A Framework for Multidimensional Design of Data Warehouses from Ontologies

Oscar Romero and Alberto Abelló

Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain
{oromero/aabello}@lsi.upc.edu

Abstract

Some research efforts have proposed the automation of the data warehouse design in order to free this task of being (completely) performed by an expert and facilitate the whole process. Most advanced approaches exclusively work over relational sources and perform a detailed analysis of the data sources to identify the multidimensional concepts in a reengineering process. Starting from a logical schema, however, may present some inconveniences. A logical schema is tied to the design decisions made when devising the system and these decisions either made to fulfill the system requirements (for instance, improve query answering, avoid insertion / deletion anomalies, preserve features inherited from legacy systems, etc.) or naively made by non-expert users, have a big impact on the quality of the multidimensional schemas got by current automatable approaches.

In this paper, we introduce our approach for automatically deriving the multidimensional schema from a domain ontology. Our goals are mainly two: i) we want to improve the quality of the output got (by working over a conceptual formalization of the domain instead of a logical one) and ii) we want to automate the process. This second goal is the main reason for choosing ontologies instead of other conceptual formalizations, as ontology languages provide reasoning services that will facilitate the automation of our task.

Key words: OLAP, Multidimensional Design, Ontologies

1. Introduction

Data warehousing systems were conceived to support decision making within organizations. On one hand, relevant data for decision making is extracted from the data sources, transformed (i.e., cleaned and homogenized) and finally integrated within a huge repository of data (the *data warehouse*), which provides a single and detailed view of the organization. On the other hand, the *exploitation tools* provide different mechanisms to navigate and perform analysis tasks over the data warehouse. Among the different kind of exploitation tools, *OLAP (On-line Analytical Processing) tools* have gained relevance in the last years so much so that the data warehousing and OLAP concepts are now tightly related. OLAP tools are intended to facilitate information analysis and navigation through the business data based on the *multidimensional* paradigm, and nowadays it is widely accepted that the conceptual schema of a data warehouse must be structured according to the multidimensional model.

Like in most information systems, the data warehouse design has been typically carried out manually, and the experts' knowledge and experience are crucial to identify relevant multidimensional knowledge contained in the sources. Nevertheless, the data warehouse conceptual schema must be derived by a *reengineering* process from the data sources and to some extend, this process should be automatic. In this sense, some research efforts have proposed the automation of the data warehouse design in order to free this task of being (completely) performed by an expert, and facilitate the whole process. Mostly, these approaches carry out this process from relational OLTP (*On-Line Transaction Processing*) systems, assuming that a RDBMS is the most common kind of data source we may find, and taking as starting point a relational schema (i.e., a logical schema).

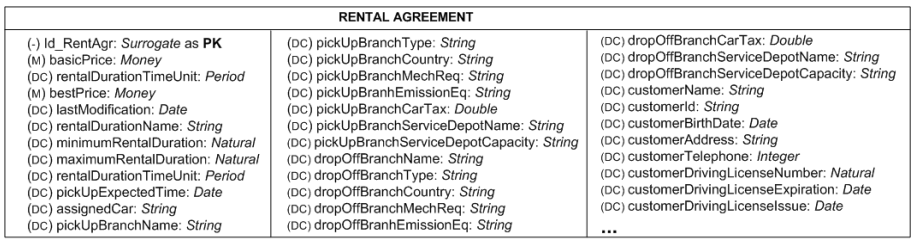

Figure 1: A logical implementation in a RDBMS of a car rental agreement

Starting from a logical schema, however, may present some inconveniences. A logical schema is tied to the design decisions made when devising the system and these decisions either made to fulfill the system requirements (for instance, improve query answering, avoid insertion / deletion anomalies, preserve features inherited from legacy systems, etc.) or naively made by non-expert users, have a big impact on the quality of the multidimensional schemas got by current automatable approaches. In fact, these approaches require a certain degree of normalization in the input logical schema to guarantee that it captures as much as possible the to-one relationships existing in the domain. As detailed in section 2, discovering this kind of relationships is crucial in the design of the data warehouse, and the most common way to represent them at a logical level is by means of "foreign" (FK) and "candidate key" (CK) constraints. This scenario can be clearly seen in the example shown in figure 1.

There, a single relation (named `rental agreement`) models data related to a `car rental agreement` in a RDBMS. Each row represents an attribute of the relation (in *italics* its data type). The "primary key" of the relation is identified by the PK label and the capital letters in brackets besides each attribute represent the multidimensional role that attribute should play according to its semantics (*M* stands for *measure*; i.e., interesting business measures of our fact of study, and *DC* for *dimensional concept*; i.e., interesting perspectives of view of our fact -a detailed definition of the multidimensional concepts may be found in section 3-). Only those concepts that would play a meaningful role in the multidimensional schema are shown in the figure, but additional attributes could be found in the relation (depicted by the ellipsis at the end). In this case, current methodologies would either i) overlook all the dimensional concepts (since they are not involved in any CK or FK), or ii) identify all the non-numerical attributes as dimensional concepts (i.e., even those not making multidimensional sense and not shown in the figure). Furthermore, even if they were able to identify any dimensional concept they would not be able to identify potential aggregation paths (or *roll-up* relationships) that would give rise to *dimension hierarchies*. Thus, they are not able to answer the following questions: is each dimensional concept conforming a dimension by itself? which of them would conform the same dimension hierarchy (i.e., which are *levels* and which *descriptors* within the same dimension)? which belong to the same dimension and which to dimensions semantically related?

Dimension hierarchies are crucial in the multidimensional model which is based on two main features: 1) placement of data in the multidimensional space and 2) summarizability of data. A bad design of the dimension hierarchies would directly impact on the aggregation paths we may have. Modify data granularity when showing data to the user is a key feature of OLAP tools (performed through the "roll-up" and "drill-down" operators) and overlooking aggregation paths in the system design task would impact on the success of the whole system. Indeed, any intermediate situation between a denormalized schema and a logical schema in 3NF would affect the output quality of current multidimensional design methods.

This scenario can be avoided modeling the data warehouse from a conceptual formalization of the domain. The role of a conceptual layer on the top of information systems has been discussed in depth in the literature (see, for instance, [19]). In case of reengineering processes like the data warehouse conceptual design, the benefits are clear: the conceptual layer provides more and better knowledge about the domain to carry out this task. For instance, consider now the ontology depicted in figure 2. This ontology plays a conceptual role regarding the logical implementation depicted in figure 1. The piece of ontology depicted in the figure (that will be used as example along the paper) refers to a `car rental agreement` between a `branch` and a

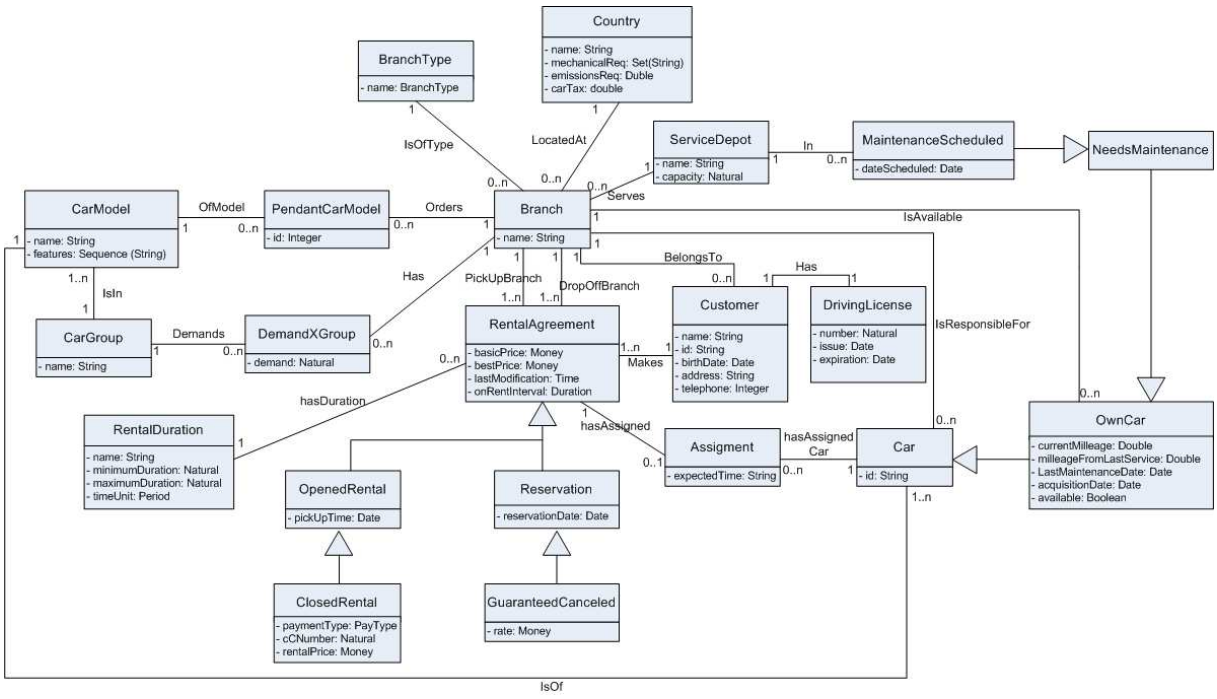

Figure 2: Diagrammatic representation (based on UML notation) of a piece of a car renting ontology. The whole EU-Car Rental ontology [9] in OWL DL notation can be found at www.lsi.upc.edu/∼ oromero/EUCarRental.owl

`costumer`. For a given `rental agreement` (which can still be ongoing -i.e., an `opened rental` or already closed -i.e., a `closed rental` and/or be booked by `reservation` -with or without `guaranteed canceled`-), a `car` is `assigned`. Several information about the `branch` is captured as well, such as `pendant car models` to be assigned to `rental agreements`, the `demand` got by a given kind of `car group` or the `service depot` associated to a `branch`. Moreover, a `car` belongs to a `branch` and it is assigned to a `service depot` when `maintenance needed`. There, each relevant concept of the domain is clearly stated as well as its relationships with the other concepts, and for instance we will able to propose a `car` to be summarizable into two different aggregation paths (into `car model` and `car group` but also through the `branch` to the `country`, `branch type` and `service depot` they belong to), that would conform, as a whole, the `car` dimension.

In this paper we introduce AMDO (*Automating Multidimensional Design from Ontologies*), our approach for automatically deriving the multidimensional schema from a domain ontology. Our goals are mainly two: i) we want to improve the quality of the output got (by working over a conceptual formalization of the domain instead of a logical one) and ii) we want to automate the process. This second goal is the main reason for choosing ontologies instead of other conceptual formalizations, as ontology languages provide reasoning services that will facilitate the automation of our task. Specifically, we choose OWL DL [27], a W3C recommendation, as our input ontology language. Later, we will show that our algorithm can be adapted to other ontology languages (even less expressive, as we discuss in section 6).

Nowadays ontology languages are widely used in different areas like data integration [18] and the Semantic Web [5], but in other areas, like software engineering, UML [13] and ER [8] are the most common choices. In these cases, our approach requires a *pre-process* to generate an OWL DL ontology from the UML or ER diagram. This process can be automated nowadays [2, 4, 7, 10, 19] and the expressivity needed in DL to capture UML / ER diagrams has already been addressed in the literature [2, 4, 7].

At this point it is important to remark that if a conceptual formalization of the domain is not available then, by means of reverse engineering we may extract the ontology from the logical schema. However, the output got by AMDO in this case would be equivalent to results got by those approaches automating the

design from logical schemas. The reason is that the ontology derived would reflect the logical design decisions made and thus, its potential lack of semantics would entail the problems described previously.

The structure of this paper is organized as follows. Section 2 discusses the related work underlining automatable approaches. Section 3 sets the foundations of our method that is presented in section 4. Section 6 sets a discussion about different issues regarding our approach and section 7 concludes the paper.

2. Related Work and Main Contributions

There have been some other works proposing to model the data warehouse from a conceptual formalization of the domain. However, most of them must be carried out manually since they present a step-by-step guide to be followed by a data warehousing expert (see [25] for further details). Only a few of them automate this process somehow but the degree of automation achieved is rather low. Specifically, these approaches consist of a detailed requirement elicitation stage (to be performed manually) and an automated analysis of the data sources. Later, both stages are put in common, conciliating in this way the data sources and requirements (for instance, see [6, 11]). In these methods the requirement elicitation stage leads the process and main design decisions are captured in this step whereas the analysis of the data sources is rather superficial (i.e., design patterns described do not exploit the whole semantics of the multidimensional concepts). Indeed, the analysis of the data sources can be thought as a complementary task to the requirement elicitation processes described in these approaches. For instance, facts are mostly identified manually from requirements and the automation of the process is reduced to discover to-one relationships from each fact identified. To our knowledge, there is only one approach working at a conceptual level and following a similar framework to AMDO:

- Song et al. [26] present a method to semi-automate the data warehouse design process from ER diagrams. After a pre-process to transform the input diagram into a binary ER diagram (i.e., without ternary nor many-to-many relationships) they compute the number of to-one relationships a concept may have. According to this topological value, they identify facts and their potential dimensional concepts. This approach achieves a fair degree of automation and unlike previous approaches, proposes to identify facts in an automatic way, but the issue of how to automatically give rise to dimension hierarchies from the dimensional concepts identified is not addressed. The authors suggest a lexical method based on Wordnet and annotated dimensions. Finally, measures are overlooked as well and no clues about how to identify them are given.

Oppositely, the automation of the data warehouse design process from relational schemas has been thoroughly addressed in the literature [11, 12, 15, 20, 23]. These methods rely on a thorough analysis of the relational sources. In these cases, facts are discovered based on weak heuristics to identify facts (such as table cardinalities or numerical attributes that may identify fake facts or overlook real ones) whereas dimensional data is discovered by design patterns based on "foreign" (FK) and "candidate key" (CK) constraints. In multidimensional design, it is well-known that facts and dimensions must be related by many-to-one relationships, to give rise to a meaningful multidimensional space (see section 3 for further information). In a relational schema this kind of relationships (i.e., *mandatory* functional dependencies) are modeled by means of CK and FK constraints. For this reason the accuracy of results got by these methods depend on i) the decision to define CK and FK constraints (since some DBA may get rid of them to improve the insertion / deletion performance) and to ii) the degree of normalization of the logical schema (since some FK's and CK's are lost if we do not consider a schema up to 3NF):

- Golfarelli et al. [12] introduced a semi-automatable method to derive the multidimensional schema from relational schemas (indeed, they also propose to do it from ER diagrams, but tied to the same constraints discussed previously). Gathering requirements and mapping them onto the ER diagram is an important step in this approach. Therefore, the degree of automation is rather low and for instance, it demands to manually identify facts. Next steps can be performed semi-automatically: firstly, for each concept identified as a fact they build its *attribute tree*. Nevertheless, the automation of this step

completely relies on FK and CK constraints. Secondly, they identify measures, dimensions and their hierarchies by pruning and grafting this tree (in a semi-automatic way) and eventually, giving rise to the multidimensional schema. In an example like the one presented in figure 1 they would not be able to identify any dimensional concept automatically.

- Phipps and Davis [20] present a method largely automatable. In this approach they use a heuristic based on the number of numerical attributes a relation has to identify it as a potential fact (which they call fact relation), and any relation within a to-many relationship (identified by means of FK - CK constraints) is likely to play a dimensional role. To avoid their dependence on FK - CK constraints, any non-numerical concept within a fact relation is considered a dimensional concept (conforming a dimension by itself). Dimension hierarchies are deployed following FK - CK relationships.
  This approach uses heuristics and design patterns rather generic and it gives rise to results containing too much noise. Consequently, a post-process based on the user requirements is carried out to filter results. This post-process must be carried out manually and therefore, it would be hard, even for an expert, to filter and clean all results got. Regarding our example in figure 1, the `rental agreement` relation would be identified as a fact since it contains 6 numerical attributes. Later, all the non-numerical attributes (even those not making multidimensional sense) would be labeled as dimensional concepts and each one would be considered a dimension by itself (overlooking dimension hierarchies).

- In [23] we present our approach from relational sources. This approach requires to formalize the end-user multidimensional requirements into SQL queries over the relational sources so that we are able to complement the lack of semantics of a relational schema with external knowledge coming from the end-user. This information is integrated in the process and our approach is able to cope with denormalization up to some extend. In short, neither denormalization nor the lack of FK - CK definitions affect to the output if the proper multidimensional requirements are provided. However, a representative set of requirements must be provided and in any case, there are some questions that could not be answered. For instance, if two different dimensions are semantically related in the input requirements our method cannot distinguish if they are, indeed, two different dimensions or just one.

- Finally, Jensen et al. [15] present a method which complements the relational schema metadata with data from the instances. Assuming that the database does not contain composite keys, this method derives valuable metadata such as functional and inclusion dependencies and key information not stated in the schema. To do so, they take advantage of data mining techniques over the data sources. In short, they assume that a functional or inclusion dependency that holds in the data is assumed to hold in the schema as well. However, they completely rely on CK - FK patterns to identify functional and inclusion dependencies (i.e., two attributes of two different tables are checked to know if a to-one relationship holds between them despite a FK - CK relationship is not explicitly stated in the schema). Thus, the impact of denormalization is as big as in the rest of approaches, but they are able to identify CK - FK relationships overlooked in the schema. Finally, this approach can present computational problems for huge schemas due to the high number of permutations computed when looking for inclusion dependencies. In our example, this method would not be able to identify any of the multidimensional concepts in the `rental agreement` relation.

2.1. Main Contributions

Our proposal is a reengineering process to derive the multidimensional schema from a conceptual formalization of the domain. Working from conceptual formalizations improves the quality of the output, as discussed earlier in this section. Despite other works already proposed to work at a conceptual level, AMDO is the first method presented in the literature automating the whole process: i.e., identifying facts, measures and dimension hierarchies. Previous approaches mainly rely on their requirement elicitation stages to discover the multidimensional concepts rather on an accurate analysis of the data sources. In this sense, AMDO follows a completely different framework based on a thorough and fully automatic analysis of the sources and then, carrying out a guided requirement elicitation stage a posteriori, as discussed in section 4. Therefore, unlike previous approaches, the automatic analysis of the sources leads the process.

Moreover, AMDO considers all the multidimensional concepts in depth by analyzing their semantics and how they should be identified from the sources. As result we propose new and original design patterns. For instance, we are able to identify *aggregate measures* (see section 4.2 for further details) that have been completely overlooked in the literature; handle measures and dimensional concepts uniformly in an automatic way (see section 4.1.2); introduce formal rules to distinguish between *descriptors* and *levels* in a dimension hierarchy as well as identify semantic relationships between dimensions (see section 4.4), and a more accurate heuristic to discover facts than the ones used in previous approaches is provided (see section 4).

A possible reason why previous approaches working at a conceptual level have overlooked the automation of the process could be that ER (or UML) are conceptual formalizations thought to graphically represent the domain, but unlike ontologies, not thought for querying and reasoning. To our knowledge, our approach is the first one considering the data warehouse design from ontologies. Hence, we do believe that this work opens new interesting perspectives. For instance, we can extend the data warehouse and OLAP concepts to other areas like the Semantic Web, where ontologies play a key role providing a common vocabulary. One consequence would be that despite the data warehouse design has been typically guided by data available within the organization, we would be able to integrate external data from the web into our data warehouse to provide additional up-to-date information about our business domain (this novel concept of data warehousing is known in the literature as *Web-Warehousing* [22]).

Finally, the work presented in this paper is an evolution of the one introduced in [24]. We have improved our previous algorithms (both semantically and regarding its computational complexity) and moreover, the *AMDO tool* has now been devised and we have been able to extend our previous work including practical considerations and by providing a detailed example of a case study that shows the feasibility of our method.

3. Method Foundations

Since our goal is to generate multidimensional schemas in an automated way, this section aims to concisely define those criteria our proposal will be based on; i.e., those criteria allowing us to identify ontology concepts making multidimensional sense. Our method applies patterns over the input domain ontology to detect concepts that are able to be analyzed from a multidimensional perspective and therefore, able to give rise to a multidimensional schema. Concisely, multidimensionality pays attention to two main aspects; placement of data in a multidimensional space and correct summarizability of data:

- **[C1] The Multidimensional Model**: Multidimensionality is based on the fact/dimension dichotomy. **Dimensional concepts** give rise to the multidimensional space where the **fact** is placed. By **dimensional concepts** we refer to any concept likely to be used as a new perspective of analysis. Traditionally, they have been classified as **dimensions**, **levels** and **descriptors**. Thus, we consider a **dimension** to contain a hierarchy of **levels** representing different granularities (or levels of detail) to study data, and a **level** to contain **descriptors** (i.e., **level** attributes). On the other hand, a **fact** contains **measures** of analysis. One **fact** and several **dimensions** to analyze it give rise to a multidimensional schema.

- **[C2] The multidimensional space arrangement constraint**: **Dimensions** arrange the multidimensional space where the **fact** of study is depicted. Each instance of data is identified (i.e., placed in the multidimensional space) by a point in each of its analysis **dimensions**. Conceptually, it embraces that a **fact** must be related to each analysis **dimension** (and by extension, to any **dimensional concept**) by a many-to-one relationship. That is, every instance of the **fact** is related to, at least and at most, one instance of an analysis **dimension**, and every **dimension** instance may be related to many instances of the **fact**.

- **[C3] The summarization integrity constraint**: Data summarization performed must be correct, and we warrant this by means of the three necessary conditions (intuitively also sufficient) [17]: (1) **disjointness** (the sets of objects to be aggregated must be disjoint), (2) **completeness** (the union of

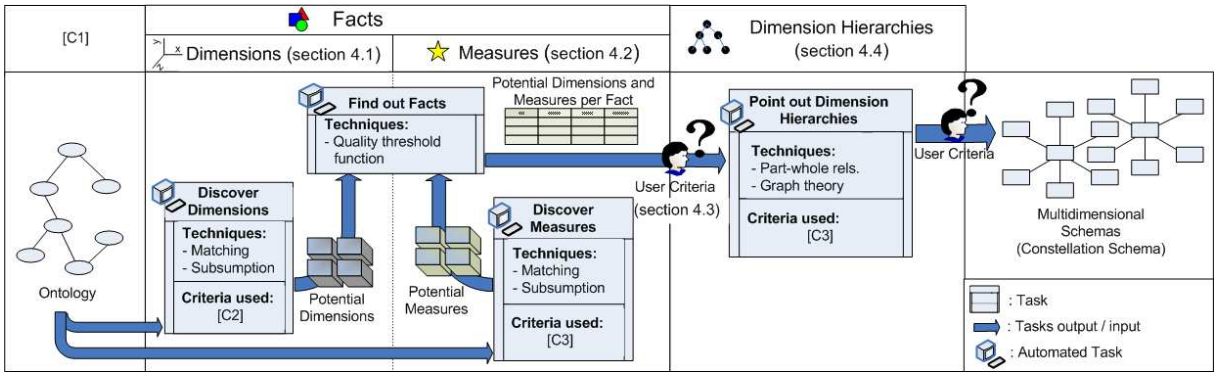

Figure 3: Method Overview

subsets must constitute the entire set), and (3) **compatibility** of the **dimension**, kind of **measure** being aggregated and the aggregation function. Compatibility must be satisfied since certain functions are incompatible with some **dimensions** and kind of measures. For instance, we cannot aggregate `Stock` over `Time` **dimension** by means of sum, as some repeated values would be counted. However, compatibility will not be automatically checked in our method unless additional metadata was provided (for instance, a list of compatibilities could be asked to the user for each **measure** identified).

4. Our Method

This section presents a detailed view of AMDO and how it applies the criteria exposed in section 3. Figure 3 depicts a schematic overview of AMDO. Along two well-differentiated tasks, it identifies concepts likely to play multidimensional roles [C1] and therefore, likely to give rise to multidimensional schemas:

- The first task looks for potential subjects of analysis (i.e., facts). In the literature we can find different approaches to discover facts but most of them are hardly automatable. Identifying facts automatically is a hard task [20], and most methods rely on heuristics such as table cardinalities or numerical attributes that may identify fake facts or overlook real ones. The rest of approaches demand to identify facts manually. According to the multidimensional paradigm, the analysis of data must facilitate the decision making within organizations and in this sense, the better knowledge you have, the better decisions you make. We say, thus, that an ontology concept is likely to play a fact role if it has as many measures as possible and it can be analyzed from as many different perspectives as possible. Eventually, this fact may not be of interest for the user (this will be considered later in our approach), but objectively, it will provide many different measures to analyze from many different perspectives.

  This task, therefore, is divided in two main subtasks; (1) discover potential dimensional concepts and (2) point out potential measures. The reader must notice that we do not talk about dimensions but about dimensional concepts. This step will find potential points of view to analyze the subject of analysis but, at this point, we are not able to distinguish their dimensional role (i.e., a dimension, a level or even a descriptor). This job is carried out in the next task of the algorithm, where dimension hierarchies will be shaped.

  Now, for each ontology concept we can estimate its likeliness of being a fact. In general, those concepts with most potential dimensional concepts and measures are good candidates, but we may weight each input according to our preferences. In our approach we define f as a function that, given the number of dimensional concepts and measures of a concept c, it evaluates c as a promising fact. This quality function will prune those concepts below a given *threshold*. This threshold will depend on the quality function provided (AMDO is not tied to any specific function). In fact, we can use any function we

would like to, like the "Connection Topology Value" (CTV) [26] introduced in the literature that only gives weight to dimensional concepts found, or develop our ad hoc formula. For instance, it would also be possible to rate facts according to the relevance of a concept in the application domain. Finally, potential facts not pruned are ranked according to its f value and presented to the user.

- The second task gives rise to dimension hierarchies. Among the dimensional concepts identified in the previous step we aim to identify dimensions of analysis. In other words, we aim to identify relevant aggregation paths looking for typical part-whole relationships. In this step, AMDO builds up graphs giving shape to each dimension hierarchy that the user may tune up to his / her needs.

AMDO carries out an exhaustive search of potential facts among all the concepts of the domain (in the literature, this framework is known as *supply-driven* [28]). This paradigm has a main benefit with regard to those approaches which derive the schema from requirements and later, map them onto the data sources (known as *demand-driven* [28]): in many real scenarios, the user may not be aware of all the potential analysis contained in the data sources and, therefore, overlook relevant knowledge. Demand-driven and hybrid approaches (those merging in the same framework supply-driven and demand-driven stages) do not consider this and assume that requirements are exhaustive. Thus, knowledge derived from the sources not depicted in the requirements is not considered and disregarded. In our approach, we claim to derive all the multidimensional knowledge contained in the ontology and let the user filter results got according to his / her requirements. On one hand, we are conciliating requirements with data available as hybrid approaches do. On the other hand, we believe that it is easier to carry out the requirement elicitation process from knowledge proposed by AMDO than carrying out it from scratch.

Specifically, after computing the likeliness of each concept as a fact, AMDO presents a ranked list of potential facts to the user (according to the quality function and threshold selected). For each concept, a value estimating its likeliness is provided. Moreover, if the user would like to, AMDO can show the list of potential measures and dimensional concepts computed for this fact. In the end, the user must select those relevant facts for his / her decision making. Later, for each fact, a list of dimensions (with their corresponding dimension hierarchies shown as a directed graph) is presented. The user may tune these graphs (i.e., delete a whole dimension, a level, a descriptor or split a proposed dimension into two different ones semantically related). As counterpart, supply driven approaches tend to generate too much results and mislead the user. In this sense, our approach overcomes this problem by minimizing the amount of data shown to the user (i.e., by means of the concepts of quality function and threshold). For further details, an example over a realistic case is shown in section 4.3.

Finally, the output of AMDO will be a multidimensional schema for each fact identified by the user and as a whole, we may get a *constellation schema* [16].

4.1. Discovering Dimensional Concepts

According to [C2], a dimensional concept is related to a fact by a one-to-many relationship; that is, every instance of factual data is related to one, and just one, of its instances. Hence, we can express our pattern to look for dimensional concepts as follows:

$$F \sqsubseteq = 1r.D, \quad where \quad r \equiv (r_1 \circ \ldots \circ r_n)$$

Notice it is expressed in "Description Logic" (DL) notation [3] (which OWL DL is based on), where r and D are variables, and F the ontology concept we are trying to identify as a potential fact. About the terminology used, we consider a *class* to be a unary predicate (i.e., D and F), and a *property* (i.e., r) as a binary predicate expressing a relationship between two classes. Briefly, the $\sqsubseteq$ symbol stands for *subsumption*, the basic inference on classes in DL. Subsumption (i.e., $A \sqsubseteq B$) is the problem of checking if the *subsumer* (B) is considered more general than the *subsumee* (A). That is, if the subsumee can always be considered a subset of the subsumer. $\equiv$ stands for a logic *equivalence* and can be defined as a specific kind of subsumption, that is: $A \sqsubseteq B$ and $B \sqsubseteq A$. $\circ$ stands for property *composition* (i.e., $\{a, c\} \in r \circ s$ iff $\exists b$ such that $\{a, b\} \in r$ and $\{b, c\} \in s$). Finally, $= 1$ stands for *functionality* that is a specific *number restriction* where, in our case, the number of individuals belonging to class D related to a given individual of the class F,

through the property r, must be exactly one. Thus, we are looking for classes (D) such that every instance of a given fact (F) is related, directly or by property composition (r), to, at least and at most, one of its instances. For each ontology class F we look for those classes fitting as its potential dimensional concepts of analysis evaluating the pattern presented above; where the dimensional concept is defined by the class D (from here on, the *ending* concept) and the set of properties r (from here on, the path of properties).

Definition 1. *A dimensional concept is defined by an ending concept and a path of properties. From a multidimensional point of view, the path must be considered because it adds relevant semantics. Two classes related by means of* n *different to-one paths must give rise to* n *different perspectives of analysis, since all these paths will potentially identify different sets of instances in the ending concept.*

For instance, consider the conceptual schema in figure 2. There, `rental agreement` has two to-one relationships to `branch` (i.e., `pickUpBranch` and `dropOffBranch`). Thus, {`branch`, `pickUpBranch`} and {`branch`, `dropOffBranch`} must be considered as two different points of view from where analyze a `rental agreement`, and the semantics of each dimensional concept identified is provided by the combined semantics of the path and the ending concept.

Unfortunately, we may not take advantage of generic DL reasoning algorithms for computing this pattern as most common reasoning services are not decidable when considering composite properties [3]. Moreover, it is not even expressible in OWL DL. Nevertheless, it is feasible to decompose this pattern as follows:

- First, for a given class $\mathcal{F}$, we look for its *direct* dimensional concepts.
- Next, we propagate this knowledge to compute the transitive closure of dimensional concepts according to the *transitive rule*:

  Definition 2. *If* {A, r} *(being* A *a class and* r *a property) is a dimensional concept of* B*, and* {C, r1} *is a dimensional concept of* A*, then* {C, r $\circ$ r1} *is a dimensional concept of* B *as well.*

The first step can be completely computed using generic algorithms provided by DL reasoners and the second step requires an adhoc algorithm that will also partially benefit from these algorithms.

4.1.1. Computing Direct Dimensional Concepts

Computing direct dimensional concepts is equivalent to consider r as a single property instead of a composite property in the pattern introduced in previous section:

$$F \sqsubseteq = 1r.D,$$

Where r is a single property. In our approach, we also consider OWL *datatypes* to play the role of D. Hence, a datatype may play a measure role (as it would seem more natural to think and we will discuss later) but also the role of an analysis dimensional concept. Handling facts and dimensions uniformly is not new. In fact, it was introduced by Agrawal et al. [1] and since then, it has also been proposed in many other design methodologies. That is the reason why, in the first iteration of our example, `Money` (`basicPrice` and `bestPrice`) and `Date` (`lastModification`) are considered potential dimensional concepts of `rental agreement`. A dimensional concept and a measure derived from the same datatype must be semantically related in the multidimensional schema (for instance, by the "equivalence" construct in OWL or by an "association" relationship in UML).

This pattern can be computed by basic reasoning (see section 6 for further details about basic reasoning in DL) and for each class F we keep track of pairs $\{D, \{r_1, ..., r_n\}\}$ (where each r_i is a property between F and D) which define its potential dimensional concepts. According to definition 1, each path (i.e., each r_i) will give rise to a different multidimensional concept. Using reasoning means that any assertion stated in the ontology (by using OWL DL constructs) are automatically considered. For instance, subsumption of classes, subsumption of properties, cardinality restrictions, functional (or inverse functional) properties, etc. Section 6 shows some statistics about the benefits of using reasoning regarding any adhoc algorithm.

4.1.2. Propagating Dimensional Concepts by Transitivity

In this section we present an adhoc algorithm to compute the transitive closure of dimensional concepts. Despite this algorithm cannot be fully computed by using reasoning services, we can take advantage of subsumption to propagate this knowledge through class taxonomies, as we will show later.
Our algorithm aims to build a matrix M of $N \times N$ elements (where N is the number of classes in the ontology) such that each row depicts a class and its potential dimensional concepts:

$$\forall\{D, \{r, ..., r_n\}\} \in M{<}F{>} \rightarrow \begin{cases} F \sqsubseteq = 1r.D, \\ \cdots \\ F \sqsubseteq = 1r_n.D \end{cases}$$

Where M is the $N \times N$ matrix, F and D are classes, r, ..., r_n are composite properties and $M{<}F{>}$ an operator over M that retrieves a list of classes related to F by, at least, a to-one path (i.e., its list of dimensional concepts). Each class in this list is represented as $\{D, \{r, ..., r_n\}\}$ where D is the class (or datatype) itself and each r_i is a to-one path depicted as a composite property. Therefore, we may derive as many dimensional concepts from D as different paths it has. Roughly speaking, each cell $C_{(F,D)}$ of M contains a list of composite properties ($\{r, ..., r_n\}$) such that each instance of F is related at least and at most to one instance of D.

Along this step we aim to build this matrix, and we achieve so by means of the next algorithm:

typedef list <properties> path
typedef tuple < concept, **list**<path> > paths_to_concept
typedef tuple < concept, **list**<paths_to_concept> > to-one_rels

function*create_matrix* **returns** Matrix

1. **vector**< list<to-one_rels> > M;
2. initialize(M);
3. compute_trivial_deadlocks(M, ontology);
4. first_iteration(M, ontology);
5. propagate_path(M, converge);
6. **return** M;

Figure 4: An algorithm to deploy matrix M

Since M is a sparse matrix, the function *create matrix* implements it as a *vector* of *lists* (step 1). That is, every position in the vector represents a class and its list of potential dimensional concepts (see the $to-one\_rels$ *typedef* declaration). Lists are created and initialized to the empty list in step 2. Step 3 finds and breaks trivial deadlocks. The need of this step will be justified later in this section.

Step 4 corresponds to the pattern presented in section 4.1.1. Each potential dimensional concept identified is added to the proper list in the vector. Figure 6.1 (depicted like a matrix) shows results got after step 4 for some of the concepts introduced in figure 2. For instance, the class `maintenance scheduled` has a to-one relationship to `date` (through the `dateScheduled`, `acquisitionDate` and `lastMaintenanceDate` properties), `service depot` (through the `In` property), `branch` (through the `isAvailable` and `isResponsibleFor` properties), `car model` (through the `isOf` property), `boolean` (through the `available` property) and `double` (through the `currentMilleage` and `milleageFromLastService` properties). The user may notice that some of these classes (or datatypes) may play more than one dimensional role regarding `maintenance schedule` (i.e., there are more than one path between both classes -or datatypes-). Moreover, some of the properties here described are inherited from its superclasses.

Step 5 propagates the dimensional concepts identified in the previous step according to the transitive rule (see definition 2). We show its feasibility by introducing the *propagate_path* function (see figure 5):

void *propagate_path* (Matrix ↕M)

7. **list** <paths_to_concept> ending_concepts;
8. **foreach** C **in** M **do**
   (a) **if** M_c<C> = ∅ **then**
      i. $C.closed$:= true;
9. **do** {
10. bool conceptsClosed := false;
11. **foreach** C **not closed in** M **do**
   (a) reachable_concepts := M_c<C>;
   (b) **foreach** D **in** reachable_concepts **such that** !M<C,D>.$treated$ **do**
      i. **if** $D.closed$ **then**
         A. **foreach** r **in** M_p<C,D> **do**
         B. listEls := listEls ∪ (r ∘ M<D>);
         C. M<C> := M<C> ∪ listEls;
         D. M<C,D>.$treated$:= true;
   (c) **if** all_closed(reachable_concepts) **then**
      i. **list** <concept> parents := compute_direct_superconcepts(C, M);
      ii. **if** all_closed(parents) **then**
         A. **foreach** P **in** parents **do**
         B. M<C> := M<C> ∪ M<P>;
         C. $C.closed$:= true;
         D. conceptsClosed := true;
12. if(!conceptsClosed)
   (a) break_deadlocks(M);
13. } **while** concepts_not_closed(M) > 0

Figure 5: An algorithm to propagate dimensional concepts by transitivity

This function implements a smart algorithm to compute this step. Essentially, the list of dimensional concepts of each class is propagated only once when we know that it cannot vary. To do so, dimensional concepts are propagated from the end of the to-one paths (from here on, *leaf classes*) to the beginning, according to the definition of *closed class*:

Definition 3. *We say a* class C is closed *or that a given class C closes in the* ith *iteration of our algorithm* Closed(C,i)*, if all its dimensional concepts have been computed in* i *or any iteration previous to* i*. Said in other words, if a class C closes in a certain iteration* i*, no other dimensional concept will be identified for C in any iteration* j *such that* i < j*. In our notation, we define* Closed(C,i) *as a recursive function:*

$$Closed(C,i) := \forall\{D, \{r, ..., r_n\}\} \in M<C> : Closed(D, j) \wedge j < i,$$

Our algorithm only *propagates* closed classes (see the *closed* method in the algorithm). If a given class C closes in the i^{th} iteration of the algorithm, we propagate its dimensional concepts in the $i+1^{th}$ iteration (once propagated, it is never considered again thanks to the *treated* method. The reader may notice that the treated method does not hold at a class level but as dimensional concept -i.e., regarding C: M<C,D>.$treated$-). Thus, our algorithm aims to identify closed classes and propagate them just once. Propagating knowledge is done in two different ways:

- Let C and D be two classes such that D is in the dimensional concepts list of C (see step 11b). Then, the dimensional concepts list of D is propagated to C by transitivity according to definition 2 (see step 11(b)iA):

$$\forall D \in M_c{<}C{>}, \forall D_i \in M_c{<}D{>} \rightarrow \{D_i, \{M_p{<}C, D{>} \circ M_p{<}D, D_i{>}\}\} \in M_p{<}C, D_i{>},$$

Where $M_c{<}C{>}$ and $M_p{<}C, D{>}$ are two operators over matrix M. The first one retrieves the list of classes in the dimensional concept list of a given class C (i.e., it is equivalent to the operator $M{<}C{>}$ but overlooking the path lists), and the second one retrieves the list of paths between two classes C and D such that D is in the dimensional concept list of C (i.e., it retrieves the path information between C and D in $M{<}C{>}$). D_i are the set of classes in the dimensional concepts list of a concept D (i.e., $\forall D_i,\ D_i \in M_c{<}D{>}$) and $\{M{<}C, D{>} \circ M{<}D, D_i{>}\}$ represents the concatenation of each path in $M{<}C, D{>}$ with each path in $M{<}D, D_i{>}$ (for the sake of readability, this formalization is depicted with an slight abuse of notation in step 11(b)iB of the algorithm). Roughly speaking, we are concatenating each to-one path from C to D with each to-one path from D to D_i (see step 11(b)i).

- Let P be a class such that P is a parent (i.e., a direct superclass) of C. Then, all the dimensional concepts of P must be inherited by C (i.e., $M{<}C{>} := M{<}C{>} \cup M{<}P{>}$, see step 11(c)iiB). In our algorithm, this kind of propagation is done when all the dimensional concepts of C have closed (see step 11c). Then, we can take advantage of DL basic reasoning (see section 6 for further details) to compute the list of superclasses to propagate. Moreover, we do not propagate them until they have closed to avoid propagating more than once (see step 11(c)ii).

Finally, how closed classes are detected can be summarized as follows:

- First iteration: Leaf classes and datatypes (as discussed in section 4.1.1, our algorithm considers the datatypes as potential dimensional concepts and in this sense, they can be considered leaf classes) close in this iteration (see step 8).
- Second iteration: Classes closed in the previous iteration are now propagated. In this case, propagating them is trivial, since their lists of dimensional concepts are empty. Now, according to our definition of closed class, any class whose dimensional concepts have closed (and therefore, already propagated), closes in this step.
- N^{th} Iteration: A given class C will close in this iteration if the last class to close in its list of dimensional concepts have already closed in the $(n\text{-}1)^{th}$ iteration (see step 11c). Therefore, all the dimensional concepts of C have already been computed and we can now propagate C in the next iteration (see step 11(c)iiC).

Following our example, figure 6.1 shows direct dimensional concepts identified for each ontology class and figure 6.2 depicts how we propagate them by transitivity. For instance, `rental agreement` is related by two to-one relationships to `branch` (**bolded** in the figure), and `branch` is related by to-one relationships to `country`, `string`, `branch type` and `service depot`. Hence, the latter are also considered dimensional concepts of `rental agreement` according to the transitivity rule (see the arrow in figure 6.2). Moreover, notice that the path list of these newly identified dimensional concepts have been properly updated when adding them to the list of `rental agreement`. For instance, we can get from `rental agreement` to `branch` by two different paths (i.e., `pickUpBranch` and `dropOffBranch`) and from `branch` to `country` by the `locatedAt` property. Therefore, from `rental agreement` we can get to `country` through the composition of `pickUpBranch` and `locatedAt`, and `dropOffBranch` and `locatedAt`. Analogously for the rest of dimensional concepts.

When detecting closed classes it is important to detect potential *deadlocks* due to cycles of to-one properties between classes. When a cycle is detected (in our algorithm, when no class closes in the current iteration; see step 12) it is broken propagating just once among them (and therefore, sharing all their potential dimensional concepts). Moreover, this situation will be notified to the user to let him / her know that each recurrent propagation within the cycle may add new interesting semantics (i.e., new analysis dimensional concepts) that could be considered. The most common and also easiest case to detect are one-to-one and reflexive relationships. To facilitate the process, AMDO treats these two basic cases immediately after being identified (see step 3 of figure 4). Detecting trivial deadlocks can be computed by reasoning (see section 6 for further details) and we may use any of the current algorithms presented in the graph theory to

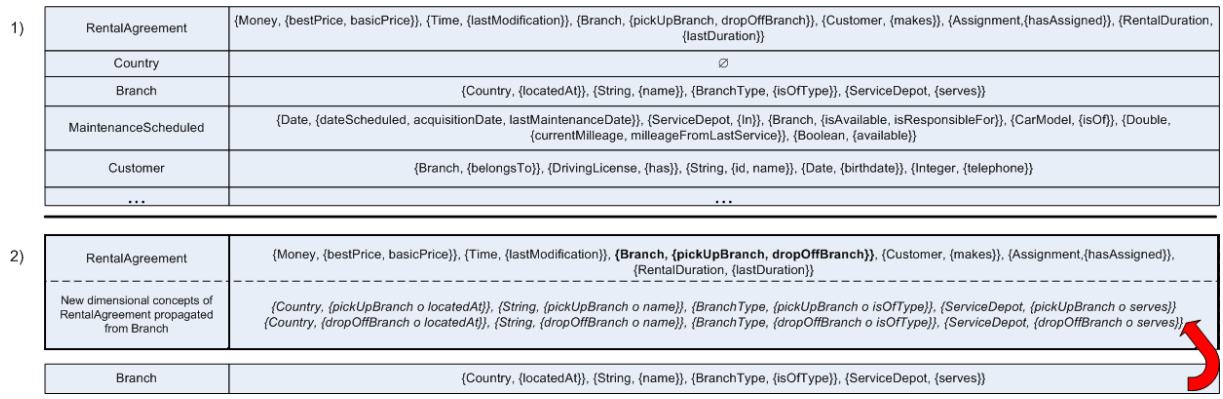

Figure 6: Example of propagation of to-one paths by transitivity

detect general cycles. For instance, a *depth-first-search* (DFS) remembering previous visited nodes would fit properly. For instance, `customer` and `driving license` through the one-to-one `has` property is computed in the first iteration. At this point, we are able to compute its list of dimensional concepts breaking up the deadlock (i.e., do not apply the transitive rule over it) and notifying to the user that, in case s/he would be interested, it is possible to derive infinite new dimensional concepts just following the cycle semantics.

4.1.3. Complexity of the Algorithm

The computational cost of this algorithm has $\Theta(N \times c^l)$ as upper bound; where N is the number of classes in the ontology; c the maximum to-one *connectivity* (i.e., direct to-one relationships from a class) and l the maximum chain of to-one properties. However, this upper bound is theoretical and hardly achievable in practice since real ontologies neither have all classes with maximum to-one connectivity nor all to-one paths are of maximum length. Moreover, along the process, classes computed in previous iterations are not considered in the forthcoming ones.

In practice, the computational complexity raised by AMDO is polynomial for most ontologies. For instance, consider the EU-Car Rental ontology (see figure 2). The whole EU-Car Rental ontology has 65 classes and 170 properties (or relationships) of which 94 properties are between classes (30 of them are subsumption assertions) and 76 are properties among classes and datatypes. The maximum to-one connectivity (i.e., c) is 21 (raised by `LateReturn`). In the worst case (i.e., assuming the practical consideration introduced in section 5), the longest to-one path has length 5. Consequently, the theoretical upper bound for this simulation would be $\Theta(65 \times 21^5)$.

However, using the AMDO tool, the execution of the algorithm was immediate (less than one second in a regular desktop computer). The algorithm converges in just 5 iterations: closing 2 classes before starting (i.e., besides datatypes, there are two classes with empty list of dimensional concepts), 12 classes in the first iteration, 13 in the second one, 19 in the third one, 15 in the fourth one and 4 in the last one. In each iteration, only some classes are propagated and those previously propagated are not computed again, so, a better estimation of the answer time would be:

$$\sum_{i=1}^{l} N_i \times c_i$$

Where N_i is the number of classes not yet closed (i.e., that still have to be considered) in that iteration, c_i the maximum functional connectivity in that iteration and l the number of iterations (i.e., the size of the bigger to-one path in the ontology). In our example, it would raise:

$$\sum_{i=1}^{5} N_i \times c_i = (63 \times 21) + (51 \times 24) + (38 \times 54) + (19 \times 87) + (4 \times 109)$$

Result got, drastically smaller than the theoretical upper bound, is still an upper bound of the answer time of AMDO. Notice that we are considering the maximum connectivity for each class in each iteration, something completely false. However, we want to underline some important things depicted in this formula: on one hand, we may appreciate that the value of N_i is strictly decreasing and, on the other hand, the value of c_i is never exponential. In fact, in the last iteration, its value is 109, far away from 21^5. All in all, despite the EU-Car Rental ontology size, AMDO behaves well and the answer time is good enough to develop an interactive tool.

4.1.4. Soundness and Completeness of the Algorithm

Our algorithm is clearly sound, since it computes direct to-one relationships and propagates them according to the transitivity rule presented in section 4.1.2.

Our algorithm is complete if we can assure that it converges; that is, if it would fully explore each to-one path (starting from the *end*, by identifying leaf classes, and going through the paths up to the *beginning*). We can say so if we can assure that if in a given iteration the vector M is not updated then, in any of the following iterations it will not be updated either. It can be guaranteed because:

- We detect and break deadlocks and,
- in the worst case, if P is the maximum number of to-one properties chained in the ontology, in each iteration the *propagate_path* function (see step 5 of figure 4) will propagate, at least, one property. Indeed, the *invariant* of the main loop of the algorithm (see step 9 of figure 5) guarantees that the length of each to-one path explored up to current iteration is *strictly increasing* and at most, in P iterations we would have explored (and propagated) all chained to-one properties in the ontology. Thus, step 5 will not be able to propagate any other property in next iterations.

4.2. Identifying Measures

In this step we look for measures (i.e., factual data). Typically, measures are numeric attributes allowing data aggregation. AMDO considers any summarizable *datatype* (i.e., those allowing data aggregation by its own nature) to be a measure of a given fact F if, according to [C3], it preserves a correct data aggregation from F; that is, if they are conceptually related by a one-to-one relationship. (1) The to-one multiplicity in the measure side enforces that each fact instance is related to just one measure value, and forbidding zeros we preserve *completeness*, (2) whereas the to-one multiplicity in the fact side preserves *disjointness* (i.e., a measure is related to one and just one fact).

It is important to remark that our definition of measure is wider than the definition used by previous approaches. Previous approaches identify them among class / entity numeric attributes (in case of working over ER or UML diagrams) or among numeric attributes of relations (in case of working over relational schemas). In our framework (i.e., from an ontology) the definition of measure according to previous approaches would be equivalent to only consider datatypes directly related to classes. Oppositely, our definition is not restricted to direct datatypes but to any (i.e., to any reachable by means of property composition) preserving the one-to-one relationship demanded. This kind of measures (from here on, *aggregate measures*) have been overlooked in the literature but identifying them is important to discover meaningful and additional factual data.

Figure 7 shows the two kind of relationships we look for. 1) The first figure depicts a measure directly related to the fact. This pattern is the equivalent one in OWL to those used by previous approaches in

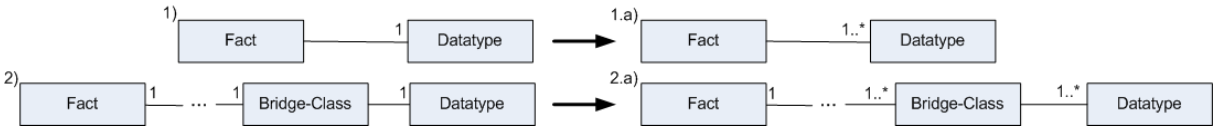

Figure 7: Multiplicities looked for

the relational model or UML / ER diagrams. A datatype does not have an *object identifier* (i.e., *oid*) and we allow any multiplicity in the fact side without violating *disjointness*. For instance, in our example the `bestPrice` and `basicPrice` datatypes related to `rental agreement` would follow this pattern. As a particular scenario, 1.a) we may accept mandatory multivalued datatypes (i.e., each class may be related to at least one value and at most, many of them) as well. In this case, those many values related to each fact instance should be aggregated by means of a *compatible* aggregation function (see [C3]) prior to be inserted in the data warehouse once built. AMDO will provide this information to be taken into account during the ETL process.

The next one is used to discover aggregate measures. 2) The second pattern depicts a class (from here on, a *bridge-class*) that is both directly related to a datatype and by means of a one-to-one path (i.e., it may contain property composition), to a fact. This datatype is also a potential measure for that fact since *disjointness* and *completeness* are guaranteed. In this case, similar to the scenario discussed in the previous pattern, 2.a) along the path between the fact and the bridge-class it is enough to ask for a mandatory relationship. Again, all those (many) measures values related to each fact instance must be aggregated by a *compatible* aggregation function, prior to be inserted in the data warehouse. Therefore, [C2] is also guaranteed in this pattern. For instance, in our example the `bestPrice` and `basicPrice` per `customer` and `branch` would follow this pattern. In this case the fact would be `branch`, the bridge-class would be `rental agreement` and `bestPrice` or `basicPrice` the datatype related to the bridge-class. Depending on the aggregation function used, we can derived different aggregate measures. If we use the `min` operator we would get the best `bestPrice` (or `basicPrice`) offered to a `customer` per `branch`; using the `avg` operator we would get the average `bestPrice` (or `basicPrice`) offered to a `customer` per `branch` and so on. These patterns discussed above may be captured by the following algorithm:

- (Pattern 1) For each class C, look for summarizable datatypes *directly* related to it. In OWL it is equivalent to look for those mandatory to-one properties such that their *domain* is C and their *range* are datatypes. We can take advantage of basic reasoning to compute this pattern (see section 6 for further details):

  $$C \sqsubseteq = 1r.dt,$$

  Where r is a property and dt any OWL datatype allowing aggregation of data (for instance, *int*).

  - (Pattern 1a) For considering mandatory multivalued datatypes we must consider any mandatory property such that its *domain* is C and its *range* is a datatype:

    $$C \sqsubseteq \exists r.dt,$$

- (Pattern 2) This pattern can be directly computed by matrix M (see section 4.1.2). A datatype D, directly related to a class B, is a potential measure for a class F if B is a bridge-class for F. Analogously to definition 1, each property (i.e., path) between F and dt will give rise to a measure:

  $$\text{Measure}(F,\{dt, r_1, ..., r_n\}) := \exists B, \exists r_1, ..., r_n \mid \forall r_i, 1 \leq i \leq n,\ B \sqsubseteq \exists r_i.dt \wedge (F \in M_c{<}B{>} \wedge B \in M_c{<}F{>})$$

  If so, it means that, by transitivity, we have been able to identify B as a dimensional concept of F and viceversa (i.e., each r_i is a one-to-one path between them).

  - (Pattern 2a) To compute this pattern we need to consider matrix M_1 (of $N \times N$ elements, where N is the number of classes in the ontology). This matrix represents, for each class F, the list of classes we may get to by means of *mandatory paths*. Said in other words, if every instance of F is related to, at least, one instance of the ending concept. Thus, analogously to the definition of matrix M, each row of M_1 can be defined as follows:

$$\forall\{D,\{r,...,r_n\}\} \in M_1{<}F{>} \rightarrow \begin{cases} F \sqsubseteq \exists r.D, \\ \cdots \\ F \sqsubseteq \exists r_n.D \end{cases}$$

Where M_1 is the $N \times N$ matrix, F and D are classes, r and r_n are composite properties and $M_1{<}F{>}$ an operator over M_1 that retrieves a list of classes related to F by, at least, a mandatory (i.e., `1..N`) path. Like in the definition of matrix M, each class in this list is represented as $\{D,\{r,...,r_n\}\}$ where D is the class (or datatype) itself and each r_i is a mandatory path depicted as a composite property. Now, we can formally describe this pattern as:

$$\text{Measure}(F,\{dt,r_1,...,r_n\}) := \exists B,\exists r_1,...,r_n \mid \forall r_i, 1 \leq i \leq n, B \sqsubseteq \exists r_i.dt \wedge (\text{F} \in M_c{<}B{>} \wedge B \in M_{1c}{<}F{>})$$

Where $M_{1c}{<}B{>}$ is the equivalent operator of $M_c{<}B{>}$ for M_1. Matrix M_1 can be computed with an algorithm analogous to the one presented in section 4.1.2 (see figure 4) to compute M, but instead of looking for direct to-one relationships in step 4, we will look for direct mandatory relationships:

$$F \sqsubseteq \ \exists r.D,$$

Where r is a single property. The rest of the algorithm (i.e., propagating this knowledge) remains the same. The addition of matrix M_1 do not modify the overall computation complexity. Indeed, the computational cost of M_1 is analogous to that of M and it has $\Theta(N \times c^l)$ as upper bound; where N is the number of classes in the ontology; c the maximum mandatory *connectivity* (i.e., mandatory relationships of a class) and l the maximum chain of mandatory properties in the ontology. Similarly, the same practical considerations discussed in section 4.1.2, as well as considerations about the soundness and completeness of the algorithm can be considered here.

4.3. User Interaction

Once potential dimensional concepts and measures of analysis have been computed for each class, AMDO presents the final result to the user. Results got are ordered according to a function f that evaluates how good candidate each class is. Those values not reaching a pre-defined quality threshold are directly pruned and not presented to the user.

Table below summarizes the information presented to the user at the end of this step. There, each class is followed by its number of potential dimensional concepts and measures identified along the process. In our case, results are ordered according to the `FactEstimation` function (`FactEstimation` := $(M^*2 + DC)$ if M is bigger or equal to one or 0 otherwise, where M is the number of measures and DC the number of dimensional concepts identified for that class). As discussed in section 4, AMDO is completely flexible in this issue and the user is able to provide any function which meets his / her requirements better. Next, we present those classes AMDO would propose if our treshold regarding the `FactEstimation` function was 80:

Concept	#**Dimensional Concepts**	#**Potential Measures**	**FactEstimation**
DamageCost	81	3	90
LateReturn	78	5	88
Prepared	81	3	87
AssignedCar	80	3	86
PaidWithPointsRental	74	4	82
ClosedRental	74	4	82
EarlyReturn	74	4	82

In general, those classes in the `rental agreement` taxonomy are the best candidates to play a fact role, and *subclasses* are better rated than *superclasses*. This result is sound, since we should expect `rental agreement` to be the keystone concept of a rental service. Classes in the list which does not belong to the `rental agreement` taxonomy are just three: `damage cost` (1st place), `prepared` (3rd place) and `assigned car` (4th place). However, all these classes are closely related to the `rental agreement` taxonomy and `prepared` is subclass of `assigned car`.

At this point, among results shown, the user should select those facts of interest to him / her (normally, more than one). So that, this decision is up to the user requirements.

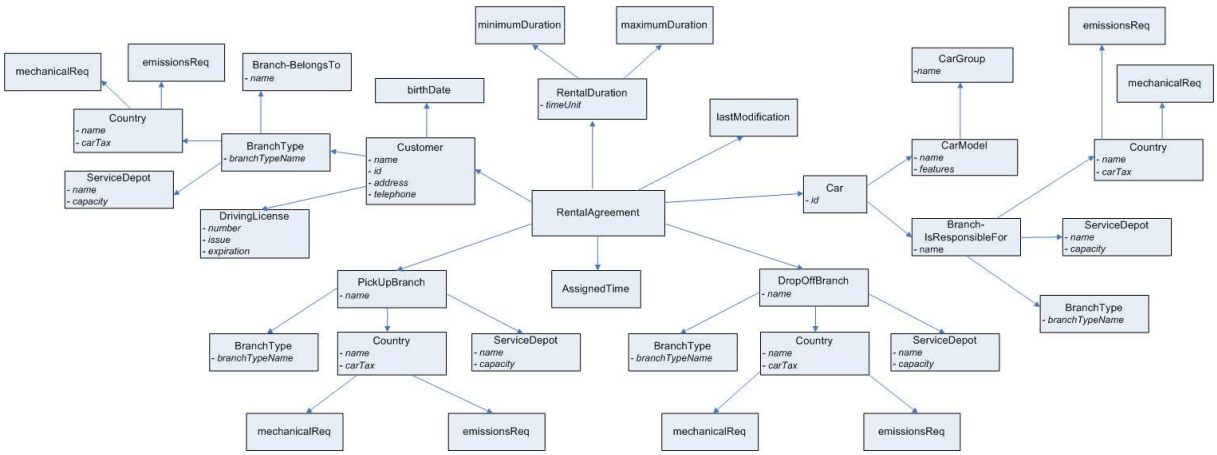

Figure 8: Resulting multidimensional schema

4.4. Giving rise to Dimension hierarchies

In the previous step we have identified dimensional concepts of interest. However, we still need to shape their hierarchies in order to allow summarizability of data; one of the multidimensionality principles. Dimension hierarchies must guarantee a correct summarizability of data (see [C3]). In this step we look for to-one relationships (also known as "roll-up" relationships) giving rise to hierarchies allowing a correct data aggregation: the to-one multiplicity preserves *disjointness* of aggregated data, and forbidding zeros we also preserve its *completeness*. Starting from each class identified as an ending concept of a dimension, a directed graph following all to-one relationships paths is depicted, using the matrix M built in section 4.1. Notice that, at this moment, we cannot differentiate the role played either as a level or as a descriptor by each graph node within the dimension hierarchy. However, two specific rules (considered in the following algorithm) can give us more information about it:

- (1) If any class (C_3) is placed in more than one graph, we consider it to be a level (since it seems interesting to show data at this granularity level). Following with DL notation we could formalize this pattern as:

$$\exists C_1, C_2, \exists r_1, r_2 \mid (\exists r_1 \sqsubseteq C_1) \sqcap (\exists r_1^- \sqsubseteq C_3) \sqcap (\exists r_2 \sqsubseteq C_2) \sqcap (\exists r_2^- \sqsubseteq C_3)$$

  Where C_1 and C_2 are classes and r_1 and r_2 are disjoint paths through graph edges (i.e., the set of classes typing each path is disjoint. Said in other words, there is not any class *visited* by both paths). Intuitively, we are looking for classes placed in two different graphs. Eventually, these classes will give rise to two different levels in each graph. Both levels, though, must be related semantically to denote their conceptual relationship.

- (2) If two classes C_1 and C_2 are related by means of a one-to-one relationship in any graph:

$$\exists r_1 \mid (C_1 \sqsubseteq = 1r_1.C_2) \sqcap (C_2 \sqsubseteq = 1r_1^-.C_1)$$

  it depicts either a semantic relationship between two different dimensions (if the *ending* class -notice the graph is directed- was identified as a dimension) or an attribute level (i.e., a descriptor) if the ending class is not already identified as a dimension. In the latter case we consider it to be a descriptor of the *initial* concept. Finally, in the first case both concepts must be properly related by semantic relationships.

Directed graphs deployed are presented to the user as dimension hierarchies, altogether with those semantic relationships between dimensions pointed out. With these premises, in some cases we have not

been able to identify each graph node either as a level or a descriptor. However, this is sound, since it is up to a design decision to spot each class as an attribute of an existing level or as a new level; giving rise to implicit or explicit dimension hierarchies in the resulting schema. Consequently, this differentiation should be made by the user, if s/he is interested in aggregating data at this level, when stating his/her requirements.

Finally, the user is also asked to reshape analysis dimensions derived from datatypes. AMDO does not propose any hierarchy for these dimensions automatically, but we allow the user to use predefined functions (for instance, *get_day*, *get_month* or *get_year* in case of dates) or aggregate data to give rise to value ranges in case of numerical datatypes (for instance, from 0 to 20, 21 to 50, 51 to 99 in case of ages) in order to create ad hoc dimension hierarchies.

For instance, if in section 4.3 the user would have chosen `rental agreement` as an interesting subject of analysis, AMDO would generate the dimension hierarchies shown in figure 8. There, each arrow starting from `rental agreement` depicts a dimension hierarchy. Concepts identified as descriptors by AMDO are depicted in *italics*. In total we give rise to 9 directed graphs (starting from `customer`, `lastModification`, `assigment`, `dropOffBranch`, `pickUpBranch`, `rentalDuration`, `car`, `bestPrice` and `basicPrice`). Finally, the user must tune up dimension hierarchies got up to his necessities. For instance, by considering some descriptors as interesting aggregation levels (it is the case of `minimumDuration` and `maximumDuration` that were identified as descriptors by AMDO but we have considered to be interesting enough to represent levels of aggregation) or by dropping dimensions of no interest (in our case we have not draw in the figure `bestPrice` and `basicPrice`). It is important to remark that all those levels derived from the same class in different graphs would be related by semantic relationships. For instance, `country` or `branch type`.

The computational cost of this step is negligible since matrix M is already calculated and we only need to navigate and explore it.

5. Additional Practical Considerations

In this section we would like to discuss why we force the fact instances to be related to at least and at most to one instance of a dimensional concept (see section 4.1). In practice, it would be possible to relax this relationship allowing zeros (i.e., fact instances not related to any instance of a dimensional concept and automatically create a dummy dimension concept instance (for instance, named `others`) related to those fact instances not related to any instance of the dimensional concept. Then, our pattern to look for dimensional concepts would look like as follows:

$$F \sqsubseteq \leq 1r.D, \quad where \quad r \equiv (r_1 \circ \ldots \circ r_n)$$

In the AMDO tool we introduced an option to allow the user choose if s/he wants to enforce this restriction or relax it due to practical considerations in the domain ontology. The algorithm and its computational cost would not change, and we would only need to consider to-one relationships allowing zeros in matrix M. In practice, it entails to consider some more direct dimensional concepts in step 4.1.1 and therefore, some more propagations when computing the transitive closure of dimensional concepts.

Analogously, we can extend this practical consideration for the pattern looking for measures as well. We could relax this pattern to allow zeros in the bridge-class directly related to the datatype (i.e., not forcing each fact to have a numerical value for that measure). In this case, if the user selects this measure we would need to introduce an specialization of that measure in the data warehouse conceptual schema to preserve completeness. About the algorithm used, in practice, these considerations would entail that we do not need anymore matrix M_1. Using matrix M we will be able to compute if a class F can use a given class B as bridge-class (i.e., $F \in M_c{<}B{>}$) as any multiplicity would be allowed in the bridge-class side of the relationship (see figure 7). Once computed, we will be able to know if we need to aggregate data through a *compatible* aggregation function if $B \notin M_c{<}F{>}$.

Finally, we strongly recommend to enforce the theoretical patterns presented as much as possible. Relaxing them may entail the identification of meaningless dimensions or give rise to sparser multidimensional spaces, which may mislead the user.

6. Discussion

AMDO's output is produced in a fully automatic way and it benefits from DL reasoning techniques provided by DL reasoners. Indeed, most of our patterns can be reduced to basic reasoning tasks that any commercial reasoner available in the market could answer by its querying services. In our implementation we have used FaCT [14]. This reasoner supports OWL DL except for nomimals, but nominals fall out of our needs and do not affect our reasoning tasks. FaCT provides basic tasks such as discover *class taxonomies* (i.e., given a class find all its subclasses, superclasses, ancestors or successors), *property taxonomies* (analogous to class taxonomies reasoning but over properties) and *subsumption* (given two OWL DL assertions say if one is subsumed by the the other). Any of the reasoning tasks mentioned along this paper can be reduced to this three query services. Using DL reasoning have considerably reduced the complexity of our task and have facilitated the whole automation of AMDO. Indeed, most of the work done in AMDO have been reduced to reasoning over FaCT. For instance, consider the EU-Car Rental ontology used as example along the paper (see figure 2). With AMDO we have been able to identify 2069 dimensional concepts (the reader may notice that it does not mean that the produced multidimensional schema contains 2069 dimensional concepts but that the number of dimensional concepts identified regarding all the ontology classes is 2069 -most of them disregarded when choosing facts of interest; see section 4.3-). 1375 out of this 2069 dimensional concepts were identified using reasoning (i.e., querying FaCT) while 694 out of 2069 were identified by our adhoc algorithm.

Nevertheless, it would be possible to increase the number of patterns presented in our method that were computed by reasoning. It could be achieved by using a less expressive ontology language as AMDO input. One of the most promising ones is DL-Lite [21] which is a well-behaved DL with polynomial reasoning over the TBox. DL-Lite does not capture the whole semantics of ER or UML class diagrams, but it does capture the most important features of it and in addition, it would allow to increase the number of patterns computed by reasoning. DL-Lite would be a future alternative to explore for introducing better and possibly more expressive design patterns (more expressive with regard to the multidimensional model but less expressive and with less inference power in the general case) that may raise an unaffordable computational complexity for OWL DL but work well for DL-Lite.

7. Conclusions

In this paper we have presented AMDO: an automatic method to identify concepts likely playing multidimensional roles, from an ontology representing our business domain. In our approach, we have presented a set of patterns to identify the multidimensional concepts that in most cases can be computed by generic DL reasoning algorithms. Moreover, we have also discussed the theoretical computational complexity of our algorithms and thanks to the AMDO tool, we have also been able to discuss its behavior with real world ontologies in which turns to have a polynomial computational complexity.

We believe this work to be the first to address the issue of automating the multidimensional design from ontologies. Up to now, traditional approaches were typically carried out manually or were designed to work in an automatic way over relational sources. In our approach, AMDO carries out the data warehouse design process from a domain ontology (i.e., at a conceptual level) which improves the quality of the multidimensional schemas automatically generated. Furthermore, working over ontologies opens new interesting perspectives. For instance, we can extend the data warehouse and OLAP concepts to other areas like the Semantic Web and one consequence would be that despite the data warehouse design has been typically guided by data available within the organization, we would be able to integrate external data from the web into our data warehouse to provide additional up-to-date information about our business domain.

References

[1] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimensional Databases. In *Proc. of the 13th Int. Conf. on Data Engineering*, pages 232–243. IEEE, 1997.

[2] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. Reasoning over extended er models. In *Proc. of 26th Int. Conf. on Conceptual Modeling*, volume 4801 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2007.
[3] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
[4] G. Berardi, D. Calvanese, and D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.
[5] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5), 2001.
[6] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, and S. Paraboschi. Designing Data Marts for Data Warehouses. *ACM Trans. Softw. Eng. Methodol.*, 10(4):452–483, 2001.
[7] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. A formal framework for reasoning on uml class diagrams. In *Proc. of 11th Int. Symposium on Foundations of Intelligent Systems*, volume 2366 of *LNCS*, pages 503–513. Springer, 2002.
[8] P. P.-S. S. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
[9] L. Frías, A. Queralt, and A. Olivé. EU-Rent Car Rentals Specification. Technical report, 2003.
[10] D. GaŽevic, D. Djuric, and V. Devedđic. MDA-based Automatic OWL Ontology Development. *Int. Journal on Software Tools for Technology Transfer*, 9(2):103–117, 2007.
[11] P. Giorgini, S. Rizzi, and M. Garzetti. Goal-oriented Requirement Analysis for Data Warehouse Design. In *Proc. of 8th Int. Workshop on Data Warehousing and OLAP*, pages 47–56. ACM Press, 2005.
[12] M. Golfarelli, D. Maio, and S. Rizzi. The Dimensional Fact Model: A Conceptual Model for Data Warehouses. *Int. Journal of Cooperative Information Systems*, 7(2-3):215–247, 1998.
[13] O. Group. Unified Modeling Language (UML), Version 2.1.2. http://www.omg.org/technology/documents/formal/uml.htm.
[14] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of 6th Conf. on Principles of Knowledge Representation and Reasoning*, pages 636–649. Morgan Kaufmann, 1998.
[15] M. R. Jensen, T. Holmgren, and T. B. Pedersen. Discovering Multidimensional Structure in Relational Data. In *6th Int. Conf. on Data Warehousing and Knowledge Discovery*, volume 3181 of *LNCS*, pages 138–148. Springer, 2004.
[16] R. Kimball, L. Reeves, W. Thornthwaite, and M. Ross. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley & Sons, Inc., 1998.
[17] H. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Proc. of 9th Int. Conf. on Scientific and Statistical Database Management*, pages 132–143. IEEE, 1997.
[18] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of 21th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 233–246. ACM, 2002.
[19] A. Olivé. On the role of conceptual schemas in information systems development. In *Proc. of 9th Int. Conf. on Reliable Software Technologies (Ada-Europe 2004)*, volume 3063 of *Lecture Notes in Computer Science*, pages 16–34. Springer, 2004.
[20] C. Phipps and K. C. Davis. Automating Data Warehouse Conceptual Schema Design and Evaluation. In *Proc. of 4th Int. Workshop on Design and Management of Data Warehouses*, volume 58, pages 23–32. CEUR-WS.org, 2002.
[21] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *Journal on Data Semantics*, 10:133–173, 2008.
[22] S. Rizzi, A. Abelló, J. Lechtenbörger, and J. Trujillo. Research in data warehouse modeling and design: Dead or alive? In *Proc. of ACM 9th International Workshop on Data Warehousing and OLAP*, pages 3–10. ACM, 2006.
[23] O. Romero and A. Abelló. Multidimensional Design by Examples. In *Proc. of 8th Int. Conf. on Data Warehousing and Knowledge Discovery*, volume 4081 of *LNCS*, pages 85–94. Springer, 2006.
[24] O. Romero and A. Abelló. Automating Multidimensional Design from Ontologies. In *Proc. of ACM 10th Int. Workshop on Data Warehousing and OLAP*, pages 1–8. ACM, 2007.
[25] O. Romero and A. Abelló. A Survey of Multidimensional Modeling Methodologies. *Int. Journal of Data Warehousing and Mining (IJDWM)*, 5(2):1–23, 2009.
[26] I.-Y. Song, R. Khare, and B. Dai. SAMSTAR: A Semi-Automated Lexical Method for Generating STAR Schemas from an ER Diagram. In *Proc. of the 10th Int Workshop on Data Warehousing and OLAP*, pages 9–16. ACM, 2007.
[27] W3C. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/.
[28] R. Winter and B. Strauch. A Method for Demand-Driven Information Requirements Analysis in DW Projects. In *Proc. of 36th Annual Hawaii Int. Conf. on System Sciences*, pages 231–239. IEEE, 2003.