Journal of Universal Computer Science, vol. 9, no. 7 (2003), 697-718
submitted: 3/2/03, accepted: 21/7/03, appeared: 28/7/03 © J.UCS

On the Role of the Librarian Agent in Ontology-based Knowledge Management Systems

Nenad Stojanovic
(Institute AIFB, University of Karlsruhe, Germany
nst@aifb.uni-karlsruhe.de)

Abstract: In this paper, we present an ontology-based approach for the improvement of searching in an information portal. The approach is based on incremental refinement of user's queries, according to the ambiguity of a query's interpretation. The so-called Librarian Agent plays the role of the human librarian in the traditional library – it uses information, about the domain vocabulary, the capacity of the knowledge repository and the behaviour of previous users in order to help users find the resources they are interested in. Moreover, 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 topics should be introduced or which topics users are no more interested in. We partially implemented the approach in the Web Portal of our Institute and some initial evaluation results are shown.

Key Words: Information Retrieval, Ontology, Query Refinement
Categories: H.3 Information Storage and Retrieval H.3.3 Information Search and Retrieval

1 Introduction

The efficiency of the searching for information in an information portal highly depends on the knowledge of a user about the content of that portal (i.e. which topics are covered by the documents) as well as on the familiarity of a user with the vocabulary used in underlying documents (i.e. which terms are used for describing a topic). By using this information a user can express his information need in a query that filters only highly relevant documents. For example, the users can avoid forming queries that, according to the underlying vocabulary, describe their information needs too generally and therefore result in lot of irrelevant documents. Furthermore, in the case that there is no result for a user's query, the user can detect which term from the query to relax (delete) in order to get some results. However, most of the portals do not explicate the underlying content and vocabulary, leaving the users to explore them on their own. Consequently, the searching is performed as a try-and-error process: a user forms a short initial query, analyses the list of results and tries to refine/change the query in order to get more relevant results. Moreover, by considering lists of results for such "trial" queries, a user can change slightly his initial information need as well, as a consequence that his initial assumption about what can be found in the portal has been changed. This leads to the further refinements of the user's query.

Therefore, it is clear that the efficiency of the searching for information in a portal depends on the possibility to support users in such a "slightly" changing of their initial queries in order to tailor them to the underlying repository and vocabulary. Such queries express user's information need more clearly and retrieve more relevant documents (information resources). Since the users are reluctant to provide explicit feedback about the relevance of a document, such a relevance should be determined according to their usage and the query refinement process should reflect this phenomenon: the query should be refined such that most frequently used documents are prioritised. Last, since searching is a user-specific activity, the preferences of users have to be accounted in the query refinement process

Not surprisingly, the mentioned query refinement process is a basic method that people use in searching in the brick-and-mortar environment: there is a shop assistant who helps a customer to find an information (product) by considering the types and naming of the available products, the stock information, the preferences of previous users and behaviour of previous users.

However, modern IR systems (information portals) provide very weak support for the query refinement process. This support is mainly based on displaying most frequently appearing terms in the documents relevant for the given query (e.g. www.altavista.com) or, in the systems with the directory structure, on showing the number of documents found in a directory (e.g. www.yahoo.com). Recently, the user feedback is used for filtering the most frequently appearing terms only from frequently used documents [Wen et al. 01]. However, none of these approaches enables a user to orient himself in a larger context of the searching space, i.e. to navigate, regarding the vocabulary and the repository, through the query space in order to find the query that results in more relevant documents.

In this paper we present such an approach for the query navigation, called Librarian Agent, which simulates the role a human librarian plays in the searching for information resources in a library. The 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' activities in the portal. The agent, through an interactive dialogue, guides the users in closing the initial query to the original user's information need, in the query refinement process. Moreover, the Agent supports efficient ranking/clustering of retrieved results. The approach assumes the existence of a common vocabulary that is used for expressing queries, as well as for providing meta-information about the content of information resources. In order to simulate background knowledge that a human librarian uses in searching, we extended the vocabulary to the conceptual model of the given domain, i.e. an ontology. Such a formalisation enables more extensive inspection of a query's properties, which leads to more efficient query refinement. We partially implemented the approach in the Web Portal of our Institute and some initial evaluation results are shown.

The paper is structured as follows: in [Section 2] we derive requirements for the efficient searching in a portal, whereas in [Section 3] we present the architecture of a system which fulfils these requirements. The details about the subsystem for query refinement are given in [Section 4]. [Section 5] contains more information about the evaluation study. Related work is presented in [Section 6]. [Section 7] contains concluding remarks.

2 The Efficient Querying in a KM System – the Need for a Query Refinement Subsystem

The problem of satisfying a user's information need in an Information Portal [Baeza-Yates, Ribeiro-Neto (99)] is the question of whether a relevant information resource for that need exists in the information repository, and if the answer is positive, whether that resource(s) can be found by the user. More precisely, the efficient searching for information in a portal depends on:

1. the "quality" of the information repository,
   - if information resources reflect the needs of users, e.g. if the information repository contains information resources which users are interested in and
2. the "quality" of the retrieval process, i.e. when a relevant information resource exists in the repository, how easily (if any) the resource can be found. This problem can be divided into two sub-problems:
   a) if a resource which is relevant for the user's information need can be found by the querying mechanism and
   b) if a user can (easily) find the resource which is highly relevant for his information need in the list of retrieved results.

The first criterion (1) is the matter of the so-called "collection management policy", which manages the deletion of old information resources and entering of new ones, corresponding to the changes in the user's interests.

The retrieval of resources which are relevant for the user's need (2a) depends on the expressivity of the vocabulary used in the portal. There are two factors which influence finding a relevant resource:

1) the clarity of expressing user's information need in the query posted to the system [Baeza-Yates, Ribeiro-Neto (99)], [Wen et al. 01], since a query is just an approximation of the, often unarticulated, information need
2) the quality of the annotation (indexing) information resources in the repository, i.e. the relevance of the metadata assigned to a resource.

The part of this problem, a so-called prediction game between providers and users of information, can be resolved by using a commonly-agreed, formalized vocabulary, i.e. an ontology [Guarino and Giaretta 95] as the semantic backbone of the portal. We assume that such an ontology exists in the given domain, and that the retrieval system, consequently, benefits from using this conceptual structure in searching for information [Guarino et al. 99]. For example, the usage of synonym terms in two queries will be mapped into the same retrieval process. Moreover, a query can be automatically expanded with new terms, according to the structure of the ontology.

Since users tend to read only few top ranked resources for a query, (easily) finding an information resource that satisfies user's information need (2b) depends on the possibility to calculate precisely the relevance of the resources for the user's query. In other words, an average user will not discover the highly relevant resources placed down in the list of retrieved results.

[Figure 1] summarizes the above-mentioned discussion about the factors which influence the searching for information in a Portal.

However, a user might be not satisfied with the results of a query. The most characteristic "unsatisfactory" situations regarding a query arise when: there is no result for that query, there is few relevant results and there are too many results, what indicates that there might be a lot of irrelevant results.

Each of these situations is caused by some problems in the previously mentioned factors that influence searching for information. For example:

- a problem in the information repository leads to:

  - no relevant resource for the user's query (gap in the repository)

  - too many relevant resources for the user's query (information overload)

- a problem regarding the model used for describing underlying domain (vocabulary/ontology) leads to:

  - representing a user's need ambiguously in a query

  - representing the content of resources ambiguously

  Both of them can result in too much irrelevant/to less relevant results for the user's need.

- a problem in the mechanism/model used for calculating relevance leads to:

  - placing a highly relevant resource below a low-relevant resource in the list of results.

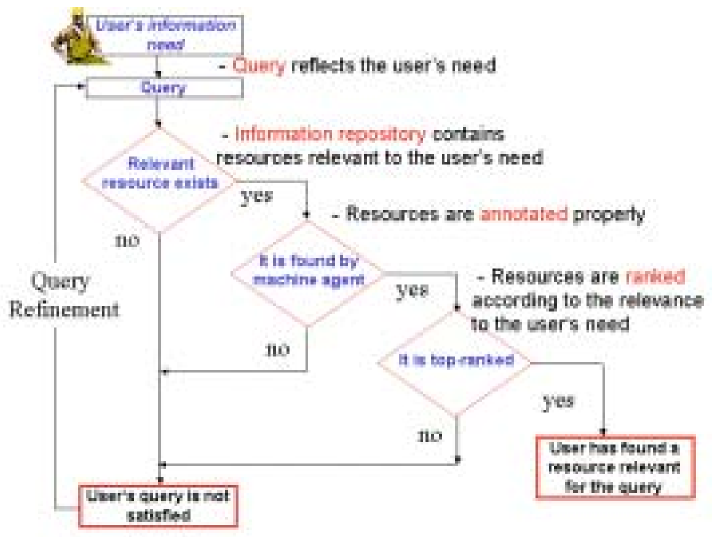

User's information need
- Query reflects the user's need
Query
- Information repository contains resources relevant to the user's need
Relevant resource exists
yes
no
- Resources are annotated properly
It is found by machine agent
yes
no
Query Refinement
- Resources are ranked according to the relevance to the user's need
It is top-ranked
yes
no
User has found a resource relevant for the query
User's query is not satisfied

Figure 1: The factors which influence the efficiency of the searching for information in a Portal

For example, due to an ambiguous interpretation of query terms the list of results can be too long and can contain irrelevant results which are top ranked. Let us assume that a user, who is searching for professors, makes the query "Researcher", whereas the ontology concept Researcher is modelled through three subconcepts: Professor, PosDoc and PhDStudent (see [Figure 3]). Such a query represents initial user's need very ambiguously and results in lots of irrelevant results. Another example can be the gap in the information repository regarding a user’s query, which results in the empty resulting list.

In such situations users try to change the initial query regarding the arisen problems in order to ensure that highly relevant results are top ranked. For the above example, the user can change the query “Researcher” in the query “Professor”, which returns fewer number of resources, which are, on the other side, more relevant for the user’s query. Such a refinement assumes that the user can recognize what is “wrong” in his query. However, generally, a user does not know explicitly what can be the problem in his query since he does not have enough knowledge about the portal’s structure (e.g. the information repository and the vocabulary). For example, in the case that there is no result for a query, a problem is to determine which term causes such a constraint. Moreover, the problems in searching can arise from various reasons. For example, no relevant results for a user need might be caused by a problem in the information repository (no such a resource) or by a problem in the domain model (a “wrong” query term is used). Consequently, the different refinement strategies should be applied in these situations. Leaving a user to gees what can be a problem in a query and what can be the most suitable refinement, makes this refinement a very tedious and error-prone process, i.e. a user tries some refinements by chance and cannot be ever sure that there are any more suitable refinements.

Recently performed large-scale case study about the behaviour of users in searching Web [Silverstein et al. 98] has shown that in one third of the subsequent query modifications the users tend to make a frustrating “total change”, where no word is shared between the two modifications. It can be interpreted as the need of a user to make a more complex refinement of a query, but without knowing how to do that efficiently. In a smaller query transformation analysis, Bruza [Bruza and Dennis 97] found that repeating a query is a frequent transformation, which indicates the high percentage of unsuccessful refinements of the initial query. In such situations, after several refinements, the user comes back to the initial query.

From the previous discussion it is clear that an efficient system for querying a portal should support users in doing refinement of their queries. Indeed, the query refinement can be seen as a way to deal with (to compensate) “problems” in a portal, we mentioned above. For given example, by changing the query “Researcher” in the query “Professor”, the user tries to decrease the ambiguity in the interpretation of his query. Furthermore, by adding a new term with the similar meaning (e.g. a synonym: Scientist - Researcher), the user compensates some constraints in the vocabulary used for the annotation of documents, or some problems in the annotation process.

Therefore, the query refinement enables a user to find relevant resources for his information need in the case that the initial query failed due to some problems in factors which influence the searching process. It enables a user, who made a query, to easily inspect the corresponding (i.e. query-related) part of the information repository and vocabulary, in order to determine what are the sources of the arisen problems (i.e. to determine the ambiguities of the query). On the basis of these ambiguities and the user’s preferences, the query is refined, which results in more relevant results for the user’s information need.

In this paper we present the conceptual architecture of a query refinement system that fulfils above-mentioned requirements. The system is called Librarian Agent since it simulates the role a human librarian plays in the searching for information resources in a library.

3 The Librarian Agent – the Usage Scenario

The role of the Librarian Agent is (i) to resolve the disambiguation of the queries posted by users (query management), (ii) to enable efficient ranking and/or clustering of retrieved answers (ranking) and (iii) to enable the changes in the knowledge repository regarding the users' information needs (collection management).

[Figure 2] sketches the conceptual architecture of the Librarian Agent. In order to make the ideas behind the architecture more understandable, we describe it through several examples of querying the Information Portal of an Institute. It is assumed that the backbone of that Portal is the *ResearchInstitute* ontology, a part of which is depicted in the [Figure 3].

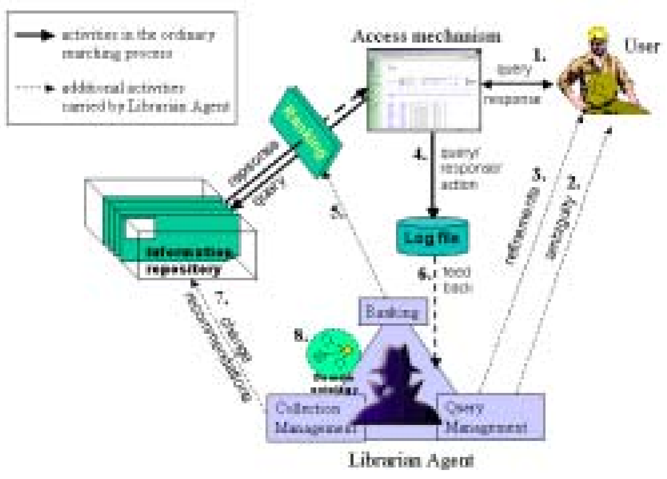


Figure 2: The roles of the Librarian Agent in the process of searching for knowledge

A user posts the query (cf. 1 in [Figure 2]), which is processed firstly by the Librarian Agent. Let us assume that the query is "`Researcher and Project and KM`", e.g. the user is searching for the information resources about "researchers in projects related to the knowledge management (KM)". The Agent measures the ambiguity of the query (cf. 2 in [Figure 2]) by considering the capacity of the knowledge repository and the domain vocabulary - ontology. The user is provided with an explanation what is ambiguous in the query and how this ambiguity can influence the result of the querying.

For the given query, the Agent might find the following ambiguities (more elaborations on these ambiguities are given in the next section):

1) The sense of the term `KM` is not clear: `KM` can be a research area or a lecture, see [Figure 3];

2) The context of the query is not clear: since there are two relations between the terms `Researcher` and `Project` (i.e. `worksIn` and `manages` – see [Figure 3]), the user

should clarify which of these meaning she is interested in. Otherwise, she could get some irrelevant answers;

3) The clarity of the term `Researcher` used in the query is not well determined: since there are three subtypes of `Researcher` (see [Figure 3]), the user should specify which type of `Researchers` she is interested in. Otherwise, she could get some irrelevant results;

4) By analysing the information repository, it follows that the list of answers for the given query is the same as for the query "`Researcher and Project`", which means that all existing `Projects` are about `KM`.

synonyms(Researcher,		
Scientist, Forscher)	Researcher(rst)[2]	workIn(rst, LA)[3]
	Researcher(nst)	workIn(nst, LA)
isA(PhDStudent,	Researcher(ysu)	workIn(ysu, LA)
Researcher)[1]	Researcher(jan)	workIn(jan, LA)
isA(PosDoc,	Researcher(meh)	workIn(meh, LA)
Resaercher),	Researcher(sha)	
isA(Professor,		ResearchArea(KM)
Researcher),	project(LA)	
workIn(Researcher,		researchIn(rst, KM)[4]
Project)	Lecture(KM)	researchIn(ysu, KM)
manages(Researcher,		researchIn(nst, KM)
Project)	PhDStudent(nst)	researchIn(meh, KM)
about(ResearchArea,	PhDStudent(ysu)	researchIn(rst, CBR)
Project)	PhDStudent(meh)	researchIn(nst, CBR)
researchIn(Researcher,	Professor(rst)	researchIn(ysu, CBR)
ResearchArea)	Professor(jan)	
		subtopic(KM, CBR)
teaches(Researcher,		
Lecture)		

Figure 3: A part of the ontology we use for illustrating our approach

Moreover, the Agent recommends the user some changes (refinements) in the query (cf. 3 in [Figure 2]), considering the underlying vocabulary, the information repository and the agent's experience (the past behaviour of the users). For example, beside the refinements related to the cases 1) - 3), the Agent can "recognise" that in the underlying repository there are a lot of resources about `PhD_Students` involved in `projects` in `KM` and it can probably be a suitable refinement of the given query (i.e. "`PhD_Student and project and KM`").

The Agent receives the feedback information about how many (and which) refinements' steps the user performed (cf. 4 in [Figure 2]), and it uses this information to improve its own strategies for creating recommendations.

The **Query Management module** is responsible for the previous two tasks, i.e. for the ambiguity measurement and for the recommendations for the refinements of a query.

Let us assume that the user refined her query into "`PhD_Student and Project and KM`", and the retrieved results are `meh`, `nst`, `ysu` (see [Figure 3]). The retrieved list of results is ranked according to the relevance for the given query. The **Ranking**

[1] It means that `PhDStudent` is a subtype of `Researcher`
[2] It means that `rst` is a `Researcher`
[3] It means that `rst` works in the project `LA`
[4] It means that `rst` researches in the `KM`

module analyses the domain ontology, the underlying repository and the searching process in order to determine the relevance of the retrieved answers (cf. 5 in [Figure 2]) For example, it finds that the answer `nst` is more relevant than the answer `meh`, since `nst` researches in the areas `KM` and `CBR`, whereas `meh` researches only in `KM`. Moreover, the results can be clustered into semantically related groups of results, in order to enhance searching.

The information about which of the retrieved results were clicked by the users can be used for the management of the searching process. In order to avoid disturbing the users by additional questioning, the feedback information is collected implicitly by analysing the activities of the users that are captured in the log file of the system.

Moreover, the list of queries is further analysed (cf. 6 in [Figure 2]) by the Librarian Agent, in order to make recommendations for the changes in the collection (cf. 7 in [Figure 2]) and in the underlying ontology (cf. 8 in [Figure 2]). This is the task of the **Change Management module**. This recommendation takes into account the analysis of the queries posted by users and the used vocabulary, as well. For example, if a lot of users post the query "`Project and Coordination`", which returns zero answers, then it can be interpreted as an unsatisfied information need of lots of users. Consequently, the repository should be extended with such an information resource. Or, if in the underlying ontology the concept Project has no subconcepts, a lot of queries containing the term "`Project`" can be an indicator to split (specialise) the concept Project in several subconcepts (e.g. national project, EU-projects, etc.), in order to support fine-tuning of users' queries.

The conceptual model of the given domain – the domain ontology (cf. 8 in [Figure 2]) supports the processing of each step in this approach. Moreover, the searching mechanism and the information repository are based on the domain ontology.

In the rest of the paper, we present the query management's capabilities of the Librarian Agent in details. More information about the Change Management Module can be found in [Stojanovic and Stojanovic 02].

4 Query Management Module

The goal of this module is to support users in the query refinements process, i.e. to enable a user to find the results relevant for his information need, even if some problems appear in the searching process. As we mentioned in the [Section 2], these problems lead to some ambiguities (e.g. the misinterpretations of the user's need) in the query, such that lots of irrelevant results and/or few relevant results are retrieved. The Query Management module estimates these ambiguities of the initial query (so called Problem Discovery phase) in order to provide suitable modification of that query, which will decrease the number of irrelevant results or/and increase the number of relevant results (so called Query Refinement phase). In the next two subsections we explain these two phases in more details.

4.1 Problem Discovery

We see two general situations[5] in which a user is unsatisfied with the searching process:

- no results for a query
- too many results for a query, what indicates that there might be a lot of irrelevant results.

In both cases, the problems are caused by an ambiguity in the user's query:

1. - the query does not correspond to the information repository and
2. - the query cannot be clearly (uniquely) interpreted

, respectively.

In the first case there is an ambiguous constraint in the query, which cannot be fulfilled in the given repository, since it corrupts domain model (ontology) or it overfits the content of the repository. In order to resolve this problem, that ambiguous constraint has to be discovered and relaxed in the most suitable way (i.e. in a way which introduces the smallest destruction of the initial user's need – for example it is possible to relax an ambiguous constraint by replacing it with another unambiguous constraint). This kind of the ambiguity is out of the scope of the paper.

In the second case there are ambiguous interpretations of the user's query which cause lot of irrelevant results. The problem is how to discover and estimate these ambiguities, in order to enable a user to select the most suitable interpretation of his query, i.e. his information need. This is the topic of the next subsection.

4.1.2 Estimating the Ambiguity in the Interpretation of a Query

Since users tend to post very short queries (average length of a Boolean query is between 2 and 3 terms, according to analysis in [Silverstein et al. 98], a query just represents an approximation of a user's information need [Saracevic 75] and as such it can be misinterpreted in the searching process. This misinterpretation is caused by a problem regarding:

a) the vocabulary (ontology)

e.g. the query "`Researcher`", regarding the ontology represented in Fig 3., can be (mis)interpreted as the information need for information resources about (i) Researchers, (ii) PhDStudents, (iii) PosDocs or (iv) Professors.

b) the information repository

e.g. the query "`Researcher and Project`", regarding the same ontology, can be (mis)interpreted as the information need for information resources about (i) Researchers and Projects or (ii) Researchers, Projects and KM

In the first case, a user's query can be semantically (based on its meaning regarding the underlying ontology) mapped into several queries. In the second case that mapping is syntactic (based on query's results regarding underlying repository). It is obvious that these misinterpretations are caused by some ambiguities in the query regarding the vocabulary and the repository, respectively. By measuring these

[5] We assume that a user just see the list of results, not inspecting the relevance of results. Otherwise, the problems in searching can be analyzed with respect to the relevance of results (e.g. too few relevant results)

ambiguities, the sources of the misinterpretations (problems) can be discovered. Consequently, they have to be resolved in the refinement process.

Therefore, we define two types of the ambiguity[6] in the interpretation of a query: (i) the *semantic ambiguity*, as the characteristic of the used vocabulary and (ii) the *content-related ambiguity*, as the characteristics of the repository.

4.1.2.1 The Semantic Ambiguity

As we already mentioned in the introduction, we consider that the users make Boolean queries (a list of terms concatenated with a logical operator[7]), because forcing users to make formal logic queries slows and constrains information retrieval process. However, we assume that these terms are selected from an ontology. Since an ontology vocabulary allows using synonyms and homonyms, the meaning of some terms in a query can be ambiguous. Therefore, the very first step in our approach is the disambiguation of the meaning of the terms in a query, done by measuring *SenseAmbiguity*. Next, we measure the clarity of the context (defined by relations with other terms) in which a term appears – *ContextClarity*. Finally, we estimate the generality/speciality of a query term by measuring its *Clarity*.

In the following, we define these three ambiguity parameters.

SenseAmbiguity

In order to combine formal modelling of a domain and the user-friendly searching, the abstract model of ontology we use in our research, presented in [3], contains an additional modelling layer, the so-called lexical layer, which is responsible for mapping the terms used by the users in searching into the formal entities of an ontology (i.e. concepts, relations and instances). Due to lack of the space, we omit here the formal definition of the ontology, which can be found in [3], and give an informal explanation. For instance, returning to the example shown in the Fig. 2., the user can use the terms “`Researcher`”, “`Scientist`” or “`Forscher`” in searching for the resources related to the (domain-specific) concept `Researcher` from the ontology. Moreover, a term can be used for encoding several ontology entities, i.e. for representing several meanings. For example, the term “`KM`” can be used for encoding a `Lecture` or a `ResearchArea`, and we say that the term “`KM`” has two senses[8]. Consequently, if a query contains the term “`KM`”, then the Query Management Module has to clarify the sense of that term, i.e. if the query is about a `Lecture` or a `ResearchArea`. The sense can be clarified by analysing the relations between the senses of that term with the senses of other terms in the query. For example, in the query “`KM and Projects`”, the meaning of the term “`KM`” should be the ontology concept `ResearchArea`, because in the *ResearchInstitute* ontology there is a relation

[6] 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, i.e. 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.

[7] Although our approach can be applied to disjunctive queries as well, in order to simplify the explanation of the approach, in the following examples, we use only conjunctive queries.

[8] Similarly to WordNet [Rila 98] synsets

between concepts `ResearchArea` and `Project`, but there is no relation between concepts `Lecture` and `Project`. In case more than one sense is possible, ranking of the senses is needed. It can be done by considering the information repository. For example, in the query "`KM and Researcher`", the meaning of the `KM` can be a `ResearchArea`, as well as a `Lecture`, since there are relations between both of these concepts (`ResearchArea, Lecture`) and the concept `Researcher`, i.e. `researchIn` and `teaches`, respectively. By considering the number of information resources which are about "researching in KM area" and "teaching KM course", the ranking of these two senses of the query "`KM and Researcher`" can be done. Such a discussion is out of the scope of this paper.

In order to estimate this ambiguity, we define *SenseAmbiguity* factor for the query Q = " $t_1, t_2, ...t_n$ " as follows:

$$\mathrm{SenseAmbiguity}(Q) = \frac{\sum_{\forall t_i, t_j \in Q} \mathrm{NumberOfSensesInContext}(t_i, t_j)}{\mathrm{NumberOfSenses}(Q)}$$, where,

$$\mathrm{NumberOfSensesInContext}(t_i, t_j) = \left| \{ i_p \in \mathrm{Sense}(t_i), i_k \in \mathrm{Sense}(t_j) : \mathrm{Relation}(i_p, i_k) \} \right| ,$$

$\mathrm{NumberOfSenses}(Q) = \sum_{\forall t_i \in Q} |\mathrm{Sense}(t_i)|$, $|a|$ denotes the cardinality of the set a.

$\mathrm{Sense}(t_i)$ is the set of the senses of the term t_i in the ontology. For example, $\mathrm{Sense}(KM) = \{\mathrm{lecture},\ \mathrm{researchArea}\}$;

$\mathrm{Relation}(i_p, i_k)$ is the function that returns 1 if there is a relation between i_p and i_k in the given ontology, for the case that i_p and i_k are concepts. In case that i_p and i_k are instances $\mathrm{Relation}(i_p, i_k)$ returns 1 if there is the relation between the ontology concepts which corresponds to the instances i_p and i_k. Analogy definition holds for the case that one of i_p , i_k is an instance and other is a concept.

For example, for the query Q_{initial} = "`Researcher and Project and KM`", we get:

$$\mathrm{SenseAmbiguity}(Q_{\mathrm{initial}}) = \frac{2+2+1}{1+1+2}$$, since

$\mathrm{Sense}(\mathrm{researcher}) = \{\mathrm{researcher}\}$, $\mathrm{Sense}(\mathrm{project}) = \{\mathrm{project}\}$ and $\mathrm{Sense}(KM) = \{\mathrm{lecture},\ \mathrm{researchArea}\}$, i.e. `KM` is the term which is assigned to the instance of a `Lecture` or a `ResearchArea`,

$\mathrm{NumberOfSensesInContext}(\mathrm{researcher}, \mathrm{project}) = 2$, i.e. a `Researcher workIn` or `manages a Project`,

$\mathrm{NumberOfSensesInContext}(\mathrm{researcher}, KM) = 2$, i.e. a `Researcher researchIn a ResearchArea (KM)` or `teaches a Lecture (KM)`and

$\mathrm{NumberOfSensesInContext}(\mathrm{project}, KM) = 1$, i.e. a `Project` is about a `ResearchAarea`.

ContextClarity

This parameter models the existence of incomplete information in a query, regarding the used concepts/relations. It means that the query can be *automatically* expanded, in order to clarify the meaning of the query. For the given ontology, the query "`Researcher and Project and KM`" is incomplete, because there are two relations between concept `Researcher` and `Project`, namely `workIn` and `manages`, which can be used to specify the query more precisely.

For measuring the context clarity of a query, we use the following formulas:

$$\mathrm{ContextClarity}(Q) = \prod_{\substack{i=1,n\\ j=1,n}} \mathrm{Contextuality}(Q, Ci, Cj) \text{ where } Ci, Cj \in Q \text{ , where}$$

$$\mathrm{Contextuality}(Q, C1, C2) = \begin{cases} \dfrac{1}{|\mathrm{Properties}(C1, C2)| + 1}, & |\mathrm{Properties}(C1, C2)| \;\; \geq \;\; 1 \wedge \forall x \in \mathrm{Properties}(C1, C2), x \notin Q \\ \\ 1 & \text{else} \end{cases}$$

$\mathrm{Properties}(C1, C2)$ is the function which returns the set of all properties between `C1` and C2, Q is the given query.

For example, $\mathrm{ContextClarity}(Q_{\mathrm{initial}}) = \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{3}$, whereas each of multiplicands corresponds to the number of the senses calculated for the SenseAmbiguity. The values for NumberOfSensesInContext and Contextuality are similar, because there are no terms which correspond to a relation in the given query. In the case of the query "`Researcher and Project and KM and workIn` " the context of the `Researcher-Project` pair can be treated as "fixed" (i.e. `workIn`) and $\mathrm{ContextClarity}(Q) = 1 \cdot \frac{1}{2} \cdot \frac{1}{3}$.

Clarity

The clarity factor represents the uncertainty to determine the user's interest in the given query. For example, when the user makes a query using the concept `Researcher`, which contains two subconcepts `Professor` and `PhDStudent`, it could be a matter of discussion whether she is interested in the concept `Researcher,` or in one of its subconcepts. Anyway, she failed to express it in a clear manner. The formula for the clarity factor depends on the entity type:

$$\mathrm{Clarity}(Q) = \frac{\sum\limits_{\forall t_i \in Q, i_p \in \mathrm{Sense}(t_i)} \mathrm{TermClarity}(i_p)}{\mathrm{NumberOfSenses}(Q)} \text{, where}$$

$$\mathrm{TermClarity}(E) = \begin{cases} \cdot \dfrac{1}{\mathrm{numSubConcepts}(E) + 1} & \text{E is a concept} \\ \dfrac{1}{\mathrm{numSubPropeties}(E) + 1} \cdot \dfrac{1}{\mathrm{numDomains}(E)} & \text{E is a propetry} \end{cases},$$

numSubConcepts(E) is the number of subconcepts[9] of a concept E, numSubProperties(E) is the number of subproperties of a property E and numDomains(E) is the number of domains defined for the property E.

For the given query $\mathrm{Clarity}(Q_{\mathrm{initial}}) = (\frac{1}{3} + 1 + \frac{1}{4}) / 4$, in case that the concept `Researcher` has two subconcepts and `KM` (as a research area) has 4 subtopics.

[9] It holds for each transitive relation and not only for the isA relation. For example, `subTopic` is a transitive relation.

4.1.2.2 The Content-related Ambiguity

The *content-related ambiguity* of a query depends on the capacity of the information repository. Since this capacity determines the list of the results for a query, the content-related ambiguity of a query can be defined by comparing the results of the given query with the results of other 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 and O is the list of results for the query Q. M and O are called query_terms and query_objects, respectively. Further, we define:

1. *Structural equivalence* (=) by: $(M_1, O_1) = (M_2, O_2) \leftrightarrow O_1 = O_2$

Two query-answering pairs (queries)[10] are structurally equivalent if their result sets are the same.

2. *Structural subsumption* (parent-child): (<) by: $(M_1, O_1) < (M_2, O_2) \leftrightarrow O_1 \subset O_2$.

A query (M_2, O_2) subsumes another query (M_1, O_1) if the result set of the second query pair subsumes the results of the first one. For query-answering pairs Q_1, Q_2 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, Uniqueness, 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: $Q_{a\,min} \in \{(\times(M_i \cap M_a), O_a) \mid Q_a <_{dir} Q_i, i = 1,..n\}$

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:

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

Covering and *CoveringTerms* are parameters 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:

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

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

[10] Due to simplicity, in the rest of the text, we will use the term query for referring to a query-answering pair.

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) [Ganter, Wille (99)] for organising data in the so-called concept lattices which correspond to the multi-inheritance hierarchical clusters. Each of these clusters can be considered a query posted to the repository and, consequently, the lattice represents the clustering of the query space. A cluster is called a formal concept and it contains query terms and resources retrieved for that query. By analysing such a lattice, many interesting relations between queries can be discovered and used for measuring the 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 [Ganter, Wille (99)]. 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. The organisation of the data is achieved via a mathematical entity called a formal context. A formal context is a triple (G, L, I) where G is a set of objects, L 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, L, I) is a pair (A, B) where $A \subseteq G$, $B \subseteq L$, $A = B' = \{g \in G \mid \forall l \in B: (g,l) \in I\}$ and $B = A' = \{l \in L \mid \forall g \in A: (g,l) \in I\}$. For a formal concept (A, B), A is called the 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 by all the objects in A. As the number of attributes in B increases, the concept becomes more specific, i.e. a specialisation ordering is defined over the concepts of a formal context.

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.

Attr. / **Obj.**	Researcher	Pro-fessor	Project	workIn ->>LA **(= LA: Project)**	Research Area	researchIn ->>CBR **(= CBR: ResearchArea)**	ResearchIn->>KM **(=KM: ResearchArea)**
rst	x	x	x	x	x	x	x
nst	x		x	x	x	x	x
ysu	x		x	x	x	x	x
jan	x	x	x	x	x		x
meh	x		x	x	x		x
sha	x						

Table 1: A part of the ResearchInstitute ontology given in the [Section 3]

Note: Since an ontology uses the three-dimensional space for presenting information (object-attribute-value), a transformation into the two-dimensional space (attribute-value) is needed. Due to the lack of the space, we avoid here the discussion about this transformation. For example, the information `rst[worksIn->>LA]`is represented as the pair (`rst,worksIn->>LA`)in the table. In order to enhance the readability of the table, we replace the relations with the name of the domain of that relation (for example - `LA:Project` is the replacement for the `workIn->>LA`, because the relation `workIn` has for the range the concept `Project`).

Such a representation enables a very intuitive interpretation of a query: one can see a formal concept as a representation of a query state, where the intent of the formal concept represents the query itself, and the extent represents all resources that match the query. For example, the query "`Researcher and Project and KM`" will be mapped into the formal concept described as (`{Project, LA:Project, KM:Research_Area}, {meh}`) in the concept lattice. Note that a formal concept encompasses all objects from its super-concepts – i.e. the (attribute, object) set for that formal concept is: (`{Researcher, Project, LA:Project, KM:Research_Area}, {meh, jan, nst, rst, ysu}`).

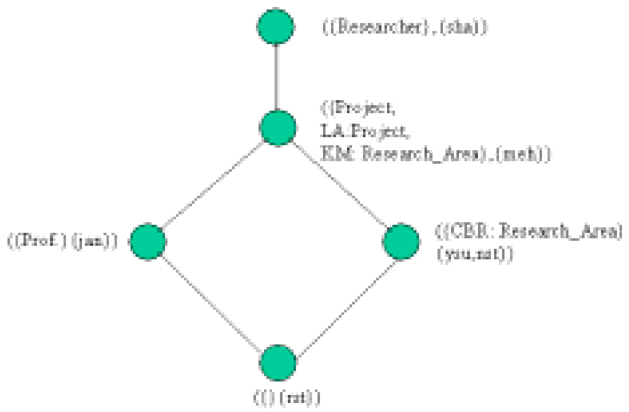


Figure 4: An example which shows the process of generating a concept lattice from a set of data given in the table 1. The concepts represented in the lattice should be read as in the following example: foremost left concept, (`{Prof.}, {jan}`), corresponds to the objects (`jan, rst`) and attributes (`Researcher, Prof., Project, LA:Project, KM:ResearchArea`) – some attributes are inherited from upper formal concepts.

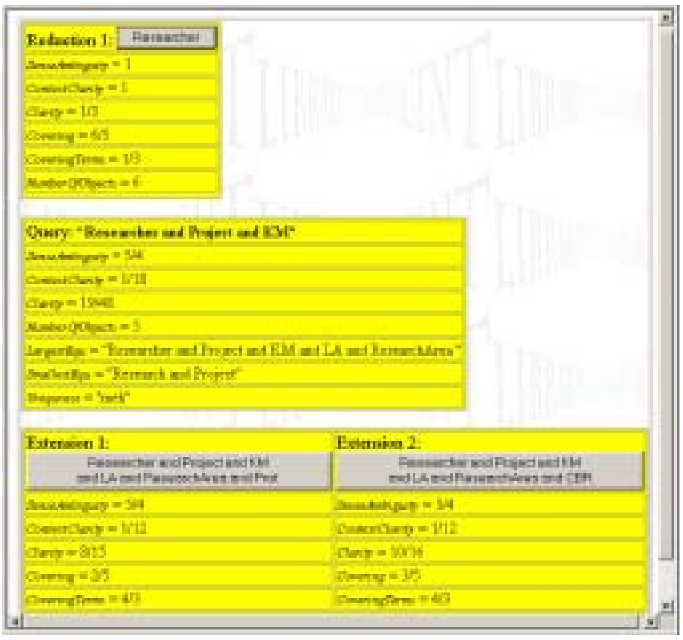
Such an ordering in the query space enables a very easy interpretation of query results regarding their ambiguity. Moreover, the values for the content-related ambiguity parameters can be read directly from the concept lattice. For the given query "`Researcher and Project and KM`", these parameters are as follows:

Largest equivalent query:	"`Researcher and Project and KM` *`and LA and ResearchArea`*"
Smallest equivalent query:	"`Researcher and Project`"
Uniqueness:	"meh"
Covering for upper formal concept:	6/5
CoveringTerms for upper formal concept:	1/3

These parameters are very useful for estimating the ambiguity. A user is provided with this information, in order to determine the position of her query with respect to other queries. That can enhance the efficiency of the query refinement process. For the given example, according to the *Largest equivalent query*, expanding the initial query with the term `ResearchArea` will not cause any changes in the set of answers. Moreover, the *Smallest equivalent query*, "`Researcher and Project`", means that the request `KM` in the query “`Researcher and Project and KM`” is redundant, because all the researchers research in the `KM` research area. Further, according to the *Covering* parameter, almost all results from the query “`Researcher`” are contained in the results of the query “`Researcher and Project and KM`”, which means that the importance of the terms “`Project`” and “`KM`” for the given is not so high. In the next section, we give more details about using *content-related ambiguity* for the query refinement.

4.2 Refinement

Our approach for query refinement reflects the refinement model which a human librarian (or a shop assistant) uses in his daily work. It means that we use three sources of information in suggesting query refinement: (i) the structure of the underlying ontology (vocabulary), (ii) the content of the knowledge repository and (iii) the users’ preferences (what is his task and how users with similar preferences refine their queries).


Reduction 1: Researcher
SenseAmbiguity = 1
ContextClarity = 1
Clarity = 1/3
Covering = 6/5
CoveringTerms = 1/3
NumberOfObjects = 6

Query: “Researcher and Project and KM”
SenseAmbiguity = 3/4
ContextClarity = 1/18
Clarity = 15/48
NumberOfObjects = 5
LargestRelationship = “Researcher and Project and KM and LA and ResearchArea”
SmallestRelationship = “Research and Project”
Response = “tank”

Extension 1: Researcher and Project and KM and LA and ResearchArea and Prof
SenseAmbiguity = 3/4
ContextClarity = 1/12
Clarity = 8/15
Covering = 2/5
CoveringTerms = 4/3

Extension 2: Researcher and Project and KM and LA and ResearchArea and CEO
SenseAmbiguity = 3/4
ContextClarity = 1/12
Clarity = 10/16
Covering = 3/5
CoveringTerms = 4/3

Figure 5: Librarian Agent in the action: The neighborhood of a query. A screenshot from the Portal, which is used in the evaluation. The ambiguity parameters are calculated using formulas presented in [Section 4.1.1]

Since the first two sources are used for measuring the ambiguity of a query, the refinements based on them are treated cooperatively as the *ambiguity-driven query refinement*. Copying with users’ preferences require modelling a user in the short-, mid- and long-term. The short-term user model deals with the current task of the user and can be derived from the current activities of a user. The mid- and long-term models requires maintaining a user profile with “global” preferences of that user.

They enable the personalization of the refinement process. However, that works only for non-anonymous users. Such a personalization is out of the scope of the paper.

4.2.1 Ambiguity-driven Query Refinement

The ambiguity parameters presented in the previous section are combined and presented to the user in case she wants to make a refinement of the initial query. [Figure 5] presents the visual metaphor to present the information about the ambiguities of a query to the user. Each of ambiguity parameters has its role in quantifying ambiguity. [Table 2] presents the most common cases of the ambiguities and their role in the query refinement process. For each of the parameters, query term(s) that affect the ambiguity most importantly are determined. In that way, the user receives the most specific suggestions.

The current version of the Query Management module allows the user to navigate through the query neighbourhood. By clicking on a neighbour, the focus of the map is changed, and all parameters are calculated for that query (see [Figure 4]). In that manner, the user can inspect the queries around the initial query, in order to find the most suitable refinement. This process is called querying by navigation [Bruza and Dennis 97]. More details are given in [Section Evaluation].

Value of Ambiguity Parameters	**Meaning**	**Action**
High *SenseAmbiguity*	Too many interpretations of some terms from the query	To specify the meaning of some terms more precisely – to determine which sense of a term is valid
Low *ContexClarity*	Too many interpretations of the relation between (two) terms	To add a relation in the query in order to specify one of many possible relations between terms
Low *Clarity*	Too general query	To replace a term with a more specific term (from its isA hierarchy)
Big difference between *Smallest equivalent query* and given query	Query contains redundant terms	To reconsider whether the smallest equivalent queries correspond to the initial information need. If this is not the case, then change the query. Define which part of the query is missing in the smallest equivalent query
Big difference between *Largest equivalent query* and given query	Query is too general for the repository	To reconsider if the largest equivalent query corresponds to the initial information need. If this is not the case, then change the query. Define which part of the query is irrelevant in the largest equivalent query
Too low *Uniqueness*	The query shares almost all results with other queries (it contains very few of its own results)	If more results are needed replace the query with a neighbourhood query
Too high *Covering/CoveringTerms*	The query gives similar results as a query from its neighbourhood	If more results are needed move the query in the direction of that “similar” query

Table 2: The suggestions for the query refinement, which are based on the analysis of the ambiguity parameters presented in the previous section.

4.2.2 User-based Refinement

By avoiding logging of users in a portal, the only information about a user's preferences is the current searching session, i.e. which resources form the list of results for a query the user found relevant. Since we avoid explicit feedback about the relevance of resourfces, we assume that clicking (reading) a resource is an evidence that the user is interested in that resource. By analysing these "relevant" resources, the Librarian Agent discovers which properties in resources are of the primary importance for the user. Furthermore, it tries to find more such "relevant" results, i.e. the resources which contain these properties.

A useful recommendation how to make a refinement of a query can be obtained by analysing the refinements made by users whit similar preferences. It requires an analysis of the users' activities in an ontology-based application. In [Stojanovic et al. 02], we presented a framework for capturing the user's activities in a semantic query log file. This query log is "mined", in order to discover query patterns (i.e. regularities in refining the queries). This analysis is out of the scope of this paper.

5 Evaluation

The research presented in this paper is a part of the Librarian Agent, a management system we have developed for the improvement of searching in an information portal. The Librarian Agent is developed using the KAON ontology engineering framework (kaon.semanticweb.org). As a test bed for presented research, we use the VISION Portal (www.km-vision.org), a semantics-driven portal that allows browsing and querying of the state-of-the-art information (researcher, projects, software, etc.) related to the knowledge management. It is developed in the scope of the EU-funded VISION project, which should provide a strategic roadmap towards the next-generation organisational knowledge management. The backbone of the system is the VISION ontology, which includes the *ResearchInstitute* ontology presented in [Section 2]. It is used as a common vocabulary for providing and searching for information. The ontology lexical layer contains about 1000 terms and the information repository consists of about 500 information resources (the web page of concrete person, project, etc.). Each of the information resources is related to a concrete instance in the ontology (e.g. to the person Dietmar Ratz). The query refinement system is implemented as an additional support in the searching process. When the refinement support is turned on, after posting a query, the user gets the query's neighbourhood, similarly to the situation presented in [Figure 4].

The goal of our experiment was to evaluate how the effectiveness of Boolean retrieval is changed when the query process is enhanced with the presented refinement facility. Actually, we evaluated the possibility of our system to help the user define her information need more precisely. To obtain the basic Boolean retrieval system with which to compare our system, we simply turned off the query-refinement support.

For the experiment, we randomly selected 20 queries which cannot be expressed precisely using the defined vocabulary, but whose answers are contained in the information repository. For example, a question was: "Find researchers with diverse experiences about Semantic Web", which cannot be directly expressed using the given ontology vocabulary, but it can be answered by considering the information repository. For example there are two persons who work in five projects related to the Semantic Web. They can be treated as the broadly experienced experts for the Semantic Web.

We tested six subjects in the experiment. The subjects were computer science students with little knowledge of the ontology domains (or domain) and no prior knowledge of the system. The six subjects were asked to retrieve the documents relevant to the 10 queries in one session using the two retrieval methods. For assigning the queries to the methods, we used a repeated-measures design, in which each subject searched each query using each method. To minimise sequence effects, we varied the order of the two methods. The subjects were asked to confirm explicitly when they found a relevant answer. Otherwise, the searching was treated as unsuccessful.

For each search, we considered four measures: success, quality, number of Boolean queries, and search time (i.e. the time needed by the user to perform her task). The quality (0 – 1) is the subjective judgment of the three domain experts about the relevance of the results which are proclaimed by the user as a success. The results are displayed in [Table 3]. The table shows that searching with query refinement support results in better evaluation scores for all measures. These results are not surprising, because our approach complements the basic capabilities of a Boolean retrieval system with additional useful features. In particular, it allows smooth query refinement/enlargement, which is likely to be the key factor for obtaining the improvement in the searching time [Carpineto and Romano 98]. Moreover, the experiment shows that our system can play the role of a query-assistant who, according to the user's query, provides more (quantified) information about the queries "around" the initial query, making the process of expressing/satisfying the user's needs more efficient (about 85% of searching was highly relevant).

Method	**Success for the session**	**Quality for the session**	**Number of queries pro a question**	**Search time (sec) for session**
Boolean	57%	0.6	10.3	2023
Our	85.7%	0.9	5.2	1203

Table 3: Average values of retrieval performance measures

6 Related Work

Query Ambiguity. The determination of an ambiguity in a query, as well as the sources of such an ambiguity, is the prerequisite for the efficient searching for information. Word sense disambiguation of the terms in the input query and words in the documents have shown to be useful for improving both precision and recall of an IR system [Rila 98]. In [Voorhees 94], the set of experiments using lexical relations from WordNet for the query expansion is described, but without treating the query ambiguity. Although some work has recently been done in quantifying the query ambiguity based on the language model of the knowledge repository [Ponte and Croft 98], [Cronen-Townsend and Croft 02], the IR research community has not explored the problem of using a rich domain model in modelling the querying. Some very important results in the query analysis can be found in the deductive database community [Chakravarthy et al. 90], 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 the 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 modifying queries or query results to the needs of 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, usually based on hyper-index [Bruza and Dennis 97] or concept-hierarchies [Joho et al. 02] automatically constructed from the document repository. In [Wen et al. 01] the model of the query-document space is used for the interactive query expansion. *Recommendation systems* [Balabanovic and Shoham 97] 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 [Joachims et al. 97] 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 the rich domain model for the refinement of a query, i.e. the reasons for doing a refinement are not based on the deep understanding of the structure of a query, or the deep 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.

7 Conclusion

In this paper, we presented an approach for the query management in ontology-based IR systems. The system realises a library scenario in which users search for information resources in a repository. 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 find the resources they are interested in. Based on various analyses, the agent, through an interactive interface, guides the users in more efficient searching for information. We presented an evaluation study, which showed that this approach decreases the time, and enhances the precision of the retrieval process.

We find that our approach represents a very important step in using paradigms from searching in the brick-and-mortar environment for the improvement of searching for information in the virtual world. Moreover, this approach leads to the self-adaptive knowledge portals, which can discover some changes in the user's preferences automatically, and evolve the structure of the portal correspondingly.

Acknowledgement

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

References

[Baeza-Yates, Ribeiro-Neto (99)] Baeza-Yates, R., Ribeiro-Neto, B., *Modern Information Retrieval*, Addison-Wesley-Longman Publishing co., 1999.

[Balabanovic and Shoham 97] Balabanovic, M., Shoham, Y: Content-Based, Collaborative Recommendation. CACM 40 (3): 66-72 (1997)

[Bruza and Dennis 97] Bruza, P.D., Dennis, S.: Query Reformulation on the Internet: Empirical Data and the Hyperindex Search Engine. RIAO97, Computer-Assisted Information Searching on Internet, Montreal (1997)

[Carpineto and Romano 98] Carpineto, C., Romano, G.: Effective re formulation of boolean queries with concept lattices. Flexible Query Answering Systems FQAS'98, Springer-Verlag (1998) 277-291

[Chakravarthy et al. 90] Chakravarthy, U., Grant, J., Minker, J.: Logic-based approach to semantic query optimization. ACM Transactions on Database Systems, 15(2) (1990) 162-207

[Cronen-Townsend and Croft 02] Cronen-Townsend, S. and Croft, W.B., Quantifying Query Ambiguity, HLT 2002 (2002) 94-98.

[Ganter, Wille (99)] Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer (1999)

[Guarino and Giaretta 95] Guarino, N. and Giaretta, P. 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing 1995. IOS Press, Amsterdam: 25-32.

[Guarino et al. 99] N. Guarino, C. Masolo, and G. Vetere, “OntoSeek: Content-Based Access to the Web”, *IEEE Intelligent Systems*, 14(3), pp. 70-80, (May 1999).

[Joachims et al. 97] Joachims, T., Freitag, D., Mitchell, T.: Webwatcher: A tour guide for the World Wide Web. IJCAI-97 (1997)

[Joho et al. 02] Joho, H., Coverson, C., Sanderson, M., Beaulieu, M.: Hierarchical presentation of expansion terms, ACM SAC, (2002)

[Maedche (02)] Meadche, A.: Ontology Learning for the Semantic Web, Kluwer Academic Publishers (2002)

[Ponte and Croft 98] Ponte, J., Croft, W.B.: A language modeling approach to information retrieval, ACM/ SIGIR’98 (1998) 275-28

[Rila 98] Rila, M.: The Use of WordNet in information retrieval. ACL Workshop on the Usage of WordNet in Natural Language Processing Systems (1998) 31-37.

[Saracevic 75] Saracevic, T. (1975). 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

[Silverstein et al. 98] Silverstein, C., Henzinger, M., Marais, H., Moricz., M. Analysis of a Very Large Alta Vista Query Log, *SRC Technical Note*, 1998-14, 1998.

[Stojanovic and Stojanovic 02] Stojanovic, N., Stojanovic, L.: Usage-oriented Evolution of Ontology-based Knowledge Management Systems, ODBASE 2002, LNCS (2002)

[Stojanovic et al. 02] Stojanovic, N., Stojanovic, L., Gonzalez, J.: More efficient searching in a knowledge portal – an approach based on the analysis of users' queries, PAKM 2002, Vienna, LCNS/LNAI (2002)

[Voorhees 94] Voorhees, E.,: Query expansion using lexical-semantic relations, 17th ACM/SIGIR, Dublin, (1994)

[Wen et al. 01] Wen, J.-R., Nie, J.-Y. and Zhang, H.-J. Clustering User Queries of a Search Engine. WWW10, May 1-5, 2001, Hong Kong (2001)