An Ontology connected to several data repositories: query processing steps

Alfredo Goñi
Depto. LSI. Universidad del País Vasco
alfredo@si.ehu.es
http://siul02.si.ehu.es/~jirgbdat

Arantza Illarramendi
Depto. LSI. Universidad del País Vasco
jipileca@si.ehu.es

Eduardo Mena
Depto. IIS. Universidad de Zaragoza.
mena@prometeo.cps.unizar.es

José Miguel Blanco
Depto. LSI. Universidad del País Vasco
jipblarj@si.ehu.es

Abstract

The great expansion of communication networks has made available to users a huge number of heterogeneous and autonomous data repositories that present different structures/organizations, query languages and data semantics. In that context it is clear that new information retrieval techniques with a strategy that focuses on information content and semantics are needed. We propose to use domain specific Ontologies to capture the information content of such repositories whenever available. We describe such Ontologies using a system based on Description Logics. In this paper we present all the stages of the processing of a query formulated over an Ontology when the answer must be found in the underlying data repositories. Those stages make up a subpart of the global query processing strategy defined for a set of loosely-coupled Ontologies. We show first how the query is transformed into a semantically equivalent one and how inconsistent queries are detected. Then, we explain the test to verify if the query can be answered from the cache memory. Next, we present a set of heuristics used during the query decomposition process. Later on, we show how to optimize plans associated to subqueries that access the underlying data repositories and finally we illustrate how the answers retrieved from the repositories are correlated in order to generate the query answer.

Keywords: Ontologies, Knowledge-based systems, Query Processing.

1 Introduction

On the global information infrastructure, the great expansion of communication networks has made available to users a huge number of heterogeneous and autonomous data repositories. However, these repositories present different structures/organizations, query languages and data semantics, making it very difficult for the users to access the data stored in them.

A possible solution to lighten the problem of lack of uniformity when dealing with available repositories consists of defining new information retrieval techniques with a strategy that focuses on information content and semantics. We propose to represent intensional descriptions of the objects in the repositories as metadata, introducing in this way a semantic level over the repositories. In the existing syntactic-keyword navegational approaches, in which the query is a set of keywords, the user has to do most of the information filtering and correlation [Alt, Inf].

In order to define a semantic level over a repository different techniques can be used. We use one or more pre-existing Ontologies, characterizing information in different domains, to create the semantic level. An Ontology may be defined as the specification of a representational vocabulary for a shared domain of discourse which may include definitions of concepts, relations, functions and other objects [Gru93]. Ontologies have been used to describe information content in repositories independent of the underlying syntactic representation of the data [KS96, MP97].

In our proposal, Ontologies are described using a system based on Description Logics (DL) [BBMR89] and the mapping information[1] is expressed using the extended relational algebra. Reasoning mechanisms from DL are useful to perform query optimization and in particular semantic and caching optimization. DL systems are also appropriated to offer intensional answers to the users. Mapping descriptions play a key role in encapsulating the heterogeneity due to different formats and organization of the data in the various repositories. They act as an intermediary language between the DL expressions and the query languages of the local repositories. Our focus in this paper is to present the query processing steps defined for searching efficiently the data stored in one or more data repositories that may contain overlapping data under one relevant Ontology. With respect to other related works that also consider the problem of accessing efficiently underlying data repositories from semantically rich views [ASD+91, AKS96, CHS91, FRV96, LSK95], our particular contribution consists of incorporating new enhancements in the different query processing steps. So, we show: 1) the definition and use of the notion of *Most Immediate Superconcepts* in the step of semantic transfor-

[1] Mapping information is the information that relates Ontologies with one or more repositories where the actual data are stored.

mation and decomposition. This notion allows one to obtain some terms that do not appear in the query but that can be used to obtain a better query execution plan, as well as to eliminate constraints and redundant terms; 2) a test to decide if the initial query can be answered with the data stored in the cache memory; 3) a set of heuristics that make an efficient query decomposition; 4) how to optimize the mapping information corresponding to the subqueries previously decomposed and that have to be processed in the underlying repositories.

In the rest of the paper we present first the overview of the query processing. In the following sections we focus on the different steps of the query processing: semantic transformation, cache optimization, query decomposition, optimization of the mapping information and correlation of the answer.

2 Query Processing: an overview

Five main steps are followed during the query processing: parsing of the query; semantic transformation, cache optimization and query decomposition; query processing in the underlying repositories; loading of the answers brought from the repositories in the cache memory; and, query processing over the cache memory.

During the parsing step, lexical and syntactical errors are detected. In our case this is achieved by using the parser of the DL system. In the following sections we explain the features of the second and third steps. Finally we do not explain the last two steps because there are not new enhacements in them.

2.1 Query expression

The general form of a query formulated over an Ontology described using a DL system is:

[rf(role) for] getall concept_description

where *rf(role)* means to project the role values for that *concept_description*. The conditions that form part of the *concept_description* are a conjunction of concept names and role restrictions. A *concept*[2] groups individual elements of the real world. A *role* represents a binary relationship between concepts or between concepts and scalar data types. However what distinguishes the notion of concept from the class specification in semantic data models or object-oriented databases, is that it is possible to describe concepts using intensional descriptions phrased not only in terms of necessary properties that must be satisfied by their instances (in this case the concept is called a *primitive concept* and is denoted as $:<$ like using notation of the BACK system [vLNPS87]) but also in terms of necessary and sufficient properties (in this case the concept is called a *defined concept* and is denoted as $:=$) [Bor92]. The role restrictions can be cardinality restrictions like *atleast(n,role)* or *atmost(m,role)* or value restrictions like *all(role,concept_type), role: value* and *role: close(value)*[3]

Queries in DL systems are considered as new concepts that describe the conditions that the instances that constitute their answers must satisfy. Queries are *classified* in the Ontology. The *classification* mechanism consists of discovering the *subsumption* relationships between concepts when a new concept is declared, i.e. the new concept is automatically located into the hierarchy of terms. One concept *subsumes* another one if in all possible circumstances, any instance of the second one must be in the first one. Using a DL system, it is possible to know whether one concept *is subsumed* by another one simply by looking at the definition of the concepts, without accessing the instances.

2.2 Example query

To illustrate the query processing approach, we show the different steps performed when answering the next query:

rf(number-of-pages) for getall document and periodical_publication and multimedia_document and atleast(1,doc-author-name) atmost(1,doc-author-name) and all(doc-author-name,organization)

that retrieves the page numbers for all the documents that verify the following conditions: they are periodical, multimedia, with only one author and that the author name corresponds to an organization. For example, *<25,JavaWorld1>* could be an element of the answer where *JavaWorld1* is the instance that refers to a number of the JavaWorld on-line publication, published in http://www.javaworld.com and considering that the organization IDG is the author.

The previous query is formulated over the STANFORD-I Ontology (http://siul02.si.ehu.es/~jirgbdat/OBSERVER) and that is a subset of the Bibliographic-Data Ontology [Gru94] developed as a part of the ARPA Knowledge Sharing Effort.

3 Semantic Transformation

The semantic transformation consists of finding a semantically equivalent query to the user query. For that, the set of *Most Immediate Superconcepts* ($\mathcal{MIS}$) is calculated. Let $\mathcal{T} = \{T_1,\ldots,T_P\}$ be the set of concepts and roles that form the Ontology, and let $\mathcal{C} = \{C_1,\ldots,C_N\}$ be the set of concepts and role restrictions that form the concept description of the query then the set $\mathcal{MIS}$[4] corresponding to the query is:

$$\mathcal{MIS} = \{D | (D \in \mathcal{T} \vee D \in \mathcal{C}) \wedge (D \text{ subsumes } (C_1 \text{ and..and } C_N)) \wedge \forall E (E \in \mathcal{MIS} \wedge E \neq D \rightarrow E \text{ does not subsume } D)\}$$

The concept description of the query $\mathcal{C}$, C_1 *and* ... *and* C_N, is semantically equivalent to the intersection of all the immediate superconcepts in the set $\mathcal{MIS}$ D_1 *and* ... *and* D_M. In [GIMB97] the proof of the previous statement

[2] In some DL systems, the names *class* and *attribute* are used instead of *concept* and *role*.

[3] The restriction *role: close(value)* means that the *role* must take that *value* and only that.

[4] In order to get the $\mathcal{MIS}$ set the system uses the subsumption notion.

appears and also it is reasoned that the complexity of the operation to calculate the $\mathcal{MIS}$ set is polynomial.

3.1 Application to the example

The $\mathcal{MIS}$ set corresponding to the query presented in section 2.2 is:

$$\{\textit{multimedia\_document, atleast(1,doc-author-name), atmost(1,doc-author-name), magazine}\}$$

By calculating the $\mathcal{MIS}$ set some more specific concepts to be used in the next steps and that do not appear in the initial query are detected (in the example *magazine*) and other redundant concepts and role restrictions of the initial query are identified (in the example *document, all(doc-author-name,organization)* because they do not belong to the $\mathcal{MIS}$ set).

The semantically equivalent query is therefore

rf(number-of-pages) for getall multimedia_document and atleast(1,doc-author-name) and atmost(1,doc-author-name) and magazine

3.2 Detection of inconsistent queries and generation of intensional answers

Using the $\mathcal{MIS}$ set, the initial query may be detected as inconsistent and also intensional answers may be given to the user.

It is detected that the query is inconsistent when *getall nothing* is a semantically equivalent query to the user query, that is, when *nothing* is in the set of Most Immediate Superconcepts.

For example, if the query were

getall multiple-author-document and atmost(1,doc-author-name)

then it would be detected as inconsistent due to *multiple-author-document* is defined as

multiple-author-document := *document and atleast(2,doc-author-name)*

and therefore *nothing* would be in the $\mathcal{MIS}$ set of the previous query.

When working with DL systems, it is also possible to give intensional answers, that is, answers in terms of the descriptions that the instances that form the extensional answer satisfy. These intensional answers can be offered to the user before the extensional answer is ready. Two different types of intensional answers are possible: *Most Specific Formulation* (MSF) of the query and *Extended Formulation* (EF) of the query. The first one is formed by the elements of the $\mathcal{MIS}$ set corresponding to the query (notice that the MSF expression is the semantically equivalent query) and the second one is formed by recursively substituting the concepts in the initial query by their definitions. Both types of intensional answers (MSF and EF) are offered to the user by request but, in particular, the EF is also offered when the initial query is inconsistent, because it becomes more explicit where the inconsistency is.

The EF of the previous inconsistent query, *getall multiple-author-document and atmost(1,doc-author-name)*, appears below, where the inconsistency can be identified.

getall document and
atleast(2,doc-author-name) and atmost(1,doc-author-name)

4 Cache optimization

When dealing with Ontologies connected to several data repositories, it is worth having some data cached within the Ontologies in order to avoid accessing the underlying repositories each time a user formulates a query. Then, during the query processing, it is necessary to detect if the query can be answered with the data stored in the cache. In general, it is not easy to verify if the answer to a query is contained in the cache memory because it depends on the query language and how the cached data are represented. Furthermore, verification that the query is not cached should be as fast as possible.

In this section we state first the notion of explicit and implicit cache, then the test to decide if the query answer is in the cache memory and finally we point out the problem of finding the contents of the optimal cache memory and give a possible optimal cache memory for the example.

4.1 Explicit and implicit cache memory

Working with DL systems it is possible to cache *explicitly* the instances of concept descriptions and the values of roles. Taking into account that queries are considered as new concepts (possibly with projection of roles) that means that query answers can be *explicitly* cached. Moreover, there are other queries built using as base sets only sets in the explicit cache, that are *implicitly* cached.

A concept C that belongs to the Ontology, $C \in \mathcal{T}$, is explicitly cached by storing the instances in the cache memory and adding *getall C* to the set $\mathcal{ECQ}$.

A role $r \in \mathcal{T}$ is explicitly cached for a concept $C \in \mathcal{T}$ by storing their corresponding values in the cache memory and adding *rf(r) for getall C* to the set $\mathcal{ECQ}$. A role can be explicitly cached only for concepts that are explicitly cached.

The set $\mathcal{ECQ}$ is formed by all the concept descriptions and roles that are cached.

A query $q \equiv$ *[rf(r) for] getall* C is implicitly cached, denoted by $q \in \mathcal{ICQ}$, if all the terms (concepts and roles) that form part of the semantically equivalent query to q are cached (either implicitly or explicitly).

4.2 How cached data can be identified

The problem of verifying if a query can be answered completely with the contents of the cache memory is similar to the problem of verifying if a query is implicitly cached by the set of explicitly cached queries. The only difference

is that the concept expression of the query can be anyone (*q* ≡ *[rf(r) for] getall* C_1 *and . . . and* C_N instead of *q* ≡ *[rf(r) for] getall* C). The test to decide if the query answer is in the cache memory is the next one:

A query *q* ≡ *[rf(r) for] getall* C_1 *and . . . and* C_N is cached if all the concepts whose names (D_i) appear in the set $\mathcal{MIS} = \{D_1,\ldots,D_M\}$ corresponding to *q* are explicitly or implicitly cached (*getall* $D_i \in \mathcal{EC}\,\mathcal{Q} \cup \mathcal{IC}\,\mathcal{Q}$) and all the roles that appear in the set $\mathcal{MIS}$ and the roles to be projected in the query (r_j) are also cached (*rf*(r_j) *for getall* $D \in \mathcal{EC}\,\mathcal{Q} \cup \mathcal{IC}\,\mathcal{Q}$ where D subsumes C_1 *and . . . and* C_N)

The complexity of the previous test is polynomial because it is based on verifying if all the elements in the set $\mathcal{MIS}$ are cached, that is, if they are in the previously computed $\mathcal{EC}\,\mathcal{Q}$ and $\mathcal{IC}\,\mathcal{Q}$ sets.

Another novelty of our approach is that it does not matter how the user formulates the query

getall document and atleast(2,doc-author-name)
getall atleast(2,doc-author-name) and document
getall biblio_thing and atleast(2,doc-author-name)
getall multiple_author_document

because the $\mathcal{MIS}$ set {*multiple_author_document*} is calculated first and the verification is made based on this set using the $\mathcal{EC}\,\mathcal{Q}$ and the $\mathcal{IC}\,\mathcal{Q}$ notions.

4.3 Contents of the optimal cache memory for the example.

A different problem not directly related to the query processing steps but that needs a solution is to define the contents of the optimal cache memory. In [GIMB97] we present an approach to define an optimal cache, the cost model and the algorithm used to decide which queries are worth caching. The algorithm has to calculate the set $\mathcal{EC}\,\mathcal{Q}$ that produces the greatest benefit for a limited size of the cache memory. In order to know the benefit produced by a set $\mathcal{EC}\,\mathcal{Q}$ then its corresponding set $\mathcal{IC}\,\mathcal{Q}$ has to be calculated and a set of parameters that measure the benefit have to be available. We use an algorithm based on the A*, instead of an exhaustive search algorithm, that calculates quasi-optimal solutions and that our experimental results show that is more efficient. Moreover, the algorithm is not performed during query processing but between sessions (at night for example). Within a session the LRU (Least Recently Used) strategy is used as the strategy for replacement in the cache memory.

The algorithm proposed is:

1) create the initial state
2) while exists a state S not visited yet
2.1) build the new states resulting from adding a new query to the $\mathcal{EC}\,\mathcal{Q}$ and calculate the new $\mathcal{IC}\,\mathcal{Q}$, benefit B and cost C
2.2) order the list of states by decreasing order of benefit B
2.3) eliminate the states $(\mathcal{EC}\,\mathcal{Q}1, \mathcal{IC}\,\mathcal{Q}1, B1, C1)$ where there exists another state $(\mathcal{EC}\,\mathcal{Q}2, \mathcal{IC}\,\mathcal{Q}2, B2, C2)$ with cost ($C2<C1$) and benefit ($B2>B1$)

Finally, in order to follow with the rest of the query processing steps, let us suppose that the content of the cache memory, as calculated by the previous algorithm, is:

{*multimedia_document, agent, organization, publisher, university, [agent_name, agent]*[5]*, [doc_title, multimedia_document]*}

It can be seen that the example query cannot be completely answered with the contents of the cache memory because its corresponding semantically equivalent query expression contains terms (*magazine*, *doc-author-name* and *number-of-pages*) that are not in the cache memory.

5 Query Decomposition

Query decomposition implies to obtain and analyze all the possible combinations of subqueries that can be made on the underlying repositories in order to get the query answer. In this section we present the heuristics defined in order to perform an efficient query decomposition and their application to the example.

5.1 Heuristics

In general, query decomposition is a very complex task because there can be many different ways of decomposing a query into sets of subqueries and statistic information and access paths in the local systems are not available. For that reason, instead of calculating all the possible partitions[6], that is sets of subqueries, that can be made starting from the semantically equivalent query *[rf(r) for] getall* D_1 *and . . . and* D_M and estimating the cost for each partition, we have defined a strategy that avoids searching all the possibilities by applying a set of heuristics. The strategy consists of, first of all, obtaining the terms of the query that are not cached to which the heuristics H1, H2, H3 and H4 (defined below) are tried to apply. After this step it is known the set of terms that *necessarily have to be* in at least one of the decomposed subqueries to answer from the data repositories. Next, those non-cached terms are grouped into subqueries that have to be answered from *the same repository* to which the heuristics H5, H6 and H7 are tried to apply. H5 and H6 try to reduce the size of the subqueries and H7 tries to reduce the computation cost. The general goals of these heuristics are:

- Goal 1: To avoid that the answer sent from each repository is too large[7].
- Goal 2: To try that the computation cost in each repository is small, unless it is needed to reach goal 1.

[5] It means that the role *agent_name* is cached for the concept *agent*.

[6] If the query is composed by n terms, there are as much possible decompositions as the number of possible partitions in a set of cardinality n. Moreover, the possibilities are even more because a term can appear in more than one subquery and a defined concept can be substituted by its definition.

[7] In the following, terms *large* and *small* are relative to the limited size of the cache memory where the answers are stored during each session.

- Goal 3: To try not to bring parts that are already cached, unless it is needed to reach the previous goals.

- Goal 4: To try to send subqueries that satisfy the two first goals to be executed in parallel in different repositories. This avoids communication cost and a greater computation cost among the repositories.

Heuristic H1: *Substitute a non-cached defined concept without alternative*[8] *mapping information by its most specific definition.* In general, it is not worth bringing the instances corresponding to a defined concept from the repositories because there may happen that an already cached concept or a redundant element is brought from the repositories. Once the defined concept is substituted by its most specific definition the redundant elements in that definition are removed.

Heuristic H2: *Maintain a non-cached defined concept with alternative mapping.* The non-cached defined concept is not substituted by its most specific definition because it mainly favors goal 2 (alternative mapping informations are provided for defined concepts only if they are better than the automatically calculated from their definitions).

Heuristic H3: *Substitute a non-cached primitive concept by one of its non-cached subconcepts only if the rest of the subconcepts are all cached and if it is the total generalization of all its subconcepts.* A primitive concept that is not cached and that is the total generalization of several subconcepts where only one is not cached can be substituted by the only non-cached subconcept. This favors the goals 1, 2 and 3.

Heuristic H4: *Not to send a subquery with projection of a role that is already cached.* If the role *r* to be projected in the query is cached for the query concept, then it is not needed to retrieve the role values from the underlying repositories but only the instances of the query description. Once answers to the subqueries are loaded into the cache memory then it is possible to retrieve the role values corresponding to the instances that satisfy the query description. This heuristic favors the goal 1 because the answer sent from the underlying repositories is smaller (role values are not sent). It also favors the goal 3, because some parts already cached are not brought again: the role values.

Heuristic H5: *Reduce the size of the answer for a subquery within a repository.* If the size of the answer for a subquery sent to a repository is too great, specially if the description is *unsafe*[9], then that size has to be reduced by adding some term to the subquery that can be answered in the same repository and that reduces the size because the intersection of all the terms is calculated. If the terms $\{E_1,...,E_p\}$ have to be brought from the same repository and it is considered that the size of the answer is too great then another term E_q belonging to the initial query or that subsumes the query has to be added to the subquery. E_q has to verify that it does not subsume E_1 *and ... and* E_p because E_q would not reduce the size of the answer. It is convenient that E_q has the smallest possible size in order to reduce the size of the answer to E_1 *and ... and* E_p *and* E_q, and that it does not have a high cost to compute E_q in the repositories. If statistics are not available then it can be supposed that primitive concepts, restrictions of the type *role: value*, and defined concepts with alternative mapping do not have a high computation cost, but restrictions of the type *atleast, atmost* and *all* have a high cost. The best terms to be added are: 1) a non-cached term of the query with an alternative mapping in the same repository and 2) a cached term of the query that can be retrieved from the same repository. If there are not terms that verify these conditions then the heuristic H6 has to be applied.

Heuristic H6: *Merge subqueries whose sizes of answer have to be reduced with subqueries with smallest sizes of answer from other repositories.* If the size of the answer to a subquery in a repository is too great and the heuristic H5 cannot be applied any longer, then the merge with other subqueries in other repositories has to be made, in order to reduce that size. With this heuristic the goal 4 is favored.

Notice that this heuristic is applied instead of calculating the best set of subqueries to be merged which is a costly process as we explain next. Let us suppose that there are n subqueries $N_1, \ldots, N_n$ to bring from n different nodes and that $N_1, \ldots, N_m$ (with $m \leq n$) need to reduce its size. For that, the response times and sizes of the answers of all the different combinations among N_i ($1 \leq i \leq n$) would have to be estimated and the best one chosen. As statistics corresponding to the repositories are not available then it would be needed to use the statistics stored about concepts and roles of the Ontology (response times and sizes of the answers) and estimate the response times and sizes for each N_i.

Heuristic H7: *Transform a subquery with several restrictions over the same role whose size is not too great by a subquery with the projection of that role.* If the size of the extension of a role is not too great and there are several restrictions over that role, then the role values can be brought instead of the conjunction of the restrictions. This favors goal 2 (but not goal 1) and that is why it is worth only if the size of the extension is really small enough. In particular if there exists a restriction of the form *all(r,c)* and *c* is cached, then to compute *all(r,c)* in the repositories has a high cost due to the kind of mapping information corresponding to it and it is more interesting to bring the values of the role *r*.

The algorithm used to apply the heuristics appears in the following. The complexity of the algorithm is polynomial in the number of terms in the Ontology, elements in the query and number of repositories involved in the query.

[8] Concepts and roles may have more than one mapping information to which we call alternative mapping informations.

[9] A query expressed in DL is unsafe if only contains restrictions of the type *all* or *atmost* [Dev93].

{ **find the cached and non-cached parts of the query** }
1) initialize `cached` **and** `non-cached` **with elements in** $\mathcal{MIS}$ **that are cached and not cached respectively**
2) for each element C in `non-cached`
2.1) try to apply H1 or H2 to C and update `cached` **and** `non-cached`
2.2) try to apply H3 to C and update `cached` **and** `non-cached`
2.3) try to apply H4 and update `cached` **and** `non-cached`
{ **obtain subset of subqueries to ask in the repositories** }
3) initialize `subqueries` **with subsets of non-cached parts to be brought from same repository node**
4) for each subquery S in `subqueries`
4.1) try to apply H5 to S and update S in `subqueries`
4.2) try to apply H7 to S and update S in `subqueries`
5) try to apply H6 to queries in `subqueries`

The general ideas behind the previous heuristics agree with other works that also consider their definition for a similar context. However, our contribution consists of defining them in the context of DL systems where there exist defined and primitive concepts and where some of them may be already cached.

5.1.1 Application to the example

In this subsection we show the application of some heuristics to the semantically equivalent query appearing in subsection 3.1 that corresponds to the query presented in the 2.2, and supposing that the contents of the cache memory are those defined in the 4.3.

As there are some elements in the set $\mathcal{MIS}$ that are not cached (*magazine* and *doc-author-name*) then heuristics are applied to the semantically equivalent query.

after having tried the set of heuristics, only H2 and H5 have been applied and therefore it has been decided that the queries Q1' and Q2 have to be answered in the underlying repositories (called **rep1** and **rep2**.

Q2: getall magazine and atleast(1,doc-author-name) and atmost(1,doc-author-name)
{*in repository* **rep1**}

Q1': rf(number-of-pages) for getall multimedia_document
{*in repository* **rep2**}

that are considered to be of a reasonable size to be loaded into the cache memory once they have been answered from the underlying repositories.

6 Query Processing in the Repositories

The set of subqueries that have been selected in the previous step have to be asked in the underlying repositories. Before generating the expressions in the query languages of the repositories involved, it is convenient to optimize the mapping information that has been expressed in a language independent of the query languages of the underlying repositories: the extended relational algebra. Therefore, in this step, for each DL subquery a corresponding optimized mapping information (that is, a more simplified extended relational algebra expression) is generated.

Definition of mapping information for a concept. Let E be a set of entities, a *mapping information for a concept* defined upon E is a set[10] of triples <R,(atr$_1$,...,atr$_n$),T>, where R is an extended relational algebra expression upon E; $atr_1, \ldots, atr_n$ are attributes of R; and $T = D_1 \times \ldots \times D_n$, where D_i is the domain of the attribute atr_i for all i between 1 and n.

Definition of mapping information for a role. Let E be a set of entities, a *mapping information for a role* defined upon E is a set of 6-tuples

<R$_{rl}$,(atrd$_1$,...,atrd$_n$),(atrn$_1$,...,atrn$_m$),T$_C$,f$_{rl}$,T$_r$>,

where R_{rl} is an extended relational algebra expression upon E; $atrd_1, \ldots, atrd_n$ and $atrn_1, \ldots, atrn_m$ are attributes of R_{rl}; T$_C$=T$_1\times\ldots\times$T$_n$, where T_i is the domain of the attribute $atrd_i$ for all i between 1 and n; f_{rl} is a function with definition f_{rl}:D$_1$ $\times \ldots \times D_m \rightarrow T$, where D_j is the domain of the attribute $atrn_j$ for all j between 1 and m and T is the range of the attribute. Finally, T$_r$=D$_1\times\ldots\times$D$_m$.

Taking into account that the mapping information has already been defined for all the concepts and roles of the Ontology, the mapping information for all the constructors that may appear in subqueries: *atleast(n,role)*, *atmost(m,role)*, *all(role,concept_type)*, *role: value*, *role: close(value)* and for combinations of concepts has to be defined. Furthermore, this mapping can be optimized when some information about the underlying data repositories is known, e.g., *functional*, *inclusion and exclusion dependencies*, *ranges of values for attributes*, *information about null values*, and when some combinations of constructors happen. Due to space limitations we present the mapping information only for one constructor, *atleast(n,role)*, and one combination, *atleast(n,role) and atmost(m,role)*, but they show the kind of optimizations performed.

6.1 atleast(n,role)

The constructor **atleast(n,role)** defines a concept, whose instances are the instances of the concept domain of *role* and that take at least n values for that *role*. Remember that the mapping for a role defines the pairs (instance,value) corresponding to a role extension.

Let r be a role, S$_r$ the mapping of r and n an integer greater than 0, the constructor $atleast(n, r)$ has S$_{atleast}$(n,r)[11] as mapping, where

[10] That set { <R$_1$,(atr$_{11}$,...,atr$_{1n_1}$),T$_1$>, <R$_2$,(atr$_{21}$,...,atr$_{2n_2}$),T$_2$>, ...<R$_m$,(atr$_{m1}$,...,atr$_{mn_m}$),T$_m$> } refers to the union of the instances represented by all the triples: <R$_1$ $\cup_{(atr_{11},\ldots,atr_{1n_1})=(atr_{21},\ldots,atr_{2n_2})}$ R$_2$ $\cup \cdots \cup$ R$_m$,(atr$_{11}$,...,atr$_{1n_1}$),T$_1$>

[11] The notation used to define a mapping information A is:
$A = \{y_1$ if cond$_1(x) \ldots y_N$ if cond$_N(x)$: $x \in$ B$\}$
The set of triples in A is the calculated by next algorithm:

```
initialize A with the empty set
for each x in set B
  if cond_1(x) add y_1 to A
  else ... else if cond_N(x) add y_N to A
```

$S_{atleast}$(n,r) = {
<R_{rl},$\overline{atrd}$,T_C>
if $n = 1 \wedge not(can\_be\_null(\overline{atrn})$[12]
<$\sigma_{\overline{atrd} \neq NULL}$($R_{rl}$),$\overline{atrd}$,$T_C$>
if $n = 1 \wedge can\_be\_null(\overline{atrn})$
ε[13]
if $n > 1 \wedge functional\_dependency(\overline{atrd} \rightarrow \overline{atrn})$[14]
<$\sigma_{count>n-1}(_{\overline{atrd}}\mathcal{F}_{count\ \overline{atrn}}(R_{rl}))$,$\overline{atrd}$,$T_C$>
in other case
: <R_{rl},$\overline{atrd}$,$\overline{atrn}$,T_C,f_{rl},T_r> ∈ S_r}

6.2 atleast(n,role) and atmost(m,role)

We present the mapping information for the combination *atleast(n,role) and atmost(m,role)* and explain why it is better than the mapping information calculated as the conjunction of the mapping informations of *atleast(n,role)* and *atmost(m,role)*. So, the mapping information for the description *atleast(n,r) and atmost(m,r)*, when $n > 1 \wedge \overline{atrd} \not\rightarrow \overline{atrn} \wedge m \geq n$ is:

$S_{atleast}$(n,r)$_{\wedge atmost}$(m,r) =

conjunction($S_{atleast}$(n,r),S_{atmost}(m,r)) =
conjunction($S_{atleast}$(n,r),complement($S_{atleast}$(m+1,r))) =

{< $\sigma_{count>n-1}(_{\overline{atrd}}\mathcal{F}_{count\ \overline{atrn}}(R_{rl}))$ –
$\sigma_{count>m}(_{\overline{atrd}}\mathcal{F}_{count\ \overline{atrn}}(R_{rl}))$, $\overline{atrd}$, T_C>
: <R_{rl},$\overline{atrd}$,$\overline{atrn}$,T_C,f_{rl},T_r> ∈ S_r}

However, a better mapping information for this expression is the following one:

{< $\sigma_{m \geq count>n-1}(_{\overline{atrd}}\mathcal{F}_{count\ \overline{atrn}}(R_{rl}))$, $\overline{atrd}$, T_C>
: <R_{rl},$\overline{atrd}$,$\overline{atrn}$,T_C,f_{rl},T_r> ∈ S_r}

The last mapping expression is much less complex than the first one and avoids one scan and computing one difference whatever it is the kind of data repository involved. In general, the procedure consists of grouping restrictions over the same role and obtaining the corresponding optimized mapping expression for each combination of restrictions.

Our main contribution in this step is that we have used the extended relational algebra as a language independent of the query languages of the underlying repositories to define the mapping information among an Ontology and the repositories. This formalism allows one to describe a wide spectrum of possible mappings and to define some optimization cases.

[12] $can\_be\_null(\overline{atrn})$ is a expression that takes the value *true* if it is possible that $\overline{atrn}$ takes *NULL* values and *false* in other case.

[13] In this case, it is avoided accessing the underlying data repositories because the answer is empty.

[14] $functional\_dependency(\overline{atrd} \rightarrow \overline{atrn})$ is a expression that takes the value *true* if the functional dependency $\overline{atrd} \rightarrow \overline{atrn}$ is satisfied and *false* in other case.

6.3 Application to the example

The mapping information for the first query:

Q1': rf(number-of-pages) for getall multimedia_document

is the following:

{ <σ rep2.rec.003$[24-27]="m"
(rep2.rec),rep2.rec.010$a,rep2.rec.300$a,str,-,int> }

This expression cannot be optimized because there is not a combination of restrictions over the same role nor information about data dependencies.

The mapping information for the second query:

Q2: getall magazine and atleast(1,doc-author-name) and atmost(1,doc-author-name)

is the next one:

{ <σ $1 \geq count \geq 1$ ((loc $\mathcal{F}$ count(name) (σ series_title="magazine" *(doc)* $\bowtie$(name=name) *rep1.doc*))),*rep1.doc.loc,str*> }

However, it can be optimized taking into account that there exists the functional dependency `rep1.doc.loc` → `rep1.doc.name`. The optimized mapping information is:

{ <σ rep1.doc.series_title="magazine" *(rep1.doc),rep1.doc.loc,str*> }

where the computation of a join and an aggregate function are avoided.

7 Correlation of the answer

Each optimized mapping information previously obtained has to be translated into a plan that depends on the concrete data repositories. A different wrapper that translates the mapping information into the concrete query language is needed for each kind of data organization involved in the Global Information System.

The corresponding SQL query in the repository *rep1* (it is an object-oriented relational database) for the next mapping information

{ <σ rep1.doc.series_title="magazine" *(rep1.doc),rep1.doc.loc,str*> }

would be:

```
select loc
from doc
where series_title="magazine"
```

The corresponding query in the repository *rep2* (it is a MARC [Pie94] file system) for the next mapping information

{ <σ rep2.rec.003$[24-27]="m"
(rep2.rec),rep2.rec.010$a,rep2.rec.300$a,str,-,int> }

would be:

```
Files:  /home/grad/MARC/UGA/oclcwkly.unicat
Projections:  010$a | 300$a
Conditions:  008$[24-27] = m
```

This information would be processed by a wrapper that accesses MARC files and retrieves the records that satisfy the corresponding conditions.

The final correlation of the answer is made in the cache memory. Different answers to the subqueries are *loaded in the cache memory* and then the answer is obtained from the cache memory using the DL system query capabilities.

In cases where it is decided not to cache all the subqueries answers, correlation should be performed outside the DL system.

8 Conclusions

We propose to use pre-existing Ontologies in order to facilitate to the user the task of querying about stored data in heterogeneous and distributed data repositories. In this paper we have presented a strategy that solves the specific problem of giving answers to formulated queries over one Ontology by accessing data stored in different data repositories connected to that Ontology. This strategy makes up a subpart of the global query processing strategy defined for a set of loosely-coupled Ontologies. In the proposed solution, we have shown first, how to obtain a semantically equivalent query that eliminates redundant elements and detects others that may increment the efficiency of the query processing. This new equivalent query expression is obtained by using the capabilities of the DL systems. In this step inconsistent queries are also detected and intensional answers may be given to the user. Next, we have explained the test to verify if the query can be answered from the cache memory. This test has to be checked only if a cache memory is available. Then, we have presented a set of heuristics applied in order to perform an efficient decomposition process of the query. Last, we have shown how optimized mapping informations corresponding to the decomposed subqueries can be obtained as a previous step to generate the sentences in the query languages used in the underlying data repositories. In summary, we can say that our work treats all the steps needed to answer queries in the considered context and presents new proposals for all of them, although due to space limitations each step is not explained in detail.

References

[AKS96] Y. Arens, C.A. Knoblock, and W. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2-3):99–130, 1996.

[Alt] Altavista. http://www.altavista.digital.com.

[ASD+91] R. Ahmed, P. Smedt, W. Du, W. Kent, M. Ketabchi, and W.A. Litwin. The Pegasus heterogeneous multidatabase system. *IEEE Computer*, 24:19–27, December 1991.

[BBMR89] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *Proceedings ACM SIGMOD-89, Portland, Oregon*, 1989.

[Bor92] A. Borgida. From type systems to knowledge representations: Natural semantics specifications for description logics. *International Journal on Intelligent and Cooperative Information Systems*, 1(1), 1992.

[CHS91] C. Collet, M. N. Huhns, and W. Shen. Resource integration using a large knowledge base in CARNOT. *IEEE Computer*, pages 55–62, December 1991.

[Dev93] P.T. Devanbu. Translating Description Logics to Information Server Queries. In *Proceedings of the ISMM International Conference on Information and Knowledge Management CIKM*, 1993.

[FRV96] D. Florescu, L. Raschid, and P. Valduriez. A methodology for query reformulation in CIS using semantic knowledge. *International Journal of Cooperative Information Systems*, 5(4):431–467, 1996.

[GIMB97] A. Goñi, A. Illarramendi, E. Mena, and J.M. Blanco. An optimal cache for a federated database system. *Journal of Intelligent Information Systems*, 9(2):125–156, September/October 1997.

[Gru93] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition, An International Journal of Knowledge Acquisition for Knowledge-Based Systems*, 5(2), June 1993.

[Gru94] T. Gruber. Theory BIBLIOGRAPHIC-DATA, September 1994. http://www-ksl.stanford.edu/-knowledge-sharing/ontologies/html/bibliographic-data/index.html.

[Inf] Infoseek. http://www.infoseek.com.

[KS96] V. Kashyap and A. Sheth. Semantic and Schematic Simililarities between Databases Objects: A Context-based approach. *The VLDB Journal*, 5(4), December 1996.

[LSK95] A.Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5(2):121–143, September 1995.

[MP97] S. Milliner and M. Papazoglou. Scalable information elicitation in large heterogeneous database networks. To be published in IEEE Internet Journal, 1997.

[Pie94] S. Piepenburg. Easy MARC: A simplified guide to creating catalog records for library automation systems: Pre-format integration, 1994.

[vLNPS87] K. von Luck, B. Nebel, C. Peltason, and A. Schmiedel. The anatomy of the BACK system. Technical Report KIT Report 41, Technical University of Berlin, Berlin, F.R.G., 1987.