Abstract – This paper proposes a bottom-up method that helps designers of decision support systems to construct data mart schemas based on an object database. Our method extracts the database schema from the repository of the object DBMS in order to work on the most recent version of the transactional system’s data model. It then applies a set of rules to identify facts, measures, dimensions and their attributes, all of which it organizes as star schemas. The obtained star schemas could be used to help decision makers expressing their analytical needs. In addition, being identified from objects of the transactional system, they carry information useful in the ETL definition phase.

I. INTRODUCTION

Data warehouses have become an essential component of modern decision support systems in large organizations. They offer efficient access to integrated data sources in order to support managers in their planning and decision-making.

Building a data warehouse (DW) is a very challenging project because, compared to other software, it is quite a young domain and does not yet offer well-established strategies and techniques for the development process [1].

The conceptual design of a DW takes into account the documentation related to the reconciled database like an E/R diagram [2][3][4], a relational database [5], a UML diagram [6][7], and XML schemas [8][9][10]. To our knowledge, almost all methods relying on an object data-source start from a UML class diagram. However, we underline that, in practice, either companies do not always have such class diagrams, or even when they do have one, it is usually obsolete because the modifications due to the evolution of the operational system are not regularly/strictly performed to detain an updated version.

To overcome this problem of absence/obsoleteness of these diagrams, we propose in this paper a method that relies on the recent version of the object data source. We extract this recent version from the object database (ODB) repository. Besides being automatic, our method has the merit of producing all analyzable subjects from the ODB.

The remainder of this paper is organized as follows. Section II overviews DW design approaches that start from UML-based models. Section III briefly presents the standard object model from the Object Database Management Group (ODMG) and which we use to define our approach. Section IV presents our design approach of star schemas from an ODB. Finally, the paper closes up with a summary of the presented work and its perspectives.

II. RELATED WORK

Data-driven (or bottom-up) DW design approaches enjoy a double advantage: First, they reduce the task of decision makers since they build potential analytical schemas from the source data model and, consequently, they guarantee that the enterprise’s information system can feed the user-selected schemas with pertinent data. In these approaches, user-requirements solicitation and analysis are voluntary neglected. In fact, Bill Inmon argued that requirements are the last thing to be considered in a decision support system development [20] since they are well understood after the DW is populated with data and query results are analyzed by the decision makers.

Note that, for complex data sources, a data-driven approach may be time consuming and complex. This limit can be overcome by adopting a mixed approach, cf. [16], [17]. However, when the data-driven approach is fully automatic, its complexity can be overlooked especially that it would be applied once to construct all potential DM schemas. Considering the advantages of data-driven approaches, we elected to propose a data mart design method within this category.

Within the literature, almost all data-driven approaches start either from E/R (cf. [3], [4]), UML, relational (cf. [5]) or XML models (cf. [8], [9], [10]). Due to space limitation, in this section, we restrict our review to works that start from UML-based models.

Prat et al., [6] propose a conceptual design phase that starts from a UML class diagram representing the decision makers’ initial requirements. This UML model is then enriched/transformed in order to facilitate subsequent mapping of the UML conceptual model into a logical multidimensional schema. To achieve this enrichment, the authors define four transformations applied on the UML conceptual model: 1) determination of identifying attributes necessary to determine attributes of the participating classes; 2) manual determination of attributes representing measures to distinguish them from attributes expressing qualitative values; 3) migration of 1-1 and N-1 associations into association attributes; and finally 4) transformation of generalizations into aggregations that represent hierarchies. The resulting class diagram is represented in an extended version of UML in order to distinguish the multidimensional concepts.

Also for UML-based models, Zribi et al. [7] propose the construction of Data Mart (DM) schemas from a UML class diagram in five semi-automatic steps: 1) identification of transaction entities representing facts; 2) construction of decisional UML-packages each containing a transaction entity and its associated classes; 3) graphical annotation of packages to identify multidimensional concepts resulting from the preceding stage; 4) validation of the annotation by
the decisional designer; and 5) automatic generation of a DM star schema from each annotated package.

To our knowledge, almost all data-driven approaches relying on an object data-source start from a UML class diagram. However, in practice, such diagram either does not exist within a company, or it is obsolete and does not reflect the modifications due to the evolution of the operational system. To overcome this problem, one has two strategies: 1) apply a reengineering tool to first construct the UML class diagram from the object data source and then apply any of the given approaches. The resulting DM schemas depend, in this case, on the faithfulness of the constructed class diagram to the object data source; or 2) work directly on the object data source as extracted from the Object Database (ODB) repository. This second strategy has the merit of producing DM schemas directly traceable to the source.

III. ODMG: THE OBJECT DATA STANDARD

To define a generic design method, we relied on the object model standardized by the Object Database Management Group (ODMG), a consortium of the leading ODB vendors. This section overviews the Object Model supported by ODMG according to the last release of the standard ODMG 3.0 [21].

The object model specifies the semantics that can be defined explicitly by an ODBMS. This semantics determines the characteristics of objects, how objects can be named and identified, and how objects can be related to one another. As shown in Figure 1, the ODMG standard is based on a common object model and uses several aspects of OMG’s Object Model. It supports types (interface) and classes (implementation), encapsulation, inheritance and polymorphism. However, a few database specific features have been added, among which we find the following (see Figure 1):

- A **Key** uniquely identifies an instance of a type. Simple and compound keys are also supported.
- **Object Identifiers (OID)** are unique within a storage domain. The value of an OID never changes over the lifetime of an object and is never re-used.

In terms of typing, as illustrated in Figure 2, a number of predefined Definition Collection Types are available to build complex objects. The Structured Literals Date, Time, Timestamp and Interval are also supported. The standard model also defines inheritance between types (i.e., subtyping): If $S$ is a subtype of a type $T$, then $S$ inherits all the operations and properties of $T$, and $S$ may define new specific operations and properties applicable to its instances.

![Fig. 1. ODMG standard Meta model (essential concepts).](image1.png)

![Fig. 2. The ODMG Full Set of Built-in Types.](image2.png)

In terms of typing, if illustrated in Figure 2, a number of predefined Definition Collection Types are available to build complex objects. The Structured Literals Date, Time, Timestamp and Interval are also supported. The standard model also defines inheritance between types (i.e., subtyping): If $S$ is a subtype of a type $T$, then $S$ inherits all the operations and properties of $T$, and $S$ may define new specific operations and properties applicable to its instances.

In this example, an **Estimate** is prepared for a **Client** and contains all the **Estimate details**. Each **Estimate detail** is related to one model, which needs a **Study** to define associated **Material** and **Tasks** that may be either **automatic** or **manual**. We will use this example to illustrate our star schema design rules.

IV. DM SCHEMA IDENTIFICATION RULES

Our data-driven design method generates DM schemas from an ODB. It is domain independent because it relies on the structural properties of the data source independently of their semantics. It automatically applies a set of rules to extract from the ODB multidimensional concepts: candidate facts with their measures and dimensions with their attributes organized in hierarchies. Moreover, it keeps track of the origin (object name, attribute name, data type and length...) of each multidimensional concept in the generated DM schemas. This traceability is fundamental as we intend to help automating the generation of ETL (Extract Transform and Load) procedures to load the DM designed using the generated schemas.
In our extraction rules, we adopt the following notation:
- Given a relationship \( R \) between two objects \( O_1 \) and \( O_2 \) with multiplicity \((m_1, M_1)\) on the side of \( O_1 \) and \((m_2, M_2)\) on the side of \( O_2 \), \( \text{Max}(O_1, O_2) \) returns the two maximum multiplicities \((M_1, M_2)\).

- For an object \( O \), \( \text{TClosure}(O) \) denotes the set containing \( O \) and all objects \( O' \) directly or transitively linked to \( O \) by relationships with \( \text{Max}(O, O') = (1,1) \). Note that, if an object \( O' \) belongs to \( \text{TClosure}(O) \) then, \( \text{TClosure}(O') = \text{TClosure}(O) \); this optimizes the computation of \( \text{TClosures} \). In our running example (Figure 3), \( \text{TClosure}(\text{Model}) = \{\text{Model}, \text{Study}\} \) and thus \( \text{TClosure}(\text{Model}) = \text{TClosure}(\text{Study}) \).

In our work, we consider that the set \( \text{TClosure}(O) \) can be semantically seen as a single (real-world) object; indeed, all of the objects in this set are linked by strong functional dependencies.

Based on the above notation, we define in the remainder of this section our identification rules for multidimensional concepts from an object database. The identification starts with facts and then follows with measures, dimensions and hierarchies.

A. Fact identification

The fact concept records observations describing information to be analyzed. A fact may be extracted from an ODB from either relationships (rule F1) or objects (rules F2-F2.2):

**F1.** Each relationship among two objects \( O_1 \) and \( O_2 \) such that \( \text{Max}(O_1, O_2) = (N, M) \) where \( N>1 \) and \( M>1 \) is identified as a fact. We name such a fact \( F-\text{O1-path_name}-O2 \) where \( \text{path_name} \) is the name of the relationship transversal path (cf., Figure 1).

Note that relationships between objects do not contain attributes in ODB. Thus, each fact generated by rule F1 is an empty fact, i.e., it has no measures and serves as a record of event (the fact) occurrences [22].

Applying F1 on our running example (Figure 3), produces the two empty facts \( \text{AutomaticTask-executed-by-Machine} \) and \( \text{ManualTask-carriedout-by-personnel} \).

In addition to relationships, objects can also represent facts. To identify facts from objects, we first need to compute their transitive closures since a set of objects with a strong functional dependency may be semantically seen as a single object.

**F2.** Each \( \text{TClosure}(O) \) containing an object with a non key numeric attribute or with a method returning numeric value(s) is identified as a fact. The name of such a fact is the concatenation of \( F- \) with the names of the objects in \( \text{TClosure}(O) \).

In our rules, we use the term numeric attribute to cover attributes of ODB that are atomic, a collection and in a structure (Figure 2).

For our running example, rule F2 identifies from \( \text{TClosure}(\text{Model}) \) the fact \( F-\text{Study-Model} \); the object Model contains the non key numeric attribute \( \text{Price} \).

Note that, for each object \( O \), if \( \text{TClosure}(O) = \{O\} \) and \( O \) has a non key numeric attribute or a method returning numeric value(s), then according to F2, the fact \( F-O \) will be identified. In our running example, we have six such facts: \( F-\text{F-estimate_detail} \), \( F-\text{F-estimate} \), \( F-\text{F-Automatic} \), \( F-\text{F-Manual} \), \( F-\text{F-Machine} \) and \( F-\text{F-Personnel} \).

In addition, each object of a \( \text{TClosure}(O) \) that is identified as a fact will be useful to identify measures and dimensions associated to \( F \). Thus, considering the set of functionally dependent objects (as opposed to a single object) provides for building facts that cover a large variety of analyses.

Note that if an object does not have a key attribute, then the designer can intervene to select one. Such an optional intervention improves the results of our identification method (cf. section IV.B).

Using the rule F2, facts can be identified from both a specialized and a generalized object. In this section, we examine if these facts are better grouped into a single fact. To do so, we examine the instances of these objects in the ODB according the next two rules:

**F2.1:** If a specialized object \( S \) is identified to build a fact \( F-S \) (through F2), then replace \( F-S \) with a fact constructed from \( \text{TClosure}(S) \) together with all of its generalizing objects. The name of the replacing fact is \( F-S \) followed by the concatenation of all of its generalizing objects.

The main intuition behind F2.1 is that a specialized object can not exist without its generalizing objects. That is, there is a strong functional dependency between a specialized object and its generalization. Hence, by grouping the specialized object with its generalizing objects, we construct a fact with richer analyses potential.

**F2.2:** If a generalized object \( G \) is identified (through F2) to build a fact, and \( G \) has no instance of specialized objects in the ODB, then maintain \( \text{TClosure}(G) \) as a fact independently of its specialized objects.

The main intuition behind F2.2 is that such an object \( G \) can exist independently and thus its fact should be maintained.

The application of F2.1 and F2.2 on objects \( S \) and \( G \) (with \( S \) being a specialization of \( G \)) can identify simultaneously \( F-G \) and \( F-G-S \) as facts (with their transitive closures). These facts are conceptually included \( F-G \subseteq F-G-S \). However, at the logical level, \( F-G \) is
loaded with instances of \( G \) that are not attached to any instance of \( S \) and \( F-G-S \) with those linked to \( S \).

For our example, \( F2.1 \) replaces \( F\text{-Automatic} \) and \( F\text{-Manual} \) by, respectively, \( F\text{-Automatic\text{-Task}} \) and \( F\text{-Manual\text{-Task}} \) that include the generalized object \( \text{Task} \).

### B. Measure identification

A fact contains a finite set of measures. In most cases, measures serve to compute summarized results (e.g., total amount of sales by month and year, by mart...) using aggregate functions; hence, measures have numeric values. Therefore, we extract measures from objects identified as fact or from those belonging to a transitive closure identified as fact.

**M1.** Operations that return numeric values in an object that is either identified as a fact \( F \), or that belongs to a transitive closure identified as a fact \( F \) are candidate measures for \( F \).

An operation is a service that can be requested from an object. When the requested service returns (even through a parameter) a numeric value, then this value is important. For this, we consider it as a measure.

**M2.** Non key numerical attributes in an object that is either identified as a fact \( F \) or that belongs to a transitive closure identified as fact \( F \) are candidate measures for \( F \).

This rule excludes key-attributes from the set of candidate measures because keys are generally artificial and redundant data; moreover, keys do not trace/record the enterprise business activity [5].

**M3.** Numerical attributes in a structure attribute (Structure<<>) within an object that is either identified as a fact \( F \) or that belongs to a transitive closure identified as fact \( F \) are candidate measures for \( F \).

Note that **M3** considers only atomic structures and ignores collection of structures in measure identification. In fact, an atomic structure \( \text{struct} \) in an object \( O \) can be simulated a relationship between \( O \) and \( \text{struct} \) where \( \text{Max}(O, \text{struct}) = (1, 1) \).

Furthermore, as shown in the ODB meta-model of Figure 2, a numeric attribute can be atomic or collection (Set, Bag, List, Array ...). A collection is therefore multi-valued. However, in a data warehouse, measures cannot be multi-valued. Thus, for numeric collection attributes identified as measures, a numeric function that associates to the numeric collection an atomic value must be added by the designer. Such a function is evidently semantic/domain dependent.

In our running example, the measure identification rules produce the measures shown in Table I.

### C. Dimension identification

A dimension presents information according to which measures are recorded and analyzed, i.e., it is an analysis axis. A dimension is generally made up of a finite set of attributes. Some of these attributes take part to define various levels of details (hierarchies), whereas others are less significant but used, for instance, to label results or to restrict data processed in queries. These latter are called weak (or non dimensional) attributes. The set of candidate dimensions for a given fact can be built either on object (\( D1-D2 \)) or attributes (\( D3-D5 \)).

<table>
<thead>
<tr>
<th>Fact</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-Study-Model</td>
<td>Price</td>
</tr>
<tr>
<td>F-Estimate-detail</td>
<td>Total_detail()</td>
</tr>
<tr>
<td>F-Estimate</td>
<td>Total_Price()</td>
</tr>
<tr>
<td>F-Automatic-Task</td>
<td>AO_cost()</td>
</tr>
<tr>
<td>F-Manual-Task</td>
<td>MO_cost()</td>
</tr>
<tr>
<td>F-Machine</td>
<td>Hour_cost</td>
</tr>
<tr>
<td>F-Personnel</td>
<td>Wage</td>
</tr>
</tbody>
</table>

**D1:** Every \( TClosure(O) \) of an object \( O \) that participates in the formation of a fact \( F \) resulting from the rule \( F1 \) is identified as a dimension for \( F \). Conventionally, the name of this dimension is \( D1 \) followed by the concatenation of the of objects in \( TClosure(O) \).

**D2:** Let \( O1 \) be an object directly linked to an object \( O2 \) with \( \text{Max}(O1, O2) = (1,*)) \) and \( O2 \) is either identified as a fact \( F \), or it belongs to a transitive closure identified as a fact \( F \). Then, \( TClosure(O1) \) is a candidate dimension for \( F \). Conventionally, the name of this dimension is the concatenation of \( D1 \) with the objects in \( TClosure(O1) \).

This rule identifies as dimensions the objects maximally linked by (\( 1,*)) \) because, at the logical level of a DW, one instance of the fact is linked to its dimension instance by a maximally \((*,1)) \) relationship. In addition, by considering \( TClosure(O1) \) as the dimension (as opposed to \( O1 \)), we will be able to identify all dimensional attributes.

Besides dimensions built on objects, we can build a dimension both on an attribute of a particular data type as well as on some collection attributes. Such a dimension is known in data warehousing as a degenerated dimension [11].

**D3:** Every attribute with a structure collection type in an object that is either identified as a fact \( F \) or that belongs to a transitive closure identified as fact \( F \) is a dimension for \( F \).

A structure attribute \( S \) can be simulated as an object composed of a set of attributes. Thus, a collection of \( S \) (Set<\( S \), List <\( S \), ... ) in an object \( O \) can be simulated as a link between the objects \( O \) and \( S \). In addition, in ODB, it is impossible to have two objects share the same instance of an attribute. Thus, for a collection of \( S \) in an object \( O \), we have the guarantee that the link between \( S \) and \( O \) has \( \text{Max}(S,O) = (1, *) \). Thus, we identify the collection of \( S \) as a dimension for \( F \).

**D4:** An atomic Boolean attribute in an object that is either identified as a fact \( F \) or that belongs to a transitive closure identified as a fact \( F \) is a dimension for \( F \).
Naturally, a Boolean attribute splits data of its object into two subsets; thus, such an attribute can be an analysis axis.

Furthermore, the data warehouse community assumes a DW as a chronological collection of data [11]. Consequently, the time dimension appears in all data warehouses.

D5: A temporal attribute (Date, Time, Timestamp, Interval) in an object that is either identified as a fact F or that belongs to a transitive closure identified as fact F is a dimension for F.

Each dimension has an identifier that can be extracted through the following four rules:

ID1: A key attribute in an object D identified as a dimension is the identifier of D.

ID2: The identifier of a dimension issued from a transitive closure of an object TClosure(O) is a key attribute belonging to one of the objects in TClosure(O); the others are taken as weak attributes of the selected identifier.

Note that if none of the objects being considered in the rules ID1 and ID2 has a key attribute, then the identifier is the OID of any of these objects (given by the ODBMS).

ID3: The identifier of a dimension issued from an atomic attribute A is A.

ID4: The identifier of a dimension issued from a structure attribute A is a sequential artificial identifier named id_A.

For each identified dimension of our running example, Table 2 shows its determined identifier.

D. Hierarchy identification

Hierarchies are made up of discrete dimension attributes. They determine how facts may be aggregated and selected significantly for the decision-making process. The dimension identifier in which a hierarchy is rooted defines its finest aggregation granularity; the other dimension attributes define progressively coarser granularities. Hierarchies may include non-dimension attributes. A non-dimension attribute contains additional information about a dimension attribute of the hierarchy; unlike dimension attributes, it cannot be used for aggregation [22].

Once dimension identifiers are already identified, we continue the construction of hierarchies by identifying their remaining parameters; afterwards, we identify for each parameter its descriptive attributes, if any.

For parameters, we first extract the parameters located immediately after the dimension identifier; secondly, for each of these parameters, we extract its successor parameters using the following four rules.

P1: Let O1 be an object directly related to an object O2 such that Max(O1, O2) = (I, +) and O1 is either identified as a dimension D or it belongs to a transitive closure identified as a dimension D. Then, the key attribute of every object in TClosure(O1) is a parameter of level two for D.

Similar to the dimension identifiers, if an object does not contain a key attribute, then the OID given by the ODBMS is selected as a default parameter of level two.

P2 Every attribute with a structure collection type in an object either identified as a dimension D or belonging to a transitive closure identified as a dimension D is a parameter of level two for D.

P3 Every atomic Boolean attribute in an object identified as a dimension D or belonging to a transitive closure identified as a dimension D is a parameter of level two for D.

P4 A temporal structured attribute (Date, Time, Timestamp, and Interval) in an object identified as a dimension D or belonging to a transitive closure identified as a dimension D is a parameter of level two for D.

Parameters at a level higher than two are identified by the recursive application of: 1) the above four rules on objects from which a parameter is extracted by P1; and 2) P2, P3 and P4 on attributes identified by P2 as parameters. We exclude P1 because an attribute cannot be linked to an object by a relationship.

For a temporal attribute (Date, Interval, Time, and Timestamp by interfaces) resulting from the rule P4, we have studied the factory of ODMG operations for creating such objects and, we have defined for each temporal attribute its corresponding hierarchy. For example, for the representation of a calendar date consisting of a year (unsigned short year), month (unsigned short month) and day (unsigned short day), we define the hierarchy (day, month, year).

Table 3 lists, for each identified dimension of the example, the corresponding identified hierarchies.

A parameter may functionally determine other attributes within its object; these attributes describe the parameter and, therefore, are called descriptive. More precisely, descriptive attributes for a parameter p are non-key textual or numerical attributes in the object supplying p. In addition, as mentioned in the rule ID2, after selecting the identifier of a dimension, the primary key attribute of each object remaining in the transitive closure identified as a dimension is also a descriptive attribute.
TABLE III
PARAMETERS FOR THE DIMENSIONS OF TABLE II.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Hierarchy parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>Model, Material</td>
</tr>
<tr>
<td></td>
<td>Model, Article</td>
</tr>
<tr>
<td></td>
<td>Study, Material</td>
</tr>
<tr>
<td></td>
<td>Study, Article</td>
</tr>
<tr>
<td></td>
<td>A_duration</td>
</tr>
<tr>
<td>Machine</td>
<td>Purchase_date</td>
</tr>
<tr>
<td></td>
<td>State</td>
</tr>
<tr>
<td>Manual</td>
<td>Model, Material</td>
</tr>
<tr>
<td></td>
<td>Model, Article</td>
</tr>
<tr>
<td></td>
<td>Study, Material</td>
</tr>
<tr>
<td></td>
<td>Study, Article</td>
</tr>
<tr>
<td></td>
<td>M_duration</td>
</tr>
<tr>
<td>Personnel</td>
<td>Affectation_date</td>
</tr>
<tr>
<td></td>
<td>qualification, Date</td>
</tr>
<tr>
<td>Article</td>
<td>P_id</td>
</tr>
<tr>
<td>Material</td>
<td>M_id</td>
</tr>
<tr>
<td>Client</td>
<td>Address</td>
</tr>
<tr>
<td>Estimate</td>
<td>Client, Address</td>
</tr>
<tr>
<td>Model-Study</td>
<td>Article, Material</td>
</tr>
<tr>
<td></td>
<td>Model, Material</td>
</tr>
<tr>
<td></td>
<td>Study, Material</td>
</tr>
<tr>
<td></td>
<td>Study, Article</td>
</tr>
<tr>
<td></td>
<td>A_duration</td>
</tr>
<tr>
<td></td>
<td>Date, Day, Month, Year</td>
</tr>
</tbody>
</table>

Discussion

Note that while most of the above rules are highly inspired from the transformation rules for relational databases, the rules F2, F2.1 and F2.2 are specific to ODB. In fact, F2 treats methods returning numeric value(s), and F2.1 and F2.2 exploit generalization/specialization concepts. Also, rules D3 and P2 deals with the extraction of dimensions and parameters from a collection of structure attributes.

IV. CONCLUSION

This paper presented a set of rules that can automatically identify all facts, dimensions, dimension attributes and hierarchies from a given ODB. The produced schemas can be a starting point in the definition of the OLAP requirements. In addition to being automatic, the set of rules produces star schemas whose elements are traceable to the source objects. Such traceability facilitates the generation of ETL procedures. We are in the process of defining an approach to assist DM designers in this task. In addition, we are implementing our rules within a CASE toolset.

REFERENCES