On the Query Refinement in the Ontology-Based Searching for Information

Nenad Stojanovic

Institute AIFB,
University of Karlsruhe,
76128 Karlsruhe, Germany
nst@aifb.uni-karlsruhe.de

Abstract. One of the main problems in the (web) information retrieval is the ambiguity of the users' queries, since the users tend to post very short queries. This seems to be valid for the ontology-based information retrieval in which the domain ontology is used as the backbone of the searching process. In this paper we present a novel approach for determining the possible refinements of an ontology-based query. The approach is based on measuring the ambiguity of a query with respect to the original user information need. We defined several types of the ambiguities concerning the structure of the underlying ontology and the content of the information repository. For each of these ambiguities we defined the set of refinements/explanations, which help in more efficient searching for information in an ontology-based information repository.

1 Introduction

The basic problem in the searching for knowledge provided by traditional IR systems is that it only partially reflects the process which humans use in searching for goods in the bricks-and-mortar environment. Briefly, in the non-virtual search, there exist a shop assistant, who helps a user to express his need more clearly and guides the user through the searching space. It means that a user refines his query incrementally according the suggestion made by a shop assistant. In [1] we present the conceptual architecture of the Librarian Agent which simulates the role a human librarian plays in the searching for information resources in a library. The Librarian Agent analyses a user's query based on: (i) the structure of the used vocabulary, (ii) the capacity of the information repository and (iii) the information about the users' past activities in the portal. Based on these analyses, the agent, through an interactive dialogue, guides the users in more efficient searching for information. Particularly, for a query given by a user, the agent measures its ambiguity. In case of high ambiguity, the agent suggests the user the most effective reformulation of the query, considering the underlying vocabulary, the information repository and the agents' experience (based on the past behaviour of users). Last, the agent analyses the users' requests off-line and compares the users' interests with the capacity of the information repository, in order to find which "new titles" should be obtained or which topics are no more interesting for users. The approach assumes the existence of a common vocabulary which is used for creating the queries as well as for providing meta-information about the content of information resources. In this case the problem of merging query's and document's term space is resolved on the semantic level, by using synonyms – very similarly to the solution a human librarian uses in resolving the meaning of a term. Moreover, in order to simulate background knowledge which a human librarian uses in searching, we extend the vocabulary to the conceptual model of the given domain, i.e. an ontology. Therefore, we consider ontology-based searching for information [2].

In this paper we focus more on the problem of determining the ambiguity in a user' request and providing corresponding refinements. We define the concept ambiguity, provide a categorization of the ambiguities of users queries and develop methods for measuring and resolving these ambiguities. The approach is based on the analysis of the structure of the underlying vocabulary (ontology) and the content of the information repository. For the analysis of the content of an information repository we use the formal concept analysis (FCA) [3], a mathematic theory about order sets. FCA clusters given data very efficiently and the analysis of the concept lattice formed around a query, discovers many interesting properties of that query, including its ambiguity and possible refinements. Moreover, the additional source for the query's ambiguity resolution (i.e. query refinement) is the query-transaction-log, which is analysed in order to discover frequently used query sequences (so called query patterns).

The paper is organised as follows: In the second section we give the basic terminology we use in this paper, whereas the section 3 presents our Librarian Agent scenario. In the section 4 we give a motivating example for the proposed approach, which is described in the section 5. In the section 6 we discuss some of the related works, whereas section 7 contains conclusion remarks.

2 Background

In this section we give the basic terminology we use in this paper. The presentation is arranged as a sequence of definitions so that the reader, if necessary, can come back to this section in order to clarify statements in the subsequent sections of the paper.

Let $\Re$ and $\aleph$ denote real and natural numbers, respectively. All subscripts are in $\aleph$, unless otherwise specified. If S is a set, 2^S denotes its power set, i.e., the set of all subsets of S, and $|S|$ denotes its cardinality.

Definition 1: Ontology
A core ontology is a structure $O := (C, \leq_c, R, \sigma, \leq_r)$ consisting of

- two disjoint sets C and R whose elements are called concept identifiers and relation identifiers, respectively,
- a partial order $\leq_c$ on C, called concept hierarchy or taxonomy
- a function $\sigma : R \rightarrow C^+$, called signature
- a partial order $\leq_r$ on R called relation hierarchy, where $r_1 \leq_r r_2$ implies $|\sigma(r_1)| = |\sigma(r_2)|$ and $\pi_i(\sigma(r_i)) \leq_c \pi_i(\sigma(r_i))$ for each $1 \leq i \leq |\sigma(r_i)|$

Often we call concept identifiers and relation identifiers, respectively, for the sake of simplicity.

Definition 2: Domain and Range
For a relation $r \in R$ with $|\sigma(r)| = 2$, we define its domain and its range by
$dom(r) := \pi_1(\sigma(r))$ and $range(r) := \pi_2(\sigma(r))$.

Definition 3: Axioms
Let *L* be a logical language. A *L*-axiom system for an ontology $O := (C, \leq_c, R, \sigma, \leq_r)$ is a pair $A := (AI, \alpha)$, where

- *AI* is a set whose elements are called axiom identifiers and
- $\alpha : AI \rightarrow L$ is a mapping.
  The elements of $A := \alpha(AI)$ are called axioms.

An ontology with *L*-axioms is a pair *(O, A)*, where *O* is an ontology and *A* is an *L*-axiom system for *O*.

In the FOL notation axioms (or rules) are written as " $p \leftarrow q_1,...,q_n$.". This is read declaratively as q_1 and … and q_n implies *p*. Each of p (the head) and the q_i's (the symbols of the body) are atomic formula (also referred to as literals), consisting of a predicate (relation) applied to terms, which are either constant or variables. A literal is an atom or a negated atom. In our research, we use F-Logic [4] as the logical language.

Definition 4: Relation properties
We define three properties of the relations: reflexivity, symmetry and transitivity, which are defined by functions r_r, r_s, r_t respectively, in the following manner: $r_t : R \rightarrow R_t$.

Definition 5: Knowledge Base
A Knowledge Base is a structure $KB := (C_{KB}, R_{KB}, I, l_c, l_r)$ consisting of

- two sets C_{KB} and R_{KB}
- a set *I* whose elements are called instance identifiers (or instances or objects in brief)
- a function $l_C : C_{KB} \rightarrow I$ called concept instantination
- a function $l_r : R_{KB} \rightarrow I^+$ called relation instantination

A relation instance can be depicted as $r(I_1, I_2,..., I_n)$, where $r \in R_{KB}, I_i \in I$. *r* is called predicate and I_i is a term.

Definition 6: (ontology-based) Query
A (conjunctive) query is of the form or can be rewritten into the form:

$$\forall \overline{X} \quad \overline{P}(\overline{X}, \overline{k}) \ \wedge not(\overline{N}(\overline{X}, \overline{k})) \quad ,$$

with $\overline{X}$ being a vector of variables $(X_1,...,X_n)$, $\overline{k}$ being a vector of constants (concept instances), $\overline{P}$ being a vector of conjoined predicates (relations) and $\overline{N}$ a vector of disjoined predicates (relations).

For example, for the query "*forall x worksIn(x, KM) and researchIn(x, KMsystems)*" we have $\overline{X} := (x)$, $k := (KM, KMsysteme)$, $\overline{P} := (P_1, P_2)$,
$P_1(a,b,c) := worksIn(a,b)$, $P_2(a,b,c) := researchIn(a,c)$.

Each part of a query that can be treated as a regular ontology-based query is called *sub-query* of that query. A query can be viewed as an axiom without a head.
Note: in this paper we consider only conjunctive queries. Since a disjunctive query can be represented as a disjunction of several conjunctive queries our approach can be very easily extended to the disjunctive queries.

3 The Librarian Agent

The role of the Librarian Agent is (i) to support the disambiguation of the queries posted by users (query management) and (ii) to enable the changes in the knowledge repository regarding the users' information needs (collection management).

Fig.1. sketches the application scenario for the Librarian Agent. A user posts the query (cf. 1 in the Fig.1.) which is first processed by the Librarian Agent. The Agent measures the ambiguity of the query (cf. 2 in the Fig.1.) by considering the capacity of the information repository (knowledge base) and the domain vocabulary - ontology. The user is provided by the explanation what is ambiguous in the query and how this ambiguity can influence the result of the querying. Moreover, the agent recommends the user some changes (refinements) in the query (cf. 3 in the Fig.1.). The Agent receives the feedback information about how many (and which) refinements' steps a user performed (cf. 4, 5 in the Figure 1), and it uses this information to refine its own strategies for creating recommendations.

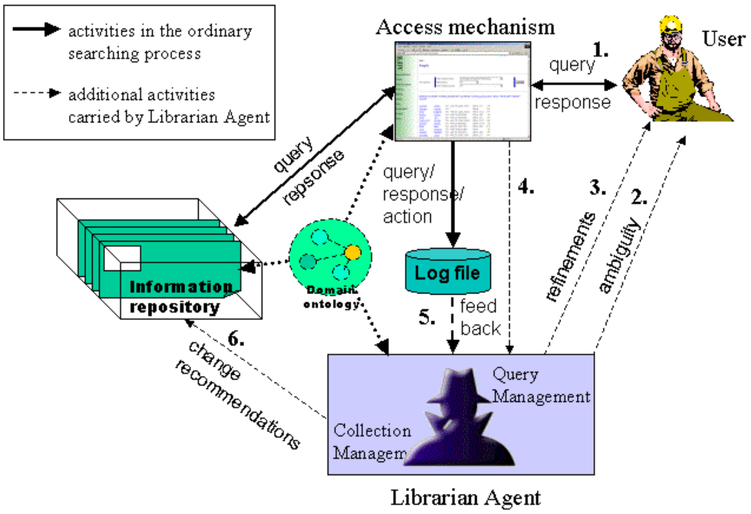

Fig. 1. The roles of the Librarian Agent in the process of searching for knowledge

As the result of the querying, the access mechanism retrieves a list of resources, which is analyzed (cf. 5 in the Fig.1.) by the Librarian Agent in order to make recommendations for the changes in the collection (cf. 6 in the Fig.1.). This recommendation takes into account the analysis of the queries posted by users and the used vocabulary, as well. In order to avoid disturbing the users by additional questioning, the feedback information is collected implicitly by analyzing the activities of the users captured in the log file of the system (cf. 5 in the Fig.1.).

The conceptual model of the given domain – the domain ontology, supports the processing of each step in this approach.

In the rest of this paper we focus on the Query Management component of the Librarian Agent. More information about Collection Management can be found in the [5].

4 Motivating Example

In order to make the description of our approach more understandable we give here an example of the problems which can occur in the ontology-based searching.

Let us consider the following example related to the activities of an institute, referred as the *Research ontology* in the rest of the text:

knowledge base:

`Researcher(rst)`[1]
`Researcher(nst)`
`Researcher(ysu)`
`Researcher(jan)`
`Researcher(meh)`
`isA(PhD_student, Researcher)`[2]
`isA(Professor, Researcher),`
`PhD_student(nst)`
`PhD_student(ysu)`
`Professor(rst)`
`Professor(jan)`
	`workIn(rst, KM)`[3]
	`workIn(nst, KM)`
	`workIn(ysu, KM)`
	`workIn(jan, KM)`
	`workIn(meh, KM)`
	`research_group(KM)`
	`researchArea(KMsystem)`
	`researchArea(CBR)`
	`group_Area(KM, KMsystem)`[4]
	`researchIn(rst, CBR)`[5]
	`lecture(Ontology)`
		`teaches(rst, Ontology)`[6]
		`teaches(nst, Ontology)`
		`teaches(ysu, Ontology)`
		`teaches(jan, Ontology)`
		`teaches(meh, Ontology)`
		`teaches(meh, DM)`
		`project(OntoWeb)`
		`head(rst,OntoWeb)`
		`manages(ysu,OntoWeb)`
		`participate(meh,OntoWeb)`

axioms:

$$\forall\ x,\ y,\ z\ \mathtt{researchIn(x,z)} \leftarrow \mathtt{workIn(x,y)} \wedge \mathtt{group\_Area(y,z)} \qquad (1)$$

$$\forall\ x,\ y\ \mathtt{Professor(x)} \leftarrow \mathtt{head(x,y)} \wedge \mathtt{project(y)} \qquad (2)$$

Let analyse the following cases of querying this repository:

1. Case

Suppose that a user posts the following query:

$$\forall\ x \leftarrow \mathtt{worksIn(x,\ KM)} \wedge\ \mathtt{researchIn(x,\ KMsystems)},$$

which searches for all researchers who work in the group `KM` and who research in the `KMsystems`.

According to the axiom (1) and the fact `group_Area(KM, KMsystem)` it follows that all researchers who work in the group `KM`, research in the `KMsystems` as well. It means

[1] it means that rst is a Researcher

[2] it means that `PhD_student` is a subtype `Researcher`

[3] it means that `rst` works in the group `KM`

[4] it means that `KM` group is focused on research in `KMsystems`

[5] it means that `rst` researches in the `CBR`

[6] it means that `rst` teaches course about `Ontology`

that the constraints in the query are redundant (`worksIn(x, KM)` and `researchIn(x, KMsystems)`) and the user should be informed that one of constraints provides no effects on the searching. This information should help the user to understand how his information need is interpreted in the query-system and probably he will refine the query. Moreover, the system should analyse the ontology and the information repository in order to recommend some refinements. For example, the user might be interested in researchers who research in a sub-area of `KM` and not in the whole `KM` .

2. Case

Suppose that a user makes the query:

$\forall$ `x` $\leftarrow$ `worksIn(x, KM)` $\wedge$ `not researchIn(x, KMsystems).`

Similar to the previous example, this query will be expanded with the information `researchIn(x, KMsystems)`, i.e. that researcher x researches in the area `KMsystems`. However, this information is contradictory with a constraint posted originally by the user. The user should be informed that such a query will never return a result, i.e. that a researcher, who works in the group `KM` and does not research in the `KMsystems` cannot exist. The system should recommend a suitable refinement automatically (by analysing ontology and content).

3. Case

Suppose the query:

$\forall$ `x` $\leftarrow$ `researcher(x)` $\wedge$ `head(x, Ontoweb).`

According to the axiom (2) only a `Professor` can be the `head` of a `project`. It means that the query will consider only a special sub-type of `researchers` (i.e. `Professor`) and the user should be informed about this new constraints. Probably, the predicate `head` has not been properly used regarding the user information need - his information need can be to find all `researcher` who manage `project OntoWeb`. The system should discover this potential refinement automatically and suggest the user such query modification (predicate `head` -> `manages`), which does not implies new constraints in the query.

4. Case

Suppose the following query:

$\forall$ `x` $\leftarrow$ `worksIn(X,KM)` $\wedge$ `teaches(X, Ontology).`

Although there are no additional constraints implied by the ontological structure (i.e. rules), the system should discover that in the underlying information repository, all `researchers` who work in the `KM group` are included in the `Ontology lecture`. Therefore, the constraints posted by the user are redundant and he should be informed and guided how to resolve that problem, similarly as in the case 1.

It is clear that the resolving of these situations requires a careful analyse of the underlying ontology and knowledge base regarding the user's query in order to suggest the most suitable refinements of that query. In the next section we present an approach for such a query management.

5 Query Management Component of the Librarian Agent

5.1 The Ambiguity of a Query

The goal of the query management system is to discover ambiguities in queries and to recommend corresponding refinements. We treat a query as an ambiguous one, when the user's information need cannot be uniquely determined from that query[7]. Practically, it means that the same list of results can be obtained for at least one query, which is different from the given one.

The users often estimate the ambiguity of a query through the number of results: a lot of results can be an indicator that there are some irrelevant results, i.e. that some other information needs are covered by that query. In most of the existing IR system, the user gets only this information, the number of results, as the characterisation of the ambiguity. However, the ambiguity of a query is a more complex category and it requires handling by using a more formalised approach.

We have found two main factors, which affect an ambiguity of a query:

1. used vocabulary (i.e. ontology)

(i) e.g. the query for all researchers has a different ambiguity in two vocabularies in which the concept Researcher is modelled through, for example, five and ten subconcepts, respectively (this is called *Clarity*), or

(ii) e.g. the query for "researcher and project" has different ambiguity in a vocabulary where just one relation between the concepts Researcher and Project exists (e.g. participates), than in the vocabulary where two relations between these concepts exist (e.g. participates and headOf) (it is called *Context Clarity*) and

2. the information repository

e.g. the query for a Professor in a repository related to an academic institution will result in much more results than the same query in a repository related to an industry organisation;

Consequently, in order to handle these situations we define two types of the ambiguity of a query:

- the *semantic ambiguity*, as the characteristic of the used vocabulary (i.e. how many interpretations can be assigned to the given query);

- the *content-related ambiguity*, as the characteristics of the information repository (i.e. how a query is related to its "neighbour's" queries).

In the next two subsections we elaborate more on these two types of ambiguities.

The Semantic Ambiguity

A term from the query can be interpreted in various ways depending on the relation between that term and other terms from the vocabulary. For example, the term `Researcher` from the *Researcher ontology* in motivating example, can be interpreted as `Professor` or `PhD_Student`. The possibility to measure this ambiguity increases with the representation power of the vocabulary. In the case of an ontology-based query we define following four types of the *semantic ambiguity*, based on the definition of an ontology given in the section 2:

[7] This statement reflects the distinction between the user's unarticulated information need and the query which results from that [6]. One of the common pitfalls in modern IR approaches is the ignorance of this distinction.

1. Incompleteness

The query contains incomplete information about the used concepts/relations. It means that the query can be *automatically* expanded in order to clear the meaning of the query. For the *Research ontology* the query "`forall x,y,z researcher(x) and project(y) and lecture(z) and head(x,y)`" is incomplete because there is a relation between concept `researcher` and `lecture`, namely `teaches`, which can be used to specify the query more precisely.

Regarding the definition of a query given in the section 2, we give the following formalisation of the incompleteness of a query Q which contains subquery $p_1(\overline{X},k) \wedge p_2(\overline{X},k)$:

$$\forall p_1, p_2 \in \overline{P}, y, z \in \overline{X}, C_1, C_2 \in \overline{C}, \exists \; r_1, r_2 \in R \wedge (p_1(\overline{X},k) = l_c(C_1, y)) \wedge$$
$$(p_2(\overline{X},k) = l_c(C_2, z)) \wedge (dom(r_1) = C_1) \wedge range(r_1) = C_2 \wedge \neg\exists p_x \in \overline{P} \big| \; (p_x = r_1)$$
$$\rightarrow the \quad query \quad Q \quad is \quad incomplete$$

2. Clarity

A query is incompact or redundant if it contains more terms than are needed and desired to express the same "idea". Concept hierarchy and property hierarchy from the domain ontology are used to check this criterion. The important source of the unclarity are the transitive relation, in general. For example, for the query "`forall x researcher(x) and PhD_Student(x)`" is not clear whether the user is interested in all researchers or only in PhD students. Another example is the case 3 in motivating example.

The formalisation of a query Q which contains predicates $p_1(\overline{X},k) \wedge p_2(\overline{X},k)$:

$$\forall p_1, p_2 \in \overline{P}, y \in \overline{X}, C_1, C_2 \in \overline{C}, \exists \; r_1, r_2 \in R \wedge (p_1(\overline{X},k) = l_c(C_1, y)) \wedge$$
$$(p_2(\overline{X},k) = l_c(C_2, z)) \wedge (C_1 \leq c C_2) \rightarrow the \quad query \quad Q \quad is \quad unclear$$

or in general case for any transitive relation:

$$\forall p_1, p_2 \in \overline{P}, y \in \overline{X}, C_1, C_2 \in \overline{C}, \exists \; r_1 \in R_t \wedge i_1, i_2 \in I \wedge p_1(\overline{X},k) = r_1(y, i_1) \wedge$$
$$p_2(\overline{X},k) = r_1(y, i_2) \wedge r_1(y, i_1) \leq_t r_1(y, i_2) \rightarrow the \quad query \quad Q \quad is \quad unclear$$

3. Redundancy

The query is redundant when it contains the same constraints. We define two types of the redundancy:

1) redundant constraints in the query (case 1 in the motivating example)

2) redundancy in the information repository (case 4 in the motivating example)

The formalisation of a query Q which contains sub-query $p_1(\overline{X},k) \wedge p_{21}(\overline{X},k) \wedge ... \wedge p_{2f}(\overline{X},k)$:

$$\forall p_1, p_{21}, ..., p_{2f} \in \overline{P}, y \in \overline{X}, C_1, C_2 \in \overline{C}, \exists \; r_1 \in R \wedge i_1, i_2 \in I \wedge p_1(\overline{X},k) = r_1(y, i_1)$$
$$\wedge \; p_{21}(\overline{X},k) = r_{21}(y, i_2) \wedge ... \wedge p_{2f}(\overline{X},k) = r_{2f}(y, i_2) \wedge derive(r_1, r_{21}, ..., r_{2f})$$
$$\rightarrow the \quad query \quad Q \quad is \quad redundant$$

Predicate *derive(a, b,..., l)* depicts the existence of the axiom $a \leftarrow b \wedge ... \wedge l$.

4. Un-satisfiable query

When a query produces no results we treat it as unsatisfable. Similar to the redundancy we define two types of this criterion:

1) contradictory constraints in the query (example 2 in the motivating example)

2) gap in the information repository

The formalisation of the first type for a query Q which contains predicates $p_1(\overline{X},k) \wedge p_{21}(\overline{X},k) \wedge ... \wedge p_{2f}(\overline{X},k)$:

$$\forall p_1 \in \overline{N}, p_{21},..., p_{2f} \in \overline{P},, y \in \overline{X}, \exists \; r_1 \in R \wedge i_1, i_2 \in I \wedge p_1(\overline{X},k) = r_1(y,i_1) \wedge$$

$$\wedge\, p_{21}(\overline{X},k) = r_{21}(y,i_2) \wedge ... \wedge p_{2f}(\overline{X},k) = r_{2f}(y,i_2) \wedge derive(r_1, r_{21},..., r_{2f})$$

$$\rightarrow the \;\; query \;\; Q \;\; is \;\; contradictor$$

In [5] we have defined measures for estimating first two types the semantic ambiguity of an ontology-based query: *ContextClarity* and *Clarity*, respectively. Very briefly, *ContextClarity* is inversely proportional to the number of relations between two concepts, whereas, the *Clarity* is inversely proportional to the number of subconcepts of a concept. Due to lack of space we omit here the formal definitions which can be found in the original paper.

The Content-Related Ambiguity

The *content-related ambiguity* of a query depends on the capacity of the information repository (i.e. which resources are stored in the repository). Since this capacity determines the list of results of a query, the content-related ambiguity of a query can be defined by comparing the results of the given query with the results of another queries. In the rest of this subsection, we define several relations between queries in order to estimate this type of the ambiguity of a query.

Let $Q = (M, O)$ be the query-answering pair, whereas *M* is an ontology-based query (i.e. metadata) and *O* is the list of results (i.e. objects) on the query *Q*. *M* and *O* are called query_terms and query_objects respectively.

Definition 7: Structural equivalence (=) by:

$(M_1, O_1) = (M_2, O_2) \leftrightarrow O_1 = O_2$, which can be written as $Q_1 = Q_2 \leftrightarrow O_1 = O_2$

Two queries are structurally equivalent if their result sets are the same.

Definition 8: Structural subsumption (parent-child) (<) by:

$(M_1, O_1) < (M_2, O_2) \leftrightarrow O_1 \subset O_2$.

A query-answering pair (M_2, O_2) subsumes another pair (M_1, O_1) if the result set of the second query-answering pair subsumes the results of the first one. For a query-answering pair Q_1 we define two subsumption relations:

– direct parent ($<_{dir}$): If $Q_1 < Q_2 \wedge \neg\exists Q_i, Q_1 < Q_i < Q_2$, Q_2 is direct_parent of the Q_1 ;

– direct child ($>_{dir}$): If $Q_2 < Q_1 \wedge \neg\exists Q_i, Q_2 < Q_i < Q_1$, Q_2 is direct_child of the Q_1 .

For a query Q_a we define five properties which characterise its structural ambiguity: *Largest equivalent query, Smallest equivalent query Uniquness*, *Covering* and *CoveringTerms.*

The largest equivalent query for the query Q_a is its equivalent query with the maximal query_terms. It is calculated in the following way: $Q_{a\max} = (\bigcup_{Q_i <_{dir} Q_a} M_i, O_a)$. It means that the largest equivalent query contains the union of query_term of all direct_child.

The smallest equivalent query for the query Q_a is its equivalent query with minimal query_terms. There can be several such queries. They are calculated in the following way: $Q_{a\min} \in \{(\times(M_i \cap M_a), O_a) | \ Q_a <_{dir} Q_i, i = 1,..n\}$

For a query Q_a it is possible to define a subset of objects, which are unique for that query, i.e. they cannot be obtained for any direct_child query. We call that the *Uniqueness* of the query and it is calculated in the following way.

$Uniquness(Q_a) = \{O_a / \{\cup O_i\} | Q_i <_{dir} Q_a, i = 1..n\}$

Covering and *CoveringTerms* are measures which define the percent of identical answers and query_terms, respectively, in two queries. More formally, for two queries Q_a and Q_b we define:

$Covering(Q_a, Q_b) = |O_a \cap O_b| / \max\{|O_a|, |O_b|\}$

$CoveringTerms(Q_a, Q_b) = |M_a \cap M_b| / \max\{|M_a|, |M_b|\}$

It is clear that the calculation of the above mentioned parameters could be time-consuming. In order to make this calculation more effective we use formal concept analysis (FCA) for organising data in the so-called concept lattices, which correspond to the multi-inheritance hierarchical clusters. Each of these clusters can be considered as a query posted to the repository and consequently, the lattice represents the query clustering. By analysing such lattices many interesting relation between queries can be discovered and used for measuring query ambiguity and/or for the query refinement.

Due to the lack of space we omit here the detailed introduction of the FCA which can be found in [3]. We mention only the main concepts needed for the understanding of our approach.

Formal Concept Analysis (FCA) is a technique derived from the lattice theory that has been successfully used for various analysis purposes. Organization over the data is achieved via a mathematical entity called a formal context. A formal context is a triple (G, M, I) where G is a set of objects, M is a set of attributes, and I is a binary relation between the objects and the attributes. A **formal concept** of a formal context (G, M, I) is a pair (A, B) where $A \subseteq G$, $B \subseteq M$, A = B'= $\{g \in G \mid \forall m \in B: (g,m) \in I\}$ and B= A' = $\{m \in M \mid \forall g \in A: (g,m) \in I\}$. For a formal concept (A, B), A is called extent and is the set of all objects that have all the attributes defined in B. Similarly, B is called the intent and is the set of all attributes possessed in common by all the objects in A. As the number of attributes in B increases, the concept becomes more specific, i.e. a specialization ordering is defined over the concepts of a formal context by: $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow B_2 \subseteq B_1$.

In this representation more specific concepts have larger intents and are considered "less than" (<) concepts with smaller intents. The same partial ordering is achieved by considering extents, in which case more specific concepts have smaller extents. The partial ordering over concepts is always a lattice.

In the Fig.2. we present an example of the concept lattice generated (partially) for the example given in the section 2.

Table 1. A part of the *Research ontology*

Attributes / **Objects**	Researcher	Prof.	workIn ->>KM	researchIn ->>CBR	researchIn- >>KMsystems	teaches- >>Ontology
rst	x	x	x	x	x	x
nst	x		x	x	x	x
ysu	x		x	x	x	x
jan	x	x	x		x	x
meh	x		x		x	x

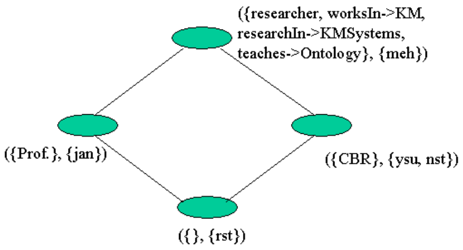
({researcher, worksIn->KM, researchIn->KMSystems, teaches->Ontology}, {meh})
({Prof.}, {jan})
({CBR}, {ysu, nst})
({}, {rst})

Fig. 2. Example showing the process of generating a derived concept lattice from a set of data given in the table 1. The concepts represented in the lattice should be read as in the following example: foremost left formal concept, (`{Prof.}, {jan}`), corresponds to the objects (`jan, rst`) and attributes (`Researcher, Prof., workIn->>KM, researchIn->>KMSystems, teaches->>Ontology`)

Such a representation enables very intuitive interpretation of a query: one can see a formal concept as a representation of a query state, where the intent of the concept represents the query itself and the extent represents all resources that match the query. For example, the query "`forall x Prof(x) and researchIn(x, KMSystem)`" will be mapped onto the formal concept (`{Prof.}, {jan}`) (note that a concept encompasses all objects from its subconcepts).
Such an ordering in the query space enables very easily interpretation of query results regarding their ambiguity. For example, in the case 4 in the motivating example, the query "`forall x worksIn(X,KM) and teaches(X, Ontology)`" is mapped onto the formal concept (`{Researcher, worksIn->KM, researchIn->KMSystems, teaches->Ontology}, {meh}`). By analysing the properties of this formal concept we reason about the ambiguities of the query:

1) the query contains redundant constraints: all `Researcher` which work in `KM group`, teach the `Ontology lecture`
2) this query is semantically equivalent to the queries

```
forall x Researcher(x)
forall x researchIn(x, KMSystems)
forall x teaches(x, Ontology)
```

In other words, by reading the attributes of the formal concept which corresponds to the query, we can estimate the proposed ambiguity parameters directly. The formal expression of the query's ambiguities using FCA is out of the scope of this paper. In the next subsection we give more details about the possibilities to use FCA as a query refinement mechanism.

5.2 Refinement

Our approach for query refinement tries to reflect the refinement model which a human librarian (or shop assistant) uses in his daily work. It means that we use three sources of information in suggesting query refinement:

1. the structure of the underlying ontology (vocabulary)
2. the content of the knowledge repository
3. the users' behaviour (how users refine their queries on their own)

Since all of our ambiguity measures are formally defined, it is very easily to determine which part of the given query is ambiguous and to focus user's attention on these "ambiguous" constraints. For example, by using the definition of the *incompleteness* in the section 4.1.1, in the case of such an ambiguity in his query, the user should be provided with the recommendation to add a relation (r_1) in the query, in order to clarify the meaning of the query.

Moreover, using the quantification of the ambiguities we can determine which of the ambiguities should be resolved firstly. For example, for the case 1 from the motivating example, `forall x worksIn(x, KM) and researchIn(x, KMsystems)`, the system suggests the refinement of the second predicate (`researchIn(x, KMsystems)`) because its clarity according to the *Clarity* factor, presented in the previous subsection, is greater than the clarity of the first one. Moreover, the approach uses the FCA to determine the possible refinements with respect to the content of the knowledge repository. For example, by considering concept lattice presented in Figure 3 the system will suggest refinement `researchIn(x, CBR)` for the case 1 in the motivating example.

Hence, our approach tries to combine semantics of the domain model (1. source) with the capacity of the information repository (2. source) in order to provide recommendations for the most relevant refinements, by assuming that the user will refine its query linearly (not to change the whole query).

The third source for making the query's refinement recommendation (i.e. user's behaviour) requires an analysis of the users activities in an ontology-based application. In [7] we presented a framework for capturing user's activities in a semantic query log file. This query log is "mined" in order to discovery so called query patterns, which describes regularities in the behaviour of the users during searching a portal. That analysis is out of the scope of this paper.

Discussion

In the following we present briefly how the concept lattice[8] can be used in supporting these types of the refinement. Due to lack of space the approach is described quite informally.

The first interesting feature that can be found in the lattice structure is that adding additional attributes into a query will always move the query state to some formal concept below the current state, which maps mathematically to the definition of the

[8] we consider that concept lattice represents all implicit information which can be derived from the domain axioms

sub-concept relation. To find all query states that model such extensions of the current query we have to find all subconcepts of the current position.

By comparing the intents of the concepts along the edges we can map the attributes that can be added to the new position in the lattice. In the example given in Figure 3 the possible refinements for the query `forall x researchIn(x, CBR)` will be adding the constraint "`Prof.`", pointing to the concept with objects `rst` in the extent. From the concept lattice it is clear that all other refinements, e.g. "`workIn->>KM`", "`Researcher`" etc will not change the list of results, since all these attributes are related to the concepts above the concept (`{CBR},{rst, ysu, nst}`).

The same discussion can be applied for the deleting attributes from a query, which will always move the query state to some concept above the current state, which maps mathematically to the definition of the superconcept relation. To find all query states that model such reduction of the current query we have to find all superconcepts of the current position. For example, for the query `forall x worksIn(x, KM) and researcher(x) and researchIn(x, CBR)`, which corresponds to the concept (`{CBR},{rst, ysu, nst}`), the reduction of the constraints "`Researcher(x)`" or "`worksIn(x, KM)`" will not provide any new results. From the structure of the concept lattice it is clear that only the reduction of the constraint "`researchIn(x, CBR)`" effects the query results.

However, the concept lattice does not provide explicit specification of the generalisation and specialisation of an attribute, because all concepts which are below a concept in the lattice have more attributes than that concept. The trivial case is when a concept X has only one parent concept in the lattice: then one of the attributes associated to the parent concept is the "real" generalisation of the "new" attribute in the concept X. For example, the concept (`{Prof.}, {rst, sst}`) has only one parent which contains the generalisation of the attribute "`Prof.`" (attribute "`Researcher`"). For the case of more parent concept, the semantic of the attributes has to be processed.

The next case is the query that contains some contradiction, i.e. query for something that does not exist in the data set. Such a query will point to a bottom element with an empty extent and we can avoid offering these refinements easily. Moreover, by using the properties of the concept lattice, it is possible to determine ideal position of an *unsatisfiable* query in the lattice and by moving along the edges in order to determine:

1) which constraints should be relaxed from the original query in order to get results and

2) what is the first approximation of the result of that unsatisfiable query

Such a query relaxation can be very useful mechanism to cope with the uncertainties in the queries or gap in the knowledge repository. For example, in a skill management application an unsatisfiable query for an expert in several research areas, will result in the nearest real neighbour of that ideal request.

Another useful feature when finding query refinements this way is that once we found the states that are possible refinements we have the size of the new result set which is the size of the extent of the concept that will be reached. Using this feature of the lattice we can give the user the information of the size of the next result set before she chooses a refinement, a front-end can allow sorting or filtering the refinements by this size.

6 Related Work

Query Ambiguity
The determination of an ambiguity in a query, as well as the sources of such an ambiguity is the prerequest for the efficient searching for information. Although some recent work is done in quantifying the query ambiguity based on the language model of the knowledge repository [8], [9], the IR research community did not explore the problem of using rich domain model in modelling the querying. Some very important results in the query analysis can be found in the deductive database community [10], namely semantic query optimisation. That approach, although revolutionary for using domain knowledge for the optimal compilation of the queries, does not consider the ambiguity of the query regarding user's information need at all.

Query Refinement
There is a lot of research devoted to the query refinement in the Web IR community. In general, we see two directions of the modifying queries or query results to the need of a users: query expansion and recommendation systems respectively. *The query expansion* is aimed at helping the users make a better query, i.e. it attempts to improve retrieval effectiveness by replacing or adding extra terms to an initial query. The i*nteractive query expansion* supports such an expansion task by suggesting candidate expansion terms to users, based usually on hyper-index [11] or concept-hierarchies [12] automatically constructed from the document repository. In [13] the model of the query-document space is used for the interactive query expansion. *Recommendation systems* [14] try to recommend items similar to those a given user has liked in the past (content-based recommendation) or try to identify users whose tastes are similar to those of the given user, and recommend items they have liked (collaborative recommendation). Personalised web agents, e.g. WebWatcher [15] track the users browsing and formulate user profiles which are used in suggesting which links are worth following from the current web page.

However, none of these approaches uses rich domain model for the refinement of a query, i.e. the reasons for doing a refinement are not based on the deeply understanding of the structure of a query or the deeply exploring of the interrelationships in the information repository. Moreover, none of them tries to determine (measure) the ambiguity in a query and to suggest a refinement which will decrease such an ambiguity.

Formal Concept Analaysis
In [16], the authors described an approach, named REFINER, to combine Boolean information retrieval and content-based navigation with concept lattices. For a Boolean query REFINER builds and displays a portion of the concept lattice associated with the documents being searched centred around the user's query. The cluster network displayed by the system shows the result of the query along with a set of minimal query refinements/enlargements. A similar approach is proposed in [17], by adding the size of the query result as an additional factor of the navigation. Moreover, the distance between queries in the lattice is used for similarity ranking.

The main drawback in these approaches is that both start from the characteristics of the concept lattice and try to map these characteristics into the query refinement process. In that way, practically, they constrain the queries which are analysed to the set of queries which correspond to the formal concepts in a formal context (i.e. a query is treated only when it is the largest query of an other queries in the repository). Thus, they do not model the query refinement process and miss some very important properties for the refinement (such as a set of equivalent queries, max_Equality, min_Equality, uniqueness). Another problem is that they do not consider the partial ordering of the metadata into a vocabulary, losing the possibilities to find a real specialisation/generalisation of a query, which is one of the very frequently used refinements. Moreover, the ambiguity of the query, as the crucial reason for the query refinement process, is not treated at all.

7 Conclusion

In this paper, we presented an approach for the query management in ontology-based IR systems. The system treats a library scenario in which users query the repository for knowledge resources. Consequently, the so-called Librarian Agent plays the role of the human librarian in the traditional library – it uses all possible information, about the domain vocabulary, the behaviour of previous users and the capacity of the knowledge repository, in order to help users to find the resources they are interested in. Based on various analyses, the agent, through an interactive dialogue, guides the users in more efficient searching for information. Particularly, for a query given by a user, the agent measures its ambiguity, regarding the underlying vocabulary (i.e. ontology), as well as the content (capacity) of the information repository. In case of high ambiguity, the agent suggests the user the most effective reformulation of the query. The recommendation can be based on the query-patterns, found by analysing users' past behaviour in the given application.

We find that our approach represents a very important step in simulating the brick-and-mortar environment and benefiting from applying the practical results obtained in that area in the searching for information in the virtual world. Moreover, this approach leads to the self-adaptive knowledge portals, which can discover some changes from the user's interactions with the system automatically and evolve their structure correspondingly.

One of the very important benefits we did not elaborate in this paper is the optimisation of an ontology-based query using presented approach. Indeed, a query can be translated in several semantically equivalent forms and one with the minimal cost will be evaluated. It will be the part of our future works.

Acknowledgement. The research presented in this paper would not have been possible without our colleagues and students at the Institute AIFB, University of Karlsruhe. Research for this paper was partially financed by BMBF in the project "SemiPort" (08C5939).

References

1. Stojanovic, N.: On the role of a Librarian Agent in Ontology-based Knowledge Management Systems, Workshop Ontologiebasiertes Wissensmanagement, 2. Konferenz Professionalles Wissensmanagement, Luzern, April 2003

2. N. Guarino, C. Masolo, and G. Vetere, "OntoSeek: Content-Based Access to the Web", *IEEE Intelligent Systems*, 14(3), pp. 70–80, (May 1999).
3. Ganter, B and Wille, R: *Formal Concept Analysis: Mathematical Foundations*. Springer Verlag, 1999.
4. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages, Journal ACM, 42:741–843, 1995
5. Stojanovic, N., Stojanovic, L: Usage-oriented Evolution of Ontology-based Knowledge Management Systems, *Proceedings of the 1st Int'l Conf. on Ontologies, Databases and Application of Semantics (ODBASE-2002*), Irvine, CA, 2002.
6. Saracevic, T.: Relevance: A Review of and a framework for the thinking on the notion in information science. Journal of the American Society for Information Science, 26, (6), 321–343, 1975
7. Stojanovic, N., Stojanovic, L., Gonzalez, J: More efficient searching in a knowledge portal – an approach based on the analysis of users' queries, Fourth International Conference on Practical Aspects of Knowledge Management (PAKM 2002), Vienna, 2002, Springer LCNS/LNAI
8. Ponte, J. and Croft, W.B.: A language modeling approach to information retrieval, Proceedings of ACM SIGIR'98, 275–28, 1998.
9. Cronen-Townsend, S. and Croft, W.B.: Quantifying Query Ambiguity, in the Proceedings of HLT 2002,pp. 94–98.
10. Chakravarthy, U., Grant, J. and Minker, J: *Logic-based approach to semantic query optimization*. ACM Transactions on Database Systems, 15(2):162–207, 1990.
11. Bruza PD and Dennis S. Query Reformulation on the Internet: Empirical Data and the Hyperindex Search Engine. In: Proceedings of RIAO97, Computer-Assisted Information Searching on Internet, Montreal, June 1997
12. Joho, H., Coverson, C., Sanderson, M., Beaulieu, M.: Hierarchical presentation of expansion terms, ACM SAC, 2002
13. Wen, J.-R., Nie, J.-Y. and Zhang, H.-J. Clustering User Queries of a Search Engine. WWW10, May 1–5, 2001, Hong Kong.
14. Balabanovic, M., Shoham, Y.: Content-Based, Collaborative Recommendation. CACM 40 (3): 66–72 (1997)
15. Joachims, T.; Freitag, D.; and Mitchell, T. 1997. Webwatcher: A tour guide for the World Wide Web. In Proc. IJCAI-97.
16. Carpineto, C., Romano, G. Effective re formulation of boolean queries with concept lattices. *In Flexible Query Answering Systems FQAS'98*, pp. 277{291, Berlin Heidelberg, Springer-Verlag, 1998.
17. Becker, P., Eklund, P., Prospects for Document Retrieval using Formal Concept Analysis, *Proceedings of the Sixth Australasian Document Computing Symposium*, Coffs Harbour, Australia, December 7, 2001.