Data & Knowledge Engineering 69 (2010) 1138–1157

Contents lists available at ScienceDirect

Data & Knowledge Engineering

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

A framework for multidimensional design of data warehouses from ontologies

Oscar Romero [a,*], Alberto Abelló [b]

[a] Universitat Politècnica de Catalunya - BarcelonaTech, Dept. Llenguatges i Sistemes Informàtics, Barcelona, Spain
[b] Universitat Politècnica de Catalunya - BarcelonaTech, Dept. d'Enginyeria de Serveis i Sistemes d'Informació, Barcelona, Spain

ARTICLE INFO

Available online 15 July 2010

Keywords:
OLAP
Multidimensional design
Ontologies

ABSTRACT

The data warehouse design task needs to consider both the end-user requirements and the organization data sources. For this reason, the data warehouse design has been traditionally considered a *reengineering* process, guided by requirements, from the data sources.
Most current design methods available demand highly-expressive end-user requirements as input, in order to carry out the exploration and analysis of the data sources. However, the task to elicit the end-user information requirements might result in a thorough task. Importantly, in the data warehousing context, the analysis capabilities of the target data warehouse depend on what kind of data is available in the data sources. Thus, in those scenarios where the analysis capabilities of the data sources are not (fully) known, it is possible to help the data warehouse designer to identify and elicit unknown analysis capabilities.
In this paper we introduce a *user-centered* approach to support the end-user requirements elicitation and the data warehouse multidimensional design tasks. Our proposal is based on a reengineering process that derives the multidimensional schema from a conceptual formalization of the domain. It starts by fully analyzing the data sources to identify, without considering requirements yet, the multidimensional knowledge they capture (i.e., data likely to be analyzed from a multidimensional point of view). Next, we propose to exploit this knowledge in order to support the requirements elicitation task. In this way, we are already conciliating requirements with the data sources, and we are able to fully exploit the analysis capabilities of the sources. Once requirements are clear, we automatically create the data warehouse conceptual schema according to the multidimensional knowledge extracted from the sources.

1. Introduction

Data warehousing systems were conceived to support decision making within organizations. According to [17], a data warehousing system is a *collection of methods, techniques, and tools used to support knowledge workers — e.g., senior managers, directors, etc. — to conduct data analysis that helps with performing decision making processes and improving information resources*. Typically, relevant data for decision making is extracted from the organization data sources, transformed (i.e., cleaned and homogenized) and finally integrated within a huge repository of data (the *data warehouse*), in what is known as the ETL (extraction/transform/loading) process. The data warehouse provides a single and detailed view of the organization, and it is intended to be exploited by means of the *exploitation tools*, which provide different mechanisms to navigate and perform analysis tasks over the data warehouse. Among the different kinds 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

* Corresponding author.
E-mail addresses: oromero@lsi.upc.edu (O. Romero), aabello@essi.upc.edu (A. Abelló).

0169-023X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2010.07.007

intended to facilitate information analysis and navigation through the business data based on the *multidimensional* paradigm. The multidimensional view of data is distinguished by the *fact/dimension* dichotomy, and it is characterized by representing data (i.e., the fact of interest) as if placed in an n-dimensional space (with as many axes as dimensions of analysis of interest). This paradigm allows to easily understand and analyze data showing the different points of view from where a subject can be analyzed. In consequence, the multidimensional model fits for non-expert users like knowledge workers (from here on, the data warehouse *end-users*). For example, a typical fact of interest would be the business `sales`, whereas its typical dimensions of analysis would be the `item` sold, `where` it was sold (i.e., the place) and `when` (i.e., the time). One fact and several dimensions of analysis form what is known as *multidimensional schema* or *star-schema*. Nowadays, it is widely accepted that the conceptual schema of a data warehouse must be structured according to the multidimensional model, so that it can be exploited by OLAP tools.

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. Data warehousing systems need to consider both the end-user requirements and the organization data sources. In this context, the end-user requirements often come as business queries or service level agreements (SLAs), and represent the end-user analytical necessities, whereas the data sources are needed to know from where to extract the required data (and how to eventually populate the target data warehouse) in order to give answers to the information requirements. For this reason, the data warehouse design task has been considered a *reengineering* process, ideally guided by requirements, from the data sources: i.e., creating a data warehouse does not require the addition of new information but rearrange the existing information (indeed, the data warehouse is nothing else than a strategic view on the organization data). 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. However, the more the process gets automated, the more the integration of requirements is overlooked on the way.

In our previous work [39] we addressed how to automatically validate requirements and conciliate them with the data sources, to support the data warehouse design task. This work fits in traditional scenarios in which the end-user requirements are known before hand (i.e., by the point we start the design task). However, this scenario does not always hold in data warehousing, and the task to elicit the end-user information requirements might result in a thorough task. Importantly, note that the analysis capabilities of the target data warehouse depend on what kind of data is available in the data sources. Thus, in those scenarios where the analysis capabilities of the data sources are not (fully) known, it is possible to help the data warehouse designer to identify and elicit unknown analysis capabilities. Eventually, these unknown capabilities may provide strategic advantages for the organization.

In this paper we introduce a *user-centered* approach to support the end-user requirements elicitation and the data warehouse multidimensional design tasks. It consists of three steps:

- First, our approach starts by fully analyzing the data sources to identify, without considering requirements yet, the multidimensional knowledge they capture (i.e., data likely to be analyzed from a multidimensional point of view).
- Next, we propose to exploit this knowledge in order to support the requirements elicitation task. In this way, we are already conciliating requirements with the data sources, and we are able to fully exploit the analysis capabilities of the sources.
- Finally, once requirements are clear, we automatically create the data warehouse conceptual schema according to them, and the multidimensional knowledge extracted from the sources.

Thus, we say it is a user-centered approach since the feedback of the user[1] is needed to filter and shape results obtained from analyzing the sources, and eventually produce the desired conceptual schema. In this scenario, our main contribution is the AMDO (*Automating Multidimensional Design from Ontologies*) method, our proposal for discovering the multidimensional knowledge contained in the data sources (i.e., corresponding to the first stage discussed above). Importantly, note that AMDO focuses on identifying the multidimensional knowledge contained in the sources regardless of the requirements (as discussed in Section 2, this kind of approaches are known as *supply-driven* approaches). Relevantly, current supply-driven approaches suffer from two major drawbacks, which we claim to overcome with AMDO.

The first one is that supply-driven approaches tend to generate too many results. Consequently, they unnecessarily overwhelm users with blindly generated combinations whose meaning has not been analyzed in advance. Eventually, they put the burden of (manually) analyzing and filtering results provided onto the designer's shoulder, but the time-consuming nature of this task can render it unfeasible when large data sources are considered.

Filtering the results provided by these approaches is a must, and AMDO aims at filtering the results obtained by means of objective evidences. Specifically, we introduce the concepts of *filtering function* and *searching patterns* (see Section 4 for further information), which filter and rank results obtained and eventually facilitate the analysis of AMDO's output.

The second drawback is that current supply-driven approaches mostly carry out the design task from relational OLTP (*On-Line Transaction Processing*) systems, assuming that a RDBMS is the most common kind of data sources we may find, and taking as starting point a relational schema (i.e., a logical schema). As a result, 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

[1] Note that we distinguish between the end-user (i.e., the users that will exploit the data warehouse once devised), and the users benefiting from our approach (i.e., the data warehouse designers).

way to represent them at the logical level is by means of "foreign" (FK) and "candidate key" (CK) constraints. This scenario can be clearly seen in the example shown in Fig. 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 multidimensional role 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 methods 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*.

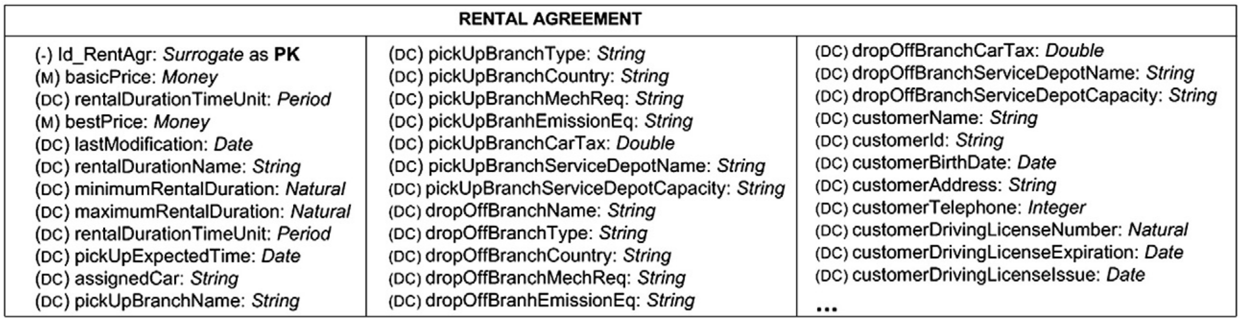
Dimension hierarchies are crucial in the multidimensional model which is based on two main features: 1) placement of data in the multidimensional space and 2) summarizability of data. A bad design of the dimension hierarchies would directly impact on the aggregation paths we may have. Modify data granularity (by means of the aggregation paths) when showing data to the end-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 [27]. 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 top of information systems has been discussed in depth in the literature (see, for example, [30,35]). 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 example, consider now the ontology depicted in Fig. 2. This ontology plays a conceptual role regarding the logical implementation depicted in Fig. 1. The piece of ontology depicted in the figure (that will be used as running example all over the paper) refers to a `car rental agreement` between a `branch` and a `costumer`. For a given `rental agreement` a `car` is `assigned`. In turn, `rental agreements` are classified into ongoing agreements — i.e., an `opened rental` — and bookings — i.e., `reservations`. An opened rental is considered `closed` when the agreement expires and the car is returned, whereas a reservation might have booked a `guaranteed canceled`. Information about the `branch` is also captured, 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 is needed`.

In this picture, each relevant concept of the domain is clearly stated as well as its relationships with the other concepts, and for instance we will be 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 form, as a whole, the `car` dimension.

Summing up, AMDO main goal is to facilitate the requirements elicitation and the data warehouse design tasks. AMDO achieves so because i) it improves the quality of the output obtained in the analysis of the data sources by working over a conceptual formalization of the domain (instead of a logical one), and ii) it automates the process as much as possible. This second feature is the main reason for choosing ontologies instead of other conceptual formalizations, since ontology languages provide reasoning services that will facilitate the automation of our task. Specifically, we choose OWL DL [46], a W3C recommendation based on *Description Logic* (DL) [4], as our input ontology language. Later, we will show that our algorithm can be adapted to other ontology languages (even less expressive, as discussed in Section 6).

Nowadays ontology languages are widely used in different areas like data integration [25] and the Semantic Web [6], but in other areas, like software engineering, UML [18] and ER [10] 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. As discussed in the literature (see for example, [3,5,9,14,30]), this process can be automated.



RENTAL AGREEMENT		
(-) Id_RentAgr: *Surrogate* as **PK**	(DC) pickUpBranchType: *String*	(DC) dropOffBranchCarTax: *Double*
(M) basicPrice: *Money*	(DC) pickUpBranchCountry: *String*	(DC) dropOffBranchServiceDepotName: *String*
(DC) rentalDurationTimeUnit: *Period*	(DC) pickUpBranchMechReq: *String*	(DC) dropOffBranchServiceDepotCapacity: *String*
(M) bestPrice: *Money*	(DC) pickUpBranhEmissionEq: *String*	(DC) customerName: *String*
(DC) lastModification: *Date*	(DC) pickUpBranchCarTax: *Double*	(DC) customerId: *String*
(DC) rentalDurationName: *String*	(DC) pickUpBranchServiceDepotName: *String*	(DC) customerBirthDate: *Date*
(DC) minimumRentalDuration: *Natural*	(DC) pickUpBranchServiceDepotCapacity: *String*	(DC) customerAddress: *String*
(DC) maximumRentalDuration: *Natural*	(DC) dropOffBranchName: *String*	(DC) customerTelephone: *Integer*
(DC) rentalDurationTimeUnit: *Period*	(DC) dropOffBranchType: *String*	(DC) customerDrivingLicenseNumber: *Natural*
(DC) pickUpExpectedTime: *Date*	(DC) dropOffBranchCountry: *String*	(DC) customerDrivingLicenseExpiration: *Date*
(DC) assignedCar: *String*	(DC) dropOffBranchMechReq: *String*	(DC) customerDrivingLicenseIssue: *Date*
(DC) pickUpBranchName: *String*	(DC) dropOffBranhEmissionEq: *String*	...

Fig. 1. A logical implementation in a RDBMS of a car rental agreement.

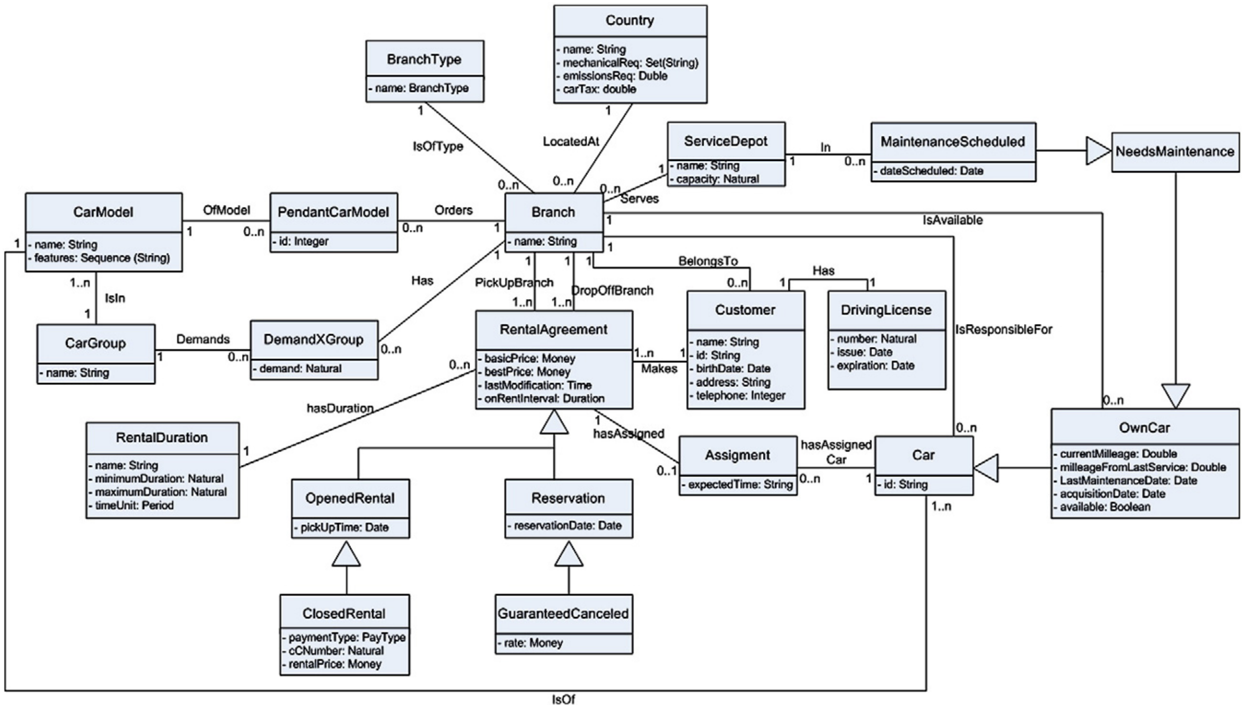


Fig. 2. Diagrammatic representation (based on UML notation) of a piece of a car renting ontology. The whole EU-Car Rental ontology [13] in OWL DL notation can be found at http://www.lsi.upc.edu/~ oromero/EUCarRental.owl.

Nevertheless, if there is not any conceptual formalization of the domain available we can use integration and reverse engineering techniques to obtain it. For example, Ref. [43] shows that a variety of structured and unstructured data stores can be elegantly represented as graphs, and it describes how an appropriate ontology for such data stores can be constructed by integrating a domain vocabulary with the data sources' vocabulary.

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

Nowadays, we can find several methods aimed at supporting the data warehouse design process. According to Ref. [47], multidimensional modeling methods may be classified within a *demand-driven* or a *supply-driven* framework. They properly define each framework as follows:

- Demand-driven approaches: also known as *requirement-driven* or *goal-driven*, focus on determining the end-user multidimensional requirements (as typically performed in other information systems) to produce a multidimensional schema. Only in the later stages, they map the output schema onto the data sources. For example, [23,34,47].
- Supply-driven approaches: also known as *data-driven*, start from a detailed analysis of the data sources to determine the multidimensional concepts in a reengineering process. End-user requirements are eventually considered in the later stages to filter results obtained. For example, [16,22,31,44].
- Hybrid approaches: some works propose to combine both frameworks. Mostly, these approaches start with a demand-driven stage to identify facts of interest and then, they identify its dimensional concepts by means of a supply-driven stage. For example, [8,15,21,29].

Demand-driven approaches[2] follow a classical software engineering approach, whereas supply-driven approaches exploit the inherent features of data warehouses and propose a reengineering approach from the data sources. Carrying out an exhaustive search of dimensional concepts among all the concepts of the domain (like supply-driven approaches do) has a main benefit with regard to those approaches that derive the schema from requirements and later conciliate them with the data sources (i.e., demand-driven approaches): in many real scenarios, the user may not be aware of all the potential analysis

[2] In this section, we also mean demand-driven stages within hybrid approaches when talking about demand-driven approaches. And analogously for supply-driven approaches/stages.

capabilities contained in the data sources and, therefore, he/she may overlook relevant knowledge. Oppositely, demand-driven approaches 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 discarded. As a counterpart, supply-driven approaches risk wasting resources by handling much unnecessary information [47], and tend to generate too many results (since they overlook the multidimensional requirements, they must apply their design patterns across all data sources). Furthermore, demand-driven approaches have not been traditionally automated, whereas supply-driven ones tend to facilitate their automation. The main reason is that demand-driven stages would require to formalize the end-user requirements (i.e., translate them to a language understandable by computers). In general, current methods handle requirements mostly stated in languages (such as natural language) lacking the required degree of formalization. Thus, matching requirements over the data sources must be performed manually.

As discussed in previous section, AMDO follows a supply-driven approach with the aim of supporting the elicitation of end-user requirements and the data warehouse design tasks. Thus, in this section, we focus on supply-driven approaches. The interested reader may find our demand-driven proposal and a thorough discussion about this kind of approaches in [39], and a detailed state of the art on multidimensional modeling in [12,38].

Most supply-driven approaches focus on the automation of the data warehouse design process from relational schemas [16,22,31,44]. These methods rely on a thorough analysis of the relational sources. On the one hand, facts are discovered based on simple heuristics (such as table cardinalities or numerical attributes, which may identify fake facts or overlook real ones). On the other hand, 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 form 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 obtained by these methods depends 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 ii) the degree of normalization of the logical schema (since some FKs and CKs are lost if we do not consider a schema up to 3NF):

- Golfarelli et al. [16] introduced a semi-automatable method to derive the multidimensional schema from relational schemas (indeed, they also propose to do it from ER diagrams, but they overlook the automation of the task and mostly rely on requirements, as discussed here). Gathering requirements and mapping them onto the relational schema is an important step in this approach. Therefore, the degree of automation is rather low and for example, 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 Fig. 1 they would not be able to identify any dimensional concept automatically.

- Phipps and Davis [31] 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). Any relation in 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 produces results containing too much noise. Consequently, a post-process based on the user requirements (i.e., a demand-driven stage) is carried out to filter results. This stage must be carried out manually and therefore, it would be hard, even for an expert, to filter and clean all results obtained. Regarding our example in Fig. 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).

- Finally, Jensen et al. [22] presented a method in which data mining techniques are used to analyze the relational sources. Assuming that the database does not contain composite keys, this method derives valuable metadata such as functional and inclusion dependencies and key or cardinality information, which identifies potential *snowflake schemas* [23]. To infer the metadata, the authors access the instances and apply data mining techniques, which could be unsuitable for large data sources. Moreover, since they are looking for snowflake schemas, they rely on FK-CK relationships 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), as in the methods discussed previously. Thus, the impact of denormalization is as big as in the rest of approaches. Finally, the complexity of this approach may become problematic due to the high number of combinations computed when searching for inclusion dependencies (as all combinations of potential candidate keys and foreign keys are constructed with the consequent computational cost). In our example, this method would not be able to identify any of the multidimensional concepts in the `rental agreement` relation.

To our knowledge, there is only one approach working at the conceptual level and following a similar framework to AMDO:

- Song et al. [44] 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 that each concept has. 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.

Finally, we would like to discuss those methods presenting a supply-driven stage within a hybrid approach. These methods have not been discussed above because they mainly rely on their demand-driven stages to discover the multidimensional concepts rather than on an accurate analysis of the data sources. Indeed, only a few of them automate this process somehow [8,15,21,29], but the degree of automation achieved is rather low. Specifically, these approaches consist of a detailed requirements 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. In these methods the requirements elicitation stage leads the process and the main design decisions are made in this step, whereas the analysis of the data sources is rather superficial. As a consequence, they introduce simple automatic design patterns. In fact, the analysis of the data sources can be thought as a complementary task to the requirements elicitation process. For example, facts are identified manually from requirements and the automation of the process is mainly reduced to discover to-one relationships from each fact identified (i.e., its dimensional concepts).

As an example, Ref. [29] introduces a commonly cited method for supporting the multidimensional schema. It requires each domain entity to be classified as a *transactional* (the basis for fact tables), *component* (details or components of business events that will produce dimensions) or *classification* (that will be used to shape the dimension hierarchies) entity. The authors give advice on how these entities can be identified. Thus, "transactional entities must describe events that happens at a point in time and contain measures or quantities that can be summarized". Formal rules are given for each type of entity to give shape to the multidimensional schema. For example, a typical rule (also discussed in [21]) is discovering *functional dependencies* (FDs) to identify dimensional data. However, manual discovery of FDs is an unfeasible task for most systems [11,45], and the automatic methods for identifying FDs (as suggested in these works) need to address this task by using the instance semantics. These methods have various drawbacks: they propose solutions that are computationally expensive, and register drops in performance when a large number of attributes or instances are processed [19,26,42,45]. However, this is the most common scenario in data warehousing systems [17].

Summing up, supply-driven approaches either do not achieve a fair automation degree or, if they automate the process to a fair degree, they use simple design patterns/heuristics to identify the multidimensional concepts.

2.1. Main contributions

Our proposal is a reengineering process that derives the multidimensional schema from a conceptual formalization of the domain. Our main contribution is the AMDO method: a fully *automatic* supply-driven approach working at the *conceptual* level.

Working from conceptual formalizations improves the quality of the output, as discussed earlier in this section. Although 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 requirements 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 requirements elicitation stage a posteriori, as discussed in Section 4. Therefore, unlike previous approaches, the automatic analysis of the sources leads the process.

Specifically, 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 example, 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 between levels and measures (see Section 4.1.1). Relevantly, a more accurate approach to discover facts than the ones used in previous approaches is provided (see Section 4). With regard to this last contribution, AMDO introduces the search pattern and filtering function concepts, which are used to guide the exploration and analysis of knowledge retrieved by AMDO. Our main contribution in this issue lays on the flexibility that these concepts provide. Indeed, we claim that we are able to capture any of the heuristics previously introduced in the literature to identify facts and, at the same time, we provide mechanisms to capture more expressive ones. To our knowledge, this is the first approach providing such mechanisms.

A possible reason why previous approaches working at the 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 example, 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, although 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 concept of data warehousing is known in the literature as *Web-Warehousing* [36]).

Based on this main contribution (i.e., AMDO), we present an approach for supporting both i) the end-user requirements elicitation and the ii) data warehouse design task.

Finally, the work presented in this paper is an evolution of the one introduced in [37]. We have improved our previous algorithms in two ways: now, they are semantically richer (for example, we are able to distinguish between levels and descriptors and identify aggregate measures) and we have improved their computational complexity. Indeed, our current algorithms can be mostly computed with generic reasoning algorithms [40], which allow us to take advantage of the advances in a continuing evolving area such as the DL community. Moreover, the *AMDO tool* has now been devised and thanks to it, we have been able to extend our previous work by considering practical considerations (which came up when testing our tool; see Section 5) 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 identify multidimensional concepts 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 form 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) [24]: (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, 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 example, 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 example, a list of compatibilities could be asked to the user for each measure identified).

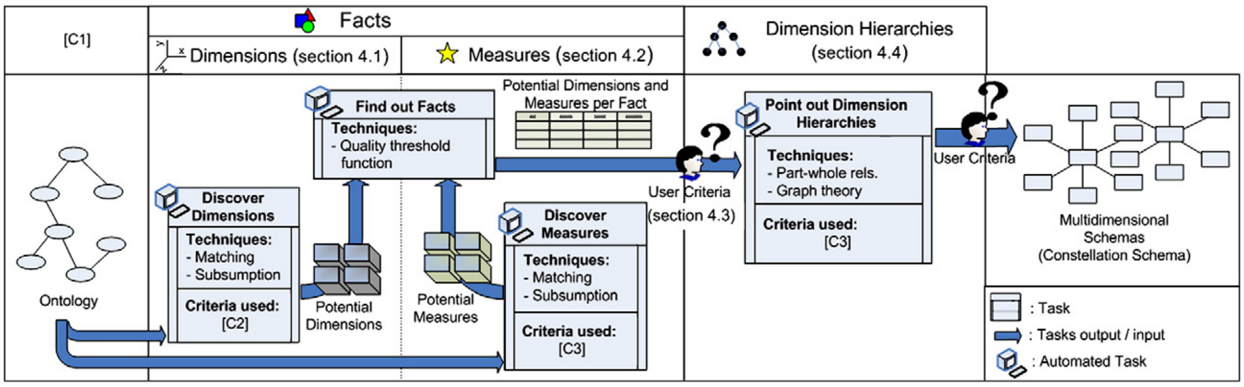


Fig. 3. Method overview.

4. Our approach

This section presents a detailed view of AMDO and how it can be used for supporting requirements elicitation and the data warehouse design tasks. Fig. 3 depicts a schematic overview of AMDO and how it applies the criteria introduced in Section 3.

Our approach's input is the ontology domain containing both, the data sources and the domain vocabularies. Note that no requirements are demanded as input (i.e., at the beginning), since our approach aims to support their elicitation. About AMDO, it is a two-task method to be used within a user-centered approach. AMDO aims at discovering the ontology 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 [31], and most methods rely on simple heuristics such as table cardinalities or numerical attributes. 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 note 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 second task of AMDO, where dimension hierarchies will be shaped.

Up to this point, AMDO's main goal is to identify, for each ontology concept, its potential measures and dimensional concepts. Note that this is a fully supply-driven task and requirements have not been considered and, according to our discussion in Section 2, it might have generated loads of results. For this reason, next step in this task aims at filtering results obtained before presenting them to the user. We do so by introducing the concepts of *search pattern* and *filtering function*.

A search pattern is a *topological pattern* that does not consider semantics (i.e., it does not represent business requirements) but provides information about how the target schema must look like. For example, consider the number of numerical attributes of a given concept. This search pattern is the most widely used heuristic to identify facts in the literature (see Section 2). In AMDO, search patterns are implemented by their corresponding *filtering functions*. Thus, a filtering function is used to assess and rank (regarding the other ontology concepts) every ontology concept, according to the search pattern it represents. Following the same example, we implement it with $Attrs(C) := num\_attrs$ (where C is a given ontology concept and num_attrs the number of numerical attributes it has). Relevantly, AMDO can capture any previous automatable heuristic introduced in the literature and, in addition, it provides a framework to capture more expressive ones, as discussed later in this section. Finally, according to the filtering function, AMDO would provide a ranked list of ontology concepts. The higher a concept is ranked, the better is estimated as a fact (according to the search pattern captured by this filtering function).

Note the following and interesting properties of our approach:

- It is completely flexible regarding search patterns and filtering functions. Indeed, the designer can formulate as many filtering functions as needed and for each function provided, the ontology concepts would be ranked in a different way.
- Relevantly, all the previous heuristics used in the literature can be captured by AMDO (for example, the "Connection Topology Value" [44]). In addition, AMDO also provides two new *variables* that have not been considered in previous works: we compute the number of measures and dimensional concepts. To compute these variables, we introduce novel *design patterns*[3] to compute potential measures and dimensional concepts for a given ontology concept (see Sections 4.1 and 4.2 for further details).
- Note that the designer can formulate and compute as many filtering functions as desired, and each function will assess the ontology concepts as potential facts in a different way. For example, consider the design of a data warehouse in the bionformatics field. By analyzing already existing multidimensional schemas in this field, we may realize that a schema tends to be relevant if it provides plenty of measures and different analysis perspectives of interest for a given fact. Nevertheless, specific facts that do not fit such generic patterns can also be identified by providing the proper searching pattern. For example, if we aim at identifying *factless facts* [23], measures should not be considered, and we would like to weight each concept by its dimensional concepts (and even give more weight to those closer to the given ontology concept).

This approach provides a powerful framework so much so that we can even capture generic requirements in the search pattern (i.e., involving semantics). For example, data warehousing aims at providing historical data and thus, we could be interested in weighting the number of numerical attributes a concept has with regard to the topological distance to the `date` datatype. This search pattern can be captured in the following filtering function: $AttrsByDistance(C) := \frac{num\_attrs}{dist(date)}$ (where *dist(date)* computes the distance — i.e., the minimum number of properties between C and `date`). In this way, the closer to `date`, the better it is ranked.

[3] The reader will note that we distinguish between design patterns and search patterns. The first ones are those aimed at identifying the multidimensional concepts (i.e., they are in compliance with the multidimensional model), whereas search patterns are used to *filter* the results obtained by computing the design patterns over the sources. Thus, search patterns might depend on the domain and kind of data warehouse devised.

For each filtering function, AMDO presents a ranked fact list to the user (indeed, it would be possible to present partial orders regarding a given set of filtering functions). At this point, this valuable information about the sources is supposed to be exploited in order to support the requirements elicitation task. We can use any of the approaches discussed in the literature with this aim; for example [15,28,33,41,47]. However, we do believe that it is easier to carry out the requirements elicitation process from knowledge proposed by AMDO (indeed, if the user would like to, AMDO can show the list of potential measures and dimensional concepts computed for each fact) than carrying it out from scratch. On the one hand, we are already conciliating requirements with data available. On the other hand, we fully exploit the analysis capabilities of the sources. In the end, the user must select those relevant facts for his/her decision making. For each fact identified, we perform the second task of AMDO.

The second task in AMDO gives rise to dimension hierarchies. For each fact chosen in the previous task, we rearrange its dimensional concepts to form its dimensions of analysis. In other words, we aim at identifying 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 (i.e., delete a whole dimension, a level, a descriptor or split a proposed dimension into two different ones semantically related).

Once the user has chosen which facts are of interest, the reader will note that it is immediate to produce the multidimensional schema. In the first task, AMDO already computed the measures and dimensional concepts for each fact, whereas the second task arranges the dimensional concepts into dimension hierarchies. Thus, the output of AMDO will be a multidimensional schema for each fact identified that will produce, as a whole, a *constellation schema* [23]. Relevantly, the schema produced is known to be in agreement with the end-user requirements and the data sources (since it has been derived from a detailed analysis of the sources and according to the end-user requirements).

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:

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

Note that it is expressed in Description Logic (DL) notation [4] (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 $B \sqsubseteq A$. o 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.

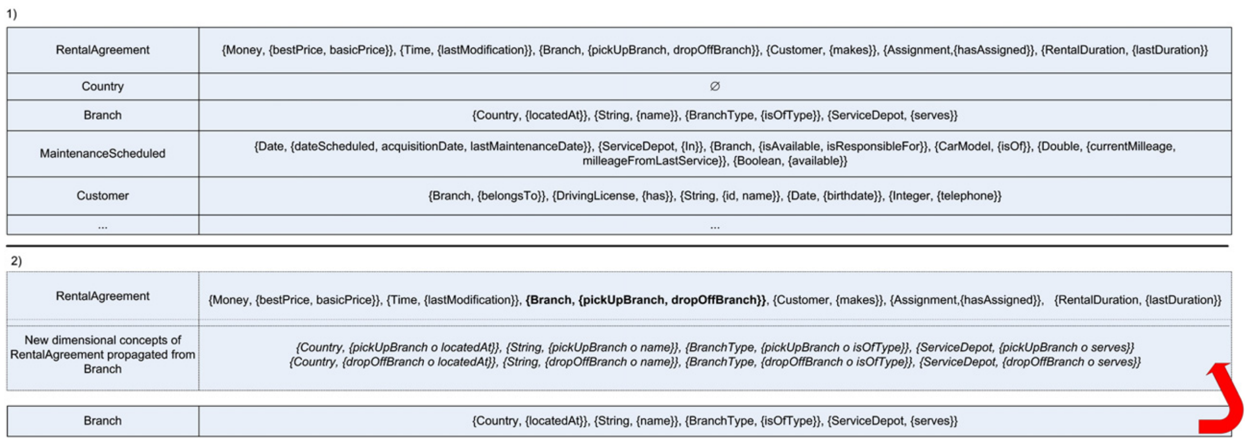
For example, consider the conceptual schema in Fig. 2. There, `rental agreement` has two to-one relationships to `branch` (i.e., `pickUpBranch` and `dropOffBranch`). Thus, {`branch`, `pickUpBranch`} and {`branch`, `dropOffBranch`} must be considered as two different points of view from where analyze 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 [4]. Moreover, it is not even expressible in OWL DL. Nevertheless, it is feasible to decompose this pattern as follows:

- First, for a given class *F*, we look for its *direct* dimensional concepts (see Section 4.1.1).
- Next, we propagate this knowledge to compute the transitive closure of dimensional concepts according to the *transitive rule* (see Section 4.1.2):

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∘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 ad hoc algorithm that will also partially benefit from these algorithms.



1)

RentalAgreement	{Money, {bestPrice, basicPrice}}, {Time, {lastModification}}, {Branch, {pickUpBranch, dropOffBranch}}, {Customer, {makes}}, {Assignment,{hasAssigned}}, {RentalDuration, {lastDuration}}
Country	∅
Branch	{Country, {locatedAt}}, {String, {name}}, {BranchType, {isOfType}}, {ServiceDepot, {serves}}
MaintenanceScheduled	{Date, {dateScheduled, acquisitionDate, lastMaintenanceDate}}, {ServiceDepot, {In}}, {Branch, {isAvailable, isResponsibleFor}}, {CarModel, {isOf}}, {Double, {currentMilleage, milleageFromLastService}}, {Boolean, {available}}
Customer	{Branch, {belongsTo}}, {DrivingLicense, {has}}, {String, {id, name}}, {Date, {birthdate}}, {Integer, {telephone}}
...	...

2)

RentalAgreement	{Money, {bestPrice, basicPrice}}, {Time, {lastModification}}, **{Branch, {pickUpBranch, dropOffBranch}}**, {Customer, {makes}}, {Assignment,{hasAssigned}}, {RentalDuration, {lastDuration}}
New dimensional concepts of RentalAgreement propagated from Branch	*{Country, {pickUpBranch o locatedAt}}, {String, {pickUpBranch o name}}, {BranchType, {pickUpBranch o isOfType}}, {ServiceDepot, {pickUpBranch o serves}}* *{Country, {dropOffBranch o locatedAt}}, {String, {dropOffBranch o name}}, {BranchType, {dropOffBranch o isOfType}}, {ServiceDepot, {dropOffBranch o serves}}*

Branch	{Country, {locatedAt}}, {String, {name}}, {BranchType, {isOfType}}, {ServiceDepot, {serves}}

Fig. 4. Example of propagation of to-one paths by transitivity.

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 the previous section:

$$F \sqsubseteq {=}1r.D, \textit{where } r \textit{ is a single property}$$

In our approach, we also consider OWL *datatypes* (which traditionally have only been considered as measures by supply-driven approaches) 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 in Section 4.2), but also the role of a dimensional concept. Handling measures and dimensions uniformly is not new. In fact, it was introduced by Agrawal et al. [2] and since then, it has also been proposed in other design methods. Thus, in our example, `Money` (`basicPrice` and `bestPrice`) and `Date` (`lastModification`) would be considered potential dimensional concepts of `rental agreement`. Importantly, dimensional concepts and measures derived from the same ontology datatype must be semantically related in the multidimensional schema (for example, by the “equivalence” construct in OWL or by an “association” relationship in UML). This conceptual relationship will eventually be exploited by multidimensional operators such as “drill-across” and “pivoting”.

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 its dimensional concepts as pairs $\{D, \{r_1, ..., r_n\}\}$[4] (where each r_i is a property between F and D).

Finally, note that using reasoning to compute this pattern means that any assertion stated in the ontology (by using OWL DL constructs) is automatically considered. For example, 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 ad hoc algorithm.

4.1.2. Propagating dimensional concepts by transitivity

This section presents an ad hoc algorithm to compute the transitive closure of dimensional concepts. Although 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:

$$\forall \{D, \{r_1, ..., r_n\}\} \in M{<}F{>} \rightarrow \begin{cases} F \sqsubseteq {=}1r_1.D, \\ \cdots \\ F \sqsubseteq {=}1r_n.D \end{cases}$$

Where M is the $N \times N$ matrix, F and D are classes, $r_1, ..., r_n$ are composite properties, and $M{<}F{>}$ an operator over M that retrieves the dimensional concepts of F: i.e., a list of classes related to F by to-one paths. 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 represented as a composite property. Therefore, we may derive as many dimensional concepts as different paths we have from F to D.

For example, let us consider `rental agreement` as F. Thus, M<`rental agreement`>will contain the following dimensional concepts: {`Branch`, {`pickUpBranch`, `dropOffBranch`}} and {`Country`, {`pickUpBranch`∘`locatedAt`, `dropOffBranch` ∘ `locatedAt`}}, among others. `Branch` and `country` play the role of D in each pair, respectively, whereas {`pickUpBranch`,

[4] The reader is addressed to Definition 1 to recall that each combination of *ending concept — path* (i.e., each $\{D, r_i\}$) will give rise to a different multidimensional concept.

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

functioncreate_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);
    6. return M;
```

Fig. 5. An algorithm to compute matrix M.

`dropOffBranch`} and {`pickUpBranch∘locatedAt`, `dropOffBranch ∘ locatedAt`}, are the list of to-one paths from `rental agreement` to `branch` and to `country` respectively.

In this section we present how to compute this matrix, and we achieve so by means of the next algorithm:

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 computes the pattern presented in Section 4.1.1. Each potential dimensional concept identified is added to the proper list in the vector. Fig. 4.1 (depicted like a matrix) shows results got after Step 4 for some of the concepts introduced in Fig. 2. For example, 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 reader will note that, according to Definition 1, those classes (or datatypes) related to `maintenance schedule` by several to-one paths (i.e., `date`, `branch` and `double`), produce several dimensional concepts.

Step 5 propagates the dimensional concepts identified in the previous step according to the transitive rule (see Definition 2). The *propagate_path* function describes in detail this step (see Fig. 6). This function implements a smart algorithm to compute the transitive closure. 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 i^{th} iteration of our algorithm, if all its dimensional concepts have been computed in any iteration previous to *i*. In other words, if a class *C* closes in a certain iteration *i*, no other dimensional concept will be identified for *C* in a iteration *j* such that $i <= j$. In our notation, we say that a concept *C* closes in an iteration *i* iff:

General case: $\forall \{D, \{r_1, ..., r_n\}\} \in M{<}C{>}, D$ closed in a given iteration j such that $j <= i$,
Basic case: leaf classes and datatypes are, by definition, closed classes.

The main idea behind this algorithm is the following. Our algorithm only *propagates* knowledge from closed classes (see Steps 11(b)i and 11(c)iiB). 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[5]; see Steps (11b and 11(b)iA).

Knowledge from closed concepts is propagated 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):

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

[5] The reader will note that the treated method does not hold at the class level but at dimensional concept level (i.e., regarding *M*<*C*,*D*> and not *M*<*D*>), since *D* can be in the list of several concepts.

```
void propagate_path (Matrix ↕M) // M is an input / output parameter (i.e., ↕ )
    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
                    listEls := listEls ∪ (r ∘ M<D>);
                    M<C> := M<C> ∪ listEls;
                    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
                    M<C> := M<C> ∪ M<P>;
                B. C.closed := true;
                C. conceptsClosed:=true;
    12. if(!conceptsClosed)
        (a) break_deadlocks(M);
    13. } while concepts_not_closed(M) > 0
```

Fig. 6. An algorithm to propagate dimensional concepts by transitivity.

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*>). For example, regarding Fig. 4.1, M_c`<branch>` would retrieve {`country,branchType,serviceDe-pot`}, whereas M_p`<branch,country>` would retrieve {`locatedAt`}.

Hence, D_i are the set of classes in the dimensional concepts list of *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 is captured with a slight abuse of notation in Step 11(b)iA of the algorithm). Intuitively, we are *concatenating* each to-one path from *C* to *D* with each to-one path from *D* to D_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)iiA). In our algorithm, this kind of propagation is done when all the parents of *C* have closed (see Step 11(c)ii). Note that we can take advantage of DL reasoning (see Section 6 for further details) to compute the list of direct superclasses of a given concept. Eventually, we only propagate each superclass to each of its subclasses once.

How our algorithm works, can be summarized, in an intuitive way, as follows:

- First iteration: leaf classes and datatypes[6] close in this iteration (see Step 8). By definition, none of them have to-one relationships.
- 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 reachable 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 already closed in the $(n-1)^{th}$ iteration (see Step 11c). Now, we can guarantee that all the dimensional concepts of *C* have been computed and we can now propagate *C* in the next iteration (see Step 11(c)iiB).

Following our example, Fig. 4.1 shows direct dimensional concepts identified for some ontology classes, and Fig. 4.2 depicts how we propagate them by transitivity. For example, `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`.

[6] As discussed in Section 4.1.1, our algorithm considers datatypes as potential dimensional concepts. For the purpose of our algorithm, they can be considered leaf classes.

Hence, these concepts are also considered dimensional concepts of `rental agreement` according to the transitivity rule (see the arrow in Fig. 4.2). Moreover, according to Definition 1, the path list of these newly identified dimensional concepts have been properly updated when adding them to the list of `rental agreement`. For example, from `rental agreement` we can reach 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.

Our algorithm relies on following the to-one paths from their end (i.e., leaf classes) to the beginning, so the knowledge of each concept is propagated just once. Therefore, it is essential to detect potential *deadlocks*. A deadlock is a cycle (in the sense of graph theory) of to-one properties. When a cycle is detected (in our algorithm, when no class closes in the current iteration; see Steps 10 and 12), it is *broken*: i.e., by propagating *once*, among all the concepts involved, their dependencies (roughly speaking, *sharing* their dimensional concepts lists). This situation must 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. For example, the typical example would be the `person` class with the reflexive to-one relationship `father_of`. By iterating over the cycle, we may infer new semantics. For example, `father _ of∘father _ of` would give rise to the `grandfather` concept, and so on. However, our algorithm just iterates one over each cycle (i.e., breaks the cycle), and we inform the user just in case he/she would like to further exploit the cycle semantics.

The most common and also easiest deadlocks to detect and break are the one-to-one and reflexive relationships. For this reason, AMDO treats these two basic cases before computing the *propagate_path* function (see Step 3 in Fig. 5). 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 detect general cycles (i.e., those detected and broken in Step 12). For example, a *depth-first-search* (DFS) remembering previous visited nodes would fit properly.

In our example, we only dispose of trivial deadlocks, such as the one formed by the one-to-one `has` property between `customer` and `driving license`. In this case, before the *propagate_path* function is triggered, it is annotated that these two concepts form a deadlock. Thus, when all the concepts in their respective lists close (but themselves, which are waiting for each other) we break the deadlock. In our example, it happens in the first iteration. Thus, we propagate once between them and notify to the user that, in case he/she would be interested, it is possible to derive new dimensional concepts for both concepts by 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, in our algorithm, 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 example, consider the EU-Car Rental ontology (see Fig. 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 never propagated again. Thus, 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 yield:

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

The result obtained, drastically smaller than the theoretical upper bound, is still an upper bound of the answer time of AMDO. Note that we are considering the maximum connectivity for each class in each iteration, which in real ontologies will hardly hold. Nevertheless, we want to underline some important features of our algorithm that come up in this formula: on the one hand, note that the value of N_i is strictly decreasing. On the other hand, the value of c_i is never exponential. In fact, in the last iteration, its value

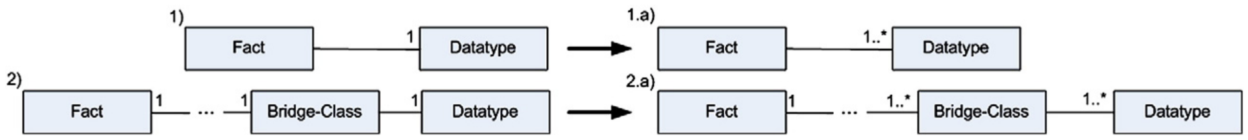


Fig. 7. Multiplicities looked for.

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: i.e., 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 Fig. 5) will propagate, at least, one property. Indeed, the *invariant* of the main loop of the algorithm (see Step 9 of Fig. 6) 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. Accordingly, 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). Similar to Definition 1, we must take into account the semantics of the paths between the fact and the datatype when producing measures. Thus:

Definition 4. A measure is defined by a *datatype* and a *path of properties* (i.e., a composite property). From a multidimensional point of view, the path must be considered because it adds relevant semantics. A fact related to a datatype by means of n different to-one paths must give rise to n different measures, as all these paths will potentially relate each fact instance with different datatype values.

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.

Fig. 7 shows the two kinds of relationships we look for. 1) The first pattern depicts a measure directly related to the fact. This pattern is the equivalent one in OWL to those used by previous approaches in the relational model or UML/ER diagrams.[7] For example, in our example the `bestPrice` and `basicPrice` datatypes related to `rental agreement` would follow this pattern. Nevertheless, note that 1.a) this pattern can be relaxed by accepting mandatory *multivalued datatypes* (i.e., each class may be related to at least one value and at most, many of them). In this case, we can produce the required to-one relationship by aggregating the *many* values related to each fact instance. When doing so, it is compulsory to use a *compatible* aggregation function (see [C3]). Relevantly, note that this information should be considered in the ETL process. Otherwise, if this aggregation is not performed, data loaded in the data warehouse will not preserve [C3].

The second pattern in Fig. 7 is used to discover aggregate measures. 2) This 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. Clearly, 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) in the path between the fact and the bridge-class, it is enough to ask for a

[7] A datatype does not have an *object identifier* (i.e., *oid*) and we allow any multiplicity in the fact side without violating *disjointness*.

mandatory relationship. Again, all the (*many*) measure 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 example, 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 derive 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 are captured and computed in 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 example, *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. According to Definition 4, each property (i.e., path) between F and dt will produce a measure:

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

  If so, it means that, by transitivity, we have been able to identify B as a dimensional concept of F and vice versa (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*. 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:

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

    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., `..N`) path. Like in the definition of matrix M, each class in this list is represented as $\{D,\{r,\ldots,r_n\}\}$ 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:

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

    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 Fig. 5) to compute M, but instead of looking for direct to-one relationships in Step 4 (see Fig. 5), 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 obtained are filtered and ranked according to as many search patterns as desired (see Section 4). For example, consider the following search pattern for our running example: “The more measures and dimensional concepts, the better. However, measures are considered to be more relevant, and we are not interested in factless facts”. One possibility to capture this pattern would be the following filtering function:

$$FactEstimation(C) := \begin{cases} M*2 + DC, & \text{if M} > 0; \\ 0, & \text{otherwise.} \end{cases}$$

where M is the number of potential measures, and DC the number of potential dimensional concepts of a given ontology class. As discussed in Section 4, AMDO is completely flexible and tunable regarding the filtering function, and users are able to provide any function which better meets his/her necessities. Regarding the example introduced above, note that the FactEstimation function implements a more expressive pattern than the *Attrs* function (see Section 4), since it considers relationships to other concepts (i.e., potential measures and dimensional concepts). Interestingly, it could also be extended in many ways. For example, computing the topological distance,[8] as discussed in the *AttrsByDistance* function (see Section 4), if desired. According to this function, the most promising concepts are ranked as follows:

Concept	#Dimensional concepts	#Potential measures	FactEstimation
LateReturn	78	5	88
DamageCost	81	3	87
Prepared	81	3	87
AssignedCar	80	3	86
PaidWithPointsRental	74	4	82
ClosedRental	74	4	82
EarlyReturn	74	4	82

Having a look at the results, in general, 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. Furthermore, subclasses will be rated, in the worst case, as high as superclasses, since they inherit all their measures and dimensional concepts. Classes in the list that do not belong to the `rental agreement` taxonomy are just three: `damage cost` (2nd place), `prepared` (3rd place) and `assigned car` (4th place). However, they represent events, which traditionally have been good candidates as facts (see, for example, [7,29]).

Eventually, after exploring AMDO's output as much as needed, the user should select those facts of interest to him / her (normally, more than one).

4.4. Shaping the dimension hierarchies

In the previous task, for each fact chosen, we identified its dimensional concepts. However, we still need to shape its dimension hierarchies in order to allow summarizability of data; one of the multidimensional paradigm principles. Dimension hierarchies must guarantee a correct summarizability of data (see [C3]). Thus, in this task, we look for to-one relationships (also known as “roll-up” relationships) that will produce hierarchies preserving a correct data aggregation: the to-one multiplicity guarantees *disjointness* of aggregated data, and forbidding zeros we also guarantee its *completeness*.

From the set of dimensional concepts identified for a given fact, a *directed* graph following all the to-one relationship paths is depicted. At this point, two important remarks must be done. On the one hand, note that some graphs will overlap. Consider two concepts A, B such that B is a dimensional concept of A. Clearly, the graph created from B will be subsumed by the graph created from A. In these cases were a graph is subsumed by another one (i.e., it is completely overlapped), the *subsumee graph* is disregarded and not considered during the process. Intuitively, we only work with *maximal* graphs, as the rest can be derived from them straight-forward. On the other hand, note that, at this moment, we cannot differentiate the role played by each graph node (either as a level or as a descriptor) within the dimension hierarchy. Two specific patterns are introduced to distinguish levels from descriptors:

- (Levels): if a class (C_3) is either placed in more than one maximal graph or it can be reached from two different paths in the same maximal 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:

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

[8] Note that the topological distance can be computed from matrix M, which was introduced in Section 4.1.2.

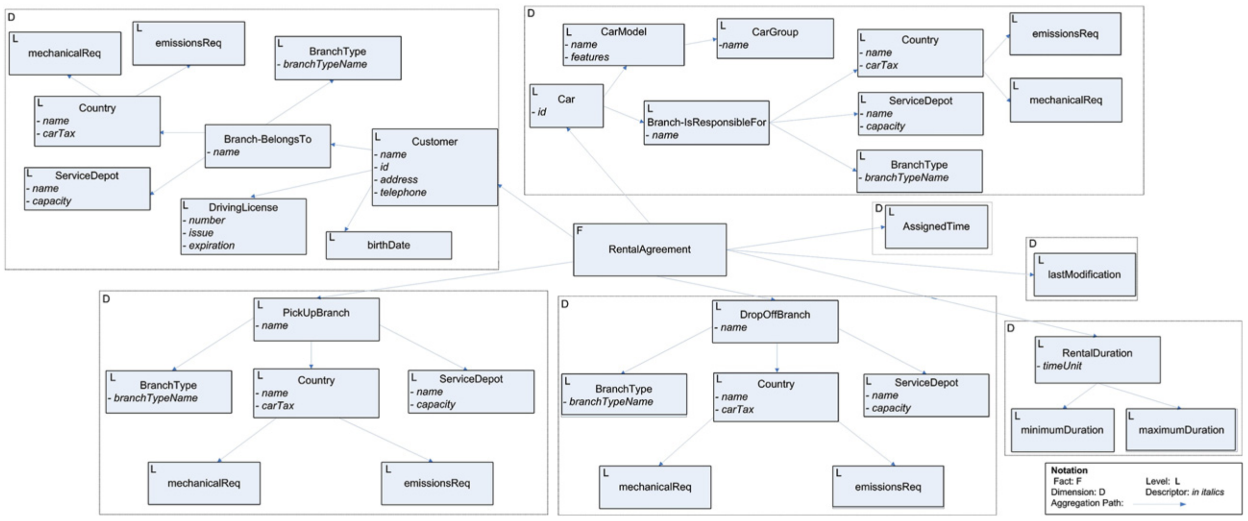

D
L mechanicalReq
L emissionsReq
L BranchType - branchTypeName
L Country - name - carTax
Branch-BelongsTo - name
L Customer - name - id - address - telephone
L ServiceDepot - name - capacity
L DrivingLicense - number - issue - expiration
L birthDate
L CarModel - name - features
L CarGroup -name
L Car - id
L Branch-IsResponsibleFor - name
L ServiceDepot - name - capacity
L BranchType - branchTypeName
F RentalAgreement
L AssignedTime
L lastModification
L PickUpBranch - name
L DropOffBranch - name
L RentalDuration - timeUnit
L minimumDuration
L maximumDuration
Notation
Fact: F
Dimension: D
Aggregation Path:
Level: L
Descriptor: in italics

Fig. 8. Diagrammatic representation of the resulting multidimensional schema for the rental agreement fact.

Where C_1 and C_2 are classes and r_1 and r_2 are properties.[9] Thus, we are asking if we can reach C_3 from, at least, two disjoint paths formed by graph edges.[10] If this holds, C_3 is considered to be a level in both graphs. Importantly, if r_1 and r_2 belong to different maximal graphs, it is mandatory to semantically relate both *levels* (i.e., C_3 in each of the maximal graphs) by means of a semantic relationship in the output multidimensional schema (eventually, the multidimensional operators, such as changeBase [1], will be able to exploit this relationship).

• (Descriptors): If two classes C_1 and C_2 are related by means of a one-to-one relationship in the same maximal graph:

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

This relationship can represent a (i) semantic relationship between two different dimensions if the *ending* class (note that the graph is directed) was identified as a level by the previous pattern, or (ii) as an attribute level (i.e., a descriptor) otherwise. In the first case, it means that the ending class provides a relevant data granularity (i.e., it was identified as a level) and thus, we also consider its counterpart to be of a level of interest. Thus, they are considered to represent the same granularity level in two different dimensions and, consequently, this semantic relationship must be explicitly asserted in the output schema. In the second case, we consider it to be a descriptor of the *initial* concept, since they have not been identified as interesting analysis levels.

For example, consider that the user selected `rental agreement` in Section 4.3. For this fact, AMDO would create 9 directed graphs (starting from `customer`, `lastModification`, `assignment`, `(dropOff)Branch`, `(pickUp)Branch`, `rentalDuration`, `car`, `bestPrice` and `basicPrice`). Consider `branch` to play the role of C_3 in the first pattern. If we are able to find two concepts (C_1, C_2), such that we can reach to `branch` by, at least, two different paths in any of the eight maximal graphs, AMDO will propose `branch` to play a level role. Indeed, it holds if we consider `customer` (through the `belongsTo` property) and `assignment` (through the `hasAssigned∘dropOffBranch` path) to play the role of C_1 and C_2. Consequently, all the graph nodes representing `branch` in any of the eight maximal graphs is considered a granularity level of interest. Furthermore, according to the first pattern, all these nodes are semantically related in the output schema by means of one-to-one associations (for example, with the OWL equivalence constructor). Similarly, regarding the second pattern, `birthDate` fulfills the conditions to be considered a descriptor, since it only appears in a one-to-one relationship in the `customer` maximal graph.

Graphs computed 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 the end-user requirements to identify each node 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 he/she is interested in aggregating data at this level.

For example, following the `rental agreement` example, AMDO would generate the dimension hierarchies shown in Fig. 8. There, each arrow starting from `rental agreement` depicts a dimension hierarchy. Concepts identified as descriptors by AMDO are depicted in *italics*. Finally, note that some tuning has been performed on this graphs. For example, we have considered as interesting aggregation levels some graph nodes identified by AMDO as descriptors (e.g., `minimumDuration` and `maximumDuration`) or by dropping dimensions of no interest (in our case we have not draw in the figure `bestPrice` and

[9] Note that r_1^{-} and r_2^{-} are the inverse properties of r_1 and r_2 respectively.

[10] For the sake of understandability, we recall that $\exists r_1 \sqsubseteq C_1$ is equivalent to say that C_1 is in the domain of r_1, and $\exists r_1^{-} \sqsubseteq C_3$ is equivalent to say that C_3 is in the range of r_1.

`basicPrice`). It is important to remark that some levels are present in different hierarchies. This is sound since, according to Definition 1, the path provides different semantics when identifying dimensional concepts. Thus, reaching `country` from `customer` has different semantics from reaching it from `rental agreement`. Nevertheless, all those levels derived from the same class but present in different graphs are related by semantic relationships. For example, `country` or `branch type`.

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 example, *get _day*, *get _month* or *get _year* in case of dates, such as `assignedTime` or `lastModification`) or aggregate data to produce value ranges of interest in case of numerical datatypes (for example, from 0 to 20, 21 to 50, 51 to 99 in case of ages), in order to create ad hoc dimension hierarchies.

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 example, 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:

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

Note, however, that this relaxed pattern can only be considered in ontologies assuming that every property is *strictly typed*. Importantly, in an arbitrary ontology, *properties are not necessarily typed*: i.e., they do not necessarily have a specified class as domain and a specified class as range. Therefore, we cannot establish, in the general case, that a property relates one class to another class. As a consequence, considering the pattern introduced above, every *functional untyped property* would potentially allow to infer that two arbitrary classes are functionally dependent on each other, provided that the property relates one instance to, at most, a single other instance (i.e., that it is functional). The interested reader is addressed to [40] for further details in this issue.

In the AMDO tool we introduced an option to allow the user choose if he/she 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 Section 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 a 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., $F \in M_c{<}B{>}$), because any multiplicity would be allowed in the bridge-class side of the relationship (see Fig. 7). Once computed, we will also be able to know if we need to aggregate data through a *compatible* aggregation function (i.e., when $B \notin M_c{<}F{>}$).

Finally, we strongly recommend to enforce the theoretical patterns presented in this paper 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 design patterns are computed in a fully automatic way, and we benefit from DL reasoning techniques provided by DL reasoners.

Although we need to compute the transitive to-one closure by an ad hoc algorithm, most of our patterns can still 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++ [20]. This reasoner supports OWL DL except for nominals, 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*[11] (given two OWL DL assertions say if one is subsumed by the other).

Any of the reasoning tasks mentioned in this paper can be reduced to these 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 example, consider the EU-Car Rental ontology used as example in this chapter (see Fig. 2). With AMDO we have been able to identify 2069 dimensional concepts (note that it does not mean that the output 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

[11] Note, however, that class taxonomies and properties taxonomies are computed by a specific case of subsumption (the third reasoning task enumerated), but they are typically differentiated in commercial products.

this 2069 dimensional concepts were identified using reasoning (i.e., querying the EU-Car Rental TBox using FaCT++) while 694 out of 2069 were identified by our ad hoc algorithm.

Interestingly, it is possible to fully compute the design patterns presented in our method by means of DL generic reasoning. However, we would need to use a less expressive ontology language as AMDO input. The most promising ones form the DL-Lite family [32], which are well-behaved DLs 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 them. For example, regarding the EU-Car Rental example (see Fig. 2), it can be fully specified in DL-Lite through $DL-Lite_F$. In particular, DLs of the DL-Lite family do not allow for the specification of generalizations that are complete, cardinality constraints different from 1..*, 1..1, 0..*, 0..1, and the arbitrary use of functionality on roles and generalizations between roles; constructs that are absent in the EU-Car Rental example. The reader is addressed to [40] for a detailed description on how to compute AMDO's patterns in DL-Lite.

7. Conclusions

In this paper we have presented a novel approach to support the design of the data warehouse based on AMDO: an automatic method to identify concepts likely to play multidimensional roles, from an ontology representing our business domain. AMDO performs a thorough analysis of the data sources. Since supply-driven approaches are known to generate too many results, we propose to exploit AMDO's output by means of searching patterns. These patterns will guide the exploration of results obtained, and facilitate their comprehension. As discussed, AMDO is completely flexible when dealing with searching patterns, and we propose to use this guided exploration of the multidimensional knowledge contained in the sources to support the end-user requirements elicitation. In this way, we are already conciliating requirements with data available, and we also fully exploit the analysis capabilities of the sources.

This scenario is relevant since, in some cases, the users are not aware of the hidden analysis capabilities of their own data sources. In this sense, AMDO provides a framework to support the end-user requirements elicitation and once identified, automatically generate the multidimensional schema. Importantly, we have shown that, in such context, disposing of high-quality formalizations of the data sources we can overcome the fact of lacking of very expressive end-user requirements beforehand. However, although we do not decline that, eventually, the user will need to tune up the schemas produced, we claim that AMDO facilitates and gives support to the always hard and time-consuming design task of data warehouses.

Regarding AMDO, we have presented a set of novel 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 AMDO and thanks to the AMDO tool, we have also been able to discuss its behavior with a realistic case, in which turns to have a good answer time.

We believe this work to be the first to address the issue of supporting the multidimensional design from ontologies. Up to now, traditional approaches were typically carried out manually or were designed to work in a largely automatic way over relational sources. In our approach, AMDO carries out the data warehouse design process from a domain ontology (i.e., at the conceptual level), which improves the quality of the multidimensional schemas generated. Furthermore, working over ontologies opens new interesting perspectives. For example, we can extend the data warehouse and OLAP concepts to other areas like the Semantic Web and one consequence would be that, although 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.

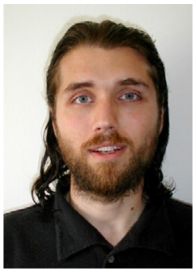
Acknowledgments

This work has been partly supported by the Ministerio de Ciencia e Innovación, under project TIN2008-03863, and the Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya.

References

[1] A. Abelló, J. Samos, F. Saltor, YAM^2 (Yet Another Multidimensional Model): an extension of UML, Inf. Syst. 31 (6) (2006) 541–567.
[2] R. Agrawal, A. Gupta, S. Sarawagi, Modeling multidimensional databases, Proc. of the 13th Int. Conf. on Data Engineering, IEEE, 1997, pp. 232–243.
[3] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, M. Zakharyaschev, Reasoning over extended ER models, Proc. of 26th Int. Conf. on Conceptual Modeling, volume 4801 of Lecture Notes in Computer Science, Springer, 2007, pp. 277–292.
[4] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.
[5] G. Berardi, D. Calvanese, D. Giacomo, Reasoning on UML class diagrams, Artif. Intell. 168 (1–2) (2005) 70–118.
[6] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Sci. Am. 284 (5) (2001).
[7] M. Böhnlein, A.U. vom Ende, Deriving initial data warehouse structures from the conceptual data models of the underlying operational information systems, Proc. of 2nd Int. Workshop on Data Warehousing and OLAP, ACM, 1999, pp. 15–21.
[8] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, S. Paraboschi, Designing data marts for data warehouses, ACM Trans. Softw. Eng. Method. 10 (4) (2001) 452–483.
[9] A. Calì, D. Calvanese, G.D. Giacomo, M. Lenzerini, A formal framework for reasoning on UML class diagrams, *Proc. of 11th Int. Symposium on Foundations of Intelligent Systems*, volume 2366 of *LNCS*, Springer, 2002, pp. 503–513.
[10] P.P.-S.S. Chen, The entity-relationship model: toward a unified view of data, ACM Trans. Database Syst. 1 (1) (1976) 9–36.
[11] J. Demetrovics, G. Katona, D. Miklós, Functional dependencies distorted by errors, Discrete Appl. Math. 156 (6) (2008) 862–869.
[12] D. Dori, R. Feldman, A. Sturm, Transforming an operational system model to a data warehouse model: a survey of techniques, IEEE Int. Conf. on Software-Science, Technology and Engineering (SwSTE 2005), IEEE Computer Society, 2005, pp. 47–56.
[13] L. Frías, A. Queralt, A. Olivé, EU-Rent Car Rentals Specification. Technical report, Departament de Llenguatges i Sistemes Inform'atics, 2003.
[14] D. Gaševic, D. Djuric, V. Devedžic, MDA-based automatic OWL ontology development, Int. J. Software Tools Technol. Transfer 9 (2) (2007) 103–117.
[15] P. Giorgini, S. Rizzi, M. Garzetti, Goal-oriented requirement analysis for data warehouse design, Proc. of 8th Int. Workshop on Data Warehousing and OLAP, ACM Press, 2005, pp. 47–56.

[16] M. Golfarelli, D. Maio, S. Rizzi, The dimensional fact model: a conceptual model for data warehouses, Int. J. Coop. Inf. Syst. 7 (2–3) (1998) 215–247.
[17] M. Golfarelli, S. Rizzi, Data Warehouse Design. Modern Principles and Methodologies, McGraw-Hill, 2009.
[18] O. Group. Unified Modeling Language (UML), Version 2.1.2. http://www.omg.org/technology/documents/formal/uml.htm.
[19] J. Hainaut, M. Chandelon, C. Tonneau, M. Joris, Contribution to a theory of database reverse engineering, Proc. of the 1st Working Conf. on Reverse Engineering, IEEE, 1993, pp. 161–170.
[20] I. Horrocks, Using an expressive description logic: fact or fiction? Proc. of 6th Conf. on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, 1998, pp. 636–649.
[21] B. Hüsemann, J. Lechtenbörger, G. Vossen, Conceptual data warehouse modeling, Proc. of 2nd Int. Workshop on Design and Management of Data Warehouses, CEUR-WS.org, 2000, p. 6.
[22] M.R. Jensen, T. Holmgren, T.B. Pedersen, Discovering multidimensional structure in relational data, 6th Int. Conf. on Data Warehousing and Knowledge Discovery, volume 3181 of LNCS, Springer, 2004, pp. 138–148.
[23] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross, The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses, John Wiley & Sons, Inc., 1998
[24] H. Lenz, A. Shoshani, Summarizability in OLAP and statistical data bases, Proc. of 9th Int. Conf. on Scientific and Statistical Database Management, IEEE, 1997, pp. 132–143.
[25] M. Lenzerini, Data integration: a theoretical perspective, Proc. of 21th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM, 2002, pp. 233–246.
[26] H. Mannila, K. Räihä, On the complexity of inferring functional dependencies, Discrete Appl. Math. 40 (2) (1992) 237–243.
[27] J.-N. Mazón, J. Lechtenbörger, J. Trujillo, A survey on summarizability issues in multidimensional modeling, Data Knowl. Eng. 68 (12) (2009) 1452–1469.
[28] J.-N. Mazón, J. Trujillo, J. Lechtenbörger, Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms, Data Knowl. Eng. 23 (3) (2007) 725–751.
[29] D. Moody, M. Kortink, From enterprise models to dimensional models: a methodology for data warehouse and data mart design, Proc. of 2nd Int. Workshop on Design and Management of Data Warehouses, CEUR-WS.org, 2000.
[30] A. Olivé, On the role of conceptual schemas in information systems development, Proc. of 9th Int. Conf. on Reliable Software Technologies (Ada-Europe 2004), volume 3063 of Lecture Notes in Computer Science, Springer, 2004, pp. 16–34.
[31] C. Phipps, K.C. Davis, Automating data warehouse conceptual schema design and evaluation, Proc. of 4th Int. Workshop on Design and Management of Data Warehouses, volume 58, CEUR-WS.org, 2002, pp. 23–32.
[32] A. Poggi, D. Lembo, D. Calvanese, G.D. Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semant. 10 (2008) 133–173.
[33] N. Prakash, Y. Singh, A. Gosain, Informational scenarios for data warehouse requirements elicitation, 23rd International Conference on Conceptual Modeling, ER 2004, volume 3288 of Lecture Notes in Computer Science, Springer, 2004, pp. 205–216.
[34] N. Prat, J. Akoka, I. Comyn-Wattiau, A UML-based data warehouse design method, Decis. Support Syst. 42 (3) (2006) 1449–1473.
[35] I. Reinhartz-Berger, Towards automatization of domain modeling, Data Knowl. Eng. 69 (5) (2010) 491–515.
[36] S. Rizzi, A. Abelló, J. Lechtenbörger, J. Trujillo, Research in data warehouse modeling and design: dead or alive? Proc. of ACM 9th International Workshop on Data Warehousing and OLAP, ACM, 2006, pp. 3–10.
[37] O. Romero, A. Abelló, Automating multidimensional design from ontologies, Proc. of ACM 10th Int. Workshop on Data Warehousing and OLAP, ACM, 2007, pp. 1–8.
[38] O. Romero, A. Abelló, A survey of multidimensional modeling methodologies, Int. J. Data Warehouse Min. (IJDWM) 5 (2) (2009) 1–23.
[39] O. Romero, A. Abelló, Automatic Validation of Requirements to Support Multidimensional Design, Data & Knowledge Engineering 69 (2010) 917–942.
[40] O. Romero, D. Calvanese, A. Abelló, M. Rodriguez-Muro, Discovering functional dependencies for multidimensional design, Proc. of ACM 12th Int. Conf. on Data Warehousing and OLAP, ACM, 2009, pp. 1–8.
[41] J. Schiefer, B. List, R.M. Bruckner, A holistic approach for managing requirements of data warehouse systems, 8th Americas Conference on Information Systems (AMCIS 2002), 2002, pp. 77–87.
[42] Y. Sismanis, P. Brown, P.J. Haas, B. Reinwald, Gordian: efficient and scalable discovery of composite keys, Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006), ACM, 2006, pp. 691–702.
[43] D. Skoutas, A. Simitsis, Ontology-based conceptual design of ETL processes for both structured and semi-structured data, IJSWIS, 2007, pp. 1–24.
[44] I.-Y. Song, R. Khare, B. Dai, SAMSTAR: a semi-automated lexical method for generating STAR schemas from an ER diagram, Proc. of the 10th Int Workshop on Data Warehousing and OLAP, ACM, 2007, pp. 9–16.
[45] H.B.K.T., Y. Zhao, Automated elicitation of functional dependencies from source codes of database transactions, Inf. Softw. Technol. 46 (2) (2004) 109–117.
[46] W3C. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/.
[47] R. Winter, B. Strauch, A method for demand-driven information requirements analysis in DW projects, Proc. of 36th Annual Hawaii Int. Conf. on System Sciences, IEEE, 2003, pp. 231–239.



Alberto Abelló has an MSc and a PhD in computer science from the Universitat Politècnica de Catalunya (Polytechnical University of Catalonia). He is an associate professor at the Facultat d'Informàtica de Barcelona (Computer Science School of Barcelona). He is also a member of the GESSI research group (Grup de recerca en Enginyeria del Software per als Sistemes d'Informació) at the same university, specializing in software engineering, databases and information systems. His research interests are database design, data warehousing, OLAP tools, ontologies and reasoning. He is the author of articles and papers presented and published in national and international conferences and journals on these subjects.

Oscar Romero has an MSc and a PhD in computer science from the Universitat Politècnica de Catalunya (Polytechnical University of Catalonia). Currently, he is an assistant professor at the Escola Tècnica Superior d'Enginyeria Industrial i Aeronàutica de Terrassa (Industrial and Aeronautical Engineering School of Terrassa). He is also a member of the GESSI research group (Grup de recerca en Enginyeria del Software per als Sistemes d'Informació) at the same university, specializing in software engineering, databases and information systems. His research interests are database design, data warehousing, OLAP tools, ontologies and reasoning. He is the author of articles and papers presented and published in national and international journals on these subjects.