Ontology-based structured web data warehouses for sustainable interoperability: requirement modeling, design methodology and tool

Selma Khouri [ab], Ilyes Boukhari [a], Ladjel Bellatreche [a], Eric Sardet [c], Stéphane Jean [a], Michael Baron [a]

[a]LISI/ENSMA - Poitiers University
Futuroscope - France

[b]National High School for Computer Science (ESI), Algiers - Algeria

[c]CRITT Informatique, Futuroscope - France

The spectacular growth of the Internet and its widespread adoption by worldwide corporations lead to an enormous quantity of heterogeneous, distributed and autonomous data sources. To facilitate the access to these huge amounts of data and make these sources interoperable, two technologies may be combined: *data warehousing* and *ontologies*. Data warehouses are designed to aggregate data and allow decision makers in these companies to obtain accurate, complete and up to date information. In the past decade, data warehouse technology (DWT) has been successfully applied in several domains such as telecommunication, retail, finance and many other industries. It supports a wide range of applications throughout the enterprise. The DWT has been largely used to offer sustainable solutions for enterprises. On the other hand, ontologies are models for specifying the semantics of concepts used by various heterogenous sources in a well defined and unambiguous way. Ontologies exist in various domains (E-commerce, Engineering, Tourism, etc.) and are used to increase interoperability between sources. They may be used to improve communication between decision makers and users collaborating together, by specifying the semantics of the used concepts. In this paper, we propose a methodology for designing data warehousing applications from various sources. Each source has its local ontology referencing a global one. For satisfying its local requirements and giving to sources more autonomy, each source may specialize/extend the global ontology. The presence of ontologies has three main contributions: (i) each owner of each source may use it to define his/her requirements, (ii) it reduces most important types of conflicts that may exist in sources and requirements (schematic and semantic) and (iii) it facilitates the sustainable urbanisation of the target data warehouse. Our methodology is supported by a case tool facilitating the tasks of data warehouse designers.

Keywords: Structured Web Data, Integration Systems, Ontology-based Databases, Requirement Engineering, Case Tool, OWL formalism, PLIB formalism.

1. Introduction

The rapid development of Information Technology solutions enforces the theory stating that the World is a Global Village. This theory breaks geographical barriers and distances between companies and organisations. Faced to this globalization phenomenon, enterprises have had to adapt their traditional behaviours to reach high level of competitiveness. To achieve this goal, enterprises need to share data and knowledge beyond their own boundaries by the mean of advanced Web technologies. New types of applications are emerging such as: *e-tailing, e-government, e-learning* following the concept of *e-entreprise*, *online services* like finance, publishing and marketing, all managed by web-based enterprises. We also face an outburst of start-ups and spin off that are blooming every day, which need to integrate existing web data into their own information systems. Additionally, recent Google research projects demonstrated that increasing quantities of these web data are structured. The prime example of such data is the deep web, referring to content on the web that is stored in databases and served by querying HTML forms (14). More recent examples of structure are a va-

riety of annotation schemes (e.g., Flickr, the ESP game, Google Co-op) that enable people to add labels to content on the web, and Google Base a service that allows users to load structured data from any domain they desire into a central repository (14). In such applications, web data are stored within structural databases. Sharing these mountains of web data efficiently between different companies collaborating together is a crucial issue that needs to be addressed. Note that these data are particularly heterogeneous, distributed, and autonomous and evolving in a dynamic environment.

Data integration is one of the relevant solutions that offers these functionalities to companies. It can be viewed as a process by which several heterogeneous sources are consolidated into a single data source associated with a global schema. It recently received a great attention from academic and industrial communities. This is due to many data management applications for example peer-to-peer applications, data warehouses (DW), E-commerce, and Web services. In a decisional context, DW integration systems are the most suitable solutions offering various analysis and visualization tools used within a decision-making process. A DW can be seen as an integration system, where relevant data of various sources are *extracted*, *transformed* and *materialized* in a warehouse (18) (Figure 3). A DW has additionally a multidimensional layer organizing its data into central subjects to analyze (*Fact concept*) according to different analysis perspectives (*Dimension concepts*). Dimensions can also be organized into hierarchies representing different levels of granularity. This multidimensional aspect enables various OLAP (*Online Analytical Processing*) analyses for decision makers, that traditional integration systems do not. DWs have been successfully applied in various applications fields like telecommunications[1], health care (62), and by different companies such as Continental Airlines, Wal-Mart, Toyota and Hewlett-Packard (43).

The main task in data integration is the identification of syntactic and semantic conflicts between heterogeneous data. Different categories of conflicts may be encountered at the *schema level* and at the *data level* and should be solved. Goh et al. (32) suggest the following taxonomy (Figure 1): *naming* conflicts, *scaling* conflicts, *confounding* conflicts and *representation* conflicts. For example a scaling conflict occurs when the price of a product is given in *Dollar* for a company in the United States or in *Euro* for a company in Europe.

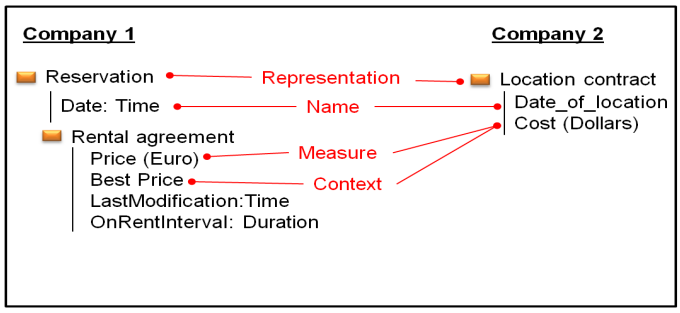


Figure 1. Types of semantics conflicts.

The lack of a generally agreed terminology followed by enterprises collaborating together has been recognized as the bottleneck for efficient integration (19). An effective approach to deal with integration and interoperability is the use of common standards. Many integration applications used domain ontologies as a reference model, where data reconciliation and mapping is obtained according to concepts of the domain ontology. Several ontology-based integration systems were proposed (16; 32; 45). Based on the way how ontologies are employed in these systems, (65), three different architectures are distinguished (Figure 2): (i) *single ontology methods* relating each source to the same global domain ontology, which limits considerably source schematic autonomy, (ii) *multiple ontologies methods* where each source has its own ontology developed without respect to other sources, which requires difficult tasks of inter-ontology mapping and (iii) *hybrid methods* where each source has its own ontology, but all ontologies are connected by some means

[1]Teradata company: http://www.teradata.com

to a common shared vocabulary. The main assumption of these systems is that all the sources use the same shared ontology. This assumption is relaxed by the emergence of many dedicated reference models covering many industrial areas. We can cite global standards and models developed in *tourism industry* that are used by big companies like *Open Travel Alliance* (www.opentravel.org) and *Hospitality Industry Technology Integration Standards* (www.hitis.org), or the standard *ISO 10303* for product information in industrial environments, etc. Those ontologies have been extensively used for traditional Web applications (Web services, pervasive computing, etc.) and Web 2.0, like ontologies FOAF[2] and SIOC[3] used for social networks domain. Ontologies are specified using different formalisms developed during last fifteen years like languages of the semantic Web (Daml+Oil, RDF, RDFS and OWL) for web applications, and *PLIB* (Parts Library) (54) formalism used for engineering applications. OWL is currently accepted as W3C recommendation for publishing ontologies on the Web. The PLIB formalism defines a domain-oriented (for engineering products) ontology model that has been developed during the 90's at the ISO level. The result is a standard series, ISO 13584.

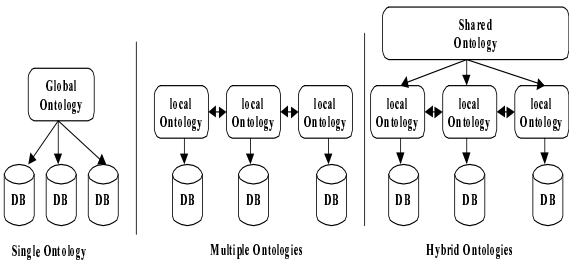


Figure 2. Different Ontology Architectures

The outburst of ontologies in various domains and their use by different companies leads to the creation of important amounts of web data referencing ontologies. These data are called Ontology-Based Data (OBD). Some solutions proposed to manage OBD in main memory like Jena (50) and OWLIM (44). Such systems allow fast loading and fast update of data but become very fastidious when data grows and a big amount of semantic data is available. To overcome this problem, database solutions have been proposed offering efficient storage and querying mechanisms for ontological data. The generated databases are called *ontology-based databases* (OBDB). OBDB are databases storing both ontology and data in the same database schema (figure 8).

OBDBs kept the attention of both industrial (like Oracle and IBM) and academic (OntoDB) communities, where different OBDBs were proposed like Rdfsuite (4), OntoDB (22), Sesame (12) and Oracle (21). Those OBDBs rapidly become candidate sources for the integration process. OBDB are suitable solutions for managing OBD for OLTP (OnLine Transactional Processing) applications, but not for OLAP applications. Following the same idea of OBDB, Ontology-Based Data Warehouses (ODBW) are adequate solutions for analyzing those ontology-based data. Our architecture scenario is in this case similar to hybrid integration methods of Figure 2 introduced previously. Furthermore, some recent research efforts in DW design proposed ontology-based methods and frameworks for the definition of the DW model. This is due to the role of domain ontologies in clarifying semantics of sources during the integration process, and their strong similarities with conceptual models (26). Ontologies can actually be seen as conceptual models of a whole domain via a logical theory offering reasoning capabilities that conceptual models do not have.

Additionally to sources, *user requirements* are actually recognized as an important component for DW design. Some design methods followed a *data-driven approach* generating the DW model only from data sources. Different studies identified limitations of this approach and proposed *requirements-driven* or *mixed design methods*. It is currently recognized that DW life cycle includes

[2]http://www.foaf-project.org
[3]http://sioc-project.org/

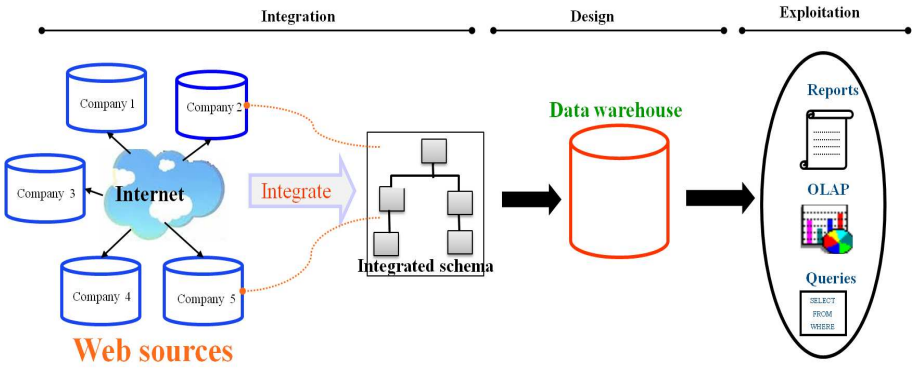


Figure 3. Traditional interoperability scenario by DWs.

a first phase of requirements analysis (35). We noticed furthermore that users' requirements are not only used during the design phase, but are also used during different exploitation phases of DW life cycle:

- requirements are saved a posteriori as users' queries in most DBMS (like Oracle or Sql Server) for optimization purposes,
- user's preferences, which are considered as non-functional requirements, are exploited for personalization and recommendation processes like in (29),
- quality of the DW can be measured according to its capacity to fulfil the different requirements and goals like in (64). Requirements are also traced in order to manage the evolution of the decisional application.

All these works, suppose the existence and availability of requirements. However, most of integration systems proposed (virtual and materialized) do not care sufficiently about user's requirements and do not give them their right value within the system developed.

We propose in our approach to specify user's requirements at the ontological level, and also to store these requirements in the DW repository. This is particularly important when we know that requirements analysis step is a tedious task requiring much time and effort from users and from designers. One cannot afford to redo this costly task more than once (Figure 4). Actually, the fusion between the two domains (Requirement Engineering and ontologies) aroused the interest of research community since the 80s where ontologies showed their effectiveness for requirements specification, their unification, their formalization and for reasoning about requirements. Ontologies allow requirements engineers to analyze a requirements specification with respect to the semantics of the application domain. They are useful for detecting incompleteness and inconsistency as sated in IEEE requirements specification (1). More concretely and in the same way as for data sources, ontologies are used to identify and manage semantic conflicts between requirements formulated by different users that do no share the same vocabulary.

Another dimension that has to be considered for integration systems is the complex dynamicity of the environment they evolve in, continually shaken by new requirements and business needs. This dimension is called sustainable interoperability, which goal is to make interoperability between companies easily adaptable to environment changes. Changes can be of different forms: changes occurring in sources, in business requirements or in the ontology. Changes can occur at two levels: data level and schema level. An effective evolution policy must be able to predict and limit certain changes during the design phase

of the integration system, and should be able to adapt the system according to additional changes occurring once the system is developed.

We propose in this paper a methodology for designing OBDWs using domain ontologies. The presence of sources and requirements specified at the ontological level is used: (i) during the integration process to eliminate semantic conflicts between sources and requirements, (ii) during design phase to specify the different deign models (conceptual and logical) and (iii) to forecast the evolution of the system once developed.

1.1. Contributions

The main contributions of this paper are:

1. A new classification of DW design methods, showing their convergence to ontology-base conceptualisation.
2. A methodology for designing DWs from OBDB and users's requirements supported by a case tool.
3. A new structure for representing both data and requirements and their respective semantics in the OBDW.
4. A discussion about different DW evolution scenarios, and how our defined OBDW can prevent environment changes.

1.2. Organization of the Paper

The rest of this paper is organized as follows. Section 2 presents the case study that will be used as an illustrating example that we consider along the paper. Section 3 presents the background required for facilitating the understanding of our approach. Section 4 presents the related work, where a classification of the different DW design methods is proposed. Section 5 presents our DW design methodology comprising a description of a proposed requirements model, the design process used to make requirements persistent within the DW model, and the definition of the OBDW. Section 6 describes a tool implementing the design methodology proposed. Section 7 discusses different evolution scenarios of the developed system. Section 8 concludes the paper and suggests some future issues.

2. Case study

The ontology depicted in figure 5, that will be used as example along the paper, refers to the rental car ontology[4], representing cars reservation domain. Our scenario is that an online company is managed in a given region but have *Branches* based in different countries. *Customers* of this company from a given *Country* can rent a *Car* of a given *CarModel*. Once the request validated, a *Rental agreement* is made. The *Reservation* of the car has a certain *Duration* and can be *Cancelled* according to certain conditions.

Decision makers of this company aim at performing OLAP analysis related to sales of agreements. For this reason, they specify some decisional requirements such as:

- *Pricing Summary Report* provides a report of all reservations made at a given reservation date. The report lists totals of bestPrice, basicPrice and rate of guaranteed. Note that the rate is equal to the sum [basicPrice - bestPrice].
- This requirement aims at retrieving the 10 reservations with the highest value for all customers in a given region.
- This requirement aims at retrieving the 5 branches of all regions with highest number of customers having cancelled their reservation.
- This requirement consists in finding the average price of reservations of a given car model.

3. Background

In this section, we present in details four concepts that we consider important for our work: *Ontologies*, *OBDBs*, *Materialized integration systems* (DW) and *Requirements*. We will explain each concept in the following sections.

3.1. C1: Ontologies

Ontologies have been defined for various domains for a wide range of applications. As a con-

[4] www.lsi.upc.edu/õromero/EUCarRental.owl

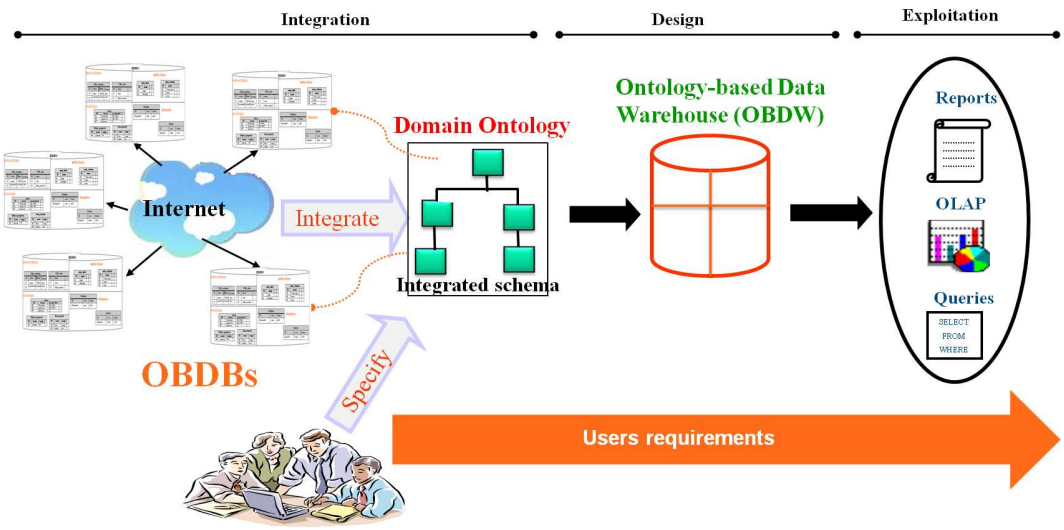


Figure 4. New interoperability scenario by DWs.

sequence, existing ontologies are not all alike. In order to define precisely the type of ontologies needed in our work, we propose the following taxonomy of ontologies.

3.1.1. Taxonomy: the onion model

Two main categories of ontologies have emerged in the literature (55). *Conceptual Ontologies* (CO) represent object categories and properties that exist in a given domain whereas *Linguistic Ontologies* (LO) represent the terms used in a given domain eventually in different natural languages. However existing COs present quite different characteristics according to the ontology model used to define them and to the applications they are designed for. As a consequence we have divided this category into two subcategories. This new classification is based on the notions of primitive and defined concepts given by Gruber (36). Primitive concepts define the border of the domain conceptualized by the ontology. Each concept in the domain is represented in a unique way by a primitive concept. Primitive concepts are the foundations on which new concepts called defined concepts can be introduced. These defined concepts are specified by a complete axiomatic definition expressed in terms of other concepts (primitive or defined). For example the concept *Car-Reservation* is defined as a type of *Rental-Agreement* concept. Based on this distinction between primitive and defined concepts, our ontology taxonomy is composed of three categories:

- *Canonical Conceptual Ontologies* (CCO) represent all the concepts in a given domain in a single way without any redundancy. Thus, CCOs only include primitive concepts. An example of a CCO for electronic components can be found in (IEC61360-4,1999). Defining a canonical (non redundant) vocabulary in a given domain is particularly useful for data exchange. For example, in the STEP project, exchange models are defined in the EXPRESS language as a STEP Application Protocol (AP). These canonical exchange models are used by industrial users to exchange product descriptions between different organizations. Thus, CCOs can also be used as exchange models. PLIB projet (54), which is the continuity of STEP project, aims at providing a canoni-

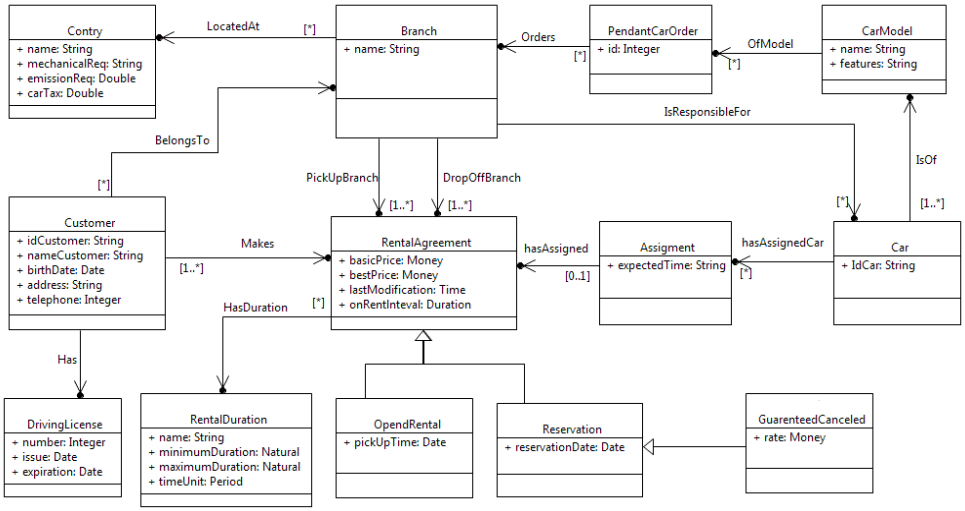


Figure 5. Rental car ontology model.

cal representation of concepts of a given domain. In that representation context, properties play a key role. First, classes are only specified when they are required to define a property domain. Secondly, each property is defined in the domain of a class, and is meaningful only for that class and its possible subclasses.

- *Non Canonical Conceptual Ontologies* (NCCO) include not only primitive concepts but also defined concepts. Using defined concepts, different reasoning tasks can be performed on the ontology (e.g., classification of classes or instances). Moreover, defined concepts can also be used by a user to query and manipulate the ontology. NCCO are supported by different languages of the semantic web like RDF and OWL.

- *Linguistic Ontologies* (LO) define the terms that appear in a given domain. LOs include relationships between terms such as synonymous-of or homonymous-of. They are useful for system-user communication as well as providing the terms used in a given domain in different natural languages. *Wordnet*[5] is a well-known example of such ontologies.

Currently ontology design focuses on one of these three categories, the canonical, non canonical or linguistic part. However, as we have seen previously, these three categories of ontologies are complementary. The model, presented in Figure 6, called the *onion model* (37), illustrates how these three categories of ontologies can be combined. The CCO layer provides the foundation to represent and exchange the knowledge of a domain. It is extended by a NCCO layer to map different conceptualization made on this domain. Finally the LO layer is the natural language representation of all the concepts defined in the CCO and NCCO layers.

Ontologies that we use in our work follow this model. In next sections we will see how we can benefit from the different layers of an ontology that follows this model.

3.1.2. Formal Definition

We focus in our study on conceptual ontologies, whose schema can be formally defined as follows:

[5] http://www.cogsci.princeton.edu/wn

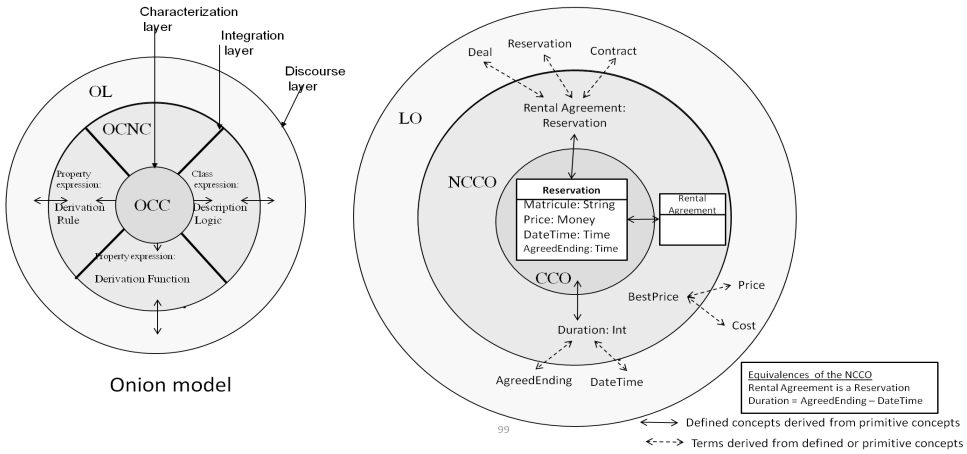


Figure 6. The onion model of domain ontologies.

(9): $\mathcal{O} :< \mathcal{C}, \mathcal{P}, \mathcal{S}ub, \mathcal{A}pplic >$:

- $\mathcal{C}$ is the set of classes describing the concepts of a given domain.
- $\mathcal{P}$ is the set of properties describing the instances of $\mathcal{C}$.
- $\mathcal{S}ub : \mathcal{C} \rightarrow 2^{\mathcal{C}}$ is the subsumption function, which associates each class C_i to its direct subsumed classes. Two subsumption relationships are introduced in our framework: (i) OOSub: describing the usual subsumption of inheritance relationship, where the whole set of applicable properties is inherited. (ii) OntoSub: describing a subsumption relationship without inheritance, where a part of the whole applicable properties may be imported from a subsuming class to the subsumed one (Figure 7). OntoSub is the formal operator of modularity. OntoSub is formalized by 'case-of' relationships in PLIB formalism. Similar mechanisms exist in OWL ontologies. (66) gives an overview of different modularity methods defined for OWL ontologies.
- $\mathcal{A}pplic : \mathcal{C} \rightarrow 2^{\mathcal{P}}$ is a function that associates to each ontology class, the properties that are applicable for each instance of this class.

3.1.3. Ontologies VS Conceptual models

Conceptual ontologies can be seen as the continuity of conceptual models describing a whole domain. Ontologies have actually been used for designing various database applications. The first works proposed to use ontologies in the design of traditional DBs. (63) proposed a method similar to Chen methodology (20) for designing databases by adding a semantic layer and proposed to assist designers to define conceptual models using linguistic ontologies. (24) proposed an extension of the ANSI/X3/SPARC architecture giving ontologies their place during design process of database scheme. (8) proposed a methodology for designing OBDB covering conceptual and logical design phase basing on functional dependencies embedded in the used ontology. Ontologies are effectively similar to conceptual models as they both define a conceptualization of the universe of discourse using a set of classes and properties, but ontologies have specific characteristics that gives the added value:

- *Modeling Objective*: An ontology *describes* concepts and properties of a domain independently of any objective. A conceptual

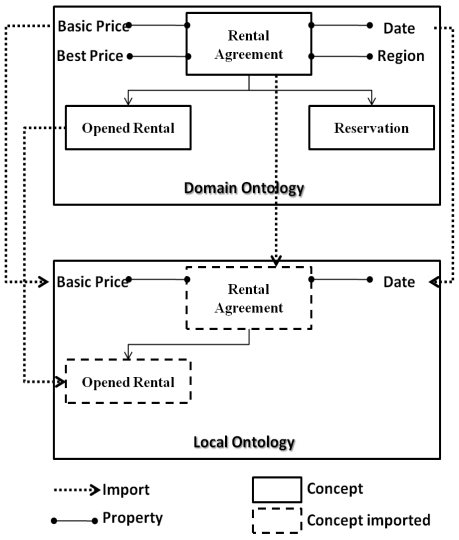


Figure 7. OntoSub operator.

model *prescribes* concepts and properties of a domain according to applicative objectives.

- Consensuality: the consensual aspect that characterizes ontologies facilitates the designer task by offering him a global view of the domain of interest.

- *Concepts identification*: data in a conceptual model have a meaning only in the particular context for which the model is defined. Concepts in an ontology have universal identifiers and can be referred from any other context. This eases ontology usage for data exchange or integration.

- *Reasoning*: ontologies are formal models offering reasoning capabilities that can be used either to check consistency of information or to deduce new information from existing facts.

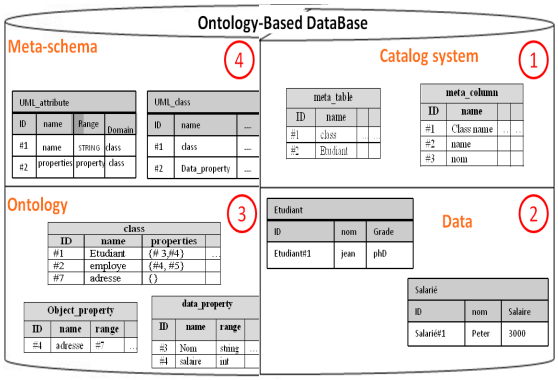


Figure 8. General architecture of OntoDB.

3.2. C2: Ontology-Based DataBase

3.2.1. Storage of ontologies in databases: the OntoDB ontology-based database

As we said in the introduction, several OBDB have been proposed by industrial and academic communities. In this paper, we consider OBDB followed OntoDB architecture. This architecture is composed of 4 parts as illustrated in Fig. 8.

Parts 1 and 2 are traditional parts available in all DBMSs, namely the *meta-base* part that contains the system catalog and the *data part* that contains instance data. Parts 3 (*ontology*) allows to represent ontologies in the database. The ontology model supported by this architecture is the PLIB ontology model but we will see that the part 4 of this architecture can be used to adapt and extend this architecture to other ontology models such as OWL. The part 4 (*meta-schema*) is specific to OntoDB. It records the ontology model into a reflexive meta model. For the ontology part, the meta-schema part plays the same role as the one played by the meta-base in traditional databases. Indeed, this part may allow: (1) generic access to the ontology part, (2) support of evolution of the used ontology model, and (3) storage of different ontology models.

Let us now see how the different concepts are stored in the 4 parts of OntoDB. We use the example presented in Fig. 9.

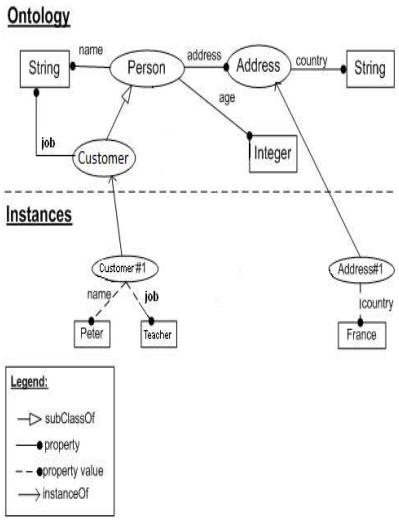


Figure 9. An toy example of an ontology and its instances

Customer			
ID	name	age	address
Cust#1	Name1	Age1	Adrs#1
...	...		

Address	
ID	Country
Adrs#1	France
...	...

Figure 10. Storage of instance data in the data part of OntoDB

Part 2 stores instance data following a relational approach. Indeed, all instances of a given ontology class is stored in a single table. However, contrary to relational databases, all properties of a given class may not be valued by instances of this class. Thus, the table used to store instances of a class only includes columns for the properties that are used at least by one instance of the class. Fig. 10 shows an example of the storage of the instances of the *Customer* and *Address* classes of an ontology.

Part 3. of OntoDB stores ontologies using a relational schema defined according to the ontology

Class	
ID	Name
1	Person
2	Customer
3	Adress
...	...

SubClassOf	
ID	VALUE
2	1
...	...

Property	
ID	Name
1	name
2	age
3	adress
...	...

Domain	
prop	class
1	1
2	1
...	...

Range	
prop	type
1	xsd:string
2	xsd:integer
3	xsd:string
...	...

Figure 11. Storage of ontologies in the ontology part of OntoDB

Entity			Attribute				Type	
ID	Name	SuperEntity	ID	Name	Domain	Range	ID	Name
Entity#1	Resource		Att#1	name	Entity#1	Type#1	Type#1	String
Entity#2	Class	Entity#1	Att#2	domain	Entity#3	Type#2	Type#2	Entity#2
Entity#3	Property	Entity#1	...	...	...	...		

Figure 12. Storage of ontologies in the meta schema part of OntoDB

model supported. Fig. 11 presents the main tables of this schema. These tables are used to store the hierarchy of classes as well as the properties of the ontology.

Finally, *part 4* stores the ontology model used to define ontologies of part 3. Fig. 12 presents the storage of the main components of an ontology model i.e., constructors of classes and properties. This schema can be extended to support a specific ontology model.

Managing all the data stored in OntoDB using SQL would be difficult since this language is based on the relational model and thus does not include any operators for ontologies. As a consequence we have designed a specific language called *OntoQL*.

3.2.2. Querying ontologies and its instances: the OntoQL query language

Specific languages have been designed to query ontologies and their instances such as SPARQL[6]. The OntoQL language that we have proposed has three main characteristics that distinguish it from

[6]http://www.w3.org/TR/rdf-sparql-query/

other proposed languages: (1) the OntoQL language is independent of a given ontology model. Indeed, it is based on a core ontology models containing the constructors shared by different ontology models and this core ontology model can be extended by the OntoQL language itself, (2) the OntoQL language exploits the linguistic information that may be associated to a conceptual ontology allowing to express queries in different natural languages and (3) the OntoQL language is compatible with the SQL language and thus, it allows to exploit data at the logical level of an OBDB. Moreover, it extends this language to exploit data at the ontological level, independently of the logical representation of the data, and still to manipulate the structure of these data from this level.

Let's now see different OntoQL statements that allow users to manipulate all the data of OntoDB. These statements are based on the ontology example shown in Fig. 9.

Queries one instance data can be expressed. For example the following query searches the name and country of all the direct instances of the Person class (keyword ONLY).

```
SELECT name, address.country
FROM ONLY (Person)
```

As we can see, the OntoQL language has a syntax similar to SQL and provides operators to navigate through the hierarchy of classes (by default, a query searches on all direct and indirect instances of a class) and the composition of properties through path expressions (address.country). The OntoQL language exploits the LO layer of an ontology to allow user to express their queries in different natural languages. For example if all the concepts of our example of ontology are associated to a term in French defined in the LO part of the ontology, the previous query can be expressed as follow

```
SELECT nom, adresse.pays
FROM ONLY (Personne)
USING LANGUAGE FR
```

The OntoQL language also supports query on ontologies. For example the following query searches the names in French and in English of all the classes of the ontology.

```
SELECT #name[fr], #name[en]
FROM #class
```

As we can see all the elements of the ontology level (e.g, class or name) are prefixed by #; This prefix is used to distinguish query on instances from query on ontologies.

Since query on ontologies and query on instances can be expressed, these two capabilities can be combined in OntoQL. The following query exploits this capability to find all direct and indirect instances of the Person class showing in the same time the name in English of the direct class the instance belongs to.

```
SELECT p.name, TYPEOF(p.#name[en])
FROM Person p
```

The *TYPEOF* operator retrieves the belonging class of an instance. Thus this query retrieves all persons and it shows if this person is a customer.

The OntoQL language is also equipped with a definition and manipulation language. In particular, the definition language can be used to extend the ontology model used to define ontologies. For example the following statement extends the ontology model with the AllValuesFrom constructor from the OWL ontology model.

```
CREATE ENTITY #OWLAllValuesFrom UNDER #Class
(#onProperty REF(#Property),
#allValuesFrom REF(#Class))
```

An OWL restriction is a particular class. Thus the new constructor #OWLAllValuesFrom extends the basic constructor of class (#Class). An OWL restriction is defined on a particular property (#onProperty) and takes its values from a given class (#allValuesFrom). This new constructor is added to the meta schema part of OntoDB.

To conclude this part, we show in Fig. 13 the complete architecture of our software to handle ontologies and their instances. OntoDB stores all the data and OntoQL provides an access at the knowledge level to query and exchange these data.

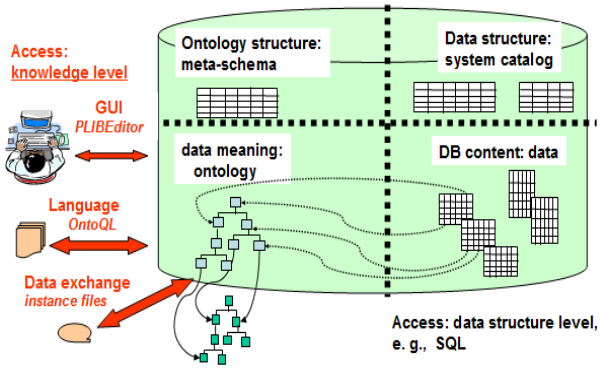


Figure 13. OntoDB and its associated access layers OntoQL and OntoML

3.3. C3: DWs seen as materialized integration systems

DWs are actually considered as *integration* systems materialized by *multidimensional* concepts for facilitating OLAP analysis. We explain these two aspects in the following sections.

3.3.1. Integration

The ontology used in our design approach is a conceptual domain ontology covering and integrating the set of OBDBs. Formally, a data integration system is a triple $\mathcal{I} :< \mathcal{S}, \mathcal{G}, \mathcal{M} >$, where $\mathcal{S}$ is a set of source schemas which describe the structure of sources participating in the integration process, $\mathcal{G}$ is the global schema which provides a reconciled and an integrated schema and $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$ which establishes the connection between the elements of the global schema and those of the sources (9).

In our approach, $\mathcal{S}$, $\mathcal{G}$ and $\mathcal{M}$ are defined as follows:

- $\mathcal{S}$ is represented by the set of $OBDB$ containing their own local ontology $O_i :< C_i, P_i, Sub_i, Applic_i >$ defined from a global shared ontology,
- $\mathcal{G}$ is the global ontology (GO) formalized as $GO :< C_p, P_p, Sub_p, Applic_p >$
- $\mathcal{M}$ is the mapping between GO and $OBDBs$ represented by the subsomption relationship $OntoSub_{i,p} : C_p \rightarrow 2^{C_i}$ between GO et O_i that associates to each class c_p of C_p the set of classes $c_i \in C_i$ that are subsumed directly by c_p:
$\forall c_p \in C_p, OntoSub_{i,p}(c_p) = \{c_i \in C_i | (c_p$ subsume $c_i) \wedge (\forall c'_i | c_i \in Sub_i(c'_i) \Rightarrow c'_i \notin OntoSub_{i,p}(c_p)) \wedge (\forall c'_p \in Sub_p(c_p) \Rightarrow c_i \notin OntoSub_{i,p}(c'_p))\}$.

Several automatic integration scenarios may be defined in this context. We analyzed and formalized in (9) three scenarios corresponding to different articulations between locale ontologies and the global one: (i) *FragmentOnto* that assumes that the shared ontology is complete enough to cover the needs of all local sources. Each local ontology is a fragment of the shared ontology. (ii) *ExtendOnto* where the integrated ontology is composed of the shared ontology extended with all the local specialization of the variations of local sources. (iii) *ProjOnto* where each local source defines its ontology by projecting its instances onto the applicable properties of its smallest subsuming class in the shared ontology. We focus for the rest of the method on the global ontology *GO* covering the available OBDBs.

Additionally to integration methods, and to prepare data integration process, some exchange based formats have been proposed to facilitate and automate data exchange such *OntoML*. OntoML stands for "Product Ontology Mark-up Language" (ISO13584-32). It has been developed in the ISO TC184/SC4 framework. It is an XML Schema designed for use by applications that need to exchange and process PLIB compliant domain ontologies, possibly together with their related instances, in various Web-oriented environments. When designing OntoML, an underlying idea was to clearly separate the representation of ontology concepts from the representation of instances. Indeed, in an engineering context, what is important is to ensure that instances exchanged between business partners can be interpreted without any ambiguity, whatever be the underlying ontology language used to describe the related classes and properties (product exchange interoperability). For that purpose, the

ISO 29002 series has been developed. It aims to provide general mechanisms guaranteeing that the concept information can be exchanged independently from the data model of the underlying ontology model. Therefore, ISO 2002 provides (among other things) a neutral concept identification mechanism (ISO29002-5) and a neutral instance representation mechanism (ISO29002-10) (reference to the underlying class associated to a set of property reference/typed value couples).

3.3.2. Multidimensional paradigm

Multidimensional design organizes data into *Facts* (subject of analysis) and *Dimensions* (analysis perspectives). Dimensions can form hierarchies, where each level of the hierarchy represents a level of granularity. Fact tables are composed of measures (fact attributes) and dimensions are composed of dimension attributes. Different studies proposed models following the multidimensional paradigm. After studying different multidimensional normal forms proposed, we retain the following multidimensional constraints:

- Data summarization: data summarization guarantees a correct aggregation of data (46) by means of three necessary conditions agreed by most multidimensional models: (i) *Disjointness*: meaning that the sets of objects to be aggregated must be disjoint, (ii) *Completeness*: meaning that the union of subsets must constitute the entire set, and (iii) *Compatibility* of the Dimension, the kind of measure being aggregated and the aggregation function used. Many design methods ensure the first two conditions by looking for many-to-one relationships between each fact and its dimension and a one-to-many relationship between levels of the same hierarchy. The compatibility condition requires to access semantics of data to check whether it is possible to apply aggregation operation to aggregate measures along dimension levels. These three conditions are considered as the first step ensuring the quality of the DW model and some studies used them to propose multidimensional normal forms.
- Dimension hierarchies: (48) enumerates and classifies different types of dimension hierarchies (simple, multiple, generalized). A multidimensional model should represent these different hierarchies. When translating hierarchies to the logical level, some of them may have the same relational representation. It is thus important to keep trace of dimension semantics within the DW.

Three different *logical* implementations of a multidimensional model are possible: *Multidimensional OLAP* approach (MOLAP), *Relational OLAP* approach (ROLAP) and *Hybrid OLAP* approach (HOLAP). *MOLAP* approach stores multidimensional data in array-based multidimensional storage and implement the OLAP operations over these special data structures. Each dimension of the DW model is represented as an array dimension. MOLAP approach offers good access times, but the array update and management are more difficult.

ROLAP approach maps DW multidimensional model into relational tables. Each fact concept becomes a fact table having measure attributes and each dimension concept becomes a dimension table having dimension attributes. Fact tables store secondary key attributes of dimensions tables related to the fact. ROLAP servers include optimization, scalability, implementation of aggregation navigation logic and additional services, which make the relational approach the most popular representation adopted by several important DBMS editors. ROLAP systems model DW applications by means of a star schema or its variant. A star schema consists of a huge fact table and a number of dimension tables (14). The fact table is joined to dimension tables. The snowflake schema is a variant of the star schema where dimension tables are split into several dimension level tables according to their granularity.

HOLAP approach combines both MOLAP and ROLAP approaches to get benefits from the scalability of ROLAP and the faster computation of MOLAP. HOLAP manages frequently used data (generally aggregated data) in a MOLAP system and other data in a ROLAP system.

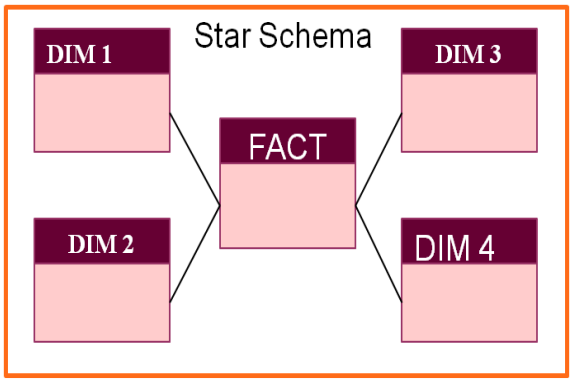


Figure 14. DW star schema

3.3.3. C4: Requirements engineering

Requirements engineering (RE) plays a crucial role in information systems development processes to reduce the risk of failure. Requirements of an enterprise-wide DW system determine its functional behaviour and its available information, for example what data must be accessible, how it is transformed and organized, as well as how it is aggregated or calculated. Requirements enable stakeholders to communicate the purpose, establish the direction and set the expectations of information goals for the enterprise. On the other hand, the development team of a data warehouse system expects a complete, correct and unambiguous specification of the system to build, which means a further refinement of the business requirements from the stakeholders.

Generally, a RE process can be divided into four activities: (i) requirements elicitation, (ii) requirements specification, (iii) requirements validation and (iiii) requirements management. The requirements specification presents a significant stage that relates to the requirements formalization. A good requirements specification implies thus a good DW design. There are several techniques for requirements specification to assist requirements analysts and stakeholders in producing requirements specification of higher quality, and some of them are put into practice in industry:

- Informal techniques: are built in natural language, sometimes with structuring rules. Their use introduces risks of ambiguities because neither their syntax, nor their semantics are perfectly defined. Among these techniques, we can quote questionnaires and interviews.

- Semi-formal techniques: are generally based on graphic notations with a specified syntax allowing having a clear vision of the system. These models are good vectors of communication between designers and users system. Among these techniques we can quote UML notation.

- Formal techniques: are based on mathematical or logical notations which provide a precise and no ambiguous framework for requirements modelling. We can quote models using the B notation and ontological models.

4. Related work: Towards a conceptual continuity of DW design methods

Many recent studies proposed to combine DW and ontology technologies principally for the following tasks: Extract-Transform-Load (ETL) process, validation of the multidimensional structure, and DW design. Some proposals exploit ontologies for the ETL phase: (15) proposed ETL method for integrating data from sources into the DW using an ontology as a global schema describing sources, (60) proposed to automate the ETL process by constructing OWL ontology linking schemas of sources to the DW target schema. In another perspective, some proposals use ontologies for the validation or extension of the DW multidimensional schema: (40) proposed to use hyperonymy and hyponymy relationships of Wordnet ontology to enrich and complete dimensions hierarchies. To the best of our knowledge, only *three studies proposed the use of ontologies for designing DW applications* (57; 51; 41). Before describing these three works, we first present a classification of DW design methods that shows their convergence to the ontologi-

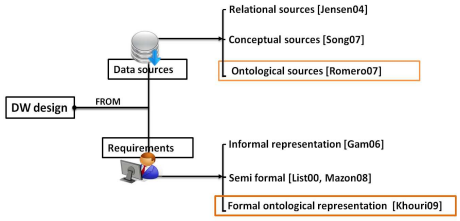


Figure 15. Classification of DW design methods.

cal methods through a conceptual continuity approach.

When exploring the main DW design works, we notice a trend of methods following a supply-driven approach, where the DW model is generated from an analysis of *data sources* (38; 61). User's requirements are ignored in these works. The domain of requirement analysis is related to *Requirements Engineering* (RE). RE for DW design aims at identifying needs of users and decision makers to implement a DW satisfying them. The experience showed that revisiting requirements specification is generally needed even after DW implementation[7]. Many methods (47; 67), called *demand-driven methods*, were proposed to generate a DW model after analysing users' requirements. Other methods (11; 30), called *mixed methods*, generate the DW model from both sources and requirements.

These methods generate DW model either from logical schemas of sources (38), or from conceptual schemas of sources (61; 34) requiring generally reverse engineering efforts, or recently from ontological schemas (Figure 15). Ontologies have been proposed in DW design, essentially as a solution to data integration problems caused by *semantic heterogeneities*. These ontological methods are mainly *supply-driven methods*, where domain ontologies are used as conceptual schemas integrating data sources. (57) defines the DW multidimensional model (facts and dimensions) from an OWL ontology by identifying functional dependencies (*Functional ObjectProperties*) between ontological concepts. (51) defines a framework for analyzing annotations of the semantic web by designing a semi-structured DW.

From requirements perspective, we notice that proposed methods define DW schema from informal requirements models (28), or from semi-formal models (generally UML models or models of i*[8] framework) (49; 47; 13). These methods consider users' requirements as the set of specifications of the given application. Ontologies on the other hand can be seen as the set of specification of the entire domain. We proposed in (41) an ontological design method that consists in defining a conceptual model from an OWL domain ontology integrating data sources and specifying *requirements*. This method is also supported by a design tool (42). Users' requirements in this method are expressed in terms of concepts of the domain ontology to generate a DW ontology, from which a multidimensional model is defined. Following the hybrid approach, (58) also proposed a mixed method producing a multidimensional model from an OWL ontology describing sources. Requirements are then used to identify the ETL operations needed for mapping the sources to target data stores.

In this paper, we propose an ontological design method following a *mixed approach*, extending our previous method (41), where we define a new DW storage structure representing requirements *persistently*. Requirements analysis in DW design litterature can differ according to the object analyzed. We distinguish process-driven, user-driven or goal-driven analysis. Process-driven analysis (47; 13; 59) analyzes requirements by identifying business processes of the organization. User-driven analysis (67) identifies requirements of each target user and unifies them in a global model. Goal driven analysis (11; 30; 40) identifies goals and objectives that guide decisions of the organization at different levels. It is recognized by different authors that goal-driven analysis provides a good definition of users requirements (56).

[7] http://www.redbooks.ibm.com

[8] i* is a framework using notions of Goals and Agents. i* is used for identifying users of the system to develop, and represent them into social actors dependent each other through the goals to realize, tasks to be made and resources used.

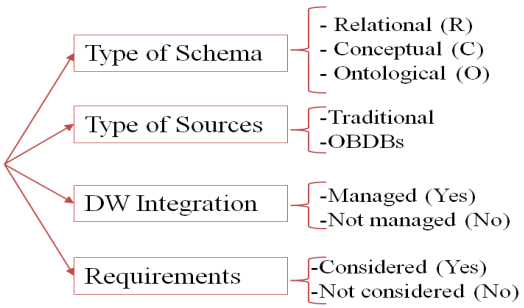


Figure 16. Criteria for analyzing related works.

Identifying goals of users at the beginning of the project is an important step in the development process, especially in decisional applications that needs to analyze the activity of an organization and where goals is an important indicator of this activity. To define requirements, we thus followed the *goal-driven approach* that has been used in DW context (11; 30).

According to the four concepts detailed in the background section (Schemas, Sources, Integration, Requirements) (See Figure 16), we summarize the most relevant works discussed in Table 1:

Our proposed approach defines the DW from the *global ontology integrating* available *OBDB sources*. *Requirements* are specified at the ontological level, and are made persistent in the DW structure.

5. Our Design Approach

Our objective is to define an OBDW making users requirements persistent within the DW structure. To fulfill this objective, we need to define: (i) a requirements model, (ii) the process making those requirements persistent and (iii) the structure of the OBDW proposed. The following sections will describe each step.

5.1. The requirements model

The requirements model we propose follows a goal-driven approach. A goal is an objective that the system under consideration should achieve. Goal-oriented requirements specification produces the so called goal graphs, which represent goals and their logical relationships in an AND-OR graph form (18). From our analysis of various existing works suggested in goal-oriented litterature, we propose our Goal-oriented model to assist DW analysts and stakeholders in producing requirements specification of high quality. We formalize a goal based on the GQM definition (Goal Question Metric) (6): "A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment".

Fig. 17.b presents the goal oriented requirements model we defined, represented as an UML class diagram designed with *Eclipse Papyrus plugin*. This model is composed of a main class *GoalModel* that stores all *Goals*, and is aimed at promoting requirements reuse.

Let us consider the first requirement of the case study $Goal_1$: "*This requirement provides a report of all reservations made at a given reservation date. The report lists totals of bestPrice, basicPrice and rate of guaranteed cancelled. Note that the rate is equal to the sum [bestPrice - basicPrice].*". A goal is described by the following properties: goal identifier (Id), name ($Goal_1$), description, purpose (*Report*) and its priority (mandatory or optional goal). Three reflexive relationships between goals are distinguished: (i) Influence relationships: positive or negative influence relationships exist between goals. For example $Goal_i$: *Reduce number of cars for branch C* negatively influences $Goal_j$: *Increase rental agreements for branch C.* (ii) ParentChild relationships: decomposing a general goal into subgoals using AND/OR relations, (iii) GoalRelation representing relations of conflicts, refinement, containment and equivalence between goals.

A goal is characterized by three classes related to Goal class through aggregation relationships, that we call **coordinates** of the goal. They are used for easing the gathering of the goals from users. Coordinates are: a *Metric* measuring the satisfaction of a goal (*sum [bestPrice - basicPrice]* for $Goal_1$, a *Result* to analyze (*bestPrice, basicPrice and rate* for Goal1), and Parameters reflecting the object of the goal (*reservation date*

Authors/Criteria	Type of Schema	Type of Sources	DW Integration	Requirements
(34)	Conceptual or relational	Traditional	No	No
(38)	Relational	Traditional	No	No
(11)	Conceptual	Traditional	No	Yes
(30)	Relational	Traditional	No	Yes
(57)	Ontological	Traditional	No	No
(58)	Ontological	Traditional	Yes	Yes
(51)	Ontological	Ontology-based	No	Yes
(41)	Ontological	Traditional	No	Yes

Table 1
DW design works analyzed following 4 criteria

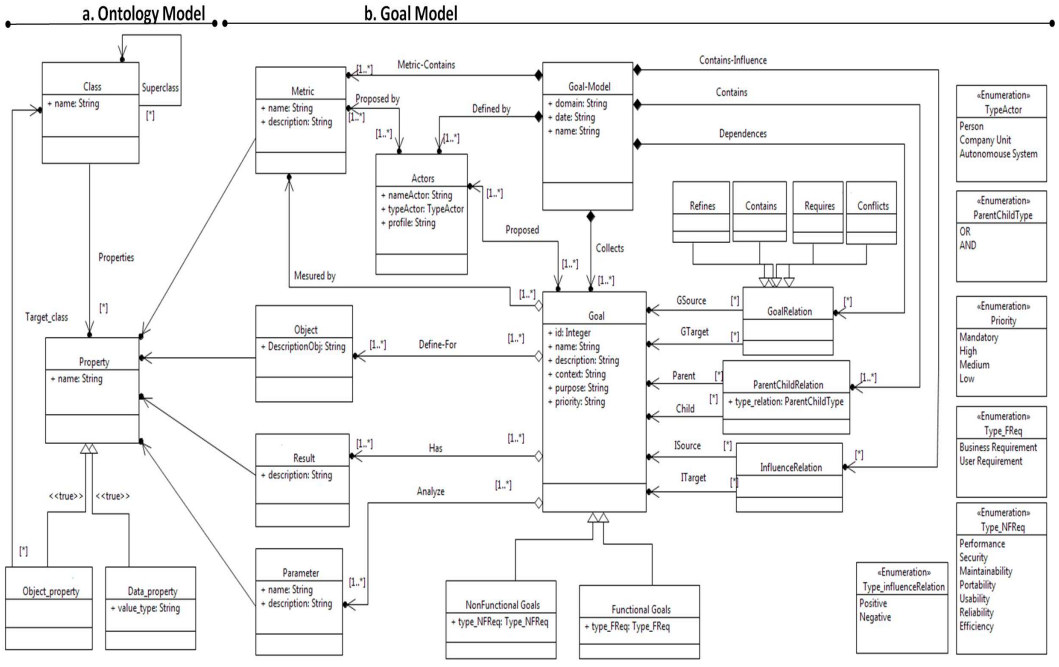


Figure 17. Requirement model connected to the ontology model

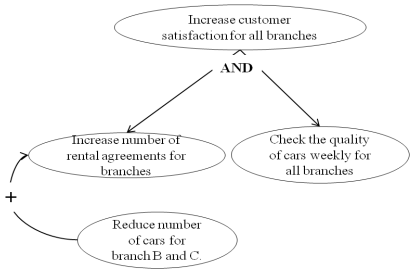


Figure 18. Graph of goals of the rental car domain

for $Goal_1$).

Two types of goals are identified: *functional* and *non-functional*. A non-functional requirement is defined as an attribute or constraint of the system (such as security, performance, flexibility, etc) (31). Each goal involves one or more actors that interact with the system to fulfil the goal (eg. *Sales manager* for $Goal_1$). Having a set of goals, a graph of goals can be produced as presented in figure 18. The graph is composed of the different goals as nodes and the described relationships between goals as edges.

Users' goals are expressed at the ontological level where we defined a mapping between coordinates of each goal (Metric, Result and Parameter) and the properties of the domain ontology. Coordinates of the goal are expressed using the properties of the ontology in order to allow the designer to choose the most relevant properties of each class that can be used to define the goal. The domain ontology represents the global model integrating the whole sources. Goals are used in this process, to inform us about the most relevant information to store in the DW model. Figure 17.a represents a fragment of the OWL ontology meta-model.

Fig. 19.c and Fig. 19.d presents an example of instantiation of the ontology and goal models using Goal1 requirement expressed on classes and properties of Rental car ontology.

5.2. The design process proposed

DW design typically includes the three usual design phases: conceptual, logical and physical phases. We first extract a DW ontology from the global domain ontology (GO), that is considered as the DW conceptual model that will be annotated by multidimensional concepts. A set of transformation rules is used to translate the conceptual model into a logical relational model.

5.2.1. Definition of the DW ontology Extracting the DW ontology

A DW ontology (DWO) is defined as a module of the Global Ontology (GO) by extracting concepts and properties used to express users goals. This DWO represents the DW conceptual model that will be stored in the DW structure for describing its data semantics. Three scenarios are possible: (1) $DWO = GO$: the GO corresponds exactly to users' goals and requirements, (2) $DWO \subset GO$: the DWO is extracted from the GO using OntoSub operator and covers all requirements, (3) $DWO \supset GO$: the GO does not fulfill the entire users' requirements. The designer extracts the fragment of the GO corresponding to requirements and can locally enrich it with new concepts and properties. Following the first formalization of domain ontologies, the DWO is formally defined as follows: $\mathcal{O} :< \mathcal{C}_{\mathcal{DW}}, \mathcal{P}_{\mathcal{DW}}, \mathcal{S}ub_{DW}, \mathcal{A}pplic_{DW} >$.

Once the DWO is defined, we exploit its reasoning capabilities to correct all inconsistencies, and to reason about the goals model.

Reasoning on the DW ontology

Two usual reasoning mechanisms are first used: checking the consistency of the ontology (classes and instances) and inferring subsumption relationships. This reasoning is supported by most existing reasoners, and allows the detection of design errors. Another reasoning mechanism is used in order to propagate influence relationships between goals. Influence relationships are used afterwards for exploring the multidimensional structure of the DW model, where we consider 'fact' concepts as central concepts and 'dimensions' as concepts *influencing* them. For propagating influence relationships, we defined propagating rules inspired from works on casual graphs (17) that use the same semantic of influ-

ence relationships. Two operations are used: *addition* corresponding to the idea of cumulative influences of numerous paths having the same target goal, and *multiplication* operation corresponding to the idea of transitivity of influences in a path. Those rules are applied on the graph of goals to find a sort of transitive closure of influence relationships.

Based on the goal model of figure Fig. 17.b, we defined the following propagating rules:

$Positive - influence(G_1, G_2) \wedge Negative - influence(G_2, G_3) \longrightarrow Negative - influence(G_1, G_3)$

$Negative - influence(G_1, G_2) \wedge Negative - influence(G_2, G_3) \longrightarrow Positive - influence(G_1, G_3)$

$Positive - influence(G_1, G_2) \wedge Positive - influence(G_2, G_3) \longrightarrow Positive - influence(G_1, G_3)$

$Negative - influence(G_1, G_2) \wedge Positive - influence(G_2, G_3) \longrightarrow Negative - influence(G_1, G_3)$

$Positive - influence(G_1, G_2) \wedge Negative - influence(G_1, G_3) \longrightarrow Undetermined - influence(G_1, G_2)$

Annotating the DW ontology

The multidimensional role of concepts and properties are then identified based on an analysis of defined goals. We consider for each goal, properties used to define the result (or its metric) as potential fact measures as it represents a result to analyze. Parameters are entities possibly influencing a goal result and are thus considered as potential dimension attributes. Parameters of goals influencing this given goal (existing and inferred) are also considered as potential dimensions. Let us take the following goals "Increase rental agreements for country C" and the goal "Reduce number of cars for all branches" influencing the first goal negatively. Parameters *Branch* (of the second goal) and *Country* (of the first goal) are both candidates to represent dimensions for the measure *Rental Agreements* (See figure 20 for this example). We validate the link between facts and their potential dimensions by looking for (1,n) relationships from the DW ontology. (1,n) relationships are identified as direct or transitive functional ObjectProperties in OWL ontologies (57). Algorithm 1 formalizes these steps.

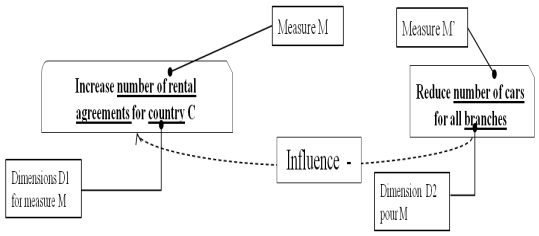


Figure 20. Influence relationships used for completing DW multidimensional structure

5.2.2. Definition of the DW logical model

The logical model of the $\mathcal{DW}$ is generated by translating the annotated DWO into a relational model. Several works in the literature proposed methods for translating ontologies described in a given formalism (PLIB, OWL, RDF) to a relational or object-relational representation (27). This translation can follow three possible relational representations (2): *vertical*, *binary* and *horizontal* (See Figure 21). Vertical representation is used for RDF ontologies, and stores data in a unique table of three columns (subject, predicate, object). In a binary representation, classes and properties are stored in tables of different structures. Horizontal representation translates each class as a table having a column for each property of the class. We proposed in (23) a set of translation rules for representing PLIB and OWL ontology (classes, properties and restrictions) in a relational schema following the binary and horizontal representations.

In our case study scenario (ontology model and requirements), the DWO was equal to the GO. The multidimensional algorithm identified two facts: (i) *Rental-Agreement* having the following dimensions *Rental-duration*, *Car-Model*, *Country*, *customer*, and (ii) *Branch* fact having *Customer* and *Country* as dimensions. Figure 22 presents the relational star schema obtained after translating the DWO into a relational schema.

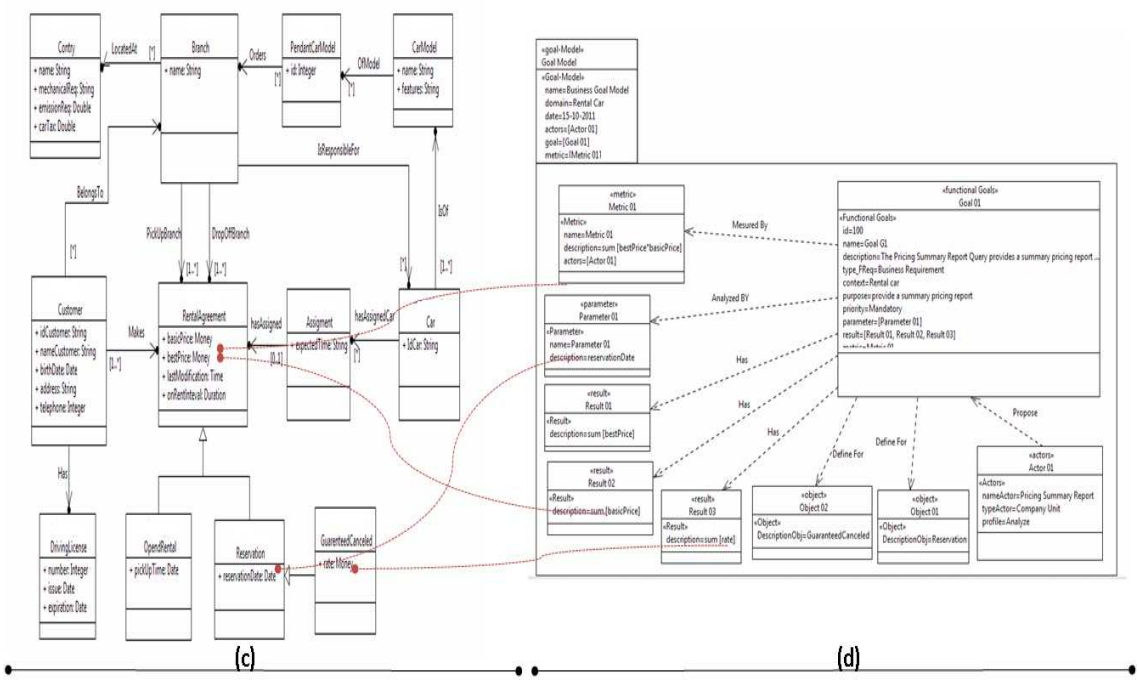


Figure 19. Instantiation of the requirements and ontological models

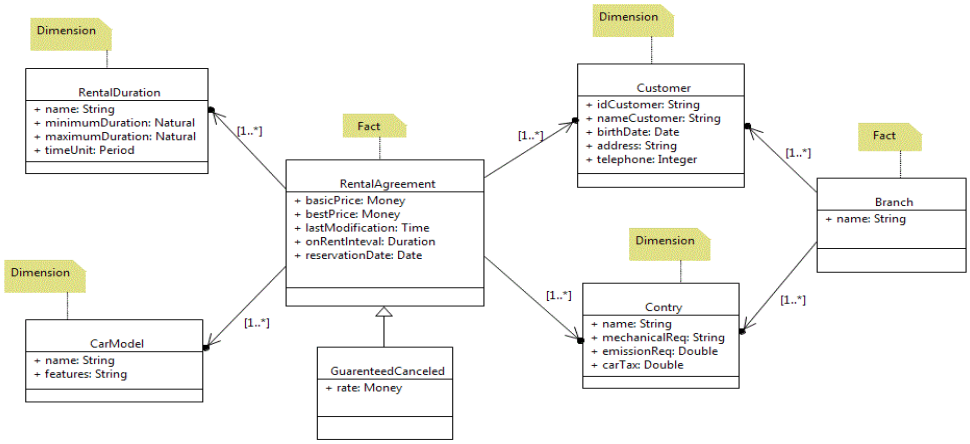


Figure 22. Rental car Star schema defined by the design process proposed.

begin
  for *Each goal G* **do**
    Each ontological property used as result or metric of G is a measure candidate;
    Each property used as a parameter of G is a dimension attribute candidate;
    Parameters of the goals influencing goal G are dimension attributes of the measure identified for G;
    Classes that are domains of properties that are measures candidates are facts candidates;
    Classes that are domains of dimension attributes are dimension candidates;
    if *fact class F is linked to a dimension by (1,n) relationship* **then**
      keep the two classes in the model;
      else
        R
      end
      eject the dimension class;
    end
    Hierarchies between dimensions are constructed by looking for (1,n) relationships between classes identified as dimensions (for each fact);
  end
end

Algorithm 1: Multidimensional annotations

5.3. Definition of the OBDW

To validate our proposal, we choose the OBDB *OntoDB* (22) introducing an additional part called the meta-schema (Figure 23). The DW ontology defined is loaded in OntoDB. The meta-schema is extended to include the requirements model (Fig. 17.**b**). This extension is done by defining OntoQL statements. OntoDB currently supports only the horizontal relational representation for the data part. The main characteristic of this solution is that it materializes all different models used along the life cycle of $\mathcal{DW}$ applications: the conceptual model (represented by the ontology), the logical model that may represented following a binary or horizontal relational representation according to the choice of the designer, the meta-schema extended by the requirements model we defined for storing requirements persistently in the $\mathcal{DW}$, and the local ontology defining semantics of stored data and requirements.

Vertical

TRIPLES		
Sujet	Prédicat	Objet
Cust#1	Type	Customer
Cust#1	Name	Name1
Cust#1	Age	Age1
Cust#1	makes	Agreement1
Agreement#1	Type	Rental Agreement
…	…	…

Binary

Unique Table	
URI	CLASSID
Customer#1	Customer
Agreement#1	Rental Agreement
…	…

Name	
ID	VALUE
Cust#1	Name1
…	…

Age	
ID	VALUE
Cust#1	Age1
…	…

Horizontal

Customer			
ID	Name	Age	makes
Cust#1	Name1	Age1	Agreement#1
…	…		

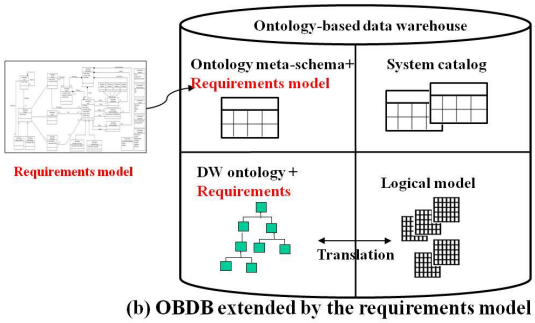
Figure 21. Rental car Star schema defined by the design process proposed.

6. Case tool implementing the proposed method

To validate our proposal, a case tool has been developed implementing the design steps described (Figure 24). This tool is designed to offer designers similar functionalities offered by classical database case tool tools such as: *PowerAMC*, *Rational Rose*, etc. We chose OWL formalism as an example to implement our design process. Our tool is a graphical system developed in Java Figure 25. Considered OBDBs reference an existing integrated ontology (GO) formalized in OWL. The GO integrating the selected OBDBs is identified by its URI, and its information is displayed

```
Create Entity #GoalModel(#Property (#Domain String, #Name
String, #Date Date), #collects REF(#Goal))

Create Entity #Object (#its_properties REF (#Property) Array)

Create Entity #Result(#its_properties REF (#Property))

Create Entity #Metric(#its_properties REF (#Property))

Create Entity #Parameter (#its_properties REF (#Property)
Array)

Create Entity #Metric(#its_properties REF (#Property) Array)

Create Entity #Goal (#Property (#Id Int, #Name String,
#Description String, #Context String, #Purpose String),
#defined_for REF (#Object), #has REF (#Result), #Measured_by
REF (#Metric), #analyze REF (#Parameter), #concern
REF(#Actor))

Create Entity Functional_Goal UNDER #Requirement

Create Entity NonFunctional_Goal UNDER #Requirement
```

(a) OntoQL statements



(b) OBDB extended by the requirements model

Figure 23. OBDW: Extended OBDB of type III by the requirements model.

on the same way as *Proteg editor*. Classes are presented as a hierarchy. The selection of a class displays its information (*Concept name, description, properties, super-classes and instances*). Access to all ontologies is made through the OWL API. Designers can either express users' requirements through an interface allowing them to select the different classes and properties used in requirements or by choosing a text file containing these requirements. Requirements expressed must follow the goal model we presented above. The tool checks the syntax of each requirement. The projection of requirements on the GO generates the DW ontology (DWO) that is a module of the GO. DWO is extracted using ProSé plug-in available within Protégé editor, ensuring the logical completeness of the extracted ontology (39). Fact++ reasoner is invoked to classify the DWO classes taxonomy and to check its consistency. Influence rules defined on goals are implemented using SWRL (Semantic Web Rule Language) language [9], which must be combined with Jess [10] inference engine to execute defined SWRL rules and apply them on the DWO. A parser analyzes requirements specified in order to identify the multidimensional role of classes and properties according to the annotation algorithm defined (Algorithm 1). The logical star schema is then defined after translating the DWO annotated. This multidimensional model is presented in two ways: (i) as a tree whose nodes are identified facts, their measures, their respective dimensions and dimensions attributes, (ii) graphically as a relational star schema that can be printed. The designer can modify this model graphically by adding and deleting classes and properties. A demonstration video of this tool is available at http://www.lisi.ensma.fr/members/bellatreche.

[9] http://www.w3.org/Submission/SWRL/
[10] http://www.jessrules.com/

7. Open issues on evolution in OBDW defined

The second issue when making heterogeneous systems interoperable is to keep and maintain this interoperability operational to different changes characterizing the complex environment these systems evolve in. This new direction is called sustainable interoperability. Increasingly research efforts are proposed in this direction for evolving interoperable systems sustainably (52; 7; 33; 25; 45). Managing the DW evolution implies maintaining and updating the warehouse so that it fulfils the organization requirements as well as possible.

Based on concepts seen in the background, we

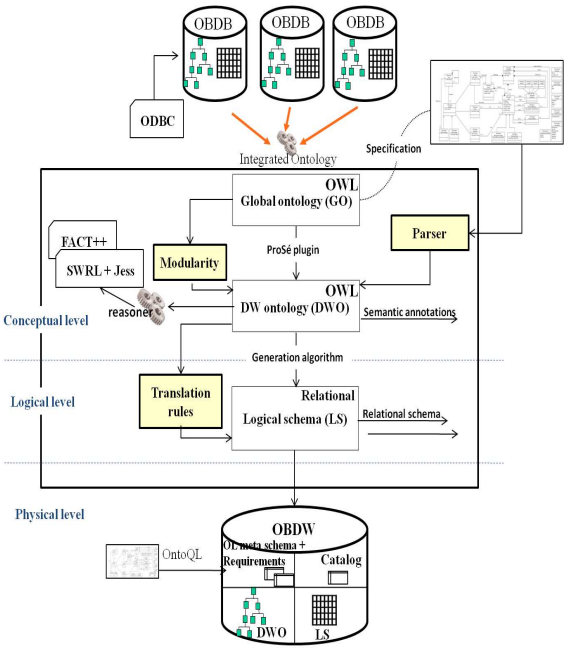


Figure 24. Architecture of the proposed case tool.

discuss in what follows some scenarios of DW evolutions and we show how our ontology-based data warehouse can adapt to different changes arising to *ontologies*, *sources* and *requirements*. These scenarios are as follows: (i) The changes occur in the ontology but does not affect the DW model, (ii) The changes occur in the ontology and affect the DW model, (iii) The changes occur in requirements, (iv) The changes occur in sources.

7.1. Ontology changes do not affect the DW model

The DW model we defined is designed based on domain ontology. An ontology, by definition is a shared and consensual conceptualization of a given domain. This is one of the advantages of considering the domain ontology schema as a first design level where most domain changes are already limited. One can make the reasonable assumption that the shared ontology evolves slowly. However, in some few cases the ontology can evolve. (53) identified three causes for an ontology evolution: (i) Changes in the domain (domain evolution due to changes in the real world), (ii) Changes in conceptualization (changes in usage perspective), (iii) Changes in the explicit specification (occur when an ontology is translated to another knowledge-representation language). These changes can interest decision makers or not. If not, the DW model remains the same. The second case where the changes interest decision makers is discussed in the following point.

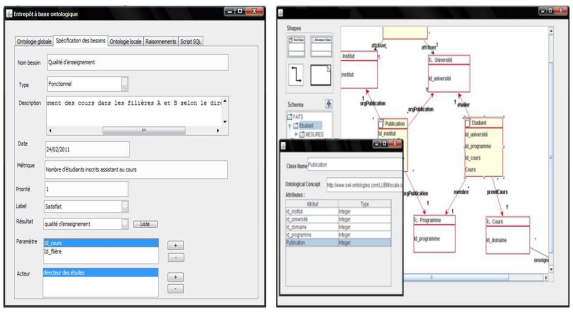


Figure 25. Interfaces from the case tool proposed.

7.2. Ontology changes affect the DW model

In this scenario, changes occurring in the ontology have to be impacted on the DW model. The exchange interoperability is based on the use of a shared ontology for which definitions have been defined at a given time. At that given time, the interpretation of concepts descriptions is effectively unambiguous. But, during an ontology lifetime, evolutions/corrections may be required and, in order to preserve the right interpretation of concepts at any time, a versioning management must be provided. In PLIB for example, some ontology change management rules have been defined. These rules are based on the ontology continuity principle (10). The idea is to keep track of evolutions by ensuring in a single ontology (the more recent one) that whatever have been its evolutions, the interpretation of any concept descriptions that was conforming to that ontology at a given instant can be performed. As a consequence, it means for instance that a con-

cept cannot be removed, but must be annotated " *deprecated*". This evolution management distinguishes ontology evolution that corresponds to revision changes (no impact in the concept description interpretation), version change (the old ontology cannot interpret all the evolutions provided by the new ontology) and error correction (the concept shall be deprecated).

7.3. Requirements change

Handling requirements evolution means maintaining the changes that occur to the decisional requirements and studying their impact on the DW model. Indeed, new requirements may arise and others may become inappropriate. These changes can either just be updated the DW model, or can affect the whole DW structure (for example by rising new facts and dimensions). The requirements model in our approach is stored in the DW structure extending the DW conceptual model. The impact of each requirement change on DW schema and data can thus be identified and can be managed more accurately. As stated in (3), when requirements and their connection with semantics sources are represented using formal expressions and stored on dedicated knowledge bases, this facilitates a sustainable interoperability through the creation of systems capable to reason and deduce mapping readjustments according to requirements changes.

The impact of requirements changes on other requirements is also an important dimension. A change made to one requirement may affect several other requirements making them to change as well. Neglecting these dependencies can increase the cost of implementing a requirement, and in turn cause budget or schedule problems (5)

7.4. Sources change

Changes in a DW sources, like in the traditional databases, have two categories: (i) Content changes (insertion, update, deletion of data and records), (ii) Schema changes (addition, modification and dropping data structures and their attributes) (52). In order to tackle the problem of schema changes, two different ways are possible: schema evolution and schema versioning. The first approach consists in updating a schema and transforming data from an old schema into a new one, and only the current version of a schema is present. In contrast, the second approach keeps track of the history of all versions of a schema (52). We proposed in (52) a framework for managing the versioning model of a DW constructed form ontology-based sources. The approach deals with asynchronous evolution of local ontologies of sources and the shared one. An important advantage in our approach is the link between the different models. Changes can thus be identified and managed on the conceptual level (the local ontologies), and can be propagated to the other models along the design cycle (logical and physical models).

8. Conclusion

Systems interoperability is one of the most important issues that needs to be solved for enterprises that have to deal with different partners collaborating beyond their own frontiers. In order to cooperate and to exchange information in an efficient way, enterprises data need to be integrated into a unified view. Ontologies have been proved to be effective solutions for the developing integration solutions allowing enterprise sources to share the same vocabulary. The presence of ontologies during the development of integration systems facilitates the management of data and requirements ambiguity.

In parallel to this situation, ontologies have been extensively adopted by companies. As consequence, mountains of ontological data are produced. To offer solutions for storing, managing, querying these data, ontology based database (OBDB) systems have been proposed by academic and industrial communities. As consequence, ontological instances become a gold mine for decision makers to get efficient knowledge to improve the profit of their enterprises. Data warehouse is the enterprise memory that materializes data from different sources. In this paper, we propose a new approach for designing ontology-based data warehouse from a set of OBDBs. Additionally to the nature of sources participating in the construction of our data warehouse, a second important component that our methodology consid-

ers is the user's requirements. We propose to extend the use of ontologies, on the same way as for data sources, to clarify (eliminate ambiguity) and unify users requirements. User's requirements are furthermore stored in the DW model. The presence of these requirements may contribute in reducing the complexity of other phases of the DW life cycle (optimization, personalization and evolution management). To do so, a requirement model following a goal-driven formalism is given and mechanism to ensure their persistency into the DW is developed. The final DW structure defined is validated within the OBDB OntoDB, extending its ontological meta-schema by the requirements model. A tool implementing the design method proposed is developed. A discussion about different evolution scenarios and their management within our proposed design framework is presented.

This work leads to many other tasks currently in progress including: (i) evaluating our method for large scale case study, (ii) a validation of different evolution management scenarios, (iii) study of the impact of requirements persistency in different phases of life cycle of DW development (query processing and physical design) and (iv) the extension of our design method (including user requirement persistency) for OLTP applications.

References

[1] IEEE 830. Recommended practice for software requirements specifications. 1998.

[2] D.J. Abadi, A. Marcus, S. Madden, and K. Hollenbach. Sw-store: a vertically partitioned dbms for semantic web data management. *VLDB Journal*, 18(2):385–406, 2009.

[3] C. Agostinho and R. Jardim-Gonçalves. Dynamic business networks: A headache for sustainable systems interoperability. In *OTM Workshops*, pages 194–204, 2009.

[4] S. Alexaki, Christophides V., G. Karvounarakis, D. Plexousakis, and K. Tolle. In proceedings of the second international workshop on the semantic web. In *SemWeb*, 2001.

[5] A. Aybuke and W. Claes. *Engineering and Managing Software Requirements*. Springer, 2005.

[6] V. Bazili, C. Gianluigi, and H. D. Rombach. The goal question metric approach. Technical report, Computer science Technical Report Series CS-TR-2956, 1992.

[7] B. Bebel, J. Eder, C. Koncilia, T. Morzy, and R. Wrembel. Creation and management of versions in multiversion data warehouse. In *In Proc. ACM SAC*, pages 717–723, 2004.

[8] L. Bellatreche, Y. Ait Ameur, and C. Chakroun. A design methodology of ontology based database applications. *Logic Journal of the IGPL, Oxford University Press*, 19(5):648–665, 2011.

[9] L. Bellatreche, D. Nguyen Xuan, G. Pierra, and H. Dehainsala. Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases. *Computers in Industry Journal Elsevier*, 57(8-9):711–724, 2006.

[10] L. Bellatreche, G. Pierra, and E. Sardet. *Evolution Management of Data Integration Systems by the Means of Ontological Continuity Principle*. Recent Trends in Information Reuse and Integration Book, edited by Springer, 2011.

[11] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, and S. Paraboschi. Designing data marts for data warehouses. *ACM Transactions on Software Engineering and Methodology*, 10(4):452–483, 2001.

[12] J. Broekstra, A. Kampman, and F. V. Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. *In International Semantic Web Conference*, pages 54–68, 2002.

[13] R. Bruckner, B. List, and J. Schiefer. developing requirements for data warehouse systems with use case. *Seventh Americas Conference on Information Systems*, 2001.

[14] M. J. Cafarella and A. Y. Halevy. Web data management. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1199–1200, 2011.

[15] D. Calvanese, G. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data integration in data warehousing. *Int. J. Cooperative Inf. Syst.*, 10(3):237–271, 2001.

[16] S. Castano, V. Antonellis, and S. D. C. Vimercati. Global viewing of heterogeneous data sources. *Transactions on Knowledge and Data Engineering*, 13(2):277–297, 2001.

[17] B. Chaib-draa. Causal maps: theory, implementation, and practical applications in multiagent environments. *IEEE TKDE*, 14(6):1201–1217, 2002.

[18] S. Chaudhuri and U. Dayal. n overview of data warehousing and olap technology. *A Sigmod Record*, 26(1):65–74, March 1997.

[19] D. Chen, G. Doumeingts, and F. Vernadat. Architectures for enterprise integration and interoper-

ability: Past, present and future. *Special issue on Enterprise Integration and Interoperability in Manufacturing Systems. Computers In Industry. Elsevier*, 59(5), 2008.

[20] P. Chen. The entity relationship model - towards a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[21] E. I. Das, G. Eadon Chong, and J. Srinivasan. Supporting ontology-based semantic matching in rdbms. *In VLDB*, pages 1054–1065, 2004.

[22] H. Dehainsala, G. Pierra, and L. Bellatreche. Ontodb: An ontology-based database for data intensive applications. *In DASFAA*, pages 497–508, April 2007.

[23] C. Fankam. Ontodb2 : un systme flexible et efficient de base de donnes base ontologique pour le web smantique et les donnes techniques. Ph.d. thesis, Poitiers University, 2009.

[24] C. Fankam, S. Jean, L. Bellatreche, and Y. Ait Ameur. Extending the ansi/sparc architecture database with explicit data semantics: An ontology-based approach. *In Second European Conference on Software Architecture (ECSA'08*, pages 318– 321, 2008.

[25] J. Ferreira, C. Agostinho, J. Sarraipa, and R. Jardim-Gonçalves. Monitoring morphisms to support sustainable interoperability of enterprise systems. In Robert Meersman, Tharam S. Dillon, and Pilar Herrero, editors, *OTM Workshops*, volume 7046 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 2011.

[26] E. Franconi, F. Baader, U. Sattler, and P. Vassiliadis. *Fundamentals of Data Warehousing, chapter Multidimensional Data Models and Aggregation*. Springer Verlag Berlin Heidelberg, 2000.

[27] A. Gali, C.X Chen, K. Claypool, and R. Uceda-Sosa. From ontology to relational databases. *ER Workshops*, pages 278–289, 2004.

[28] I. Gam and C. Salinesi. A requirement-driven approach for designing data warehouses. *REFSQ'2006*, 2006.

[29] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for olap discovery driven analysis. *International Journal of Data Warehousing and Mining IJDWM*, 2011.

[30] P. Giorgini, S. Rizzi, and M. Garzetti. Goal-oriented requirement analysis for data warehouse design. *DOLAP'05*, pages 47–56, 2005.

[31] M. Glinz. On non-functional requirements. *15th IEEE International Requirements Engineering Conference RE07*, pages 21–26, 2007.

[32] C.H. Goh, S. Bressan, E. Madnick, and M.D. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–293, 1999.

[33] M. Golfarelli, J. Lechtenborger, S. Rizzi, and G. Vossen. Schema versioning in data warehouses: enabling cross-version querying via schema augmentation. *Data and Knowledge Engineering*, 2006.

[34] M. Golfarelli and S. Rizzi. Methodological framework for data warehouse design. In *DOLAP*, pages 3–9. ACM, 1998.

[35] M. Golfarelli and S. Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*. McGraw Hill, 2009.

[36] T. R. Gruber. A translation approach to portable ontology specifications. *In Knowledge Acquisition*, 5:199–220, 1993.

[37] S. Jean, G. Pierra, and Y. Aït Ameur. Domain ontologies: A database-oriented analysis. In José A. Moinhos Cordeiro, Vitor Pedrosa, Bruno Encarnação, and Joaquim Filipe, editors, *WEBIST (1)*, pages 341–351. INSTICC Press, 2006.

[38] M. R. Jensen, T. Holmgren, and T.B. Pedersen. Discovering multidimensional structure in relational data. *6th International Conference on Data Warehousing and Knowledge Discovery, LNCS Springer*, 3181:138–148, 2004.

[39] E. Jiménez-Ruiz, B.C. Grau, U. Sattler, T. Schneider, and R. Berlanga Llavori. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Description Logics*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[40] Mazon J.N., J. Trujillo, M. A. Serrano, and M. Piattini. Improving the development of data warehouses by enriching dimension hierarchies with wordnet. *in: M. Collard (Ed.), ODBIS*, 4623 of Lecture Notes in Computer Science, Springer:85–101, 2006.

[41] S. Khouri and L. Bellatreche. A methodology and tool for conceptual designing a data warehouse from ontology-based sources. *DOLAP10*, pages 19–24, 2010.

[42] S. Khouri and L. Bellatreche. Dwobs: Data warehouse design from ontology-based sources. In *DASFAA*, pages 438–441, 2011.

[43] M. A. King. A realistic data warehouse project: An integration of microsoft acces and microsoft excel advanced features and skills. *Journal of Information Technology Education Innovations in Practice.*, 8, 2009.

[44] A. Kiryakov, D. Ognyanov, and D. Manov. Owlim

- a pragmatic semantic repository for owl. *WISE Workshops*, pages 182–192, 2005.

[45] H. Kondylakis and D. Plexousakis. Exelixis: evolving ontology-based data integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1283–1286, 2011.

[46] H.G. Lenz and A. Shoshani. Summarizability in olap and statistical data bases. In *SSDBM*, pages 132–143, 1997.

[47] B. List, J. Schiefer, and A. M. Tjoa. Process-oriented requirement analysis supporting the data warehouse design process a use case driven approach. *11th International Conference on DEXA*, pages 593–603, 2000.

[48] E. Malinowski and E. Zimányi. Hierarchies in a multidimensional model: From conceptual modeling to logical representation. *Data Knowl. Eng.*, 59(2):348–377, 2006.

[49] J. Mazon, J. Pardillo, E. Soler, O. Glorio, and J. Trujillo. Applying the i* framework to the development of data warehouses. *In The 3rd International i* Workshop - istar08*, pages 79–82, 2008.

[50] B. McBride. Jena: Implementing the rdf model and syntax specification. *Technical report Hewlett Packard Laboratories*, 2001.

[51] V. Nebot, R. Berlanga, J. M. Perez, M. J. Aramburu, and T. B. Pedersen. Multidimensional integrated ontologies: A framework for designing semantic data warehouses. *Journal on Data Semantics JoDS*, 13:1–36, 2009.

[52] D. Nguyen Xuan, L. Bellatreche, and G. Pierra. A versioning management model for ontology-based data warehouses. In *DaWaK*, pages 195–206, 2006.

[53] Klein M.C.A. Noy N.F. Ontology evolution: Not the same as schema evolution. *Knowl. Inf. Syst.*, 6(4):428–440, 2004.

[54] G. Pierra. Context-explication in conceptual ontologies: Plib ontologies and their use for industrial data. *Journal of Advanced Manufacturing Systems, World Scientific Publishing Company*, 2006.

[55] G. Pierra. Context representation in domain ontologies and its use for semantic integration of data. *Journal of Data Semantics (JoDS)*, 10:174–211, 2008.

[56] C. Rolland. Reasoning with goals to engineer requirements. *Enterprise Information Systems Springer*, pages 1–8, 2005.

[57] O. Romero, D. Calvanese, A. Abello, and M. Rodriguez-Muro. Discovering functional dependencies for multidimensional design. *DOLAP09*, pages 1–8, 2009.

[58] O. Romero, A. Simitsis, and A. Abelló. Gem: Requirement-driven generation of etl and multidimensional conceptual designs. In *DaWaK*, pages 80–95, 2011.

[59] J. Schiefer, B. List, and R.M. Bruckner. A holistic approach for managing requirements of data warehouse systems. *8th Americas Conference on Information Systems*, 2002.

[60] D. Skoutas and A. Simitsis. Designing etl processes using semantic web technologies. *In Proc. of the ACM 9th International Workshop on Data Warehousing and OLAP*, pages 67–74, 2006.

[61] I.Y. Song, R. Khare, and B. Dai. Samstar: A semi-automated lexical method for generating star schemas from an er diagram. *DOLAP07*, pages 9–16, 2007.

[62] N. Stolba. Towards a sustainable data warehouse approach for evidence-based heathcare. Ph.d. thesis, Vienna University of Technology, 2007.

[63] V. Sugumaran and V. C. Storey. The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM TODS*, 31(3):10641094, 2006.

[64] P. Vassiliadis, M. Bouzeghoub, and C. Quix. Towards quality-oriented data warehouse usage and evolution. *Inf. Syst.*, 25(2):89–115, 2000.

[65] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information - a survey of existing approaches. *Proceedings of the International Workshop on Ontologies and Information Sharing*, pages 108–117, August 2001.

[66] P. Bao J. Wang Y., Haase. A survey of formalisms for modular ontologies. *In: IJCAI 2007. Workshop SWeCKa.*, January 2007.

[67] R. Winter and B. Strauch. A method for demand driven information requirements analysis in data warehousing projects. *36th HICSS*, page 231, 2003.