measure.

- Ins$_1$$\langle M \rangle$ : < “Sale”, “Purchase”; $\emptyset$; $\delta_1$> With $\delta_1$ = disjoint (Sales, Purchase).
  For the Target “T”, there is no solution since the two names are disjointed.
  Here there is no need to move to compare the measures.
  In such case, where the two tables are disjoint, we keep both of them with their measures even if there are some measures in common.

5.2.2. Mapping dimensions and attributes

5.2.2.1. The case of two same dimensions
Let two dimensions tables having the same name “Supplier” (Figure 9). The first table contains the following attributes “F-Name”, “L-Name”, “CompanyName” and “Phone”. The second one has “FirstName”, “LastName”, “HomePage” and “Fax”.

- Ins$_1$$\langle M \rangle$ : < “Supplier”, “Supplier”; “Supplier”; $\emptyset$; $\delta_1$> With $\delta_1$ = Intersection (Supplier, Supplier).
  For the Target “T”, there is only one solution “Supplier”.
- Ins$_2$$\langle M \rangle$ : < “F-Name”, “FirstName”; “F-Name” or “FirstName”; $\delta_2$> With $\delta_2$ = Intersection (F-Name, FirstName).
  For the Target “T”, two solutions are possible “F-Name” or “FirstName”.
- Ins$_3$$\langle M \rangle$ : < “L-Name”, “LastName”; “L-Name” or “LastName”; $\delta_3$> With $\delta_3$ = Intersection (L-Name, LastName).
  For the Target “T”, two solutions are possible “L-Name” or “LastName”.
- Ins$_4$$\langle M \rangle$ : < “CompanyName”, $\emptyset$; “CompanyName”; $\delta_4$> With $\delta_4$ = Union (CompanyName, ).
  For the Target “T”, there is only one solution “CompanyName”.
- Ins$_5$$\langle M \rangle$ : < “Phone”, $\emptyset$; “Sales”; $\delta_5$> With $\delta_5$ = Union (Phone, $\emptyset$).
  For the Target “T”, there is only one solution “Phone”.
- Ins$_6$$\langle M \rangle$ : < $\emptyset$, “HomePage”; “HomePage”; $\delta_6$> With $\delta_6$ = Union ($\emptyset$, HomePage).
  For the Target “T”, there is one solution “HomePage”.
- Ins$_7$$\langle M \rangle$ : < $\emptyset$, “Fax”; “Fax”; $\delta_7$> With $\delta_7$ = Intersection ($\emptyset$, Fax).
  For the Target “T”, there is one solution “Fax”.
Different results are possible. In Figure 10, we present some of them.

5.2.2.2. The case of different dimensions table
Let the two dimensions “Supplier” and “Customer” (Figure 11). The first table has as attributes “Name”, “HomePage”, “Fax”, “Company Name” and “Phone”. The second table contains “Name”, “Email” and “Phone”.

Figure 11 Example of two different dimension tables

- Ins$_1$ M : <“Supplier”, “Customer”; $\emptyset$; $\delta_1$> With $\delta_1$=disjoint (Supplier, Customer).
  For the Target “T”, there is no solution since the two names are disjointed.
Here there is no need to compare the attributes, because even they are equivalents they do not present the same information. In such case, where the two tables are disjoint, we keep both of them with their attributes.

5.2.3. Mapping hierarchies and parameters

5.2.3.1. Case of same dimensions with some common parameters
Let the two hierarchies (Figure 12). The first is composed by “Day”, “Month” and “Year” and the second contains “Minute”, “Hour” and “Day”.

Figure 12 Example of two hierarchies having one common parameter “Day”.

- Ins$_1$ M : <“Date”, “Date”; “Date”; $\delta_1$> With $\delta_1$=Intersection (Date, Date).
  For the Target “T”, there is only one solution “Date”.
- Ins$_2$ M : <“Day”, “Day”; “Day”; $\delta_2$> With $\delta_2$=Intersection (Day, Day).
  For the Target “T”, there is only one solution “Day”.
- Ins$_3$ M : <“Month”, $\emptyset$; “Month”; $\delta_3$> With $\delta_3$=Intersection (Month, $\emptyset$).
  For the Target “T”, there is only one solution “Month”.
- Ins$_4$ M : <“Year”, $\emptyset$; “Year”; $\delta_4$> With $\delta_4$=Intersection (Year, $\emptyset$).
  For the Target “T”, there is only one solution “Year”.
- Ins$_5$ M : <$\emptyset$, “Minute”; “Minute”; $\delta_5$> With $\delta_5$=Intersection (Minute, $\emptyset$).
  For the Target “T”, there is only one solution “Minute”.
- Ins$_6$ M : <$\emptyset$, “Hour”; “Hour”; $\delta_6$> With $\delta_6$=Intersection (Hour, $\emptyset$).
  For the Target “T”, there is only one solution “Hour”.

In this case, there is only one target (Figure 13).
5.2.3.2. The case of the same dimensions with the same parameters

Let the two following dimensions “Product” and “ProductDimension” have the same parameters “Subcategory” and “category” (Figure 14).

- Ins₁<M> : <“Product”, “ProductDimension”; “Product” or “ProductDimension”, δ₁> With δ₁ = Intersection (Product, ProductDimension).
  For the Target “T”, there are two possible solutions (Product, ProductDimension).
- Ins₂<М> : <“Subcategory”, “Subcategory”; “Subcategory”; δ₂> With δ₂ = Intersection (Subcategory, Subcategory).
  For the Target “T”, there is only one solution “Subcategory”.
- Ins₃<М> : <“Category”, “Category”; “Category”; δ₃> With δ₃ = Intersection (Category, Category).
  For the Target “T”, there is one solution “Category”.

The possible targets are (Figure 15):

5.2.3.3. The case of same dimensions with different parameters

Let the two hierarchies (Figure 16); they have the same dimension “Patient”. The first hierarchy contains “ConsultationDate”, “Hour”, “Day” and “Year”. The second one is composed by “Address”, “Area” and “Country”.

- Ins₁<M> : <“Patient”, “Patient”; “Patient”; δ₁> With δ₁ = Intersection (Patient, Patient).
  For the Target “T”, there is only one solution “Patient”.
- Ins₂<М> : <“ConsultationDate”; ∅; “ConsultationDate”; δ₂> With δ₂ = Union (ConsultationDate, ∅).
  For the Target “T”, there is only one solution “ConsultationDate”.
- Ins₃<М> : <“Hour”; ∅; “Hour”; δ₃> With δ₃ = Union (Hour, ∅).
  For the Target “T”, there is only one solution “Hour”.
  For the Target “T”, there is only one solution “Day”.
- Ins₅<М> : <“Year”; ∅; “Year”; δ₅> With δ₅ = Union (Year, ∅).
  For the Target “T”, there is only one solution “Year”.
- Ins₆<М> : <∅; “Address”; “Address”; δ₆> With δ₆ = Union (∅, Address).
  For the Target “T”, there is only one solution “Address”.
- Ins₇<М> : <∅; “Area”; “Area”; δ₇> With δ₇ = Union (∅, Area).
  For the Target “T”, there is only one solution “Area”.
- Ins₈<М> : <∅; “Country”; “Country”; δ₈> With δ₈ = Union (∅, Country).
  For the Target “T”, there is only one solution “Country”.

There is no intersection between the two hierarchies, so we keep them related to the same dimension as follow (Figure 17):
5.2.3.4. The case of two different dimensions.

Let take as the two hierarchies (Figure 18). In such case, we keep the two dimensions separately, since they have nothing in common. For example

![Figure 18 Example of two hierarchies with different dimension tables](image)

Remark: Here we need the intervention of the designer to specify exactly what he needs, especially with the importance of the order of the parameters.

6. THE ALGORITHMS

The principal algorithm takes as input two schemas to get as result one schema containing the fusion of the elements (Figure 19). It is divided into two parts, the first one extracts the matching elements and the second merges them using the technique of mapping.

![Figure 19 “Merging” algorithm](image)

The Figure 20 presents the “MatchingFact” algorithm. From each schema it extracts the facts, then, it compares them. The comparison function calculates the coefficient of the two facts, and then it stores them into listfact.

![Figure 20 “MatchingFact” algorithm](image)

The Figure 21 presents the “MatchingDimension” algorithm. It extracts the set of dimensions from the two schemas. Using the comparison function we calculate the coefficient of similarity of the different dimensions, and we keep into listdimension those having the maximum coefficient.
Algorithm MatchingDimension

Input: sch1, sch2
Output: listdimension

Begin
D1 = extractDimension(sch1)
D2 = extractDimension(sch2)
listdimension = comparison(D1, D2)

End

Figure 21 “MatchingDimension” algorithm

The Figure 22 presents the “MatchingMeasure” algorithm. It extracts the set of measures from the two schemas. Using the comparison function we calculate the coefficient of similarity of the different measures, and we keep into listmeasure those having the maximum coefficient.

Algorithm MatchingMeasure

Input: sch1, sch2
Output: listmeasure

Begin
M1 = extractMeasure(sch1)
M2 = extractMeasure(sch2)
Listmeasure = comparison(M1, M2)

End

Figure 22 “MatchingMeasure” algorithm

The Figure 23 presents the “MatchingAttribute” algorithm. We use a loop to go through the dimensions existing in two schemas. For two dimensions, we extract the set of attributes. Using the comparison function we calculate the coefficient of similarity of the different attributes, and we keep into Listattribute those having the maximum coefficient.

Algorithm MatchingAttribute

Input: listdimension
Output: listattribute

Begin
For each couple of dimension (Di, Dj)
A1 = extractAttribute(Di)
A2 = extractAttribute(Dj)
listattrib = comparison(A1, A2)
Listattribute = concat(listattrib, listattribute)
End For

End

Figure 23 “MatchingAttribute” algorithm

The Figure 24 presents the “MatchingParameter” algorithm. We use a loop to go through the dimensions existing in two schemas. For two dimensions, we extract the set of parameters. Using the comparison function we calculate the coefficient of similarity of the different parameters, and we keep into Listparameters those having the maximum coefficient.

Algorithm MatchingParameter

Input: listdimension
Output: listparameter

Begin
For each couple of dimension (Di, Dj)
P1 = extractParameter(Di)
P2 = extractParameter(Dj)
listParam = comparison(P1, P2)
Listparameter = concat(listParam, listparameter)
End For

End

Figure 24 “MatchingParameter” algorithm
The Figure 25 corresponds to “MappingFact”. It takes as input the list of facts extracted from the previous step, to get as output one merged fact. This algorithm requires the intervention of the user to specify the exactly action to perform.

<table>
<thead>
<tr>
<th>Algorithm MappingFact</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: listfact</td>
</tr>
<tr>
<td><strong>Output</strong>: MergedFact</td>
</tr>
<tr>
<td><strong>Begin</strong></td>
</tr>
<tr>
<td>If the two facts are the same or equivalent</td>
</tr>
<tr>
<td>The user selects one of the two tables through the intersection of tables</td>
</tr>
<tr>
<td>If the measures are the same or equivalent</td>
</tr>
<tr>
<td>The user selects the proposed measures</td>
</tr>
<tr>
<td>Else (if they are different)</td>
</tr>
<tr>
<td>The user units the measures</td>
</tr>
<tr>
<td>End if</td>
</tr>
<tr>
<td>Else (If the two tables are different)</td>
</tr>
<tr>
<td>The system keeps the two tables separated each one with its measures</td>
</tr>
<tr>
<td>End if</td>
</tr>
<tr>
<td><strong>End</strong></td>
</tr>
</tbody>
</table>

**Figure 25 “MappingFact” algorithm**

The Figure 26 corresponds to “MappingDimension”. It takes as input the list of dimensions extracted from the previous step, to get as output the merged dimensions with their attributes and parameters. This algorithm requires the intervention of the user to specify the exactly action to perform.

<table>
<thead>
<tr>
<th>Algorithm MappingDimension</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: listdimension</td>
</tr>
<tr>
<td><strong>Output</strong>: MergedDimensions</td>
</tr>
<tr>
<td><strong>Begin</strong></td>
</tr>
<tr>
<td>For each couple of dimensions</td>
</tr>
<tr>
<td>If the dimensions are the same or equivalent</td>
</tr>
<tr>
<td>The user selects one of the two tables</td>
</tr>
<tr>
<td>If the attributes are the same or equivalent</td>
</tr>
<tr>
<td>The user selects what he needs</td>
</tr>
<tr>
<td>Else (if the attributes are different)</td>
</tr>
<tr>
<td>The system units the attributes</td>
</tr>
<tr>
<td>End if</td>
</tr>
<tr>
<td>If the hierarchies are the same or equivalent</td>
</tr>
<tr>
<td>The user selects one of the two dimensions including its attributes and hierarchies</td>
</tr>
<tr>
<td>Else (if the hierarchies are different)</td>
</tr>
<tr>
<td>The system keeps the two hierarchies and links them to the same dimension</td>
</tr>
<tr>
<td>End if</td>
</tr>
<tr>
<td>Else (If the dimensions are different)</td>
</tr>
<tr>
<td>The system keeps the two tables separated. Each one with its attributes and hierarchies</td>
</tr>
<tr>
<td>End if</td>
</tr>
<tr>
<td><strong>End For</strong></td>
</tr>
<tr>
<td><strong>End</strong></td>
</tr>
</tbody>
</table>

**Figure 26 “MappingDimension” algorithm**

7. CONCLUSION

In this work we presented an automatic method allowing the generation of the schema of the data mart from OLAP Requirement schemas (ORSs). We used the schema integration technique to merge the ORSs that belong to the same domain. In order to facilitate this task, we decomposed each schema into categories (fact, dimension, measure, etc). The integration technique is divided into two parts: the first one is concerned with the comparison of the schemas to extract the semantic correspondence and to detect the conflicts using schema matching techniques. Then, in the second part using the schema mapping technique it resolves the conflicts to merge the schemas element by element.
As future work, we will validate our data mart schema through its confrontation to the data source to build next our data warehouse schema.

References


