Information Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

A requirement-driven approach to the design and evolution of data warehouses

Petar Jovanovic [a,*], Oscar Romero [a], Alkis Simitsis [b], Alberto Abelló [a], Daria Mayorova [a]

[a] Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain
[b] HP Labs, Palo Alto, CA, USA

ARTICLE INFO

Keywords:
DW
ETL
Multidimensional design
Conceptual design

ABSTRACT

Designing data warehouse (DW) systems in highly dynamic enterprise environments is not an easy task. At each moment, the multidimensional (MD) schema needs to satisfy the set of information requirements posed by the business users. At the same time, the diversity and heterogeneity of the data sources need to be considered in order to properly retrieve needed data. Frequent arrival of new business needs requires that the system is adaptable to changes. To cope with such an inevitable complexity (both at the beginning of the design process and when potential evolution events occur), in this paper we present a semi-automatic method called *ORE*, for creating DW designs in an iterative fashion based on a given set of information requirements. Requirements are first considered separately. For each requirement, *ORE* expects the set of possible MD interpretations of the source data needed for that requirement (in a form similar to an MD schema). Incrementally, *ORE* builds the unified MD schema that satisfies the entire set of requirements and meet some predefined quality objectives. We have implemented *ORE* and performed a number of experiments to study our approach. We have also conducted a limited-scale case study to investigate its usefulness to designers.

1. Introduction

Data warehousing ecosystems have been widely recognized to successfully support strategic decision making in complex business environments. One of their most important goals is to capture the relevant organization's data provided through different sources and in various formats with the purpose of enabling analytical processing of such data. The most common design approach suggests building a centralized decision support repository (like a DW) that gathers the organization's data and which, due to its analytical nature, follows a multidimensional (MD) design. The MD design is distinguished by the fact/dimension dichotomy, where facts represent that the subjects of analysis and dimensions show different perspectives from which the subjects can be analyzed (e.g., we can analyze *shopping orders* for *customers* and/or *suppliers*). Furthermore, the design of the extract-transform-load (ETL) processes responsible for managing the data flow from the sources towards the DW constructs, must also be considered.

Complex business plans and dynamic, evolving enterprise environments often result in a continuous flow of new information requirements that may further require new analytical perspectives or new data to be analyzed. Due to the dynamic nature of the DW ecosystem, building the complete DW design at once is not practical. Also, assuming that all information and business requirements are available from the beginning and remain intact is not realistic either. At the same time, for constructing a DW design (i.e., its MD schema) the heterogeneity and relations among existing data sources need to be considered as well.

The complexity of the monolithic approach for building a DW satisfying all information requirements has also been

* Corresponding author.
E-mail addresses: petar@essi.upc.edu (P. Jovanovic), oromero@essi.upc.edu (O. Romero), alkis@hp.com (A. Simitsis), aabello@essi.upc.edu (A. Abelló), mayorova@essi.upc.edu (D. Mayorova).

0306-4379/$ - see front matter © 2014 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2014.01.004

Table 1
The DW Bus Architecture for IR1–R5.

	Customer	Supplier	Nation	Region	Orders	Part	Partsupp	Lineitem_dim
ship. qty.(IR1)	√	√	√		√		√	
profit(IR2)		√	√					√
revenue(IR3)		√	√	√		√	√	
avil. stock val.(IR4)		√	√			√		
ship. prior.(IR5)	√				√			√

largely characterized in the literature as a stumbling stone in DW projects (e.g., see [1]). As a solution to this problem, a step-by-step approach for building a DW has been proposed in [1] (a.k.a. *Data Warehouse Bus Architecture*). This approach starts from data marts (DM) defined for individual business processes and continues exploring the common dimensional structures, which these DMs may possibly share. To facilitate this process, a matrix as the one shown in Table 1 is used, which relates DMs (i.e., their subsumed business requirements) to facts and dimensions implied by each DM. Such matrix is used for detecting how dimensions (in columns) are shared among facts of different DMs (in rows), i.e., if a fact of a DM in the row *x* is analyzed from a dimension of the column *y*, there is a tick in the intersection of *x* and *y*. The content of the matrix in Table 1 follows our running example based on the TPC-H benchmark [2], which is introduced in more detail in Section 2.1. We consider five information requirements (i.e., IR1–IR5) and for each of them a single DM. Each requirement analyzes some factual data (e.g., `IR3` analyzes the `revenue`), from different perspectives (e.g., `revenue` is analyzed in terms of parts – i.e., `partsupplier-part` hierarchy–, and supplier's region –i.e., `supplier-nation-region` hierarchy–). Finally, based on this matrix, different DMs are combined into an MD schema of a DW. However, such design guidelines still assume a tremendous manual effort from the DW architect and hence, DW experts still encounter the burdensome and time-lasting problem of translating the end-user's information requirements into the appropriate MD schema design.

Automating such process has several benefits. On the one hand, it supports the complex and time-consuming task of designing the DW schema. On the other hand, automatically produced results guarantee that the MD integrity constraints [3] are met as well as some DW quality objectives used to guide the process [4]. Accordingly, several works tried to automate the process of generating MD schemas (e.g., [5–7]). However, for the sake of automation, these approaches tend to overlook the importance of information requirements and focus mostly on the underlying data sources. Such practices require an additional manual work in conforming the automatically produced MD designs with the actual user requirements, which often does not scale well for complex scenarios. For example, it has been shown that even for smaller data source sizes the number of potential stars produced by means of a blind search of MD patterns over the sources is huge [7]. Consequently, it is not feasible to assume that the DW architect will be able to prune and filter such results manually.

To the best of our knowledge only three works went further and considered integrating the information requirements in their semi-automatic approaches for generating MD models [8–10]. At different levels of detail, these approaches support transforming every single requirement into an MD model to answer such requirement. However, how to integrate such individual MD models into a single, compact MD view is left to be done manually (although [8,9] introduce strict manual guidelines in their approaches to assist the designer). Our experiments, described in Section 6, have shown that integrating MD requirements is not an easy task and this process must also be supported by semi-automatic tools.

In this work, we present a semi-automatic method for Ontology-based data warehouse REquirement-driven evolution and integration (*ORE*). *ORE* complements the existing methods (e.g., [5–7]) and assists on semi-automatically integrating partial MD schemas (each representing a requirement or a set of requirements) into a unified MD schema design. Moreover, *ORE* could also be used to integrate existing MD schemas of any kind (e.g., as in [11]). *ORE* starts from a set of MD interpretations (MDIs) of individual requirements (which resemble the rows of Table 1). Intuitively, an MDI is an MD characterization of the sources that satisfies the requirement at hand (see Section 2.2 for further details). Iteratively, *ORE* integrates MDIs into a single MD schema which satisfies all requirements so far. Importantly, *ORE* generates MD-compliant results (i.e., fulfilling the MD integrity constraints) and determines the best integration options according to a set of quality objectives, to be defined by the DW designer. To guarantee so, *ORE* systematically traces valuable metadata from each integration iteration. During this entire process, the role of the data sources is crucial and *ORE* requires a characterization of the data sources in terms of a domain ontology, from where to automatically explore relationships between concepts (e.g., synonyms, functional dependencies, taxonomies, etc.) by means of reasoning.

Our method, *ORE*, is useful for the early stages of a DW project, where we need to create an MD schema design from scratch, but it can also serve during the entire DW lifecycle to accommodate potential evolution events. As we discuss later on, in the presence of a new requirement, our method does not create an MD design from scratch, rather it can automatically absorb the new requirement and integrate it with the existing MD schema.

Contributions: The main contributions of our work are as follows.

- We present a semi-automatic approach, *ORE*, which, in an iterative fashion deals with the problem of designing a unified MD schema from a set of information requirements.
- We introduce novel algorithms for integrating MD schemata, each satisfying one or more requirements. Results produced are guaranteed to subsume all requirements so far, preserve the MD integrity constraints, and meet the user defined quality objectives.
- We introduce the traceability metadata structure to systematically record information about the current integration opportunities both for finding the best

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

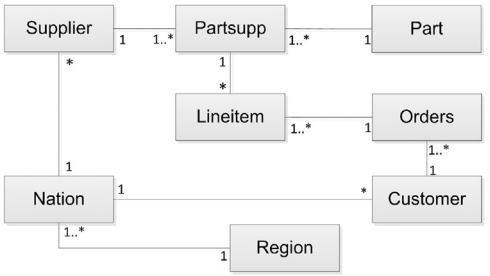

Fig. 1. TPC-H Schema.

integration solution w.r.t. the chosen quality objectives and linking the final MD schema to the data sources by means of ETL processes.

- We experimentally evaluate our approach by using a prototype. A set of empirical tests have been performed to assess the output correctness, a characterization of *ORE*'s internal algorithms and the manual effort for providing an integrated MD design from several requirements and in turn, demonstrate the need for automated design approaches like *ORE*.

Outline: The rest of the paper is structured as follows. Section 2 presents an abstract overview of our approach through an example case based on the TPC-H schema [2]. Sections 3 and 4 formally present the main structures and stages our method goes through for integrating new information requirements into a unified MD schema. In Section 5 we provide the theoretical validation of our approach, while Section 6 presents the experimental evaluation of *ORE* (both of its internals and from the end-users' perspective). Section 7 discusses the related work, while Section 8 concludes the paper and discusses our future directions.

2. Overview of our approach

In this section, we first introduce an example case based on the TPC-H schema and formalize the notation used throughout the paper. Then, we present the overview of our solution to semi-automatically create an MD schema from a set of information requirements.

2.1. Running example

Our example scenario is based on the schema provided by the TPC-H benchmark [2]. The abstraction of the schema is illustrated in Fig. 1. The TPC-H is a decision support benchmark which, besides the schema, also provides a set of business oriented queries (from here on, *information requirements* – IR). For the sake of our example, let us assume a subset of five information requirements, which are the following:

- IR1: The total *quantity* of the *parts* shipped from Spanish *suppliers* to French *customers*.
- IR2: For each *nation*, the *profit* for all supplied parts, *shipped* after 01/01/2011.
- IR3: The total *revenue* of the *parts* supplied from East Europe.
- IR4: For German *suppliers*, the total *available stock value* of supplied *parts*.
- IR5: *Shipping priority* and total potential *revenue* of the parts *ordered* before certain *date* and *shipped after certain date* to a *customer* of a given *segment*.

2.2. Formalizing information requirements

In this section, we describe how information requirements are formalized in order to be automatically processed. Requirements posed during the DW project lifecycle differ from usual user requirements in other software projects in that they typically have analytical flavor. For this reason, it has been previously discussed that information requirements can be elegantly represented in terms of the MD model as *data cubes* (e.g., see [8,12]), and they are usually elicited with natural language template similar to the following one: "I want to analyze `fact` (e.g., *revenue* or *profit*) from `dimensions` (e.g., *time*, *customer*) where `descriptors` (e.g., *previous year*)". This is a straightforward enunciation of the cube-query metaphor [1] in natural language, which distinguishes two basic kinds of concepts, namely *dimensional* and *factual* concepts.

Thus, from an information requirement we can extract two kinds of information: (1) which data the user wants to analyze (i.e., factual data) and (2) from which perspectives (i.e., the multidimensional space conformed by the dimensions of analysis), which may contain descriptors (i.e., dimension attributes). We further formalize the data cube in terms of a grammar, which can be conveniently used to serialize the content such as textual requirements or MD interpretations in a machine readable form (e.g., XML, JSON, etc.).

In a domain that is being analyzed, we can identify the concepts that retrieve the data asked by a requirement, i.e., `domain_concept`.

`Fact` (`F`) is the core MD concept in the domain and represents the focus of the analysis. It usually contains a set of numerical measures (`M`), e.g., *revenue*, *quantity*, *extended price*, which can be analyzed and aggregated from different perspectives (dimensions), using the appropriate aggregation functions (`AggFunc`), e.g., `SUM`, `AVERAGE`, `MAX`. Additionally, numerical measures can be derived by applying arithmetic operations (`ArithmOp`) or built-in functions (`DFunc`) over basic measures. Formally:

```
DFunc := 'sqrt'|'abs'|...;
AggFunc := 'SUM'|'AVG'|'COUNT'|'MEDIAN'|...;
ArithmOp := '+'|'-'|'/'|'x'|...;
Expr := Expr,ArithmOp,M|M;
M := DFunc(M)|Expr|<domain_concept>;
F :=AggFunc(F)|M;
```

The multidimensional space (`S`) is determined by the analysis dimensions (i.e., $S = D_1 \times \cdots \times D_n$). Each `dimension` (`D`) represents a perspective from which we analyze facts. In general, a single dimension consists of a hierarchy of partially ordered set ($\prec$) of `levels` (`L`). In a data cube, dimensions appear at a given level of detail (a.k.a. dimension level). Note that a meaningful cube can only refer to a single level per dimension. A function (`EFunc`) can be applied on levels to extract additional or normalized information – e.g., *month ('2001-09-28 01:00:00')* would return *'09'*. Additionally,

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

descriptors (Dsc) may be defined over the dimension levels and used inside of logic predicates, -e.g., *'year=2012'*-. Note that the literal in the following grammar rules stands for any numerical or textual constant value. Formally:

```
Oper := '>'|'<'|'>='|'<='|'='|'!='|'IN'|
    'MATCHES'|...;
EFunc := 'day'|'month'|...;
Dsc := Operand, Oper, Operand;
Operand := L | <literal>;
L := <domain_concept>|EFunc(L)|Dsc;
D :=D ≺ L |L
S := S × D |D
```

Given the above formalizations, we represent an information requirement or data cube (IR) as a fact (F) and (at least) one dimension (usually a list of dimensions), conforming its multidimensional space. Formally:

$$IR := F, S;$$

Various approaches have tackled the issue of modeling the MD data cube for each requirement at hand. Basically, they identify the needed data sources' subset that retrieves the data to answer each IR (intuitively a *query*) and map it into an MD interpretation (i.e., assign MD roles to each concept in the subset) to guarantee its compliance with the MD model. Eventually, they conform MD schemas answering all the requirements. Each of these methods makes different assumptions and achieves different degrees of automation but, to our knowledge, only three of them largely automate the process of identifying, for each requirement, the MD cube that answers such requirement [8–10]. However, none of these provides automatic means to integrate several requirements into a single MD schema.

ORE starts from the MD knowledge extracted from each requirement (e.g., by means of any of the previously mentioned MD schema design approaches) and aims to incrementally derive a unified MD schema satisfying the entire set of requirements.

For the purpose of formalizing our approach, we define here the notion of MD interpretation, regardless of the approach used to obtain these MD interpretations.

An MD interpretation (MDI) represents a subset of the data sources' concepts placed in a valid MD space that answers the IR at hand. Thus, an *MDI* can be formally defined as a tagged graph, meeting IR and satisfying MD integrity constraints, such as

$$MDI = (V, E, role : V \rightarrow_{IR} \{L, F\}) \text{ such that } MDI \vDash MD_{ic}$$

where V is the set of nodes, corresponding to source concepts, and E, the set of edges representing the associations between these concepts at the sources. Note that for a single MDI, "$\rightarrow_{IR}$" is a complete function and thus each node (V) plays an MD role according to IR (i.e., it is either a level or a fact). Additionally, the arrangement of nodes (V) and edges (E) in an MDI must satisfy the MD integrity constraints (MD_{ic}). Consequently, *ORE* relies on the previous approaches and assumes MDIs that are *sound* (i.e., that satisfy the MD integrity constraints) and *complete* (i.e., that satisfy the requirement at hand). Following the work in [13], we define here the MD integrity constraints that need to hold in order to satisfy both the soundness and completeness of input MDIs.

- *Information requirements*: Information requirements (IR) are expected to have an analytical layout, i.e., having the subject of the analysis (*fact*) and the perspectives from which the subject is analyzed (*dimensions*). Such requirements resemble the natural language template provided earlier in this section.
- *The multidimensional space arrangement*: The dimensions of a requirement must arrange the MD space in which the factual data of a requirement is depicted, i.e., each instance of factual data is identified by a point in each of its analysis dimensions.
- *The base concept*: A minimal set of levels which functionally determine a fact must guarantee that given a point in each of the dimensions, such set of points determines one and only one instance of factual data.
- *Data summarization*: Data summarization must be correct which is ensured by applying necessary conditions for summarization, discussed in [3]: (1) Disjointness (the sets of objects to be aggregated must be disjoint); (2) completeness (the union of subsets must constitute the entire set); and (3) compatibility of the dimension, the type of measure being aggregated and the aggregation function.

Furthermore, an MDI contains two kinds of concepts. Those explicitly demanded in the input requirement and those implicitly needed to properly relate the explicitly demanded concepts in order to produce a single cube. Thus, the latter depend on the data sources implementation and do not appear in IR. From here on, we refer to them as *intermediate concepts*. It is worth noting that initially, intermediate concepts may not have a strict MD role associated with them. Therefore, considering the potential ambiguity of business requirements and the diversity of the underlying associations in the data sources, several MDIs may result from a single requirement (each of them, capturing different semantics) [13]. We write $\mathbb{MDI}_{IR}$ to refer to the set of all MDIs satisfying IR.

In [10], we describe our approach to generate and validate such set of MDIs per requirement (as the ones shown in Fig. 2). However, *ORE* aims to be robust enough to process any valid MD cube-like form (i.e., respecting the fact/dimension dichtonomy) generated manually or automatically from information requirements through any of the current approaches available in the literature.

Example. Each requirement, defined at the beginning of Section 2 (i.e., IR1–IR5), gives rise to a set of one or more MDIs (see Fig. 2). Levels are depicted with white, and facts with gray boxes. The attributes of levels or facts are placed inside the corresponding boxes. The empty levels or facts (i.e., the boxes without any attributes) represent intermediate concepts and in the case their MD knowledge is *ambivalent* (i.e., either playing a factual or dimensional role, the MD integrity constraints are preserved) they are depicted as both level and fact. Consider now IR3, in plain text, and the MDIs produced for that requirement. There, we distinguish between concepts explicitly demanded (i.e., revenue, parts, and region name) and

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

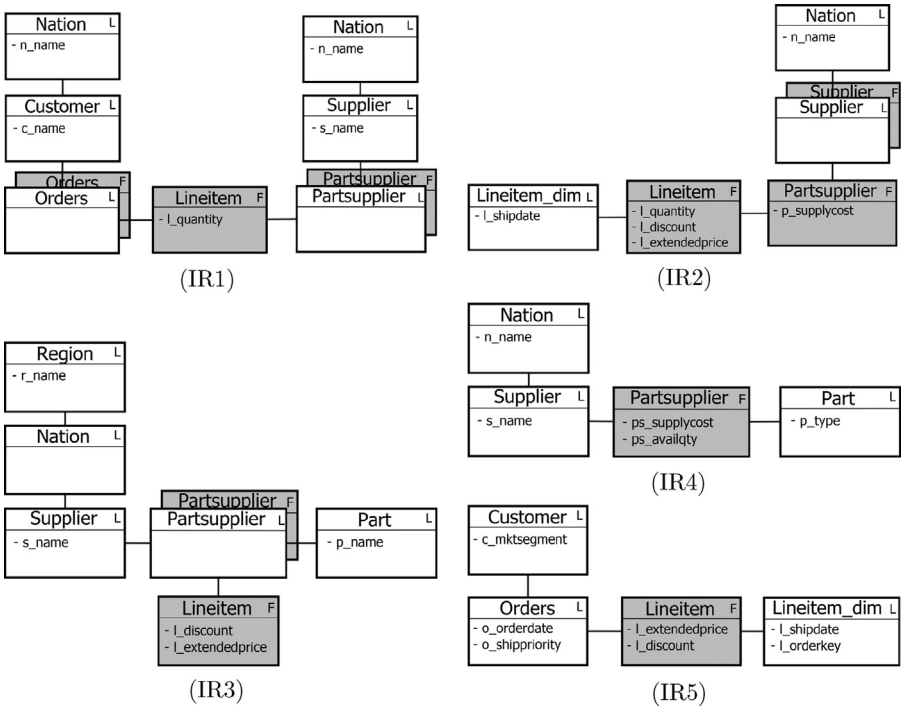

Fig. 2. Single MD interpretations for IR1–IR5.

intermediate concepts (i.e., `partsupplier`, and `nation`). In this case, although `nation` is an intermediate concept, its role is set to dimensional as to preserve the MD integrity constraints. However, `partsupplier` remains as an ambivalent concept since its MD interpretation has not been explicitly set in the requirement and cannot be unequivocally inferred without further information (either as factual or dimensional data the MDI satisfies the MD integrity constraints). Similarly, three out of the five requirements in our example (IR1, IR2 and IR3) contain ambivalent concepts and thus, they produce several MDIs (e.g., IR3 would produce two, one where `partsupplier` plays a factual role and thus, it can be used to find measures of potential interest, and another one where it plays a dimensional role and consequently, it is used as an analysis perspective). □

2.3. Formalizing the problem

MDIs, by themselves, cannot serve as a final MD schema. Each of them answers a single requirement (i.e., cube-query) and may show unnecessary information (e.g., intermediate concepts may be of no interest at all and only be relevant for the ETL process). Intuitively, starting from a set of input requirements (each of them represented as an $\mathbb{MDI}_{\mathtt{IR}}$) and the data sources, the problem at hand looks for overlapping subsets of the input MDIs such that produce a single MD schema and meet some quality objectives.

Thus, we start from $\{\mathbb{MDI}_{\mathtt{IR}_i} | i = 1..m\}$ and following an iterative approach, we derive the unified MD schema satisfying the complete set of requirements. In order to lead the integration process, *ORE* uses a cost model (*CM*) which is defined with a set of parametric cost formulae ($\mathbb{CF}$) that evaluate some DW quality factors (*QF*) (e.g., structural complexity). A cost model together with its formulae is a parameter of *ORE* and can be alternatively customized by the designer to consider other quality factors (e.g., see [14]).

Without loss of generality, and for the sake of presentation, we assume that the resulting unified MD schema is a set of star schemas (`SS`) where each star may answer one or more requirements. Similar to MDIs, we define `SS` as follows:

$$SS = (V, E, role : V \rightarrow_{IR_1 \ldots IR_m} \{D_1, \ldots, D_n, F\}), \text{ such that } SS \vDash MD_{ic})$$

where m is the number of input requirements and n is the number of dimensions in the output. This definition is similar to that of MDI, but guarantees that the MD role assigned to each node honors all the input requirements (i.e., "$\rightarrow_{IR_1,\ldots,IR_m}$"). Moreover, an MD schema, unlike a data cube, contains dimension hierarchies, which correspond to sets of levels with a partial, strict order ($\prec$) between them (e.g., *nation* $\prec$ *region*). Next, we search for a set of star schemas ($\mathbb{SS}$) such that

$$\forall i = 1..m : \mathbb{MDI}_{IR_i} \subseteq_q \mathbb{SS}$$

where $\subseteq_q$ implies $\exists MDI_k \in \mathbb{MDI}_{IR_i}$ and $\exists SS_j \in \mathbb{SS}$ such that

$$V_{MDI_k} \subseteq V_{SS_j},$$

$$E_{MDI_k} \subseteq E_{SS_j},$$

$$\forall (a, b) \in E_{MDI_k} : role_{MDI_k}(a) = role_{MDI_k}(b) \Rightarrow role_{SS_j}(a) = role_{SS_j}(b)$$

Intuitively, each `SS` is a superset of the nodes, and edges of, at least, one MDI per requirement. Each connected subgraph with the same tagging in all corresponding MDIs gives rise to either a fact or one dimension in the star. Each `SS` is generated in accordance with the defined quality factors (*QF*) which are implemented by the chosen cost model (*CM*). Below, we further elaborate on how to express *CM* and meet the quality objectives (Section 2.4.2).

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

Example. Starting from the $\mathbb{MDI}_{IR1}$…$\mathbb{MDI}_{IR5}$ identified for our requirements (see Fig. 2), we aim at producing an MD schema meeting certain quality objectives (e.g., minimal structural complexity). *ORE* follows an iterative approach. Thus, it would start integrating the MDIs from IR1 and IR2 and produce a partial result (see Fig. 3). Then, iteratively we integrate the remaining requirements. At each iteration, the quality factors *QF* are evaluated to choose among alternative integration options. Eventually, our method produces a single $\mathbb{SS}$ satisfying the input requirements and meeting the chosen quality objectives (see Fig. 4). □

2.4. ORE in a nutshell

This section presents the core components of our method (i.e., *ORE*). We first describe the inputs to our system and then, the processing stages of *ORE*.

2.4.1. Inputs

Data sources: Disparate internal and external sources may be of interest for an organization during the decision making processes. To boost the integration of new information requirements spanning diverse data sources into the final MD schema design, we capture the semantics (e.g., concepts, properties) of the available data sources in terms of a domain OWL ontology. The main role of the domain ontology is supporting the integration of heterogeneous sources. Moreover, the use of an ontology, as proposed in [15], allows us to automatically infer (by means of reasoning) relations between concepts (e.g., synonyms, functional dependencies, taxonomies, etc.). In the literature, many works have previously proposed to tackle data integration by means of ontologies. In this paper, we assume the ontology is already available (how to obtain such ontology is out of the scope). For example, we can create or maintain a domain ontology as proposed in [7,16].

Information requirements (IR): Information requirements are pre-processed as previously explained in Section 2.2 and for each requirement, an $\mathbb{MDI}_{IR}$ is generated. Thus, the input of *ORE* is a set of $\mathbb{MDI}$s (one per requirement). As discussed in Section 6, our prototype expects $\mathbb{MDI}$s serialized as XML files. Thus, our method is flexible and the designer can choose any approach to generate the $\mathbb{MDI}$s as far as the results are serialized in a propriety format, which can be directly processed by *ORE* (see Section 6.1).

Example. In addition to the $\mathbb{MDI}$s for requirements IR1–IR5 (see Fig. 2), which have already been discussed in Section 2.2, we use a domain ontology that corresponds to the TPC-H data stores (see Fig. 1). In our example, we followed the approach presented in [16] to automatically produce the TPC-H ontology. However, due to space considerations, we do not further elaborate on this and we refer the interested readers to that paper for additional details. Both, the TPC-H ontology and the set of $\mathbb{MDI}$s in Fig. 3 are inputs of *ORE*. □

2.4.2. Stages

A schematic overview of our approach is illustrated in Fig. 5. Our method semi-automatically integrates new information requirements and incrementally produces an MD schema satisfying the requirements so far. At the same time, the set of operations that illustrates such integration step is identified (i.e., *integration operations*) and further weighted (see Table 2) according to the cost model to assist designer's choice on meeting the chosen quality objectives (e.g., minimal structural complexity).

Our method, *ORE*, comprises four stages, namely *matching facts*, *matching dimensions*, *complementing the MD design*, and *integration* (see Fig. 5). The first three stages gradually match different MD concepts and explore new design alternatives. The last stage considers these matchings and designer's feedback to generate the final MD schema that accommodates a new IR. If a new requirement does not entail any modification on the current MD schema it means that it is subsumed by the previous requirements considered.

Throughout all these stages, we use an internal structure, namely *traceability metadata* (*TM*), for systematically tracing the MD knowledge integrated so far. With *TM*, we avoid overburdening the produced MD schema with insignificant information for the final (business) user. For example, this structure keeps the information about *all* alternative $\mathbb{MDI}_{IR}$, while *only one* MDI_{IR} is chosen per requirement to be included in the final MD schema. We keep these alternatives because, in the future, due to new requirements, we may need to reconsider the integration strategy chosen for a given concept. More details about the *TM* structure are provided in Section 3. At each iteration, the *TM* grows with the integration of each requirement and we use a cost model to prioritize the most promising solutions and prune the rest. *TM*, along with the user feedback, provides the basis for obtaining the final *MD schema*.

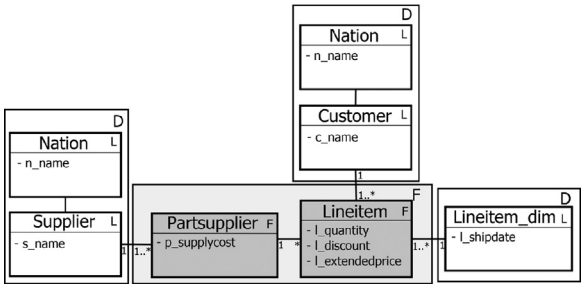

Fig. 3. MD schema satisfying IR1 and IR2.

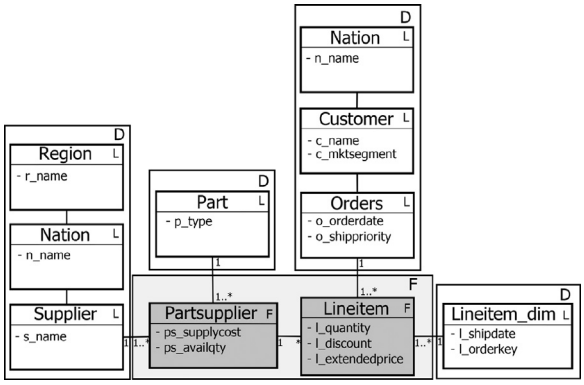

Fig. 4. MD schema satisfying IR1–IR5.

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

Consequently, a model for determining the cost of the output alternative solutions is considered as a part of *TM*. Depending on the DW quality factors that the user is interested in, the cost model can be arbitrary selected by the designer to support the end-user choice. Thus, *ORE* is not coupled to any specific cost model. For the purpose of explanations and prototyping we consider the *structural complexity* as a DW quality factor for building our cost model. The structural complexity is discussed in [14] and different metrics are provided to support the cost model. These metrics refer to the correlation between DW quality factors, like *understandability*, *analyzability* and *maintainability*, and different structural characteristics of a DW schema, like number of dimensional and factual concepts, their attributes (i.e., measures or descriptors), and number of functional dependencies. Table 2 shows the set of theoretically and empirically validated weights that express how introducing different DW concepts (by means of different integration operations) affects the considered DW quality factors (in this case structural complexity). We use this set of weights as an example case for *ORE*, and refer the reader to [14] for more details about how these values are obtained and later validated. We build our example cost model with the following formula ($f \in \mathbb{CF}$) that calculates the overall cost (i.e., structural complexity) of the MD schema. Note that the formula is parametrized with the weights defined in Table 2 and considers the number of different structural elements of an MD schema.

$$f = \#level \cdot 0.21 + \#rollUp \cdot 0.27 + \#dimDescriptor \cdot 0.04 + \#fact \cdot 0.31 + \#factMeasure \cdot 0.36.$$

Example. The MD schema satisfying *IR1–IR5*, depicted in Fig. 4, contains the following MD structural elements: two facts with five measures, eight levels with nine level attributes and four roll-up relations between the levels. Taking into account the formula *f*, we can calculate the overall cost in terms of structural complexity of the MD schema as follows:

$$f = 8 \cdot 0.21 + 4 \cdot 0.27 + 9 \cdot 0.04 + 2 \cdot 0.31 + 5 \cdot 0.36 = 5.54. \quad \square$$

Next, we give a high-level description of the four stages, during which *ORE* iteratively integrates each new requirement into the MD schema that satisfies the requirements so far, preserves the MD integrity constraints and meets the chosen quality objectives.

Stage 1: Matching facts: Facts are the core MD concepts. Thus, we start looking for potential matches between facts in the current IR and the MD schema at hand. Consequently, we first search for different alternatives to incorporate the new IR (i.e., $\mathbb{MDI}_{IR}$) into *TM*. If *ORE* does not succeed, a new SS is created to support the new requirement. For matching facts, *ORE* searches for fact(s) in *TM* that produce a compatible set of points in the MD space. As a result, different possibilities to match the factual concepts in IR with already processed requirements are identified, as well as the appropriate sets of integration operations. The costs of these integration possibilities are further weighted using the chosen cost model, as explained in the previous example.

Stage 2: Matching dimensions: After matching facts, *ORE* conforms the dimensions producing the MD space of the merged fact. Different matches among levels are considered and thus, the different valid integration possibilities are obtained. With each possibility, a different set of integration operations for conforming these dimensions is considered and weighted.

Stage 3: Complementing the MD Design: *ORE* further explores the domain ontology and searches for new analytical perspectives (e.g., proposing new MD concepts of potential interest). Nevertheless, *ORE* never enriches the design without the designer's acknowledgement and it is up to her to extend the current schema with new MD concepts in the ontology (i.e., *levels*, *descriptors*, and *measures*). Consequently, the designer is asked to (dis)approve the integration of the

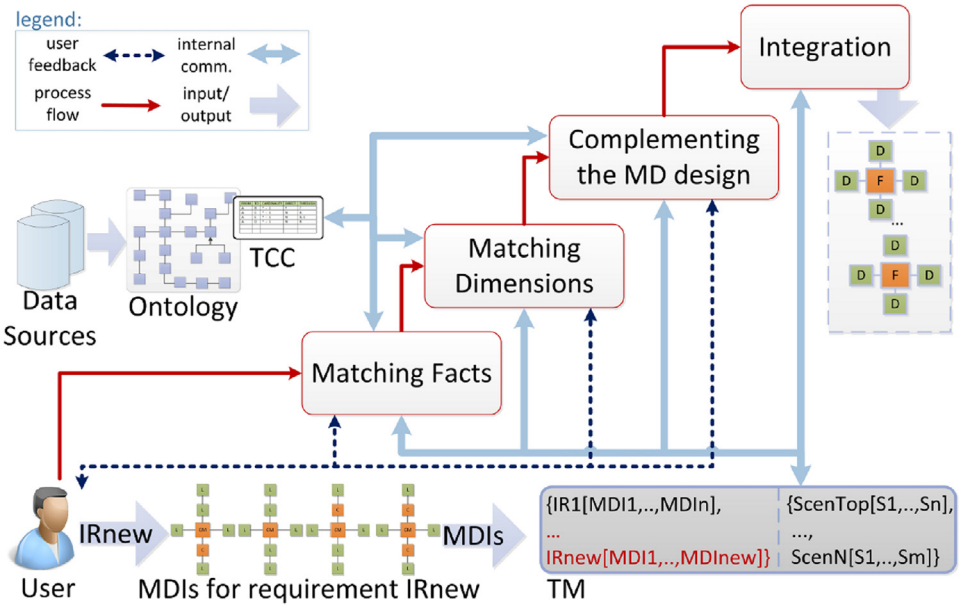

Fig. 5. Abstract representation of *ORE* stages.

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

discovered concepts into the final MD schema.

Stage 4: Integration: The MD schema is finally obtained in two phases. First, we identify possible groupings of concepts containing an equivalent MD knowledge and which are directly connected by means of an MD compliment relationship (i.e., *adjacent concepts*). Then, we collapse them to capture the minimal information relevant to the user. Nevertheless, the complete *TM* is still preserved in the background to assists further integration steps.

3. Traceability metadata

Introducing traceability into software projects has been widely recognized as beneficial (e.g., [17]) for several reasons: (i) the comprehension and understanding of the final products, (ii) maintenance and reusability of the existing software and, (iii) identifying the parts of the final product that fulfill particular input requirements and hence, the impact that the changes in those requirements have on the final product.

Table 2
Integration operations.

DW concept	Operation name	Weight
Dimensional	insertLevel	0.21
	insertRollUp	0.27
	insertDimDescriptor	0.04
	MergeLevels	0.04 ×(# insertDimDescriptor)
Factual	insertFact	0.31
	insertFactMeasure	0.36
	MergeFacts	0.36 ×(# insertFactMeasure)
Factual/Dimensional	renameConcept	0

For the design of DWs, however, the potential advantages of traceability have been largely overlooked. In a recent research work [18], the attention is given to the benefits of keeping traces during the DW lifecycle. The authors identify three kinds of valuable traces in the DW context: (i) traces coming from information requirements, (ii) traces coming from the underlying data sources, and (iii) traces linking elements in the MD conceptual models. Through our iterative approach, we aim at keeping the similar set of traces to enhance the MD schema design. To do so, we introduce a structure, namely *traceability metadata* (*TM*), for systematically keeping the interesting information about the MD knowledge integrated so far.

In Fig. 6, we present the conceptual model of our *TM* structure. For the sake of comprehension, it has been divided in four areas (**1**–**4**), describing four types of traceable information which helps to assess the impact that particular requirements have on the resulting MD design. In comparison to the work in [18], we keep the traces coming from information requirements, i.e., *requirement-related metadata* (see **1** in Fig. 6) and the traces linking elements in the MD conceptual models, i.e., *resulting MD schema* and *MD integration metadata* (see **2** and **3** in Fig. 6). Notice that in this paper, we only discuss the metadata related to schema transformations that are essential for *ORE* and thus, the traces related to the underlying data sources and data, which are essential for building the ETL processes, are not considered here but in our overall framework for the design and evolution of DW (see Section 8). Additionally, we keep the metadata about the cost model being used and w.r.t. the cost model, we produce and store *a cost-based space of alternative solutions* (i.e., **4**). *TM* also assists *ORE* to respond to different changes that may occur during the DW lifecycle and to accordingly adapt the MD design of a DW to support such changes. *TM* stores the following information (**1**–**4** in Fig. 6):

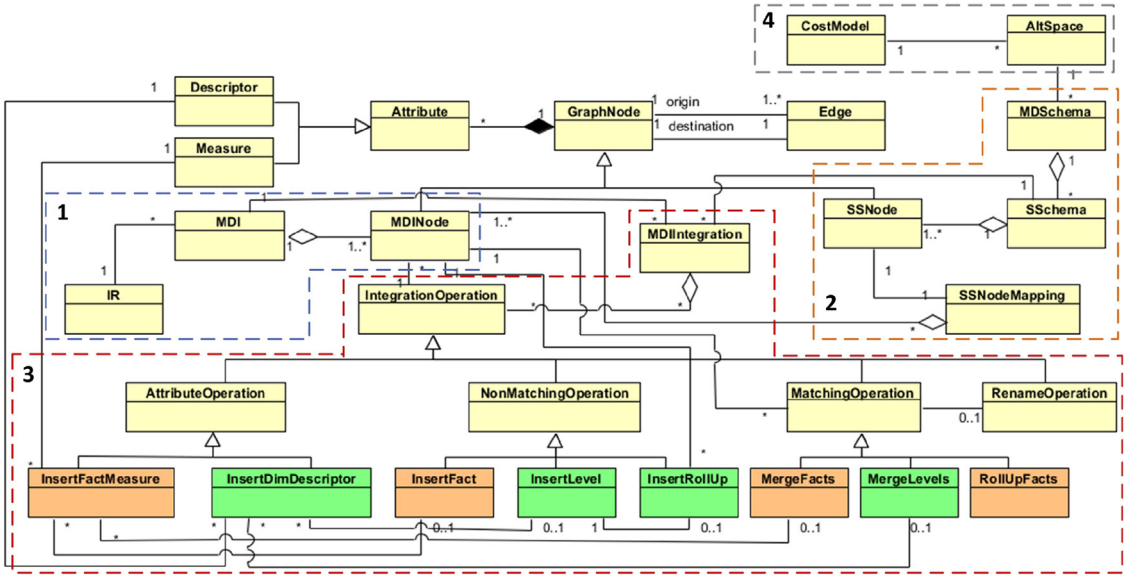

Fig. 6. *TM* conceptual model.

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

1. *Requirement-related metadata*: As discussed in Section 2.2, IRs are represented as MDIs, which are modeled as tagged graphs. For this reason, it is mandatory that every MDINode is *tagged* with an MD role (either *fact* or *level*). Thus, in *TM*, for each *IR*, we store its $\mathbb{MDI}_{IR}$s as tagged nodes (MDINode) and edges (Edge). Furthermore, nodes (GraphNode) may contain attributes either dimensional (Descriptor, if the node is a level) or factual (Measure, if the node is a fact).
2. *Resulting MD schema*: An MD schema (MDSchema) may contain several star schemas (SSchema), which are essentially extended MDIs (see Section 2.2). Therefore, they are also implemented as tagged graphs (with nodes, SSNode, and edges Edges). Importantly, *TM* stores many different MDSchemas, each one reflecting different integration alternatives. These integration alternatives are described through mappings (SSNodeMapping) between MDINodes and SSNodes.
3. *MD integration metadata*: When an evolution event occurs (namely *adding* new information requirement), *ORE* efficiently accommodates the existing MD schemas to these changes and records their occurrences for assisting the user in future design steps. In *TM*, we store these changes as a set of operations (IntegrationOperation), which are necessary to integrate individual MD concepts (MDINodes) coming from new requirements (i.e., new MDIs) into the existing MD design. There are several types of integration operations: (1) operations for inserting new MD concepts into the existing design (i.e., InsertFact, InsertLevel and InsertRollUp), (2) operations for enriching the existing concepts with new MD attributes (i.e., InsertFactMeasure and InsertDimDescriptor) and (3) operations for combining new MD concepts with the existing ones (i.e., MergeFacts, MergeLevels, RollupFacts and RenameOperation). According to Fig. 6, in order to integrate two MDIs (MDIIntegration) we may need several integration operations (IntegrationOperation). Note that all operations are applied to one MDINode, except MatchingOperations and InsertRollUp that refer to an additional MDI. Furthermore, when inserting or merging, levels or facts, an additional set of operations may be identified (see bottom part of Fig. 6). These operations can either insert new attributes to these concepts (i.e., InsertFactMeasure and InsertDimDescriptor), or when inserting a new level (insertLevel), to insert an additional roll-up relation (i.e., InsertRollUp).
4. *Cost-based space of alternative solutions*: Finally, *TM* also stores a reference cost model (CostModel) and a space of alternative solutions (AltSpace). When integrating new requirements, it may occur that several alternative scenarios appear to be valid outcomes of the MD design process (this is mainly due to ambivalent concepts; see Section 2.2). However, considering the chosen DW quality factors (see Section 2.4.2), not all of these output schemas would have the same overall score. In our example case the cost depends on the size and the structural complexity of each final solution. To assist the end user in efficiently determining the most suitable solution, in *TM* we maintain an ordered space of alternative solutions (AltSpace), according to the overall score that is calculated as explained in Section 2.4.2 by using the referenced cost model (CostModel). Thus, we compute, for each integration alternative, its cost. More specifically, each of the integration operations presented above adds a certain weight (see Table 2) to the overall cost of the solution. As discussed, to integrate two MDIs we may need several integration operations and consequently, *ORE* keeps track of the overall cost needed to integrate them, which is used to sort the space of alternatives.
   We define this space as a partially ordered graph which is represented with a triplet $(\mathbb{S}, \mathbb{NB}, S_t)$. $\mathbb{S}$ represents a set of valid scenarios in terms of final MD schemas and $\mathbb{NB}$ (*Next Best*) is a set of directed edges that order these alternative solutions inside the space. At any moment, the solution with the lowest overall cost can be identified as *the top* solution (S_t). In the case that the end user does not find the proposed top solution suitable for her needs, the $\mathbb{NB}$ edges going from that solution will lead her to its *nearest* alternatives, i.e., the next best solutions. Relevantly, our search space only contains alternatives considering all requirements at hand (i.e., partial results are not kept). Moreover, considering the chosen cost model, the search space can be pruned to include only the top-N alternative solutions (see Section 5).

TM is a building block of *ORE* and it is constantly maintained to reflect and fulfill the set of IRs at hand. For example:

Adding an IR. Whenever a new requirement (*IR*) is posed, the *TM* is enriched with the missing metadata to answer such requirement. In particular, new $\mathbb{MDI}_{IR}$ that are identified for a given requirement are added to the *TM*. The added $\mathbb{MDI}_{IR}$ initiates a new iteration of *ORE* to identify how the requirement at hand can be integrated into the existing structures. To this end, we identify all parts of the existing schemas (i.e., SSNodes), where the MDIs of a new requirement can be integrated. Going from the identified SSNodes, through the SSNodeMapping we further identify possible relationships of the new MDIs with the MDIs of previous requirements. In the following section, we describe how an iteration of *ORE* is performed for such addition.

Example. Fig. 3 shows the top solution S_t for integration of IR1 and IR2. If a new requirement comes (e.g., IR3), *ORE* will ask for $\mathbb{MDI}$ of IR3 and go through its four internal stages to match these MDIs with the current space of alternatives ($\mathbb{S}$). At the end, it will produce new space ($\mathbb{S}_{new}$) of MD schemas ($\mathbb{SS}_{new}$), each of them showing different integration alternatives, such that

$$\forall \mathbb{SS}_{new} \in \mathbb{S}_{new}, \quad \forall i = 1..3 : \mathbb{MDI}_{IR_i} \subseteq_q \mathbb{SS}_{new}$$

These new schemas are properly ordered (according to our cost model) and stored in the space of alternative solutions (i.e., AltSpace). □

Removing and changing an IR: Besides accommodating the current MD design to satisfy new IRs, *ORE* also needs to react in response to a changed or disregarded requirement. In the case of disregarding the existing requirement, the requirement (*IR*) is removed together with its $\mathbb{MDI}_{IR}$. *ORE* is then relaunched for the remaining set of information requirements. When the exiting requirement is changed, *ORE* first removes

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

the obsolete requirement and its MDIs and then adds a new (changed) requirement (*IR*) together with its $\mathbb{MDI}_{IR}$. The main process of *ORE* is then relaunched for the updated set of requirements. Note that relaunching *ORE* is not that costly at this point, considering the fact that the integration options for the existing MDIs are already stored in *TM* together with the user's feedbacks and preferences.

Algorithm: ORE.

inputs: $\mathbb{MDI}_{IR}, \mathbb{S}$, **output** : $\mathbb{S}_{new}$
1. *options*:=∅;
2. **For each** $\mathbb{SS}_{cur} \in \mathbb{S}$ **do**
   (a) **For each** $[MDI_i \in \mathbb{MDI}_{IR}, SS_j \in \mathbb{SS}_{cur}]$ **do**
      i. *matchedFactsOpers* := *FM*(getFact(MDI_i),getFact(SS_j));
      ii. **If** *matchedFactsOpers* ≠ {*insertFact*(F_{MDI_i})} **do**
         A. $\mathbb{D}_{MDI_i}$:= searchDimsOverFact (F_{MDI_i});
         B. $\mathbb{D}_{SS_j}$:= searchDimsOverFact (F_{SS_j});
         C. **For each** $D_{MDI_i} \in \mathbb{D}_{MDI_i}, D_{SS_j} \in \mathbb{D}_{SS_j}$ **do**
         If *related*(*bottom*(D_{MDI_i}), *bottom*(D_{SS_j})) **then**
         matchedDimsOpers ∪ = *DM*({*bottom*(D_{MDI_i})}, {*bottom*(D_{SS_j})});
         D. *options* ∪ = [$\mathbb{SS}_{cur} \backslash SS_j, SS_j$, matchedFactsOpers, matchedDimsOpers];
3. **For each** $o \in$ *findBestN*(*options*) **do**
   (a) $\mathbb{S}_{new}$ ∪ = *applyOperations*(*o*);
4. $\mathbb{S}_{new}$:=*INT*(*complementingMDSchema* (*applyOperations*(*findTop*(*options*))));
5. **return** $\mathbb{S}_{new}$;

4. The ORE approach

As we shown in the previous section, after each iteration the *TM* structure in *ORE* contains information that satisfies the current set of requirements and no intermediate results are stored. The space of alternative solutions is also created at the end to support users in finding the most suitable MD schema. For each new requirement *ORE* tries to find the valid correspondences among the MD concepts and to incorporate the new requirement into the existing MD design. At the same time, *ORE* attempts to meet previously set quality objectives and to produce the minimal cost of the output schema according to the proposed cost model (see Section 2.4.2). An exhaustive search inside the current space of alternatives is performed to achieve such goals. The main process of *ORE* is shown in the ORE algorithm.

Note that in order to uniquely reference the steps of the algorithms throughout the paper we use the notation (Step "algorithm:step").

Internally, whenever a new requirement arrives (IR_{new}), ORE tries to match each MD interpretation (MDI_i) of that requirement with the schemas (SS_j) of each solution in the current space of alternatives ($\mathbb{S}$). The correspondences between an MDI_i and an SS_j are found through the matchings of their individual MD concepts, i.e., *facts* (FM call in Step ORE:2(a)i) and *dimensions* (DM call in Step ORE:2(a)iiC). For such matchings, *ORE* benefits from the ontology reasoning mechanisms to find relationships among the concepts from heterogeneous data sources, in an efficient and scalable manner (i.e., synonyms, taxonomies, direct and transitive associations). Notice that only the associations that preserve the MD integrity constraints are accepted. After all the operations to incorporate new concepts into the existing design have been found, ORE considers creating the new space of alternative solutions (Step ORE:3). The cost of each solution is calculated using the selected cost model as described in Section 2.4.2. Eventually, the set of solutions is pruned by taking into account their overall costs, such that only the best *N* solutions are kept for further consideration (the value of *N* can be parametrized by the user). Among these *N* solutions ORE considers the top one to produce the final MD schema by complementing the current one with new knowledge and adapting the level of details of the output MD schema to the end-user's needs (INT in Step ORE:4).

4.1. Matching facts

Respecting the MD integrity constraints described in Section 2.2, in order to match two facts, these should produce an equivalent set of points in the MD space. This is formally defined with the following condition (*C*):

(*C*): The fact $F_{MDI_i} \in MDI_i$ *matches* the fact $F_{SS_j} \in SS_j$ if and only if there is a *bijective function f* such that for each point x_{MDI_i} in the MD space arranged by the dimensions $\{D_1 \times D_2 \times .. \times D_n\}$ implied by F_{MDI_i}, there is one and only one point y_{SS_j} in the MD space arranged by the dimensions $\{D'_1 \times D'_2 \times .. \times D'_m\}$ implied by F_{SS_j}, such that $f(x_{MDI_i}) = y_{SS_j}$.

The abstraction of the fact matching process that guarantees the fulfillment of the above condition (*C*) is described by the FM algorithm. Thus, FM tests whether *C* is satisfied by checking whether $F_{MDI_i}(F_{SS_j})$ functionally determines all the dimensions of $F_{SS_j}(F_{MDI_i})$, i.e., whether $F_{MDI_i}(F_{SS_j})$ is related by means of "1 −1" or "* −1" relationship to each dimension of $F_{SS_j}(F_{MDI_i})$ (Step FM:3(c)). If this happens, it is because either they share the same MD space (Step FM:3(c)i) or one can be rolled up to the other (Steps FM:3(c)iiA and FM:3(c)iiiA). These relationships are searched by means of reasoning over the domain ontology (see Section 2.4.1). As a special case, such matching is achieved if both facts coincide or transitively if one fact functionally determines the other, which is cheaper to be checked. Therefore, as an optimization, we firstly check the latter (Steps FM:1 and FM:2, respectively). It is worth noting that if we roll up a fact, we must check whether the new granularity is still acceptable for the associated requirements.

If the full match between both facts is found, i.e., they are equal (Step FM:1) or they functionally determine each other (Steps FM:2a and FM:3(c)i), the *mergeFacts* operation is added, followed by *renameConcept*, if they are not equal. Alternatively, if one fact functionally determines the other (Steps FM:2(b)i and FM:2(c)i) or if it functionally determines all the dimensions of the other fact (Steps FM:3(c)iiA and FM:3(c)iiiA) the *rollupFacts* operation is identified to roll-up from the MD space of one fact to the MD space of the other (only when the coarser granularity is acceptable for all involved requirements). Otherwise, if FM cannot identify a valid matching for the fact F_{MDI_i}, it generates the *insertFact* operation (Step FM:3d).

Algorithm: FM.

inputs: F_{MDI_i}, F_{SS_j}, **output** : *intOps*
1. **If** $F_{MDI_i} = = F_{SS_j}$ **then** *intOps*:={*mergeFacts*(F_{MDI_i}, F_{SS_j})};
2. **ElseIf** $F_{MDI_i} \rightarrow F_{SS_j} \vee F_{SS_j} \rightarrow F_{MDI_i}$ **then**
   (a) **If** $F_{MDI_i} \rightarrow F_{SS_j} \wedge F_{SS_j} \leftarrow F_{MDI_i}$ **then**
   intOps:={*mergeFacts*(F_{MDI_i}, F_{SS_j}), *renameConcept*(F_{MDI_i}, F_{SS_j})};
   (b) **ElseIf** $F_{MDI_i} \rightarrow F_{SS_j}$ **then**
      i. **If** *acceptableGranularity*(MDI_i) **then**

intOps:={*rollupFacts*(F_{MDI_i}, F_{SS_j})};
(c) **Else** // $F_{SS_j} \rightarrow F_{MDI_i}$
i. **If** *acceptableGranularity*(SS_j) **then**
intOps:={*rollupFacts*(F_{SS_j}, F_{MDI_i})};
3. **Else**
(a) $\mathbb{D}_{MDI_i}$:=searchDimsOverFact (F_{MDI_i});
(b) $\mathbb{D}_{SS_j}$:=searchDimsOverFact (F_{SS_j});
(c) **If** $F_{MDI_i} \rightarrow \mathbb{D}_{SS_j} \vee F_{SS_j} \rightarrow \mathbb{D}_{MDI_i}$ **then**
i. **If** $F_{MDI_i} \rightarrow \mathbb{D}_{SS_j} \wedge F_{SS_j} \rightarrow \mathbb{D}_{MDI_i}$ **then**
intOps:={*mergeFacts*(F_{MDI_i}, F_{SS_j}), *renameConcept*(F_{MDI_i}, F_{SS_j})};
ii. **ElseIf** $F_{MDI_i} \rightarrow \mathbb{D}_{SS_j}$ **then**
A. **If** *acceptableGranularity*(MDI_i) **then**
intOps:={*rollupFacts*(F_{SS_j}, F_{MDI_i})};
iii. **Else** // $F_{SS_j} \rightarrow \mathbb{D}_{MDI_i}$
A. **If** *acceptableGranularity*(SS_j) **then**
intOps:={*rollupFacts*(F_{MDI_i}, F_{SS_j})};
(d) **Else** *intOps*:={*insertFact*(F_{MDI_i})};
4. **return** *intOps*;

Example. [Step FM:1] Fig. 7 shows the case when the fact `Lineitem` of the requirement *IR2* matches the same fact in the existing *TM*. □

Example. [Step FM:2] Fig. 8 illustrates the case where the fact `Partsupplier` of *IR4* matches the fact `Lineitem` that functionally determines it, which is identified in the ontology that captures the TPC-H data sources (see TPC-H schema in Fig. 1). Therefore, the operation *rollupFacts*(*Lineitem*, *Partsupplier*) is identified. Notice that in this case, we still should check whether the new granularity allows to answer the requirements associated to SS_j. □

Example. [Step FM:3c] Fig. 9 shows an example inspired by TPC-H. A new `PartMarket` fact is introduced, which assesses the convenience of releasing a specific `Part` in a specific market (i.e., `Nation`). Even though we may find no direct matching (i.e., Steps FM:1 and FM:2) between `PartMarket` and the `Supplier-PartSupplier` fact, their MD spaces do coincide as they both functionally determine the dimensions (i.e., `Nation` and `Part`) of each other. On the other side, in Fig. 10, in the variation of the above example we have the case that the dimensions of the fact `PartMarket` are functionally determined by the `Lineitem-PartSupplier` fact but the opposite does not hold and thus, the *rollupFacts*(*Lineitem* – *PartSupplier*, *PartMarket*) operation should be added. As in the previous example, we still have to guarantee that the new granularity allows to satisfy the requirements associated to SS_j. □

Finally, all integration possibilities represented through the identified operations (i.e., *rollupFacts*, *mergeFact*, *renameConcept* or *insertFact*) are listed with the weights they add to the final solutions (see Table 2). *ORE* then creates an alternative solution for each of the integration possibilities. To prioritize the resulting solutions, the overall cost of each solution is evaluated according to predefined quality factors.

4.2. Matching dimensions

In this stage, *ORE* conforms the dimensions which form the MD spaces of the facts (F_{MDI_i}, F_{SS_j}) for which it previously found a matching (Step ORE:2(a)iiC).

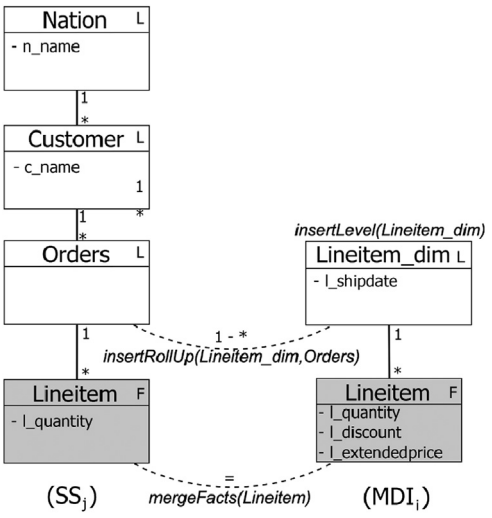

Fig. 7. Matching the MD Interpretation for IR2.

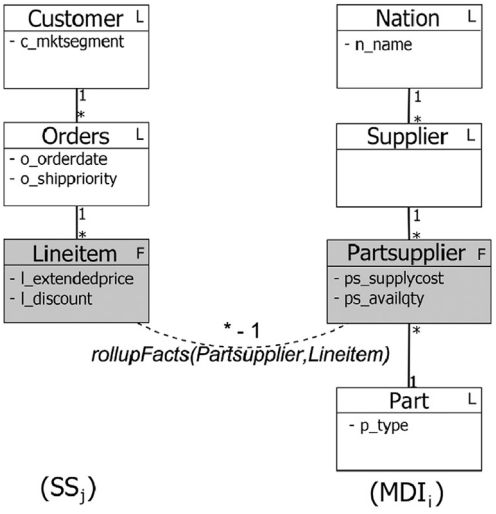

Fig. 8. Matching facts with *rollupFacts*.

Since a single dimension D_x consists of a partially ordered set of individual levels (see Section 2.2), with single bottom (i.e., atomic) level and with "to-one" relationships among the levels, each dimension may be seen as a directed acyclic graph (DAG), where the vertices of the graph are individual levels and the directed edges between them are "to-one" relationships (i.e., functional dependencies). Note that we assume the dimension graph is guaranteed to be acyclic (this should be checked in a preliminar step), since the loops would violate the MD integrity constraints [3].

Algorithm: DM.

inputs: $candidates_{MDI_i}$, $candidates_{SS_j}$, **output:** *intOps*
1. **If** $L_{MDI_i} = = L_{SS_j}$ **then**
intOps $\cup$ ={*mergeLevels*(L_{MDI_i}, L_{SS_j})}⊎*DM*(*getNext*(L_{MDI_i}), *getNext*(L_{SS_j}));
2. **ElseIf** $L_{MDI_i} \rightarrow L_{SS_j} \vee L_{SS_j} \rightarrow L_{MDI_i}$ **then**
(a) **If** $L_{MDI_i} \rightarrow L_{SS_j} \wedge L_{SS_j} \rightarrow L_{MDI_i}$ **then**
intOps $\cup$ ={*mergeLevels*(L_{MDI_i}, L_{SS_j}), *renameConcept*(L_{MDI_i}, L_{SS_j})}⊎
DM(*getNext*(L_{MDI_i}), *getNext*(L_{SS_j}));

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

(b) **ElseIf** $L_{SS_j} \rightarrow L_{MDI_i}$ **then**
$intOps \cup = \{insertLevel(L_{MDI_i}), insertRollUp(L_{SS_j}, L_{MDI_i})\} \uplus$
$DM(\{L_{MDI_i}\}, getNext(L_{SS_j}))$;
(c) **Else** // $L_{MDI_i} \rightarrow L_{SS_j}$
$intOps \cup = \{insertLevel(L_{MDI_i}), insertRollUp(L_{MDI_i}, L_{SS_j})\} \uplus$
$DM(getNext(L_{MDI_i}), \{L_{SS_j}\})$;

3. **Else** // No matching for current levels
(a) $intOps \cup = DM(\{L_{MDI_i}\}, getNext(L_{SS_j}))$;
(b) $intOps \cup = \{insertLevel(L_{MDI_i})\} \uplus DM(getNext(L_{MDI_i}), \{L_{SS_j}\})$;
4. **return** *intOps*;

Having this in mind, the problem of matching dimensions may be seen as a graph matching problem. However, taking into account the MD context, *ORE* must additionally preserve the MD integrity constraints (see Section 2.2). Thus, we present here the DM algorithm, which solves the problem in our case.

DM is launched from the main (ORE) algorithm (Step ORE:2 (a)iiC) for each pair of dimensions coming from MDI_i ($\mathbb{D}_{MDI_i}$) and SS_j ($\mathbb{D}_{SS_j}$) with the previously matched facts (F_{MDI_i}, F_{SS_j}) and bottom (atomic) levels connected (*related*) with an MD-compliant relationships (i.e., "=", "1-1", "1-∗", and "∗-1").

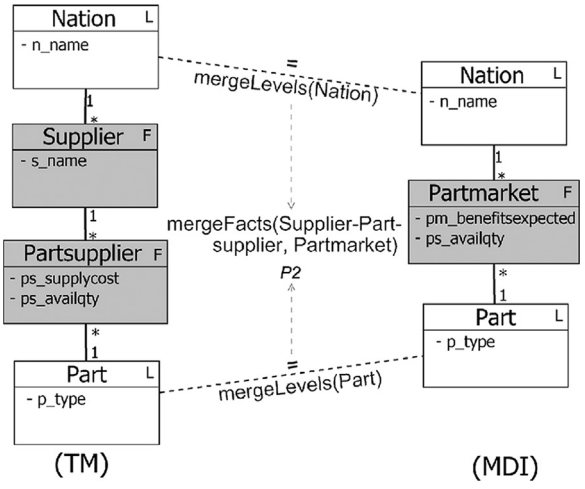

Fig. 9. Matching facts with *mergeFacts*.

Considering the topological order of levels in each of the dimensions, DM starts by matching the atomic (*bottom*) levels of these dimensions.

In each call, DM searches for the matchings between all pairs of the candidate levels of MDI_i (i.e., $candidates_{MDI_i}$) and each candidate from the dimensions of SS_j (i.e., $candidates_{SS_j}$), starting from the bottom of each of them and hence, recursively moves forward through the corresponding hierarchy, depending on the multiplicity of the relationship found. For each pair of *candidate* levels, DM first checks if they exactly coincide (Step DM:1). If not, it may be that one functionally determines the other (Step DM:2). This can be either the case when both levels functionally determine each other, i.e., "1-1" relationship (Step DM:2a), or the case when only one of them functionally determines the other one, i.e., either "1-∗" or "∗-1" relationship (Steps DM:2b and DM:2c, respectively). Again, these relationships are identified by means of reasoning over the input ontology.

If the levels are equal or the "1-1" relationship is found (i.e., *a full match*), a recursive call moves forward in both hierarchies. However, when a "1-∗" or "∗-1" relationship is identified, DM can still explore the possibility that the level being in the "to-one" side matches the next one in the to-many side of the relationship. Thus, it only moves forward in one of the hierarchies. Finally, if there is no relationship between the levels (Step DM:3), DM considers both alternatives, i.e., moving forward in either one or the other hierarchy. Thus, DM exhaustively visits all meaningful matchings for dimensions of MDI_i and of SS_j, and builds the set of possible integration options (*intOps*).

For each integration option, DM stores the information about the corresponding operations to be applied (see Table 2). When the *full match* between L_{MDI_i} and L_{SS_j} is found, either by equality or "1-1" relationship (Steps DM:1 and DM:2a), we consider either only the *mergeLevels* or the *mergeLevels* with the *renameConcept* operation to be applied, respectively.

Example. Fig. 11 shows an integration possibility for the `Orders` dimension, in the case of integrating IR5 into the

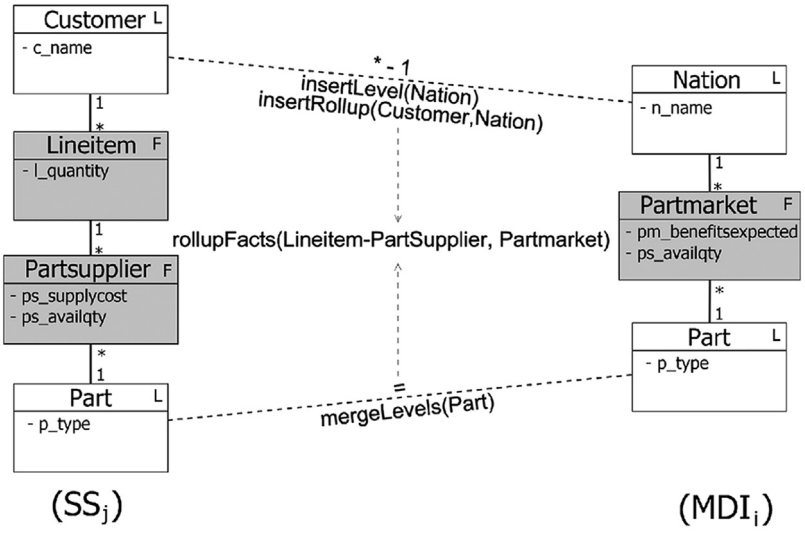

Fig. 10. Matching facts with *rollupFacts*.

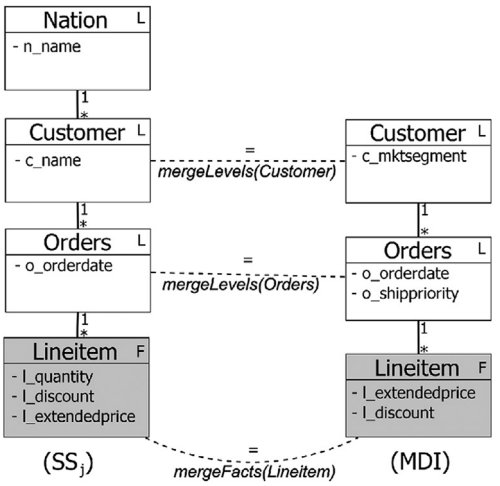

Fig. 11. Matching the MD Interpretation for IR5.

TM satisfying IR1–IR4. A full matching (i.e., equality) is then found for both `Orders` and `Customer` levels of the corresponding MDI_i. Thus, the *mergeLevels* operations are proposed for both `Orders` and `Customer` levels and they respectively involve *insertDimDescriptor* operations for transfering `o_shippriority` and `c_mktsegment` descriptors. □

On the other hand, if "1 - ∗" or "∗ - 1" relationships are identified (Step DM:2b and DM:2c), we consider inserting a roll-up relation (i.e., *insertRollUp* operation) in the star SS_j to a new inserted level.

Example. In Fig. 7, for the "∗ - 1" matching found between levels `Lineitem_dim` and `Orders`, inserting a roll-up relation is proposed (i.e., *insertRollUp(Lineitem_dim, Orders)*), with the involved insertion of the level (`Lineitem_dim`). □

Similar to the previous stage, we may identify different options to integrate levels in MDI_i with different *candidate* levels of the hierarchies from SS_j. Thus, we must consider the possibility of combining the current integration operations with all the alternatives of the successors (this is shown by the symbol ⊎ in the algorithm). Once all the integration options are found with their corresponding weights, *ORE* creates an alternative solution for each of the integration possibilities.

4.3. Complementing the MD design

After the previous two stages, *ORE* identifies the space of possible solutions for incorporating the new information requirement into the existing schema. As we discussed before, this space can be partially ordered considering user preferences and the cost of the individual solutions. In such a space we can obtain the *top* element that corresponds to the solution currently most suitable for the end user according to predefined quality objectives.

Considering the *top* solution found, in this stage *ORE* continues by analyzing the ontology to complement the future MD design with new analytically interesting concepts. This stage is optional and may be disregarded by the user. By exploring the functional dependencies ("to-one" relationships) in the ontology, *ORE* identifies new *levels* for the previously conformed dimensions. Furthermore, different *datatype properties* in the ontology may also be identified either as *measures* of the existing facts or *descriptive attributes* of the levels.

We distinguish two cases:

- If the property has a numerical data type (e.g., *integer*, *double*), we consider using it as a measure if and only if the domain concept of the property is identified as a fact.
- Otherwise, in the case that the domain concept of a property is identified as a level, our method suggests using the property as a new *descriptor*.

Algorithm: INT.

input: $\mathbb{SS}_{top}$, **output:** $\mathbb{SS}_{new}$
1. $\mathbb{SS}_{new} = \emptyset$;
2. **For each** $SS_j \in \mathbb{SS}_{top}$ **do**
   (a) $SS_{new} := [\emptyset, \emptyset, \emptyset]$;
   (b) $seedF :=$ findFactualConcept(SS_j);
   (c) $\mathbb{F} :=$ group($seedF$);
   (d) setFact(SS_{new},collapse ($\mathbb{F}$));
   (e) **For each** $(f, L_0) \in F_{SS_j} \wedge f \in \mathbb{F} \wedge L_0 \notin \mathbb{F}$ **do**
      i. $\mathbb{D} =$ group(L_0);
      ii. setDimension(SS_{new},collapse ($\mathbb{D}$));
   (f) $\mathbb{SS}_{new} \cup = \{SS_{new}\}$
3. **return** $\mathbb{SS}_{new}$;

Different possibilities for enriching the current design are presented to the designer as different integration operations (i.e., *insertLevel*, *insertFactMeasure*, *insertDimDescriptor*). The designer may decide how to complement the MD design. As shown in [7], automation of such exploration process is highly achievable with polynomial complexity for the ontologies capturing the data source semantics.

Example. For the `Orders` dimension in Fig. 11, *ORE* explores the ontology and proposes concept `Region` as a new top level of the given dimension. Also, *ORE* proposes different descriptors for the levels `Orders`, `Customer`, `Nation`, and `Region`; e.g., `c_phone`, and so on. □

4.4. Integration

Having the *top* solution for producing the MD schema identified in the first two stages and optionally complemented in the third stage, here, *ORE* produces the final MD schema. The integration process is described with the INT algorithm as follows.

The algorithm starts from the set of star schemas of the *top* solution ($\mathbb{SS}_{top}$) produced in the previous stages of *ORE*, which now additionally answers the *IR* at hand.

The factual and the dimensional concepts of each star (SS_j) in $\mathbb{SS}_{top}$ go through two phases, namely *grouping* and *collapsing*.

(1) Grouping: As the ontological concepts can be represented by a directed acyclic graph (DAG), these are combined together to produce different groups (subgraphs), so that all concepts in one group:

1. produce a **connected** subgraph and
2. have the same **MD interpretation** (i.e., all concepts are either factual or dimensional).

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

(II) Collapsing: Starting from these groups of concepts we obtain the final star schema. Inside each subgraph captured by a single group, we consider only the concepts currently required by the user, either provided with the requirement at hand or discovered when complementing the MD design in the ontology (i.e., Section 4.3). The concepts considered inside each group are then collapsed to produce one element (i.e., fact or dimension) of the final MD schema.

The INT algorithm first initializes the output MD schema ($\mathbb{SS}_{new}$, Step INT:1). Next, INT initializes the new star schema (SS_{new}) that results from integrating each star of $\mathbb{SS}_{top}$ (Step INT:2a), with the empty sets of vertices (i.e., facts or dimensions), edges (i.e., relationships among the MD concepts) and MD roles that the vertices play (see how *SS* is defined in Section 2.3). INT then searches for any factual concept (Step INT:2b) and starting from it, INT performs *grouping* of the factual concepts transitively related (Step INT:2c). As a result, the subgraph of related facts is produced and then *collapsed* in the next step (Step INT:2d) to produce a single fact added to SS_{new}. Starting from the collapsed fact, the levels arranging the dimensions over the new fact are explored (Step INT:2e) and starting from the atomic level (L_0), the adjacent levels are *grouped* in a similar way as the facts and *collapsed* to produce a single dimension. The dimensions produced are then also added to SS_{new}. Finally, a new star (SS_{new}) resulted from integrating each SS_j is added to the output MD schema (Step INT: 2f).

In these two phases, we free the final schema from knowledge currently irrelevant to the designer's choices and conform the output MD schema to the level of details suitable to the end-user. For example, collapsing the adjacent levels simplifies the corresponding dimensions and lowers the number of roll-up relationships. The process of integration can optionally be assisted by the user, who can determine the level of integration (details) by means of selecting the type of the finally produced schemas (i.e., *star* or *snowflake*).

While concepts with currently no interest to the designer may be hidden from the final MD schema design, *TM* structure still preserves all this knowledge for using it in future integration steps. Furthermore, as *TM* traces the knowledge about the complete MD interpretations (i.e., including all the concepts) and also the mappings of these concepts to the data sources we can benefit from it when producing the appropriate data flow design (e.g., ETL process) to manage loading the data from the sources to the produced target MD schema. This is the part of our overall research work as it is discussed in Section 8 (e.g., see [19]).

5. Theoretical validation

In this section we present the theoretical validation of our approach where we examine the satisfaction of four major properties:

- *Soundness*: The resulting MD schema must satisfy the MD integrity constraints (see Section 2.2).
- *Completeness*: The set of information requirements integrated so far can be answered from the resulting MD schema.
- *Commutativity*: Independently of the order of the input information requirements, *ORE* must produce an equivalent MD schema at the output.
- *Associativity*: Independently of the order in which information requirements are integrated, *ORE* must produce an equivalent MD schema at the output.

Finally, we discuss the theoretical complexity of the problem of the requirement-driven DW design and our approach.

5.1. Soundness and completeness

Here, we formally prove the soundness and completeness of *ORE* by analyzing the set of integration operations applied through its four stages (see Table 2).

Precondition: For each new requirement (*IR*), *ORE* starts from a set of MDIs ($\mathbb{MDI}_{IR}$) whose soundness (i.e., respecting the MD integrity constraints) and completeness (i.e., satisfying *IR*) are ensured by means of the constraints described in Section 2.2.

Trivial case: When the first requirement arrives (IR_{first}), *ORE* chooses an MDI_i ($MDI_i \in \mathbb{MDI}_{IR_{first}}$) such that MDI_i has the lowest overall cost in $\mathbb{MDI}_{IR_{first}}$ (w.r.t to the chosen cost model) and produces a MD schema by means of the INT algorithm. According to our precondition, all input MDIs are considered to be sound and complete. Notice that the soundness and completeness of the resulting MD schema is guaranteed because the INT algorithm focuses on tuning the level of details in the schema for the presentation purposes rather than affecting its semantics.

Hereinafter, we consider the general case when a new requirement is integrated into an existing (sound and complete) MD schema.

Invariant of the process: Given an $MDI_i \in \mathbb{MDI}_{IR_{new}}$ coming from a new requirement IR_{new} and a star SS_j from a current MD schema $\mathbb{SS}_{cur}$, *ORE*, through the four stages explained in the previous section, applies the set of transformations (i.e, integration operations, see Table 2) to produce the final MD schema. This process must guarantee the *soundness* and *completeness* of the output schema and thus we subsequently show that the results produced by each integration operation of *ORE* always **preserve the MD integrity constraints** (i.e., the *operation* is sound) and able to **answer the current set of requirements** (i.e., the *operation* is complete).

In what follows, we evaluate the above invariant by analyzing each integration operation individually. To analyze the soundness, for each operation, we test whether it only considers relationships that are compliant with the MD integrity constraints; whilst for the completeness, we test whether the newly created MD schema subsumes the complete MD knowledge (i.e., semantics) of the MDIs from which it was created.

Factual concepts operations

- *mergeFacts*(F_{MDI_i}, F_{SS_j}): The FM algorithm of *ORE* (Section 4.1), merges facts $F_{MDI_i} \in MDI_i$ and $F_{SS_j} \in SS_j$ if and only if the MD spaces of facts F_{MDI_i} and F_{SS_j} are equivalent (see condition *C* in Section 4.1). From the *precondition* above (i.e., MDI_i is sound and complete), and by preserving the same MD space and keeping a single fact in the existing sound and complete star SS_j (*invariant*), it is not hard to see that

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

the *mergeFacts* operation also preserves the MD integrity constraints of SS_j and of the resulting MD schema (i.e., *mergeFacts* is *sound*). Moreover, the existing star SS_j, after merging two facts, still answers the previous requirements and additionally the new one through the existing or newly added measure attributes (i.e., *mergeFacts* is *complete*).

- *insertFact*(F_{MDI_i}): In the case that the FM algorithm does not find a fact with the equivalent MD space as the F_{MDI_i} fact of the new requirement, it inserts F_{MDI_i} and creates a new star SS_{new} inside the output MD schema. The *insertFact* operation, similar to the *trivial case*, does not affect the current set of stars in $\mathbb{SS}_{cur}$, but it creates a new star (SS_{new}) in the output MD schema. The soundness of SS_{new} is guaranteed by the soundness of the MDI_i from which it is created (i.e., *insertFact* is *sound*). As the rest of stars in $\mathbb{SS}_{cur}$ stay intact, $\mathbb{SS}_{cur}$ still answers all the previous requirements, while the newly added star (SS_{new}) answers the requirement at hand (IR_{new}) (i.e., *insertFact* is *complete*).
- *insertFactMeasure*($m_{F_{MDI_i}}, F_{SS_j}$): When merging facts $F_{MDI_i} \in MDI_i$ and $F_{SS_j} \in SS_j$, the FM algorithm may insert a new measure $m_{F_{MDI_i}} \in F_{MDI_i}$ into the existing fact F_{SS_j}. It is trivial to see that the *insertFactMeasure* operation does not affect the MD integrity constraints of F_{SS_j} and thus it does not violate the soundness of SS_j and of the output MD schema (i.e., *insertFactMeasure* is *sound*). Moreover, *insertFactMeasure* enriches F_{SS_j} to additionally answer new requirement (i.e., *insertFactMeasure* is *complete*).

Dimensional concepts operations

Note that the DM algorithm (Section 4.2) is only run for the dimensions that form the MD spaces of the facts (F_{MDI_i}, F_{SS_j}) for which it previously found a matching. The DM algorithm can introduce new analysis perspectives (i.e., dimensions) to the existing fact F_{SS_j} only if it finds a relationship between the F_{SS_j} and the new dimension at the data sources (i.e., inside the ontology that captures them). Notice that this does not change the existing answerability of the fact but adds a new perspective through which the fact can be analyzed.

- *mergeLevel*(L_{MDI_i}, L_{SS_j}): The DM algorithm (Section 4.2), merges levels $L_{MDI_i} \in MDI_i$ and $L_{SS_j} \in SS_j$ if and only if it finds $L_{MDI_i} = L_{SS_j}$ or the "1-1" relationship between levels. By keeping the existing level L_{SS_j} in the sound and complete star SS_j (*invariant*), the *mergeLevel* operation does not affect the MD integrity constraints of SS_j (i.e., *mergeLevel* is *sound*), but only potentially enriches the existing level L_{SS_j} with new level attributes to additionally answer the new requirement (i.e., *mergeLevel* is *complete*).
- *insertLevel*(L_{MDI_i}): In the case that the DM algorithm does not find any relationship of level $L_{MDI_i} \in MDI_i$ with levels of a current hierarchy of the existing sound and complete star SS_j (*invariant*), it creates a new branch of the current hierarchy of SS_j and inserts L_{MDI_i} by connecting it with the last matched level in a hierarchy. As DM guarantees to move only through MD-compliant hierarchies of MDI_i and SS_j, *insertLevel* will always connect L_{MDI_i} using a "to-one" relation of the MDI_i hierarchy, and thus it will preserve the MD integrity constraints of SS_j and of the output MD schema (i.e., *insertLevel* is *sound*). Moreover, as lower levels of a hierarchy remain intact, SS_j will still answer all the previous requirements and additionally the new one using a newly added level (L_{MDI_i}) (i.e., *insertLevel* is *complete*).
- *insertRollUp*(L_{SS_j}, L_{MDI_i}): In the case that the DM algorithm finds a "1-∗" or "∗-1" relationship between levels $L_{MDI_i} \in MDI_i$ and $L_{SS_j} \in SS_j$, it adds a new level L_{MDI_i} to the existing sound and complete star SS_j (*invariant*) (by means of the *insertLevel* operation) and then inserts the roll-up relation to relate matching levels, i.e., L_{SS_j} and L_{MDI_i}. By considering only the relationships that fulfill the MD integrity constraints (i.e., "1-∗" or "∗-1"), *insertRollUp* preserves the MD integrity constraints of SS_j and of the output MD schema (i.e., *insertRollUp* is *sound*). As with *insertLevel*, the *insertRollUp* operation keeps the existing levels intact, hence they still answer the previous requirements, while the inserted roll-up relation enables answering the new requirement (i.e., *insertRollUp* is *complete*).
- *insertDimDescriptor*($d_{L_{MDI_i}}, L_{SS_j}$): When merging levels $L_{MDI_i} \in MDI_i$ and $L_{SS_j} \in SS_j$, the DM algorithm may insert a new level descriptor $d_{L_{MDI_i}} \in L_{MDI_i}$ into the level L_{SS_j} of the existing, sound and complete star SS_j (*invariant*). It is trivial to see that the *insertDimDescriptor* operation does not affect the MD integrity constraints of L_{SS_j} and thus preserves the soundness of SS_j and of the output MD schema (i.e., *insertDimDescriptor* is *sound*). Moreover, *insertDimDescriptor* enriches the star SS_j with newly added level descriptors to additionally answer new requirement (i.e., *insertDimDescriptor* is *complete*).

Factual/dimensional concepts operations

- *renameConcept*(C_{MDI_i}, C_{SS_j}): In the case of merging two facts or two levels, *ORE* generates *renameConcept* operation to record that in order to integrate concepts C_{MDI_i} and C_{SS_j} it may be necessary to rename the new concept C_{MDI_i} to the name of the existing one C_{SS_j}. The *renameConcept* operation introduces a syntactic change and thus it does not affect the MD integrity constraints of the existing sound and complete star SS_j (*invariant*) or of the output MD schema (i.e., *renameConcept* is *sound*). For the same reason, *renameConcept* operation neither affects the ability of the output MD schema to answer all the current requirements (i.e., *renameConcept* is *complete*).

From the above analysis, we can conclude that given a sound and complete $MDI_i \in \mathbb{MDI}_{IR_{new}}$ of a new requirement IR_{new} (*precondition*), and a sound and complete star $SS_j \in \mathbb{SS}_{cur}$ resulted from a single requirement (*trivial case*), applying any valid combination of integration operations in the ORE algorithm results in an output MD schema that preserves the MD integrity constraints (i.e., all operations are *sound*) and answers the entire set of requirements (i.e., all operations are *complete*). Thus, the initial *invariant* always holds. This further proves the soundness and completeness of our *ORE* approach and its algorithms.

5.2. Commutativity and associativity

Commutativity. Following the general algebra definition of commutativity, *ORE* needs to satisfy the following

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

property (Note that the binary operator "$\cdot_{ORE}$" represents the application of ORE algorithm over the two information requirements, by means of first integrating the left requirement and then the right one):

$$\forall i,j \in \{1..n\}, IR_i \cdot_{ORE} IR_j = IR_j \cdot_{ORE} IR_i$$

This property sates that for any two requirements IR_i and IR_j, the order in which they are coming at *ORE*'s input does not change the resulting space of alternative solutions.

Associativity. Similarly, if we follow the general algebra definition of associativity, *ORE* needs to satisfy the following property:

$$\forall i,j,k \in \{1..n\}, (IR_i \cdot_{ORE} IR_j) \cdot_{ORE} IR_k = IR_i \cdot_{ORE} (IR_j \cdot_{ORE} IR_k)$$

This property sates that for any three requirements in a fixed order (IR_i, IR_j, and IR_k), the order in which *ORE* integrates them (i.e., first IR_i and IR_j and then IR_k, or first IR_j and IR_k and then IR_i) does not change the resulting space of alternative solutions.

As stated before, when the new information requirement (IR_i) arrives, each of its MDIs is compared to each of the stars of all the MD schemata in the current space of alternative solutions. Only in the cases when the full matching between facts is found (i.e., "=" and "1-1") and when no matching is found at all, the FM algorithm creates a single solution by merging the matched facts (i.e., *mergeFacts*) or inserting the new fact into the output MD schema (i.e., *insertFact*), respectively. In all other cases, two different solutions are created: one, where the facts are merged into a single fact, through their dimensions or changing the fact's granularity (see Steps FM:2(b)i, FM:2(c)i, and FM:3); and another one where two different stars are created around the facts that are previously compared. Furthermore, the DM algorithm preserves all the alternative solutions previously generated by FM and potentially adds new ones by matching the dimensions. *ORE* stores all these alternative solutions inside *TM* (see Section 3).

This characteristic of the ORE algorithm guarantees that no matter in which order the requirements arrive (*commutativity*), or in which order they are integrated (*associativity*), *ORE* exhaustively explores all integration options and thus it always produces the same set of solutions (i.e., MD schemata) at the output either by merging several stars into one or by creating separate stars.

5.3. Computational complexity

We first analyze the complexity of the requirement-driven DW schema design problem. For each new information requirement (IR_i), we compare all the MDIs of that requirement ($MDI_1, .., MDI_p$) with all the stars ($SS_1, .., SS_q$) of all alternative solutions ($\mathbb{SS}_1, .., \mathbb{SS}_r$). Furthermore, due to the different integration options explained in Sections 4.1 and 4.2, we produce different alternative solutions that are compliant with the *soundness* and *completeness* properties discussed in the previous subsection. Considering the fact that for each new requirement we need to compare all its MDIs with all the current alternative solutions and find all the possible integration alternatives, it is not hard to see from the above that the general problem of the requirement-driven DW design is complex (clearly exponential). Formally, we can present it as follows:

If we assume a scenario where n information requirements are arriving at the input.

$$IR_i, i = 1..n$$

For each requirement IR_i, there can be m_i MD interpretations.

$$m_i = |\mathbb{MDI}_{IR_i}|$$

After the first requirement (IR_1) arrives, its MDIs are inserted into the space of alternative solutions $\mathbb{S}$. The size of $\mathbb{S}$ will correspond to the number of MDIs of IR_1, i.e., m_1.

$$|\mathbb{S}| = m_1$$

Next, when we integrate the second requirement (IR_2) into the existing space of alternative solutions $\mathbb{S}$, the size of $\mathbb{S}$ will correspond to the product of m_1 for IR_1 and m_2 for IR_2 multiplied by the coefficient of alternative solutions (i.e., γ_1). Note that when comparing a single *MDI* with one star *SS* there is at least one solution (i.e., $\gamma_i \geq 1$). However, there may be additional solutions which result from having different alternatives to integrate an *MDI* with *SS*. For example, in dimension matching algorithm in Step DM:3, if we do not find the direct matching between current levels we explore both the hierarchy of the first and the second dimension, which in turn can create more then one valid solution.

$$|\mathbb{S}| = m_1 \cdot m_2 \cdot \gamma_1$$

Moreover, the algorithm will perform $m_1 \cdot m_2$ comparisons of the MDIs of incoming requirement and the stars of the existing space of solutions.

Considering the above, after integrating requirement IR_n into the existing space of alternative solutions $\mathbb{S}$, the size of $\mathbb{S}$ will be the following.

$$|\mathbb{S}| = (((m_1 \cdot m_2 \cdot \gamma_1) \cdot m_3 \cdot \gamma_2) \cdot \ldots \cdot m_{n-1} \cdot \gamma_{n-2}) \cdot m_n \cdot \gamma_{n-1}$$

Let us consider that the average number of MDIs per requirement is m (i.e., $\forall m_i \in m_1..m_n, m_i \approx m$).

$$|\mathbb{S}| = (((m \cdot m \cdot \gamma_1) \cdot m \cdot \gamma_2) \cdot \ldots \cdot m \cdot \gamma_{n-2}) \cdot m \cdot \gamma_{n-1}$$

Additionally, if we assume that each comparison produces in average γalternative solutions (i.e., $\forall \gamma_j \in \gamma_1..\gamma_{n-1}$, $\gamma_j \approx \gamma$). Then, we can approximate the size of $\mathbb{S}$ as follows.

$$|\mathbb{S}| = m^n \cdot \gamma^{n-2} \cdot \gamma = m^n \cdot \gamma^{n-1}$$

Here, we can see that the total number of comparisons among the MDIs of incoming requirements and the stars of the existing space of solutions for n requirements will be $m^n \cdot \gamma^{n-2}$.

If we further assume a realistic scenario where the number of incoming requirements is big enough, we can then infer that $n \gg m$ and $n \gg \gamma$. Taking the above analysis into account, we can conclude that theoretical computation complexity for the problem of requirement-driven DW schema design is exponential (i.e., $O(2^n)$).

As discussed in the paper, and more specifically in Sections 2.4 and 3, we introduce a cost model that evaluates some quality factors as a parameter of *ORE*. In our example scenario, we use the structural complexity of the output MD schema as a quality factor. We further use this cost model for introducing heuristics to tackle the inherent complexity

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

of the problem at hand. To this end, the cost-based space of alternative solutions, explained in detail in Section 3, is created and pruned after each integration iteration to maintain only the top-N solutions.

Assuming that N is the limiting size of the space of alternative solutions $\mathbb{S}$, we revisit the analysis of the computational complexity of *ORE*.

With each new requirement IR_i, we need to perform the maximum of $m_i \cdot N$ comparisons between the MDIs of IR_i and the stars of $\mathbb{S}$. For n incoming requirements we perform the total of $m_1 \cdot N + .. + m_n \cdot N$ comparisons. If we again consider that m is the average number of MDIs per requirement, it is not hard to see that the total number of comparisons for n requirements is $m \cdot N \cdot n$.

As before, if we assume n being big enough, i.e., $n \gg m$, we can conclude that by introducing the cost-based heuristics, the computational complexity of *ORE* is guaranteed to be linear (i.e., $O(n)$).

Such linear complexity, as we will experimentally show in the next section, can be further parametrized with the size of the space of alternative solutions (i.e., N). Moreover, as it will be also empirically shown, for a manageable size of N, *ORE*, in most of the cases does not miss the optimal solution w.r.t. chosen quality objectives.

6. Evaluation

In this section, we first describe a prototype system which implements ORE's functionalities. Next, we validate the correctness of the output MD schemata, obtained for the TPC-H benchmark example used in this paper, by comparing these to the ones manually derived in the Star Schema Benchmark (SSB) [20]. Finally, we report on our experimental findings from both scrutinizing the *ORE*'s prototype and performing an empirical case study with a group of participants. On the one hand, we scrutinize *ORE* to validate our theoretical conclusions w.r.t.: (1) the algorithm's complexity (in terms of the execution time), (2) quality of the results (in terms of the chosen quality factors), (3) the algorithm's characteristics (in terms of the number of performed integration operations), and (4) scalability (in terms of the growing amount of requirements). On the other hand, we evaluate the manual effort of the real end-users needed for the MD schema design w.r.t.: (5) elapsed time, (6) quality and accuracy of the results, and (7) the emotional response (by means of the user's confidence in the provided solutions), in order to show the inherent complexity of the design tasks when performed manually.

6.1. Prototype

As a proof of concept, we have built a prototype implementation of *ORE*. *ORE* expects two kinds of inputs: information requirements expressed as MDIs and an OWL ontology representing the sources. In order to integrate the end-to-end solution of our research work (more details in Section 8) and to automate our testing exercises, we linked *ORE* to *GEM*. *GEM* is our previous solution to produce MDIs from requirements, [10] (see Fig. 12). Moreover, the output generated by *GEM* is guaranteed to be sound and complete.

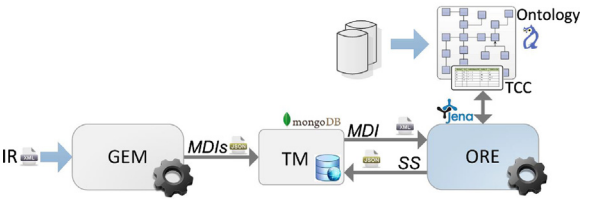

Fig. 12. Prototype setup.

These two modules (i.e., *GEM* and *ORE*) are communicating through the TM module (Section 3). We have chosen *MongoDB* for implementing a storage repository for TM. We justify our choice with the fact that handling semistructured, document-oriented data (e.g., information requirements, MDIs, Star schema (SS)) is more efficient with NOSQL databases, like *MongoDB*. Moreover, the consistency tradeoffs (i.e., eventual consistency) that boost the scalability, availability and latency of a MongoDB, fit the TM's needs for data storage. Later in this section we show some experimental findings that confirm such design choice.

When a new requirement arrives, it is processed by *GEM*, which produces a set of MDIs describing the MD data cube representing such requirement and stores them into the TM repository. Next, the produced MDIs of a single requirement are read from TM, and serialized in a proprietary XML format understood by *ORE*. *ORE* then processes these MDIs as explained throughout the paper and produces the alternative MD schemata, each one in terms of a set of stars (i.e., $\mathbb{SS}_i$), and stores them inside the *TM*.

Fig. 12 shows the architecture of our prototype. This architecture is modular in that another module could be chosen to replace *GEM* if necessary, as far as the output MDIs are sound and complete (see Section 2.2) and they are expressed in *ORE*'s proprietary XML format.

For describing the sources as an OWL ontology we followed the method proposed in [16], which largely automates the process for well-formed sources. *ORE* communicates with the ontology describing the sources by means of Jena API [21]. Depending on the complexity of such ontology, the requests for finding the relationship between the ontology concepts may be costly (e.g., [15]). Since such requests are frequent, in our implementation we used a component called *Transitive Closure Cache* (*TCC*) to facilitate the probing of the ontology. *TCC* receives requests from *ORE* for discovering the relationship between two concepts, e.g., (A,B). If no entry is found, it means that no previous iteration asked for the relationships of A. At this point, *TCC* accesses the ontology and looks for the requested relationship, but it also further explores the ontology to find the transitive closure for A through "1-1", "1- ∗" or "∗ -1" relationships and loads the corresponding entries to its structures. If a following request relating A, say (A,C), is posed, it will be answered from *TCC*. *TCC* works with a limited size of the internal memory and the overwriting is done following the *least recently used* cache algorithm.

6.2. Output validation

Following our running example scenario (i.e., the TPC-H benchmark), we performed a set of experiments

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

to validate the reliability of *ORE* by examining the correctness of its resulting MD schemata. TPC-H is an ad hoc decision support benchmark, hence it provides a convenient source for analyzing data from different perspectives. Some efforts have been invested in adapting the schema of the TPC-H benchmark for MD purposes. In [20], the authors present the Star Schema Benchmark (SSB). They manually design an MD schema based on the TPC-H benchmark by applying traditional optimization techniques (e.g., denormalization) to the classical TPC-H schema. Here, we list the manual modifications of the TPC-H schema that are proposed in [20] and describe how each of these design choices is automatically supported in *ORE*.

- *Combine the TPC-H* `Lineitem` *and* `Orders` *tables*: The fact matching stage of *ORE* first merges the facts of different requirements that are placed in the same MD space. Furthermore, in the last stage (i.e., integration), *ORE* denormalizes the schema and combines the adjacent MD concepts into a single fact or dimension.
- *Drop the* `Partsupplier` *table*: As explained in [20], the `Partsupplier` table is removed from the design as it has a different temporal granularity from the `Lineitem` and `Orders`. The authors further discuss that `Partsupplier` can only be treated as a separate fact table belonging to a different data mart. As a matter of fact, when in the fact matching stage, *ORE* does not find any existing fact whose MD space matches a fact from the new requirement, it creates a new star inside the existing MD schema with the new coming fact.
- *Drop the* `comment` *attribute from* `Lineitem`: We discuss in Section 2.2 that *ORE* only accepts MDIs that satisfy the MD integrity constraints. As explained in Section 2.2, one of the requirements to guarantee this is a correct data summarization which is further guaranteed among others, by the condition of *Compatibility*. This condition indeed ensures that the type of a measure being aggregated and the aggregation function are compatible, which guarantees that the textual attributes, like `l_comment`, will be left out from the fact table design.
- *Drop the tables* `Nation` *and* `Region` *outboard to the dimensions* `Customer` *and* `Supplier`: In [20], the authors drop the tables `Nation` and `Region` and add this information as attributes to the `Customer` and `Supplier` tables (i.e., `address`). Such design choice is widely supported in *ORE*'s last stage (i.e., integration), where adjacent levels are combined and the dimension is denormalized and collapsed into one table.
- *Adding the* `Date` *dimension*: To adapt the schema to the standard DW design principles, the authors in [20] propose to complement the schema with the new `Date` dimension which is very common when analyzing the factual data such as `sales`. In *ORE*, after the fact and dimension matching stages, the new MD schema can be additionally tuned with the analytically interesting concepts in the stage of complementing the MD design. In this stage, the domain ontology is searched and the new valid MD concepts are proposed to the user. Likewise, the new dimension (e.g., `Date`) can be added to the MD design.

As an empirical proof, we ran *ORE* for a set of information requirements adapted from the queries that are provided by the TPC-H benchmark and we automatically obtained the same MD schema as SBB. The only exception is the `Partsupplier` fact which *ORE*, due to different granularity from other facts (e.g., `Lineitem`), separates in a different star.

6.3. Experimental setup

The experimental setup involves a system called LEARN-SQL [22], which has been used at UPC-BarcelonaTech since 2003 to assess assignments related to exercises on the SQL language. The system is implemented using the open-source, learning management system, i.e., Moodle. The data gathered so far involve 2550 students in five different subjects and concerns more than 600 different SQL items (assignments). The students issued more than 150.000 submissions, which created a significant data source to be analyzed for gaining an insight into the students' performance regarding different aspects of the subjects. Three main data sources are considered for the analysis in the LEARN-SQL system: (1) the operational DB deployed in PostgreSQL, (2) the students' data in XML, and (3) the evaluation results (per semester) in spreadsheet format. As a goal, a target MD data store should be provided to conveniently support the demands from different analytical requirements posed by the system users (e.g., lecturers).

For evaluating our approach, we conducted a set of experiments using the prototype being implemented (see Fig. 12), with the aim to scrutinize *ORE* for measuring its performances (i.e., computational complexity, results quality, and different characteristics of the algorithm) and scalability (in terms of the number of requirements). As in the rest of the paper, here we also use the structural complexity as the example quality factor for evaluating the cost of the output MD schemata. As an additional validation step for demonstrating the need for automating the MD schema design, we performed a set of empirical tests to measure the manual efforts of the real users when manually designing an MD schema from information requirements (in terms of the elapsed time, the quality and accuracy of provided solutions, and the emotional response of the users).

6.4. Scrutinizing ORE

As previously discussed in this paper (see Section 5.3), the MD schema design from information requirements is a computationally complex problem due to the exponential growth of the space of possible solutions. However, as it can be seen in Fig. 13, not all these solutions have the same cost when it comes to structural complexity of an MD schema. Therefore, we introduced a cost-based approach based on our cost model to prune the space of alternative solutions. Taking this into account, we used the previously described *ORE*'s prototype and performed a series of experiments. These experiments aimed at validating the linear complexity of our cost-based solution (see Section 5.3) and generally the practical feasibility of our approach. We collected different indicators about *ORE*'s performance and scalability, and at the same time, for our example scenario we aimed to

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

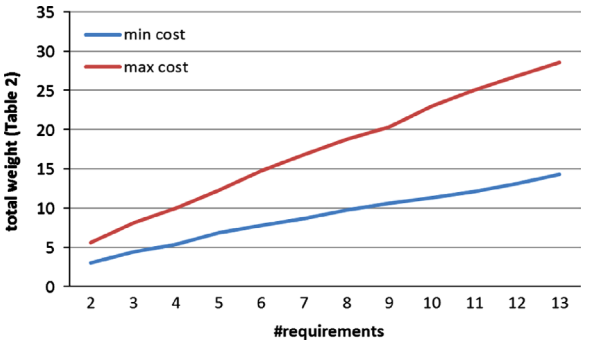

Fig. 13. Cost differences.

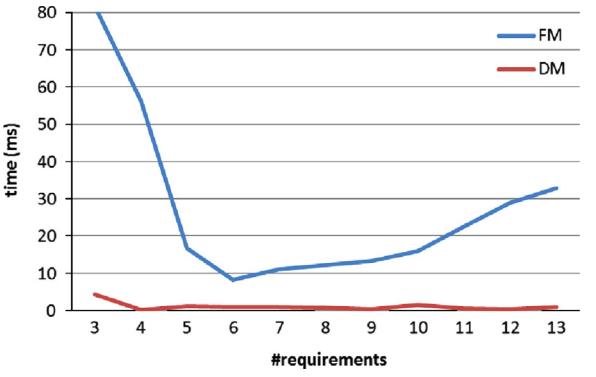

Fig. 15. Time per stage.

Fig. 14. Time comparison.

show that *ORE* meets the chosen quality objectives, i.e., the resulting MD schema is with minimal structural complexity.

Inputs. As we explain shortly, we had a case study conducted with the help of a number of participants, real users of the LEARN-SQL system. During this process, one group of participants (i.e., DB lecturers) analyzed the domain ontology that captures the semantics of data sources in the LEARN-SQL system, and provided us with a set of thirteen requirements. Taking a look at them, the users most often wanted to analyze the success of students focusing on different indicators (e.g., *final marks*, *outcomes of single assignments*, *results of the experiments executions over students' solutions*, etc.) and from different perspectives (e.g., *candidates*, *semesters*, *kind of assignment items*, *subjects*, etc.). These requirements represent a real world case as they came from real users showing real demands on a real system. What these requirements also showed is that different users may be interested in performing the same or similar analysis which is a common case in real business environments. We use this set of thirteen requirements as input dataset for performing the experiments over *ORE*.

Experimental Methodology. Linking *ORE* to *GEM* allowed us to provide the collected requirements as inputs in our setting (see Fig. 12). From each individual requirement, *GEM* produced the corresponding set of *MDIs*, which then fed *ORE* together with the domain OWL ontology we created for the experiments with the users. We then ran *ORE* for different permutations of the input set of requirements. In each permutation, we simulated iterative arrivals of new requirements to the system i.e., starting from a single requirement a new requirement was considered in each iteration to be incorporated into the current MD schema design. Through a series of experiments that we explain shorty, we collected different indicators for each permutation and iterations (e.g., overall time per iteration, time spent on fact matching, time spent on dimension matching, overall time per complete permutation, #matching facts, #matching dimensions).

Experiments. Next, we discuss different experiments we performed over *ORE* in more detail and report our findings.

Time Considering that in general the requirement-driven DW design problem is exponential, which we theoretically validated in Section 5.3, we performed a set of experiments following the above methodology and with varying input loads and constraints. The results are shown in Fig. 14. As assumed, dealing with the complete space of solutions is expensive. After only ten requirements (considering all the *MDIs* of those requirements) the time to iteratively integrate each requirement starts rapidly to grow and it later bursts for additional requirements (see dotted part of the purple line, i.e., *w/o threshold*, in Fig. 14). However, as previously discussed, our approach is cost-based (in this experiment, we used as a quality factor the structural complexity of an output schema) and we only keep the top-N best solutions after integrating a new requirement. In the experiments, we used different values for N (i.e., 25, 50, and 100) to analyze how N affects the execution time of *ORE*. The linear complexity of our cost-based approach discussed in Section 5.3, we empirically confirm here after introducing the cost-based heuristics (i.e., keeping the top-N solutions). The additional tests with different thresholds (i.e., N) showed us that after new requirements come, the time tends to stay in a certain range depending on the number of top solutions kept (i.e., the value of N). Additionally, Fig. 14 (dashed line), shows that peaks may appear since the size of inputs (i.e., *#MDIs* per requirement) is not limited. However, our cost-based pruning keeps the problem manageable even in the worst case, regardless the number of requirements already dealt with.

Another interesting observation is the higher latency that appears at the beginning, when integrating the first few requirements (see Fig. 14). This comes from the fact that at the beginning we make a direct access to the ontology for checking different relationships among concepts, which tends to be costly. As we discuss in Section 6.1, in our prototype we

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

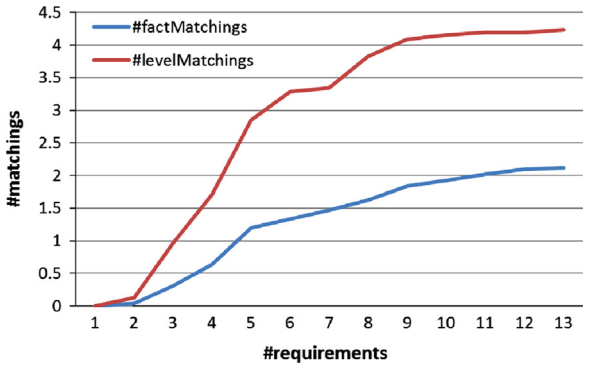

Fig. 16. Matchings found.

introduce a component called Transitive Closure Cache (TCC), where we cache relationships among ontology concepts, so over time we avoid accessing the ontology directly as the relationships are mostly found in TCC. Notice in Fig. 14 that even with the expensive initial accesses to the ontology the time needed for integrating new requirement stays within a manageable rate (200–240 ms).

When measuring the times of different stages in *ORE*, we noticed that the average ratio of time spent on the fact matching stage (*FM*) appears to be significantly higher as opposed to the other stages. As an example, the average ratio of time between fact and dimension matching stages (*FM* and *DM*) is shown in Fig. 15. Such behavior showed us that the real problem of integrating MD designs is indeed finding the matches of MD spaces of their factual concepts which is done during the FM algorithm. Moreover, matching the MD spaces of facts through all their dimensions tends to be more complex than finding the relation between the dimensions of the already matched facts.

Results quality: Within the same set of experiments we also analyzed the cost of the output MD schemas that *ORE* produces w.r.t. the quality objectives we chose (i.e., minimal structural complexity). By testing different thresholds (i.e., values for N) we noticed that the optimal solution was always in the top-N (independent of the size of N) for the current setting. Clearly, as it usually happens with pruning techniques, at the moment there is no formal guarantee that pruning will always consider the optimal solution. At the same time, we also analyzed the alternative solutions with the highest cost. As one may notice in Fig. 13, there is a significant and increasing difference of the costs between the optimal solution (i.e., min cost) that *ORE* finds and the worst solution (i.e., max cost) that may result from some other approach (e.g., manual design). Such observations additionally demonstrated the importance of automating the DW design process with the guarantee of meeting predefined quality objectives.

Algorithm's characteristics: We also studied the behavior of characteristics of ORE's algorithms. Fig. 16 shows how the number of matchings of different concepts (factual and dimensional) is affected by the size of the problem. The number of matchings represents the average matchings found between *MDIs* of incoming requirement (1–13) and each of the alternative schemas at the output. Starting from scratch, *ORE* first, as it finds the empty *TM*, inserts the facts and dimensions and creates first designs at the output. Then, it exhaustively tries to match the facts and also their corresponding dimensions, as explained in Section 4. However, as the incrementally integrated design matures, there are lesser novel opportunities for matches, i.e., MD design becomes more complete. In Fig. 16, one may notice that the number of the facts matched is proportionally lower then the number of levels, as we often have the situation of having a single fact and several of its dimensions. On the other hand, as we have already shown in Fig. 15 time to perform the fact matching is still significantly higher from the time for dimension matching.

Scalability: As we showed with the previous set of experiments, *ORE* is able to handle the growing amount of requirements, by using the introduced cost-based heuristics (i.e., top-N solutions). We first theoretically (in Section 5) and then practically (in this section) proved that the complexity of the approach by using these heuristics is linear and manageable regardless of the number of incoming requirements. Additionally, for the sake of analyzing the scalability and maintainability of the *TM* structure (see Section 3) for the growing amount of information requirements, we performed a set of tests where we measured the times for accessing the *TM* on MongoDB for reading and writing of different elements (e.g., *MDIs*). These tests showed us that the accessing times do not depend on the current size of the *TM* but mostly on the size of individual elements that are read or stored. As an example, reading MDIs from TM with the average size of five concepts (facts or dimensions) has the average latency of 5.2 ms and it goes to maximum of 9ms for MDIs larger than ten concepts, independent of the current size of *TM*.

Summary: This set of experiments demonstrated the feasibility of our semi-automatic approach for MD schema design. Furthermore, our cost-based heuristics (see Section 5.3) proved to maintain the linear complexity of the approach which makes it scalable regardless of the number of requirements and yet showed that *ORE* tends to generate the optimal solution. The main recommendation for the future users of the system would be to carefully choose the size of the space of alternative solutions which is used for the cost-based pruning (top-N). If N is too small (e.g., lower than 15), *ORE* does not always provide the optimal solution at the output. However, if N is too high (e.g., larger than 500), more solutions will be produced (including the optimal), but the performances will drastically drop. In fact, in our experimental scenario we showed that even with N in the rage [25,100] which is rather manageable, *ORE* always provides the optimal solutions at the output.

6.5. The LEARN-SQL case study

After evaluating the performances and observing different characteristics of *ORE*, we additionally conducted a series of empirical tests with the goal to measure the emotional aspects and the amount of human efforts needed for manually designing MD schema from requirements, as well as for incorporating new requirements into an existing schema.

Experimental methodology: These tests resembled the common practice of a DW design project and aimed at capturing all the stages of such process, from the collection of information requirements to the final design of the MD schema that satisfies

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

them. We considered the following variables: (a) *independent*, i.e., the participants and the information requirements; (b) *controlled*, i.e., the object of the experiment (the domain ontology capturing data sources' semantics of the system); and (c) *dependent*, the observed parameters. We divide the parameters observed during the study in three groups: (1) *Time* - How long did it take for participants to complete the task? (2) *Accuracy and Quality* - How many mistakes participants made? How far their solutions are from the ideal case? (3) *Emotional response* - How confident the participants felt about the completed tasks? The analysis was both quantitative and qualitative.

We considered two groups of users with different backgrounds.

1. Six DW and conceptual modeling experts – researchers and lecturers from the Department of Service and Information System Engineering (ESSI) at UPC BarcelonaTech.
2. Six graduate students following the DW course at the Facultat d'Informatica de Barcelona (FIB).

The tests were organized in three separate rounds as follows.

We first had a round where the seven real end-users of the LEARN-SQL system familiarized themselves with the vocabulary used to build the domain ontology. Next, these seven participants were asked to provide information requirements for analyzing different aspects of the system. Participants used the natural language template introduced in Section 2.2 as guidelines for providing the requirements. For example, the following is a requirement demanded by one of the lecturers: "*Analyze failed experiments, from the point of view of the day of the week, student's group and difficulty of assignment, where the final mark is lower than 7.*" Each participant came up with maximum two requirements totalling in thirteen requirements for the study.

In the second round, the live session is organized and the participants of both groups are introduced to the problem under the study. As some of the participants, especially among the students were not DW experts, this session provided some insights about the MD design principles, MD integrity constraints and the structural complexity of MD schema design as an example quality factor used throughout this paper. At the end of this session, several participants posed questions and after the discussion the participants were positive about being familiar with the problem under the study.

The final round, organized also as a live session, included two assignments of the manual DW design that the participants were solving. In the final round, both groups of participants (lecturers and students) were assessed.

Assignment 1: One of the requirements produced in the first round was handed out. Using the ontology graphical representation describing the sources (see Fig. A1), the participants were asked to provide the MD interpretation of the ontology subset needed to answer the provided requirement (i.e., to identify the concepts that are needed to answer the requirement and determine a valid MD role(s) for such concepts -i.e., factual and/or dimensional-). To answer this assignment, the participants were asked to fill in a grid as the one presented in Appendix A (see Fig. A2), one per each valid MDI, and to record the start and the end time of the work. The time limit for this task was 10 min.

After the participants provided their solutions, the actual correct solutions for the given assignment were presented to the user (see Fig. A3). Additionally, the reasons why several MDIs resulted from a single requirement are explained to the participants (i.e., existence of the intermediate concepts).

Assignment 2: In this assignment, we introduced a referent MD schema (see Fig. A4) meeting a set of previous requirements. In this assignment, participants were asked to integrate the requirement considered in the previous assignment with the referent MD schema. Intuitively, they are asked to look for an MDI (out of the four produced in the previous assignment) such that it achieves the best integration with the referent schema according to the set of quality objectives, i.e., we asked them to minimize the structural complexity of the resulting MD schema, as discussed in Section 2.4.2. As a result, the participants were supposed to produce the final unified MD schema that additionally satisfies the new requirement, respects the MD integrity constraints and meets the quality objectives (i.e., minimal structural complexity of a MD schema). To answer this assignment the participants were asked to sketch the new MD schema on paper and record the starting and ending time. The time limit for this task was 15 min.

Results: Next, we report on our findings from the performed tests, in two parts. The first part aims at identifying the background of the users and the efforts invested in solving the given assignments. In the second part, we evaluate the solutions provided in the given assignments and investigate the possible reasons why the participants failed to provide the correct solutions.

Background analysis: The first group of six lecturers are database and conceptual modeling experts. However, four users were non experts when it comes to DW and ETL technical issues. This was an excellent opportunity to see how our methods could work with business users, who are familiar with the domain and the requirements, but sometimes lack the technical expertise to perform some preprocessing steps that would facilitate and advance their business analysis. Also, only one of them had seen LEARN-SQL internals before, which in a sense resembles practice where not all analysts are initially familiar with the system under the analysis. The second group of six users are graduate students following the DW course during their master studies and thus not well experienced users. However, as being part of the DW course, they showed a strong commitment towards the activity and wanted to perform well. The first and second rounds were used to smooth out these differences among our users and a second interviewing round afterwards showed that all twelve users were feeling comfortable with the technologies used in this experiment.

An expected observation among these two groups is the time/effort needed to finalize the given tasks. In average the students needed around twice as much time in comparison to lecturers (experts) (see Fig. 17).

Design evaluation. Next, we evaluated the solutions resulting from the given assignments (see Fig. 18). We focus here on the results of Assignment 2 (A2), which in a way resembles the work that *ORE* facilitates through its algorithms. A2 required

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

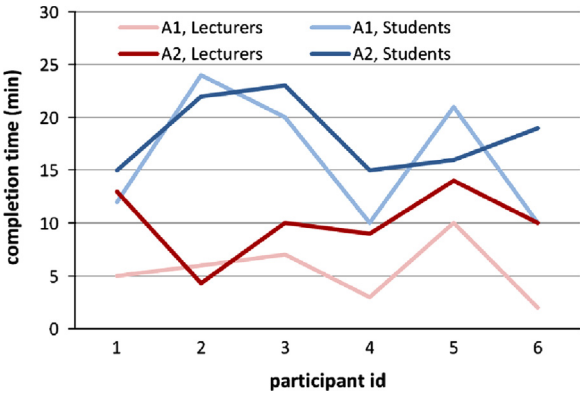

Fig. 17. Completion time for correct designs.

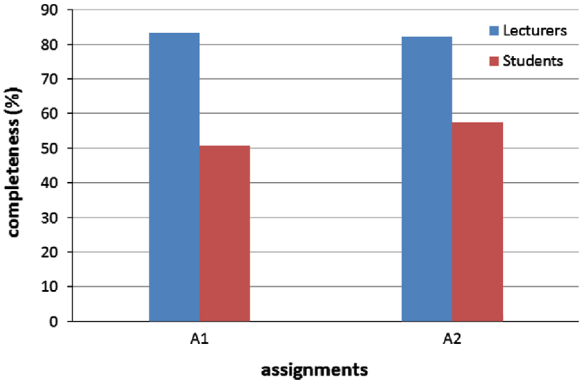

Fig. 18. Completeness of solutions with fixed time.

that the participants looked for potential fact matching. In principle, given that the `valid_response_processing` fact (in the new requirement) is a subclass of `response_processing` (which appears in the reference MD schema) users tended to integrate the new requirement through these facts. However, after a detailed analysis, they were supposed to recognize at least one more promising alternative: merging `evaluation` (in the reference MD schema) with the `valid_response_processing` (in the new requirement) as in the top-right MDI in Fig. A3. These two facts share the same MD space (`year_month` × `candidate`) and thus, the facts could be properly combined.

Nevertheless, these were not the only two valid solutions. Indeed, the results provided for this assignment showed us that even a small example like this can produce many valid solutions, from which we should find the one with the lowest structural complexity. Fig. 18 compares the level of completeness for lecturers and students. Note that among all participants only one lecturer provided solutions that were above 90% close to the ideal solution. We also observed that the average completeness percentage for the lecturers was around 83% and for the students it was lower – around 56%.

Along with the previously discussed difference in the amount of effort (i.e., elapsed time), the noticeable difference of the two groups (lecturers and students) in the quality and accuracy of the provided solutions showed us that the DW design problem is indeed complex and it requires high level of experience and expertise to achieve satisfactory results.

The most common error made by the participants was the attempt to match `valid_response_processing` with `response_process` by means of the ontology taxonomy they belong to. Such a matching may result in a valid solution that answers the new requirement, but as previously discussed, it is not the one with the lowest overall cost, in terms of structural complexity as an example quality factor, as it introduces new concepts to the design. Furthermore, intuitively, this solution should be disregarded because we are merging two facts at different granularities when there is another fact at the same granularity (i.e., `evaluation`).

Summary: These empirical tests aimed at simulating the environment where the users are supposed to provide the design of MD schema starting from the input information requirements. We started with a moderately small problem size and introduced time limits to simulate the complexity of larger DW design problem (the complexity of a task is related to the time given to complete it). As observed from the results, even for a small scenarios, the number of possible design solutions is fairly high. Moreover, the best solutions do not always appear to be the most obvious ones. Additionally, after filling a simple questionnaire at the end, most of the participants (including DB experts) were not confident about having found the best solution. After combining these findings with the results we obtained from scrutinizing *ORE*'s prototype (i.e., low overall latency and high quality of generated results), we can conclude that the automation of the DW design is a necessity for dynamic business environments.

7. Related work

Following a monolithic approach for building a DW is considered problematic and thus, manual guidelines were given to overcome this issue (e.g., DW Bus Architecture [1]). Some recent works (e.g., [23]) also study how the modern software engineering methodologies can be applied in the DW context. The authors in [23] define a set of methodological principles that should be followed during the design process and inspect their impact on different quality factors (e.g., reliability, robustness, etc.). Apart from traditional DW designing approaches (e.g., [24,25,9]), various works have studied the problem of adjusting the DW systems to changes of the business environments. For dealing with such an issue, different directions have been followed.

DW evolution: The early approaches that fall into this category (e.g., [26–28] consider the DW as a set of materialized views over the source relations and propose the algorithms for either maintaining the existing set of views (i.e., *view maintenance*, [26]) or selecting new views to be materialized (i.e., *view selection*, [27,28]), in order to answer new queries. Since a pure relational approach has been followed, these works use the equivalence transformation rules to enhance the integration of new queries into the existing view set. That makes these approaches not easily applicable to the current heterogeneous environments. In fact, as it has become clear in the last decade, DW are more complex than just a set of materialized views, especially when the ETL flows come into play. Additionally, these approaches mainly consider two traditionally correlated costs in DW design: (1) view maintenance cost, and (2) query evaluation cost. *ORE*, regarding new

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

business oriented environments, complements these approaches by taking into account other quality factors that additionally minimize the end-user's efforts (e.g., the structural complexity and understandability of the MD schema). Other DW evolution approaches (e.g., [29,30]) maintain the up-to-dateness of the existing DW schemata in the event of a change by proposing a set of evolution operators (e.g., for addition/deletion of dimensions/levels/facts or their instances). Some (e.g., [31,32]) additionally study the influence of different evolution changes on the quality factors of the DW (e.g., consistency and completeness). Similar problems have been studied for ETL flows too (e.g., [33]). One contribution of these works is the formal definition of specific alteration operators, which can be applied when an evolution change occurs. However, these works do not study how the information requirements actually affect such evolution changes. In this sense, our work complements them and starting from a given set of information requirements, it aims to automatically derive the changes of the current schema necessary to be applied for satisfying these new requirements.

Schema versioning: Beside dealing with the changes by upgrading the existing schema, schema versioning approaches have also focused on keeping the trace of these changes by separately maintaining different versions of the schema (e.g., [34–36]). Some of them (e.g., [34]) in addition to the versions resulting from real world changes, also store and maintain the alternative versions which can simulate various operational/business scenarios and propose new analytical perspectives. Specifically, [36] deals with the problem of cross-version querying and introduces the concept of augmented schema, which keeps track of change actions to enable answering the queries spanning the validity of different versions. *ORE* also systematically keeps traces of the changes occurred so far, by using the TM structure (see Section 3). Contrarily, by maintaining TM, *ORE* aims at reproducing the final solution and/or potentially choosing an alternative integration option, but at any moment, the resulting MD schema must answer all the current requirements.

Incremental DW design and DW schema integration: There are works that have studied the problem of incremental DW design (e.g., [37]). The authors here propose the approach for incremental design of DW by deriving new facts and dimensions from the existing ones by applying the set of deriving operations (i.e., dimension and measure transformation algebras). However, they do not specifically discuss the influence of information requirements nor their automatic integration into the DW design. On the schema integration side, there are works that use ontologies, to bridge the semantic gap among heterogeneous data (e.g., [16]). To deal with the integration of heterogeneous DW schemas, [38] proposes two approaches: loosely and tightly coupled. But, this work assumes that a structural matching among schemas exists and proposes the d-chase procedure (inspired by the chase procedure) for the chase of dimensional instances to ensure the consistency of the given matching.

Overall, the importance of information requirements into the DW design process has been generally overlooked. One recent work [39], proposes a method to deal with the maintenance of a DW by identifying a set of semantic mappings between data sources and user requirements. In a sense, we find the motivation behind this work complementary to ours. However, the process of identifying such mappings as well as integrating them into the current design requires an extensive manual effort. In our work, we go a step further and automate a large part of this process as we discussed in the previous sections. Another exception is the work in [40] that starts from OLAP requirements expressed in sheet-style tables and later translates them to single star schemas. However, the integration proposed is based solely on union operations applied to the facts and dimension hierarchies. Moreover, this work does not consider the heterogeneity and complex relations that may exist in DW environments.

8. Conclusions and future work

In this paper, we have presented *ORE*, a method to iteratively design the MD schema of a data warehouse from requirements. To the best of our knowledge, there is no other similar work dealing with such integration, which differs from ontology or generic integration research papers in that the MD integrity constraints are preserved and considered quality objectives are met.

This work represents an important step for fulfilling our final goal, i.e., to deliver a system for providing an end-to-end, requirement-driven solution for DW design problem. Our work builds on top of the *GEM* system [10]. *GEM* starts with the information requirements at hand and generates for each requirement separately a respective MD design along with the ETL operations that build the data flow. In another work [19], we present *CoAl*, an approach to deal with the problem of integrating the ETL processes from single information requirements.

In this paper, we present our approach to incrementally design the MD schema from individual requirements. Starting from single requirements, we obtain a set of MD interpretations of the sources to answer such requirement (e.g., *GEM*). Incrementally, we build a unified MD schema satisfying the current set of requirements. At the same time, the details about the previous integration steps are traced by means of metadata to allow broader integration possibilities when new requirements arrive.

Clearly, both the MD schema and the ETL constructs are highly correlated. Our overall work is the first that considers them together. We plan to invest on that and that opens several possible future directions. As one example, it would be interesting to exploit the interdependence between *ORE* and *CoAl* through which each of them benefit from the relevant information inferred by the other approach. For instance, the aggregation and normalization levels of the produced schema could be considered, since this would effect the way the appropriate ETL process is tailored (i.e., trade-offs between materialized views and OLAP querying). Similarly, checkpointing or bottlenecks detected at the ETL level may cause some changes at the MD schema for the sake of performance.

Acknowledgments

This work has been partly supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2011-24747.

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

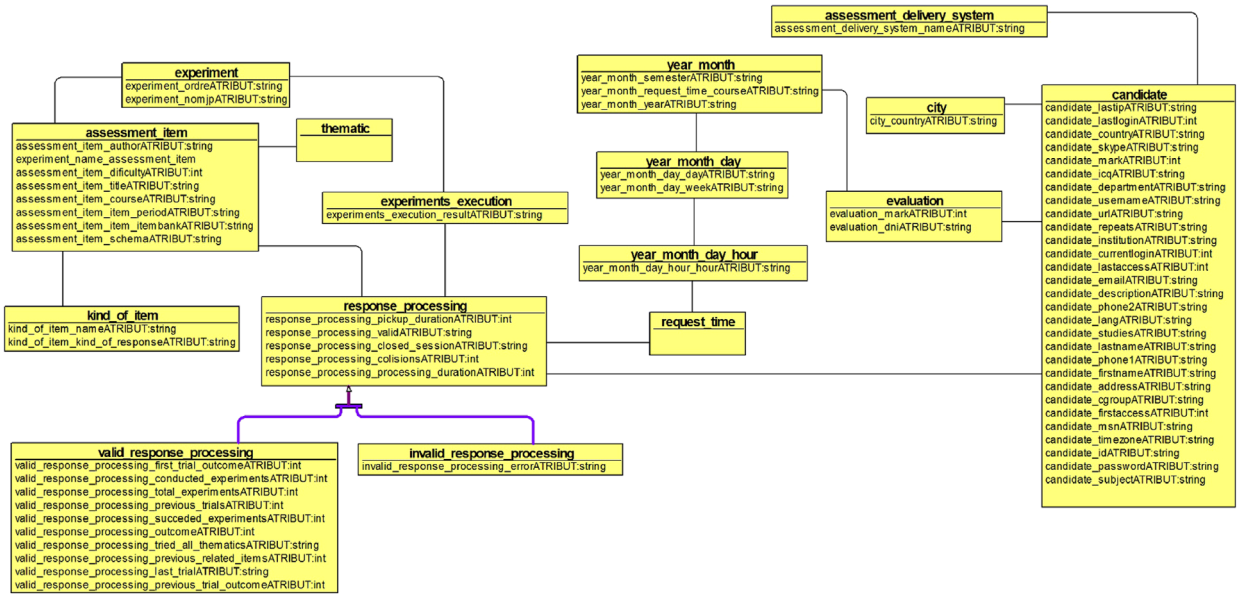

Fig. A1. LEARN-SQL ontology.

Concept / MD Context (F/D)	Evaluation	response_ processing	candidate	...	assessment_ item
Fact1	X				X
Dim1			X		
Dim2					X
...					

Fig. A2. Table for identifying *MDIs* for an *IR*.

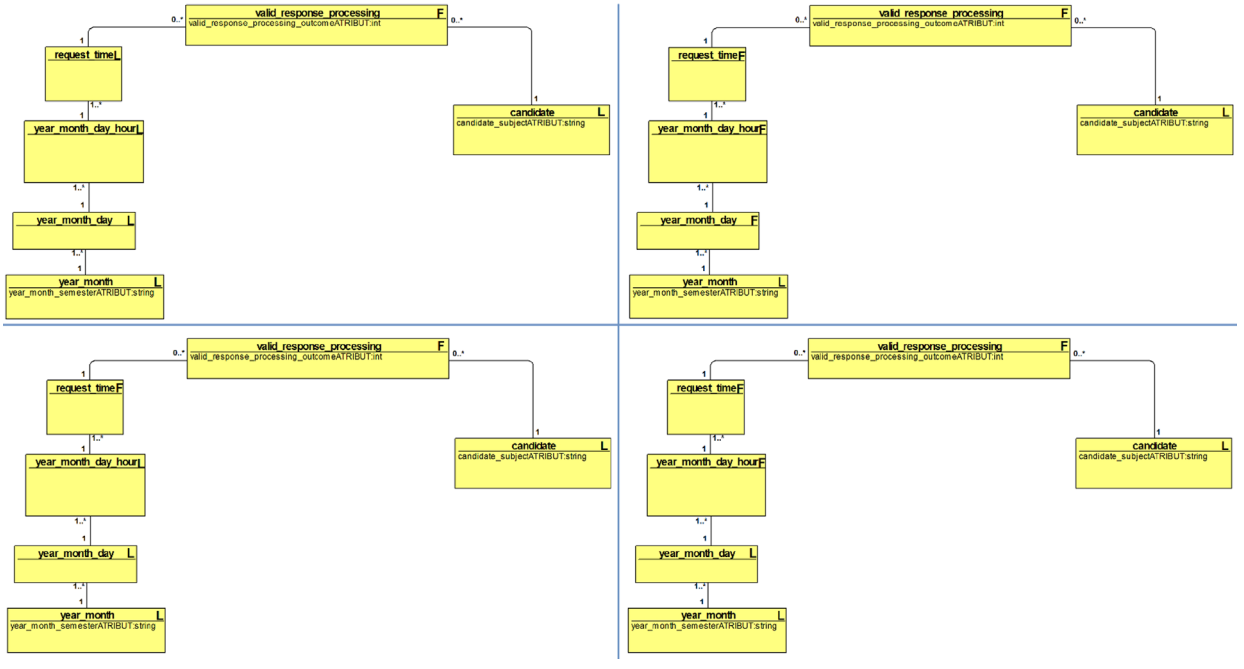

Fig. A3. *MDIs* resulted from *IR*.

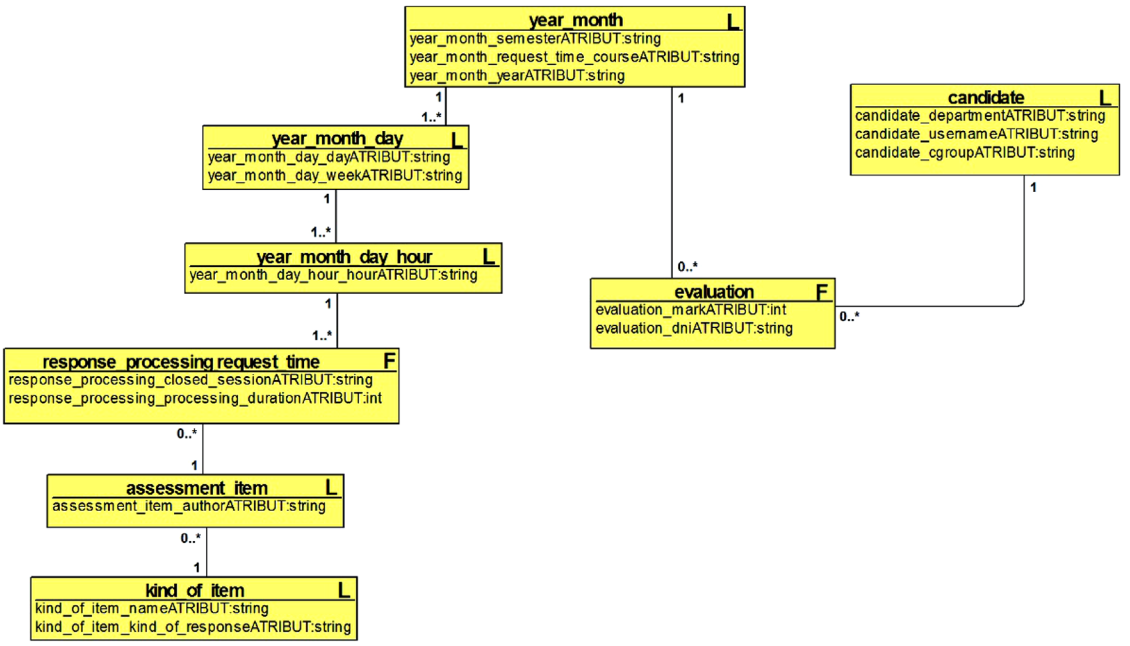

Fig. A4. Reference MD schema.

Appendix A. Materials used in the LEARN-SQL case study

A.1. Assignment 1

Fig. A1 shows the diagrammatic representation of the LEARN-SQL ontology, used in our case study.

Information requirement: The users in our case study were provided with the following information requirement (*IR*): *"I want to analyze the average* **valid processing outcome** *for each* **semester** *and the* **candidate** *where the* **candidate's subject** *is DABD."*

Fig. A2 shows the table that participants were asked to complete in order to identify the MDIs of the given requirement (*IR*).

Fig. A3 shows the MD interpretations (*MDI*) resulted from the above information requirement.

A.2. Assignment 2

For the second assignment, starting from the MDIs generated for IR (see Fig. A3), participants were asked to integrate that requirement into the reference MD schema depicted below (which resulted from integrating a set of previous requirements).

References

[1] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross, The Data Warehouse Lifecycle Toolkit, John Wiley & Sons, 1998 ISBN 0471255475.
[2] TPC-H. last accessed March, 2012. ⟨http://www.tpc.org/tpch/⟩.
[3] J.N. Mazón, J. Lechtenbörger, J. Trujillo, A survey on summarizability issues in multidimensional modeling, Data Knowl. Eng. 68 (12) (2009) 1452–1469.
[4] R. Romero, J.N. Mazón, J. Trujillo, M.A. Serrano, M. Piattini, Quality of data warehouses, in: Encyclopedia of Database Systems, 2009a, pp. 2230–2235.
[5] I.Y. Song, R. Khare, B. Dai, Samstar: a semi-automated lexical method for generating star schemas from an entity-relationship diagram, in: DOLAP. 2007, pp. 9–16.
[6] C. Phipps, K.C. Davis, Automating data warehouse conceptual schema design and evaluation, in: DMDW; CEUR Workshop Proceedings, vol. 58, 2002, pp. 23–32.
[7] O. Romero, A. Abelló, A framework for multidimensional design of data warehouses from ontologies, Data Knowl. Eng. 69 (11) (2010) 1138–1157.
[8] M. Golfarelli, S. Rizzi, Data Warehouse Design: Modern Principles and Methodologies, 1st ed. McGraw-Hill, Inc., New York, NY, USA, 2009 ISBN: 0071610391, 9780071610391.
[9] J.N. Mazón, J. Trujillo, An MDA approach for the development of data warehouses, Decis. Support Syst. 45 (1) (2008) 41–58.
[10] O. Romero, A. Simitsis, A. Abelló, GEM: requirement-driven generation of ETL and multidimensional conceptual designs, in: DaWaK, 2011a, pp. 80–95.
[11] M. Golfarelli, F. Mandreoli, W. Penzo, S. Rizzi, E. Turricchia, OLAP query reformulation in peer-to-peer data warehousing, Inf. Syst. 37 (5) (2012) 393–411.
[12] O. Romero, P. Marcel, A. Abelló, V. Peralta, L. Bellatreche, Describing analytical sessions using a multidimensional algebra, in: DaWaK, LNCS, Springer 2011b, pp. 224–239.
[13] O. Romero, A. Abelló, Automatic validation of requirements to support multidimensional design, Data Knowl. Eng. 69 (9) (2010) 917–942.
[14] M.A. Serrano, C. Calero, H.A. Sahraoui, M. Piattini, Empirical studies to assess the understandability of data warehouse schemas using structural metrics, Softw. Qual. J. 16 (1) (2008) 79–106.
[15] O. Romero, D. Calvanese, A. Abelló, M. Rodriguez-Muro, Discovering functional dependencies for multidimensional design. in: DOLAP, ACM, 2009b, pp. 1–8.
[16] D. Skoutas, A. Simitsis, Ontology-based conceptual design of ETL processes for both structured and semi-structured data, Int. J. Sem. Web Inf. Syst. 3 (4) (2007) 1–24.
[17] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia, E. Merlo, Recovering traceability links between code and documentation, IEEE Trans. Softw. Eng. 28 (10) (2002) 970–983.
[18] A. Maté, J. Trujillo, A trace metamodel proposal based on the model driven architecture framework for the traceability of user requirements in data warehouses, Inf. Syst. 37 (8) (2012) 753–766.
[19] P. Jovanovic, O. Romero, A. Simitsis, A. Abelló, Integrating ETL processes from information requirements, in: DaWaK, LNCS, Springer, 2012, pp. 65–80.
[20] P.E. O'Neil, E.J. O'Neil, X. Chen, S. Revilak, The star schema benchmark and augmented fact table indexing, in: TPCTC, LNCS, Springer, 2009, pp. 237–252.

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004

[21] Apache Jena, last accessed September, 2013. ⟨http://jena.apache.org/⟩.
[22] A. Abelló, E. Rodríguez, T. X.B. Urpí, Illa, M.J. Casany, C. Martín, et al., LEARN-SQL: automatic assessment of SQL based on IMS QTI specification, in: ICALT, 2008, pp. 592–593.
[23] M. Golfarelli, S. Rizzi, E. Turricchia, Modern software engineering methodologies meet data warehouse design: 4wd, in: DaWaK, vol. 6862, LNCS, Springer, 2011, pp. 66–79.
[24] M. Golfarelli, D. Maio, S. Rizzi, The dimensional fact model: a conceptual model for data warehouses, Int. J. Coop. Inf. Syst. 7 (2–3) (1998) 215–247.
[25] B. Hüsemann, J. Lechtenbörger, G. Vossen, Conceptual data warehouse modeling, in: DMDW; CEUR Workshop Proceedings, 2000, p. 6.
[26] D. Theodoratos, T.K. Sellis, Incremental design of a data warehouse, J. Intell. Inf. Syst. 15 (1) (2000) 7–27.
[27] D. Theodoratos, M. Bouzeghoub, A general framework for the view selection problem for data warehouse design and evolution, in: DOLAP, 2000, pp. 1–8.
[28] D. Theodoratos, T. Dalamagas, A. Simitsis, M. Stavropoulos, A randomized approach for the incremental design of an evolving data warehouse, in: ER, LNCS, Springer, 2001, pp. 325–338.
[29] M. Blaschka, C. Sapia, G. Höfling, On schema evolution in multidimensional databases, in: DaWaK, LNCS, Springer, 1999, pp. 153–164.
[30] A.A. Vaisman, A.O. Mendelzon, W. Ruaro, S.G. Cymerman, Supporting dimension updates in an OLAP server, Inf. Syst. 29 (2) (2004) 165–185.
[31] C. Quix, Repository support for data warehouse evolution, in: DMDW, CEUR Workshop Proceedings, 1999, p. 4.
[32] P. Vassiliadis, M. Bouzeghoub, C. Quix, Towards quality-oriented data warehouse usage and evolution, Inf. Syst. 25 (2) (2000) 89–115.
[33] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, Policy-regulated management of ETL evolution, J. Data Semant. 13 (2009) 147–177.
[34] B. Bebel, J. Eder, C. Koncilia, T. Morzy, R. Wrembel, Creation and management of versions in multiversion data warehouse, in: SAC, ACM, 2004, pp. 717–723.
[35] M. Body, M. Miquel, Y. Bédard, A. Tchounikine, A multidimensional and multiversion structure for OLAP applications, in: DOLAP, ACM, 2002, pp. 1–6.
[36] M. Golfarelli, J. Lechtenbörger, S. Rizzi, G. Vossen, Schema versioning in data warehouses: enabling cross-version querying via schema augmentation, Data Knowl. Eng. 59 (2) (2006) 435–459.
[37] P. Naggar, L. Pontieri, M. Pupo, G. Terracina, E. Virardi, A model and a toolkit for supporting incremental data warehouse construction, in: DEXA, LNCS, Springer, 2002, pp. 123–132.
[38] R. Torlone, Two approaches to the integration of heterogeneous data warehouses, Distrib. Parallel Databases 23 (1) (2008) 69–97.
[39] A. Maté, J. Trujillo, E. de Gregorio, I.Y. Song, Improving the maintainability of data warehouse designs: modeling relationships between sources and user concepts, in: DOLAP, ACM, 2012, pp. 25–32.
[40] A. Nabli, J. Feki, F. Gargouri, Automatic construction of multidimensional schema from OLAP requirements, in: AICCSA, IEEE Computer Society, 2005, p. 28.

Please cite this article as: P. Jovanovic, et al., A requirement-driven approach to the design and evolution of data warehouses, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2014.01.004