GEM: Requirement-Driven Generation of ETL and Multidimensional Conceptual Designs

Oscar Romero[1], Alkis Simitsis[2], and Alberto Abelló[1]

[1] Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
{oromero,aabello}@essi.upc.edu
[2] HP Labs, Palo Alto, CA, USA
alkis@hp.com

Abstract. At the early stages of a data warehouse design project, the main objective is to collect the business requirements and needs, and translate them into an appropriate conceptual, multidimensional design. Typically, this task is performed manually, through a series of interviews involving two different parties: the business analysts and technical designers. Producing an appropriate conceptual design is an error-prone task that undergoes several rounds of reconciliation and redesigning, until the business needs are satisfied. It is of great importance for the business of an enterprise to facilitate and automate such a process. The goal of our research is to provide designers with a semi-automatic means for producing conceptual multidimensional designs and also, conceptual representation of the extract-transform-load (ETL) processes that orchestrate the data flow from the operational sources to the data warehouse constructs. In particular, we describe a method that combines information about the data sources along with the business requirements, for validating and completing –if necessary– these requirements, producing a multidimensional design, and identifying the ETL operations needed. We present our method in terms of the TPC-DS benchmark and show its applicability and usefulness.

1 Introduction

> "A gemstone or gem is a piece of attractive mineral, which –when cut and polished– is used to make jewelry or other adornments. Most gems are hard, but some soft minerals are used in jewelry because of their lustre or other physical properties that have aesthetic value." (Wikipedia)

As most of the raw materials and resources, gems are out there in large varieties and quantities, but we need to dig and work hard in order to get them and make profit out of them.

Data are the gems of the enterprise. They are available at large quantities, but we need to "dig" for recognizing the relevant and useful ones, and to adjust and polish them for making our valued assets, our "jewelry". The jewelry for an enterprise is any tool or means that facilitates strategic decision making and helps in satisfying business needs. Such a tool is a data warehouse (DW) that

A. Cuzzocrea and U. Dayal (Eds.): DaWaK 2011, LNCS 6862, pp. 80–95, 2011.

organizes the raw, source data in a way that enables decision support. Building a DW requires two essential constructs: the multidimensional (MD) design of the target data stores and the extract-transform-load (ETL) process that populates the target data stores from the source ones.

Nowadays, the construction of conceptual MD and ETL designs is an error-prone, manual process that undergoes several rounds of reconciliation and re-designing, until the business needs are satisfied. It is essential for the business of an enterprise to facilitate, speed up, and automate these design processes.

This paper presents a system called *GEM* (*G*enerating *E*tl and *M*ultidimensional designs). *GEM* starts with a set of source data stores and business requirements –e.g., business queries, service level agreements (SLAs)– and based on these, it produces a MD design for the target data stores, along with a set of ETL operations required for the population of the target DW.

The semantics, characteristics, and constraints of data sources are represented by means of an OWL ontology. The business requirements are expressed in a structured form. We consider functional requirements that drive the generation of the MD design constructs and also, soft or non-functional requirements –e.g., freshness, recoverability, availability– that can be used for giving "lustre" and adding value to our designs. For example, based on a freshness requirement we may decide which data source to use and according to a recoverability requirement we may choose to enrich the ETL process with recovering techniques.

For each business requirement, we identify the relevant part of the data sources (e.g., concepts, attributes, properties) needed to answer it. If we identify conflicts, we either suggest corrections or ask for user feedback. The output of these tasks is an annotated subset of the source ontology that corresponds to a business requirement. Next, we classify the relevant concepts as dimensional or factual and validate the result. We also explore schema information for identifying the respective ETL operations. Finally, we consolidate the individual designs, one for each business requirement, and get the conceptual MD and ETL designs.

Contributions. In particular, our main contributions are as follows.

- We present *GEM*, a system that facilitates the production of ETL and MD designs, starting from a set of business requirements and source data stores. To the best of our knowledge, *GEM* is the first approach towards the semi-automatic generation of both the ETL and MD conceptual designs, since we automatically generate mappings from sources to cubes.
- We propose novel algorithms finding and validating an ontology subset as a MD schema, and identifying ETL operators at the same time.
- We are able to deal with incomplete requirements and validate them.
- We evaluate our method using the schema and constructs of the TPC-DS benchmark and show the quality of the *GEM* designs.

Outline. The rest of the paper is structured as follows. Section 2 formulates the problem at hand and presents the *GEM* architecture. Sections 3 and 4 discuss the validation and completion of business requirements, respectively. Then,

Section 5 describes the validation of the MD design and Section 6 the identification of ETL operations. Section 7 evaluates *GEM* using the TPC-DS benchmark and Section 8 presents the related work.

2 *GEM* in a Nutshell

This section gives an overview of our system, *GEM*. Given two inputs, namely information about the operational sources and a set of user requirements, *GEM* produces two designs: the MD design of the target DW constructs and the conceptual ETL flow that interconnects the target constructs to the operational sources.

2.1 Inputs

Source Data Stores. We capture the semantics of the data sources in terms of an OWL ontology. In previous work, we have shown that a variety of structured and unstructured data stores can be elegantly represented as graphs, and we have also described how we can construct an appropriate ontology for such data stores by integrating a domain vocabulary with the data sources' vocabulary [17]. Here, due to space consideration, we assume that we do have an OWL ontology annotated with the *mappings* of those concepts and properties available in the operational data sources. For further details on how we get this ontology from the sources, we refer the interested reader to our past work [17]. Figure 3 (page 92) depicts an example ontology based on the TPC-DS schema [19].

Business Requirements. In typical DW and ETL engagements, the design starts from a set of functional and non-functional requirements (respectively f-req and nf-req, from here on) expressing business needs. Example requirements could be *"examine stocks provided by suppliers"* or *"a report on total revenue per branch should be updated every 10 minutes"*. Such requirements often come as service level agreements (SLAs) or business queries and are expressed in various forms, either structured or unstructured. Much work has been done in capturing and representing business needs. For example, SLAs expressed as free-form text, require natural language processing (NLP) techniques for being interpreted in a machine processable way. How to capture such requirements are out of the scope of this work. Here, without loss of generality, we consider requirements expressed in a structured way (e.g., by means of i* profiles [22]). Such requirements are represented in an XML file that contains two main parts.

The first part involves functional or information requirements that are captured by identifying the measures and dimensions of interest. In the previous example, *stocks* would be the measure and *suppliers* the dimensional concept.

$$<measures><concept\ id = \ "stocks"/><\!/measures>$$
$$<dimensions><concept\ id = \ "suppliers"/><\!/dimensions>$$

The second part, involves the non-functional requirements of interest for each concept indicated by the functional requirements. For example, the measures

used by the *revenue report* (i.e., the respective view) should conform to a non-functional requirement for *freshness* that requires that the corresponding data should be updated at least every *10 minutes*.

$<concept\ id = "v\_revenue"><nf\_req>$
$\quad <freshness\ format = "HH24{:}MI{:}SS"> \<00{:}10{:}00 </freshness>$
$</nf\_req></concept>$

Due to space restrictions, we omit a detailed description of the XML structure for representing such requirements. Briefly, it contains:

- Levels of detail, which represent data granularity. The user may provide a discretization process for continuous (or with high cardinality) data types.
- Descriptors, which carry out selections over them (i.e., *slicers*). Type of comparison carried out; e.g., "in a given year YYYY".
- Measures, which should be analyzed. Aggregation function and a partial order between them; the latter is needed when we perform different aggregations (one order per dimension). In doing so, we would be able to distinguish between, for example, 'average of sums' and 'sum of averages'.

Note that although our XML structure captures multidimensional requirements over a domain (i.e., non-multidimensional) ontology, the expressivity we support is equivalent to that of the *dimensional expressions* introduced in [4].

In addition, we may have nf-reqs either for each one of the above three elements or for the whole design.

As a remark, different requirements affect different design levels. For example, a freshness requirement indicates how often an ETL flow should run in order to meet the required latency in updating the DW. However, such decision affects the execution level and should be taken under consideration at the physical model. Nevertheless, we may need to use this requirement during the conceptual design as well. For example, assume two source data stores containing the same data but placed in different locations for business reasons (e.g., two snapshots placed in two different branches of the organization). Assume also that the first data store is updated every hour and the second every 5 minutes or that the congestion of the network coming from the first data store is significantly greater than the one coming from the second source. If we have such information, then based on the freshness requirement we need to honor for our target data stores, we should decide to pull data from the second data store. Clearly, such decision is to be taken at the conceptual level.

However, we are interested in capturing all requirements. Those that cannot be used at the conceptual level (which is the focus of this paper) should be transferred to the subsequent, more detailed design levels, along with the outcome of this process; i.e., the conceptual ETL and MD designs. Hence, the designer of the logical and physical models does not need to revisit and reinterpret the original set of business requirements.

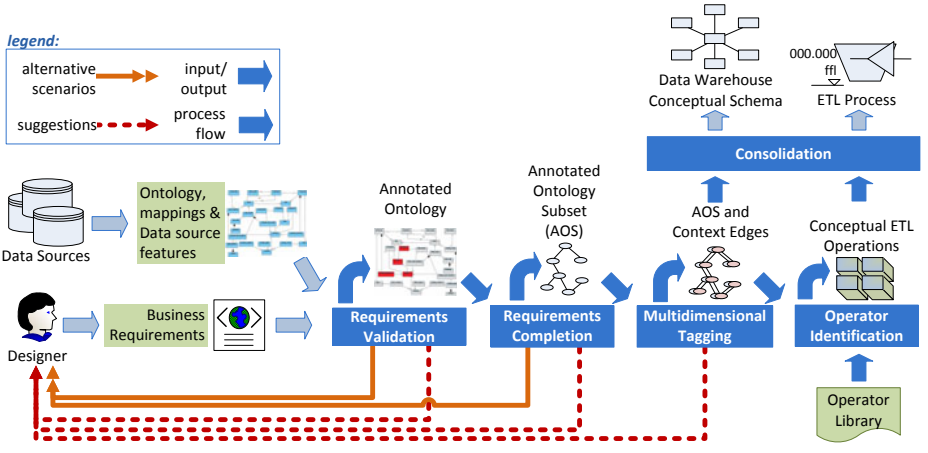

Fig. 1. System architecture

2.2 System Architecture

The process of producing the ETL and MD designs is a semi-automatic process comprising five main stages (see Figure 1). Here, we briefly describe these stages. The next sections provide more details for each stage.

Stage 1: Requirement Validation. First, the system checks if there is a mismatch among the business requirements (either functional or non-functional) in the XML and the data sources, by looking for the corresponding concepts in the ontology and checking whether they are mapped to the sources or not. In case of mismatch, it identifies the possible problems or it may suggest relaxation of the requirements. Otherwise, concepts in the ontology are selected and tagged as either Level, Descriptor or Measure. These concepts are also annotated with nf-reqs and composition of extraction mappings, if necessary.

Stage 2: Requirement Completion. After considering the business requirements, the system complements them with additional information gathered from the sources. This stage identifies intermediate concepts that are not explicitly stated in the business requirements, but are needed in order to answer the f-reqs. User feedback is welcomed for ensuring correctness and compliance to the end-user needs.

Stage 3: Multidimensional Tagging. Next, we tag the new concepts identified by the previous stage, as either factual or dimensional and validate the correctness of these completed f-reqs tagging according to MD design principles. Hence, we check two issues: i) first, whether the factual data is arranged in a MD space (i.e., if each instance of factual data is identified by a point in each of its analysis dimensions) and second, ii) whether the data summarization is correct by examining whether the following conditions hold [8]: (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.

Stage 4: Operator Identification. The ETL operations are identified in three phases. First, we use the annotations generated by the previous steps (i.e., mappings in Stage 1, intermediate concepts in Stage 2, and their taggings in Stage 3) for extracting schema modification operations. Then, we complement the design with additional information that might be found in the sources and with typical ETL operations regarding surrogate key and slowly changing dimensions.

Stage 5: Conciliation. The previous stages run once for each f-req. Eventually, the individual results obtained per f-req are conciliated in a single conceptual MD schema and a single ETL flow.

2.3 Output

At the end, we produce a conceptual, MD schema composed by facts and dimensions. In addition, we identify the ETL operations needed in order to interconnect the source data stores to the MD constructs.

3 Requirement Validation

Starting from the inputs discussed in Section 2.1, we validate the business requirements w.r.t. the available data sources, as follows: (a) we analyze the input XML file and *tag* the ontology concepts corresponding to the f-req, identifying possible mapping conflicts, and (b) we include and then validate assertions regarding nf-reqs and data sources features. The input XML file contains three kinds of concepts: measures, levels, and descriptors (see Section 2.1). So, first, we tag the corresponding concepts in the input ontology with these labels. Then, we check whether the tagged concepts can be mapped to the sources (either directly or by means of ETL operators). When an error occurs, user feedback is required. The validation method is as follows:

1. **if** the tagged concept is mapped to the sources **then** no further action is needed
2. **else if** the tagged concept is involved in a concept taxonomy **then**
   (a) **if** any of its subclass(es) has (have) a mapping **then** we annotate the tagged concept with the ETL operations 'renaming' and 'union'
   (b) **else if** any superclass has a mapping **then** we use the general concept mapped and annotate the required concept with ETL operations 'renaming' and 'selection'
      i. **if** *discriminant function* has not been specified in the input XML file **then** user feedback is required
      i. **if** the tagged node has several superclasses **then** 'minus' or 'intersection' are also considered (see Section 6 for details)
3. **else if** exists a (transitive) one-to-one association to a mapped concept **then** suggest it as a potential synonym
   (a) **if** the suggestion is accepted **then** the f-req is updated with the synonym concept
4. **else** the concept is not available in the data sources

4 Requirement Completion

This stage takes as input the annotated ontology produced in the previous stage and it *completes* the requirements regarding the sources. First, it identifies *intermediate concepts* that are not explicitly stated in the f-req, but needed to

retrieve the required information. If an f-req cannot be met, it suggests alternative solutions. Finally, it produces the ontology subset needed to answer the business query at hand and additional annotations regarding ETL operations.

This stage starts with a pruning process. We identify how tagged concepts are related in the ontology and then, (a) we disregard concepts/relationships not mapped nor tagged (if a concept taxonomy is affected, we replace the concept pruned with the first superclass mapped/tagged); and next, (b) we prune all the mapped many-to-many (i.e., *-*) associations. Note that such associations violate the three summarization necessary conditions [8] and thus, they cannot be exploited for MD design. The outcome of this pruning is a *s*ubset of the input *a*nnotated *o*ntology, which we call AOS. Since an arbitrary ontology can be represented as a graph, we will talk about *paths* between concepts and thus, we will also refer to concepts as *nodes* and to associations as *edges*.

Looking for Paths Between Tagged Concepts. For identifying how tagged concepts are related in the sources, we use the following algorithm that computes paths between tagged concepts.

1. **foreach** edge *e* **in** O **do**
   (a) **if** *right*_left_concepts(*e*) **are** tagged **then** paths_between_tagged_concepts ∪=*e*;
   (b) **else if** *right*_concept(*e*) **is** tagged **then** max_length_paths ∪=e; *//Seed edges*
2. **while** size(max_length_paths) != ∅ **do**
   (a) paths := ∅;
   (b) **foreach** path *p* **in** max_length_paths **do**
      i. extended_paths := explore_new_edges(*p*, O); *//only considering edges not in p*
      ii. **foreach** path *p1* **in** extended_paths **do**
         A. **if** *left*_concept(*p1*) **is** tagged **then** paths_between_tagged_concepts ∪=*p1*;
         B. **else** paths ∪= p1;
   (c) max_length_paths := paths;
3. **return** paths_between_tagged_concepts;

We start by identifying edges directly relating tagged concepts (step 1a) and edges reaching tagged concepts (from now on, *seed* edges; step 1b). For the sake of understandability, although the AOS has no directed edges, we say that the tagged node is in the seed edge right-end, and its counterpart to be in the the left-end. Then, the algorithm applies the transitive property starting from tagged concepts. At the first iteration, we explore new edges such that their right-end matches the left-end of a seed edge, and similarly for the forthcoming iterations (step 2(b)i). Intuitively, we explore paths starting from tagged concepts by exploring a new edge per iteration. This guided exploration has two main restrictions: we cannot explore any edge already explored in a given path (step 2(b)i) and if we reach another tagged concept we finish exploring that path (i.e., we have found a path between tagged concepts; step 2(b)iiA). Note that in a given iteration *i*, we only explore the longest paths computed in the previous iteration (steps 1b and 2c). Eventually, we explore all the paths and the algorithm finishes (step 2). Observe that step 1 can be computed by means of generic ontological reasoning.

This algorithm is *sound* since it computes direct relationships and propagates them according to the transitivity rule and *complete*, because it converges; note that each path is explored *only* once. This algorithm has a theoretical exponential upper bound regarding the size of the longest path between tagged concepts.

However, this theoretical upper bound is hardly achievable in real-world ontologies as they have neither all classes with maximum connectivity nor all paths are of maximum length. Moreover, note that *-* relationships were previously pruned. (See also our evaluation in Section 7).

Producing the Output Subset. Based on the paths between tagged concepts that the previous algorithm found, the following algorithm determines the ontology subset needed to answer the f-req.

1. **if** between two tagged concepts there are more than one path **then** we ask the user for disambiguation (i.e., which is the path fulfilling the semantics needed for the f-req at hand)
2. **foreach** pair of related tagged concepts not involving a descriptor **do**
   (a) Edges forming that path are annotated as aggregation edges, because these relationships determine the data granularity of the output

The AOS is compound by the paths selected in step 1. Note that these paths include the intermediate concepts (i.e., those not tagged but involved in the paths) and that the user may not select any path between a given pair of concepts. At this point, taxonomies are also disregarded.

Annotating the Ontology AOS. Having an AOS containing the new concepts needed to answer the f-req (besides those in the input XML file), we check whether the whole graph makes MD sense.

First, we check the semantics of each edge according to the tag -if any- of the related concepts and its multiplicity. According to these semantics, we tag each edge with MD relationships that it could represent; i.e., related MD concepts. Next, we consider *factual nodes* (those tagged as measures) and *dimensional nodes* (those either tagged as levels or descriptors). For guaranteeing the MD design principles (see Section 2.2), factual and dimensional nodes must be related properly. For example, factual data cannot be related to dimensional data by means of a one-to-many (i.e., 1-*) association, as by definition, each instance of factual data is identified by a point in each of its analysis dimensions. Dimensional data can only appear in the *-end of an edge when the other end is also tagged as dimensional data. Furthermore, non-complete associations –i.e., accepting zeros– in the dimensional end are not allowed either, as they do not preserve completeness.

Hence, we analyze the graph looking for not correct edges and try to fix them. For example, if the node in the *-end of a *-1 association is tagged as dimensional then, its counterpart should also be dimensional. If by doing so we have been able to infer an unequivocal label, this knowledge is propagated to the rest of the AOS. However, if we identify a meaningless conceptual relationship –i.e., when both ends are tagged in a forbidden way– the algorithm stops and alternative analysis scenarios are proposed. For this task, we use previously proposed techniques, as those described in [14].

5 Multidimensional Validation

This stage validates the AOS and checks whether its concepts and associations collectively produce a data cube. If the validation fails (according to the

constraints discussed in Section 2.2), *GEM* proposes alternative analysis solutions. Otherwise, the resulting MD schema is directly derived from the AOS.

The previous stage might have propagated some tags when tagging the AOS associations (i.e., inferring unequivocal knowledge), but it does not guarantee that all the concepts have a MD tag at this point. Thus, we start this stage with a pre-process aimed at deriving new MD knowledge from non-tagged concepts, and each non-tagged concept is considered to play a dimensional role or a factual role. Furthermore, it would be possible to retag a dimensional node as dimensional/factual node. Next, we validate if any of these tags, eventually, are sound in a MD sense. Thus, in this step, we determine every potential MD tagging that would make sense for the input f-req and we also determine how these alternatives would affect the output schema, deriving (in some cases) interesting analytical options that may have been overlooked by the designer.

For each possible combination of new tags, an *alternative annotation* is created if the tags do not contradict the edge semantics already depicted in the AOS. Subsequently, each of these AOS will be validated and only those that make MD sense will be finally considered. Therefore, an f-req can produce several valid MD taggings for the same AOS and thus, multiple MD schemas.

The validation process introduced in this stage guarantees the *multidimensional normal forms* presented in [6,7] for validating the output MD schema, and the summarizability constraints discussed in [10]. The following algorithm is called once for each alternative tagging generated.

1. **If** $!factualdata(AOS)$ **then** return $notifyFail$("The requirement does not include any fact.");
2. **If** $!connected(AOS)$ **then** return $notifyFail$("Cartesian product is not allowed.");
3. **For each** $subgraphOfLevels \subset AOS$ **do**
   (a) **If** $cycles(subgraphOfLevels)$ **and** $contradictoryMultiplicities(subgraphOfLevels)$ **then**
      i. **return** $notifyFail$("Cycles cannot be used to select data");
   (b) **If** $existsTwoLevelsRelatedSameFactualData(subgraphOfLevels)$ **then**
      i. **return** $notifyFail$("Non-orthogonal Analysis Levels");
   (c) **For each** $(c_1, c_2) \in getToManyEdges(subgraphOfLevels)$ **do**
      i. **If** $relatedToNodesWithMeasures(AOS, c_2)$ **then**
         A. **return** $notifyFail$("Aggregation Problems");
4. **For each** $cycle \subset AOS$ **do**
   (a) **If** $contradictoryMultiplicities(cycle)$ **then**
      i. **return** $notifyFail$("Cycles cannot be used to select data");
   (b) **else**
      i. $askUserForSemanticValidation()$;
      ii. $add(AOS, newContextEdge(bottom(cycle), top(cycle), cycle))$;
5. **For each** $(c_1, c_2) \in getToManyEdges(AOS)$ **do**
   (a) **If** $relatedToNodesWithMeasures(AOS, c_2)$ **then**
      i. **return** $notifyFail$("Aggregation problems between Measures");

Step 1 ensures that the AOS contains factual data. Note that in our preprocess we could have tagged nodes as factual data that do not contain measures. From here on, we distinguish between *factual nodes* and *factual nodes with measures*. So this function returns false if all the nodes are tagged as dimensional data. Step 2 ensures that the AOS is *connected* to avoid "Cartesian Product".

The intuition behind steps 3 to 5 is shown in Figure 2. Step 3 validates levels subgraphs (i.e., subgraphs only containing level concepts) with regard to where factual nodes are placed. First, every subgraph must represent a valid dimension

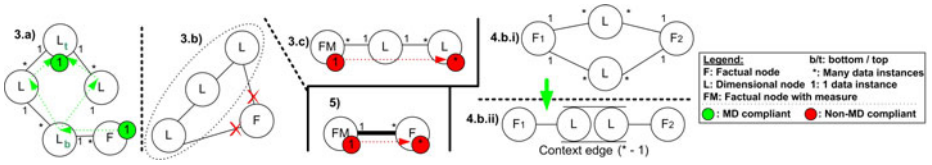

Fig. 2. Graphical representation of the multidimensional validation steps

hierarchy. We must be able to identify two nodes in the level subgraph which represent the *top* and *bottom* levels of the hierarchy (Step 3a). Two different levels in a subgraph cannot be related to the same factual node (Step 3b). Moreover, level - level edges raising aggregation problems in factual nodes with measures must be forbidden (Step 3c). Note that by convention we assume that in every *-1 edge (c_1, c_2), c_1 corresponds to the * end of the association. Hence, Step 3 validates the correspondences between dimensional nodes, whereas Step 4 generates the path of factual nodes (MD data retrieved); i.e., it validates cycles in the path of factual nodes to ensure that they are not used to select data, similarly to the validation of levels cycles in 3a. Once the cycle has been validated, the edges involved are clustered in a *context edge* (since cycles are checked to correspond a correct multi-path aggregation hierarchy, i.e., a one-to-many or one-to-one lattice) tagged with the lattice multiplicity, as shown in Figure 2. Finally, Step 5 looks for aggregation problems induced by factual nodes with measures at the 1-end of a 1-* edge –either context edge or not.

6 Operation Identification

For each graph validated as a data cube in the previous stage, we launch an ETL operation identification process, which is a semi-automatic process that comprises three phases.

Phase I. This phase identifies operations that are needed for mapping the source to target data stores, using the target schema produced in the previous stage. For example, for aggregating over states, we need a location dimension at the target site and to map it with source information about zip code, street address, and so on.

During this phase, we identify mainly schema modification operations as follows. *Selection* is generated from concepts having attached a selection condition: from slicers recorded in AOS; or when a required concept does not have any mapped source (neither it nor its subclasses), while some of its superclasses do have such mapping. *Union* appears when a required concept is not directly mapped to the sources, but some of its subclasses are. Similarly, *Intersection* and *Minus* are generated when a concept is not mapped but some of its superclasses are. *Join* is generated for every association in the ontology; if one or both of the association ends is not mandatory, we state it as outer. *Aggregation* is generated when a *-1 association is found so that there is a measure at its *-end.

Renaming is generated for each attribute in the data sources and gives to it the name of the corresponding ontological concept. *Projection* is generated for each concept and association in the ontology. *Function* expresses operations stated in the requirements, like a discretization process for an attribute to be used in a dimension or a transformation for an attribute to facilitate its interpretation as a measure.

Starting from the AOS, we iteratively synthesize several of its nodes into one single operation, as shown in the algorithm placed in the next page.

The *ETL* variable is a *directed acyclic graph* that tracks the ETL flow generated, whereas the $findOper$(ETL g, concept c) function looks for a node in g, with no successors, such that it contains c. Step 1 considers extraction operations like a single table access, a union, an intersection or a minus operation, along with the corresponding selection, projection, renaming mechanisms, and functions. Step 2 fuses all data that do not involve any aggregation. Hence, for those AOS nodes related by means of 1-1 associations (i.e., identity), we join their corresponding operations in the ETL. We also join nodes connected with edges that do not involve aggregation (i.e., stemming from slicing requirements and identified in Section 4).

1. **For each** $c \in AOS$ **do**
   (a) $add(ETL, newExtraction(c))$;
2. **For each** $(c_1, c_2) \in edges(AOS)$ **do**
   (a) **If** $multiplicity((c_1, c_2)) =$ "$1-1$" **or not** $aggregationEdge((c_1, c_2))$ **then**
      i. $o_1 := findOper(ETL, c_1); o_2 := findOper(ETL, c_2)$;
      ii. **If** $o_1 <> o_2$ **then** $add(ETL, newJoin(o_1, o_2, getGroupingAttrs(o_1)))$;
3. **For each** $o \in ETL$ **and** $successors(ETL, o) = \emptyset$ **and** $| outputEdges(AOS, o) | > 1$ **do**
   (a) $setGroupingAttrs(o, \emptyset)$; $e := outputEdges(AOS, o)$;
   (b) **For each** $(c_1, c_2) \in (e)$ **do**
      i. $o_2 := findOper(ETL, c_2)$;
      ii. $o := newJoin(o, o_2, getGroupingAttrs(o) \cup getGroupingAttrs(o_2))$;
      iii. $add(ETL, o)$;
   (c) $add(ETL, newAggr(o, getGroupingAttrs(o))$;
4. **While not** connected(ETL) **do**
   (a) $(c_1, c_2) := first(\bigcup_{o=containsMeasure(ETL)} outputEdges(o))$;
   (b) $o_1 := findOper(ETL, c_1); o_2 := findOper(ETL, c_2)$;
   (c) $o_3 := newJoin(o_1, o_2, (getGroupingAttrs(o_1) \setminus getAttr(c_1)) \cup getGroupingAttrs(o_2))$;
   (d) $add(ETL, o_3)$; $add(ETL, newAggr(o_3, getGroupingAttrs(o_3)))$;

Step 3 creates the basic cubes. First, we check the already generated operations that have no successors, and whose AOS nodes have more than one edge with the 1-end related to a concept in another ETL node without successors (observe that after step 2 only *-1 associations remain). Next, we successively join these operations. The grouping attributes of the final operation is the union of the grouping attributes of each joined operation. Note that a grouping operation is generated to guarantee that data is at the appropriate granularity.

Finally, step 4 connects all cubes produced, starting from those with measures, by following the order specified by the requirements. Since each AOS edge not used yet corresponds to an aggregation, we join the output of the operations (following the AOS aggregation edges), substitute the grouping attributes of c_1 by those of the new aggregation level c_2, and generate the grouping operation taking into account the new attributes. The choice of the aggregation function

depends on the requirements (there, it should be associated to a corresponding measure and c_2) or a default one is used; e.g., SUM.

Phase II. During this phase, the designer might want to refine the design produced by checking for additional information at the sources that might be useful. (Part of this phase can be done before Phase I too.) For example, the domain ontology might relate *state* with *zip code* and *street address*. If there is a source containing information about "location" and contains both the street address and zip code in the same field, then such information is definitely useful, but the domain ontology cannot help. We can correct this by enriching the result with such a mapping and producing the appropriate function(s).

Nf-reqs can be exploited in a similar way. For example, a strict requirement regarding *recoverability* may suggest to consider adding recovery points at points of the flow that are generally known for being expensive (e.g., after the extraction phase or after an expensive blocking operator [16]). Of course the final decision on which are the good places to add recovery points is to be taken by an optimizer at the logical level [16].

The same holds when we work with f-reqs that involve the data itself. For example, a requirement like "make sure that each customer is considered once" can add a "de-duplicate customer info" operation to the design.

Phase III. The last phase complements the design with operations needed to satisfy standard business and design needs. This task is mainly automatic and involves typical DW operations that can be identified and added to the design *after* the consolidation phase.

For example, common practices suggest replacing production keys with surrogate keys. For that, the system identifies the respective production keys and enriches the design with appropriate 'surrogate key assignment' operations. Similarly, the system adds operations that take care of slowly changing dimensions (SCDs). There are standard dimensions that are not updated very often (e.g., dimensions that keep structural information about the organization such as geographical location, customer information or product information). Hence, the design can be enriched with operations that handle the update of such dimensions. Possible update operations for SCDs can be: do nothing (do not propagate changes), keep no history (overwrite old values with new data), keep history by creating multiple records in the dimensional tables with separate keys, keep history using separate columns, keep history by storing new data to an active table and keep (all or some of the) old values to 'history tables', or use a hybrid approach. Of course, here we list just a few frequently used operations. The list can go long and our method is extensible to adapt such a list.

7 Evaluation

We evaluated *GEM* using the TPC-DS benchmark [19]. TPC-DS provides a set of DW tables –both facts and dimensions– along with a set of data sources. ETL operations (or data maintenance functions according to TPC-DS) are also provided, for maintaining fact tables and dimensions. Finally, a set of business

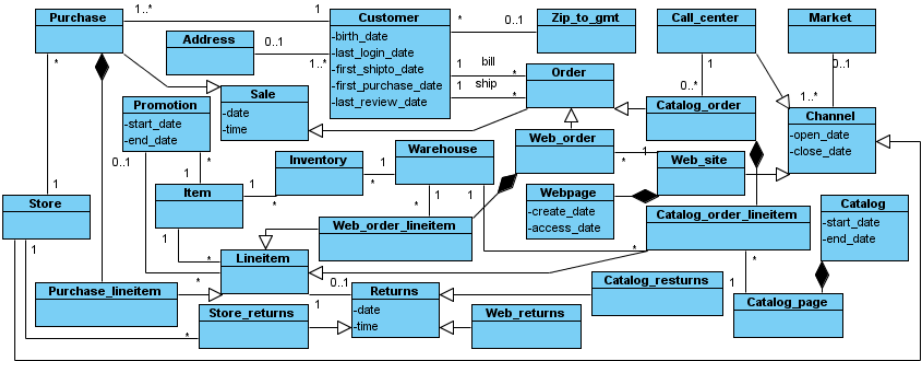

Fig. 3. Ontology for TPC-DS data sources

queries (i.e., business requirements) exists. Having all these constructs allows us to evaluate our method as follows. Starting only from the business queries and the data source, we use *GEM* for producing the DW schema and ETL operations. Then, we compare our solutions to the design constructs provided by the benchmark. Here, due to limited space, we show results concerning the store_sales cube (the results generalize throughout the whole benchmark though).

We worked as follows. We constructed an ontology containing all source tables, specializations, and added some additional concepts that do not map to data sources (see Figure 3). Thus, we intentionally make the ontology more complex by adding more classes to stress *GEM*; note, that adding more associations does not affect *GEM*, since these would be pruned during AOS creation.

First, we examine the search space produced for AOS creation. Figure 4 presents the number of algorithm iterations needed to converge, the total number of paths computed, the number of paths between tagged concepts (i.e., the output), and the maximum length of the output, per business query. The results show that the search space is not exponential regarding the length of the longest path. Indeed, although the average length of the longest path is 8, in the worst case, our algorithm computes no more than 178 paths (24 between tagged concepts). These findings verify the feasibility and efficiency of our approach in real-world cases. In fact, the worst total time did not exceed 900ms. Constructing AOS is the most expensive part of our method; the rest tasks are processed fairly fast, in much less time.

Next, we evaluated the quality of our solutions (see Figure 5). Every business query reveals a part of the final design (tables and attributes). Frequently, business queries reveal overlapping information. However, after a few iterations over these queries (in fact, after the fifth query) we identified correctly *all* target tables. Since numerous attributes are involved overall, identifying them requires digging into more requirements. After processing 11 business queries, we identified almost 40% of the total attributes. However, attributes are added throughout the whole process. For example, surrogate keys are identified after Phase III of the ETL operation identification task.

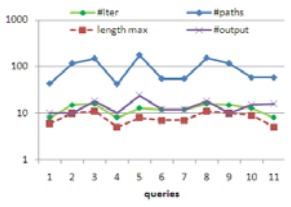

Fig. 4. Space

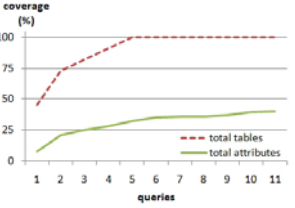

Fig. 5. MD coverage

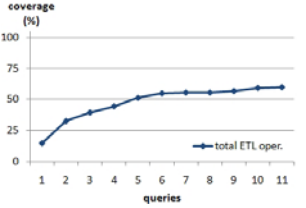

Fig. 6. ETL coverage

Two observations can be made at this point. One may find tempting the fact that the target tables are identified really fast. Thus, after a certain point of her choice, the designer might want to stop this automatic process and start refining the design by herself. As an aside issue, many business queries involve the same target design constructs. This means that these constructs (e.g., tables) should be quite popular and this information can help us in the physical design; e.g., for choosing appropriate indices or partitioning schemes.

Similar are the findings for the identification of ETL operations (see Figure 6). *GEM* returned almost 60% of ETL operations after the completion of Phase I. The remaining operations (not shown in the figure) are mostly surrogate key assignments and a few SCDs, which are identified after Phase III. Therefore, *GEM* identifies the complete set of ETL operations for the TPC-DS case.

8 Related Work

Various efforts have been proposed for the conceptual ETL modeling. These include approaches based on ad hoc formalisms [20], on standard languages like UML (e.g., [9]), MDA (e.g., [11,12]), BPMN [1], and on semantic Web technology and graph transformations [17]. Most of these works do not specifically consider business requirements and do not describe how such requirements drive ETL design. Recent research on optimization of information integration flows proposed techniques for incorporating such objectives into ETL design [2,15,16,21]. However, none of the abovementioned research efforts considers synchronous creation of MD design. In addition, commercial, off-the-shelf ETL products do not offer functionality similar to the one described in this paper.

Many works have dealt with designing DW models; e.g., [3,5,11,13,18], to mention a few, but the list is long. However, in most works, it seems that the more the process gets automated, the more the integration of requirements is overlooked on the way. Recently, the use of ontologies was considered for facilitating this task [13]. However, that work aims at identifying the MD knowledge contained in the sources and overlooks business requirements. Another approach to MD design considers business requirements too [14], but the f-req are considered in the form of SQL queries, so a major design task is done manually. *GEM* automates this part and automatically creates such queries from f-req. In addition, *GEM* is different from all previous approaches in that it identifies the ETL operation at the same time.

9 Conclusions

We have presented *GEM*. A system that facilitates the (semi-)automatic generation of ETL and MD conceptual designs, starting from a set of business requirements and data sources. In particular, we have described how the requirements can be validated and enriched, in order to produce an annotated ontology containing correct information for both the sources and the requirements. Then, we have shown how to use this ontology for producing the MD and ETL conceptual designs. Finally, we have reported on our experimental findings working on the TPC-DS benchmark. Our future plans involve extending our techniques to the logical and physical levels, for facilitating their (semi-)automatic production.

Acknowledgements. This work has been partly supported by the Ministerio de Ciencia e Innovación under project TIN2008-03863.

References

1. Akkaoui, Z.E., Zimányi, E.: Defining ETL worfklows using BPMN and BPEL. In: DOLAP, pp. 41–48 (2009)
2. Dayal, U., Castellanos, M., Simitsis, A., Wilkinson, K.: Data Integration Flows for Business Intelligence. In: EDBT, pp. 1–11 (2009)
3. Golfarelli, M., Maio, D., Rizzi, S.: The Dimensional Fact Model: A Conceptual Model for Data Warehouses. IJCIS, 215–247 (1998)
4. Golfarelli, M., Rizzi, S.: Data Warehouse Design. Modern Principles and Methodologies. McGraw-Hill, New York (2009)
5. Hüsemann, B., Lechtenbörger, J., Vossen, G.: Conceptual Data Warehouse Modeling. In: DMDW, pp. 1–11 (2000)
6. Lechtenbörger, J., Vossen, G.: Multidimensional Normal Forms for Data Warehouse Design. Information Systems, 415–434 (2003)
7. Lehner, W., Albrecht, J., Wedekind, H.: Normal Forms for Multidimensional Databases. In: SSDBM, pp. 63–72 (1998)
8. Lenz, H., Shoshani, A.: Summarizability in OLAP and Statistical Data Bases. In: SSDBM, pp. 132–143 (1997)
9. Luján-Mora, S., Vassiliadis, P., Trujillo, J.: Data mapping diagrams for data warehouse design with UML. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 191–204. Springer, Heidelberg (2004)
10. Mazón, J., Lechtenbörger, J., Trujillo, J.: A Survey on Summarizability Issues in Multidimensional Modeling. DKE, 1452–1469 (2009)
11. Mazón, J.N., Trujillo, J.: An MDA Approach for the Development of Data Warehouses. In: DSS, pp. 41–58 (2008)
12. Muñoz, L., Mazón, J.N., Trujillo, J.: Automatic Generation of ETL Processes from Conceptual Models. In: DOLAP, pp. 33–40 (2009)
13. Romero, O., Abelló, A.: A Framework for Multidimensional Design of Data Warehouses from Ontologies. Data & Knowledge Engineering 69(11), 1138–1157 (2010)
14. Romero, O., Abelló, A.: Automatic Validation of Requirements to Support Multidimensional Design. Data Knowl. Eng. 69(9), 917–942 (2010)
15. Simitsis, A., Wilkinson, K., Castellanos, M., Dayal, U.: QoX-driven ETL design: Reducing the Cost of ETL Consulting Engagements. In: SIGMOD (2009)

16. Simitsis, A., Wilkinson, K., Dayal, U., Castellanos, M.: Optimizing ETL Workflows for Fault-Tolerance. In: ICDE, pp. 385–396 (2010)
17. Skoutas, D., Simitsis, A.: Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data. IJSWIS, 1–24 (2007)
18. Song, I., Khare, R., Dai, B.: SAMSTAR: A Semi-Automated Lexical Method for Generating STAR Schemas from an ER Diagram. In: DOLAP, pp. 9–16 (2007)
19. TPC: TPC-DS specification (2010), `http://www.tpc.org/tpcds/`
20. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Conceptual modeling for ETL processes. In: DOLAP, pp. 14–21 (2002)
21. Wilkinson, K., Simitsis, A.: Designing Integration Flows Using Hypercubes. In: EDBT (2011)
22. Yu, E.S.K., Mylopoulos, J.: From E-R to "A-R" - Modelling Strategic Actor Relationships for Business Process Reengineering. In: ER, pp. 548–565 (1994)