Designing ETL Processes Using Semantic Web Technologies

Dimitrios Skoutas
Nat'l Techn. Univ. of Athens
School of Electr. and Comp. Eng
Athens, Greece
Tel: +30-210-772-1436
dskoutas@dblab.ece.ntua.gr

Alkis Simitsis
Nat'l Techn. Univ. of Athens
School of Electr. and Comp. Eng
Athens, Greece
Tel: +30-210-772-1402
asimi@dblab.ece.ntua.gr

ABSTRACT
One of the most important tasks performed in the early stages of a data warehouse project is the analysis of the structure and content of the existing data sources and their intentional mapping to a common data model. Establishing the appropriate mappings between the attributes of the data sources and the attributes of the data warehouse tables is critical in specifying the required transformations in an ETL workflow. The selected data model should be suitable for facilitating the redefinition and revision efforts, typically occurring during the early phases of a data warehouse project, and serve as the means of communication between the involved parties. In this paper, we argue that ontologies constitute a very suitable model for this purpose and show how the usage of ontologies can enable a high degree of automation regarding the construction of an ETL design.

Categories and Subject Descriptors
H.2.1 [**Database Management**]: Logical design - data models, schema and subschema.

General Terms
Algorithms, Design.

Keywords
Data warehousing, ETL, conceptual modeling, transformations, ontologies, reasoning, semantic web technology.

1. INTRODUCTION
It has been extensively argued in the literature [12, 28] that one of the most important parts during the design and deployment phase of a data warehouse project is the design of the flow of data from the source relations towards the target data warehouse relations. For the construction of such flow, specialized tools are already available under the general name of ETL tools. Extraction-Transformation-Loading (ETL) tools are pieces of software responsible for the extraction of data from several sources, their cleansing, customization and insertion into a data warehouse. Up to now, there exist several commercial ETL tools [2,10,11,16,18], while the field has extensively been studied by research too [13,25,28]. All the aforementioned efforts focus on the representation and formal description of the ETL transformations required, while, it is assumed that the task of the identification of the necessary ETL transformations is manually performed by the designer or the administrator of the data warehouse.

DOLAP'06, November 10, 2006, Arlington, Virginia, USA.

However, this is not a straightforward procedure. The design of an ETL process is driven by the semantics of the data sources and the constraints and requirements of the data warehouse application. This information is typically available in natural language format in the form of application specifications and documentation, comments embedded in the sources' schemata, or even, it has to be recorded after oral communication with the different parties involved in the project.

In this paper, we address the issue of using Semantic Web technologies to facilitate the process of selecting the relevant information from the available data sources and appropriately transforming it to populate the data warehouse. In particular, we present an ontology-based approach that facilitates the construction of an ETL workflow. The main idea underlying our approach is the use of ontologies to formally and explicitly specify the semantics of the data source schemata, as well as the data warehouse schema. We show that having this formal and explicit description of the domain, it is possible to automate in a large degree the process of the ETL workflow creation.

The decision of using an ontology-based approach, instead of using another technology for example a UML-based approach, lies in the fact that ontologies provide an elegant way to perform reasoning that is required for the automatic determination of ETL transformations as we will present later. An ontology is most commonly defined as "a formal, explicit specification of a shared conceptualization". "Explicit" means that the type of concepts used, and the constraints on their use are explicitly defined. "Formal" refers to the fact that the ontology should be machine readable [6]. An ontology describes the knowledge in a domain in terms of classes and properties, as well as relationships between them. Thus, it may be considered as an appropriate solution to confront with the main challenge in the back stage of the data warehouse, namely *heterogeneity.*

In general, integrating data from heterogeneous sources faces two main problems: *structural heterogeneity* and *semantic heterogeneity*. Structural heterogeneity refers to the fact that different information systems store their data in different structures, thus there is a need for homogenization. For instance, information that is stored in one attribute/relation in a schema may be stored in more than one attributes/relation in another schema (what is called 1:n and n:1 matching in the schema matching bibliogra-

phy). Semantic heterogeneity considers the intended meaning of the information items. In order to achieve semantic interoperability in a heterogeneous information system, the meaning of the information that is interchanged has to be understood across the systems. In [7] three main causes for semantic heterogeneity are identified: (a) "confounding conflicts", which occur when information items seem to have the same meaning, but differ in reality; e.g. owing to different temporal contexts; (b) "scaling conflicts", which occur when different reference systems are used to measure a value; e.g. different currencies or different date formats; and (c) "naming conflicts", which occur when naming schemes of information differ significantly (a frequent phenomenon is the presence of homonyms and synonyms.).

Contributions. The proposed approach deals with the problem of heterogeneity and facilitates the construction of the ETL workflow in a conceptual level. Specifically, the contributions of this work are as follows.

1. Construction of the application vocabulary. We describe the creation of a common vocabulary that deals with different naming schemes.
2. Annotation of the data stores. We provide a method for the annotation of the data sources and data warehouse w.r.t. the application vocabulary.
3. Generation of the application ontology. We introduce an algorithm for the generation of an ontology that contains information about the appropriate inter-attribute mappings and the conceptual transformations required.
4. Generation of conceptual ETL design. Finally, we provide a method to automatically derive the mappings from the source attributes to the attributes belonging to the data warehouse, along with the appropriate ETL transformations.

Outline. The rest of the paper is structured as follows. Section 2 discusses the state of the art. Section 3 presents a method for the construction of an ontology based on the schemata of the data warehouse and of the data sources. Section 4 introduces a method for the automatic derivation of the transformations required in an ETL process, based on the constructed ontology. Finally, Section 5 concludes the paper with a prospect to the future.

2. RELATED WORK

In this section, we review related work in the fields of conceptual modelling for ETL processes and of ontology-based information integration.

Conceptual models for ETL. Although, there exists several approaches [12,13,25,28] for the conceptual part of the design of an ETL scenario, so far, we are not aware of any other research approach that concretely deals with the automatic derivation of ETL transformations and inter-attribute mappings in the early stages of a data warehouse project.

Conceptual models for DW's. On the other hand, there are some approaches concerning the (semi-) automation of several tasks of logical DW design from conceptual models, but they do not provide a formal method to specifically determine the flow of data from the source recordsets towards the data warehouse. A couple of approaches concern the development of dimensional models from traditional ER-models [4,17]. In other approaches the data warehouse logical schema is generated from a conceptual schema. [20] presents a framework for generating a DW logical schema from a conceptual schema. [5] is an approach to derive initial data warehouse structures from the conceptual schemes of operational sources. [8] proposes a general methodological framework for data warehouse design, based on Dimensional Fact Model. [9] presents a modelling framework, BabelFish, concerning the automatic generation of OLAP schemata from conceptual graphical models and discusses the issues of this automatic generation process for both the OLAP database schema and the front-end configuration. [21] proposes algorithms for the automatic design of DW conceptual schemata. Finally, an approach for DW development based on the Model Driven Architecture is presented in [14].

Ontology-based information integration. A survey of existing approaches is presented in [26], where three general directions are identified: single ontology approaches [1], multiple ontologies approaches [15] and hybrid approaches [7]. A single, "global", ontology simplifies the integration process but is difficult to create and maintain, especially in the presence of changes in the data store schemata. Multiple ontologies provides flexibility; however comparing the sources becomes considerably more difficult. In hybrid approaches each source is described by its own ontology, using terms from a global, shared vocabulary. The approach followed in this work is hybrid, because a common vocabulary is provided, containing the primitive terms of the domain, and the data stores are described independently by a set of classes defined by means of these common terms.

3. ONTOLOGY CONSTRUCTION

3.1 Preliminaries

In this section we discuss the construction of an ontology to model the domain of discourse, as outlined by the schemata of the data warehouse and the data sources and in accordance to the provided application specifications. This ontology will be used to resolve the semantic conflicts and identify the operations required to achieve the information integration.

The construction of the ontology is based on a common vocabulary that the designer provides, as well as provided annotations of the involved data sources (also provided by the designer). This type of information is derived from the application description and requirements. There exist several ways to succeed in this: apart from interpreting the model that represents the schemata of the data stores, e.g. E/R model, it is also possible to extract information from accompanying documents/specifications by exploiting research results from the areas of natural language processing or deriving ontologies from natural language descriptions, as well as schema matching; still, this issue is orthogonal to the approach described here.

This manual work required during the ontology creation phase is not an additional overhead that our approach imposes to the designer. The process of integrating the information from the available data sources and populating the data warehouse through the construction of an appropriate ETL workflow requires that the designer first clarifies and resolves the semantics of the involved sources, either this is done by means of a naming convention [23] and a reference vocabulary, or a global schema, a UML model, and so on. Following an ontology-based approach presents some significant benefits over these alternatives, such as:

1. Explicit and formal representation, with well-defined semantics, allowing automated reasoning to be performed.
2. Better support for reuse and evolution.
3. Better support for visual representation and documentation.

In this work, we focus on the first point. Having well-defined semantics makes it possible to leverage on existing reasoning techniques in order to automate several parts of the process. This is not possible when using for example a UML model, because UML has no formally defined semantics, therefore no automated reasoning can be performed.

3.2 The Web Ontology Language

In our approach, we have chosen the Web Ontology Language (OWL) [24] as the representation language for the created ontologies. There are two main reasons behind this choice. First, OWL, and in particular OWL DL, is based on Description Logics [7], a decidable fragment of First Order Logic, constituting the most important and commonly used knowledge representation formalism. This means that it provides a formal representation and the ability to leverage on existing reasoners for automating several tasks of the process, such as checking subsumption relationships between classes. Second, OWL is the proposed W3C standard for representing ontologies on the Web. Even though a data warehouse application is not primarily a Web application, it may often be desired to provide a Web interface for some of the data sources, or the data warehouse itself, either in the form of dynamic Web pages or through a Web service. For example, it may be desired or required, due to policy constraints or security considerations, to expose the data source via a Web service interface, so that access to data is restricted and controlled. In such cases, the established mappings between the database schema and the ontology could be reused for the task of annotating the dynamically generated Web pages or for providing a semantic description of the service (e.g. using OWL-S). Moreover, the presented approach may be reused in order to model ETL processes as sequences of Semantic Web services.

Our model does not require the complete features provided by the language. For example, OWL allows defining properties as being symmetrical, inverse, transitive, specifying arbitrary cardinality constraints, and so on. For our aim, we focus on the notions of class and property hierarchy, property domain and range restrictions, class equivalence and disjointness. This makes the resulting model simpler, both in terms of reasoning support, as well as from the aspect of the designer, since a not very detailed knowledge of the language is required. Especially, all the required features are available in the OWL DL part of the language, i.e. no additional features from OWL Full are required (e.g. meta-modeling), which ensures that the reasoning process is decidable. Table 2 summarizes some of the main features of OWL, which are the ones used in this work.

3.3 A Reference Example

In the rest of this paper, we suppose a simple scenario comprising two data sources DS1 and DS2, and a data warehouse DW. The schemata are shown in Table 1 (underlined attributes denote primary keys, while attributes in italic denote foreign keys).The provided application description and requirements are as follows.

Table 1. The data stores of the reference example

Data Stores	Schemata
DS1	products(id,name,amount,price,guarantee,type,*sid*) stores(sid,name,location)
DS2	software(id,name,*sid*,quantity,price) hardware(id,name,*sid*,quantity,price) stores(sid,name,city,street,number)
DW	products(id,name,*sid*,quantity,price) stores(sid,name,city,street)

The data sources contain information regarding products and stores; each product is stored in one store. For each product, the available amount/quantity, as well as the price, are provided. The information regarding the amount of the product is not available for all products contained in DS1. DS1 contains also information regarding how many years guarantee are provided for the product.

There are two distinct types of products: software and hardware. In DS1 the distinction is made by the attribute "type", while in DS2 two separate relations are used. Prices in DS1 and DS2 are recorded in Euros and Dollars, respectively. Products in DS1 have a check constraint forcing a minimum price of 200 Euros. Each store has a name and an address, comprising city, street and number. In DS1 this information is contained in the attribute "location". In DW, attribute "street" contains both the street and the number of the store. We also, assume that each store is located in one of the following cities: Paris, Rome or Athens. In the data warehouse, the relation products should contain only software products, with prices ranging from 500 to 1500 euros, known quantity, and which are located either in Rome or Athens. Finally, the id's in the data warehouse relations are surrogate keys that replace the original primary keys of the sources [12].

3.4 Annotating the Data Stores

In what follows, we assume that each data store is described by a relational schema. This assumption is made for simplicity reasons and does not compromise the generality of our approach. Non-relational stores may be handled by providing suitable wrappers. For example, regarding web sources, in a previous work [19] a wrapper for extracting pieces of information from Web pages has been presented. The objects extracted by such wrapper may then be further filtered and given an internal structure according to the specified relational schema.

We present a model for semantically annotating each data store, based on the provided information. We use knowledge representation techniques to formally and explicitly describe the semantics of each store. A common vocabulary, specified by the designer, is used according to the available descriptions about the data stores and the requirements of the application. Once an ontology has been created to describe the available data stores, it is further used to automatically identify the appropriate transformations for the integration of the available information.

Application vocabulary. The designer provides a set of related terms constituting the shared vocabulary of the application. This common vocabulary is used to annotate the data stores and to construct the application ontology, and has the following form:

$$V = (V_C, V_P, V_F, V_T, f_P, f_F, f_T),$$

Table 2. Summary of OWL features used in our approach

Notation	Name	Description
A, B, C, D	Class	Defines a group of individuals sharing some properties. We use classes to represent the relations contained in the data sources, as well as custom data types.
$A \sqsubseteq B$	subClassOf	Used to create class hierarchies, by stating that a class A is a subclass of (or is subsumed by) another class B. Organizing classes in a hierarchy is used to resolve the semantic conflicts and identify the transformations required to integrate information from different data sources.
P	Property	Used to relate an instance of a class to an instance of another class (ObjectProperty) or to an instance of a datatype (DatatypeProperty). Properties are mainly used in our model for representing attributes.
dom(P)	domain	Specifies the class(-es) to which the individuals the property applies to, belong.
rang(P)	range	Specifies the class(-es) to which the individuals being the values of the property, belong.
$\forall P.C$	allValuesFrom	Used to restrict the range of a property, when this property is applied to individuals of a specific class.
$=nP$	cardinality	Specifies the cardinality of a property in respect to a specific class. We use this feature to denote whether an attribute is present in a relation or not and whether null values are allowed.
$C \sqcap D = \varnothing$	disjointWith	States that two classes A and B are disjoint, i.e. an individual may not belong to both A and B. This feature is used to prevent data records from one relation being integrated with data records from another relation.
allDiff[x,y,...]	allDifferent	States that two or more individuals are different from each other. This statement is necessary because the "unique names assumption" does not hold in OWL, i.e. two individuals x and y are not assumed to be necessarily different unless explicitly stated so.
$C \equiv D$	equivalent	Used to state that two classes are equivalent, i.e. each instance of the one class is also an instance of the other.
$C \sqcup D$	unionOf	Denotes the union of two classes.
$C \sqcap D$	intersectionOf	Denotes the intersection of two classes.
$C = \{a_1, \ldots, a_n\}$	oneOf	Used to define an enumerated class, i.e. a class that contains exactly the instances specified in the definition.
$\exists P.\{a\}$	hasValue	Restricts the property to having a specific value.

where:

- V_C is a set of terms denoting the primitive concepts of the domain of discourse.
- V_P is a set of terms denoting the features that characterize each of the above concepts. A feature may describe an aspect of the concept or may relate the concept with another concept.
- V_F is a set of terms denoting the different representation formats that may be used for a feature.
- V_T is a set of terms denoting the allowed values that an enumerated feature may take. If the feature has different representation formats, then each format has its own set of values.
- $f_P : V_P \rightarrow V_C$ is a function associating each feature to the primitive concept it describes.
- $f_F : V_F \rightarrow V_P$ is a function associating each representation format to a feature.
- $f_T : V_T \rightarrow V_F \cup V_P$ is a function associating each value to a representation format, or directly to a feature.

From the above vocabulary it is straightforward to derive the following information.

The set of features associated to a given concept $c \in V_C$:

$$V_{Pc} = \{p : f_P(p) = c\}$$

The set of representation formats for a given feature $p \in V_P$:

$$V_{Fp} = \{\varphi : f_F(\varphi) = p\}$$

Additionally, for enumerated features:

The set of values for a given representation format $\varphi \in V_F$:

$$V_{T\varphi} = \{t : f_T(t) = \varphi\}$$

The set of values for a given feature $p \in V_p$:

$$V_{Tp} = \{t : f_T(t) = p\}$$

Data store annotation. Each data store DS contains a set of relations R_{DS}, each one comprising one or more attributes A_R. The process of annotating a data store refers to providing two types of information: (a) establishing the appropriate mappings between the data store relations and attributes and the concepts and features of the vocabulary; and (b) describing each relation in terms of the cardinality, representation format and (range of) values of its associated features. The mappings are specified by means of a function $f_{RM} : R_{DS} \rightarrow V_C$, associating each relation to a primitive concept, and a relationship $M_{AM} \subseteq A_{DS} \times V_P$ between attributes and features. For each relation R, the following information is provided for each feature p of the concept c to which R is mapped:

$$I_{R,p} = (\varphi, \min, \max, T, n, R', \Gamma_f, \Gamma_\alpha)$$

where:

- $\varphi \in V_{Fp}$ is the representation format used for this feature in this relation (in the case of a feature having different representation formats).
- min and max denote the minimum and maximum value for this feature. This information is required when different value ranges are used for this feature in the data stores, meaning that comparisons are needed to determine if a given value falls within a specified range.
- $T \subseteq V_{T\varphi} \cup V_{Tp}$ is the value(s) that this feature has on the given relation (in the case of an enumerated feature).
- $n \in \{0,1,\text{null}\}$ denotes the cardinality of this feature in the given relation. n = 0 means that no information is provided in this relation for this feature, while n = 1 means that all the data records contained in this relation have a (not null) value for this feature. It is important to note that this constraint should be specified only for properties for which no null values are allowed in the data warehouse. This is because the information integration process in a data warehouse application is asymmetric: a data source can not be integrated

if necessary information is missing; however, it is not a problem if the data source contains additional information that is not required in the data warehouse.

- R′ is the relation referenced by the associated attribute (in the case that this attribute is a foreign key).
- Γ_f, Γ_P are provided in the case that the annotated property has values resulting from an aggregation operation. Then, Γ_f denotes the aggregation function (e.g. sum, max, avg) and Γ_P denotes the property(-ies) on which the aggregation is based, i.e. the one(s) that would appear on the "group by" part of the corresponding SQL statement.

Example (cont'd). We revisit the reference example to determine the application vocabulary:

V_C = {product, store}
V_{Fpid} = {source_pid, dw_pid}
V_{Pstore} = {sid, sName, city, street}
V_{Fsid} = {source_sid, dw_sid}
V_{Fprice} = {dollars, euros}
V_{Ttype} = {software, hardware}
$V_{Pproduct}$ = {pid, pName, quantity, price, type, storage}
V_{Tcity} = {paris, rome, athens}

Note: we have not included a property "guarantee", since this information is not required in the data warehouse.

We then use this vocabulary to annotate the data stores. The annotation for DS1 is illustrated in Figures 1 and 2. For the interest of space, the annotation of the rest data stores is omitted.

3.5 Ontology Generation

Given the application vocabulary and the annotations, the next step is to construct the application ontology, which comprises: (a) a set of primitive classes corresponding to the specified concepts, representation formats and ranges or sets of values; (b) a set of properties corresponding to the specified features of the concepts of the domain; and (c) a set of defined classes representing the data stores. The algorithm is outlined below.

1. A class is created for each primitive concept $c \in V_C$.
2. A class is created for each different representation format $\varphi \in V_F$.
3. An enumerated class is created for each enumerated (format of a) feature, having as instances the specified values. If these values are mutually exclusive, this fact is asserted by means of an "allDifferentFrom" statement.
4. A class is created for each range of values specified in the provided annotations. These classes are organized in a hierarchy according to the represented value ranges. That is, if classes C_1 and C_2 represent the range of values $[r_1, r_2]$ and $[r_3, r_4]$, respectively, and $r_1 \geq r_3$, $r_2 \leq r_4$, then $C_1 \sqsubseteq C_2$.
5. An object property P is created for each feature $p \in V_P$. The domain of P is set to be the class corresponding to the primitive concept $c = f_P(p)$.
6. A functional object property "hasValue" is created. Also, two functional datatype properties "hasAggregateFunction" and "hasAggregateAttributes" are created.
7. For each relation R a defined class is created. The class definition has the form: $C_R \equiv C \sqcap c_1 \sqcap \ldots \sqcap c_n$, where C is the class representing the primitive concept to which R is mapped, and each c_i is derived from the annotation of R. In particular, given each tuple $I_{R,p} = (\varphi, \text{min}, \text{max}, T, n, R')$ describing R, the following conditions apply:
   a. if $\varphi \neq$ null, and F is the class corresponding to the representation format φ, then the constraint $\forall$P.F is added.
   b. if min ≠ null or max ≠ null, and R is the class representing this range of values, then the constraint $\forall$P.R is added.
   c. if T ≠ null, the constraint $\forall P.(\exists \text{hasValue}.\{v_1\} \sqcup \ldots \sqcup \exists \text{hasValue}.\{v_n\})$ is added, where $v_1, \ldots, v_n$ are the values specified by T.
   d. if n = 0 or n = 1, the cardinality constraint =nP is added.
   e. if R′ ≠ null, the constraint $\forall$P.D is added, where D is the defined class corresponding to the relation R′.
   f. if $\{\Gamma_f, \Gamma_P\} \neq$ null, a value restriction of the form $\forall P.(\exists \text{hasAggregateFunction}.\{f\} \sqcap \exists \text{hasAggregateAttributes}.\{P\})$ is added, where f is the aggregate function denoted by Γ_f and P the property(-ies) denoted by Γ_P.

After the ontology is constructed, a reasoner is used to infer subsumption relationships, classifying the defined classes accordingly.

Example (cont'd). We revisit the reference example to construct the ontology. The following sets of classes and properties result from the provided vocabulary and annotations:

S_C = {Product, Store, Dollars_Price, Euros_Price, Source_Pid, DW_Pid, Source_Sid, DW_Sid, Type_Values, City_Values, Above_200, From_500_To_1500}.

S_P = {pid, pName, quantity, price, type, storage, sid, sName, city, street, hasValue, hasAggregateFunction, hasAggregateAttributes}.

The following classes are then defined, based on the above classes and properties, in accordance to the provided annotations of the respective data stores: DS1_Products, DS1_Stores, DS2_Software, DS2_Hardware, DS2_Stores, DW_Products and DW_Stores. For instance, the definition of class DS1_Products is shown in Figures 3 and 4. The rest of the definitions are omitted due to space limitations. Finally, the reasoner is used to infer the class hierarchy depicted in Figure 5.

4. ETL DESIGN GENERATION

Having constructed an ontology that describes the application domain and the mappings between this ontology and the source schemata, we describe how the appropriate ETL transformations for integrating data from the data sources and loading it to the data warehouse, may be automatically derived.

Table 3 presents a set of operators that are typically encountered in an ETL process. This set contains, among others, filter (σ), project (π), join (J), aggregation (γ), function (convert), that are core operators in practically every frequently used ETL transformation. However, our work does not anticipate the formal determination of the functionality of each transformation, rather we aim at the identification of a conceptual transformation, whose functionality will be determined later by the designer through a template library similar to the one proposed in [27].

The first step of our method is to determine from which sources, and more specifically, from which attributes/relations of these sources, information needs to be extracted in order to populate each attribute/relation in the data warehouse.

	DS1	
products → product	id → pid name → pName amount → quantity	price → price type → type sid → storage
stores → store	sid → sid name → sName	location → city location → street

Figure 1. The mappings for the data store DS1

DS1		φ	min	max	T	n	R´	Γ_f	Γ_a
products	I_{pid}	source_pid	-	-	-	1	-	-	-
	I_{pName}	-	-	-	-	1	-	-	-
	$I_{quantity}$	-	-	-	-	-	-	-	-
	I_{price}	euros	200	-	-	1	-	-	-
	I_{type}	-	-	-	{software, hardware}	1	-	-	-
	$I_{storage}$	-	-	-	-	1	store	-	-
stores	I_{sid}	source_sid	-	-	-	1	-	-	-
	I_{sName}	-	-	-	-	1	-	-	-
	I_{city}	-	-	-	{paris, rome, athens}	1	-	-	-
	I_{street}	-	-	-	-	1	-	-	-

Figure 2. Annotation for the data store DS1

```
<owl:Class rdf:ID="DS1_Products">
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Product"/>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#pid"/>
   <owl:cardinality rdf:datatype="&xsd;
          nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
...
  <owl:Restriction>
   <owl:onProperty rdf:resource="#price"/>
   <owl:allValuesFrom>
    <owl:Class>
     <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Euros_Price"/>
      <owl:Class rdf:about="#Above_200"/>
     </owl:intersectionOf>
    </owl:Class>
   </owl:allValuesFrom>
  </owl:Restriction>
...
 </owl:intersectionOf>
</owl:Class>
```

Figure 3. Code snippet for DS1_Products

OWL-Class: DS1_Products

Intersection of:
(= 1 price)
Product
(∀storage . DS1_Stores)
(= 1 type)
(= 1 pName)
(∀type . ((∃hasValue . {software}) ∪
(∃hasValue . {hardware})))
(∀pid . Source_Pid)
(= 1 pid)
(= 1 storage)
(∀price . (Euros_Price ⊓ Above_200))

Figure 4. Definition for DS1_Products

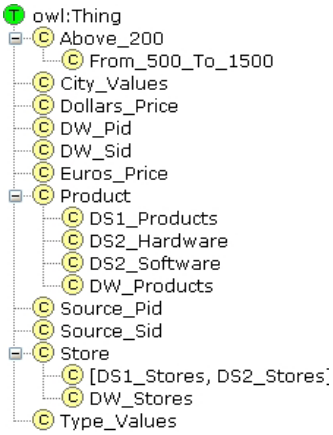

Figure 5. The class hierarchy

Suppose that R_T is a relation contained in the data warehouse, represented in the ontology by the defined class C_T. Then a relation R, contained in a data source and represented in the ontology by the defined class C, is used as a provider relation in the ETL process for R_T, iff the following conditions hold:

a) C and C_T are subclasses of the same primitive class, where by primitive class we refer to classes in the ontology representing the primitive concepts of the domain, i.e. the terms specified in V_C.

b) C and C_T are not disjoint, i.e. $C \sqcap C_T \neq \emptyset$.

The first condition ensures that the integrated data records have the same semantics. The second condition prevents the integration of information from sources having constraints that contradict the constraints imposed by the data warehouse.

The next step is to determine the transformations required to integrate data from the source relations to the target relation. The required transformations are determined by the specified mappings and by the relative position of the source and target classes in the class hierarchy. The following algorithm identifies the ETL transformations for transferring data from a source relation R_S, corresponding to the defined class C_S, to the target relation R_T, corresponding to class C_T.

Phase 1.

1. A "project" $\pi(\alpha_1, \ldots, \alpha_n)$ is inserted, where for each attribute α_i of R_S exists $p \in V_P$ such that $(\alpha_i,p) \in M_{AM}$. That is, this transformation excludes source attributes not mapped (via the vocabulary) to an attribute of the target relation.
2. A "concatenate" $c(\alpha_1, \ldots, \alpha_n, P)$ is inserted if for each i $(\alpha_i,p) \in M_{AM}$, i.e. if the source attributes α_i are mapped to the same property P. Similarly, a "split" operation $s(\alpha, P_1, \ldots, P_n)$ is inserted for each source attribute α that is mapped to more than one property.
3. If a tuple $I_{Rs,p} = (\varphi, \text{min}, \text{max}, T, n, R')$ has $R' \neq \text{null}$, then the steps 1 and 2 are performed for R´ and a "join" is inserted. For the rest of the process the attributes of R and R´ are treated similarly; i.e. the transformations required for R´ are also identified and included in the ETL process.

Phase 2. In this phase a set of "select", "convert" and "not null" operations are added to the ETL workflow, depending on the subsumption relationship between the two classes C_S and C_T, which is determined by the reasoner. Specifically, the following distinct cases exist:

1. Classes C_S and C_T are equivalent: $C_S \equiv C_T$. In this case, the data records in the source relation satisfy exactly the same constraints as the ones imposed by the target relation; thus no transformations are required.
2. Class C_S is subsumed by class C_T: $C_S \sqsubset C_T$. In this case, the data records in the source relation satisfy all the constraints imposed by the target relation, plus some additional ones; thus, again, no transformations are required.
3. Class C_S subsumes class C_T: $C_S \sqsupset C_T$. In this case, the data records in the source relation satisfy only a subset of the constraints imposed by the target relation, meaning that a filtering is required to exclude those data records not fulfilling the additional constraints. For each of these additional constraints c_i, we distinguish:
   a. c_i is a format constraint: $\forall P.F$, where F is a class corre-

sponding to a representation format. This means that no representation format is specified for the source values, which in turn means that no values exist for this property. Therefore, no transformation is performed at this point. (In Phase 3 "add" transformations are inserted to appropriately fill missing values.)

b. c_i is a range constraint: $\forall P.R$, where R is a class corresponding to a value range. A "select" is added: $\sigma(P, R)$.

c. c_i is a value constraint: $\forall P.(\exists hasValue.\{v_1\} \sqcup \ldots \sqcup \exists hasValue.\{v_n\})$. A "select" transformation is added: $\sigma(P, R)$, where R here is the set of values $v_1, \ldots, v_n$.

d. c_i is a value constraint regarding an aggregated property: $\forall P.(\exists hasAggregateFunction.\{f\} \sqcap \exists hasAggregateAttributes.\{P_\gamma\})$. In this case an "aggregate" is added: $\gamma(f, P, P_\gamma)$. Note that if two or more aggregate transformations with the same set of properties P_γ exist, they should be merged in a single aggregate transformation.

e. c_i is a cardinality constraint: =1P. This occurs when source relation allows null values while the target relation does not. Thus, a "not null" is added to exclude data records having null values for this property: nn(P).

4. Classes C_S and C_T are overlapping (neither one is subsumed by the other). Again, we are interested in the constraints of the target relation that do not apply to the source relation. Such constraints can be derived by defining a temporary class with constraints the conjunction of all the constraints of C_S and C_T. This class is a subclass of C_S, so, as before, the additional constraints are identified. Each of these constraints is handled as in the previous case, apart from the format constraints. If a format constraint $\forall P.F$ occurs, the source class includes a corresponding constraint $\forall P.F'$, where F and F′ denote different representation formats. Thus, a "convert" transformation is inserted: f(P, F′, F).

Phase 3.

1. For each additional attribute α contained in the target relation, an "add" transformation is inserted: add(α, v), where v is either a default value specified in the annotation of the source relation (specifically contained in the set T of the corresponding tuple $I_{R,p}$) or null.
2. A "union" is inserted to aggregate the (transformed) data records coming from this source relation with (transformed) data records from the other source relations.
3. A "distribute" is inserted to accordingly store the incoming data records as specified by the provided mappings.
4. A "detect duplicates" is inserted for each attribute having a "unique" constraint in the target relation.

Example (cont'd). We conclude the reference example by presenting the resulting conceptual ETL transformations.

The class corresponding to the target relation DW.products is DW_Products, which is subsumed by the primitive class Product. Therefore, the relations to be considered are DS1.products, DS2.software and DS2.hardware. DS2.hardware is excluded, because, as inferred by the reasoner: DW_Products ⊓ DS2_Hardware ≠ Ø. Similarly, DS1.stores and DS2.stores are the source relations for DW.stores.

First, the required operations for the source DS1 are identified. According to Phase 1 and the corresponding mappings, a "join" is required between DS1.products and DS1.stores, preceded by a transformation π(id,name,amount,price,type,sid) on DS1.products, to exclude attribute "guarantee" not mapped to any property, and a transformation s(location,city,street) on DS1.stores. To identify the operations resulting from Phase 2, given that none of the classes DW_Products and DS1_Products is subsumed by the other, a temporary class DW_P_DS1_P is defined, having all the constraints of these two classes. As expected, this class is classified by the reasoner as a subclass of DS1_Products, which helps in identifying the following additional constraints:

– ∀storage.DW_Stores. The corresponding constraint in DS1_Products is ∀storage.DS1_Stores. Since DW_Stores and DS1_Stores are classes representing relations, the respective transformation needs to be inserted at this point.
– ∀pid.DW_Pid. The respective constraint in DS1_Products is ∀pid.Source_Pid. Classes DW_Pid and Source_Pid correspond to representation formats, therefore a "convert" operation f(pid, Source_Pid, DW_Pid) is inserted.
– ∀price.(Euros_Price⊓From_500_To_1500). The respective constraint in DS1_Products is ∀price.(Euros_Price⊓Above_200). Thus, a "select" is inserted to restrict the range of values for property price: σ(price, From_500_To_1500).
– =1quantity. Class DS1_Products does not specify any cardinality constraint on this property. Therefore, a "not null" is inserted: nn(quantity).
– ∀type.(∃hasValue.{hardware}). The respective constraint in DS1_Products is ∀type.((∃hasValue.{software})⊔(∃hasValue.{hardware})). Thus, a "select" is inserted to exclude the additional value "hardware" allowed in the data source but not in the data warehouse: σ(type, {software}).

Figure 6 illustrates the axioms causing the inference in question. Notice that the parts of the two classes' definitions that are stricken out by the reasoner, while performing this inference, indicate the above used constraints.

Finally, resulting from Phase 3, a "union" operation is inserted to integrate the transformed data records with data records from other source relations, and a "distribute" operation to appropriately store the tuples referring to products and stores.

As a concluding comment, in a following step, the transformations produced should be ordered so that their input schemata may be successfully populated; i.e. each transformation should be placed in the ETL design as long as all its providers already exist in the design. For this procedure, we adopt the approach presented in [22].

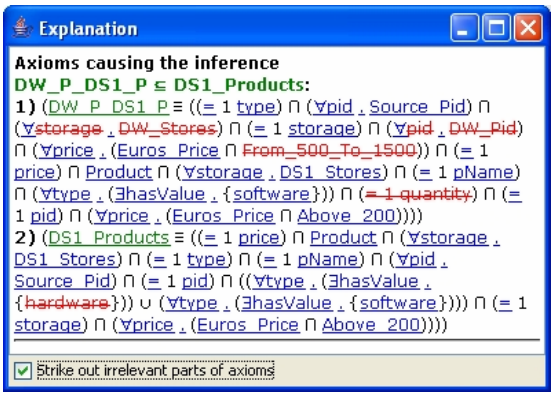

Figure 6. DS1_Products integration

Table 3. A set of common ETL workflow operations

Symbol	Name	Functionality
$\sigma(P, R)$	select	Filters the values of property P, excluding those values that do not belong to the set specified by R.
$f(P, F_1, F_2)$	convert	Converts the values of property P from the representation format F_1 to the representation format F_2.
$add(\alpha, v)$	add	Adds attribute α to the current schema, setting its values to v, where v is either a default value or null.
nn(p)	not null	Deletes the data records having a null value for property p.
dd(P)	detect duplicates	Detects, and appropriately removes, records having the same value for property P.
$\pi(\alpha_1, \ldots, \alpha_n)$	project	Projects the current schema preserving only the attributes denoted by $\alpha_1, \ldots, \alpha_n$.
$c(\alpha_1, \ldots, \alpha_n, P)$	concatenate	Concatenates the values of attributes $\alpha_1, \ldots, \alpha_n$ to set the value of property P.
$s(\alpha, P_1, \ldots, P_n)$	split	Splits the value of attribute α to set the values of properties $P_1, \ldots, P_n$.
$\gamma(f, P, P_\gamma)$	aggregate	Aggregates the values of property(-ies) P by grouping them by those of property(-ies) P_γ and applying the function(-s) f.
U	union	As the operator "union" in SQL.
J	join	As the operator "join" in SQL.
D	distribute	The "inverse" operation of join: if the data records contain attributes from multiple relations, as a result from a previous join operation, then this operation distributes data to the involved relations accordingly.

5. CONCLUSIONS

In this paper, we have focused on the problem of determining the inter-attribute mappings and identifying the ETL transformations required for the conceptual design of an ETL process. We have presented an approach based on Semantic Web technologies to facilitate the process of selecting the relevant information from the available data sources and appropriately transforming it to populate the data warehouse. We have used ontologies to formally and explicitly specify the semantics of the data stores' schemata. We have shown that having this formal and explicit description of the domain, it is possible to automate in a large degree the process of the ETL workflow creation.

Future plans include the study of the impact of changes to the ETL process using ontologies and the application of our method to real-world cases.

6. REFERENCES

1. Arens, Y., Hsu, C.-H., Knoblock, C. Query Processing in the Sims Information Mediator. Advanced Planning Tech., 1996.
2. Ascential Software Inc.url:http://www.ascentialsoftware.com
3. Baader, F., McGuiness, D. L., Nardi, D., Patel-Schneider, P. (Eds). Description Logic Handbook: Theory, implementation and applications. Cambridge University Press, 2002.
4. Ballard, C. Data Modeling Techniques for Data Warehousing. IBM Red Book, ISBN 0738402451, 1998.
5. Boehnlein, M., Ulbrich-vom Ende, A. Deriving the Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems. DOLAP, 1999.
6. Borst, W. N. Construction of Engineering Ontologies. PhD thesis, University of Twente, Enschede, 1997.
7. Goh, C.H.. Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources. MIT, 1997.
8. Golfarelli, M., Rizzi, S. Methodological Framework for Data Warehouse Design. DOLAP, 1998.
9. Hahn, K., Sapia, C., Blaschka, M. Automatically Generating OLAP Schemata from Conceptual Graphical Models. DOLAP, 2000.
10. IBM. IBM Data Warehouse Manager. url: http://www-3.ibm.com/software/data/db2/datawarehouse/
11. Informatica. PowerCenter. url:http://www.informatica.com/ products/data +integration/powercenter/default.htm
12. Kimball, R., et al. The Data Warehouse Lifecycle Toolkit. John Wiley & Sons, 1998.
13. Luján-Mora, S., Vassiliadis, P., Trujillo, J. Data Mapping Diagrams for Data Warehouse Design with UML. ER, 2004.
14. Mazon, J-N., Trujillo, J., Serrano, M., Piattini, M. Applying MDA to the development of data warehouses. DOLAP, 2005.
15. Mena, E., Kashyap, V., Sheth, A., Illarramendi, A. Observer: An Approach for Query Processing in Global Information Systems Based on Interoperability Between Pre-Existing Ontologies. CoopIS, 1996.
16. Microsoft. Data Transformation Services. url: www.microsoft.com
17. Moody, D.L., Kortink, M.A.R. From Enterprise Models to Dimensional Models: a Methodology for Data Warehouse and Data Mart Design. DMDW, 2000.
18. Oracle. Oracle Warehouse Builder Product Page. url: http://otn.oracle.com/products/warehouse/content.html
19. Papadakis, N., Skoutas, D. , Raftopoulos, K., Varvarigou, T. STAVIES: A System for Information Extraction from Unknown Web Data Sources through Automatic Web Wrapper Generation Using Clustering Techniques. IEEE TKDE 17(12), 2005.
20. Peralta, V. Data Warehouse Logical Design from Multidimensional Conceptual Schemas. CLEI, 2003.
21. Phipps, C., Davis, K. Automating Data Warehouse Conceptual Schema Design and Evaluation. DMDW, 2002.
22. Simitsis, A. Mapping Conceptual to Logical Models for ETL Processes. DOLAP, 2005.
23. Simitsis, A., Vassiliadis, P., Sellis, T. State-Space Optimization of ETL Workflows. IEEE TKDE 17(10), 2005.
24. Smith, M. K., Welty, C., McGuinness, D. L. OWL Web Ontology Language Guide. W3C Recommendation. 2004.
25. Trujillo, J., Lujan-Mora, S. A UML Based Approach for Modeling ETL Processes in Data Warehouses. ER, 2003.
26. Wache, H., et al. Ontology-Based Integration of Information A Survey of Existing Approaches. IJCAI workshop on Ontologies and Information Sharing, 2001.
27. Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., Skiadopoulos, S. A Generic and Customizable Framework for the Design of ETL Scenarios. Information Systems 30(7), 2005.
28. Vassiliadis, P., Simitsis, A., Skiadopoulos, S. Conceptual Modeling for ETL Processes. DOLAP, 2002.