Peering and Querying e-Catalog Communities

Boualem Benatallah[1], Mohand-Said Hacid[2], Hye-young Paik[1]
Christophe Rey[3] and Farouk Toumani[3]

[1] CSE, University of New South Wales, Australia, {boualem,hpaik}@cse.unsw.edu.au
[2] LIRIS, University Lyon I, France, mshacid@liris.univ-lyon1.fr
[3] LIMOS, ISIMA, University Blaise Pascal, France, {rey,ftoumani}@isima.fr

UNSW-CSE-TR-0319
July 2003

School of Computer Science and Engineering
The University of New South Wales
Sydney, 2052 Australia

Abstract

An increasing number of organisations are jumping hastily onto the online retailing bandwagon and moving their operations to the Web. A huge quantity of e-catalogs (i.e., information and product portals) is now readily available. Unfortunately, given that e-catalogs are often autonomous and heterogeneous, effectively integrating and querying them is a delicate and time-consuming task. More importantly, the number of e-catalogs to be integrated and queried may be large and continuously changing.

Consequently, conventional approaches where the development of an integrated e-catalog requires the understanding of each of the underlying catalog are inappropriate. Instead, a divide-and-conquer approach should be adopted, whereby e-catalogs providing similar customer needs are grouped together, and semantic peer relationships among these groups are defined to facilitate distributed, dynamic and scalable integration of e-catalogs.

In this paper, we use the concept of *e-catalog communities* and *peer relationships* among them to facilitate the querying of a potentially large number of dynamic e-catalogs. e-Catalogs communities are essentially containers of related e-catalogs. We propose a flexible query matching algorithm that exploits both community descriptions and peer relationships to find e-catalogs that best match a user query. The user query is formulated using a description of a given community.

1 Introduction

Nowadays, a large number of suppliers are offering access to their *product or information portals* (also called e-catalogs) via the Web. However, the technology to organise, search, integrate, and evolve these e-catalogs has not kept pace with the rapid growth of the available information. One of the key issues is *how to efficiently integrate and query large, intricate, heterogeneous information sources such as e-catalogs*.

A popular approach for locating products within e-catalogs is the use of *information retrieval based search tools* (e.g., search engines) [8]. Though search engines are useful, they present some fundamental drawbacks such as limited search capabilities and lack of information-space organisation [14]. To address this issue, there has been a renewed interest in leveraging traditional data integration techniques because of support for semantically rich queries (e.g., data source structure aware queries) [8, 13]. Clearly, a brute force data integration approach, where the development of an integrated schema requires the understanding of both structure and semantics of all schemas of sources to be integrated, is hardly applicable because of the dynamic nature and size of the Web. Therefore, the effective sharing of a potentially large number of dynamic e-catalogs, requires more scalable and flexible data sharing and querying techniques.

In this paper, we overview the design and implementation of *WS-CatalogNet*: a Web services based data sharing middleware infrastructure whose aims is to enhance the potential of e-catalogs by focusing on scalability and flexible aspects of their sharing and access. The salient features of WS-CatalogNet are:

- **Ontological Organisation:** We use the concept of *e-catalog community* as a way of organising and integrating a potentially large number of dynamic e-catalogs [8]. An e-catalog community is a container of e-catalogs (i.e., e-catalogs offering products of a common domain such as `Laptops` or `PCGames`). It provides an ontological description of desired products (e.g., offered product categories, product attributes) without referring to any actual provider. Communities of e-catalogs are established through the sharing of high-level meta-information. Actual providers can register with any community of interest to offer the desired products. E-catalog providers can join or leave any community of interest at any time. Communities provide a way to meaningfully organise and divide the information space into groups of manageable spaces (e.g., putting similar products together).

- **Decentralised Information Sharing:** Existing e-catalog organisation techniques usually use centralised categorisation and indexing schemes, whereas the participating e-catalogs are distributed and autonomous [15, 23]. A centralised categorisation and indexing model has several drawbacks including scalability, flexibility, and availability [12, 21, 18]. Given the highly dynamic and distributed nature of e-catalogs, we believe that novel techniques involving peer-to-peer centric categorisation and indexing schemes will become

increasingly attractive. Our approach features the use of *peer relationships* among e-catalog communities to allow decentralised sharing of catalog information. Query routing between communities is based on such relationships (even with no or minimal knowledge of the schema about the other party). The objective is to achieve scalable information sharing and access through the incremental meta-data driven discovery and formation of inter-relationships between e-catalog communities.

- **Flexible Selection of Relevant e-Catalogs:** Because of the variety of e-catalogs offering similar information and the large number of available e-catalogs, it is important to provide appropriate support to first select those e-catalogs that are relevant to a specific user query, before actually querying them. In addition, catalog selection techniques should support flexible matching since it is unrealistic to expect queries and catalog descriptions to exactly match. In our approach, a user query is expressed using descriptions from a community ontology. We formalise relevant e-catalog selection as a new instance of the query rewriting problem [13, 4], where a user query is reformulated in terms of:

  – local queries (i.e., the part of the user query that can be answered by some e-catalogs registered with the actual community),
  – outsourced queries (i.e., the part of the user query that can be forwarded to other communities based on specific peer relationships), and
  – remaining parts of the user query that cannot be answered by the actual community.

  We proposed a characterisation of several types of relevant query rewritings (e.g., rewritings that minimise the part of the query that can not be answered by the community). We provide a formalisation of query rewriting in the context of category hierarchy based ontologies and propose a hypergraph-based algorithm to effectively compute best rewritings of a given request.

The paper is organised as follows. Section 2 discusses the design overview, where we introduce main concepts of our approach. In section 3, we present query rewriting mechanism for selecting relevant e-catalogs. Section 4 describes an algorithm for computing query rewritings. In section 5, the implementation and experiments are presented. Finally, section 6 discusses related work and conclusions.

2 Design Overview

In this section, we introduce the concept of *catalog communities* [19] and *peer relationships* among them.

2.1 Catalog Communities

A catalog community is a *container* of product catalogs of the same domain (e.g., community of `Laptops`). It provides a description of desired products without referring to actual product providers (e.g., `www.laptopworld.com`). A community maintains categories for the domain it represents. A category is described by a set of attributes. For example, the community `Laptops` may have a category `Laptop`, which is described by attributes such as `Name`, `CPU`, `RAM`, `HDD`, `ThinkLight`, `Ultrabay`, `Weight` and `Price`. For the catalog providers to be accessible through a community, they must register their catalogs with the community. When registering, the catalog provider supplies a capability description which specifies what kind of products are supported, in terms of the community categories. The registration process allows a product provider and a community to form, a *MemberOf* relationship (e.g., `CompaqPC` provider may be a member of `DesktopPC` community).

More precisely, a community maintains two disjoint sets of categories; *core categories* and *outsourced categories*. Products that belong to the core categories are supported by the community's own members. The outsourced categories are defined as a way of extending the range of products provided in the community. The outsourced categories offer, for example, products that are not included in the core categories, that can be alternative choices, or recommendations to the products in the core categories. These outsourced categories are supported by external data sources which are outside the community (e.g., members in other communities). We use the term *community schema* to denote the description of a community which is expressed in terms of categories and attributes. Figure 1 shows the community of `Laptops` which has core, outsourced categories and registered members.

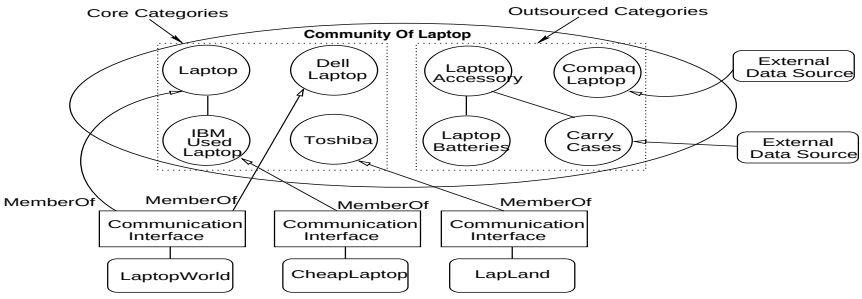

Figure 1: Community of Laptop, its categories and members

Note that the distinction between core and outsourced categories is transparent to users. In fact, users will see them as one set of categories without being aware of the differences in the sources of the data.

Synonym Index. The terms used in community schema are different from one community to another. To help solve any mismatch problem, we use synonym-based matching approach. A community description also contains a list of synonyms for each category and attribute name used in the community. For example, for the attribute `CPU`, synonyms are {`processor`, `chipset`, `chip`}. These synonyms are used to translate queries across communities.

2.2 Peering Catalog Communities

Communities are related by *PeerOf* relationships. We consider two types of peer relationships:

- **Similarity**: Community C_1 is similar to C_2, when C_1 has categories that are considered analogous, or interchangeable to C_2's core categories (e.g., `CD-RW Drives` in C_1 and `CD-RW/DVD` in C_2).
- **Companionship**: C_1 is a companion of C_2 when C_1 has categories that can support outsourced categories in C_2 (e.g., C_1 has an outsourced category `Blank CD Media` and C_2 has a category that contains such products).

Defining peer relationships determines how communities interact with each other. Figure 2 shows communities and peer relationships among them. Communities

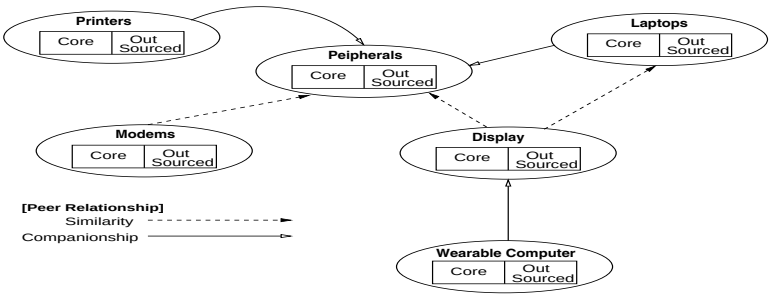

Figure 2: Peering catalog communities

and peer relationships are used to divide a vast information space into meaningful, manageable spaces.

Forming Relationships. As shown in the figure 2, the relationships are directed. Let us assume the following scenario. Community C_1 wants to form a relationship with C_2 (either in the context of similarity or companionship). Also, C_1 wants to forward the query Q to C_2 via the peer relationship. Q is composed as follows using attributes in C_1:

```
(Q)
SELECT name, displaysize
FROM   Category_DesktopReplacement
WHERE  price < 3000
```

Since the terms used in descriptions are different in the two communities, C_1 needs to know how a query described in C_1's terms should be translated to C_2's terms (i.e., mapping description). We consider three possible ways of describing the mappings.

- **Full Mapping:** C_1 provides explicit mapping description specifying how the categories in C_1 can be mapped to categories (respectively, the attributes) in C_2. This mapping description is stored in C_1. Therefore, the query expressed in C_1's terms (i.e., Q) can be translated into C_2's terms.

- **Partial Mapping:** C_1 does not specify mapping description for the attributes associated, but provide what kind of categories are available in C_1. In this case, the mapping only describes which category in C_1 maps to which is category in C_2. When C_1 only has mappings for categories, C_1 translate Q so that the category name is understood by C_2 (see FROM clause in Q.1). Then, for each attribute in Q, a list of synonyms is attached (as shown in Q.1). C_2 will use the synonyms to match the attributes.

```
(Q.1)
SELECT name (title, product),
 displaysize (display, viewable size)
FROM
 Category_PowerLaptop
WHERE
 price (retail price, listed price) < 3000
```

- **No Mapping:** C_1 does not specify any explicit mapping description with C_2. In this case, it is left to C_2 to figure out how to answer C_1's queries. When there is no mapping available, synonyms for attributes (respectively for categories) are identified and attached to Q before forwarding (as shown in Q.2).

```
(Q.2)
SELECT name (title, product),
 displaysize (display, viewable size)
FROM
 Category_DesktopReplacement (PowerLaptop)
WHERE
 price (retail price, listed price) < 3000
```

  C_2 refers to the synonyms to find alternative attribute/category names to match the terms in the query with C_2's own terms.

2.3 Basic Definitions

This section introduces a (concept) class description language that belongs to the family of description logics [3]. We use this language to describe catalog communities and peer relationships between them. The proposed language is in fact a simple description logic that is designed to represent the domain of interest in terms

of *classes* (unary predicates) and *attributes* (binary predicates). The classes and attributes characterise subsets of the objects (*individuals*) in the domain. This language will be used as a basis for formalising query reformulation across e-catalogs and communities. Class descriptions are denoted by expressions formed by means of the following constructors:

- class conjunction ($\sqcap$), e.g., the class description `Peripherals` $\sqcap$ `SCSI` denotes the class of individual instances of the class `Peripherals` and `SCSI` (e.g., a `HardDrives`),
- the universal attribute quantification ($\forall R.C$), e.g., the description $\forall$ `releaseDate.Date` specifies that the data type of the role `releaseDate` is `Date`,
- the existential attribute quantification ($\exists R$), e.g., the description $\exists$`Ultrabay` denotes the class of individuals having at least one value for the attribute `Ultrabay`.

Syntax and semantics of class descriptions are defined below.

Definition 1 ***(syntax and semantics)*** *Let $\mathcal{CN}$ be a set of class names and $\mathcal{A}$ be a set of attribute names. Class descriptions are inductively defined as follows:*

- *C is a class description for each class name $C \in \mathcal{CN}$.*
- *Let C, D be class descriptions and $R \in \mathcal{A}$ an attribute name. Then $C \sqcap D$, $\forall R.C$ and $(\exists R)$ are class descriptions as well.*

A model-theoretic semantics for this language is given by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. It consists of a nonempty set $\Delta^{\mathcal{I}}$ the domain of the interpretation, and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function associates with each class name $C \in \mathcal{CN}$ a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and with each attribute name $R \in \mathcal{R}$ a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Additionally, the extension of $\cdot^{\mathcal{I}}$ to arbitrary class descriptions has to satisfy the following equations:
$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, and
$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \forall y : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$.
$(\exists R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \exists y : (x, y) \in R^{\mathcal{I}}\}$.

Based on this semantics, the notions of subsumption and equivalence between class descriptions are defined as follows. Let C and D be class descriptions:

- C is *subsumed by* D (noted $C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretation $\mathcal{I}$.
- C is *equivalent to* D (noted $C \equiv D$) iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all interpretation $\mathcal{I}$.

The basic concepts of the data model we use (e.g., category, community, member description, etc) are defined using a class description language.

- A category represents a set of products that share common properties. A *category definition* is specified as follows: `Cname` $\equiv$ `CatDescr`, where

  - Cname is the name of the category,
  - CatDescr is a class description that defines the category Cname.

  For example, the category Laptop may be described as follows:

  $\texttt{Laptop} \equiv \texttt{Computer} \sqcap \forall\, \texttt{Name.String} \sqcap (\exists \texttt{Name}) \sqcap \forall\, \texttt{Weight.Number} \sqcap \forall\, \texttt{OS.String} \ldots$

- A *member definition* specifies the capabilities of a given product provider as follows: $\texttt{Mname\_Dname} \equiv \texttt{MDescr}$ where Mname_Dname is a member definition name made of Mname, a member name (i.e., an unique identifier of a member), and Dname, the name of a description. MDescr is a class description that specifies which data is actually provided by this member. Each member can provide several definitions. For example, the IBM provider who offers a specific type of laptops (e.g., IBM laptops) can register to the catalog community with the following member definition:

  $\texttt{IBM\_Laptop} = \texttt{Laptop} \sqcap \forall\, \texttt{ThinkLight.String} \sqcap \forall\, \texttt{Ultrabay.String} \ldots$

- A community has two types of peer relationships: companionship and similarity.
  - the companionship relationship is formalised as a set of outsourced categories, denoted by V.
  - The similarity relationship is a set of community names C_i that has similarity type peer relationships with the community, and is denoted by $\texttt{S} = \{\texttt{C}_\texttt{i}\}$.

- A community schema consists of a tuple $\texttt{CS} = (\texttt{C}, \texttt{S}, \texttt{V})$, where C is a set of category definitions, S is a set of community names that are similar to the actual community, $\texttt{V} \subseteq \texttt{C}$ is a set of outsourced categories definitions,

- A catalog community consists of a tuple $\texttt{CAT} = (\texttt{CS}, \texttt{M})$, where CAT is the catalog name, CS is a community schema, and M is a set of member definitions.

We assume that the set of class descriptions in a community catalog (i.e., core/outsourced category definitions and member definitions) is acyclic i.e., there does not exist cyclic dependencies between class definitions. Without loss of generality, we assume that the class names (respectively, the attribute names) that are used in the member definitions are disjoint from those used in the outsourced categories definitions.

3 Relevant e-Catalog Selection

In our previous work [19, 8], we used browsing and key-word based searching to locate communities of interests. In this paper, we focus on relevant e-catalogs selection. Once a community of interest is located, a query to select relevant e-catalogs can be expressed using the community schema.

Since a community does not store product data locally, answering a user query requires locating e-catalogs that are most likely relevant to a user query. It is noted that a user query is expressed over the community schema. In the remainder, we refer to such query as a community query. These e-catalogs are selected from the community members and the members of communities that can be reached via peer relationships. We propose a selection mechanism that works as follows: given a community $\texttt{CAT} = (\texttt{CS}, \texttt{M})$ and a query Q, expressed as a class description in terms of the schema of CAT, our task is to identify:

(a) a set of local member definitions that can answer all (or part of) the query Q. For each selected member, we compute the part of the query to be sent to this member.

(b) the part of the query that can be answered by companion communities. Again, for each selected companion community, we compute the precise part of the query to be forwarded to this community.

(c) the part of the query that cannot be answered by the actual local members nor by the companion communities. This part of the query will be forwarded to other communities via similarity peer relationships, according to a forwarding policy. Detailed description of the forwarding policy will be given below.

3.1 Overview

We propose a query answering approach that consists of the following elements.

A Flexible Query Rewriting Algorithm. Existing query rewriting approaches are usually based on (containment) subsumption or equivalence between a given query and its rewritings [4, 13]. These approaches are not flexible enough to cater for the identification of the part of a query that cannot be answered by a given set of communities (see, requirement (c) mentioned above). We propose a novel query rewriting algorithm where the relationship between a query Q and its rewritings goes beyond simple subsumption or equivalence. It takes as input a community $\texttt{CAT} = (\texttt{CS}, \texttt{M})$ and a query Q over the schema of CAT. It computes a set of rewritings $R(Q) = \{r_i(Q)\}$. A rewriting r_i is a triplet $r_i = (Q_{local}, Q_{outsourced}, Q_{rest})$ where:

- $Q_{local} = \{(q_j, m_j)\}$ is a set of pairs (q_j, m_j) where q_j is the part of the query Q that can be answered by the member definition m_j.

- $Q_{outsourced} = \{(Q'_j, C_{oj})\}$ where Q'_j is the part of the query Q that can be answered by the outsourced category C_{oj}. The expected results of this part of the query is a set of external members that are relevant to answering Q'_j.

- Q_{rest} is the part of the query Q that cannot be answered by the members of the actual community nor by the members of its companion communities. This part of the query will be forwarded to the similar communities according to the community forwarding policy. Again, the expected results of the forwarding is a set of external members that are relevant to answering this part of the query.

Forwarding Policy. A forwarding policy dictates whether and when the forwarding should be initiated, what should be forwarded and permitted hop count. It should be noted that this policy only concerns peer communities with similarity relationships. The basic structure of a forwarding policy is as follows:

```
forwarding policy:
   when empty|always|expand|busy
   what query|rest [-target community...]
   hop n
```

The policy requires three attributes to be specified: `when`, `what` and `hop`. The attribute `when` is used to decide when the forwarding should be initiated. If it is *empty*, the query is forwarded when the result from local members is empty (i.e., no match), whereas *always* implies that the query is always forwarded. *expand* is used when the community wishes to expand the size of the result. *busy* means that the query is forwarded when the community is busy (i.e., overloaded) with other requests.

The attribute `what` is used to decide which part of the query should be forwarded. That is, if it is set to *query*, the original query Q is forwarded. If it set to *rest*, the Q_{rest} part of Q is forwarded. When the `-target` parameter is used, the query is forwarded only to the communities that appear in the `community` clause. The default is to forward to all peers known via similarity peer relationships.

3.2 Query Rewriting

In this section, we discuss our approach for community query reformulation. The reformulation translates a community query Q [1], to queries that are expressed in terms of descriptions of community members and outsourced categories (i.e., a Q_{local} and a $Q_{outsourced}$ parts of Q) and a Q_{rest} part of a query Q. The rewriting problem can be stated as follows: Let $\mathcal{C} = \{c_i \equiv description_i,\ i \in [1, n]\}$ be a set of class definitions corresponding to member or outsourced category definitions, and let Q be a class definition that denotes a community query. Then, can Q be reformulated as a conjunction of class names $E \equiv c_{i_1} \sqcap \ldots \sqcap c_{i_m}$, with $1 \leq m \leq n$

[1] We use the terms user query and community query interchangeably.

and $c_{i_j} \in \mathcal{C}\ for\ 1 \leq j \leq m$, such that E contains as much as possible of *common information* with Q? (E is called a rewriting of Q using $\mathcal{C}$.)

To formally define this kind of query rewriting, it is necessary to characterise the notion of *"extra information"*, i.e., the information contained in one class description and not contained in the other. For that, a *difference or subtraction* operation on class descriptions is required. To tackle this issue, we use known techniques in description logics [22] as described below.

Difference Operation on Class Descriptions. An extension of description logics with a difference operation is studied in [22]. Roughly speaking, the difference of two descriptions C and D, denoted as $C - D$, is defined as being a description that contains all information which is a part of the description C but not part of the description D. This definition of difference operation requires that the second operand subsumes the first one. However, in case the operands C and D are incomparable w.r.t the subsumption relationship, then the difference $C - D$ can be computed by determining the least common subsumer[2] of C and D, that is, $C - D := C - lcs(C, D)$, where $lcs(C, D)$ denotes the least common subsumer of C and D. In the sequel, for a sake of clarity we use $C - D$ to denote $C - lcs(C, D)$.

Teege [22] provides sufficient conditions to characterise the logics where the difference operation is always semantically unique (i.e., the result of $C - D$ is unique modulo the equivalence of descriptions) and can be implemented in a simple syntactical way by constructing the set difference of sub-terms in a conjunction. From the results presented in [22], we can derive the following properties of our class description language:

- Each class description C can be expressed using a normal form, called *Reduced Clause Form* (RCF), as a conjunction of atomic clauses (e.g., $A_1 \sqcap \ldots \sqcap A_m$). In our language, a RCF of a class description C is obtained by the following normalisation process:

  - unfolding C (i.e., replacing each class name that appears in C by its description until no more defined class names occur in C).
  - recursively rewriting each description $\forall R.(B \sqcap D)$, where B, D are class descriptions, that appear in the description C into the equivalent description $\forall R.B \sqcap \forall R.D$.

  As the set of class descriptions in a community is acyclic, the normalisation process is guaranteed to terminate. At the end of the normalisation process, each conjunct that appear in the normal form of the description C constitutes an atomic clause of C.

[2] Informally, a least common subsumer of a set of (concept) class descriptions corresponds to the most specific description which subsumes all the given descriptions [3].

- The difference between two class descriptions C and D can be computed using the simple *set difference* operation between the sets of atomic clauses of C and D.

Moreover, we define the size $|C|$ of a class description C as being the number of clauses in its RCF. Note that, in the used language, a given class description has only one RCF.

Problem Statement. Now let us introduce some basic definitions to formally define the query rewriting problem. Let $\mathcal{C} = \{c_i \equiv description_i, i \in [1, n]\}$ be a set of class definitions corresponding to member definitions or outsourced categories, and E be a conjunction of some class names occurring in $\mathcal{C}$.

Definition 2 ***(query rewriting)*** *A rewriting of Q using $\mathcal{C}$ is a conjunction E of some class names c_i from $\mathcal{C}$ such that:*
$Q - E \not\equiv Q$.

Hence, a rewriting of a query Q using $\mathcal{C}$ is defined as being a conjunction of class names occurring in $\mathcal{C}$ that share some information with Q. We use the expression $rest_E(Q) = Q - E$, to denote the part of a query Q that cannot be answered by the rewriting E (i.e., the Q_{rest} part of Q, if E is selected as relevant to answering Q). In practical situations, however, we are not interested in all kinds of rewritings. Therefore, we define additional criteria to characterise the notion of *relevant rewritings*. For example, it is clearly not interesting to consider those rewritings that do not minimise the Q_{rest} part of the query Q. That is, the part of Q that cannot be answered by the local members nor by the companion communities.

Definition 3 ***(best cover rewriting)*** *A conjunction E of some class names c_i from $\mathcal{C}$ is a best cover rewriting of Q using $\mathcal{C}$ iff:*

- *E is a rewriting of Q using $\mathcal{C}$.*
- *there does not exist a rewriting E' of Q using $\mathcal{C}$ such that:* $(|(rest_{E'}(Q)|) < (|rest_E(Q)|)$.

Best cover rewritings correspond to those rewritings that minimise the size of Q_{rest} part of a query Q. Hence, they are clearly relevant rewritings in practical situations (e.g., see [5]). However, usually it may not be interesting or efficient to compute all the possible best cover rewritings. The following two definitions characterise, among the best cover rewritings, those that are more relevant in practical situations.

Definition 4 ***(non redundant rewriting)*** *A conjunction $E = c_{i_1} \sqcap \ldots \sqcap c_{i_m}$, with $1 \leq m \leq n$ and $c_{i_j} \in \mathcal{C}$ for $1 \leq j \leq m$ is a non redundant rewriting of Q using $\mathcal{C}$ iff:*

- *E is a best cover rewriting of Q using $\mathcal{C}$.*
- *$\forall j \in [1, m]$, $E' = c_{i_1} \sqcap \ldots \sqcap c_{i_{j-1}} \sqcap c_{i_{j+1}} \sqcap \ldots \sqcap c_{i_m}$ is not a best cover rewriting of Q using $\mathcal{C}$.*

Definition 4 states that it is not possible to remove any class name from the description of a *non redundant rewriting* without modifying the Q_{rest} of the query Q. In other words, the notion of non redundant rewriting characterises those rewritings that select only the local members (respectively, outsourced categories) that minimise the size of the Q_{rest} part of the query Q. Such rewritings allows minimising the number of e-catalogs that will be selected for providing the answer to a given community query. This, for example, may be useful to minimise the communication cost.

Finally, Definition 5 given below characterises the notion of *best quality rewriting*, i.e., a rewriting that maximise the user satisfaction with respect to a given set of Quality of Service (QoS) criteria. We assume that there is a *scoring function*, denoted $\texttt{qual}(\texttt{E})$, that returns a positive value which measures the quality of the rewriting E (i.e., the higher the value of $\texttt{qual}(\texttt{E})$, the higher the quality of E). This can be, for example, a multi-attribute utility that computes a quality score of an e-catalog based on pre-defined non-functional properties of the e-catalog (e.g., reliability, execution time) [25]. Further discussion about the used QoS model is outside the scope of this paper due to space reasons.

Definition 5 ***(best quality rewriting)*** *A conjunction E of some class names c_i from $\mathcal{C}$ is a best quality rewriting of Q using $\mathcal{C}$ iff:*

- *E is a best cover rewriting of Q using $\mathcal{C}$.*
- *there doesn't exist a rewriting E' of Q using $\mathcal{C}$ such that $qual(E) < qual(E')$.*

Best quality rewritings are defined as being the highest quality rewritings that minimise the size of Q_{rest} (as they are also best cover rewritings).

3.3 Mapping Rewritings to Hypergraph Transversals

In this section, we investigate the computational problems associated to our proposed query rewritings (e.g., computing all the *best quality rewritings* of a query Q). To achieve this task, we provide a full characterisation of the proposed query rewritings in terms of hypergraph transversals. Our motivation is to reuse and adapt existing techniques in hypergraph theory to efficiently solve such computational issues. First, we look at the computation of each kind of rewriting (e.g., *best quality rewritings*) and determine the complexity for obtaining a solution for it. Then, in the next section, we propose a hypergraph-based algorithm for computing the best quality rewritings of a query Q using a set of class definitions $\mathcal{C}$.

Let us first recall some useful definitions regarding hypergraphs. For more details about hypergraphs theory, we refer the reader to [6, 10].

Definition 6 ***(hypergraph and transversals)*** *[10]*
An hypergraph $\mathcal{H}$ is a pair (Σ, Γ) of a finite set $\Sigma = \{V_1, \ldots, V_n\}$ and a set Γ of subsets of Σ. The elements of Σ are called vertices, and the elements of Γ are called edges.
A set $T \subseteq \Sigma$ is a transversal of $\mathcal{H}$ if for each $\varepsilon \in \Gamma$, $T \cap \varepsilon \neq \emptyset$. A transversal T is minimal if no proper subset T' of T is a transversal. The set of the minimal transversals of an hypergraph $\mathcal{H}$ is noted $Tr(\mathcal{H})$.

We formulate rewritings computation as a problem of finding hypergraph transversals. Given a query Q and a set of class definitions $\mathcal{C} = \{c_i \equiv description_i,\ i \in [1, n]\}$, the first step is to build an associated hypergraph $\mathcal{H}_{\mathcal{C}Q}$ as follows:

- each class c_i in $\mathcal{C}$ is associated to a vertex V_{c_i} in the hypergraph $\mathcal{H}_{\mathcal{C}Q}$. Thus $\Sigma = \{V_{c_i}, i \in [1, n]\}$.
- each clause A in the normal form description of the description of the query Q is associated to an edge in the hypergraph $\mathcal{H}_{\mathcal{C}Q}$. The edge is labelled by those classes that have in their RCFs a clause A' that is equivalent to A.

For the sake of clarity we introduce the following notation: for any set of vertices $X = \{V_{c_l}, \ldots, V_{c_q}\}$, subset of Σ, we use $E_X \equiv c_l \sqcap \ldots \sqcap c_q$ to denote the class definition obtained from the conjunction of class names corresponding to the vertices in X. Inversely, for any rewriting $E \equiv c_l \sqcap \ldots \sqcap c_q$, we use $X_E = \{V_{c_l}, \ldots, V_{c_q}\}$ to denote the set of vertices corresponding to the class names in E.

Using lemmas 1 and 2 given below, we show that computing a best cover rewriting of Q using $\mathcal{C}$ (i.e., a rewriting that minimises the Q_{rest}) amounts to computing a transversal of $\mathcal{H}_{\mathcal{C}Q}$ by considering only the non empty edges.

Lemma 1 ***(characterisation of the minimal rest)*** *Let $\mathcal{H}_{\mathcal{C}Q} = (\Sigma, \Gamma)$ be the hypergraph built from a set of class definitions $\mathcal{C}$ and a query $Q = A_1 \sqcap \ldots \sqcap A_k$ provided by its RCF. The minimal rest (i.e., the rest whose size is minimal) of rewriting Q using $\mathcal{C}$ is: $Rest_{min} \equiv A_{j_1} \sqcap \ldots \sqcap A_{j_l},\ \forall j_i \in [1, k] \mid w_{A_{j_i}} = \emptyset$.*

Proof (sketch) Let $\mathcal{C} = \{c_i \equiv description_i, i \in [1, n]\}$. First, to prove the existence of a rewriting E of Q using $\mathcal{C}$ having such a rest, it is sufficient to consider E as being the combination of all the classes in $\mathcal{C}$, i.e., $E \equiv c_1 \sqcap \ldots \sqcap c_n$. Second, we show that $Rest_{min}$ has the minimal size. In the sequel, we use $\setminus_{\equiv}$ (respectively, $\in_{\equiv}$) to denote set difference of clause sets (respectively, set membership) where clauses are compared on the basis of the equivalence relation. We recall that, for any rewriting E, we have $Rest_E(Q) := Q \setminus_{\equiv} lcs_{\mathcal{C}}(Q, E)$. Assume that Q and $lcs_{\mathcal{C}}(Q, c_i), \forall i \in [1, n]$, are given by their RCFs. We have $A_{j_i} \in_{\equiv} Q$ and $A_{j_i} \notin_{\equiv} lcs_{\mathcal{C}}(Q, c_i)$ for all $j_i \in [1, k]$ such that $w_{A_{j_i}} = \emptyset$ (by construction of $\mathcal{H}_{\mathcal{C}Q}$). Then we can prove that for all $j_i \in [1, k]$ such that $w_{A_{j_i}} = \emptyset$ we have $A_{j_i} \notin_{\equiv} lcs_{\mathcal{C}}(Q, E)$ (since E is a conjunction of some classes c_{i_j} and we use a description language

with structural subsumption[3]). This implies that for all $j_i \in [1,k]$ such that $w_{A_{j_i}} = \emptyset$ we have $A_{j_i} \in Q \setminus_{\equiv} lcs_{\mathcal{C}}(Q,E)$ and thus $|Rest_{min}| \leq |Rest_E(Q)|$ for any rewriting E of Q.

□

From the previous lemma, we know that the minimal rest of a query Q using $\mathcal{C}$ is always unique and is equivalent to $Rest_{min}$. Hence, a given rewriting E of Q using $\mathcal{C}$ is a best cover rewriting if and only if $rest_E(Q) \equiv Rest_{min}$.

Lemma 2 ***(characterisation of best cover rewritings)*** *Let $\widehat{\mathcal{H}}_{\mathcal{C}Q} = (\Sigma, \Gamma')$ be the hypergraph built by removing from $\mathcal{H}_{\mathcal{C}Q}$ the empty edges. A rewriting $E_{min} \equiv c_{i_1} \sqcap \ldots \sqcap c_{i_m}$, where $1 \leq m \leq n$ and $c_{i_j} \in \mathcal{C}$ for $1 \leq j \leq m$, is a best cover rewriting of Q using $\mathcal{C}$ iff $X_{E_{min}} = \{V_{c_{i_j}}, j \in [1,m]\}$ is a transversal of $\widehat{\mathcal{H}}_{\mathcal{C}Q}$.*

Proof (sketch) The main steps of the proof are:
Lemma 1 $\Leftrightarrow \forall w_{A_i} \in \widehat{\mathcal{H}}_{\mathcal{C}Q}$, the corresponding clause $A_i \not\in_{\equiv} Q \setminus_{\equiv} lcs_{\mathcal{C}}(Q, E_{min})$
$\Leftrightarrow \forall w_{A_i} \in \widehat{\mathcal{H}}_{\mathcal{C}Q}, A_i \in_{\equiv} lcs_{\mathcal{C}}(Q, E_{min})$
$\Leftrightarrow \forall w_{A_i} \in \widehat{\mathcal{H}}_{\mathcal{C}Q}, \exists c_{i_j}$ with $j \in [1,m] \mid A_i \in_{\equiv} lcs_{\mathcal{C}}(Q, c_{i_j})$ (since we use a description language with structural subsumption)
$\Leftrightarrow \forall w_{A_i} \in \widehat{\mathcal{H}}_{\mathcal{C}Q}, \exists V_{c_{i_j}} \in X_{E_{min}} | V_{c_{i_j}} \in w_{A_i}$
$\Leftrightarrow X_{E_{min}}$ is a transversal of $\widehat{\mathcal{H}}_{\mathcal{C}Q}$ (since $X_{E_{min}}$ intersects each edge of $\widehat{\mathcal{H}}_{\mathcal{C}Q}$)

□

This lemma characterises the best cover rewritings in terms of hypergraph transversals. The following characterisation of non redundant rewritings can be straightforwardly derived from lemma 2.

Lemma 3 ***(characterisation of non redundant rewritings)*** *A rewriting $E \equiv c_{i_1} \sqcap \ldots \sqcap c_{i_m}$, with $1 \leq m \leq n$ and $c_{i_j} \in \mathcal{C}$ for $1 \leq j \leq m$, is a non redundant rewriting of Q using $\mathcal{C}$ iff $X_E = \{V_{c_{i_j}}, j \in [1,m]\}$ is a minimal transversal of $\widehat{\mathcal{H}}_{\mathcal{C}Q}$.*

Now let us consider the characterisation of the best quality rewritings. Although, a detailed description of the used quality model is outside the scope of this paper, we assume l essential quality criteria (e.g., reliability, execution time) that are common to all community members:

$$q = \{q_1, q_2, \ldots, q_l\} \tag{1}$$

Therefore, we assume that for each potential member c_i, there is a *quality vector* $q(c_i)$ which represents the *goodness* of selecting this member in relation to all essential quality criteria. This vector is defi ned as:

[3]In description languages with structural subsumption, a clause that appears in the RCF of a conjunction of some classes also appears in at least the RCF of one of these classes [22].

$$q(c_i) = (q_1(c_i), q_2(c_i), \ldots, q_l(c_i)) \quad (2)$$

For simplicity, we assume that the value of each $q_k(c_i)$ has been scaled to [0, 1] and the higher the value is, the higher the quality is. It is worth noting that QoS criteria are not considered when selecting companion communities. In fact, each companion community will be in charge of selecting the best quality external members that are most likely relevant to answer the part of the query forwarded to it. Hence, in a given community, we assume that the outsourced categories have equal quality scores as stated below:

$$q_j(c_i) = 1 \, \forall c_i \in \mathtt{V} \, and \, \forall j \in [1, l]. \quad (3)$$

In this way, the outsourced categories and local members are treated uniformly with respect to QoS criteria. Finally, to be able to evaluate the quality of a given rewriting, we assume that there is a function f, called *optimisation function*, that computes the quality of a set of member definitions (respectively, outsourced category definitions) based on their respective quality vectors. More precisely, given a rewriting $E \equiv c_{i_1} \sqcap \ldots \sqcap c_{i_m}$, the quality of E is determined as follows:

$$qual(E) = f(q(c_{i_1}), \ldots, q(c_{i_m}))$$

As before, we assume that the function f is scaled to [0, 1] and the higher the value is, the higher the quality of the rewriting is. Let us now characterise the notion of quality of a rewriting in the context of hypergraphs.

Definition 7 ***(quality of a set of vertices)***
Let $X = \{V_{c_i}, \ldots, V_{c_j}\}$ be a set of vertices of the hypergraph $\mathcal{H}_{\mathcal{C}Q}$. We define the notion of a quality of a set of vertices as: $qual(X) = f(q(c_i), \ldots, q(c_j))$.

Computing the best quality rewritings consist of determining, from the best cover rewritings, the ones that have the highest quality (See definition 5). In a hypergraph, this computation is characterised as follows.

Lemma 4 ***(characterisation of best quality rewritings)*** *A rewriting $E \equiv c_{i_1} \sqcap \ldots \sqcap c_{i_m}$, with $1 \leq m \leq n$ and $c_{i_j} \in \mathcal{C}$ for $1 \leq j \leq m$, is a best quality rewriting of Q using $\mathcal{C}$ iff:*

- *$X_E = \{V_{c_{i_j}}, j \in [1, m]\}$ is a transversal of $\widehat{\mathcal{H}}_{\mathcal{C}Q}$,*
- *there doesn't exist a transversal $X_{E'}$ of $\widehat{\mathcal{H}}_{\mathcal{C}Q}$ such that $qual(X_E) < qual(X_{E'})$.*

Complexity Analysis. Based on known results in hypergraphs theory [10], we provide hereafter complexity analysis with respect to the computation of the proposed query rewritings.

Let $\mathcal{C}$ be a set of class definitions and Q be a community query. Let n be the total number of class definitions in $\mathcal{C}$ and $m = |Q|$ be the size of a query Q.

- A best cover rewriting of Q using $\mathcal{C}$ can be found in time $O(m)$.
- A non redundant rewriting can be found in time $O(m.n)$.
- Computing all the non redundant rewritings:
  From lemma 3, this problem amounts to computing the minimal transversals of a hypergraph. It is known that computing the minimal transversals of an hypergraph is inherently exponential since the size of the outputs (i.e., the number of the minimal transversals) is exponential in the input size [10]. However, whether there is an *output-polynomial* time algorithm (i.e., an algorithm that works in polynomial time if the number of minimal transversals is taken into account) for computing the minimal transversals of a hypergraph is still an open problem. In [11], it is shown that the generation of hypergraph transversals, and hence computation of minimal transversals, can be done in incremental subexponential time $k^{O(logk)}$, where k is the combined size of the input (the size of the hypergraph) and the output (i.e., the number of minimal transversals). To the best of our knowledge, this is the best theoretical upper time bound for computing minimal transversals of a hypergraph.
- The general problem of computing a best quality rewriting is **NP**-hard. This result is based on a polynomial reduction of the **NP**-hard problem of finding a minimal cardinality transversal of a hypergraph [10] to a particular instance of the problem of computing the best quality rewritings. However, in some particular cases, depending on the characteristics of the optimisation function f, best quality rewritings can be computed efficiently. For example, in the case of additive functions, it suffices to select from each edge of the considered hypergraph the member with the highest quality score to obtain a best quality rewriting. Hence, in this case a best quality rewriting can be computed in time $O(m.n)$.

4 Algorithm for Computing Best Quality Rewritings

In this section, we describe an algorithm, called $BestQRC$, for computing best quality rewritings. Let $\mathcal{C} = \{c_i \equiv description_i, i \in [1, n]\}$ be a set of class definitions and Q be a class definition that denotes a community query. The $BestQRC$ computes the best rewritings of the query Q using $\mathcal{C}$ with respect to a given optimisation function f. We recall that the function f is defined on a set of quality vectors $q(c_i)$, where c_i denotes a class name. Each class name corresponds to a

member definition name or to an outsourced category name. When designing this algorithm, we considered an optimisation function f that satisfies the following requirements[4]:

(R1) f is strictly monotonic (i.e, $Y \subset Y'$ implies that $f(Y) > f(Y')$, where Y, Y' denote sets of quality vectors). This means that the higher the number of selected sources (i.e, local members and outsourced categories), the lower the quality of the rewriting. Consequently, in this case, the best quality rewritings should be selected among the non redundant rewritings.

(R2) f is a global optimisation function in the sense that it must be performed on the whole rewriting to get its quality score (i.e., $f(Y)$ cannot be computed incrementally using a given intermediary result $f(Y')$ where $Y' \subset Y$).

These two requirements lead to a computation problem that is hard to deal with. Indeed, based on the analysis presented in the previous section and with respect to the requirement (R2), it can be shown that computing *best quality non redundant rewritings* in this case is an **NP**-hard problem. However, these two requirements also correspond to a likely realistic and desirable situations. Based on the analysis presented in the previous section, it can be shown that computing best quality non redundant rewritings can be mapped to finding the minimal transversals, with maximal quality, of the hypergraph $\widehat{\mathcal{H}}_{\mathcal{CQ}}$.

A classical algorithm for computing the minimal transversals of a hypergraph is presented in [6, 10]. Using this algorithm, computing the minimal transversals with the highest quality consists of (i) computing all the minimal transversals, and then (ii) choosing those transversals that have the highest quality.

The $BestQRC$ algorithm presented below makes the following improvements (i.e., optimisations) with respect to the classical algorithm:

1. it reduces the number of candidates in the intermediary steps by generating only the minimal transversals, and

2. it uses, at the intermediary steps, a combinatorial optimisation technique, namely *Branch-and-Bound* [17], in order to prune those candidate transversals which will not generate transversals with a maximal quality.

The first optimisation generates minimal transversals at each iteration (line 10 of the algorithm). We use a necessary and sufficient condition (provided by Theorem 1 described below) to describe a pair (X_i, c_j) that will generate a non minimal transversal at iteration i, where X_i is a minimal transversal generated at iteration $i-1$ and c_j is a vertex of the i^{th} edge.

[4]The requirements (R1) and (R2) are inspired from a real life situation encountered during the application of our work in the context of an European project (cf. section 5).

Algorithm 1 $BestQRC$ (skeleton)

Require: a query $Q = A_1 \sqcap \ldots \sqcap A_k$ provided by its RCF and a community catalog $CAT = (CS, M)$, with $CS = (C, S, V)$.
Ensure: The set $R(Q) = \{r(Q)\}$ of the best quality rewritings of Q using $M \cup V$.
1: Let $\mathcal{C} = M \cup V$.
2: Build the associated hypergraph $\mathcal{H}_{\mathcal{CQ}} = (\Sigma, \Gamma')$.
3: compute $Q_{rest} = A_{j_1} \sqcap \ldots \sqcap A_{j_l}$, $\forall j_i \in [1, k] \mid w_{A_{j_i}} = \emptyset$.
4: Build the associated hypergraph $\widehat{\mathcal{H}}_{\mathcal{CQ}} = (\Sigma, \Gamma')$.
5: $R(Q) \leftarrow \emptyset$ – Initialisation of the best quality rewriting set.
6: $Tr \leftarrow \emptyset$ – Initialisation of the minimal transversal set.
7: Compute a minimal transversal Y of $\widehat{\mathcal{H}}_{\mathcal{CQ}}$
8: $QualEval \leftarrow qual(Y)$. – Initialisation of QualEval
9: **for all** edge $E \in \Gamma'$ **do**
10: $Tr \leftarrow$ the new generated set of the minimal transversals. – Using Theorem 1.
11: Remove from Tr the transversals whose quality is less than $QualEval$.
12: **end for**
13: **for all** $X = \{V_{c_{i_1}}, \ldots V_{c_{i_m}}\} \in Tr$ such that $qual(X) = QualEval$ **do**
14: $Q_{local} = \{(q_{i_p}, c_{i_p}), p \in [1, m]\}$, where $q_{i_p} = A_{j_1} \sqcap \ldots \sqcap A_{j_l}, \forall j_i \in [1, k] \mid V_{c_{i_p}} \in w_{A_{j_i}}$, where $c_{i_p} \notin \mathbb{V}$.
15: $Q_{outsourced} = \{(q_{i_p}, c_{i_p}), p \in [1, m]\}$ and $q_{i_p} = A_{j_1} \sqcap \ldots \sqcap A_{j_l}, \forall j_i \in [1, k] \mid V_{c_{i_p}} \in w_{A_{j_i}}$ and $c_{i_p} \in \mathbb{V}$.
16: $R(Q) = R(Q) \cup \{r(Q) = (Q_{local}, Q_{outsourced}, Q_{rest})\}$
17: **end for**
18: return $R(Q)$

Theorem 1 *Let $\mathcal{H}$ be an hypergraph and its associated set of minimal transversals $Tr(\mathcal{H}) = \{X_i \mid i \in \{1, .., m\}\}$. Let $e = \{c_j \mid j \in \{1, .., n\}\}$ be an extra edge of $\mathcal{H}$. Let $\mathcal{H}' = \mathcal{H} \cup e$.*
$\forall i \in \{1, .., m\}$ it holds:

a) if $X_i \cap e \neq \emptyset$:
$\forall j \in \{1, .., n\}$:

- *$(c_j \notin X_i \cap e) \rightarrow (X_i \cup \{c_j\}$ is a transversal of $\mathcal{H}'$ that is not minimal$)$*
- *$(c_j \in X_i \cap e) \rightarrow (X_i \cup \{c_j\} = X_i$ is a minimal transversal of $\mathcal{H}')$*

b) if $X_i \cap e = \emptyset$:
$\forall j \in \{1, .., n\}$:
$(X_i \cup \{c_j\}$ is a transversal of $\mathcal{H}'$ that is not minimal$) \leftrightarrow (\exists X_k \in Tr(\mathcal{H}) \mid X_k \cap e = \{c_j\}$ and $X_k \setminus \{c_j\} \subset X_i)$

Proof a): straightforward.
b): we recall (1) $X_i \cap e = \emptyset$, (2) $X_i \in Tr(\mathcal{H})$, (3) $\mathcal{H}' = \mathcal{H} \cup e$ and (4) $c_j \in e$.
Let $(*) = X_i \cup \{c_j\}$ is a non minimal transversal of $\mathcal{H}'$.
Let $(**) = \exists X_k \in Tr(\mathcal{H}) \mid X_k \cap e = \{c_j\}$ and $X_k \setminus \{c_j\} \subset X_i$. Let's note $tr(\mathcal{H})$ the set of all transversals of $\mathcal{H}$ (recall that $Tr(\mathcal{H})$ is the set of all *minimal* transversals of $\mathcal{H}$).

$$
\begin{aligned}
(*) \quad &\overset{(1,2,3)}{\Leftrightarrow} \quad \exists Y \mid Y \in Tr(\mathcal{H}') \text{ and} \\
&\qquad\qquad Y \subset X_i \cup \{c_j\} \\
&\overset{(1,2,3,4)}{\Leftrightarrow} \quad \exists Y \mid Y \in Tr(\mathcal{H}') \text{ and} \\
&\qquad\qquad c_j \in Y \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \subset X_i \\
&\Leftrightarrow \quad \exists Y \mid \forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset \text{ and} \\
&\qquad\qquad \forall c_y \in Y,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}') \text{ and} \\
&\qquad\qquad c_j \in Y \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \subset X_i \\
&\Leftrightarrow \quad \exists Y \mid \forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \notin tr(\mathcal{H}') \text{ and} \\
&\qquad\qquad \forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}') \text{ and} \\
&\qquad\qquad c_j \in Y \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \subset X_i \\
&\overset{(2,3)}{\Leftrightarrow} \quad \exists Y \mid \forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \notin tr(\mathcal{H}) \text{ and} \\
&\qquad\qquad \forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}') \text{ and} \\
&\qquad\qquad c_j \in Y \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \subset X_i \\
&\overset{(3,4)}{\Leftrightarrow} \quad \exists Y \mid \forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \notin tr(\mathcal{H}) \text{ and} \\
&\qquad\qquad \forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}) \text{ and} \\
&\qquad\qquad c_j \in Y \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \subset X_i \\
&\overset{(3,4)}{\Leftrightarrow} \quad \exists Y \mid \forall e' \in \mathcal{H},\ Y \cap e' \neq \emptyset \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \notin tr(\mathcal{H}) \text{ and} \\
&\qquad\qquad \forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}) \text{ and} \\
&\qquad\qquad c_j \in Y \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \subset X_i \\
&\Leftrightarrow \quad \exists Y \mid Y \in Tr(\mathcal{H}) \text{ and} \\
&\qquad\qquad c_j \in Y \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \subset X_i \\
&\overset{(1,4)}{\Leftrightarrow} \quad \exists Y \mid Y \in Tr(\mathcal{H}) \text{ and} \\
&\qquad\qquad Y \cap e = c_j \text{ and} \\
&\qquad\qquad Y \setminus \{c_j\} \subset X_i \\
&\Leftrightarrow \quad (**)
\end{aligned}
$$

□

The second optimisation consists of a *Branch-and-Bound* like enumeration of transversals. First, a simple heuristic is used to efficiently compute the quality of a *good* transversal (i.e., a transversal expected to have a high quality). The quality score is stored in the variable $BestQRC$ (line 8 of the algorithm). As we consider candidates in intermediate steps, any candidate transversal that has a quality score less than $QualEval$ (line 11) is eliminated from $Tr(\widehat{\mathcal{H}}_{\mathcal{CQ}})$. As the optimisation function f is strictly monotonic, the quality score of considered candidate transversal cannot be better than that of already computed minimal transversal.

At the end of the algorithm (lines 13 to 16), each computed minimal transversal $X \in Tr$ is translated into a rewriting $r(Q) = (Q_{local}, Q_{outsourced}, Q_{rest})$ which constitutes a best quality non redundant rewriting of the query Q using C. The Q_{rest} part of a query is computed at the beginning of the algorithm (line 3). The Q_{local} (respectively, $Q_{outsourced}$) part of a given rewriting is computed as follows (lines 14 and 15): for vertices $V_{c_{i_p}}$ in the transversal X, a pair (q_{i_p}, c_{i_p}) is created and added to the Q_{local} part (respectively, the $Q_{outsourced}$ part) of the actual rewriting if c_{i_p} is a member definition name (respectively, outsourced category name). The associated query q_{i_p} consists of the conjunction of the clauses A_j of the query Q such that the corresponding hypergraph edge w_{A_j} contains the vertices $V_{c_{i_p}}$.

5 *WS-CatalogNet* Implementation

Prototype. To evaluate our approach, we have implemented a prototype called *WS-CatalogNet*, which is a web service based environment for building catalog communities. This prototype consists of a set of integrated tools that allow catalog providers to create communities, member relationships (i.e., between a catalog provider and a community) and peer relationships (i.e., between communities). It also allows users to access the communities and submit queries to them. In WS-CatalogNet, both product catalogs and communities are represented as web services (cf. http://www.w3.org/2002/ws/). Overall, the prototype has been implemented using Java and WSDK 5.0[5]. A private UDDI[6] registry (i.e. hosted by the WS-CatalogNet platform) is used as a repository for storing web services' information. In UDDI registry, every web service is assigned to a tModel. A tModel provides a semantic classification of a service's functionality and a formal description of its interfaces. We design specific tModels for product catalogs and for communities. In Table 1, we list few of the operations in the tModels[7].

When building a community, the community provider has to download two special classes named `QueryProcessor` and `QueryRouter`, which are provided by our system. The class `QueryProcessor` provides methods for processing query requests for each catalog community. It implements the $BestQRC$ rewriting al-

[5]IBM Web Services Development Kit 5.0 (www.alphaworks.ibm.com/tech/webservicestoolkit)

[6]Universal Description, Discovery and Integration

[7]For clarity reason, we omit detailed signature of the operations.

Table 1: Main operations in tModel for members and communities

Operations for Member `M`	**Description**
Query()	Invoked to query `M`
GetInterface()	Invoked to get categories and attributes of `M`
Operations for Community `C`	**Description**
Query()	To query `C`
GetInterface()	To get the categories and attributes of `C`
ForwardQuery()	To forward queries to other communities
AddPeer()	To add a peer community
RemovePeer()	To remove a peer community

gorithm. Community metadata (i.e., descriptions of categories, members, mapping and forwarding policies) are stored as XML documents. The class `QueryRouter` provides methods for routing queries based on the forwarding policies and the peer mappings. Both classes are lightweight and the only infrastructures that they require are standard Java libraries, a JAXP-compliant XML parser, and a SOAP server. In order to have a better understanding of the issue of e-catalog selection, and validate our query rewriting approach, we used our prototype in the context of the MKBEEM[8] project which aims at providing electronic marketplaces with intelligent, knowledge-based multi-lingual services. In this application, we used communities with approximately 300 categories and 50 e-catalogs. Indeed, this application has shown the effectiveness of the proposed query rewriting mechanism in two distinct end-user scenarios, namely: (i) business to consumer on-line sales, and (ii) Web based catalogs.

Preliminary Experiments. In order to evaluate the performance of the $BestQRC$ algorithm, we built a simulation testbed. In this testbed, we have implemented the two optimisations presented in section 4 as two separate options of the $BestQRC$ algorithm, namely option Pers for the optimisation provided by the theorem 1 and option BnB for the optimisation that uses the *Branch-and-Bound* technique. We have then evaluated up to 6 versions of the $BestQRC$ algorithm corresponding to different combinations of these optimisation options. The simulation testbed includes a tool that generates random XML-based test community schemas and queries. All experiments have been performed using a PC with a Pentium III 500 MHz and 384 Mo of RAM.

We have considered three test scenarios with differences in the size of community schema, number of e-catalogs, and the query expressions (see Table 2).

[8]MKBEEM stands for Multi-lingual Knowledge Based European Electronic Marketplace (IST-1999-10589, 1st Feb. 2000 - 1st Dec. 2002). *http://www.mkbeem.com*

Table 2: Configurations

Configurations	Case 1	Case 2	Case 3
Number of defined categories in communities	365	1334	3405
Number of e-catalogs	366	660	570
Number of (atomic) clauses in the query	6	33	12

We have run the 6 versions of the $BestQRC$ algorithm on the test cases. The overall execution time results are given in Figure 3[9]. This figure shows that for

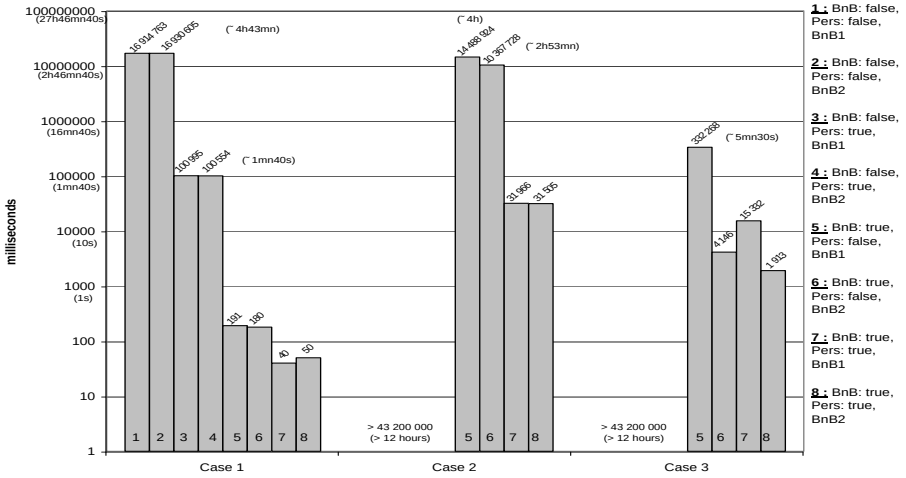

Figure 3: Execution time

cases 1 and 3 (respectively, case 2), there is at least a version of the algorithm that runs in less than two seconds (respectively, in less than 30 seconds). Although Figure 3 shows that there are significant differences in performance between the different versions of the algorithm, in each case, there is at least one efficient version of the algorithm even when community schema is quite large (i.e., large number of categories). It should be noted that this preliminary experiment only concerns the performance of the rewriting algorithm. Ongoing experiments include performance and scalability studies that involve interactions (i.e., forwarding queries) in a network of communities formed via peer relationships.

[9]Note that versions 1 and 2 of the algorithm (respectively, 3 and 4) are similar as both run $BestQRC$ without BnB, and what distinguishes 1 from 2 (respectively, 3 from 4) is the way the option BnB is implemented (BnB1 or BnB2).

6 Related Work and Conclusions

In this section, we examine work done in e-catalogs integration. We also briefly discuss peer-to-peer data sharing approaches. Existing e-catalogs integration approaches typically rely on a global schema integration style [23]. For instance, [15] proposes an e-catalog integration approach, in which all categories of products are organised in a single graph structure and each leaf links to source catalog's product attribute tree which represent local catalog's product classification scheme. As mentioned before, a static approach, where the development of an integrated schema requires the understanding of both structure and semantics of all schemas of sources to be integrated, is hardly scalable because of the potentially large number and dynamic nature of available e-catalogs. Other approaches (e.g., [2]) focus on extracting structured and integrated schemas from documents that contain unstructured product description. Our work is not concerned with this issue.

P2P computing is currently a very active research and development area. From information sharing point of view, the first generation of P2P systems (e.g., Gnutella, Napster) focused on files sharing (e.g., music, video clips). Query routing among peers in such approaches is discussed [9]. These systems support limited querying capabilities (e.g., keyword based search). Effective data sharing requires support for more structured querying capabilities to exploit the inherent semantics in data ([7, 20]). [16] proposes a super-peer based routing [24] in RDF-based P2P information sources. The proposed approach focuses on indexing RDF resources.

Few approaches that leverage database-like information sharing and querying techniques in P2P environments emerged recently [1, 7]. PeerDB [18] uses a relational model to describe the schema of a peer data source. Relations and attributes are associated to keywords (i.e, synonyms of relation and attribute names). PeerDB uses an Information Retrieval (IR) approach for query routing to avoid the explicit specifications of mapping among peer schemas. The issue information space organisation is not considered. Piazza [12] considers the issue of schema mediation in P2P environments. It uses a relational model to describe peer schemas. It proposes a language for specifying mappings among peers. It also proposes a query reformulation algorithm for the proposed mediation framework. [20] proposes the notions of mutant query plan (MQP) and multi-hierarchy namespaces to support query processing in P2P environments. A MQP includes verbatim XML encoded data, references to actual resource locations (URL) and references to abstract resource names (URN). A peer can mutate an incoming MQP by either resolving URNs to URLs, or substituting a sub-plan with the evaluated XML encoded data.

Our work provides complementary contributions to related work on data sharing and querying in peer-to-peer environment. We focus on providing support for achieving effective and efficient access to e-catalogs resident data. Since scalability is of great importance in e-catalog environments, the information space is organised in communities that are inter-related using peer relationships. The model that we use to describe community schemas relies on simple concepts (categories and attributes) that we found to be useful and commonly used to describe e-catalogs

content and capabilities. We consider different types of peer relationships between communities (i.e, companionship and similarity relationships). Such flexibility allows to establish different interaction types among communities. The specification of mappings in our approach combines: (i) IR style where no explicit mapping description is provided and (ii) full mapping description when it is possible or desired (e.g., in the case of companionship relationships). We formalised e-catalogs selection as a rewriting process for e-catalog communities. Query routing among peer communities is based on forward policies. We proposed a novel hypergraph-based algorithm to effectively select relevant e-catalogs for a given query.

It should be noted that, the main approaches to *rewriting using terminologies* [4] are: *the minimal rewriting problem* [4] and *query rewriting using views* [13]. These approaches are based on (containment) subsumption or equivalence between a given query and its rewritings. In our approach, the proposed algorithm finds rewritings that 'best match' a given query with respect to a quality function, where the relationships between a query and its rewritings goes beyond containment or equivalence. We believe that there is a need for such a flexible query rewriting approach to cope with the high dynamicity and heterogeneity of the Web-based environments.

References

[1] K. Aberer, P. Cudre-Mauroux, and M. Hauswirth. The Chatty Web: Emergent Semantics Through Gossiping. In *WWW'03*, Budapest, Hungary, May 2003.

[2] R. Agrawal and R. Srikant. On Integrating Catalogs. In *WWW'01*, Hong Kong, China, May 2001.

[3] F. Baader, D. Calvanese, D. McGuinness, and editors D. Nardi andP. Patel-Schneider. *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge University Press, 2003.

[4] Franz Baader, Ralf Küsters, and Ralf Molitor. Rewriting Concepts Using Terminologies. In *KR'00, Colorado, USA*, pages 297–308, April 2000.

[5] B. Benatallah, M-S. Hacid, C. Rey, and F. Toumani. Semantic Reasoning for Web Services Discovery. In *ESSW Workshop, Budapest, Hungary*, May 2003.

[6] C. Berge. *Hypergraphs*, volume 45 of *North Holland Mathematical Library*. Elsevier Science Publishers B.V., 1989.

[7] P. Bernstein, F. Giunchigiloa, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *WebDB'02*, Madison, Wisconsin, June 2002.

[8] A. Bouguettaya, B. Benatallah, L. Hendra, M. Ouzzani, and J. Beard. Supporting Dynamic Interactions among Web-based Information Sources. *IEEE TKDE*, 12(5):779–801, Sept/Oct 2000.

[9] A. Crespo and H. Garcia-Molina. Routing Indices For Peer-to-Peer Systems. In *ICDCS'02*, Vienna, Austria, July 2002.

[10] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[11] M.L. Freidman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21:618–628, 1996.

[12] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *ICDE'03*, Bangalore, India, March 2003.

[13] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[14] L. Liu. Query Routing in Large-scale Digital Library Systems. In *ICDE'99*, Sydney, Australia, March 1999.

[15] S. Navathe, H. Thomas, M. Satits A., and A. Datta. A Model to Support E-Catalog Integration. In *IFIP Conference on Database Semantics*, Hong Kong, April 2001.

[16] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Lser. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. In *WWW'03*, Budapest, Hungary, May 2003.

[17] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley&Sons (New York), 1988.

[18] W.S. Ng, B.C. Ooi, K.L. Tan, and A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *ICDE'03*, Bangalore, India, March 2003.

[19] H. Paik, B. Benatallah, and R. Hamadi. Dynamic Restructuring of E-Catalog Communities Based on User Interaction Patterns. *WWW Journal*, 5(4):325–366, 2002.

[20] V. Papadimos, D. Maier, and K. Tufte. Distributed Query Processing and Catalogs for Peer-to-Peer Systems. In *CIDR'03*, Asilomar, CA, January 2003.

[21] M. Papazoglou, B. Kramer, and J. Yang. Leveraging Web-Services and Peer-to-Peer Networks. In *CAiSE'03*, Klagenfurt, Austria, June 2003.

[22] G. Teege. Making the difference: A subtraction operation for description logics. In *KR'94*, San Francisco, CA, 1994.

[23] G. Yan, W. Ng, and E. Lim. Product Schema Integration for Electronic Commerce–A Synonym Comparison Approach. *IEEE TKDE*, 14(3), May/June 2002.

[24] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *ICDE'03*, Bangalore, India, March 2003.

[25] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality-driven Web Service Composition. In *WWW'03*, Budapest, Hungary, May 2003.