Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften (Dr. rer. pol.)
von der Fakultaet fuer Wirtschaftswissenschaften
der Universitaet Fridericiana zu Karlsruhe
genehmigte Dissertation.

Ontology-based Information Retrieval: Methods and Tools for Cooperative Query Answering

M.Sc. Nenad Stojanovic

Referent:
Prof. Dr. Rudi Studer, Universitaet Karlsruhe (TH)
Korreferent:
Prof. Dr. Andreas Geyer-Schulz, Universitaet Karlsruhe (TH)

Tag der muendlichen Pruefung: 25. Juli 2005

Abstract

By the explosion of possibilities for a ubiquitous content production, the information overload problem reaches the level of complexity which cannot be managed by traditional modelling approaches anymore. Due to their pure syntactical nature traditional information retrieval approaches did not succeed in treating content itself (i.e. its meaning, and not its representation). This leads to a very low usefulness of the results of a retrieval process for a user's task at hand.

In the last ten years ontologies have been emerged from an interesting conceptualisation paradigm to a very promising (semantic) modelling technology, especially in the context of the Semantic Web. From the information retrieval point of view, ontologies enable a machine-understandable form of content description, such that the retrieval process can be driven by the meaning of the content. However, the very ambiguous nature of the retrieval process in which a user, due to the unfamiliarity with the underlying repository and/or query syntax, just approximates his information need in a query, implies a necessity to include the user in the retrieval process more actively in order to close the gap between the meaning of the content and the meaning of a user's query (i.e. his information need).

This thesis lays foundation for such an ontology-based interactive retrieval process, in which the retrieval system interacts with a user in order to conceptually interpret the meaning of his query, whereas the underlying domain ontology drives the conceptualisation process. In that way the retrieval process evolves from a query evaluation process into a highly interactive cooperation between a user and the retrieval system, in which the system tries to anticipate the user's information need and to deliver the relevant content proactively. Moreover, the notion of content relevance for a user's query evolves from a content dependent artefact to the multidimensional context-dependent structure, strongly influenced by the user's preferences. This cooperation process is realized as the so-called Librarian Agent Query Refinement Process.

In order to clarify the impact of an ontology on the retrieval process (regarding its complexity and quality), a set of methods and tools for different levels of content and query formalisation is developed, ranging from pure ontology-based inferencing to keyword-based querying in which semantics automatically emerges from the results.

Our evaluation studies have shown that the possibilities to conceptualize a user's information need in the right manner and to interpret the retrieval results accordingly are key issues for realizing much more meaningful information retrieval systems.

Table of Contents

List of Figures

List of Tables

1 Introduction

Ten years ago one of the main slogans for the emerging data mining approaches was "drowning in data but starving for information", indicating a very strong (business) pressure to extract some useful meaning from megabytes of raw data. Consequently, very efficient methods for multidimensional data analyses have been developed and implemented in the mainstream database products.

Recently, due to the explosion of possibilities for a ubiquitous content production[1] and delivery, another information disaster is inevitable: "we are drowning in content but starving for information". Indeed, just consider an ordinary task to find an information in the local content management system: a short query produces several hundreds of results, whereas usually:

- we are willing to see just tens of results, neglecting the effect of the hundreds remaining ones,
- we are not satisfied with the relevance of results in general,
- we are not sure whether it makes sense to continue search with a new query,

to name but a few frequently reported problems in an information retrieval process.

In other words, we are lost in the content space. Consequently, we are short in the information supply, although relevant content could be found out there.

The situation is even worse if we consider web search, where a user on average considers just twenty results of a million retrieved results. However, note that the inter-organizational content production exploded in the last years: for example the IBM Web is now alone as large as WWW four years ago.

Although the data mining research community has expanded their analyses on text (or even more, multimedia) data, the problem of gathering useful knowledge in an information retrieval tasks remains unsolved: even using the very powerful clustering of results[2] the level of the "meaningfulness" in the keyword based querying lies far away from the human cognitive capabilities. Indeed, the problem is not only which algorithm to use to produce the list of results, but moreover how to conceptualize the entire retrieval process, e.g. how to understand what a user is searching for.

We see two main causes of this modern "starving for information":

- a very simple, pure syntactical representation of an information content and
- a very obscure representation of a user's information need using just a set of keywords.

Indeed, by representing an information content as a syntactical structure, we are loosing the possibility to interpret its meaning semantically, e.g. to say that a document about cars talks about vehicles, even when the word vehicle does not appear in that document.

[1] The latest study into information growth done at University of California at Berkley estimates that 5 exabytes of recorded information were created worldwide in 2002 (equivalent to 800 Mb for each person on the planet). www.sims.berkeley.edu/research/projects/how-much-info-2003

[2] like vivisimo.com

Further, by approximating an information need in a set of terms, we are giving just a flavor what a user might be searching for, which results in lots of irrelevant results that are delivered to the user.

Therefore, the main problem in the current approaches in information retrieval is neglecting the role that conceptualization of the domain of interest (i.e. background knowledge) plays in the process of perceiving the information by humans. Figure 1.1 illustrates the role of such a conceptualization for processing information very clearly: our background knowledge enables us to extend the information we are actually perceiving in order to get the meaning of (in this case visual) information. Indeed, although the shape represented in Figure 1.1 is not completely connected, our "brain" extends missing dots perceived by eyes and forms (quite) clear meaning about what is on the picture: a Dalmatian. Moreover, our brain focuses first of all on the most general information what the picture is about and then tries to refine it. Indeed, we recognize first a shape of a dog-like animal and afterwards we notice that it could be a Dalmatian.

Figure 1.1: An example of using background knowledge in perceiving information: One can easily recognize the head and probably one leg on the picture. All other pieces of information are derived from our "brains", like: (1) when something has a head and a leg, then this could be an animal (2) since an animal has (usually) four legs, three other legs are somewhere in the picture. Indeed, we can "recognize" two back legs, although by focusing only on that part of the picture no meaningful objects can be found

This is exactly the case in the information retrieval: we are provided with the blurred, partial, sometimes inaccurate information and a very efficient abstraction of this information is needed in order to discover what the information is about. One of the very promising means to achieve such an abstraction is to use ontologies: a very efficient paradigm for the formal and explicit specification of the conceptualization of a domain.

On the other hand, a very comprehensive notion of relevance is needed in order to determine whether the information we have found matches what we are searching for. Indeed, not only the information per se, but moreover its relations to other information and the context in which this information is considered are very important to determine the relevance. Let us consider Figure 1.2 by assuming that we are searching for (pictures of) oranges. In case a) it is not clear whether this is a fruit or another oval object. If we get an additional information (e.g.

another view) about this object, as presented in case b), we get more confidence that this is a fruit that looks like an orange. If we know to which other objects this object is related to, like presented in case c) where a tree can be found as related to the object, then we are quite confident that the picture is about a fruit (orange).

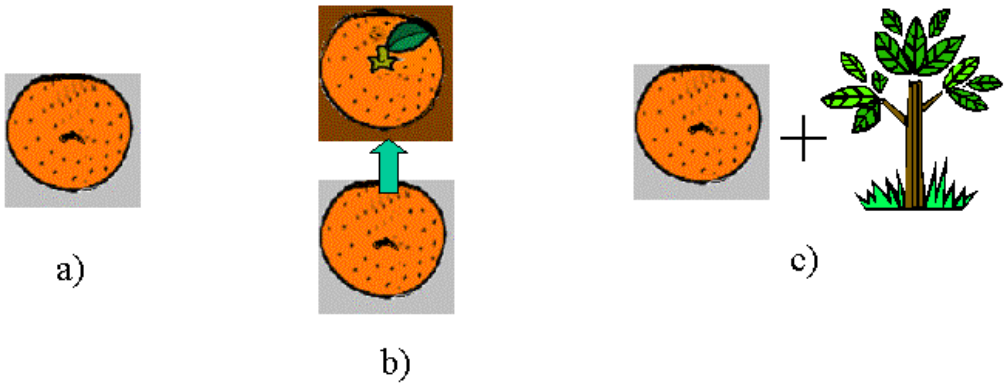


Figure 1.2: An example of using additional information about an object in order to conclude what it is about: b) illustrates the role of additional information about the object itself (e.g. various views on an object) on its interpretation, c) illustrates the role of the information from the context in which an information appears (e.g. the "neighborhood" of that object) on its interpretation

Finally, the importance of the personal context in which an information is processed is inevitable for the determination of the relevance of that information. Let us assume that we are searching for pictures about females and Figure 1.3 is retrieved. How can we classify the content of this picture: a young or an old female. Indeed, depending on our personal interpretation of some parts of the figure, we see either a young lady or an old woman. For example, that what seems to be one ear of the young lady with the black coat and black hair, can be interpreted as one eye of an old woman whose head is covered with a scarf.

Figure 1.3: Visual delusion. An illustration of the role the personal context plays in interpreting information. Do we see here a young lady or an old woman? This picture was originally published in Puck magazine in 1915 as a cartoon entitled "My Wife and My Mother-in-Law."

Another "side-effect" is worth to be mentioned here: most people recognize in the first place the face of a young lady. An explanation can be that this face seems to be already seen, whereas the old woman has some very unusual features, like a very large nose or a "strong" chin. Therefore, we derive the meaning based on some "statistical" information that, certainly, corresponds to our expectations, belief, etc.

Although we have illustrated the mentioned issues using graphical means, it is clear that they are valid for the search for information in general. Indeed, the usage of background knowledge in order to handle the missing information, analysis of the context information in order to get the "big picture" and personalized interpretation of perceived information are "standard" techniques we are (daily) using in order to "optimize" search in the "brick-and-mortar" environment. The vision of this thesis is to lay foundation for such an information retrieval system.

In particular, there are two **main objectives** of this thesis:

- To define a model for the ontology-based information retrieval that supports a semantic notion of relevance;
- To develop a set of methods and tools that enables an efficient organization of the information space, as well as the conceptualization of the user's information need (and his cognitive space) in order to properly determine how an information is relevant to a user's information need and consequently to resolve information overload.

This approach should involve a user in a search process more actively and, like in a brick-and-mortar environment, enable the retrieval system to cooperate with the user in resolving his information need by providing additional hints or similar results.

According to the previous discussion, we identify three **main issues** regarding an information retrieval process in order to resolve the above-mentioned tasks:

(i) Role of an ontology:
The crucial question is what is the most appropriate role an ontology can play in an information retrieval process. Indeed, on one side, the quantity of domain knowledge increases the quality of interpreting an information. On the other side, the process of producing an ontology and the corresponding indexing of resources is very expensive. Therefore, the solutions range from the full ontology-based approach in which the query as well as the information resources are represented using an ontology, through the attribute-value based approach, where a query has a predefined, although not formal, structure, to the keyword-based approach in which the semantic interpretation of information resources is automatically done at query time.

(ii) Notion of relevance:
In contrast to traditional approaches, we treat "relevance" as a complex artefact that depends on many factors, including the context in which a resource appears or the reason why a resource is derived. The main assumption is that as many, mutual independent, sources for relevance are considered, there are better chances that a very relevant resource will be discovered by the user, i.e. that it will be ranked in the list of retrieved results appropriately. Since we treat ontologies as non-probabilistic structures, an adequate notion of relevance has to be introduced in the ontology-based information retrieval.

(iii) Interaction with users:
As we already explained, due to the very obscure representation of a user's information need in a query, the personal context of a user is one of the most influencing factors for determining what a user is searching for. However, users are usually reluctant to give any feedback information about relevance, such that some implicit indicators have to be

analyzed. The main issue is how to organize interaction with a user (e.g. which information to present to him and how to interpret his responses) such that an efficient cooperation can be established: from as few as possible indicators regarding the user's behaviour, as much as possible information about his information need will be derived. This interaction should be realized as a query refinement process.

The **main contributions** of this thesis are:

1. A **model for ontology-based information retrieval, in which the notion of relevance is defined on the conceptual level**. We consider inference processes in which a result is derived as well as the semantic context in which it appears in order to define not only whether, but moreover why (conceptually) the result is relevant for a user's query.

2. An **information-need driven and ontology-based step-by-step query refinement process**. The process is based on a successive decrease of the ambiguity of a query w.r.t. the user's information need. In each refinement step the user is provided with a minimal but complete set of refinement that enables an incremental development of his information need. Special attention is given to comprehensive modelling of factors that determine relevance and to methods for implicit discovery of user's preferences. The process is instantiated in three models which correspond to the three most general information retrieval cases:

- the full-fledged ontology-based refinement, required for search on the Semantic Web;
- attribute-value ontology-based refinement, suitable for product-catalog applications;
- conceptual keyword-based refinement, applicable for traditional keyword-based search.

3. A **new model for cooperative query answering**. Due to our strongly user oriented interpretation of relevance, we move the focus of refinement from a user's query to his cognitive space, such that it is possible to refine not only his query, but moreover that what he is searching for (i.e. his information need). In that way we achieve a cooperation approach that is very close to that what can be found in the real world.

4. A **new, logic-based model for query refinement in the framework of keyword-based search**. We introduce a conceptual model for defining the interpretation of queries such that the query refinement process is performed on the level of the query model (i.e. meaning of a query), which ensures the relevance of generated refinements for a user's information need. Moreover, since a query is represented as a set of logic formulas, the query refinement process is modelled as an inferencing process. We have developed several formal methods for automatically instantiating the conceptual model needed for the approach, in a bottom-up fashion from the query's results.

5. **Extensive evaluations and real-world deployment**. In order to estimate practical benefits of our ideas w.r.t. existing approaches, all methods and tools that are developed in this work are extensively evaluated. This enables us to get new insights about the real advantages of using ontologies in an information retrieval task. Moreover, the conceptual keyword-based query refinement has been deployed in a large-scale information portal.

This thesis **is organized** as follows:

In Chapter 2 we introduce the basic information retrieval process and discuss the role that logic can play in it, especially with respect to the notion of relevance. Afterwards we clarify the role an ontology can play in a logic-based information retrieval process and present our approach for the ontology-based information retrieval. The notion of cognitive relevance as a model for defining relevance in ontology-based search is defined as well. Finally we elaborate on the implementation of the approach in the Ontobroker system and present an evaluation study.

In Chapter 3 we discuss the role the user's interaction can play in an information retrieval process, especially regarding the conceptualization of his information need. We emphasize the role of the query refinement in the retrieval process and introduce the Librarian Agent Query Refinement Process, as the conceptual model for handling refinement in the ontology-based retrieval.

In Chapter 4 we instantiate the Librarian Agent Query Refinement Process for the case of the full-fledged ontology-based retrieval, in which the query as well as the information resources are represented using an ontology. We consider query refinement as a θ-subsumption problem and define a logic-based refinement operator for efficiently traversing the query refinement space. An evaluation of the proposed approach is presented as well.

In Chapter 5 we instantiate the Librarian Agent Query Refinement Process for the case of the attribute-value ontology-based retrieval, in which the query is structured in the form of ontology-based attribute-value pairs. We define a comprehensive model for the lattice-based organization of such a query space and use methods from Formal Concept Analysis for its efficient calculation. We give an extensive conceptual evaluation of the benefits of our approach regarding traditional product-catalog applications.

In Chapter 6 an instantiation of the Librarian Agent query refinement process for the case of keyword-based search is given. We present the conceptual query model and the logic-based approach for the instantiation of that model. The logic-based nature of the refinement opens a palette of additional services (e.g. cooperative answering) that go beyond refinement of a user's query toward the refinement of the user's information need. A case study that considers search a bibliographic database shows the advantages of the approach.

Chapter 7 contains some concluding remarks and the outlook for future work.

In order to improve the readability of particular chapters and to emphasize the relations between chapters, "conceptual models" of the solutions that are provided in each chapter are given in Appendix A.

Parts of this thesis have been published before as follows:

Chapter 2:

A. Maedche, S. Staab, N. Stojanovic, R. Studer, Y. Sure, *SEmantic portAL: The SEAL Approach*, Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential, MIT Press 2003, ISBN 0-262-06232-1, pp. 317-359, 2003.

N. Stojanovic, R. Studer, L. Stojanovic, *An Approach for the Ranking of Query Results in the Semantic Web*, In Proceedings of the International Semantic Web Conference (ISWC 2003), Sanibel Island, FL, USA, pp. 500-516, 2003.

Chapter 3:

N. Stojanovic, *On the Query Refinement in the Ontology-based Searching for Information*, Information Systems, Elsevier Science, ISSN 0306-4379, accepted for publishing, 2005.

N. Stojanovic, *On the role of a Librarian Agent in ontology-based Knowledge Management Systems*, J.UCS Special Issue WM 2003, Volume 9 / Issue 7, pp. 697-718, July 2003.

N. Stojanovic, *On Analysing Query Ambiguity for Query Refinement: The Librarian Agent Approach*, in Proceedings of 22nd International Conference on Conceptual Modelling (ER 2003), LNCS 2813, Chicago, IL, US, pp. 490-505, 2003.

Chapter 4:

N. Stojanovic, *Information-need Driven Query Refinement*, Web Intelligence and Agent Systems, An International Journal, *IOS* Press ISSN: 1570-1263, planned for Vol. 3, No 3, 2005.

N. Stojanovic, *A Logic-Based Approach for Query Refinement*, in Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004), 20-24 September 2004, Beijing, China, pp. 477-480, 2004.

Chapter 5:

N. Stojanovic, L. Stojanovic, *On Modelling Cooperative Retrieval Using an Ontology-Based Query Refinement Process*, in Proceedings of 23rd International Conference on Conceptual Modelling (ER 2004), LNCS 3288, Shanghai, China, pp. 434-449, 2004.

N. Stojanovic, *On Ranking Refinements in the Step-by-Step Searching through a Product Catalogue*, in Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), Brighton, UK, pp. 527-530, 2004.

N. Stojanovic, *On Using Query Neighbourhood for Better Navigation through a Product Catalog: SMART Approach*, in Proceedings of 2004 IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE 04), ISBN 0-7695-2073-1, Taipei, Taiwan, pp. 405-412, 2004.

N. Stojanovic, *On the Role of Query Refinement in Searching for Information: The Librarian Agent Query Refinement Process*, in Proceedings of 4th International Conference on Web Information Systems Engineering (WISE 2003), Rome, Italy, pp. 41-52, 2003.

Chapter 6:

N. Stojanovic, *On the Conceptualization of the Query Refinement Task*, in Library Management, Emerald Group Publishing, Volume 26, Number 4/5, pp. 281 – 294, 2005.

N. Stojanovic, R. Studer, L. Stojanovic, *An Approach for Step-By-Step Query Refinement in the Ontology-Based Information Retrieval*, in Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004), 20-24 September 2004, Beijing, China, pp. 36-43, 2004.

2 Ontology-based Information Retrieval

2.1 Basic Information Retrieval process

Information Retrieval (IR) is the science and technology concerned with the effective and efficient retrieval of information from an information repository for the subsequent use by interested parties. The central problem in IR is the quest to find a set of relevant information resources, amongst a large repository, containing the information sought thereby satisfying an *information need* usually expressed by a user with a *query*. The information resources may be objects (items) in any medium, text, image, audio, or, indeed, a mixture of all three.

The basic IR process consists of two steps represented in Figure 2.1.

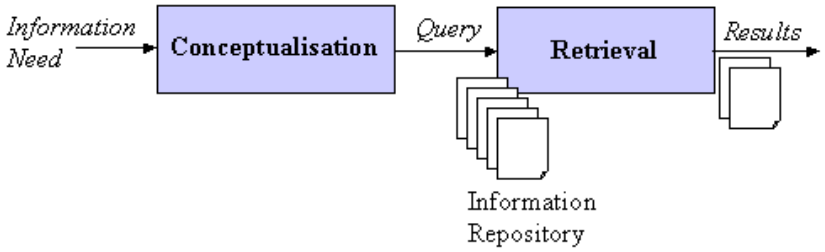


Figure 2.1: Basic IR process

In the *Conceptualisation* step (see Figure 2.1), the user tries to represent his information need using the query syntax that underlies that IR process. However, the complexity of an information need is only partially reflected in the query. Indeed, the user usually tries to approximate his need in a query [110], by representing only several, for him prominent, characteristics of his need, so that the query is usually represented as a set of terms related to some coordinating terms/symbols. Since this step usually causes ambiguities that appear in a retrieval process, it serves as the main source for the refinement process and will be elaborated in Chapter 3.

In the *Retrieval* step (see Figure 2.1) the query is "executed" against the underlying information repository according to a retrieval model, such as Boolean, vector space, probabilistic model [5]. In the nutshell of each model is the calculation of the *aboutness* of an information resource[3] r about a query q, in the notation *about*(r, q).

Usually, the threshold for the *aboutness* is defined so that all the resources whose *aboutness* for the given query is greater than the given threshold are outcomes of the retrieval process. Moreover, the quantification of the *aboutness* is used as a measure for the *relevance* of a resource for the given query, i.e. for the ranking of retrieved resources.

In other words, a *retrieval model* specifies the three basic entities of retrieval [145]:

- representation r of information resources R,
- representation q (called query) of users' information needs Q, and,
- retrieval function M, assigning a set of resources r to each information need q.

To each retrieved resource r w.r.t. query q a degree of (system) relevance is given, called *retrieval status value* (denoted by *RSV* (r, q)), indicating the confidence the system has in

[3] The notation "r" (stands for "information resource" or just "resource") is chosen in order to abstract the meaning of a "document". For example, a resource can be a multimedia document.

being resource *r* relevant to query *q*. Different retrieval methods compute *RSV*s in different ways and different scales (e.g. {0,1}, [0,1], $\Re$)[4].

Therefore, we can characterise a retrieval model formally as follows:

- let L_{Res} be the set of entities used for modelling characteristics of information resources,
- let L_{Query} be the language for modelling user's information needs $q \in Q$,
- let $IR \subseteq L_{Res}$ be a repository (collection) of information resources $r \in IR$,

then a retrieval (matching) function *M*, may be seen as a function:

$$M : 2^{L_{Res}} \times L_{Query} \rightarrow 2^{(L_{Res} \times [0,1])},$$

i.e. given an information repository *IR* and query *q*, *M*(*IR*, *q*) returns a set of pairs (*r*, *n*), where to each resource *r* confidence *n*, the system has in being resource *r* relevant to query *q*, is associated, *i.e.* $n = RSV(r, q)$.

For example, in *Text Information Retrieval* systems, text document *r* and information need *q* are represented as a vector of weighted terms, *i.e.* $L_{Res} = L_{Query} = [0, 1]^l$. Retrieval function *M* is defined in terms of a similarity function $f(q, r)$ between a document representation and a query representation. Similarity function $f(q, r)$ is then determined according to some relatedness measure between vectors $q, r \in [0, 1]^l$, *e.g.* Euclidian distance, cosine, etc.

2.1.1 Ranking of Results

The Probability Ranking Principle[5] (PRP) states that optimum retrieval (defined with respect to resource representations) is given if the retrieved resources are ranked according to their probability $Pr(rel|r,q)$ that resource *r* is relevant to the user query *q* ("probability of relevance") where *rel* stands for the event that this relationship is judged relevant by the user. This is especially important for advanced applications, like filtering. For ad-hoc retrieval, the retrieval methods do not have to estimate these probabilities of relevance directly; instead, it is sufficient to rank resources according to resources' *RSV*s if these are monotonically increasing with $Pr(rel|d,q)$.

The actual relationship between the *RSV* of a document and its probability of relevance can be approximated by a "mapping function": $mf : \Re \rightarrow [0,1]$, $mf(RSV(r,q)) \approx Pr(rel|q,d)$.

2.1.2 Effectiveness of Retrieval

The performance of the retrieval step can be evaluated through the *recall* and the *precision* of the retrieval. For the given query and the information retrieval system's response, the following numbers can be determined:

- *c* = number of relevant resources in repository,
- *n* = number of resources retrieved, and,
- n_r = number of relevant resources retrieved.

[4] In the Boolean model, RSVs are either zero or one. The well-known vector-space-model can be used with the cosine metric (RSVs in [0,1]) or the scalar product (RSVs in $\Re$) [5].

[5] If a reference retrieval system's response to each request is a ranking of the documents in the collections according to the decreasing probability of usefulness to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data made available to the system for this purpose, then the overall effectiveness of the system to its users will be the best that is obtainable on the basis of that data.

Assuming n, $c > 0$ then:

$recall = \frac{n_r}{c}$, i.e. the fraction of the relevant resources that are retrieved in the retrieval process, and,

$precision = \frac{n_r}{n}$, i.e. the fraction of retrieved resources that are relevant.

The main problem in achieving an effective retrieval is that *precision* and *recall* are inversely related, i.e. higher recall *usually* causes lower precision and vice versa, as illustrated in Figure 2.2.

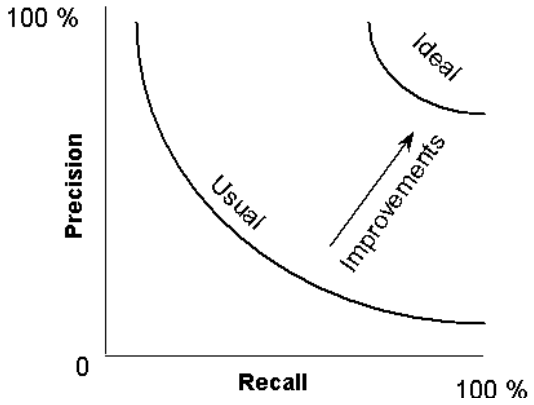


Figure 2.2: Precision-recall trade-off

Indeed, in traditional IR, the set of resources that are retrieved due to some changes in the retrieval model as an attempt to increase recall, usually contain a lot of irrelevant resources (lower precision). The main reason is that the matching process is performed on a syntactical basis.

2.2 Logic-based Information Retrieval

2.2.1 Introduction/Motivation

The main problem in classical approaches to the information retrieval (IR) is the ad-hoc nature of the retrieval process in which various syntactic and structural analyses of a resource are performed in order to define the relevance of the resource for the user's query [102]. In other words, the aboutness of a resource *r* about a descriptor (query) *q* is defined on the level of the representation of a resource and not regarding the meaning of the resource's content. For example in the text IR systems based on the Boolean retrieval model [5], a resource is represented as a set of (possibly weighted) words and it is relevant for the query if it contains all the words from that query. Various morphological analyses can be performed (e.g. stemming words and counting their distribution) in order to abstract different syntactical forms of a word. However, the semantics of the words is completely omitted: for example, (i) a document about *cars* will not be retrieved for the query about *vehicles* unless the word *vehicle* appears explicitly in that document, and, (ii) a document about *cars* that uses specific terminology (e.g. the term *automobile* instead the term *car*) is not relevant for any query about *cars*. Therefore, the retrieval process should go beyond the syntax and structure of resources and focus on the meaning of their content. This is a well-elaborated problem in the traditional IR, known as the "prediction game": the user tries to predict which terms the provider of information has used in describing the content of a resource and the provider tries to predict which terms the user will use in search for that resource [6].

Logical IR models were studied to provide a rich and uniform representation of information and its semantics, with the aim to improve retrieval effectiveness. Although the interest in

logic of IR researchers may be traced back at least to the early 70's [32], the first clear statement that IR should be understood in logical terms is due to van Rijsbergen [102]. The earliest approaches were directed to the use of classical logic, like Boolean logic.

The basis of a logical model for IR is the assumption that queries and resources can be represented effectively by logical formulas. In order to retrieve a resource, an IR system has to infer the formula representing the query from formulas representing the resource. This logical interpretation of query and resources emphasises that information retrieval is an inference process that computes whether a resource r is relevant to a query q using both information present in the resource itself and external knowledge, like for example background knowledge about the domain. An example is given in classical logic where inference is often associated with logical implication: a resource r is relevant to a query q if formula $r \rightarrow q$ is satisfied[6], where r and q are formula of the chosen logic and " $\rightarrow$ " denotes the brand of logical implication formalized by the logic in question.

Such a query evaluation formally embodies the semantics of the information represented in the query and in the resources. For example, by defining a logic formula stating that a car is a type of vehicle, the inference process will retrieve a document about cars against a query for vehicle.

This way of viewing IR is especially fascinating when the model-theoretic semantic level of the logic is considered. In that sense, the logical approach to IR which amounts to sanctioning that relevance coincides with (set-)inclusion of information content, or semantics: only resources whose information content includes that of the information need of the user are to be retrieved.

The logical models of IR are believed to be more general than classical ones so that they are able to represent within a uniform framework various features of IR systems, like structured multimedia objects [145], user's knowledge [83], the nature of IR agents [61].

In this work we focus on the following four advantages of using the logic-based view on the information retrieval problem:

(1) The quality of the information retrieval process (i.e. precision and recall) increases,

(2) The representation of documents and queries is based on logic, thus, knowledge can be added and combined in form of facts and rules as it becomes available; we achieve a powerful and modular representation model,

(3) The description of the retrieval process as logical implication enables the integration of IR with deductive databases, and,

(4) Finally, logic makes it possible to reason about an IR model and its properties. This latter possibility is becoming increasingly important since the conventional evaluation methods, although good indicators of the effectiveness of IR systems often give results which cannot be predicted or satisfactory explained.

Here we demonstrate the first advantage. Other advantages will be discussed in next sections.

The advantages of the logic-based IR regarding the effectiveness of the retrieval process can be demonstrated through the following statements:

1. IF $r \rightarrow q$ THEN *about*(r, q), i.e. if the inference mechanism is used to drive retrieval, then it is certain that q holds in all of these (information) resources, of which r is a model. In other words, all the retrieved resources are guaranteed to be about q. It means that if the inference mechanism is sound, then the retrieval process has the perfect precision (=1).

[6] In the next subsection we will elaborate this condition in detail.

2. IF *about*(r, q) THEN $r \rightarrow q$, i.e. if a resource is about a query, then the inference mechanism can discover that fact. It means that the perfect recall (=1) of the retrieval process depends on the completeness of the inference process.

2.2.2 Challenges

However, adopting a logical view on the information retrieval problem introduces some challenges for the formal nature of logic. Here we discuss the most common challenges.

2.2.2.1 Paradoxes of Material Implication

A number of researchers have recalled that material implication suffers from idiosyncratic behaviour, resulting in what is known as "the paradoxes of material implication"[7] and have implied that this renders material implication an unsuitable starting point in the attempt to model relevance of information resources to information needs. The classical example is the counter-intuitive flavour once "$\rightarrow$" is interpreted as "if... then..." in the following formulae that are valid in classical propositional logic:

(a) $\alpha \rightarrow (\beta \rightarrow \alpha)$

(b) $\neg\alpha \rightarrow (\alpha \rightarrow \beta)$

In the following we show briefly that these paradoxes can be of importance only when "pathological" resources and queries are considered, i.e. in the case of a "bad" modelling of real-world IR problems [115].

In IR terms, schema (a) is pertinent to the case of queries represented by valid formulae. In fact, from (a) and modus ponens we have that, if q is a valid formula then $r \rightarrow q$ is also valid: In IR terms this means that any resource will be deemed relevant to a query represented by a valid formula. However, note that a valid query q is represented by formulae such as ($p \vee \neg p$) (or its logical equivalents), which corresponds to a request to retrieve all resources that are "*either about p or are not about p*". Obviously, all the resources from the repository will be retrieved for such a query, as a query such as ($p \vee \neg p$) asks exactly for this. It is clear from this example that valid formulae are representations of "pathological" queries: any "normal" query will be represented by a formula that is neither valid nor unsatisfiable.

Similarly, schema (b) is pertinent to the case of resources represented by unsatisfiable formulae. In fact, from (b) and from modus ponens we have that, if $\neg r$ is a valid formula (it means r is unsatisfiable) then $r \rightarrow q$ is also valid: In IR terms it means that a resource represented by an unsatisfiable formula will be deemed relevant to any query. However, an unsatisfiable resource is represented by formulae such as ($p \wedge \neg p$) (or its logical equivalents), which asserts that a resource "*at the same time is about p and is not about p*". It is also clear from this example that unsatisfiable formulae are representations of "pathological" resources: as for information needs, any "normal" resource will be represented by a formula that is neither valid nor unsatisfiable.

2.2.2.2 Status of $r \rightarrow q$

A comparative analysis of the logic-based IR research reveals that it is far from clear what the logical status of the $r \rightarrow q$ formula should be, in order to indicate that the resource represented by r is relevant to the information need represented by the query q. In particular, researchers seem to take different stands regarding which among the following facts should indicate this:

[7] There are two paradoxes of material implication. Both are evident from its truth-table column:
1) Whenever the antecedent is false, the whole conditional is true
2) Whenever the consequent is true, the conditional is true

1. $r \rightarrow q$ is *true* in some particular interpretation of the chosen logic L,
2. $r \rightarrow q$ is valid in L,
3. r is a *logical consequence*[8] of q in L.

We discuss these interpretations in short.

Truth

Truth is not a suitable notion to base a logical model of a real-world phenomenon on. In the IR case, this may be seen by noting that, if one wanted to model the relevance of the resource represented by r to the information need represented by query q with the truth of $r \rightarrow q$, one should also specify in which interpretation the truth of $r \rightarrow q$ has to be evaluated. A formula cannot be true per se, simply because truth is not a property of formulae, but a binary relation between formulae and interpretations. Obviously, one can take "*the interpretation that corresponds to the real world*" (i.e., the interpretation in which the sentence "Karlsruhe is in Germany" evaluates to true, "Birds are mammals" evaluates to false, and so on). But what is then the truth value to which "The number of water molecules in my glass is even" should evaluate? The answer is that we do not have a clue to what the "interpretation that corresponds to the real world" is, because our knowledge of the real world (or, more to the point, of our domain of discourse) is partial, and often fallacious too, even in more mundane matters than those of molecular structure.

Validity

The key observation is that valid formulae are true (in any interpretation) by virtue of their form, and not by virtue of their content. Because of this, it is not needed to have a grasp on the real world (i.e., to know which interpretation corresponds to it) to assess the validity of a formula (in the IR case: to assess whether a resource should be retrieved or not); one only needs to perform a purely symbolic check of the formula itself. For instance, the formula of propositional logic:

$$\text{John-likes-football} \vee \neg \text{John-likes-football} \qquad (1)$$

is true in any interpretation (i.e., valid). In order to assess its validity it is not necessary to know whether in the interpretation that corresponds to the real world (or in any other particular interpretation, for that matter) John actually likes football or not; it is instead sufficient to apply the well-known syntactic rules for validity checking in propositional logic.

The formula:

$$\text{John-is-a-man} \rightarrow \text{John-likes-football} \qquad (2)$$

instead, may well be true in the interpretation that corresponds to the real world, but is false in others, hence it is logically not very interesting.

One might argue that (2) is at least an informative formula ("it says something"), and that valid formulae like (1), being tautologous, carry no information content.

However, in the IR case it is not formula $r \rightarrow q$ that carries information to us; it is the very fact that it is (or that it is not) valid, as it informs us whether we should retrieve the document or not. For instance, if we cast the Boolean model of IR in terms of the validity of formula $r \rightarrow q$ in propositional logic, the fact that formula $(p_1 \wedge p_2 \wedge p_3) \rightarrow (p_1 \vee p_2)$ is valid informs

[8] We recall that: 1) formula α is a logical consequence of a set of formulae Γ when α is true in all the interpretations in which all the formulae in Γ are true; 2) formula α is valid when it is true in all interpretations, i.e., when it is a logical consequence of the empty set.

us that a document indexed by terms p_1, p_2 and p_3 should be retrieved as a result of the information need represented by $p_1 \vee p_2$.

This notion of "meta-level informativeness" is, of course, task-oriented, and the task here is information retrieval.

In fact, only the formulae of the type $r \rightarrow q$ are interesting in the IR case. The validity or non-validity of formulae $r \vee q$ are out of interest, since such a formula has no information content, as it informs of the validity of a formula that is not meant to represent relevance of resources (r) to information needs (q).

Logical Consequentiality

As in the case of the validity, logical consequentiality is a "form-based" notion, i.e. logical consequentiality may be established formally (and thus effectively), by symbolic manipulation only.

One aspect on which validity scores better is that logical consequentiality would seem a bit unsuitable for extending the "$r \rightarrow q$ model" to a "$P(r \rightarrow q)$ model" (see next subsection), i.e., to a model which can express the probability of relevance in terms of formula $P(r \rightarrow q) = p$ (to be read "the probability that r implies q is p"). Given a suitable logic one can express formulae such as $P(r \rightarrow q) = p$, and define appropriate notions of truth and validity for them.

One aspect on which logical consequentiality gets instead a better mark is its more intuitive character. It is quite intuitive, in fact, that relevance of a resource to an information need is a consequence of the semantic content of the resource and of the information need, and possibly of other factors such as the meaning (as specified e.g., in a thesaurus) of the terms involved.

2.2.2.3 False Document Problem

Propositional logic - and classical logic in general - suffers from the problem that a so-called "false document" (i.e., a hypothetical resource which is "about nothing" - a totally uninformative document (TUD)) is deemed relevant to any information need, and this is obviously unsuitable.

In fact, suppose our term language consists of a set of propositional letters $P = \{p_1, p_2, p_3, p_4\}$, so that a TUD is represented by the formula : $\neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4$; quite obviously, in propositional logic formula $(\neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4) \rightarrow \alpha$ is true for any propositional formula α in at least one interpretation, i.e., in the interpretation that makes all propositional letters in P true (since the antecedent is false, see Footnote 5).

However, this problem does not appear in the logic models based on validity and logical consequence. It is easy to show that, if validity is adopted, no "false document problem" arises: in the case of a TUD, $r \rightarrow q$ will not be valid, i.e., there will be at least one assignment of truth values to the propositional letters for which $r \rightarrow q$ will be false. For these resources, it may well be that, among the assignments that make $r \rightarrow q$ true, there is some assignment that makes r false and q true, and hence makes the implication $r \rightarrow q$ true; but this need not worry us, as long as the implication is not valid.

Similarly, this assignment of truth values corresponds to a false consequence, so that there is one interpretation where premise is true but the consequence is false. Therefore, in a logical model based on logical consequence the false document problem does not exist, as well.

2.2.2.4 Open vs. Closed World Assumption

In the logical view on databases, the *closed-world assumption* is employed in order to make false all the facts that cannot be derived from the set of facts stored. That is, from $\neg (d \rightarrow q)$ we infer $d \rightarrow (\neg q)$. For example, when we know that Jones and Smith are the authors of

document d_1, asking the system if Miller is an author of d_1 will be answered with 'no', based on the assumption that the system knows all the authors of d_1, and Miller is not among them.

For IR queries, the situation is different: Assume that the semantic representation of d_1 consists of the terms *audio*, *video* and *retrieval*. Now a user query for documents about multimedia retrieval, i.e. $q = multimedia \wedge retrieval$ should not yield false in the case of d_1, although $\neg (d_1 \rightarrow q)$. Rather, we should assume an *open world* due to the fact that content representations are always incomplete. Vice versa, d_1 would also not be an answer when search for documents that are not about multimedia retrieval.

This view is in contrast to the Boolean retrieval, which employs the closed world assumption. Principally, for queries without negated terms, open world and closed world lead to the same result. Only when the query terms are negated, the open and closed world assumptions lead to different query results. The open world assumption would retrieve only those documents where the negated term is explicitly assigned in the negated form to a document (e.g. "*this document deals with audio and video retrieval, but not with multimedia"*).

Finally, the open world assumption may turn out to be extremely inconvenient when retrieving information resources. An incomplete information repository may not entail a statement α nor its negation $\neg\alpha$. The reason for this is that an indexer who indexes a repository, in order to obtain the desired behaviour, must specify not only what documents are about, but also what they are not. This usually amounts to an overwhelming number of negative assertions. This can lead to a conclusion that, under certain circumstances, an IR system should adopt a closed-world view of the underlying repository, using the inability of establishing a fact as evidence of the contrary. In fact, assuming that if keyword *t* does not feature in the representation of document *d*, meaning that *d* is not about *t*, is equivalent to assuming that we have total knowledge of the document (for every keyword *t*, we know whether or not the document is about *t*). In other words, the total knowledge assumption may be characterised by saying that, for every sentence α of the language, either α or $\neg\alpha$ is a logical consequence of what we know about the domain. If one adopts the Boolean model of IR, one way of making this assumption is adopting the closed world assumption.

However, the main problem in adopting the logical view on IR is the need to deal with some notion of uncertainty/relevance. The classical implication relation does not take into account the relevance of the premises to the conclusions, which has been essential to the whole IR task. This is the topic of the next subsection.

2.2.3 Relevance

Relevance is one of the most important concepts in the theory of IR. The concept arises from the consideration that if the user of an IR system has an information need, then some information stored in some resources in the information repository may be "relevant" to this need. In other words, the information to be considered relevant to a user's information need is the information that might help the user to satisfy his information need. Any information that is not considered relevant to a user's information need is to be considered "irrelevant" to that information need. This is a consequence of accepting a dichotomous concept of relevance.

Therefore, given a set of information resources and a query, the task of the retrieval process is to retrieve those resources, and only those whose information content is relevant to the information content of the query (aka user information need). Moreover, we have already seen the importance of the ranked retrieval for the quality of the retrieval process (see Footnote 5: the Probability Ranking Principle).

The importance of relevance is the main reason why the logical formalisation of information retrieval is a non trivial problem:

- first, in determining the relevance of a resource to a query, the success or failure of an implication relating the two is not enough. It is necessary to take into account the uncertainty inherent in such an implication,
- second, the introduction of uncertainty can also be motivated by the consideration that a collection of resources cannot be considered as a consistent and complete set of statements. In fact, resources in the collection could and often do contradict each other in any particular logic and not all the necessary knowledge is available, and,
- finally, what is relevant is decided by the user from session to session and from time to time, and is then heavily dependent on judgments where highly subjective and scarcely reproducible factors are brought to bear.

A logical definition of relevance was considered for the first time in the context of IR by [32], who defined it as a "logical consequence". To make this possible both queries and documents need to be represented by sets of declarative statements. The statements representing the query are called "component statements". A subset of the set of stored sentences is called a "premise set" if and only if the component statement is a logical consequence of that subset. The definition says:

> "A stored sentence is logically relevant to (a representation of) an information need if and only if it is a member of some minimal premise set of stored sentences for some component statement of that need."

This definition of relevance is essentially just a proof-theoretic notion that has been generated to be applicable to information needs involving more than one component statement. Moreover, Cooper also tried to tackle the problem of having "degrees of relevance".

An early common belief was that the logical implication needed to capture relevance was not the classical material implication. We give here the two most often used arguments to dismiss the suitability of classical material implication for IR, taken from [33] and [115] respectively:

- The first argument relates to the fact that the truth of material implication $d \rightarrow q$ is to be determined relative to a particular evaluation situation. To determine the truth of $d \rightarrow q$ we have to compare the truth of d with that of q. Due to the paradox of material implication, when d is false, $d \rightarrow q$ will always be true. Herein lies the problem: in fact d is true only when d is retrieved, but given a retrieval situation in which q is submitted, a document d is always false since it has not been retrieved yet. Therefore, the real retrieval situation corresponds to the case of d false and such a document is relevant to any query q.
- The second argument criticises classical logic on the account that they license, as theorems of the pure calculus, sentences that suffer from *fallacies of relevance*. In other words, some conditional sentences are theorems of the given logic *even if their premise is not relevant to their conclusion*. For instance, sentence $(\alpha \rightarrow (\beta \rightarrow \alpha))$ (asserting that a true proposition is implied by any proposition) is a theorem of classical logic. And this should strike one as peculiar, in that the fact that β holds does not have any "relevance" to the fact that α holds. Therefore, in order for any conditional notion "$\rightarrow$" to be adequate, a sentence such as $\alpha \rightarrow \beta$ should be valid only if there is some connection of meaning between α and β (and this consideration should strike the document retrieval theorist as familiar). To the surprise of many logicians who considered these issues to more properly belong to rhetoric rather than logic, the idea of a "connection of meaning between α and β" (or, more generally, the idea of α being *relevant* to β) has been shown to be amenable to formal treatment by a circle of logicians who defined a class of logical calculi called *relevance* (or *relevant*)

> *logics* [115]. Relevance logics attempt to formalise a conditional notion in which relevance is of a primary concern. By doing this, they challenge classical logic and its extensions in a number of ways, *i.e.* by introducing a new, non truth-functional connective into the syntactic apparatus of classical logic, by rejecting some classical rules of inference for classical connectives, and by changing the notion of validity itself by "wiring" into its considerations of relevance. Note that this issue is different from the discussion we presented in 2.2.2.1, since we discuss here not the validity but rather the relevance of a logic implication.

The idea that a non-classical form of logical implication was needed for defining relevance was proposed by van Rijsbergen [103] in the form of the *Logical Uncertainty Principle*, where the connection between logic and IR modelling was for the first time explicitly made:

> "Given any two sentences x and y; a measure of the uncertainty of $y \rightarrow x$ relative to a given data set, is determined by the minimal extent to which we have to add information to the data set, to establish the truth of $y \rightarrow x$."

However, it is now clear that it is not possible to apply the logical uncertainty principle without a combination of a (non-classical) logic formalism and uncertainty theory. The process of adding information is represented by logic, whereas the involved uncertainty is modelled by a theory of uncertainty.

Moreover, van Rijsbergen argued that an approach combining conditional reasoning and reasoning about imprecision should be used, leading to the determination through logical inference of $P(r \rightarrow q)$[9] ("the probability that r implies q") as an estimation of the probability of relevance of resource r to query q, where $P(\alpha)$ is to be read as "the probability of α".

The rationale of using conditional reasoning is clear: if we were able to give to a resource and a query *perfect* representations r and q of the information content that the user attributes to them, the resource retrieval could be seen just as the task of establishing the validity of the formula $r \rightarrow q$ in a logic in which "→" mirrors "information containment".

The rationale of using reasoning about imprecision is also clear: such perfect representations cannot be obtained because of the above-mentioned elusive character of relevance, and the system can then only make a subjective estimation of how likely it is that the user will deem the resource relevant to his information need. In the next section we discuss this reasoning in more details.

In this research we consider addition of imprecision on top of a calculus for conditional reasoning as a "correction factor" for bridging the gap between the rigidity of logical calculi and the flexible, human-centred notion of relevance. In principle, it allows to fine-tune the system estimation of relevance as a function of contextual factors, user preferences and so on. Moreover, in order to arrive at a successful logical model of document retrieval, some effort should be made in order to wire as much relevance as possible into the implication connective, i.e. to design a calculus for (without imprecision) conditional reasoning where the factors that influence relevance, as perceived by the user, are taken into account.

[9] Thus, probabilistic IR can be interpreted as estimating the probability $P(r \rightarrow q)$ that the document logically implies the query. Rijsbergen pointed out that this probability should not be considered in the traditional sense, i.e. $P(r \rightarrow q) \neq Pr(\neg r \wedge q)$, but as the conditional probability $P(q|r)$.

2.2.4 Logical-Uncertainty Models of Information Retrieval

Logical-Uncertainty Models (sometimes referred to as uncertain inference models) are based on an uncertainty theory (for instance, probability theory, semantic theory, imaging) that is defined on a logical basis[10]. They enable more complex definitions of relevance than other IR models (for instance, probabilistic relevance models which are based mainly upon statistical estimations of the probability of relevance). With logical-uncertainty models, information not present in the query formulation may be included in the evaluation of the relevance of a document. Such information might be domain knowledge, knowledge about the user, user's relevance feedback, and so on.

Another characteristics of logical-uncertainty models is that they are not as strongly collection-dependent as most other IR models. In most IR models, parameters (e.g., normalisation and weight combination parameters) are only valid for the current collection, while logical-uncertainty models can use knowledge of the user or the application domain that can be useful with many other collections.

In this section we present the logical-uncertainty model relevant for this research. A detailed discussion of other models relevant for IR can be found in [33].

2.2.4.1 Inference Network

The Bayesian or Inference networks are *directed acyclic graphs* the nodes being random variables of a problem to be solved (here: finding relevant information) and the arcs the causal relationships among them [152]. Therefore, each set of arcs into a node represents a probabilistic dependence between the node and its parents (the nodes at the other ends of the incoming arcs). A Bayesian network represents, through its structure, the conditional independence relations among the variables in the network. These independence relations provide a framework within which to acquire probabilistic information.

The graphs represent knowledge (a) qualitatively, showing the (in)dependencies among variables, and (b) quantitatively, expressing the strength of these dependencies by means of conditional probability distributions. For each node, the parents of that node reflect all its direct causes given by a respective set of conditional probability distributions. If a proposition represented by a node p "causes" or implies the proposition represented by node q, a directed edge from p to q is drawn in the graph. Node q contains a *link matrix* that specifies $P(q \mid p)$ for all possible values of the two variables. When a node has multiple parents, the link matrix specifies the dependence of that node on the set of parents and characterizes the dependence relationships between that node and all parent nodes. Given a set of prior probabilities for the roots of the DAG, these networks can be used to compute the probability or a degree of belief associated with all the remaining nodes.

The Bayesian network based information retrieval models use this network structure in order to estimate the relevance of an answer for the query. Indeed, they associate index terms, documents, user queries and/or user's information needs with random variables to model the retrieval task as an evidential reasoning process. In this context the observation of a document, index or query term is considered to be the cause for an increased belief in the respective variable that is further propagated throughout the network. The basic model, shown

[10] In IR probabilistic reasoning refers to the use of a model that ranks documents in decreasing order of their evaluated probability of relevance to a user's information need. Past and present research has made use of formal theories of probability and statistics in order to evaluate or at least estimate those probabilities of relevance. These attempts are to be distinguished from looser ones like, for example, the vector space model in which documents are ranked according to a measure of similarity within the query.

in Figure 2.3, consists of a document network representing (it is built once) the static knowledge on the overall text corpora and a query network that is built according to the user's query and can be modified and extended during each session by the user in an interactive and dynamic way. The query network is attached to the static document network in order to process a query.

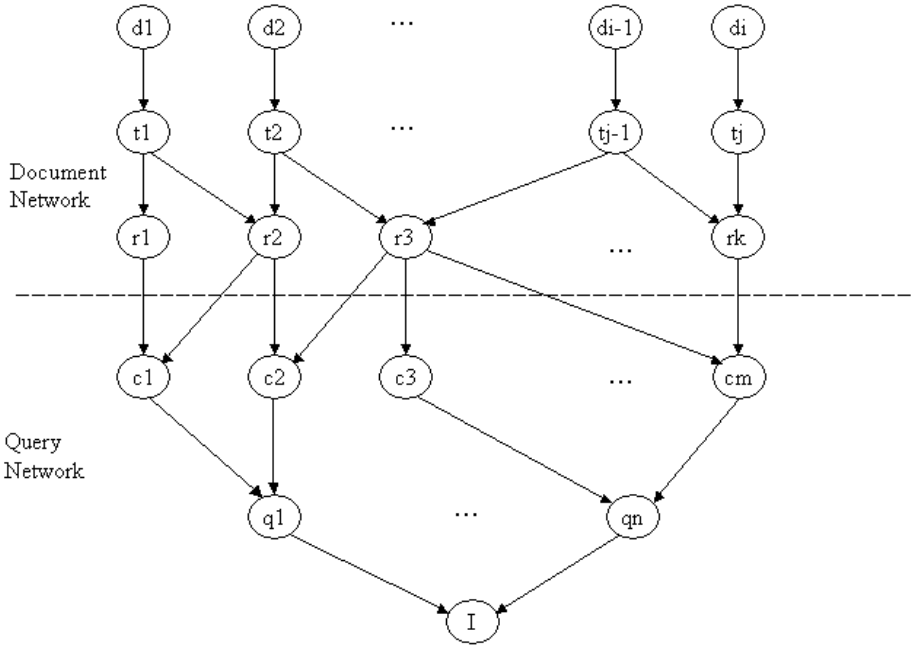


Figure 2.3: Basic inference network for IR

2.2.4.1.1 Document Network

The document network consists of (consider Figure 2.3) document nodes (d_i's), text representation nodes (t_i's) and concept representation nodes (r_i's). Each document node represents an actual document in the collection. A document node corresponds to the event that a specific document has been observed. The form of document represented depends on the collection and its intended use. The document nodes correspond to abstract documents rather than their physical representations. A text representation node or text node corresponds to a specific text representation of the document. A text node corresponds to the event that a text representation has been observed. The network model can support document nodes with multiple children representing different representation types (e.g. figures, audio or video). Similarly, a single text might be shared by more than one document. The shared components are rare in traditional collections (an example could be a journal article that appears in both a serial issue and in a reprint collection) and are not generally represented in current retrieval models.

The basic document network is a simple three level directed acyclic graph (DAG) in which document nodes are roots, text nodes are interior nodes and representation nodes are leaves. A document node has exactly one text node as a child and each text node has one or more representation nodes as children.

Each document node has a prior probability associated with it, that describes the probability of observing that document. This probability is generally set to 1/(collection size). Each text node contains a specification of its dependence upon its parent; by assumption this dependence is complete, a text node is observed (t_i = true) exactly when its parent document is observed (d_i =true).

Each representation node contains a specification of the conditional probability associated with the node given its set of parent nodes. This specification incorporates the effect of any indexing weight (e.g. term frequency) or term weight (e.g. inverse document frequency) associated with the representation concept.

2.2.4.1.2 Query Network

The query network is an "inverted" DAG with a single leaf that corresponds to the event that an information need (I) is met and multiple roots that correspond to the concepts that express the information need. As shown in Figure 2.3, a set of intermediate query nodes (q_i's) may be used when multiple queries express the information need. Indeed, an information need can be represented through several queries – it is unlikely that any of these queries will correspond precisely to the information need, but some will better characterize the information need than others and several query specifications taken together may be a better representation than any of the individual queries. In other words, it is an attempt to make the meaning of an information need explicit by expressing it in the form of one or more queries that have formal interpretations.

The roots of the query networks are query concepts (c_i's). They correspond to the primitive concepts used to express the information need. A single query concept node may have several representation concept (r_i's) nodes as parents. Each query concept node contains a specification of its dependence on the set of parent representation concepts. The query concept nodes define the mapping between the concept nodes used to represent the document collection and the concepts used in queries. In the simplest case, the query concept nodes are the same as the representation concepts so each query concept has exactly one parent.

A query node represents a distinct query form and corresponds to the event that a query is satisfied. Each query node contains a specification of the dependence of the query on its parent query concepts. The link matrices that describe these conditional probabilities depend on the query type – a link matrix simulating a Boolean operator is different from a matrix simulating a probabilistic or weighted query.

A single leaf representing the information need corresponds to the event that an information need is met. The query network is intended to capture the way in which meeting the user's information need depends on documents and their representations. Moreover, the query network is intended to allow the combination of information from multiple document representations and to combine queries of different types to form a single, formally justified estimate of the probability that the user's information need is met. If the inference network correctly characterizes the dependence of the information need on the collection, the computed probability provides a good estimate.

2.2.4.1.3 Use of the Inference Network

The retrieval inference network is intended to capture all of the significant probabilistic dependencies among the variables represented by nodes in the document and query networks [50], [152]. Given the prior probabilities associated with the documents (roots) and the conditional probabilities associated with the interior nodes, the posterior probability or belief associated with each node in the network can be computed.

The truth value of a node depends only upon the truth value of its parents. To evaluate the strength of an inference chain going from one document to the query, document node d_i is set to "true" and $P(q_k = \text{true} \mid d_i = \text{true})$ is calculated. This gives an estimate of $P(d_i \rightarrow q_k)$.

Further, if the value of any variable represented in the network becomes known, the network can be used to re-compute the probabilities associated with all remaining nodes based on this evidence.

It is possible to implement various traditional IR models on this network by introducing nodes representing Boolean operators or by setting an appropriate conditional probability evaluation function with nodes.

Link Matrix Forms

For all non-root nodes in the inference network we must estimate the probability that a node takes a value given any set of values for its parent nodes. If a node has a set of parents $\pi_a = \{p_1, \ldots, p_n\}$, then $P(a \mid p_1, \ldots, p_n)$ must be estimated.

The most direct way to encode our estimate is as a link matrix. By dealing with binary valued propositions, this matrix is of size 2 x 2^n for n parents and specifies the probability that variable a takes the value $a = true$ or $a = false$ for all combinations of parent values. The update procedures for Bayesian networks then use the probabilities provided by the set of parents to condition over the link matrix values to compute the predictive component of our belief in a or P($a = true$). Similarly, the link matrix is used to provide diagnostic information to the set of parents based on the belief in a.

The canonical link matrix forms for the Boolean operators regarding the node Q with n parents ($A_1, \ldots A_n$) look like [152]:

OR: $P(Q = true) = 1 - (1 - a_1)(1 - a_2)\ldots(1 - a_n)$

AND: $P(Q = true) = a_1 a_2 \ldots a_n$

NOT: $P(Q = true) = (1 - a_1)$

where $P(A_k = true) = a_k$.

If the parent nodes for any of these logic operators are restricted to values 0 or 1 then Q must also have a value of 0 or 1. If the terms can take on weights in the range [0, 1] and these weights are interpreted as the probability that the term has been assigned to a document text, then these inference networks provide a natural interpretation of the Boolean retrieval with weighted indexing.

For probabilistic retrieval each parent has a weight associated with it, as does the child. In this weighted-sum matrix, the belief in Q depends on the specific parents that are true; parents with larger weights have more influence in our belief. If we let $\varpi_a, \varpi_b, \varpi_c \geq 0$ be the parent weights for the parents a, b, c and $0 < \varpi_q < 1$ the child weight, and $t = \varpi_a + \varpi_b + \varpi_c$ then we have:

$$P(Q = true) = (\varpi_a \cdot a + \varpi_b \cdot b + \varpi_c \cdot c) \cdot \varpi_q / t.$$

2.3 An approach for Ontology-based Information Retrieval

2.3.1 Motivation

The logic based IR provides a sound platform to reason about the meaning of an information resources' content in the retrieval process, i.e. about the relevance of that meaning for the user's information need. In that way, the user can find resources that are relevant for his query even if there are no syntactical similarities between them. It is clear that the quality of the retrieval depends on the quantity and quality of the domain knowledge that is available to the reasoning process. Indeed, a logical system can retrieve a document about cars for a query for vehicle, if and only if there is a formally described statement that a car is a type of vehicle. Therefore, in order to enable retrieval of all semantic relevant resources for a query, the knowledge about domain has to be systematically acquired and described in the form of a domain theory. Moreover, in order to resolve the "prediction problem," the domain theory has to be commonly shared, i.e. a kind of common agreement about the used vocabulary should

exist. Since ontologies represent explicit and formal specifications of the conceptualisation of a domain of interest, they seem to be very suitable for the extension of the logic-based IR systems in the above mentioned way. This will be the topic of the next sections. Figure 2.4 illustrates these extensions of the basic IR process shown in Figure 2.1.

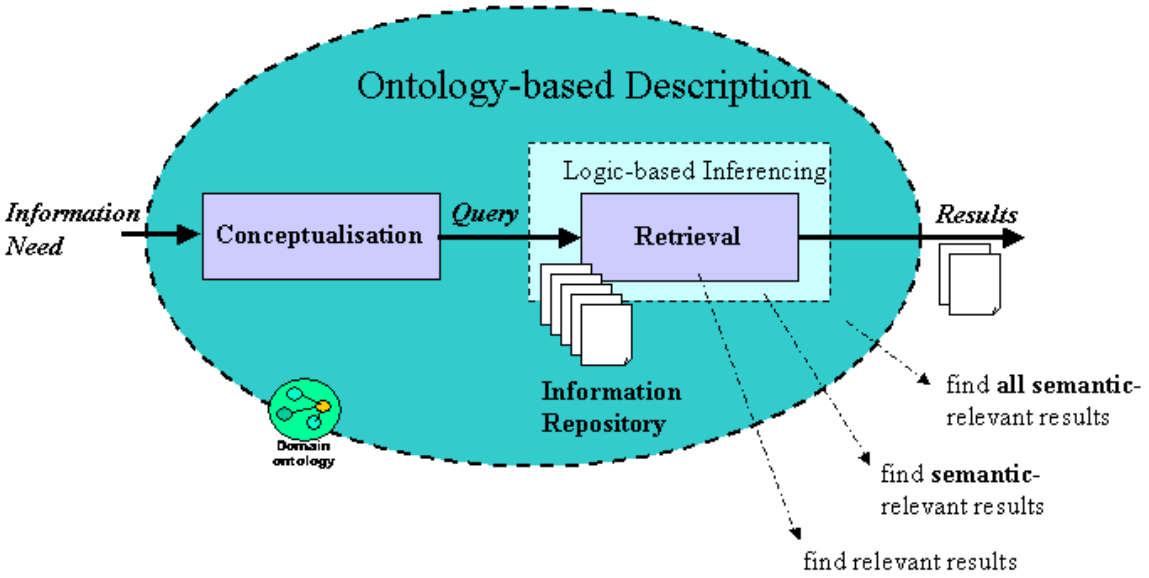


Figure 2.4: Two extensions of the basic IR process and their effects on the quality of results

2.3.2 Ontology

2.3.2.1 Introduction

The term ontology is borrowed from philosophy, where ontology is a systematic account of existence. For computer science, what "exists" is that which can be represented. Thus, in the context of computer science, the following definition is adopted [55]:

Definition 2.1: *Ontology* is a *formal*, *explicit* specification of *a shared conceptualisation* of a *domain* of interest.

Conceptualisation is an abstract, simplified view of the world that we wish to represent for some purpose. An ontology has set out to overcome the problem of implicit and hidden knowledge by making the conceptualisation of a domain explicit. It is used to make assumptions about the meaning of a specific concept. It can also be seen as an explication of the context for which the concept is normally used.

Moreover, everything (i.e., any knowledge-based system or any knowledge-level agent) is liable to some conceptualisation, explicitly or implicitly. Therefore, since there is consensus of terms, it is ***a shared conceptualisation***.

Next, the purpose of ontology is not to model the whole world, but rather a part of it - a so-called domain. A ***domain*** is just a specific subject area or area of knowledge, like medicine, tool manufacturing, real estate, automobile repair, financial management, etc. Therefore, in order to define a domain, it is important to know what ontology is *for*.

Further, ontology serves as a means for establishing a conceptually concise basis for communicating knowledge for many purposes. In order to achieve this, ontology has to be a ***formal*** description of the meaning of concepts and relationships between them. Therefore, the formal specification means that ontology is specified by means of a formal language, e.g. first order logic.

Finally, this formal model is readable, understandable and processable not only for people, but also for machines. This is achieved through the ***explicit*** specification, while there is formal semantics of all statements, i.e. the semantics of the used language is formally specified as well. Therefore, ontologies have to be specified in a language that comes with formal semantics. Only in this way a detailed, accurate, consistent, sound, and meaningful description can be made.

The study of ontologies and their use is no longer just one of the fields in the computer science literature. Ontologies are now ubiquitous in many enterprise-wide information-systems: they are used in e-commerce, knowledge management and in various application fields such as bioinformatics and medicine. Moreover, they constitute the backbone for the Semantic Web.

Ontologies have been gaining interest and acceptance in broader audiences. [57] provides a nice collection of fields that embrace ontologies including knowledge engineering, knowledge representation, qualitative modelling, language engineering, database design, information retrieval and extraction, and knowledge management and organization.

2.3.2.2 Formal Definition: Karlsruhe View of Ontologies

There are a lot of formal definitions of ontology. We are using the Karlsruhe (KA) View of ontologies [146], since it underpins most of the tools we used for implementing this research.

Let $\Re$ and N denote real and natural numbers, respectively. All subscripts are in N, 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 2.2: *Core 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. The notion $H(a,b)$ can be used for depicting this partial order,
- 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 $\pi_i(\sigma(r_1)) \leq \pi_i(\sigma(r2))$ for each $1 \leq i \leq |\sigma(r_i)|$ and $\pi_i(a)$ is the function that retrieves i-th argument of the relation a.

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

Definition 2.3: *Domain and Range*

For a relation $r \in R$ with $|\sigma(r)| = 2$, we define its domain and its range by:

$$dom(r) := \pi_1(\sigma(r)) \text{ and } range(r) := \pi_2(\sigma(r)).$$

Definition 2.4: *Axioms*

Let L be a logical language. An 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*.

Axioms (or rules) are written in the form:

$$p_{i1}, p_{i2}, \ldots, p_{in} \Rightarrow q_{i1}, q_{i2}, \ldots, q_{ir}$$

where p_{ij}, $n \geq j \geq 1$ (the symbols of the body of the *i*-th axiom) and q_{ik}, $r \geq k \geq l$ (the symbols of the head) are atomic formula (also referred to as literals), consisting of a predicate (relation) applied to terms, which are either constants or variables. As usual, each p_{ij} is called a condition and each q_{ik} is called a conclusion. This is read declaratively as $p_{i1}, p_{i2}, \ldots p_{ir}$ implies $q_{i1}, q_{i2}, \ldots, q_{ir}$.

See next section for an example.

Definition 2.5: *Relation Properties*

We define three properties for the relations: reflexivity, symmetry and transitivity, which are defined by functions r_r, r_s, r_t respectively, in the following manner:

$$r_r : R \rightarrow R_r,\ r_s : R \rightarrow R_s,\ r_t : R \rightarrow R_t,$$

where R_r, R_s, R_t represent sets of reflexive, symmetric and transitive relations, respectively.

Definition 2.6: *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 instances,
- a function $l_r : R_{KB} \rightarrow I^+$ called relation instances.

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.

Let us denote that *KB*(*O*) is the set of all instances (facts) which can be proven in the given ontology *O*. Moreover, let *I*(*O*) denote the set of all concept instances which can be proven in given ontology *O*.

Definition 2.7: *Ontology-based query*

A (conjunctive) query is of the form or can be rewritten into the form:

$$Q(O) = forall\ \overline{X}\ \ \overline{P}(\overline{X}, \overline{k}) \qquad (3)$$

with $\overline{X}$ being a vector of variables $(X_1, ..., X_n)$, $\overline{k}$ being a vector of constants and $\overline{P}$ being a vector of conjoined predicates (relations). O is the given ontology.

For example, for the query:

"`forall x <- worksIn(x, KM) and researchIn(x, KMsystems)`"

we have:

$$\overline{X} := (x),\ k := (KM, KMsystem),\ \overline{P} := (P_1, P_2),\ P_1(a,b,c) := worksIn(a,b),$$
$$P_2(a,b,c) := researchIn(a,c).$$

A query can be viewed as an axiom without a head.

Since a predicate constrains the interpretation of a variable in a query, in the rest of the text we will use the term *query constraint* as the description of a predicate. For example, `researchIn(x, y)` is a constraint for the interpretation of variable x.

(3) is the standard form of queries considered in similar research [28], [53]. Moreover, our limiting focus to conjunctive queries is not a serious limitation since the result of a disjunctive query can be considered as the union of the results of the disjuncts; that is, each disjunct can be considered as an independent query.

Let us denote that $\Omega(O)$ is the set of all possible (conjunctive) queries for the ontology O.

Definition 2.8: *Answers (results) to an ontology-based query*

Let $\prod(O)$ be the set of all relation instances which can be proven in the given ontology O (this set can be obtained by the materialisation of all rules).

For a query "*forall* $\overline{X}$ $\overline{P}(\overline{X},\overline{k})$" an answer is an element (tuple) in the set

$$F(Q(O)) = \{\overline{A_i}\} = \{(a_1,a_2,...a_n)\},$$

such that $\overline{P}(\overline{A_i},\overline{k})$ is provable, i.e. each of the relation instances $r(a_1,a_2,...,a_n,k_1,...k_l), r \in \overline{P}$ exists in the set $\prod$.

2.3.2.3 Formal Representation: Frame Logic

As we have already mentioned, an ontology is a domain theory and it requires a logical mechanism to be represented in. Moreover, an ontology inherits from that logical theory the reasoning capabilities which are used in the information retrieval process. In this research we are using the Frame Logic (F-Logic) [70], a deductive, object oriented database logic which combines declarative semantics and expressiveness of the deductive database languages with the rich data modelling capabilities supported by the object oriented data model. It accounts in a clean and declarative fashion for most of structural aspects of object-oriented and frame-based languages. From the logic-point of view, the Frame Logic is inspired by frame-based systems. The concept of a *frame* was proposed in the 1970's [79], and frame systems subsequently gained ground as basic tools for representing knowledge [26], [46], [68]. The fundamental idea of a frame system is rather simple: A frame represents an object or a concept. Attached to the frame is a collection of attributes (slots), potentially having types (or value restrictions) and potentially filled initially with values. When a frame is being used the values of slots can be altered to make the frame correspond to the particular situation at hand. According to an interpretation by Minsky, the slots of a frame might represent questions most likely to arise in a hypothetical situation represented by the frame.

An adequate language for defining the ontology has to provide modeling primitives for all entities we presented in previous section. The Frame Logic already provides such modeling primitives and integrates them into a logical framework providing a Horn logic subset. The F-Logic has clear roots in object-oriented databases and logic programming.

The Frame Logic is a full-fledged logic. It has model-theoretic semantics and a sound and complete resolution-based proof theory. The Frame Logic provides object identity, complex objects, inheritance, polymorphic types, methods, encapsulation; it integrates these features into a logic-based framework. The syntax of the Frame Logic is higher-order, which, among other things, allows an integrated view of data and schema. Both can be manipulated and defined using the same declarative language. The Frame Logic does not specify basic types, everything is an object. No distinctions between attributes and associations are made; all relationships between objects are modelled by method applications.

For a detailed description of F-Logic, [70] should be consulted. The F-Logic is described here to the extent necessary for this research.

From a syntactic point of view the F-Logic is a superset of first-order logic [37].

To define the language we introduce the F-Logic alphabet consisting of a set of predicate symbols *P*, function symbols *F* and variables *V*. An id-term is a first-order term composed of function symbols in *F* and variables in *V*.

Let *o*, *m*, *r*, *c*, *c1*, *c2*, *t*, *Vi*, *Tj* be id-terms. A molecule in F-Logic is one of the following expressions:

o:*c* (instanceOf assertion) (Object *o* is an instance of the class *c*),

c1::*c2* (subclassOf assertion) (Class *c1* is a subclass of the class *c2*) ,

o[*m@v1*, …, *vn -> r*] (functional method application) This expression denotes that the result of the application of the single-valued method m with the arguments *v1*, …, *vn* on the object o is *r*. *n* (the number of parameters) might be zero – in this case the "@" sign is omitted,

o[*m@v1*, …, *vn ->>*{*r1*, … *rm*}] (functional method application) This expression denotes that the result of the application of the set-valued method m with the arguments *v1*, …, *vn* on the object o is the set {*r1*, …, *rm*}. If *m* = 1 it is also permissible to omit the parentheses in the result specification,

c[*m@v1*, …, *vn =>*(*T1*, …,*Tm*)] (single-valued signature-molecule) In the signature molecule the *Tj* are id-terms that represent the types of the results returned by the method m when m is invoked on an object of class c with arguments of types *vi*. The result of the method must be an instance of all the *Tj*,

c[*m@v1*, …, *vn =>>*(*T1*, …,*Tm*)] (set-valued signature-molecule). The case is analogous to the above.

Inside the square brackets it is permissible to have a multiple method application separated by an";". Every id-term can itself be an object, upon which other objects are applied.

p(*F1*, …, *Fn*) (predicate molecule) A predicate molecule (*P*-molecule) corresponds to an atom in First-Order predicate logic. Predicate symbol *p* is an element of *P*.

Example: The following expression defines a complex object with id *r1* as an instance of the class *Researcher* with two attributes *authorOf* and *cooperatesWith*. Both attributes are set-valued, but only one is defined. The value of the *cooperatesWith* attribute is a complex object with id *r2* and an attribute *authorOf*.

r1:*Researcher* and *r1*[*authorOf->>article1*] and *r1*[*cooperatesWith->>r2*] and *r2*[*authorOf->> article2*].

Table 2.1 contains the translation for the molecular expressions of F-Logic in the first order notion, taken from [37].

Table 2.1: Translation from object-oriented into first-order logic primitives

Object Oriented	First Order
C::D	sub(C, D)
O:C	instance(O, C)
O[M->>V]	method(O, M, V)
O[M=>>D]	methodtype(O, M, D)
O:C[M->>V:D]	instance(O, C) ∧ method(O, M, V) ∧ instance(V, D)

From the molecules, complex expressions may be built as follows:

- Facts are ground molecules,
- A rule consists of a head, the implication sign <- and the body. The head is a conjunction of elementary expressions (connected using AND). The body is a complex formula built from elementary expressions and the usual predicate logic connectives (implies:- >, equivalent <->, AND, OR and NOT). Variables are introduced in front of the head (with FORALL-quantifier) or anywhere in the body (using EXISTS and FORALL-quantifiers). A double rule is an expression of the form: head <-> body, where the head and body are conjunctions of molecules.

Figure 2.5 shows a small example ontology written in the Frame Logic. In this research only the shown features are used.

```
//Core Ontology
Researcher::Object.
Researcher[
familyname=>>String;
schoolID=>>School].

PhDStudent::Researcher.          // PhDStudent is sub class of Researcher

PostDoc::Researcher.             // PostDoc is sub class of Researcher
PostDoc [
position=>Position].

nenad: PhDStudent.               // Nenad is an instance of PhDStudent

School::Object[
reearcherID=>Researcher].

//Knowledge Base
AIFB : School.
nenad [familyname ->> stojanovic;  schoolID ->> AIFB].
// applying method "schoolID" to Nenad  yields the object "AIFB"

ljiljana:PostDoc[familyname->>stojanovic;schoolID->>FZI].
FZI : School.
andreas:PostDoc[familyname->>abecker;schoolID->>FZI].

//Axioms
// A sample Rule specifying that "schoolID"  is  the inverse of
"researcherID"
FORALL X, Y X:Researcher[schoolID->> Y] <-> Y : School[researcherID ->> X].

// A sample query, asking for all PhDStudents of AIFB  (the result is:
nenad).
FORALL X <- X:PhDStudents[schoolID ->> AIFB].
```

Figure 2.5: A simple ontology in F-Logic

2.3.3 Ontology-based Information Retrieval Model

2.3.3.1 The Retrieval Model

The ontology-based model for information retrieval redefines the task of IR as an extraction from a given repository of information resources, of those resources *r* that, given query *q*, makes the formula $O|\text{-}\ r \rightarrow q$ valid, where *r* and *q* are formulae of the chosen logic, "$\rightarrow$" denotes the brand of logical implications formalized by the logic in the question and *O* is a set of logical sentences called domain knowledge (ontology). A derivability relationship |- is

defined between a set of formulae and a formula, if there exists a finite sequence of the inference rules that leads the set of formula to that formula.

For the ontology-based IR, we have the following interpretation of the basic retrieval model presented in section 2.1:

- $L_{Res} = KB(O)$, i.e. a resource is modelled as a set of relation instances (facts) from the corresponding knowledge base. This set can be treated as one of instance assertions; then the relations (concepts) of which a fact is asserted to be an instance constitute altogether the description of the resource;
- $L_{Query} = \Omega(O)$, i.e. a query is modelled as an ontology-based query $Q(O)$; the intuitive meaning of this choice is that all resources represented by facts retrieved for query $Q(O)$, i.e. the set of facts $F(Q(O))$, should be retrieved;
- $IR = I(O) \subseteq L_{Res}$, i.e. a repository (collection) of information resources represents a set of all concept instantiations;
- $M(I(O), Q(O))$, the matching function between the repository and the given query, is implemented through logical inference defined by the logical language used for representing ontology O. For example, for the F-Logic [70] realization in the Ontobroker system [37], which we used in our research, it is the bottom-up fix-point evaluation procedure. It means that some mappings in M are defined implicitly through the axioms from set $A(O)$ (see Definition 2.4). They can allow the specification of lexical, "thesaural" knowledge as well, i.e. they contribute to the specification of the meaning of the terms used in both document representation and query formulation. In the inferring process this kind of knowledge is brought to bear (and thus serves as) "background knowledge" according to which queries are to be interpreted. The positive effect is that axioms are in fact a recall-enhancing mechanism (a mechanism that has no negative side-effect in terms of precision), because they support the discovery of resources relevant to the query that would have otherwise gone undetected.

Given a retrieval model, the interaction with a simple ontology-based retrieval system may be described as follows (see Figure 2.6).
The set of information resources and their properties is represented as a set of instances in the knowledge base $KB(O)$. A user's information need is conceptualised in an ontology-based query $Q(O)$. This query is matched against the set of information resources, $M(I(O),Q(O))$ and the set of answers $F(Q(O))$ is returned to the user.

2.3.3.2 Relevance

Since the ontology-based retrieval model uses the logic-based matching function, it benefits from the perfect precision and recall achieved in a logic-based retrieval, as we discussed in section 2.2.1. However, the ontology-based retrieval suffers from the problem of determining the relevance of the retrieval results, due to the crisp nature of the used logical implication. In other words, it is necessary to take into consideration the uncertainty inherent in this implication. As we already mentioned in section 2.2.3, this is one of the main challenges for the logic-based information retrieval and its resolution requires introduction of some aspects of uncertainty theory.

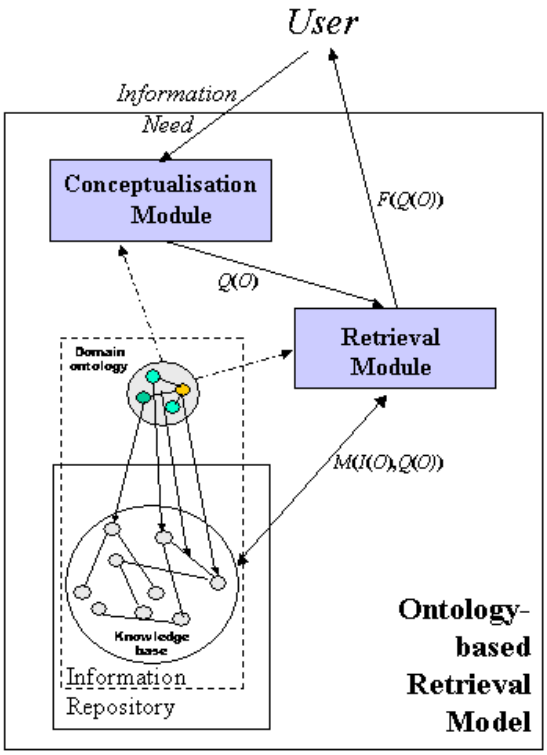


Figure 2.6: Basic ontology-based retrieval model

In order to make the discussion more understandable, we will first introduce an example. The example is taken from our evaluation study presented in section 2.4. In the rest of the text we will refer to it as the *Institute Example*.

Let us assume the following ontology:

```
1:  Project::Object[hasTopic =>> Topic].
2:  Lecture::Object[hasTopic =>> Topic].
3:  Topic::Object[subtopicOf =>> Topic].
4:  Researcher::Object.                                              (4)
5:  Professor:: Researcher.
6:  PhDStudent:: Researcher.
7:  Researcher [worksIn =>> Project; manages=>>Project; researchIn =>>
    Topic; teaches =>> Lecture].
8:  FORALL X,Y,Z   Z: Researcher [researchIn –>>Y] <– Z[worksIn ->>X] and
    X:Project[hasTopic –>>Y].
9:  FORALL X,Y,Z   Z: Researcher [researchIn –>>Y] <– Z[teaches ->>X] and
    X:Lecture[hasTopic –>>Y].
10: FORALL X,Y,Z    Z:Project[hasTopic –>>Y] <– X:Topic[subtopicOf –>>Y]
    and Z[hasTopic –>>X].
11: FORALL X,Y   X: Researcher [worksIn –>>Y] <– X[manages ->>Y] and
    Y:Project.
```

To give an intuition of the semantic of the F-Logic statements, in line 1, one finds a concept definition for a `Project` being an `Object` with a relation `hasTopic`. The range of the relation is restricted to `Topic`.

Ontology axioms as the one given in line 8 in (4) use this syntax to describe regularities in the domain. Line 8 states that if a `Researcher` *Z* works in a `Project` *X* and *X* has topic *Y*, then *Z* does research in *Y*. Let us further assume the following knowledge base:

```
12: rst:Professor.
13: gst:Professor.
```

```
14: meh:PhdStudent.
15: nst:PhdStudent.                                                    (5)
16: ysu:PhdStudent.
17: KM:Topic.
18: TextMining:Topic.
19: DataMining:Topic.
20: rst[worksIn->>OntoWeb; worksIn ->>DotCom; worksIn ->>OTK; manages->>
    OntoWeb].
21: ysu[worksIn->>OTK].
22: rst[teaches->>KnowledgeManagement].
23: nst[teaches->>KnowledgeManagement; worksIn->>OntoWeb].
24: gst[teaches->>KnowledgeManagement].
25: gst[teaches->>InfoA].
26: gst[worksIn->>OntoWeb; worksIn ->>DotCom; worksIn ->>SWAP].
27: meh[worksIn->>SWAP;teaches->>InfoB].
28: OntoWeb:Project[hasTopic->>KM;hasTopic->>TextMining;
    hasTopic->>DataMining].
29: WonderWeb:Project.
30: OTK:Project[hasTopic->>KM].
31: DotCom:Project[hasTopic->>KM].
32: SWAP:Project[hasTopic->>TextMining].
33: TextMining[subtopicOf->>KM].
34: DataMining[subtopicOf->>KM].
35: KnowledgeManagement:Lecture[hasTopic->>KM].
36: InfoA:Lecture.
37: InfoB:Lecture.
```

The definitions of instances in the knowledge base are syntactically very similar to the concept definition in F-Logic. In line 12 instance `rst` of the concept `Professor` is defined. Furthermore, in line 21, relation `worksIn` is instantiated between `ysu` and `OTK`. Similarly, in line 30 it is stated that `OTK` is a `Project` related to the topic `KM`.

Figure 2.7 illustrates a part of this knowledge base.
Now, an F-Logic query may ask for all people who do research in KM by:

Qa: FORALL Y <- Y[researchIn –>> KM]. (6)

which may result in the set $F(Qa) = \{$(rst), (ysu), (meh), (gst), (nst)$\}$.

By using the associations:

$\overline{X} :=$ (Y), $\overline{k} :=$ (KM), $\overline{P} :=$ (P_1), $P_1(Y,KM) :=$ Y[researchIn –>>KM],

the query (6) can be represented in the formal manner (see Definition 2.7).

Obviously, all the answers are correct with regard to the given knowledge base and ontology, but the question is how these answers are related to each other. Let us consider the intuitive assumptions about the relevance of the different results in the following cases:

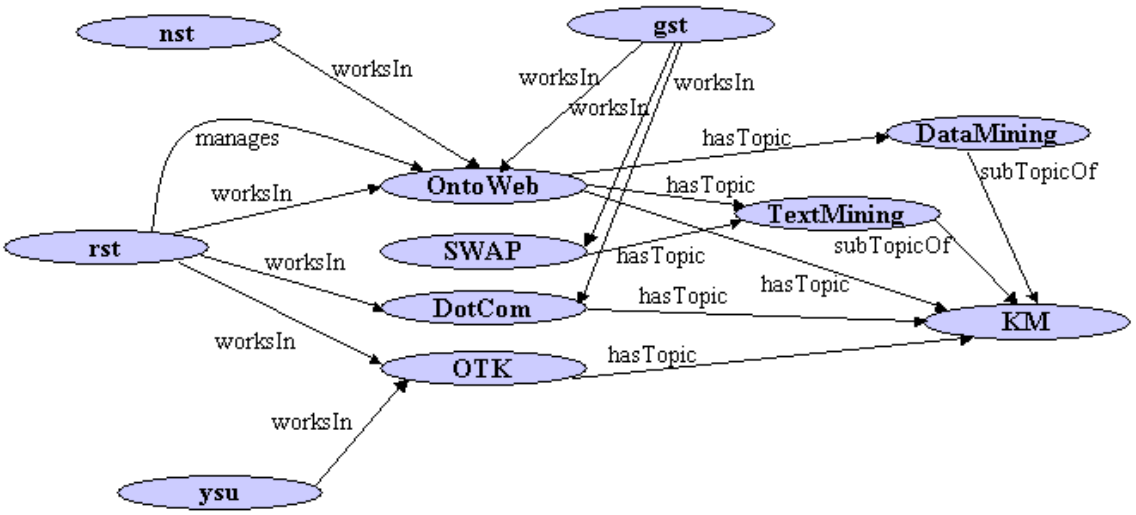


Figure 2.7: A part of the knowledge base from the institute example

1) ysu vs. gst
- ysu works in just one project and that project is about KM
- gst works in three projects but just one of them is about KM
  => ysu is more relevant for KM than gst.

2) rst vs. ysu
- ysu works in just one project and that project is about KM
- rst works in three projects, all related to KM
  => rst is more relevant for KM than ysu.

3) gst vs. nst
- gst is a Professor and gives a lecture about KM
- nst is a PhDStudent and gives the same lecture as gst (7)
  => gst is more relevant for KM than nst

4) nst vs. meh
- nst works in just one project and that project is about KM
  - meh works in just one project and that project is about TextMining, which is a special part of KM research
  => nst is more relevant for KM than meh

5) ysu vs. nst
- ysu works in just one project and that project is about KM
- nst works in just one project and that project is about KM
  => the relevance of ysu and nst is the same,

Alternatively, we can use more information about these relations:
- ysu works in just one project and that project is about KM and he works with just one other person
- nst works in just one project and that project is about KM and there are two other persons who support the research in that project.

It could be concluded that ysu is more relevant than nst, but the following arguments might be applied:
- ysu works for the whole project => ysu puts a lot of efforts in the project
- nst works with two other persons in the project => nst benefits from the collaboration with two other persons.

> Therefore, the relevance depends on additional factors, e.g. what the proportional active participation of all three members in that project is.

This last discussion shows a very important characteristic of the notion of relevance that has to be considered in the IR context, as we already mentioned in section 2.2.3:

> What is relevant is decided by the user from session to session and from time to time, and is then heavily dependent on judgments where highly subjective and scarcely reproducible factors are brought to bear.

Therefore, concerning this point, the addition of imprecision on top of a calculus for conditional reasoning can indeed work as a "correction factor" for bridging the gap between the rigidity of logical calculi and the flexible, human-centred notion of relevance, as in principle it allows to fine-tune the system estimation of relevance as a function of contextual factors and user preferences.

Due to its conceptual nature, ontologies provide an ideal abstraction level, on the top of conditional reasoning, for defining this flexible notion of relevance, which we will call *conceptual relevance*. This relevance will be explained in detail in the next section.

Note that although a query returns the set of concept instances as an answer, the relevance of these answers is defined on the level of the relation instances. The reason is that the concept instance is treated as an identifier of an object (e.g. `rst`), whereas the relation instance (e.g. `rst[researchIn->>KM]`) represents the property of that object whose relevance for the query can be determined. It means that the relevance of an answer $a = \overline{A_1} = (a_1, a_2, ...a_n)$ for query Q will be calculated on the substitution of this answer in query Q, i.e. by considering the $\overline{P}(\overline{A_1}, \overline{k})$: $P_1(\overline{A_1}, \overline{k}), P_2(\overline{A_1}, \overline{k}), ..., P_k(\overline{A_1}, \overline{k})$ - the set of *returned relation instances*, depicted as $\Lambda(a, Q)$, for an answer a. It is clear that $\forall a$, $\Lambda(a, Q) \subset \prod(\mathrm{O})$ (for the used terminology see section 2.3.2.2).

For the given example:

Λ(`rst`, Qa)={`rst[researchIn->>KM]`},
Λ(`ysu`, Qa)={`ysu[researchIn->>KM]`}, Λ(`gst`, Qa)={`gst[researchIn->>KM]`}
Λ(`nst`, Qa)={`nst[researchIn->>KM]`}, Λ(`meh`, Qa)={`meh[researchIn->>KM]`}

Since the returned relation instances for each answer can be proven in the ontology, the traditional definition of relevance via a similarity function between returned relation instances [5] is useless (similarities between two arbitrary answers are equal). We extend this notion in the following two definitions.

Definition 2.9: *Conceptual relevance of a relation instance*

$$sim\_\sigma : \prod(\mathrm{O}) \times \Omega(\mathrm{O}) \rightarrow \Re$$

It computes the conceptual relevance of a relation instance returned in the ontology-based retrieval process.

Definition 2.10: *Conceptual relevance of an answer*

The conceptual relevance of an answer (i.e. a resource) a, $sim\_\rho(a, Q)$, $a \in F(Q)$, is the geometrical mean of the relevancies of the returned relation instances for that answer, i.e.

$$sim\_\rho(a,Q) = \sqrt[|\Lambda(a,Q)|]{\prod_{x \in \Lambda(a,Q)} sim\_\sigma(x,Q)},$$

where $|S|$ denotes the cardinality of a given set S. Finally, the ranking of the answers for a query is achieved by ordering them according to their relevance.

2.3.3.3 Conceptual Relevance

There are different interpretations of probability that can be used for calculating relevance [163]. Traditionally, one can understand probability from the frequency point of view—the aleatory view. That is, probability is a statistical notion, concerning itself with the statistical laws of chance. On the other hand, probability can be interpreted as the degree of belief—the epistemological view. This view concerns the assignment of beliefs in propositions. Different interpretations of the theory of probability lead to different approaches for modelling relevance in information retrieval.

Therefore, in the traditional relevance models, relevancies are obtained simply by counting the number of resources containing a particular descriptor or index term. The argument for using the statistical notion of relevance is that probabilities should be viewed as a measure of chance at the implementation level. However, the neglect of other explanations of relevance at the conceptual level is perhaps the source of difficulties in the conventional relevance model.

Since ontologies represent a conceptual model of a domain, they seem to be an ideal source for defining this epistemological view on relevance. Moreover, the conceptual notion of relevance is one of fundamental characteristics (advantages) of the ontology-based information retrieval.

Considering conceptual level, there are two views on the relevance of a resource *r* for an information need expressed in query *q*, we define in following two definitions.

Definition 2.11: *Collection relevance*

It represents relevance of the resource regarding the given information repository (so called *collection relevance*, in the notion *ColRel*):

$$ColRel : \prod(O) \times KB(O) \rightarrow \Re$$

E.g. for the domain presented in the motivating example, for a query about researchers in `KM`, useful information for ranking can be that a researcher works in three projects about `KM` and gives two lectures regarding `KM`.

Due to a very rich relations structure of an ontology-based repository, the relevance of a resource cannot be treated in isolation, i.e. each resource (information) has to be put in the context of other semantically related resources (information). For example, the relevance of fact `rst[worksIn->>OntoWeb]` depends on information in how many projects he works and how many researchers are involved in these projects.

Figure 2.8 depicts the part of the knowledge base presented in Figure 2.7 used for calculating collection relevance for the statement `rst[worksIn->>OntoWeb]`.

Definition 2.12: *Explanation relevance*

It represents relevance of the retrieval process *M* (see previous section) in which a resource (i.e. result of a query) is retrieved (so called *explanation relevance*, in the notion *ExpRel*):

$$ExpRel : M \times \prod(O) \rightarrow \Re$$

E.g. for the domain presented in the motivating example, the query for a researcher who researches in `KM` will return the researcher who researches in `DataMining` as well, since `DataMining` is a subtopic of `KM`.

The conceptual relevance should reflect the cognitive process, mapped in a logical inference process, which explains why a resource is retrieved. For example:
`rst` researches in `KM`, since he works in the project `OntoWeb` and that project is about `KM`, i.e.

```
rst[researchIn ->>KM]
<- rst[worksIn ->>OntWeb] and OntoWeb:Project[hasTopic ->>KM]
```

(Axiom in line 8 from (4))

It is clear that as more "reasons" exist that explain why a resource is relevant and the "stronger" these explanations are, the higher this relevance should be.
For example, there are two evidences that `rst` works in `OntoWeb`: he is a member of the project and he manages the project:

```
rst[worksIn->>OntoWeb],
rst[worksIn->>OntoWeb]<- rst[manages->>OntWeb] and OntWeb:Project
```

(Axiom in line 11 from (4)).

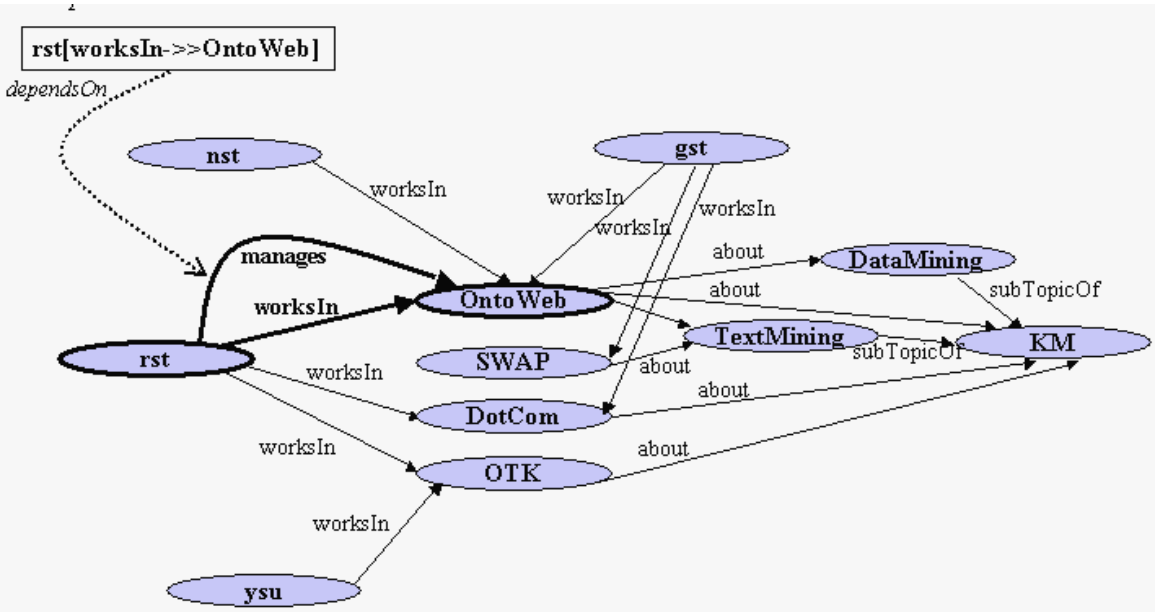


Figure 2.8: A part of the knowledge base from Figure 2.7 relevant for calculating collection relevance for the statement `rst[worksIn->>OntoWeb]`

Figure 2.9 depicts the part of the knowledge base presented in Figure 2.7 used for calculating explanation relevance for the statement `rst[worksIn->>OntoWeb]`.

Finally, for an $x \in \Lambda(a, q)$ in repository $I(O)$, conceptual relevance is calculated as:

$$sim\_\sigma(x, q) = f(ExpRel(M(I(O), q), x), ColRel(x, I(O))).$$

Function f will be described/derived in the next sections.

Discussion:
Conceptual relevance represents a kind of belief not in the information itself, in terms of "am I sure that the information is relevant, e.g. am I sure that `rst` works in `OntoWeb`" which is the key point in the traditional IR systems (e.g. "in document X the word Y appears n-times => the document X is about Y with relevance f(n)". Rather, the conceptual relevance is focused on calculating evidence that information is the most relevant for the user's information need, in terms of "how sure I am that the information is the most relevant one, e.g. how sure am I that `rst` is the most relevant regarding working in `OntoWeb`".

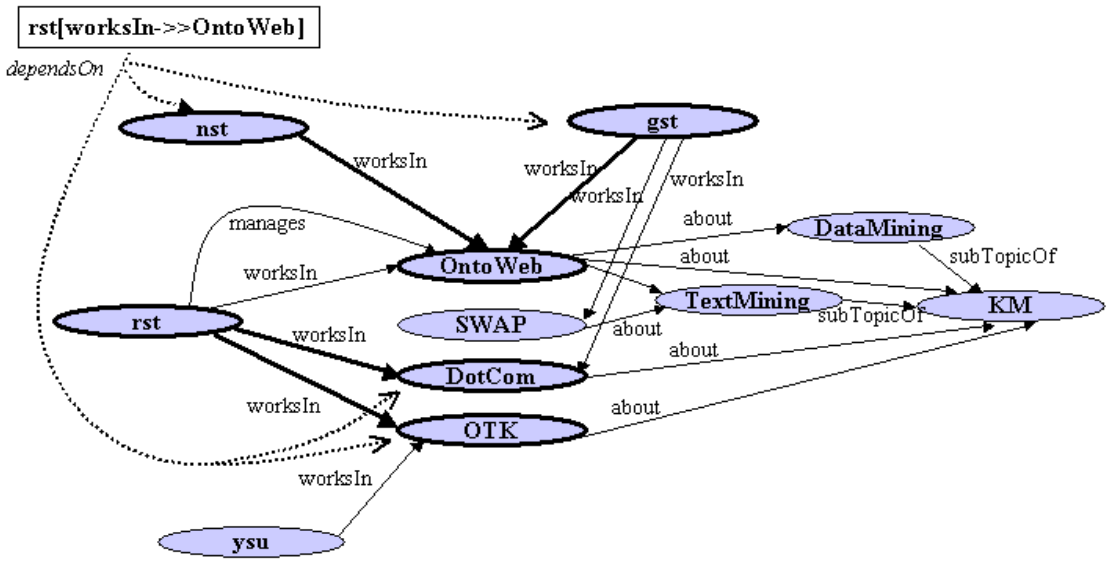


Figure 2.9: A part of the knowledge base from Figure 2.7 relevant for calculating collection relevance for the statement `rst[worksIn->>OntoWeb]`

Although one can remark that both approaches are about calculating evidences, the nature (purpose) of the calculation is completely different:

- in traditional IR systems the evidence is calculated in order to determine WHETHER an information is relevant (e.g. above a threshold),
- in ontology-based IR systems the evidence is calculated in order to determine HOW RELEVANT the information is since it is already known that it IS relevant.

This discussion helps in interpreting relevance, like that the relevance of the information `rst[worksIn ->>OntWeb]` (i.e. that `rst` is an answer to the query about who works in `OntoWeb`) is 0.7:

- regarding traditional, syntactical relevance it means that `rst` works in `OntoWeb` with about 70% of evidence, i.e. it is possible (30%) that he does not work in `OntoWeb`. This is all one can conclude from a syntactical relevance

- regarding conceptual relevance it means that e.g. `rst` contributes to the working in `OntoWeb` with 70%. This implies a lot of additional information, like:

  1) `rst` shares his working time between `OntoWeb` and other projects, and,

  2) there are other researchers who are working in `OntoWeb`.

Note that in a more complex example, this additional information is more complex (and more confident) as well.

Therefore, conceptual relevance is not just a number derived from a resource itself. It is evidence that comes from the context (relationships) in which a resource appears in a domain.

We argue that from the cognitive point of view, the (non-)confidence in a statement regarding the meaning of an information resource can be set only in the context of the (non-)confidence in the related statements assigned to that resource. For example, what does it mean that a text document is about research topic “knowledge management” with fifty percent relevance?

1) fifty percent of the text consists of words knowledge and management, or,

2) the text contains some information about another research topic.

In the case that a document is about three research topics, by default, each of them should be covered with 33,3% confidence. Therefore, we claim that no relevance of a statement that expresses the meaning of a resource should be set without considering other possible meanings of that resource. Moreover, from the cognitive point of view, the sum of the relevancies of related meaning (statements) should be 1. Of course, the notion of "related meaning" should be formally defined. Such a formal framework is provided by an ontology. We can assume that all statements (relation instances) that belong to the same relation for the same domain are related. For example, the relation instances:

`rst[researchIn ->>KM]` and `rst[researchIn ->>TextMining]`

are two related statements since both of them expresses the research activities of `rst`.

However, the notion of a single representation of meaning (i.e. relatedness) may not be practical since the meaning of a body of text is so heavily dependent upon the context in which it is to be interpreted. For example, in one context relations `researchIn` and `teaches` can be interpreted as related ones. Therefore, we find it more plausible not to calculate the relevance of a statement in advance, but rather at query time, in a concrete context (or for a concrete user).

We can use here an analogy with the Google Page Rank [92] ranking approach, where the relevance of a web page is determined not only by considering the content of the page alone (syntactical relevance) but rather some information about the relevance of a web page in a usage context as well. Indeed the number of links pointing to a page defines the relevance of that page. However, in our approach we go a (semantic) step further – we consider the relevance of the content presented in that page and not the page itself. Note the difference between (i) the relevance that an information is presented in an information resource and (ii) the relevance of the content presented in that information resource.

Regarding (i) the relevance that a page is about "knowledge and business" can be measured by considering the frequency of the appearance of these terms.

Regarding (ii) the relevance can be measured by the frequency of appearance of other terms that are related to the terms "knowledge and business".

The modelling of two above mentioned relevancies using IR techniques is the topic of the next sections.

2.3.3.3.1 Collection Relevance

In the probabilistic inference model [163], it is assumed that there exists an ideal concept space U, called the universe of discourse or the domain of reference. Elements in this space are considered to be elementary concepts. The concept space can be interpreted as the knowledge space, in which information resources and user queries are all represented as propositions. A proposition is a subset of U. This correspondence between propositions and subsets of elementary concepts is useful because it translates the logical notions of conjunction, disjunction, negation and implication into more familiar set-theoretic notions of intersection, union, complementation and inclusion.

Probability function P is defined on concept space U. Probability $P(r)$ is interpreted as the degree to which U is covered by the knowledge contained in resource r. Proposition $r \cap q$ represents the portion of knowledge in r requested by q. Similarly, probability $P(r \cap q)$ represents the degree to which U is covered by the knowledge common to both query q and resource r. Figure 2.10 illustrates the process of calculating these values.

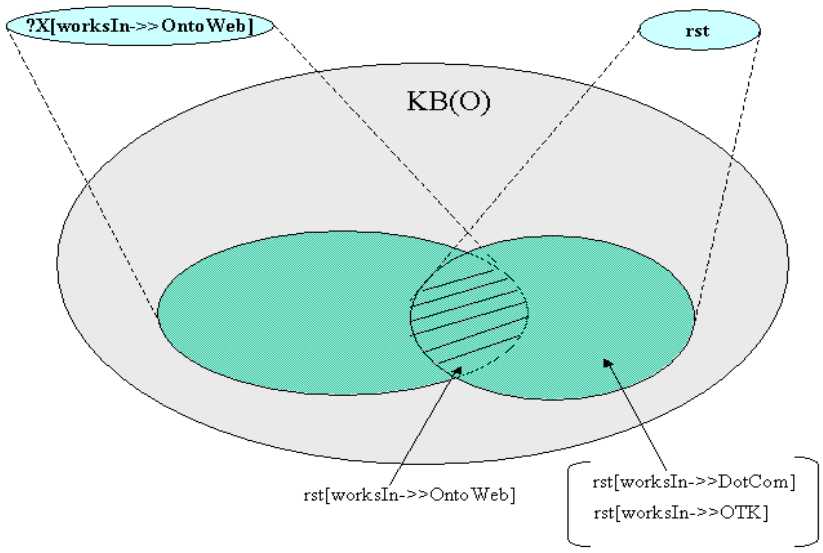


Figure 2.10: Basic principle of calculating collection relevance. An example for query `forall x <- x[worksIn->>OntoWeb]`, that is shorthanded as ?X[worksIn->>OntoWeb].

Based on these probabilities, the following measure is constructed to evaluate the relevance of resources:

$$Rel(r \rightarrow q) =_{\text{def}} P(q \mid r) = P(r \mid q) * P(q) / P(r) = P(q \cap r) / P(r) \quad \text{(Bayesian rule)}.$$

In [163] it is shown how this model can be used for representing standard techniques for calculating relevance (e.g. TF/IDF). Therefore, this model can be treated as a generic model for calculating statistical relevance.

The main problem in the model is the calculation of $r \cap q$, i.e. "the portion of knowledge in r requested by q". As we already explained, from the syntactical point of view, it is difficult to determine which portion of knowledge regarding r is relevant for query q, since the description of the knowledge is related to the syntactical structure of a resource (e.g. how often the terms from the query appear in a document). It means that the relevance of a resource is treated in isolation[11].

However, by using an ontology as the conceptual model, the semantics of a resource is described through the relations between this resource and other resources. Therefore, if we define knowledge as a relation between various resources, then the portion of knowledge in resource r requested by a query encompasses all semantic relations that are covered by the query. For example, for query `forall x <- x[worksIn->>OntoWeb]`, all relations of type `[worksIn->>OntoWeb]` are related to the query. In the standard case this portion is $1/t$, where t represents all knowledge contained in the repository (i.e. KB(O)) and it is calculated as the total number of relations that the resources participate in[12].

On the other hand, $P(r)$ represents the portion of all available knowledge in the domain that is contained in resource r. From the semantic point of view this is the portion of available knowledge (relations) that is relevant for the particular query, as we already explained in the Discussion in the previous section. For example, if resource r participates in k relations of type `worksIn` (i.e. he works in k projects), then $P(r \cap q) = k/t$, where t represents all knowledge contained in the repository.

[11] In the standard techniques for measuring relevance the frequency of the term's appearance in the entire repository is taken into account as well. However, this value is treated only as a threshold (i.e. syntactically): each very frequently appearing term is treated as a non-relevant one in that repository.

[12] In the case of the existence of a hierarchy in the given relation (e.g. worksIn) the multiplier can be greater than 1.

Therefore,

$$Rel(r \rightarrow q) = P(q \cap r) / P(r) = (1/t) /(k/t) = 1/k.$$

According to the formal definition of the ontology, k represents the number of instances that are in a relation R (given in a query) with resource r:

$$Rel(r \rightarrow forall\ x\ R(x, inst)) = 1 / |\{m \mid R(r, m) \in KB(O)\}|$$

From the implementation point of view, set $\{m \mid R(r, m) \in KB(O)\}$ is defined as the number of interpretations of the given relation instance with respect to that resource (i.e. when all other terms in the relation instance, except the considered term, are substituted with a variable). We introduce the function *Interpretation*: I(O) x $\prod$(O) $\rightarrow$ N, in the following definition:

Definition 2.13: *Interpretation*

$$Interpretation(I_j, R(I_1, I_2, ..., I_j, ..., I_n)) =$$

$$\left|\{x, y, ..., w \mid \exists x, y, ...w \quad R(x, ..., I_j, ..., w) \in KB(O)\}\right| \qquad (8)$$

where I_i represents a concept instance and $R(I_i, I_i \ ..., I_i)$ is a relation instance

For example, for the situation presented in (4) and (5),

$$Interpretation(rst, worksIn(rst, OntoWeb)) = \left|\{x \mid \exists x \quad worksIn(rst, x)\}\right| = 3$$

(i.e. person `rst` works in three projects `OntoWeb`, `DotCom` and `OTK`).

Similarly,

$$Interpretation(gst, worksIn(gst, OntoWeb)) = \left|\{x \mid \exists x \quad worksIn(gst, x)\}\right| = 3.$$

In other words, we calculate the number of relations of the selected term (i.e. concept instance) which correspond to the relation type of the given relation instance. This is a very important characterisation of the link between two nodes in a semantic network [147], the so-called type-specific fanout (TSF) factor, which defines the strength of connotation between two nodes. It is used for the disambiguation of the meaning of a node (term) in large semantic networks, like WordNet [78].

The relevance $Rel(r \rightarrow q)$, is the basic relevance required to calculate collection relevance. In other words, the relevance of a relation instance had to be calculated in order to define a collection relevance. Therefore, for x $\in \Lambda(r, q)$:

$$ColRel(x, I(O)) = \Pi_{i=1,k}\ Rel(r_i \rightarrow q_i),$$

where:
q_i is generated from x by substituting one of the instances with a variable. For example, for x = `rst[worksIn->>OntoWeb]`:
$q_i \in$ { `forall x <- x[worksIn->>OntoWeb], forall x <- rst[worksIn->>x]` }.

Indeed, if we try to calculate the relevance of fact `rst[worksIn->>OntoWeb]`, in order to take into account the entire knowledge base, the relevance has to be combined from two queries that have this fact as an answer:

`forall x <- x[worksIn->>OntoWeb]` and `forall x <- rst[worksIn->>x]`.

Figure 2.11 illustrates this discussion.

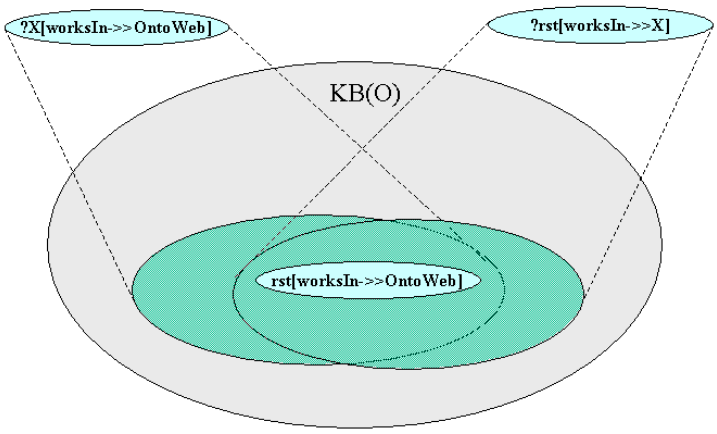


Figure 2.11: Different views on a relation instance required for the calculation of the collection relevance

Since all $q_i s$ queries contain the same relation, the calculation seems to be very simple. For x $\in \Lambda(r, q)$ and $q = forall\ x\ R(x, k)$:

$$ColRel(x, I(O)) = \frac{1}{Interpretation(I_1, R(I_1, I_2, ..., I_n))} \cdot ... \cdot \frac{1}{Interpretation(I_n, R(I_1, I_2, ..., I_n))} \quad (9)$$

For example, regarding the situation presented in (4) and (5),

$$ColRel(worksIn(gst, OntoWeb), I(O)) = \frac{1}{Interpretation(gst, worksIn(gst, OntoWeb))} \cdot \frac{1}{Interpretation(OntoWeb, worksIn(gst, OntoWeb))} = \frac{1}{3} \cdot \frac{1}{3} = \frac{1}{9}$$

The value is maximal (i.e. = 1) when none of the terms from the relational instance can be found in any other relation instance of the same type[13]. For example, relation instance `teaches(meh, InfoB)` (in F-Logic `meh[teaches->>InfoB]`) regarding (5) has the maximal value, because there is only one relation instance for relation `teaches` which contains either `meh` or `InfoB`.

As mentioned above, this kind of measuring relevance is used in defining the relevance of two nodes (terms) in a semantic network. Similar approaches for the calculation of such a similarity between terms in the WordNet are given in [16], [100], [147].

Discussion:

Our approach for calculating collection relevance can be adapted to the various (context-dependent) interpretations of the notion of relevance. Let us as assume that in a context, "the portion of knowledge in *r* requested by *q*" means/contains all relations of resource *r* which are related to the query. For example, for query `forall x <- x[worksIn->>OntoWeb]` and answer `rst`, all relation instances containing relation `worksIn` and instance `rst` (i.e. in the form `rst[worksIn->> y]`) are related to the query.

[13] A relation type is defined by the relation which is instantiated in the relation instance.

Therefore, $P(q \cap r) = l/t$, where:
l represents the number of relations, of the type used in query q, which resource r participates in i.e. for query *forall x* $R(x, k)$ it is $\{m \mid R(r, m) \in KB(O)\}|$ and
t represents all knowledge contained in the repository.

Similarly, $P(r)$ can be calculated as p/t, where p represents the total number of relations, which resource r participates in, $\{(m, R) \mid R(r, m) \in KB(O)\}|$.

In that case, $Rel(r \rightarrow q) = l/p$.

Indeed, this view of relevance seems to be very reliable in some situations. For example, case 5) from (7) can be explained through this relevance:

> `nst` works with two other persons in the project => `nst` benefits from the collaboration with two other persons.

Note that in the original form of calculating $Rel(r \rightarrow q)$, the relevance decreases in such a situation.
Moreover, an ontology can model this type of information, i.e. which relations and even in which situations (contexts) should be calculated using an existing type of relevance.

2.3.3.3.2 Explanation Relevance

Since the explanation relevance has a strong inferential nature, there is no analogy to this type of relevance in traditional IR systems. However, from the implementation point of view, the idea of considering multiple document representations schema that generates different paths in which a resource is determined as a relevant one seems to be useful for calculating explanation relevance. This idea can be found in the inference network model for IR we have already presented in section 2.2.4. In this section we are showing how explanation relevance can be calculated using this IR method for modelling uncertain inference.

We first define the structure of such an inference network regarding an ontology-based retrieval model. Second, we estimate the probability of each link in that structure.

Structure
The document nodes represent the instances from the knowledge base of the domain ontology (i.e. information resources from ontology-based information repository), $d_k \in I(O)$ (cf. Figure 2.12).
As we have already mentioned in section 2.2.4, it is common to assume one-one correspondence between documents and texts. It means that this dependency is complete, a text node is observed (t_l = true) exactly when its parent document is observed (d_l = true). Moreover, an ontology operates on the conceptual level where these differences are abstracted. Therefore, we consider document and text representation nodes as identical. They are instances from the ontology: $i_l \in I(O)$ (cf. Figure 2.12).

The concept representation nodes represent semantic interpretations of the instances. These interpretations are defined through ontology relations, i.e. relation instances, $r_k \in KB(O)$.

As explained earlier, the representation concept nodes correspond to the expression of an information need. Since in the ontology-based IR these primitive concepts are relation instances, query concepts (c_i) have the same meaning as the representation concepts so that each query concept has exactly one parent.

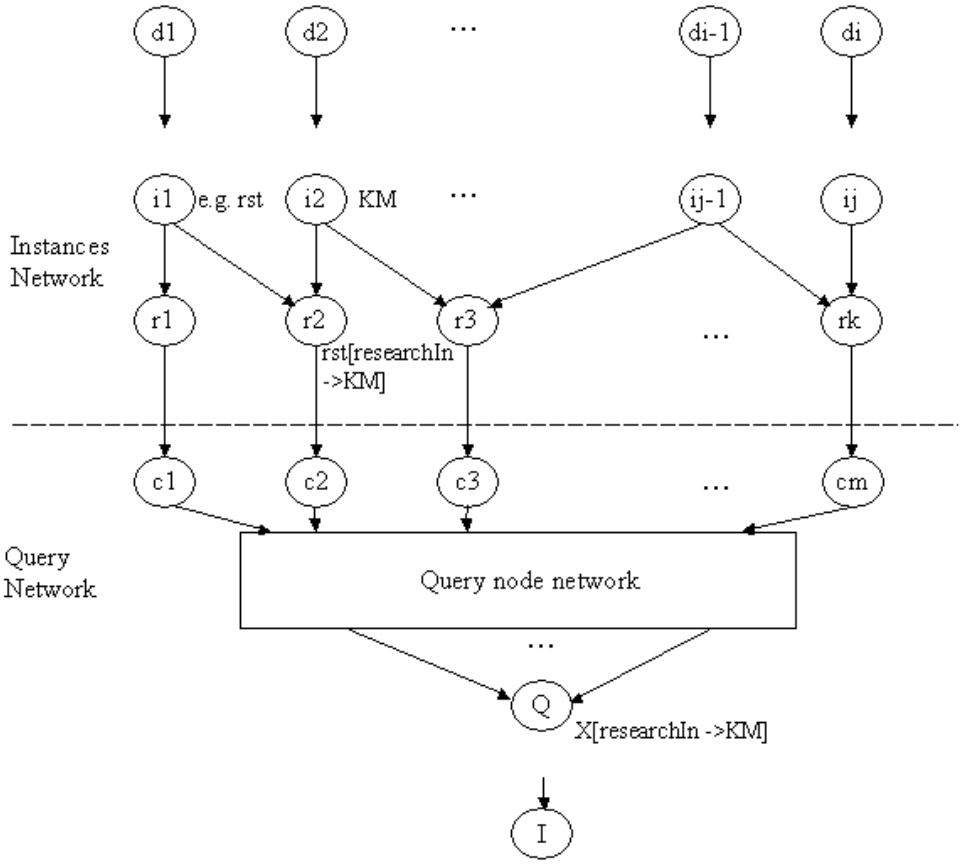


Figure 2.12: Basic inference network for ontology-based IR

The information need represents a request of the user for an information resource. According to the discussion presented in the previous section, the relevance of that request should be established on the level of the meaning of a query. In this way, the multiple query representations, enabled by a Bayesian network, will come in play. Indeed, since the retrieval mechanism M is realized as an inference process, it uses several ontology axioms in order to evaluate a query. Since each axiom depicts a different representation of a query a cascade of depending axioms can build the network of intermediate query nodes (i.e. $q_i \in \Omega(O)$), see Query node network in Figure 2.12). For example, regarding *Institute example*, the query for a researcher who works in topic X can be represented using two queries: (1) a query for a researcher who researches in a project and (2) a query for projects about X. It means that the user's information need can be better represented by using such a decomposition. However, a variety of relationships may exist among the axioms (rules) in an ontology. Indeed, the conclusion of a rule may act as conditions of other rules and different rules may share common conditions. One can imagine a process in which all possible decompositions of a set of axioms are done. In this way we obtain a list of elementary query representations whose validity should be inspected regarding the concrete knowledge base in a retrieval process.

A way to perform this decomposition is by constructing a rule-dependency graph, as presented in the following definition:

Definition 2.14: *Rule dependency graph (RDG)*

The RDG of an arbitrary set of m rules is a directed graph $G = (R, P, E, L)$:

- R is a set of R-nodes, each of which represents rule r_i, $m \geq i \geq 1$, denoted by $R(r_i)$,
- P is a set of P-nodes, serving as sources of conditions and targets of conclusions,

- E is a set of directed links connecting R-nodes and P-nodes. Let $E_P=\{e|\ e \in (P \times R)\}$ and $E_R=\{e|\ e \in (R \times P)\}$. E is a subset of $E_P \cup E_R$. Each condition p_{ij} in rule r_i is a link from P-node to R-node $R(r_i)$ and each conclusion q_{ik} in rule r_i is a link from R-node $R(r_i)$ to P-node, and,

- L is a set of labels that are literals representing conditions or conclusions.

Figure 2.13 illustrates this structure for a small rule base.

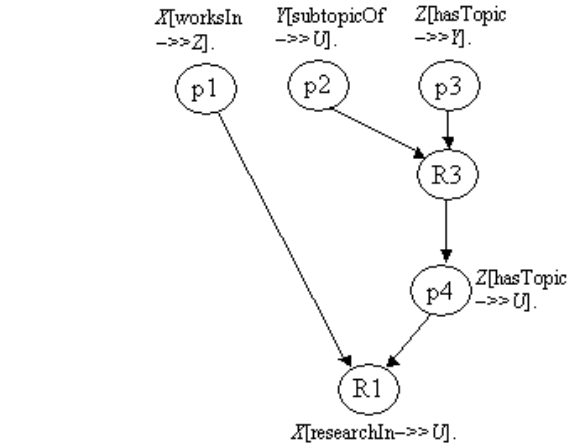


Figure 2.13: A simple dependency between rules

Since we are not dealing with uncertainty in the axioms, there is no difference between the "strength" of links (=1).

By elementary transformation[14] of RDG's labels into queries the corresponding query node network, needed for the Inference network, is obtained. The process is illustrated in Figure 2.14.

However, in order to make this process more effective some optimisation of the query network is needed. In other word, for a particular query, we should filter irrelevant information from the query network. We present such an approach in following text.

By considering the problem in the opposite way, that is, from the answers, the important question for calculating relevance of an answer for the given query is which parts of the RDG are relevant for that answer. In other words which parts of the graphs can be used as proofs for an answer. Therefore, by considering the structure of the inference process (which is, in principle, based on the RDG of that rule base) we can collect all rules evaluated, i.e. we can build an optimal inference network for a particular answer. Consequently, we can calculate its relevance.

We omit here the detailed description of the different types of the inference processes (evaluations) [153], but present instead the common inference structure which can be used for the proof of the returned results. This structure is described in the next three definitions.

[14] This transformation depends on the language used for query representation. In the case of F-Logic a query is obtained by adding the statement "FORALL *var* <-" at the beginning of the literal that is output of a node, where *var* is the name of the variable whose values have to be returned.

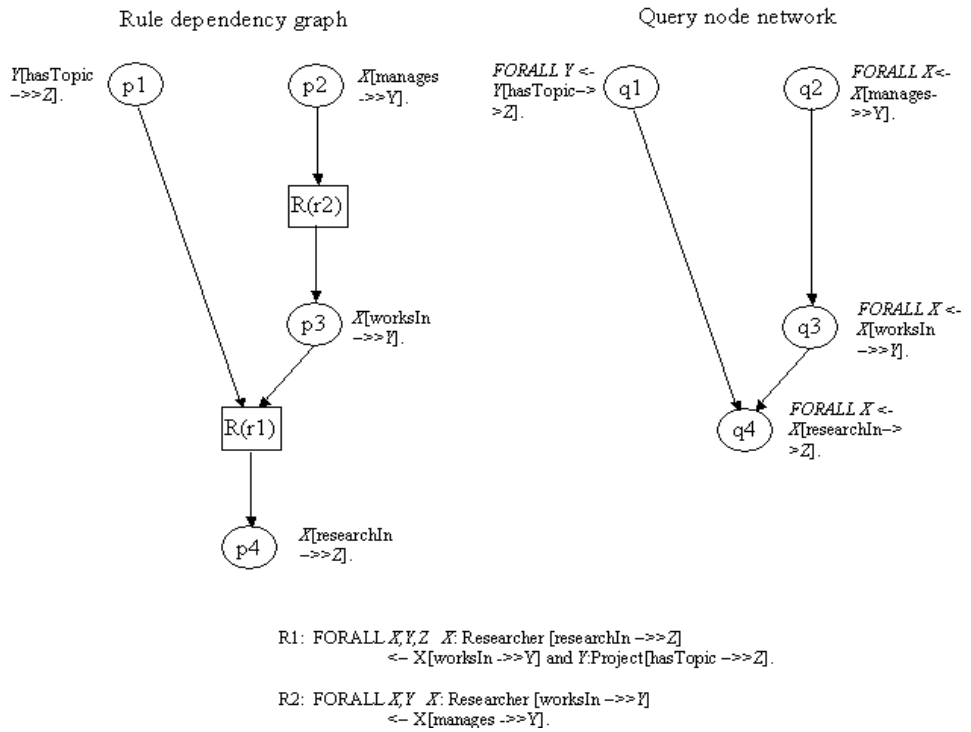


Figure 2.14: An example of transformation *rule dependency graph -> query node network*

Definition 2.15: *AND-OR tree*
An AND-OR tree is a tree structure, defined as the set T = $\{root, N, v\_and, v\_or\}$, where:

- *root* is the root of the tree,

- *N* is the set of nodes in the tree, and,

- *v_and* and *v_or* are (irreflexive, anti-symmetric) relations between nodes. They define links in the tree and are called parent-child relations in the text:

  - *v_and*: $N \rightarrow N^*$, such a link is called an and_link and the corresponding node is called an and_connector (see Figure 2.15),

  - *v_or*: $N \rightarrow N^*$, such a link is called an or_link and the corresponding node is called an or_connector.

Definition 2.16: *Derivation tree of a query*

Given an ontology O with axioms, the derivation tree of query *Q* is an AND-OR tree whose root plays the role of an or_connector between the derivation trees of each result (resulting relation instance) for the query *Q*, i.e. for the elements from *F*(*Q*) (results of the query *Q*).

$DTree(Q) = \{root, N, v\_and, v\_or\}$,

$$N = \bigcup_{r \in \Lambda(a,Q) \wedge a \in F(Q)} DTree(r)$$

v_or: root→N
v_and = {}

Definition 2.17: *Derivation tree of a relation instance* (*creation*)

The derivation tree of a relation instance $r(I_1, I_2, ..., I_n)$ is defined as follows:

- every relation instantiation *l* from KB is a derivation tree for itself; a single node with label *l*.

- Let A be the set of axioms from the given ontology O, whose heads contain the atomic formula which can be unified with relation r.

- Let A = $p \Leftarrow q_1,...,q_n$, be an axiom from A, let d_i, $1 \le i \le n$, be relation instances with derivation trees T_i, let θ be mgu[15] of $(q_1,...,q_n)$ and $(d_1,...,d_n)$. Then the following is a derivation tree for p[θ] (relation instance) : the root is a node labelled with p[θ] and each T_i, $1 \le i \le n$, is a child of the root. The root plays an and_connector role.

- The derivation tree for r is the tree with the root which plays the role of an or_connector between the derivation trees for all axioms from A. It is labelled with p[θ]*

Figure 2.15 depicts a derivation tree, created according to Definition 2.15, Definition 2.16 and Definition 2.17. It is clear that a derivation tree represents the manner in which a result was inferred. The root is the disjunction of all results. Each node in a tree is a relation instance. Each *or_connector* node connects various ways in which a relation instance is derived, whereas an *and_connector* node defines a way in which the relation instance is derived. For a relation instance there are several *and_connector* nodes (the number is equal to the number of rules, whose head parts unify with that relation instance) and an *or_connector*, which connects those *and _connector* nodes.
Therefore, an "inverted" derivation tree can be treated as complete decompositon of a query for a particular answer, i.e. as the relevant part of the query node network for the given information need.

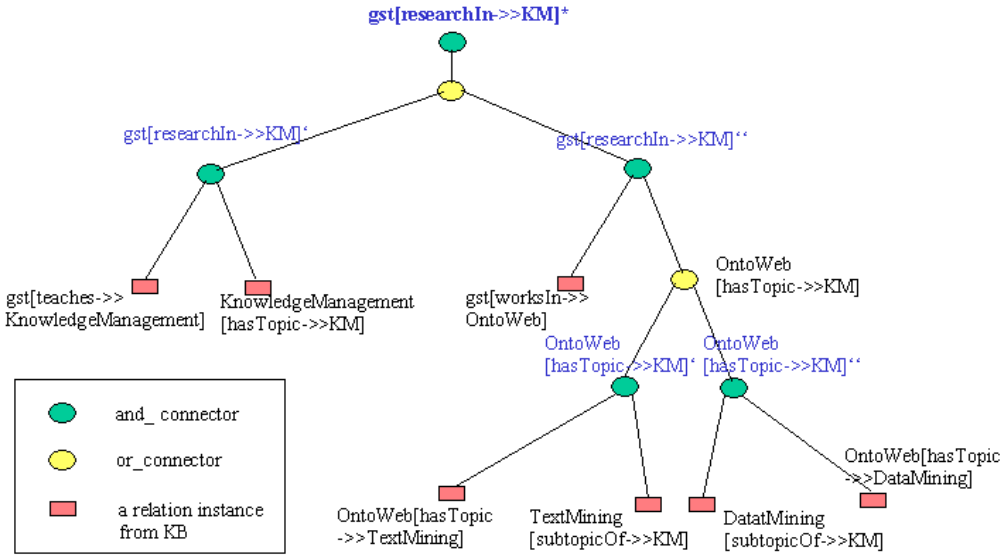


Figure 2.15: Derivation tree for the relation instance gst[researchIn->>KM] regarding the motivating example (generated according to to Definition 2.15, Definition 2.16, and Definition 2.17)

Estimating the Conditional Probabilities
In order to propagate the relevance through an inference network, the conditional probabilities between linked nodes have to be known. In section 2.2.4.1 we have already given an estimation of these probabilities in the case of Boolean operators and the weighted sum.

Regarding the network presented in Figure 2.12:

[15] A substitution is a mapping from the set of variables of the language under consideration to the set of terms. Two terms t1 and t2 are said to be unifiable if there is a substitution σ so that t1[σ] = t2[σ]; σ is said to be a unifier of t1 and t2. Note that if two terms have a unifier, they have a most general unifier (mgu) that is unique up to the renaming of variables.

Instance network

1) links ($i_i - r_k$) have the weight equal to *Interpretation*(I_i, $R(I_1, I_2, \ldots, I_i, \ldots, I_n)$),

where:

I_i is the instance that corresponds to the node i_i (e.g. `rst`)
$R(I_1, I_2, \ldots, I_i, \ldots, I_n)$ is the relation instance that corresponds to the node r_k (e.g. `rst[researchIn->>KM]`)

2) nodes r_k sum the corresponding links from i_i, i.e.

$$\text{Value}(r_k) = \Sigma_{i=1,n}\ \textit{Interpretation}(I_i, R(I_1, I_2, \ldots, I_i, \ldots, I_n)),$$

where I_i corresponds to node i_i that is connected to r_k
Note that this value reflects the collection relevance we have discussed in the previous section. Indeed, the instance network in the ontology-based inference network (see Figure 2.12) reflects the structure of the information repository.

Query Network

1) *and_connectors* play the role of an AND Boolean operator (because each child is a part of a rule that infers the parent), and,

2) *or_connectors* play the role of a weighted summarisation element (because children make an impact on the parent independently).

Such propagation corresponds to the propagation of the signals in a semantic network or so-called node activation sequences [77].

The following formulas describe the processing done by a node in the query network:

(i) for an *and_connector* node :

$$Rel\_and(node_i, QN(q), I(O)) = Col\,Rel(node_i, I(O)) \cdot \sum_{v\_or(node_i, node_j)} Rel\_or(node_j, QN(q), I(O)), \quad (10)$$

where:
$QN(q)$ represents the Query Network for query q, i.e. it reflects inference process $M(I(O), q)$ expressed in an AND-OR tree and
for root node *root*, we set $Col\,Rel(root, I(O)) = 1$. (11)

(ii) for an *or_connector* node:

$$Rel\_or(node_j, QN(q), I(O)) = \prod_{v\_and(node_j, node_l)} Rel\_and(node_l, QN(q), I(O)) \quad (12)$$

(iii) for *leaf nodes*: $Rel\_and(node_i, QN(q), I(O)) = Col\,Rel(node_i, I(O))$

Finally, the explanation relevance can be defined as the value of the root of the AND-OR tree, i.e.

$$ExpRel(M(I(O), q), x) = Rel\_and(x, QN(q), I(O)).$$

Since this calculation includes the collection relevance (see (10)), the final formula for calculating conceptual relevance of a relation instance looks like:

$$sim\_\sigma(x, q) = ExpRel(M(I(O), q), x), \quad (13)$$

for $x \in \Lambda(r, q)$ and $r = F(q)$.

Example:

In the following we give an example of calculating the relevance for the result {gst} for the query from the *Institute example* (`FORALL Y <- Y[researchIn ->> KM]`). Figure 2.15 depicts the derivation tree for this result.

The relevance can be calculated as follows[16]:

The result of the query: a = {gst}

The returned relation instances: Λ(gst, Qa) = {gst[researchIn->>KM]}

The relevance of the answer (formula 13):

$$sim\_\rho(gst) = sim\_\sigma(gst[researchIn->>KM]) = Rel\_and(gst[researchIn->>KM])$$

The relevance of the root node (formula 12):

$$Rel\_and(gst[researchIn->>KM]*) = ColRel(gst[researchIn->>KM]) \cdot ((Rel\_or(gst[researchIn->>KM]') + (Rel\_or(gst[researchIn->>KM]'')))$$

From (11) follows:

$$Rel(gst[researchIn->>KM]) = 1$$

Using (12) we get:

$$Rel\_or(gst[researchIn->>KM]') = Rel\_and(gst[teaches->>KnowledgeManagement]) \cdot Rel\_and(KnowledgeManagement[hasTopic->>KM])$$
$$= ColRel(gst[teaches->>KnowledgeManagement]) \cdot ColRel(KnowledgeManagement[hasTopic->>KM])$$
$$= (\frac{1}{3} \cdot \frac{1}{3}) \cdot (\frac{1}{1} \cdot \frac{1}{1}) = \frac{1}{9}.$$

In the similar relevance for relation instance is obtained `gst[researchIn->>KM]''`:

$$Rel\_or(gst[researchIn->>KM]'') = \frac{1}{18}.$$

Finally, the relevance of the result `gst`:

$$sim\_\rho(gst) = sim\_\sigma(gst[researchIn->>KM]) = 1 \cdot (\frac{1}{9} + \frac{1}{18}).$$

Discussion:

i) The ranking measure described with (13) deals with both factors for determining the relevance we have mentioned in the previous section (i.e. the knowledge base and the inference process)

ii) In formula (8) we treat the impact of the number of relations of the same type *r* as follows: when an instance is in several relations of the same type, then their relevance for that relation type is split into all of them. This resolves the case 2) from the section 2.3.3.2. However, when each of these relations is relevant for the given query, according to the summing up in formula (10), the instance has an impact "without losses" (= 1) on this relation. This is important for resolving the case 1) from section 2.3.3.2.

iii) Since for each transitive relation there is a rule which enables inferring over the subconcepts, the derivation trees of two subconcepts, which are placed in different depth in

[16] In order to simplify the formulas we exclude all parameters that are related to the whole ontology, like $I(O)$ and $QN(q)$.

the hierarchy of a term, are different. Consequently, their relevancies are different, i.e. the subconcepts placed deeper in the hierarchy are less relevant for the queries regarding the root concept. In that way cases 3) and 4) from section 2.3.3.2 are resolved.

iv) The case 5) from section 2.3.3.2 was treated in the discussion in section 2.3.3.3.1.

v) The formula (8) for calculating the specificity of a relation instance is general enough to model other interpretations of the relevance (see the discussion in the case 5 in section 2.3.3.2) as well as to incorporate other factors which can determine the relevance, e.g. usage information. We see the calculation of the relevance as a task/problem-sensitive decision. For example, whether the number of the instances which are in the same relation should increase or decrease the relevance is a problem-sensitive decision and we assume that the user wants to have opportunity to set such an indicator for each relation, otherwise he/she will use a default option.

vi) The approach can be easily extended with the information about the usage of a relation instance (see section 2.4.3).

2.4 Implementation & Evaluation

2.4.1 Implementation

In this section we give details about the implementation of the presented ontology-based information retrieval approach, especially the method for calculating relevance.

We implemented the presented ranking approach in the Semantic Portal of AIFB Institute. It is an ontology-based web application, which serves as the test-bed for our research related to ontologies and Semantic Web. The Semantic Portal (SEAL) [126] is an ontology-based application, which provides a "single-click" access to almost all information related to the organisation, people, research and projects of our Institute. It is widely used by our research and administrative staff as well as by our students. One of the most usable features is the possibility to search for people, research areas and projects on the semantic basis, i.e. using the corresponding Institute Ontology. The hierarchy of research areas is especially comprehensive – more than 130 concepts. The *Institute Example*, given in section 2.3.3.2 is a part of it. The portal provides a very user-friendly interface, which enables formation of arbitrary queries using entities from the underlying ontology. The search is performed as an inference through metadata, which are crawled from the ontology-based statements assigned to web pages.

As we have already mentioned, an ontology-based retrieval system requires:

- a mechanism for conditional reasoning (i.e. an inference mechanism), and,
- a method for estimating uncertainty in that reasoning process.

As the inference engine we use the Ontobroker system [37], a deductive, object-oriented database system operating either in the main memory or on a relational database (via JDBC). Ontobroker uses the bottom-up fix-point evaluation procedure. It allows general recursion and negation is allowed in the clause body. The native logical language of the Ontobroker is F-Logic. Ontobroker already supports ontology-based conceptualization of the problem domain it is dealing with, i.e. an inference process is enriched with the information from the underlying ontology. However, Ontobroker does not treat the uncertainty in reasoning at all. More details about Ontobroker can be found in [37].

On the top of this inference mechanism we have implemented the approach for calculating relevance presented in the previous section. The main challenge in the implementation is the generation of the Query node network, i.e. the derivation tree for a query's result that is

needed for calculating the Explanation relevance, see (13). In the next section we describe this process in detail. As we have already shown, the Collection relevance can be effectively calculated from the knowledge base itself.

2.4.1.1 Generation of a Derivation Tree

The latest version of Ontobroker[17] has a powerful explanation facility, which tracks the evaluation of the rules and the substitutions applied in them. We use this file in order to generate the derivation tree for a result.

For each evaluation process an explanation file (explanation.flo) is generated. Figure 2.16 illustrates such a file.

```
a337:Instantiation[ofRule->>r9;       dependsOnInstantiation->>{a320};
instantiatedVars->>{i(X,rst),i(Y,"Data Mining"),i(Z,"Rudi")}].
a320:Instantiation[ofRule->>r0; dependsOnInstantiation->>{a267,a316};
instantiatedVars->>{i(Per,rst),i(Thema,"Data Mining"),i(Dok,docz)};
a316:Instantiation[ofRule->>r1; instantiatedVars->>
      {i(Dok,docz),i(Thema,"Data Mining"),i(Pro,prox)}].
```

Figure 2.16: A part of the explanation file produced by Ontobroker for query `FORALL Y <- Y[researchIn ->> KM]` and result `rst`. The syntax of the explanation is as follows: each explanation instance (e.g. `a337:Instantiation`) contains the information about the rule that is evaluated (e.g. `ofRule->>r9`), the instances this instance depends on (e.g. `dependsOnInstantiation->>{a320}`) and the substitution that are performed (e.g. `instantiatedVars->>{i(X,rst),i(Y,"Data Mining"),i(Z,"Rudi")}`).

Ontobroker generates the explanation in the form of F-Logic statements, so that this file can be processed logically as well. Indeed, we define a small ontology for processing explanation statements, including the set of rules which enables the generation of the desired derivation graph. This ontology is presented in Figure 2.17.

Such an approach enables a very efficient customisation of the explanation process. The recursive rules and the negation do not impose any problems for the processing of the explanation file. The explanation file explains the original logic program, i.e. no optimisation technique (e.g. Magic set) affects the explanation process.

Figure 2.18 summarizes the whole ranking process.

The ranking approach is developed as an additional module for the Ontobroker system. The ranking module takes the list of results for the query and the explanation file as inputs. The output is the ranked list of results. The time delay introduced in the answering process is not significant.

2.4.2 Evaluation

2.4.2.1 Experiments

In order to prove the usability and validity of our approach we have performed two evaluation studies: (1) proving the quality of the presented approach and (2) its comparison with the standard approaches for query refinement.

(1) First experiment

The main strength of our approach is the quality of the provided ranking - the results we provide should be very relevant for the user's current information need. We have performed a study in which the relevance of the list of results produced by our approach is determined by domain experts from our Institute. Namely, after each query the list of results was evaluated

[17] www.ontoprise.com

by three experts on the scale 1 - 10. For each ranking the mark is composed as *level_of_relevance/ranking_level*, where *level_of_relevance* is the value an expert assigned to a refinement (1-10) and *ranking_level* represents the position in the refinement list. The goal was to determine the precision of the first *n* results, so called *precision@n*.

RULE ONTOLOGY:

```
Instantiation[
 ofRule=>>RuleThen;
 dependsOnInstantiation
       =>>Instantiation;
 instantiatedVars=>>
       i(Object,Object);
 nodeI=>>Node].

Node::Object.
Node[
 num=>>Instantiation;
 Var1=>>Object;
 Rel=>>Object;
 Var2=>>Object;
 ruleNum=>>RuleThen;
 Prev=>>Instantiation].

RuleThen :: Object.
RuleThen[
  Var1=>>Object;
  Rel=>>Object;
  Var2=>>Object].
```

TRANSFORMATION RULES

```
//all nodes
FORALL II,N N:Node[num->>II]<- II:Instantiation and concat(II, "s", N).

// for start nodes
FORALL II,N, R N:Node[Rel->>none;Var2->>none]<- II:Instantiation and
II[ofRule->>R] and concat(II, "s", N) and not EXISTS O O:RuleThen and equal(O, R) .

//for all edge
FORALL II, R,N,L, O1, O2,O3,O4,O11, O31, Rel1  N[Var1->>O2;Rel->>Rel1;
Var2->>O4;ruleNum->>R;Prev->>L] <- II:Instantiation[ofRule->>R;
dependsOnInstantiation->>L;instantiatedVars->>{i(O1,O2),i(O3,O4)}] and
R:RuleThen[Var1->>O1] and R[Rel->>Rel1] and R[Var2->>O3] and N:Node[num->>II] and
EXISTS M M:Node[num->>L].

// for end edges
FORALL II, R,N, O1, O2,O3,O4,O11, O31, Rel1 N[Var1->>O2;Rel->>Rel1;Var2->>O4;
ruleNum->>R;Prev->>none] <- II:Instantiation[ofRule->>R;instantiatedVars->>
{i(O1,O2),i(O3,O4)}] and R:RuleThen[Var1->>O1] and R[Rel->>Rel1] and R[Var2->>O3]
and N:Node[num->>II] and not EXISTS L II[dependsOnInstantiation->>L].

// for start edge
FORALL II, R, N, LL, L  N[Var1->>LL;Rel->>none;Var2->>none;ruleNum->>R;Prev->>L]
<- II:Instantiation[ofRule->>R;dependsOnInstantiation->>L;instantiatedVars->>LL]
and N:Node[num->>II] and not EXISTS O O:RuleThen and equal(O, R) and EXISTS OO
II[dependsOnInstantiation->>L] and equal(L, OO).

// for direct fact in start edge
FORALL II, R, N, LL  N[Var1->>LL;ruleNum->>R;Prev->>none] <- II:Instantiation
[ofRule->>R;instantiatedVars->>LL] and N:Node[num->>II;Rel->>none;Var2->>none] and
not EXISTS L II[dependsOnInstantiation->>L].
```

Figure 2.17: Rule ontology and transformation rules for generating derivation trees from an explanation file

We have selected 15 participants, who were PhD students in Computer Science. Each of them has made 10 queries related to the research he is familiar with. No additional instructions are given to them.

The results are presented in the

Table **2.2**. They demonstrate a high relevance, regarding the user's information need, for the most relevant refinements.

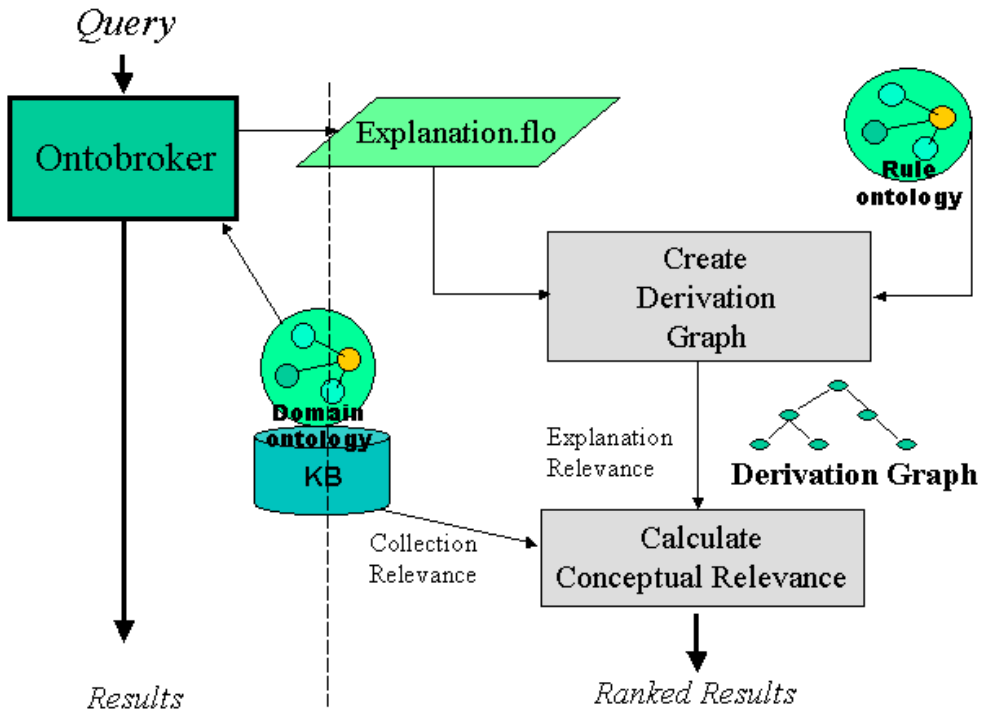


Figure 2.18: The ranking process in the Ontobroker retrieval

Table 2.2: Results of the first evaluation study, first experiment. Precision@x denotes that first x ranked results are considered. The results are normalized on interval [0,1]

Relevance Method	Precision@10	Precision@5	Precision@3
Conceptual relevance	0.85	0.91	0.98

Discussion:

Note that, since we are dealing with a logic-based approach and the retrieval approach produces all and only relevant results, the traditional precision/recall measures (in its original form) are not much useful. Therefore, we have introduced experts in order to judge how relevant the results are in the conceptual context, as we already discussed in section 2.3.3.3. Implicitly, in this experiment we are proving how well our notion of relevance performs. For example, for the query for researchers who are researching in a research area the experts analysed how relevant are these researchers for that particular area from the entire Institute point of view, therefore taking into account their experience, knowledge, research network. As the experiments have shown, our notion of relevance did succeed in modelling these complex relevance factors. Therefore, the retrieval process (Explanation relevance) and the relations between entities (Collection relevance) are very important factors in modelling a "human-like" notion of relevance.

In order to take into account which of these factors is more important for an efficient ranking, we changed the formulas given in section 2.3.3.3 for calculating conceptual relevance so that the effect of the (i) Explanation relevance or (ii) Collection relevance was dominant (and other eliminated). Table 2.3 shows the results for the same experiment we described above.

Table 2.3: Results of the first evaluation study, second experiment. Precision@x denotes that first x ranked results are considered. The results are normalized on interval [0,1]

Relevance Method	Precision@10	Precision@5	Precision@3
Explanation relevance	0.78	0.82	0.91
Collection relevance	0.81	0.85	0.91

Discussion:

The results show that none of methods for ranking performs significantly better than other ($p > 0.0001$). The Collection relevance seems to be more efficient but it could depend on the size of the knowledge base which did not enable (need) an extensive reasoning task. However, the decrease in the performances regarding the first experiment is significant for the Precision@3 and Precision@5. Therefore, our combination of relevancies seems to be a very efficient way to model the relevance in information retrieval

(2) Second experiment

In [126] we have developed a module for ranking the results retrieved by Ontobroker, based on the calculation of the similarity between terms in a hierarchy (i.e. for any transitive relation). The measure is based on the assumption that the similarity between two objects (concepts or instances) may be computed by considering their relative place in a common hierarchy *H*. *H* may but need not be a taxonomy. For instance, in our test domain we have a categorization of research topics – *H(subtopicOf)*, which is not a taxonomy. The similarity between two objects in a hierarchy is calculated using the so-called *Object Match* measure. In the following we describe very briefly this measure.

For each object we define *Upwards cotopy* (UC) as the number of objects on the paths between the given object and the root of the given hierarchy *H*. *Object Match* (OM) between two objects, O_1, O_2, is defined as

$$OM(O_1,O_2,H) = \frac{UC(O_1,H) \cap UC(O_2,H)}{UC(O_1,H) \cup UC(O_2,H)}.$$

Basically, *OM* reaches 1 when two objects coincide; it degrades to the extent to which the discrepancy between intersections and unions increases (an *OM* between concepts that do not share common super-concepts yields value 0). More details can be found in [126].

The problem in such a calculation of relevance is that a part of the domain model (i.e. the domain axioms) is not treated at all. It leads to a weak relevance model which takes into account only the hierarchy of relations in the ontology. For example, the difference of the relevance for research area `KM` between a `Researcher`, who is the leader of a `KM`-project and another `Researcher`, who is "only" a participant in a `KM`-project, cannot be expressed using our former ranking approach. For the real-world applications that we have developed by using the Ontobroker, it was a very important issue, especially in the skill-management domain [71].

By involving more semantics about the domain (i.e. axioms) in calculating the relevance, we expected that the new approach for ranking will outperform the old one. Therefore, we have set up an evaluation study in order to prove this claim.

Due to the subjective nature of relevance, it is very difficult to evaluate the performance of a ranking algorithm. Moreover, since there are no standard evaluation measures for this purpose, the comparison of the performances of two ranking algorithms is even more

difficult, and various heuristics can be applied [5]. While some attempts have been made to evaluate retrieval functions without any human judgments using only statistics about the information repository itself [120], such evaluation schemes can only give approximate solutions, and may fail to capture the users' preferences. Here we used a modification of the method for interactive comparison of two ranking algorithms, proposed in [65].

Our experiment is set up as follows:

For a query, the set of predefined answers is determined. These answers are processed by both ranking algorithms. The returned rankings are mixed, so that at any point the top *l* results of the combined ranking contain the top *ka* and *kb* rankings from *A* and *B*, $||ka - kb|| \leq 1$. The combined ranking for the given query is presented to the user, and the user is asked to select *l/2* of the most relevant results. We calculate the number of these top *l/2* results chosen from each of the two ranking algorithms, the so-called *top_results_ratio*, as

$$top\_results\_ratio(X) = \frac{top(X)}{Num(X)},$$

where *top(X)* is the number of results from the ranking *X* in the selected top *l/2* results and *Num(X)* is the total number of results from the ranking *X* presented to the user, i.e. for X=A, *Num(X)= ka* and for X=B, *Num(X)= kb*. The ranking with a higher *top_results_ratio* has a better ranking schema.

In the original method [65], the ranks are calculated in a different way: users just click on one or two results, these clicks are recorded and afterwards the clickstream data are analyzed.

For this evaluation step we used the real data from the Semantic Portal of our Institute.

For the experiment we have selected the answers of 20 predefined queries and performed the ranking of these answers by both algorithms. The queries were about researchers who research in different research area, e.g. in the F-Logic:

```
FORALL X <- X[reserachIn->>"name_of_research_area"].
```

The number of answers vary from 8 to 15. We have set the values *l*, *ka* and *kb* to 16, 8 and 8, respectively. We have used ten subject experts in the evaluation. They had no previous knowledge about the system.

In order to evaluate the additional cost of our algorithm, we have measured the processing time for all algorithms. The results of the evaluation study are given in Table 2.4. In the table wins/total is the ratio of the number of trials in which the corresponding algorithm won and the total number of trials (=200).

Table 2.4: Results of the second evaluation study

Algorithm	**Based only on hierarchy**	**Based on the full domain model**
Ranking (wins/total)	8/200	192/200
average processing time	703 ms	850 ms

Discussion:
The experiment has proved that using more semantics about the domain in a ranking algorithm improves the ranking drastically without increasing the processing time significantly. Eight losses that we have accounted in the evaluation are for the sets of answers which do not require a lot of semantics to be properly ranked. Moreover, the differences in results in these cases are very vague. For large data sets and more complex domain models (more rules) we expect even better behaviour by using our approach.

2.4.2.2 Comprehensiveness of the Approach

It can be shown that calculating the similarity in a hierarchy (e.g. the hierarchy of topics) we have used in [126] is equivalent to determining the relevance by applying our approach only on the transitive relation which models that hierarchy (e.g. on the relation `subtopicOf`). Since the transitivity of a relation is modelled in an ontology through the transitivity axiom, the derivation tree which corresponds to the query regarding that relation (e.g. "`Data Mining` is a `subtopicOf KM`") is equal to the path between these two nodes in the hierarchy. This path is used as the key factor for calculating similarity in our previous approach [126]. In other words, our previous approach for calculating similarity can be treated as a special case (when only the transitive axiom exists in the domain ontology) of our new approach (which uses more semantics - all axioms from the domain ontology).

Moreover, our approach implicitly subsumes the best practice from the research related to calculating similarity in a hierarchy, especially in the NLP (natural language processing) community [100]. Very briefly: in determining the relevance of a node regarding the root node in a hierarchy, three factors should be considered: (a) the depth in the hierarchy, (b) the density of the hierarchy at that point and (c) the strength of connotation between parent and child nodes. In our approach the strength of connotation is modelled as the *collection relevance of a resource*, the density of the hierarchy is modelled using the factor *collection relevance of a relational instance*, whereas the depth in the hierarchy is modelled as the number of the successive rules which have been evaluated for an answer (the depth of the derivation tree, see Definition 2.11).

2.4.2.3 Complexity of the Approach

Our approach is general and can be applied to any type of evaluation which relies on an abstract model of the derivation tree (like Definition 2.9 - Definition 2.11). However, the prerequisite for the approach is the external readability of derivation trees of answers which are returned for a query. Here we discuss only the complexity of the processing the derivation trees, but not the complexity of producing (externalisation of) these structures. As the results presented in Table 2.4 show, in the case of the Ontobroker system, the externalisation of derivation trees is not a time-consuming activity.
For computing the time-complexity of our approach we make the following assumptions:

- Since the number of instances in an ontology can be very large, they should be stored in a database. It means that the access to the ontology has to be considered as a time-consuming activity.
- The size of the ontology can be approximated as:
  - $\boldsymbol{l}$ – the average number of the premises in a rule
  - $\boldsymbol{k}$ – the average arity of an ontological relation
  - $\boldsymbol{g}$ – the average number of the instantiations of a relation
  - $\boldsymbol{m}$ – the average depth of a derivation tree
- We consider the case that $\boldsymbol{n}$ results are retrieved for the user's query

The time-complexity for the calculation of the *collection relevance for a resource*, is $\boldsymbol{\Omega}(\boldsymbol{g}*(\boldsymbol{k}-1))$ – it is the number of accesses to the ontology.

The time-complexity for the calculation of the *collection relevance for a fact*, is $\boldsymbol{\Omega}(\boldsymbol{g}*(\boldsymbol{k}-1)*\boldsymbol{k})$.

The complexity for processing the derivation tree of a result, (10)-(14), is $\boldsymbol{\Omega}(\boldsymbol{l}^{\boldsymbol{m}} *\boldsymbol{g}*(\boldsymbol{k}-1)*\boldsymbol{k})$.

Finally, the time-complexity of ranking all $\boldsymbol{n}$ results is $\boldsymbol{\Omega}(\boldsymbol{n}*\boldsymbol{l}^{\boldsymbol{m}} *\boldsymbol{g}*(\boldsymbol{k}-1)*\boldsymbol{k})$.

However, the exponential parameter, m - the average depth, is in practice very low (<10), as well as parameter l. Further, for our application domain, Semantic Web, the value of the parameter k is 2. The only parameter which can be very large is the number of instances, g, i.e. m, l, k $l^m \ll g$. Therefore, the time-complexity of our ranking approach is $\Omega(n*g)$, i.e. it is linear regarding g. It means that our approach scales well with increasing the number of instances in the ontology.

2.4.3 Extensions of the Approach

The presented approach exploits full semantics of the domain model (i.e. an ontology) in order to achieve an efficient ranking of retrieved results. Moreover, due to its conceptual structure, the approach can be extended with some other methods in order to get a more comprehensive retrieval approach. We mention here only three extensions of the basic approach: (i) clustering the results of the search process and (ii) ranking based on the frequency of the usage of information and (iii) search for similar results (querying by example).

2.4.3.1 Clustering

The clustering of the results retrieved for a query can be a very efficient technique for increasing the likelihood that a relevant resource will be found in the search process. Since the relevance of a result is calculated using the corresponding derivation tree, it is possible to calculate which part of the tree (i.e. which rule or relation-instance) contributes to the relevance the most. Furthermore, it is possible to group the retrieved results according to this calculation. For example, in that way, in the test domain we found that the relevance for the result `nst` for the query `FORALL x <- x[researchIn->>KM]` arises from his teaching activities, whereas the result `rst` and `gst` are relevant due to the work in a KM-related project. Such a topic-sensitive relevance enables e.g. efficient generation of communities of practices [89], informal groups of people who share the same interest and skill.

2.4.3.2 Usage Analysis

Usage is a very intuitive interpretation of the relevance. When an information resource is often used in the context of a user's information need (e.g. a query), then it is very likely that the information resource is relevant for his or her need (i.e. query). This factor is not fully exploited in the Web IR research. The only web search engine which claims that it uses this factor for ranking the results is DirectHIT[18]. However, the exploitation of usage for determining the relevance in the traditional Web is weakened due to the unknown/implicit semantics of a hyperlink. Our ranking approach can be easily extended in order to handle the semantics of the usage information. Equation (9) should be extended with the factor about the usage of the given relation instance, $U(r(I_1,I_2,...,I_n))$, as follows:

$$ColRel(r(I_1,I_2,...,I_n),I(O)) = \frac{1}{Interpretation(I_1,r(I_1,I_2,...,I_n))} \cdot ... \cdot \frac{1}{Interpretation(I_n,r(I_1,I_2,...,I_n))} \cdot U(r(I_1,I_2,...,I_n)),$$

If we consider the Figure 2.8 as the hyperlink structure of a portal (i.e. each connection is a hyperlink and each node is a web page), then in the case that the hyperlink which corresponds to statement $worksIn(gst,OntoWeb)$ is "clicked" 10 times of a total of 70 clicks from the web

[18] www.teoma.com

page related to person `gst`, then $U(r(I_1, I_2, ..., I_n)) = \frac{10}{70}$, i.e. $Col\,Rel(worksIn(gst, OntoWeb), I(O)) = \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{10}{70} = \frac{1}{42}$.

It can be used for the fine-tuning of the value obtained for this statement originally. Thus, usage can be used for fine-tuning the relevance value.

2.4.3.3 Querying by Example

Querying by Example (QBE)[19] is a method of query creation that allows the user to search for documents based on an example in the form of a selected text string or in the form of a document name or a list of documents. This method is often used for querying databases of images. The user constructs a rough query (image) using a standard modelling package. This query is matched against the database using a similarity metric.

Related to the traditional information retrieval, our querying-by-example approach enables search for results similar to the one the user puts in the query. In other words, in search for some information objects the user forms a query by combining search terms (e.g. properties of objects) and objects which are similar to the searched objects. For example, the user's query can be: `FORALL Y <- Y:Reseracher[researchIn ->> KM] and sim(Y, meh)`.This query can be interpreted as search for researchers who research in KM and who are "similar" to person `meh`. This similarity is calculated using the clustering property (see above) of our ranking approach. If we assume that `ysu` and `meh` belong to the same relevance cluster ("relevant due to the work on a KM-related project"), the most relevant result of the query should be `ysu`.

We argue that such a style of querying can enhance the search process enormously, especially for the e-commerce application in which the user often wants a product containing some properties but which is similar to previously seen or used products.

2.5 Related Work

There are several research areas that are related to our research and in this section we discuss the most important aspects. Note that we have already discussed many related approaches while explaining the proposed approach.

2.5.1 Similarity Measures in the Ranked Retrieval

Ranked retrieval plays an important role in IR and NLP. In IR, the closest match to a query is often chosen by ranking the database objects by their similarity to query [3]. Many IR systems retrieve objects from a universe represented as a vector space whose dimensions are the features of the retrievable objects, e.g., terms found in document collections [5]. The ranking functions in these systems utilize feature frequency counts [109], Boolean feature combinations [17] or probabilistic feature distributions [14]. In NLP, the best interpretation of an input is frequently selected by ranking the interpretations induced by the input in the available knowledge base [86], [99]. Many NLP systems retrieve objects from a universe represented as a semantic network. The ranking functions in these systems utilize numerical spreading activation levels [27], shapes of activation paths [86], [91] or node activation sequences [77]. All of these approaches have introduced some interesting similarity properties we use in our approach (e.g. node activation sequences). However, all of them use a shallow domain model which does not allow comprehensive calculation of the relevance.

[19] Note the difference between this approach and query-by-example in the database area, which can be seen as a "fill-in-the blanks" method of query creation.

The closest approach to our calculation of the specificity of a relation instance (formula (9)) is the research related to the similarity between two concepts in a `isA`-taxonomy such as the WordNet or CYC upper ontology [97], [98], [100], which uses edge weights between adjacent nodes as an estimator of semantic similarity. Our approach differs from this notion of similarity in a hierarchy in two main aspects: Firstly, our similarity measure is applicable to a hierarchy which may, but need not be taxonomy and secondly, it takes into account not only commonalities but also differences between the items being compared, expressing both in semantic-cotopy terms. The second property enables the measuring of self-similarity and subclass-relationship similarity, which is crucial for comparing results derived from the inference processes, executed in the background.

The initial research in this area was based on Botafogo's work on node metrics in hierarchical hypertexts, [16]. However, our research was subsequently considerably influenced by [147] about the automatic edge weighting in the massive semantic networks.

2.5.2 Semantic Methods in Information Retrieval

The use of a more structured and machine readable information about a web document for the enhancement of the retrieval of the documents is a very promising research area in the Semantic Web community. We mention three interesting approaches.

In [116] an approach is presented for information retrieval over documents that consist of both a free text and semantic annotations with statements in DAML+OIL. These statements provide both structured and semi-structured information about the documents and their content. The developed framework, called OWLIR, advocates the interdependency of search (text retrieval/extraction methods) and inference for the precise retrieval over the semantic content. However, this approach does not consider the ranking of retrieved web documents.

A very interesting approach for processing ontology-based information is given in [89]. It exploits the connectionistic structure of an ontology in order to define connectedness metrics between instances in the ontology. The approach is applied for defining communities of practice in the system called ONTOCOPI. The insight behind the approach ONTOCOPI is that if an ontology of the working domain of an organisation is created, then the links between the instances can be measured to indicate which of them are closely related. This approach does not exploit all the semantics of a domain expressed in an ontology (e.g. rules) or which can be induced from the usage of the link structure.

In [49] a new query language for information retrieval in XML documents is described. It is based on the document-centric view of XML and supports IR-related features, which are weighting and ranking, relevance-oriented search, data types with vague predicates, and semantic relativism. XIRQL integrates these features by using ideas from logic-based probabilistic IR models in combination with concepts from the database area. The ranking is based on the measure proposed in [149], which uses structural properties of an XML tree and pattern matching for estimating the relevance. However, the semantics of elements and attributes is not used in determining relevance which is the main advantage of our ranking schema.

2.6 Conclusion

In this chapter we presented the formal model for the ontology-based information retrieval, as an extension of the existing logic-based IR models, especially in defining the notion of the relevance. We proposed a comprehensive model for the relevance, the so-called Conceptual relevance that models not only whether an information resource is relevant for a query, but moreover why (and consequently, how strong) a resource is relevant for a query. In that way, by combining the relevance regarding how (i.e. why) an information is retrieved in a retrieval

system (the so-called Explanation relevance) and how semantically is this information related to other relevant information (the so-called Collection relevance), we tried to mimic the relevance reasoning found in human beings. Moreover, our evaluation studies showed experimentally the advantages of the proposed approach.

3 Ontology-based Query Refinement

3.1 Introduction

3.1.1 Motivation

The basic information retrieval process presented in the previous chapter does not enable the user to actively participate in a search process. Indeed, the user just creates a query and the retrieval module tries to match, as perfectly as possible, this query with the existing information resources. Even in the ontology-based retrieval, where this matching is performed with the maximal precision and recall, the user usually wants (and sees the opportunities) to refine the query (and, therefore, to interact with the retrieval system) in order to be more satisfied with the list of results.

The main cause of this "paradox" is that the retrieval module (see section 2.1) deals with the user's query, which is usually just a vague approximation of the user's information need that initiated the retrieval process [110]. Therefore, the retrieval algorithm tries to calculate the most relevant results for the user' query but what the user is actually interested in is expressed in his information need.

In reality, the user has some stake in the outcome of the search. Because of that stake, the actual user is able to reformulate queries according to his domain knowledge as well as his interest in completing a given task. Depending on what is at stake, the real user may iteratively query the system until he is sure that he has made a comprehensive search. Depending on what stage of the particular task he is working, different types of queries may be formed for the same general topic. Therefore, a more complex set of circumstances is realized when information problems are considered to be part of the query and relevance judgment process than when queries are sliced away from them.

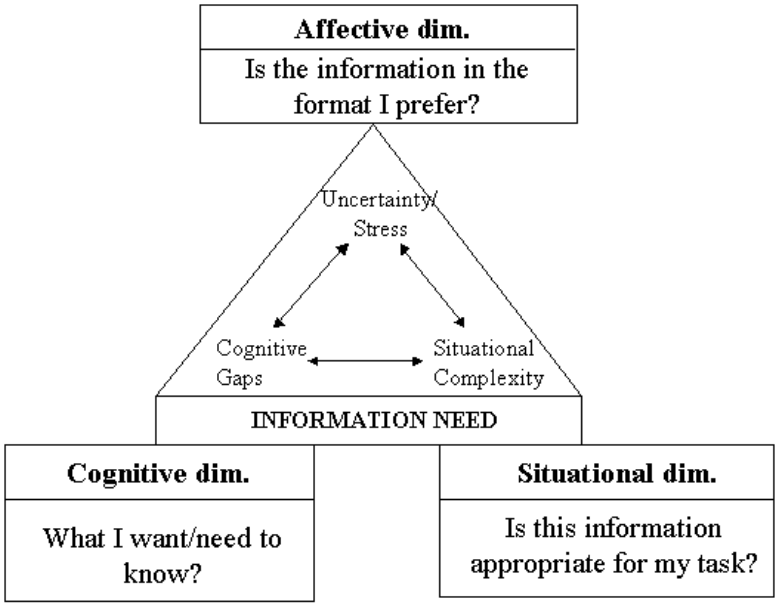


Figure 3.1: Factors influencing the creation of an information need (adapted from [30])

The mostly accepted model of the information need in the information retrieval theory is presented in Figure 3.1. An information need depends not only on the user's cognitive gap (e.g., which information he is searching for) but also his preferences (e.g. which format of the

information he prefers) and the task at hand (e.g. which problem he is trying to solve). Therefore, three dimensions of an information need can be defined:

1) **Cognitive dimension** that represents gaps or anomalies in the state of knowledge or understanding that may be represented by questions. It is driven by the performance of organizational tasks, including planning and decision making.
2) **Affective dimension** that expresses needs for achievement, for self-expression and self-actualisation. It is driven by the nature of the organization, coupled with the individual's personality structure and related to the modelling of the user's preferences.
3) **Situational dimension** that describes task, uncertainties and ambiguities encountered in specific situations (e.g. goal clarity, risk, time constraints). It is driven by the environmental uncertainty and related to modelling the task and context.

Note that each of these dimensions influences the retrieval in a particular manner. For example, the cognitive dimension is reflected on the content of the query, i.e. on those query terms that the user employs in search (like "knowledge" or "management"). The affective dimension can affect the query itself, for example, through the query terms that describe the format in which the information is presented (like the type of the file); yet, it rather affects the search process through the expectation related to the number or quality of results. The situational dimension can affect the query through some task specific terms, but it rather drives the process of inspecting particular results in order to determine their suitability for the given task. Figure 3.2 illustrates the roles that these dimensions play in the formulation of a query.

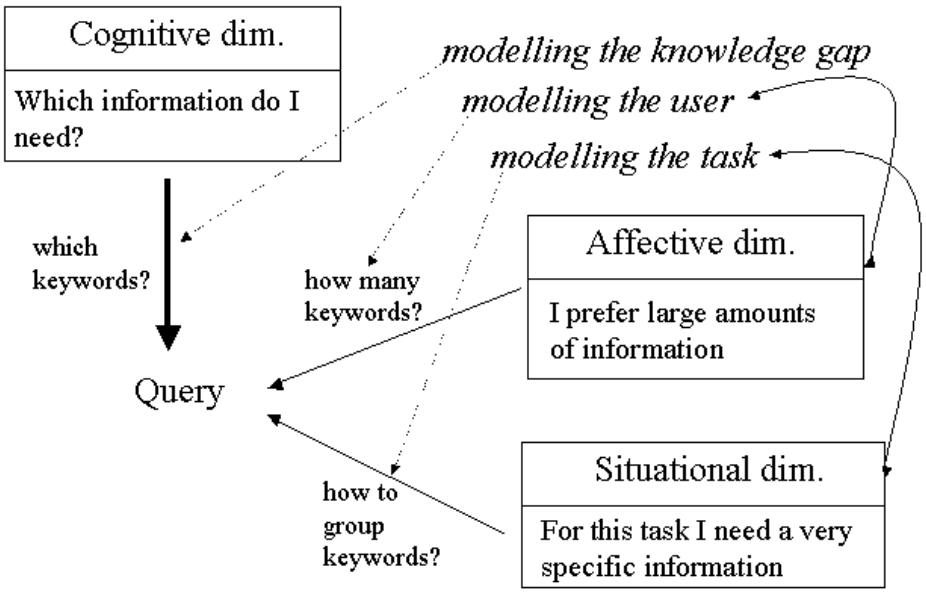


Figure 3.2: Roles of the information need's dimensions in the formulation of a query

It is clear that the process of the conceptualization of the user's need has to take into account all three dimensions we have mentioned above. However, it is hard to expect that an average user could manage this process on his own, especially in the case when he is not familiar with the information repository or with the retrieval process itself. Therefore, an IR process should enable the user to interact with the retrieval system in order to get a new insight that will help him conceptualise his need in a more precise way.

In the last twenty years several interactive information retrieval (IIR) models have been proposed [15], under the basic assumption that effective IR systems cannot be designed without some knowledge of the ways in which the user interacts with them. Each of these

models presents an alternative view to the traditional model of information retrieval in which changes to information retrieval system variables are manipulated apart from the user in laboratory situations. The inadequacies of the traditional model have been exposed by the IIR research and the main obstacle is that the traditional model does not account for the complexities of interaction: (i) among humans (e.g., users and search intermediaries); (ii) between humans and IR systems and the iterative nature of such; (iii) with respect to feedback [123].

Independently of the type of interaction, the result of the interaction process is the re-conceptualisation of the user's information need so that the new retrieval process reflects information gathered through the interaction, as illustrated in Figure 3.3.

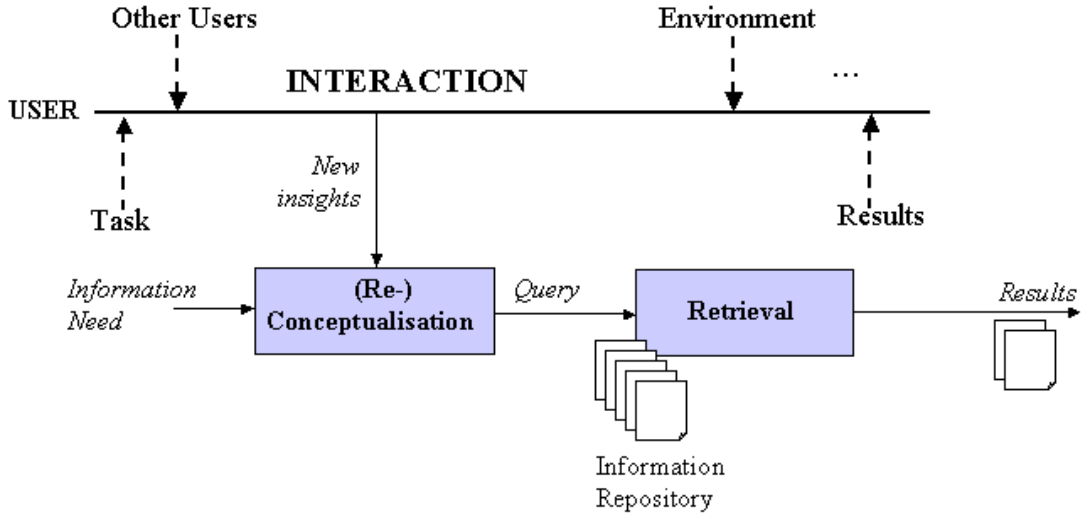


Figure 3.3: The role of the interaction in the basic IR process

In the IIR the re-conceptualisation process should be considered in a broader context. According to Bates [10], there are four levels of search activities that are performed in an interaction process and that can be changed in a retrieval process:

1) **Move**: a low-level search function (e.g. type in search term, view retrieved document). On this level only changes in the visual presentation of results are possible.
2) **Tactic**: several moves to further a search (e.g. broaden/narrow a query). On this level the changes in a query are possible as well.
3) **Stratagem**: a set of actions on a single domain (citation database, tables of contents of journals). On this level the views on the problem (like presenting Author networks, or performing Citation Search, in the process of search for publications) can be changed.
4) **Strategy**: a complete plan for satisfying an information need (e.g. subject search, browsing relevant journals, finding referenced articles). On this level the replacement of the repository or changes in the view on the task can be performed.

However, since the main initiator of an information retrieval process is the cognitive gap that is directly reflected in the user's query, the most common type of change is the reformulation of a query (i.e. a change in the tactic). Indeed, two frequently observed pitfalls in modern web IR systems [129] are related directly to the reformulation of a query:

(i) the user is usually unfamiliar with the content of the information repository. In order to avoid making an over-specified "long" query that retrieves zero results (i.e. a failing query), the user starts his search with a short query and tries to exploit the repository in several subsequent refinement steps, and,

(ii) the user often has an ill-defined information need. He starts search by assuming what the right information can be, but often, by exploring the resulting list, he redefines what he is actually searching for and modifies his query.

This process of the iterative modification of a query according to new insights observed in the interaction process in order to close the gap between the user's information need and the corresponding query is called **Query Refinement** or Query Modification process.

Moreover, all other modification actions, like the change in the retrieval strategy, require a substantially more complex retrieval system, for example the possibility to select between several information repositories[20], which is out of scope of this thesis.

Due to the same reason, we consider only these interactions that are based on the artefacts that can be found in the basic retrieval model given in Chapter 2, so that interactions with other users and with the context of the system will not be treated in this thesis.

Figure 3.4 represents the new interactive retrieval model that our research is based on.

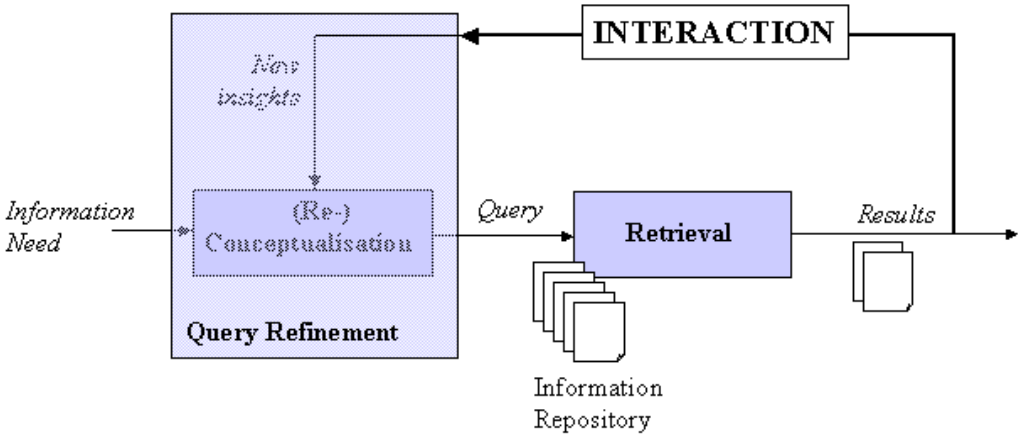


Figure 3.4: Basic interactive retrieval model

3.1.2 The importance of the Query Refinement

In order to emphasise the practical importance of a query refinement step in a retrieval process, in this section we elaborate on the complexity of the factors that can cause a need for the refinement of a query.

The problem of satisfying a user's information need in an information repository [5] 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 search for information depends on:

1. the "quality" of the information repository, i.e. whether information resources reflect the needs of users, e.g. whether 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:
   2a) if a resource which is relevant for the user's information need can be found by the querying mechanism and
   2b) if a user can (easily) find the resource which is highly relevant for his information need in the list of retrieved results.

[20] This possibility can be found in meta-search systems.

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 expressiveness of the vocabulary used in the portal. There are two factors which influence finding a relevant resource:

1) the clarity of expressing a user's information need in the query posted to the system [5], [159], since a query is just an approximation of the, often unarticulated, information need and
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, the so-called prediction game between providers and users of information, can be resolved by using a commonly-agreed, formalized vocabulary, i.e. an ontology [56].

Since users tend to read only a few top ranked resources for a query, (easily) finding an information resource that satisfies the 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 3.5 summarizes the above-mentioned discussion about the factors which influence the search for information in an information repository.

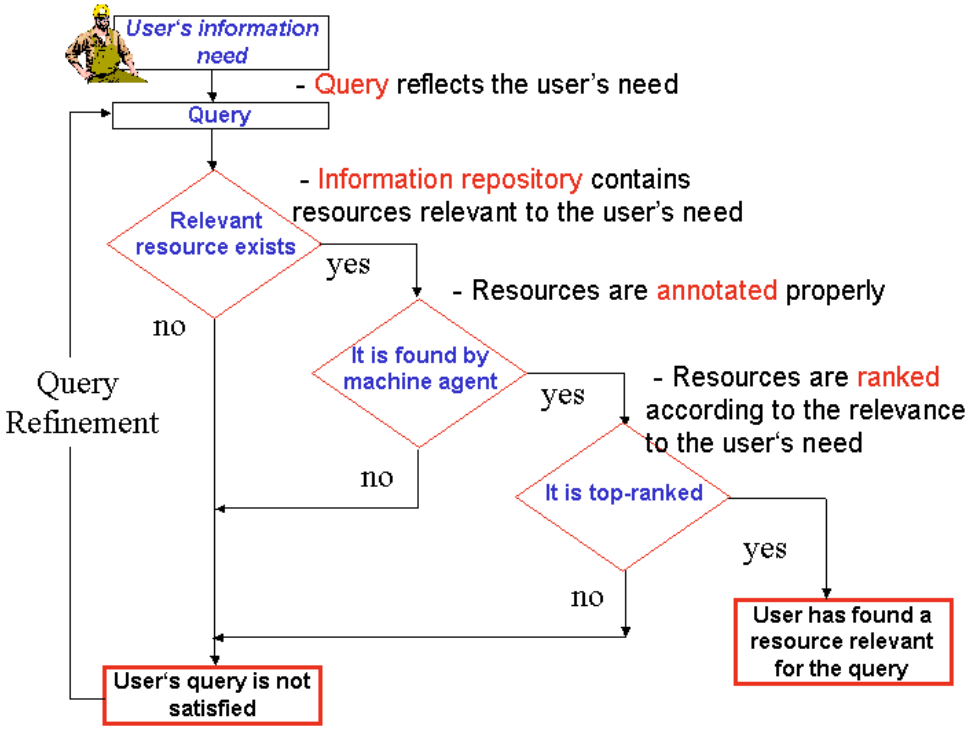


Figure 3.5: Factors which lead to a query refinement process

However, a user might not be 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 are only few relevant results and there are too many results indicating 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 the search for information. For example:

i) 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)

ii) 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

iii) 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.

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, PostDoc and PhDStudent. 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 taking into account the obvious 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 less 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 structure of the retrieval system (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 search 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 resources) 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 guess 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 sure that there are some more suitable refinements.

A recently performed large-scale case study about the behaviour of users in Web search [118] 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 [19] 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 an information repository should support users in doing refinement of their queries. The query refinement can be seen as a way to deal with (or to compensate) "problems" in an IR system, we mentioned above. For the 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 a similar meaning (e.g. a synonym: Scientist - Researcher), the user compensates for 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 search process. It enables a user to easily inspect the corresponding (i.e. query-related) part of the information repository and vocabulary, in order to determine what the causes of the arisen problems are (i.e. to determine the ambiguities in 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.

3.2 Query Refinement in the Ontology-based Information Retrieval

Since the ontology-based information retrieval model, presented in the previous chapter, focuses on the improvements of the retrieval phase only, it inherits from the basic IR model the need for an interactive improvement of the query conceptualization. Moreover, due to the formal, logic-based nature of queries:

1) the precise representation of a vague, ill-defined information need is even more difficult,
2) the anomalies which can appear in such queries (like redundancy or inconsistency) are more frequent, and,
3) the playground for modifying queries is constrained,

so that the need for an efficient query refinement support in the ontology-based retrieval is even more critical.

On the other hand, since ontologies operate on the conceptual level, they have the potential to support:

1) not only a very efficient refinement of a query (e.g. a query about "cars" can be extended with subtypes of cars, like "sportscar", "familycar" that can be represented as a concept hierarchy in an ontology),
2) but, moreover, the refinement of the information need itself (if, from the task at hand, the user generates a need and forms a query that results in no answers, e.g. by searching for "cars" that weight less than 200kg, the user should be asked whether he might be interested in a "motorbike").

Such an approach for query refinement is the topic of this section.

3.2.1 Query Refinement Basics

In this section we define the basic query refinement concepts that will be used in the rest of the text. The definitions are based on the terminology we have introduced in section 2.3.2.2 (see Definition 2.2 - Definition 2.8).

Definition 3.1: (*Types of Refinement*) A query Q is said to be refined to R if the following holds:

$$F(Q) \cap F(R) \neq \varnothing \qquad \text{for } F(Q) \neq \varnothing$$

That is, we are only interested in the query refinement as long as the query is changed so that the refined query still shares some common answers with the original query. Now, when query Q is refined to query R, the change can be categorized into one of four types:

(1) *Query Equivalency*: Q is re-written to R that generates the same set of answers with different characteristics, such as faster computation (e.g., query rewrite in the semantic query optimisation problem [63]),
(2) *Query Subsumption*: the scope of Q is restricted such that less answers will be returned,
(3) *Query Generalization*: the scope of Q is relaxed such that more answers will be returned, and,

(4) *Query Similarity*: the scope of Q is shifted to generate a partially-overlapping, but different set of answers.

Their corresponding properties are as follows:

Query Equivalency: $F(Q) \equiv F(R)$
Query Subsumption: $F(Q) \supset F(R)$
Query Generalization: $F(Q) \subset F(R)$
Query Similarity: $F(Q) \not\subset F(R) \wedge F(R) \not\subset F(Q) \wedge F(R) \cap F(Q) \neq \{\}$

As we will explain in detail later, the combination of query subsumption and query equivalence is very useful for achieving more comprehensive navigation through the information repository. Therefore, we introduce the concept of *Query Expansion*, where the definition of Q is expanded so that either less or the equivalent number of answers will be returned, i.e.

Query Expansion: $F(Q) \supseteq F(R)$

Definition 3.2: (*Query Refinement*) Query refinement is the process of search for queries that are more relevant for the user's information need than the initial query, in the case that the answer to the initial query does not meet the user's expectation.

Our definition of query refinement is based on two very reliable assumptions:

1. the user searches for information in order to satisfy an information need that is caused by his task/problem at hand [5], and,
2. a query, that the user employs in a search process, is just an approximation of his information need [110] (from the cognitive point of view – a need cannot be completely verbalized).

In other words, the user has an information need, creates an approximation of it (i.e. a query) and gets corresponding results. In an ideal case, the user can find useful information for his need in each of the information resources that is retrieved for his query. However, a query has usually an ambiguous interpretation so that the list of results does not correspond to the user's expectations and he has to perform an additional processing (interaction) of the list of results in order to find some useful results. Therefore, the user needs a view on results that can help him filter the information resources which are relevant for his information need. Since the user has already expressed/approximated his need by creating an initial query, it sounds reasonable that this additional view on relevant resources is based on the initial query. Indeed, if we make a view on relevant resources that "clusters" them according to the possible refinement of the initial query, the user gets the best opportunity to evaluate his information need – he just needs to recognize which refinement(s) is(are) most appropriate for his information need. If these refinements are ranked according to their relevance to the initial query, then the refinements that are more likely to be very close to the user's information need are on the top of the list. By selecting one of these refinements, the user actually executes a new query that approximates his information need in a better way. The user repeats this refinement process until he finds that most of his preferences are satisfied or until the system reaches some convergence point in the calculation.

Therefore, we define query refinement as an information retrieval problem, which implies that all characteristics we have elaborated in the previous chapter are valid for the query refinement as well. The most important characteristic is that the relevance of each refinement for the user's information need has to be defined.

Definition 3.2 introduces the notion of the user's expectation/acceptation in the retrieval process, which is treated in following definition.

Definition 3.3: (*Extended Query*) An extended query is a pair <*Q*, *A*>, where *A* is a Boolean function, called *Ambiguity Test* that takes as input all information relevant for the retrieval process driven by user's query *Q* (e.g. generated results), including the information from the interaction process, and returns true if the query satisfied the user's need.

This concept enables expression of constraints in the user's queries that cannot be expressed in the given ontology language. An elementary test can be to check whether there are any results for the given query.

A similar concept, the so-called *Acceptance Test*, was introduced in [28] as a mapping *A'* between query results and set {*True*, *False*}, i.e. if *A'*(*F*(*Q*)) = *True*, then the extended query is treated as an acceptable one. However, this definition ignores the interactivity, i.e. it does not enable the user to subsequently change the query. As we have already mentioned, an information need is, by nature, dynamic and could be changed during the retrieval process so that such a test cannot result in a crisp decision. Moreover, our *Ambiguity Test* encompasses all factors that could influence the acceptance of a query and not only query results.

However, we envision the *Ambiguity Test* as a step towards achieving a fully automated query refinement process: one can imagine a very complex definition of this test that includes the relationships between query results and suggested refinements of the query (like their number and quality) that can mimic the real decision process of a user (finally, a user decides to refine a query according to some information he has spotted in the retrieval process). But this discussion is out of scope of this thesis.

Therefore, the query refinement is not just an activity in which some refinement candidates can be derived, but, moreover, it is a process in which the query should be analysed in order to generate potential interesting refinements and rank them according to the perceived user's information need. Such a process is described in the next section.

3.2.2 Librarian Agent Query Refinement Process

The goal of the *Librarian Agent Query Refinement*[21] process is to enable the user to efficiently find results relevant for his information need in an information repository, even if his query does not match ideally his information need, so that either a lot of irrelevant results and/or only a few relevant results are retrieved. The process consists of three phases:

1) *Ambiguity Discovery* phase: potential ambiguities (i.e. misinterpretations) of the initial query and the whole retrieval process are discovered and assessed,
2) *Query Refinement* phase: a set of refinements that resolve the problems accounted in the first phase is generated, and,
3) *Ranking of Refinements* phase: the refinements generated in the second phase are ranked according to their relevance for fulfilling the user's information need and according to the possibility to disambiguate the meaning of the query.

In the rest of this section we present these phases in details. We will use the name *LA Process* for short.

3.2.2.1 Ambiguity Discovery Phase

We introduce the concept of *Query Ambiguity* as a gap between the user's information need and the query that results from it. It can be treated as a measure of the misinterpretation of a query regarding the user's information need. If a query is more ambiguous, then it follows that there are more (mis)interpretations of that query, i.e. the probability that the query will be interpreted in the right manner (according to the user's expectations) decreases.

[21] The name Librarian Agent comes from the analogy with the role that a human patron has in a library: he analyses received request from the user in order to suggest him the most appropriate information materials.

The ambiguity is a result of an incomplete description of the user's information need in a query as well as of the user's unfamiliarity with the information repository. In most of the existing IR systems, the user gets only the information about the number of results as the characterisation of the ambiguity: a lot of results can be an indicator that there are some irrelevant results. However, the ambiguity of a query is a more complex category and its handling requires a more complex approach. In order to model this ambiguity we introduce the concept of the *Extended Query*, based on an *Ambiguity Test* (see Definition 3.3). From the point of view of the theory of information retrieval, an *Ambiguity Test* enables an analysis of factors that influence the retrieval.

Similar to the analysis of the *Conceptualisation* phase of the basic IR process we have performed in section 3.1, the *Retrieval* phase can be analysed regarding users' cognitive aspects, as shown in Figure 3.6. Indeed, the experiencing of information needs does not always lead to information retrieval, but it is, moreover, determined through:

1) **Cognitive factors** that enable the selection of a source that is perceived to have a greater probability of providing relevant information, that is reliable and accurate.
2) **Affective factors** that are related to personal motivation and interest in the problem. They determine the energy that the user invests in seeking. The quality of the search process influences the feeling of satisfaction and accomplishment.
3) **Situational factors** that determine the amount of time and effort that is required to locate the source and to interact with the source to extract information.

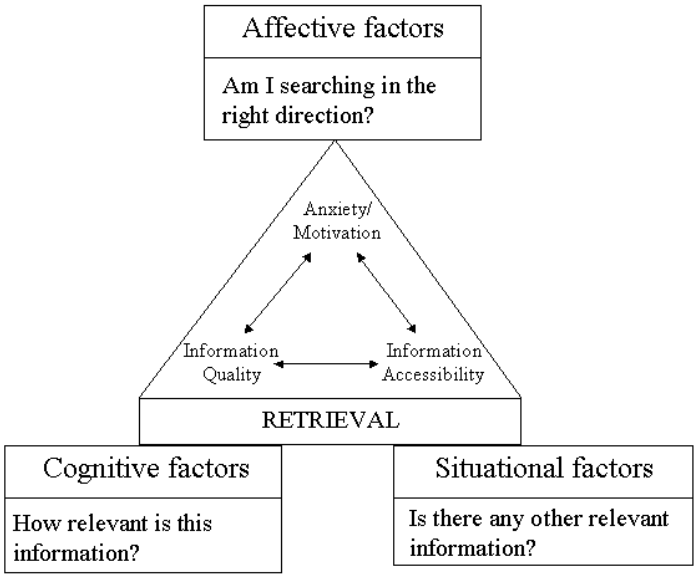


Figure 3.6: Factors that influence retrieval phase in an IR process (adapted from [30])

Note that each of these dimensions has a corresponding part in the *Conceptualisation* model presented in Figure 3.1. Therefore, we can treat each of these factors as a test for the relevance of a query for the corresponding information need's dimension, as follows:

1) **Cognitive relevance** (regarding the cognitive dimension of an information need) represents the relation between the state of knowledge/cognitive need of a user and provided information resource. It describes the informativeness and novelty of information resources.
2) **Affective relevance** describes the relation between the user's intents, goals and motivations and information resources retrieved by a system. It is the basis for the user's satisfaction with the retrieval process.

3) **Situational relevance** or utility expresses the relation between a task or problem-at-hand and retrieved information resources. It relates to the usefulness in decision-making and reduction of uncertainty regarding the given task.

As Figure 3.2 indicates and Figure 3.7 further elaborates, the quantitative evaluations of affective and situational factors (i.e. the affective and situational relevance) are related to the user- and the task modelling respectively, which is principally[22] out of scope of this thesis. However, the LA Process does enable an implicit resolving of disambiguation related to these factors by providing the corresponding *search tactics* that support the user in judging whether he is searching in the right direction and whether there are similar results, as we shall describe in the next section. The left upper part of Figure 3.8 illustrates this process.

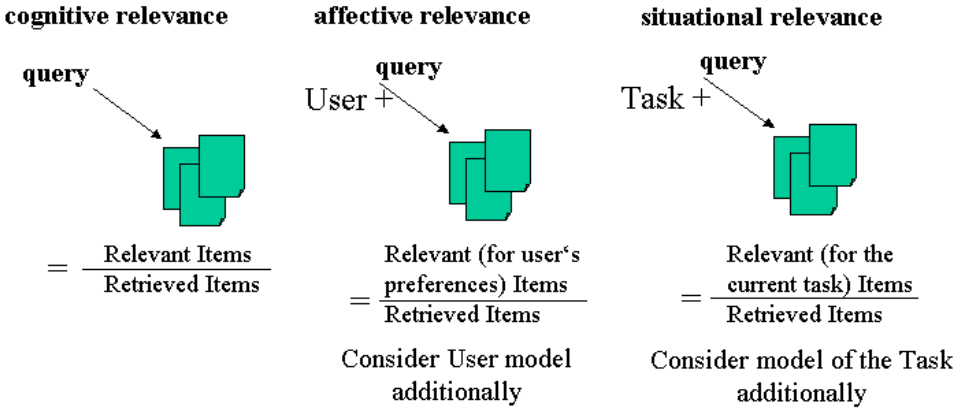


Figure 3.7: Three types of relevance regarding a user's information need

On the other side, the cognitive relevance corresponds to the notion of relevance we have introduced in Chapter 2 and will be treated as the main source for evaluating the ambiguity of a query. It is also called topical or subject relevance, namely, the relation between the topic in the query and the topic covered by the retrieved resources (aboutness).

Since, as presented in Figure 2.6, two main components of an ontology-based retrieval model are the model of the domain (i.e. the domain ontology) and its instantiation regarding a particular world (i.e. the knowledge base), we differentiate between two general types of the ambiguities *Semantic* ambiguity and *Content-related* ambiguity, which we briefly describe in the next two subsections. Figure 3.9 illustrates this discussion.

3.2.2.1.1 Semantic Ambiguity

Semantic ambiguity indicates how familiar a user is with the underlying domain model. Since there are two main entities in an ontology, i.e. concepts and relations, we have defined two general measures in order to estimate the semantic ambiguity of a query:

- considering concepts: how general/specific is a query description (so called *Generality*),

- considering relations: how strong/related are constraints applied in a query (so called *Compactness*).

The first measure estimates whether a user uses a too general description of his information need whereas the second measures shows whether he is aware of the relationships which exist between concepts. The concrete measures depends on the query language used in the retrieval process and will be described in next chapters.

[22] User modeling will be shortly discussed in section 3.2.2.3.

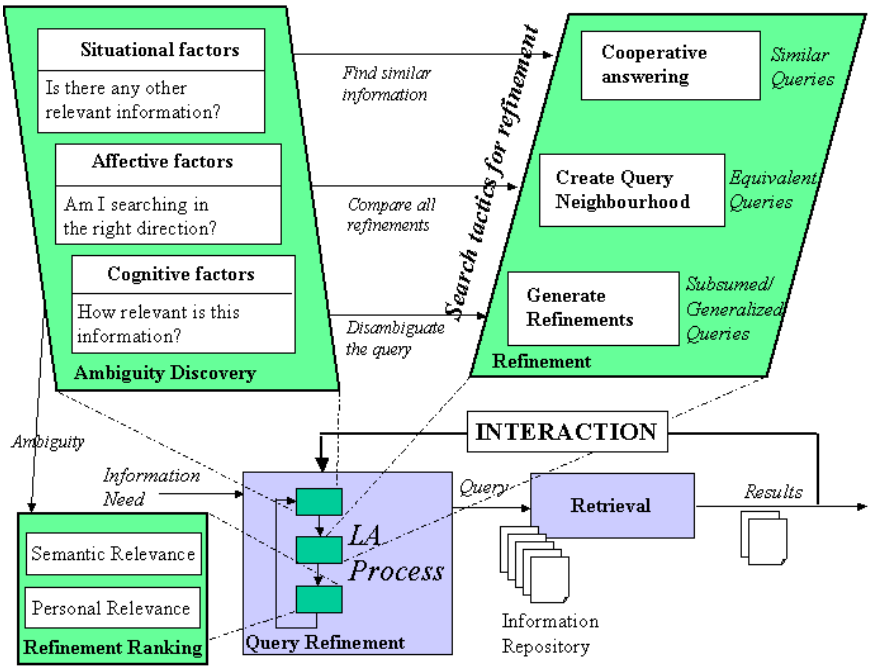


Figure 3.8: The detailed structure of the librarian agent query refinement process

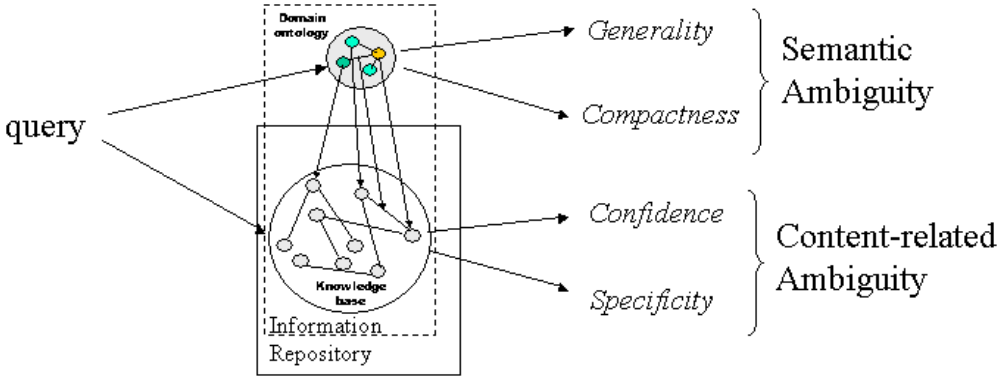


Figure 3.9: Factors that determine cognitive relevance

3.2.2.1.2 Content-related Ambiguity

Content-related ambiguity indicates how familiar a user is with the underlying information repository. This measure depends on the characteristics of the information repository, especially on how the results relevant for a user's query are distributed in the repository. Therefore, this measure requires an analysis of the results retrieved for a query.

We have defined two general measures that estimate this ambiguity:

- *Confidence*

  The traditional IR approach for selecting a term to be added to a user's query in the query refinement process is based on comparing the effects that refinement terms could have on the results [43]. In that way the specificity of the underlying information repository is taken into account for judging about the relevancy of an additional constraint for the query refinement. We adapt this experience for the case of the search in an ontology-based information repository by introducing the parameter *Confidence*.

- *Specificity*

In order to be interesting for a user, a refinement should be very relevant not only for the initial query, but moreover for other queries which that refinement depends on (see next section where a lattice-based ordering of queries is presented). The parameter *Specificity* estimates this kind of the relationships between queries.

Concrete procedures for the ambiguity measurement will be presented in next chapters, when we describe particular query refinement methods.

3.2.2.2 Query Refinement Phase

Since we treat query refinement as an information retrieval process we define several (search) tactics[23] for retrieving refinements, whereas each of these tactics corresponds to one of ambiguity tests, as illustrated in the right upper part of Figure 3.8. In particular, we define three tactics that are described in the next three subsections.

3.2.2.2.1 Refinement Tactic

Regarding *Ambiguity Test*, this tactic answers to the user's dilemma how relevant for his information need the retrieved results are. Indeed, by inspecting a list of the recommended refinements the user can judge how well his need is covered by the given query. It represents the standard tactic that can be found in query refinement approaches – according to an algorithm, a list of query refinements is generated. In the LA process, this tactic is based on the evaluation of the ambiguity of a query and results in a set of subsumed or generalized queries, depending on the user's need to get more or less results w.r.t. the list of retrieved results.

This tactic is implemented through the refinement operator *Ref*:

$Ref : \Omega(O) \rightarrow \Omega(O)^n$ ($\Omega(O)$ is a set of all the queries regarding the ontology O).

It is clear that for each type of refinements given in Definition 3.1, a special type of the refinement operator is needed.

3.2.2.2.2 Query Neighbourhood Tactic

According to the Ambiguity test, this tactic supports the user in judging whether he is searching in the right direction regarding his information need. Let us illustrate this problem on a simple example from our medical study [142]. Let us assume that when the user analyses the side-effects of using Aspirin in curing his headache, he might be interested in finding some clinical studies related to very specific cases of causing headache by taking aspirin. Such a need cannot be straightforwardly mapped into a query, since the term "specific" requires considering a query in the context of other queries. Indeed, if the user forms the Boolean query: "specific cause aspirin headache" and the list of results contains 1000 answers, then the user might be not satisfied with these results since he needs a "specific" case, which means a case not frequently reported in the clinical praxis. Consequently, he has to inspect the resulting list of answers (cases) in order to find out which case is really a "specific" one. For example, if he finds a case regarding the query "aspirin headache epilepsy" potentially interesting, then he has to determine how "specific" that case is, i.e. how many clinical studies are reported on that case. Therefore, he has to query the repository again in order to prove the "specificity" of the selected solution. In other words, he has to compare that result with the results of other potentially interesting cases (queries) and the comparison process can be recursively repeated.

From this short example it is clear that, for some information needs, the user has to inspect several queries interesting for the problem at hand in order to determine the relevance of these

[23] As we described in section 3.1, an information retrieval interaction can be considered on various levels of granularities and the search tactic is one of them.

queries for his need. All these queries are placed in the "neighbourhood" of the initial query the user has posted. The main problem for the user in such a navigation is how to orient himself in the search space, i.e. how to determine the "real" neighbourhood of his initial queries. Otherwise, the user can start testing all the changes of the initial query, which seem reasonable to him, but this can be a very tedious and time-consuming task. Moreover, it is possible that, initially, the user is not satisfied with the results of the query, but after considering the "neighbourhood" of that query, he concludes that such a query reflects his information need in the best manner. Therefore, in order to support a better estimation of how the query corresponds to the user's information need, the user has to be provided with the map of the query's neighbourhood and a "compass" which estimates the "quality" of the current position for the user's need, i.e. which enables more precise navigation through such a query map. This concept is illustrated in Figure 3.10 and described in detail in the following. Although the importance for such a navigation through the neighbourhood of a query is obvious, the systems which effectively support the user in scanning the "nearby" queries are, as known to the author, missing.

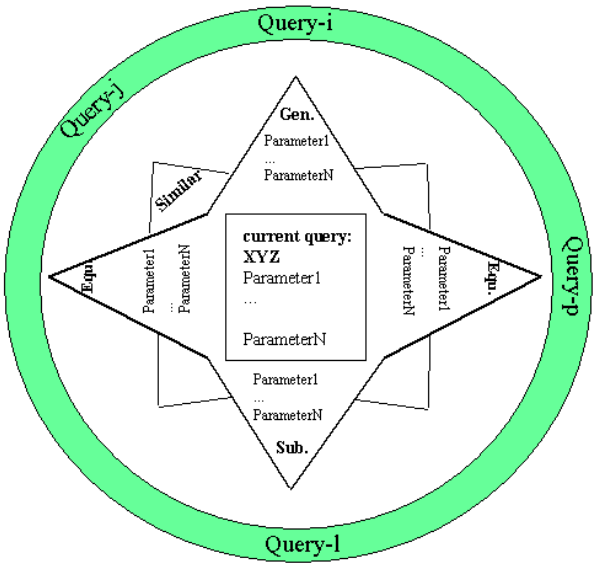


Figure 3.10: An illustration of the concept query compass

Theoretical Background

As a theoretical background for realizing "query neighborhood" we use the concept of neighborhood systems [114]. It originates from the abstraction of geometric notion of closeness: two points in a space are close to, or approximate to, each other if one point is in a neighborhood of the other. We mention here only the most important notions required to describe a query neighborhood. A more detailed description can be found in [157].

Let U denote a finite and non-empty set called the universe. For an element $x \in U$, one may associate with it a subset $n(x) \subseteq U$ called neighborhood of x. By associating a non-empty family of neighborhoods $NS(x) \subseteq P(U)$ to x one obtains a neighborhood system of x where $P(U)$ is the power set of U. A neighborhood system may be formally interpreted as an operator from U to $P(P(U))$ that maps each element of U to a family of subsets of U. If a neighborhood system consists of only one neighborhood, it is called a 1-neighborhood system.

Neighborhood systems represent the information or knowledge about relationships between elements of a universe. Intuitively speaking, the neighbors of an element are somewhat close to, or similar to that element. The notion of neighborhood systems provides a convenient and

flexible tool for representing similarity and it can be used to describe both quantitative and qualitative information. With the neighborhood system based representation of the underlying concept of similarity or distance, one may establish a solid theoretical basis for approximate information retrieval.
The common way of defining similarity is by using a binary similarity relation, like $R \subseteq U \times U$. For each two elements $x, y \in U$, we say that x is similar to y if xRy. For each element $x \in U$ elements in the following set $R_{parent}(x) = \{y | yRx\}$ are similar to x. The corresponding neighborhood system of x consists of one neighborhood and is given by $NS(x) = \{R_{parent}(x)\}$. For example, this idea can be used for defining an information retrieval system based on a thesaurus, whereas the thesaurus defines the neighborhood of an index term.

Definition 3.4: (*Query Neighborhood System*) A query neighborhood system is a 1-neighborhood system defined on set Ω and induced by the refinement operator *Ref*, i.e. $QNS(x) = Ref(x)$, for $x \in \Omega$. Since we have defined four types of the refinements, there are four types of the neighborhoods, which are depicted in Figure 3.10:

- Query-1 represents a Subsumed Query of the current query,
- Query-j represents a Similar Query to the current query,
- Query-i represents a Generalized Query of the current query,
- Query-p represents an Equivalent Query to the current query.

Parameters depicted in Figure 3.10 represent outcomes from an *Ambiguity Test*, that helps in the navigation through the repository. For example, the user can navigate in a direction in which a specific type of the ambiguity (e.g. semantic) decreases.

Ordering Induced by the Neighborhood System

Although all elements of a neighborhood of query *Q* are similar to each other (regarding *Q*) some ordering in this set can be made according to the effect that these neighborhood elements have on the retrieval process, i.e. on the set of retrieved results.

Indeed, if we introduce a binary relation ≥ on the set Ω that is a *quasi-order* relation[24] (i.e. that is reflexive and transitive), then *quasi-order* (Ω, ≥) is a lattice, i.e. for each pair $\{x, y\} \subseteq \Omega$ there is a least upper bound[25] and greatest lower bound[26] in (Ω, ≥). Consequently, such a structure introduces a lattice-order in set Ω.

If refinement operator *Ref* has this property, the lattice-order is applied to the set of query neighbours, as depicted in Figure 3.11. The main advantage of such a structuring is that the user can be provided not with an entire set of refinements, but with a minimal and complete set of refinements. "Minimal" means that none of the provided refinements can be excluded (otherwise some relevant resources will be "lost"). "Complete" means that all relevant resources can be obtained by one of refinements. Such an organization of the query neighborhood enables the user to inspect an information repository in a step-by-step manner. This refinement is called step-by-step query refinement, or query by navigation.

[24] Given set A and binary relation ≥ on A, (A, ≥) is a quasi-order if and only if:
1) ≥ is reflexive, i.e. $\forall x \in A: x \geq x$
2) ≥ is transitive, i.e. $\forall x, y, z \in A: x \geq y$ and $y \geq z \Rightarrow x \geq z$

[25] Element $x \in A$ is an upper bound of set $Z \subseteq A$ if and only if $\forall y \in Z: x \geq y$. Upper bound x of Z is a least upper bound of Z if and only if for each upper bound y of Z $y \geq x$.

[26] Element $x \in A$ is an lower bound of set $Z \subseteq A$ if and only if $\forall y \in Z: y \geq x$. Lower bound x of Z is a least lower bound of Z if and only if for each lower bound y of Z $x \geq y$.

Definition 3.5: *The Step-by-Step Query Refinement Neighborhood* of query Qx is a subset of its Neighboorhood that is minimal and complete w.r.t. the refinement operator:

$$QNS_{sbs}(Qx) = \{y \mid y \in QNS(Qx) \wedge (\neg\exists z \in QNS(Qx) \wedge y \in Ref(z))\} \quad \text{(minimal)}$$

It holds: $(\forall\ z \in QNS(Qx), z \notin QNS_{sbs}(Qx)\ (\exists y \in QNS(Qx) \wedge\ z \in Ref(y)))$ (completeness)

We are here giving an example of the query expansion. Other refinement types can be treated in an analogous way.

Definition 3.6: (*Extensional subsumption*) We introduce a binary relation ≥ on set Ω (Ω,≥) as $Q_1 \geq Q_2$ iff $F(Q_1) \subseteq F(Q_2)$.

This relation can be seen as an extensional comparison between two queries, i.e. from the extensional point of view, Q_1 is "greater" than Q_2. Since the subset relation is reflexive and transitive, relation ≥ is a quasi-order relation.
Quasi-order (Ω, ≥) is then a lattice, since the definition of the extensional subsumption i.e. Definition 3.6 corresponds to the logical implication. A proof can be found elsewhere [52]. Finally, the elements from the neighbourhood of a query can be ordered in a lattice structure as presented in the bottom part of Figure 3.11.

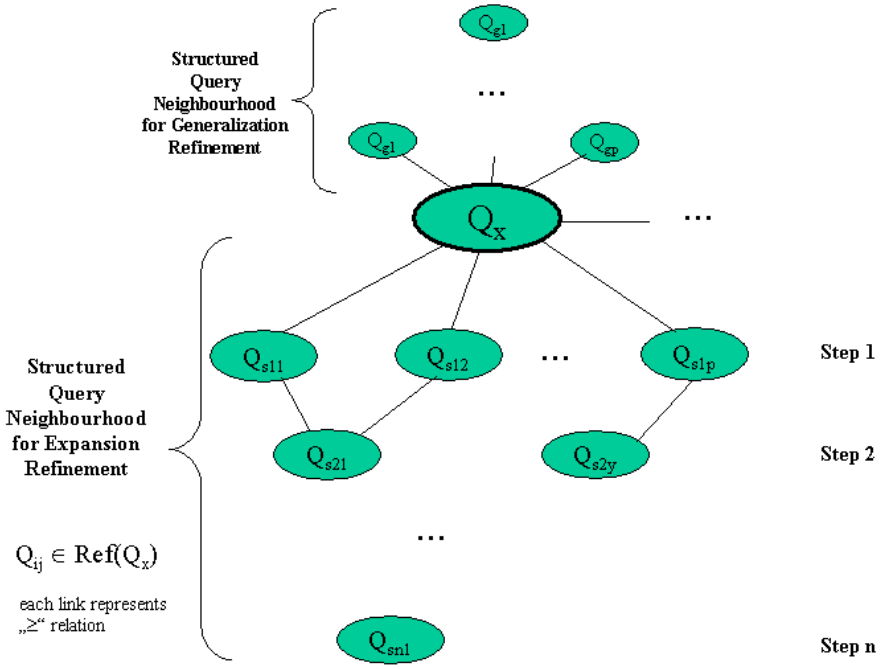


Figure 3.11: The structured query neighborhood

As we already mentioned, in this Chapter we are introducing the model that will be instantiated in the following three chapters. For an example of a concrete query neighborhood consider Figure 4.2 and Figure 5.7.

Note that this is one of the main drawbacks of the refinement systems in the current Web retrieval systems: they produce a list of refinements, but there is no criteria regarding the set of results that is optimized in that list. Consequently, the user does not get a picture of which part of the information space is covered with the refinements.

3.2.2.2.3 Cooperative Answering

Due to unfamiliarity with the information repository it is possible that the user makes a query that can be improved not only by specialization or generalization of the query's scope, but moreover by (slightly) changing the meaning of the query. This is especially important for e-

commerce applications where some characteristics of a product can significantly influence an offer, but the user might not be aware of them.

An obvious case is when there is no answer on the user's query – a so called failing query. An empty answer is surprising to the user since the user expects that there exist answers to the posted query.

Secondly, the user can change his preferences when he considers the alternatives (new products or new features). In other words, users are not aware of all preferences until they see them violated. For example, the user does not think of stating a preference for intermediate airport until a solution proposes to change airplanes in a place that he dislikes.

Finally, the user wants to be sure that the results he has achieved are the best possible solution for his need. For example, if the user is willing to buy a notebook with the Intel Pentium 4 2.4GHz processor that costs up to 1500 EURO and he finds that there is a notebook with the Intel Pentium 4 2.6GHz processor equipped with a DVD almost for the same price, then he might be very likely to decide to buy it. It is clear that in all these cases the user is "forced" to change/extend/adapt his query in order to find the most suitable product for his need. Unfortunately, most of the on-line shop portals do not provide a (cooperative) support for the query refinement and the user is left to act on his own initiative (knowledge). Although each of the on-line shopping portals makes its own view (represented as a set of initial questions the user should answer to) on the product database, the cooperativeness in the further refining steps of a search is completely missing, e.g. the user is provided with the rest of questions although some preferred/non-useful questions regarding the user's need can be discovered/induced from the past behavior (refinements) of that user. A recent experimental study [12] has shown that users, who agreed in advance to buy a product (of course, with a discount) using an on-line shopping application, have made more than one hundred search steps on average, although only twenty steps sufficed to find a suitable product. Most of these additional steps were comparisons between a selected product and its alternatives, since the users were not sure about the quality of the product they have chosen to buy until a managable number of comparisons have been made.

Therefore, a support that could help the user to change his need according to a particular situation in an information repository would be very helpful for him. Such a support will be called cooperative answering. We have found two types of cooperative answers:

1) Failing query resolution

When a query fails, a system could be more cooperative by helping to trace the reason for the query's failure, or at least to pinpoint the failure. However, the main problem is that it is possible to have a huge number of (independent) causes of a failure in a query. Consequently, finding all of them can require exponential time in the worst case [51]. Therefore, an efficient "repairing" system has to select only the most relevant *repairs* of a failing query.

Principally, an efficient method for the resolution of failing queries should ensure the minimal loss of information content that is contained in a failing query, i.e. the minimal degradation of the user's information need. Indeed, since, as we have already elaborated, a query represents a user's information need, a failing query contains some aspects of that need that cannot be fulfilled in the given repository. If the resolution process significantly discards the user's need, it might be possible that the user will not be satisfied with the resulting query. For example, for a failing query "expert system evaluation" the elimination of the term "system" can significantly change the meaning of the query so that the user should start the querying process again with a new query.

There are two important aspects regarding the above mentioned problems that can be used in the librarian agent process for resolving from failing queries:

1. Since our method supports evaluation of the ambiguity of a query, it is possible to determine which of the constraints introduces the highest ambiguity. It is reasonable to assume that a constraint with a high ambiguity has a weaker relation to the user's need than a constraint that is well defined (low ambiguity).

2. Since our method defines the neighborhood of a query according to a similarity measure, it is possible to search efficiently for an extension of the minimal repaired query in order to partially fulfill the meaning of the query that is lost due to deletion of a constraint. Indeed, by searching in the children neighbor queries of the minimal repaired one, the minimal query extension with the maximal effects on the meaning of the query can be achieved.

Bellow we give a pseudo algorithm for the resolution from failing queries that covers two of the above mentioned aspects (Steps 1, 2 correspond to discussion 1. and 2. given above). A concrete application of this algorithm can be found in section 5.3.2.

Figure 3.12 illustrates this process.

Algorithm for repairing from failing queries
Input: Q: a failing query, n: the expected number of repaired queries
Output: L: the set of repaired queries that match the user's information need

Set $L_x = \varnothing$ (for minimal repaired queries)
Set $L = \varnothing$

Step 1. *Find minimal repaired queries*
// Finds n minimal subqueries that retrieve a result
Create *all* subqueries (Q_s) by eliminating one constraint (per subquery) from the query Q.
Evaluate each query Q_s. *If* Q_s does not fail *Then* $L_x = L_x \cup Q_s$
If L_x contains more than n elements
Then measure the Ambiguity of the constraints eliminated in each subquery and select the first n with the highest Ambiguity
Go to Step2
Else If L_x contains n elements
Then go to Step2.
Else Repeat Step1 starting with a subquery that failed (if any)

Step 2. *Expand minimal repaired queries*
// Try to expand each minimal query with a constraint related to the original query
For each element Q_l from L_x
Calculate $QNS_{sbs}(Q_l)$ and select element Q_x that is most *related* [27] to the constraint from Q that is missing in Q_l.
If this *relatedness* is greater than a threshold
Then $L = L \cup Q_x$
Else $L = L \cup Q_l$

Step 3. *Ranking*
Rank elements of L according to their *Ambiguity*

Therefore, in our approach the relevance of a repair is calculated according to the loss in information that is caused by replacing/eliminating a query constraint. In that way we ensure

[27] Concrete measures for relatedness are given section 3.2.2.3.1.

that the interpretation of the new (satisfyable) query is close to the user's information need expressed in the original query.

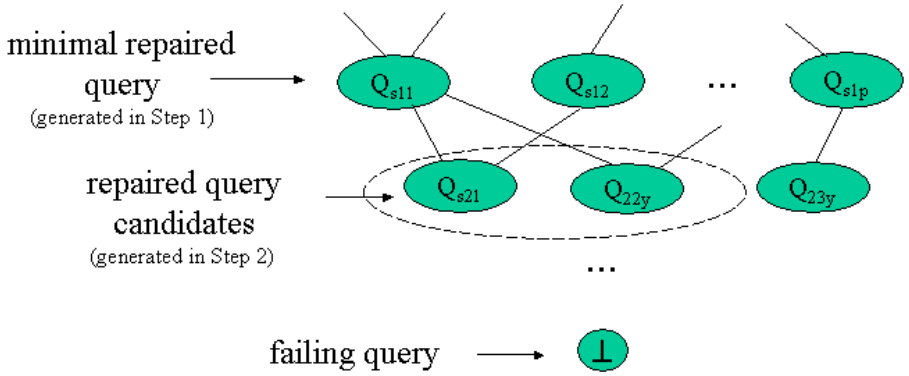


Figure 3.12: The process of resolving failing queries according to the above algorithm

2) Alternative results

An alternative result for a query is a result that does not fulfill all the constraints from the query perfectly (but fulfils some of them), but has (many) suitable features from the user's point of view (e.g. usability). In the domain of e-commerce such features are often called merchant characteristics (like price of a product, warranty, time of delivery). However, the problem is how to find which characteristics can be relaxed in the user's query and which not.

The neighborhood lattice-ordered structure indicates a similarity between queries so that an efficient approach for search for alternative queries can be implemented. Principally, the queries that "share" a parent with the given query (so called siblings query) can be considered as reasonable alternatives, since they represent the queries whose information content minimally differs from the given one. However, this difference should be compared with a threshold, i.e. the *relatedness* between constraints representing that difference has to be above a threshold. Figure 3.13 illustrates this process.

$$Alternative(Q) = \{Q_i \mid Q_i \in QNS(Q) \wedge diff(Q, Q_i) > a\},$$

where $diff(Q, Q_i)$ measures the *relatedness*[28] between different constraints and a is a threshold.

In the next sections we will present particular methods for the cooperative answering. Note that cooperative answering can be treated as a refinement of the user's information need.

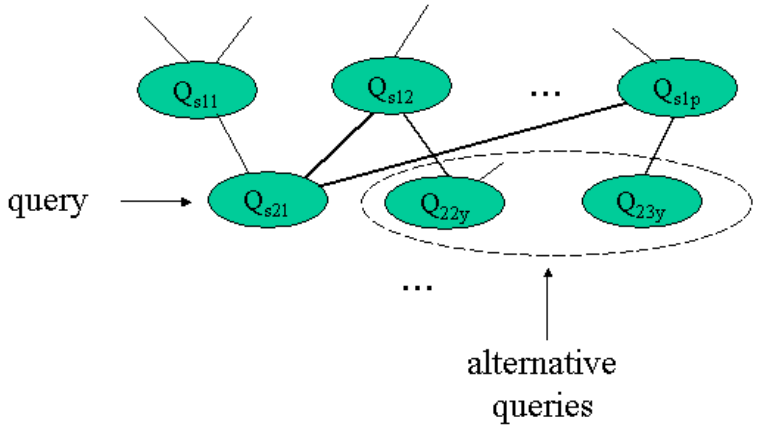


Figure 3.13: The process of finding alternative queries

[28] Concrete measures for relatedness are given section 3.2.2.3.1.

Formally, cooperative answering can be seen as a process of search for similar queries (see Definition 3.1, query similarity), i.e.

$$CooperativeAnswer(Qx) = \{y |\ F(Qx) \not\subset F(y) \wedge F(y) \not\subset F(Qx)\}.$$

3.2.2.3 Ranking

As we already elaborated in the previous chapter, relevance is one of the crucial issues in information retrieval. Since we consider query refinement as an information retrieval problem, the issue of the relevance of a particular query refinement has to be carefully treated.

However, the notion of relevance in query refinement should extend the traditional information retrieval relevance with some concepts related to the user's interaction, i.e. how is a refinement tailored to a user's behavior.

Another important aspect of the relevance in the query refinement process is that it has to be calculated w.r.t. the initial query, i.e. how suitable is a refinement for the initial query. For example, whether the most ambiguous elements/constraints of a query are refined.

Therefore, we define two dimensions of the relevance in the query refinement:

1) **Semantic Relevance** that takes into account the semantics of the domain model used for the conceptualisation. Note that, by considering the retrieval process in which a refinement is generated (explanation process), semantic relevance can be extended with the conceptual relevance we introduced in Chapter 2.
2) **Personal Relevance** that takes into account a user's preference about the problem he is trying to resolve using a retrieval process. The main idea is that ambiguities in the meaning of a query have to be analysed/discovered regarding the user's needs, i.e. regarding the resources the user is searching for. For example, it is possible that the ambiguous query "`forall x <- Car(x)`" matches very "precisely" what the user is searching for. As we already mentioned in Figure 3.2, we are interested only in the preferences that are related to a user's knowledge gap, i.e. a model of the user's knowledge/preferences about the domain involved in the search process. Since the effects of the interaction in the information retrieval process should be compared with the preference of the user, these preferences have to be modelled explicitly in the form of a user's profile. More details are given in the next subsection.

We define the relevance on the level of the query *descriptors*, whose level of the formality depends on the query language that is used in the retrieval process. In the simplest case, a descriptor is represented as a keyword. In an ontology-based query the descriptors correspond to a predicate. For example in the query

```
forall x <- Car(x) and hasType(x, FamilyCar)
```

the predicate `hasType(x, FamilyCar)` represents a descriptor (it is also called a query constraint).

3.2.2.3.1 Semantic Relevance

For a descriptor *descriptor* of the query *Q*, s*emantic relevance* can be expressed as *SemanticRelevance*(*descriptors*, *Q*).

We define two factors that determine the semantic relevance of a descriptor:

- how much improvement that descriptor introduces in the clarifying of the meaning of the query (i.e. how much ambiguity is left in the refined query),

- how close to the original user's information need is the new descriptor (i.e. how strongly is the descriptor related to the original query).

Figure 3.14 illustrates these factors.

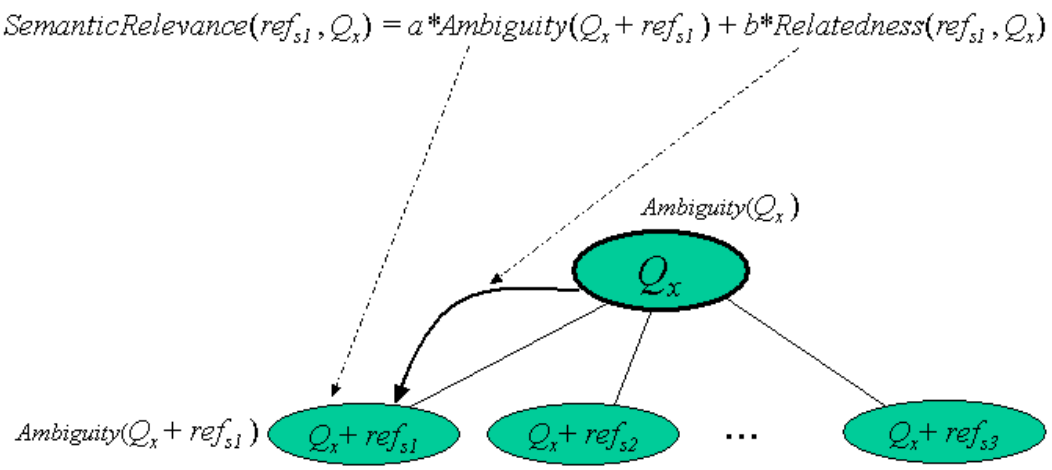


Figure 3.14: Factors that determine *semantic relevance*

Definition 3.7: *Semantic Relevance*

$$SemanticRelevance(ref, Q) = a*Ambiguity(Q+ref) + b*\ Relatedness(ref, Q)$$

where:
a and b are weighting parameters (by default: $a = b = 1$),
Ambiguity(Q) is the function that calculates the ambiguity of the query Q,
Relatedness is a function that estimates how important is a descriptor for a query, and,
Q_x+*ref* represents an extension of the query Q_x with the constraint *ref*.

See Figure 3.14 for additional explanations.

The first factor is related to the *semantic* and *content-related* ambiguity of a query, we mentioned in sections 3.2.2.1.1 and 3.2.2.1.2, respectively. In the general case this factor is proportional to the difference in the value of these ambiguities between the original and refined query. Usually, it is calculated as a weighted sum of these differences.

The second factor, *Relatedness*(*descriptor*, *query*) is related to the concept of the *Information Inference/Flow* [8], which we briefly interpret from the query refinement point of view.

Information flow

A human encountering a new concept draws it's meaning via an accumulation of experience of the contexts in which the concept appears. This opens the door to "learn" the meaning of a concept through how a concept appears within the context of other concepts.

From the query refinement point of view, the information inference can help to determine the relevance of a refinement for a user's information need. Indeed, if a refinement can be "informationally inferred" from the query, then it should be more relevant for the user's information need. For example, if a user generates the query "Car and Metallic" it is reasonable to assume that information about the "Color" of the car will be preferable for the user, since the "Color" of the car and the feature "Metallic" are conceptually similar. Anyway, regarding information flow, one can assume that the "Color" is more relevant than the feature "Automatic" for a user who selected feature "Metallic" as relevant.

Barwise and Seligman [8] have proposed an account of *information flow* in terms of state spaces which is a realization of their definition of inferential information content given in the introduction.

Definition 3.8: *Barwise-Seligman's Information Flow*

$$i_1, i_2, \ldots, i_n \mid\!\sim j \text{ iff } \cap_{n \geq k \geq 0} s(i_k) \subseteq s(j)$$

The left hand side of the formula describes a relationship between a set of types (tokens) i_1,

$i_2, \ldots, i_n$ and a type (token) j. The intuition is the information described by the combination of tokens i_1 to i_n carries the information described by j. Barwise and Seligman refer to this phenomenon as a "constraint" between the respective sets of types; the relationship can be conceived as one of information flow between the conjunction of i_1 to i_n and j. The above definition can also be applied to the earlier examples of information inference, for example: Car, Metallic |- *Color.*

The right hand side of Barwise and Seligman's definition (see Definition 3.8) describes how the inference relationship is defined in terms of an underlying state space. $s(i_k)$ is the set of states in which i_k appears and is difficult to calculate. In the example provided above, $s(i_k)$ is the set of terms in whose contexts i_k appears.

We have extended this notion by calculating the strength of an information flow as follows:

Definition 3.9: *Weighted Information Flow*

$$Strength(i_1, i_2, \ldots, i_n \mid\!- j) = \prod_{n \geq k \geq 0} Rel(i_k, j)$$

where $Rel(a, b)$ represents how two entities a and b are related.

If we consider the set $i_1, i_2, \ldots, i_n$ as a set of descriptors that defines the original user's query Q_x then j can be considered as a refinement descriptor ref_{s1} (c.f. Figure 3.14). Finally, Relatedness can be treated as the strength between a query and a refinement:

$$Relatedness(ref_{s1}, Q_x) = Strength(Q_x \mid\!- ref_{s1}).$$

Relatedness

The relatedness is defined as an $n \times n$ matrix, where n is the total number of descriptors. This matrix is called *Rel* matrix and enables the calculation of the semantic distance between descriptors.

There are several ways to calculate this *Rel* matrix:

1) an expert can define the relations between attributes,

2) relations can be "learned" from an information repository or

3) relations can be derived from the underlying ontology.

In this thesis we focus on the third issue, i.e. we use background domain knowledge to calculate how strongly the descriptor is related to the original query.

Depending on the query language different definitions of the relatedness between a query and a descriptor can be established. We here define a general relation $related(a, b)$ that defines that descriptors a and b are related to each other. In the next chapters we will define the concrete procedures for calculating the relatedness.

However, independently of the method of calculation, the relatedness between attributes can be propagated across the whole query space. If we try to calculate all the relatedness paths of length n, then we can get the relatedness between two attributes regarding the whole ontology. We describe this process briefly.

The matrix *Rel* is defined as:

$$Rel(q, q') = \begin{cases} \dfrac{1}{|\{x \mid related(q, x)\}|}, & if\ q' \in \{x \mid related(q, x)\} \\ 0, & otherwise \end{cases}$$

Each element of the matrix, $Rel(q, q_i)$, represents the probability of the "transition" from q to q_i. Moreover $\sum_{i=1,\ldots,n} Rel(q, q_i) = 1$.

Further, the matrix Rel^i represents the semantic distances of length i. $Rel^0 = I$, the identity matrix. Thus, the sum $\sum_{i=0,..\infty} Rel^i(q, q_i)$ is the probability that information flow will reach q_i from q in any number of steps.

The infinite sum converges to $(I\text{-}Rel)^{-1}(q, q_i)$ (see [151]). We will denote that matrix as Rel^{max}. This requires the calculation of the inverse matrix of *descriptors* × *descriptors*, where *descriptors* denotes the set of descriptors of a given ontology O.

As we mentioned, since these calculations depend on the query language, the concrete formulas for defining *Rel* matrix will be presented in the next chapters.

Finally, the relatedness can be calculated using following definition.

Definition 3.10: *Relatedness*

$$Relatedness(ref, Q) = \Sigma_{\forall term \in Q} Rel^{max}(ref, term)$$

3.2.2.3.2 Personal Relevance

The user's preferences can be modelled as a list of descriptions related to the content of a domain. We will use the function *Personal*(*descriptor*) to calculate the value of relevance that is assigned to each descriptor.

On the abstract level there are three levels of modelling the user's preferences:

- a general, fairly accurate, only slowly adapting **long-term profile**. This layer represents the user's general interest in some topics that are covered in the information repository. For example, the long-term profile might state that the user is quite interested in *SportsCar*, but not in the type *Porsche*,
- a more specific **medium-term profile** which represents the user's current working context. It consists of the very specific topics covering the documents in the context, characteristic attributes extracted from the metadata (e.g. authors, date) as well as a set of terms which are representative for the documents in the context. For example, a user with the long-term profile related to *SportsCar*, might be interested in documents about *Formula1*, by working in such a context,
- a very specific, quickly adapting, but possibly less accurate **short-term profile**. It represents the information need during the user's current interaction with the system and contains the same kind of information as the medium profile, but puts a stronger emphasis on document terms and keywords than on more general topics. The main difference to the medium term model is that the short term model is adapted after each interaction with the system, which means it has to cope with few and uncertain evidences as well as hard time constraints for analyzing them. For example, the short term profile for the above mentioned user can contain the keyword "*Michael Schumacher*".

Figure 3.15 depicts the characteristics of the three models regarding the accuracy of the model and the effectiveness of the retrieval.

The first two types of profiles require that the user specifies what his preferences are in advance: for a long term model, the user has to define his profile explicitly, whereas for a medium term model he has to define the context which he is working in. However, due to reluctance of users to define their preferences explicitly [160], we focus in our research on the profile that can be learned from the feedback the user generates regarding a search session. Therefore, Personal relevance takes into account only a short-term user model, i.e. it is bound

to the current query session and the current query: *Personal*(*descriptor, session, query*). See Definition 3.17.

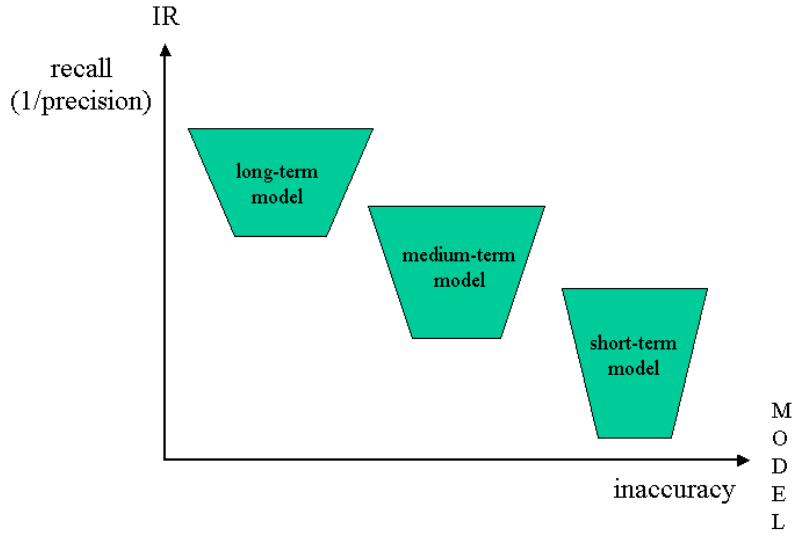


Figure 3.15: The characteristics of the three user's preference models: long-term model provides likely accurate models and high recall and low precision of the IR process, whereas short-term model provides likely inaccurate models and high precision and low recall of the IR process.

Moreover, even in e-commerce applications, where the user's profile can be well defined, it is possible that the "current" search requires more specialized information than provided by the long-term user profile. It means that the user's information need in the current search session has to be discovered (short-term personalization) [34]. Moreover, in some product areas (such as appliances or home electronic equipment) buyer interests typically only relate to a single buying session, and profile information is typically not re-used in later sessions. This is called *ephemeral personalization* [113].

In the information retrieval community a class of so-called relevance feedback [107] methods is used for determining a user's profile from his feedback. This is an iterative process where the user assesses the relevance of a number of documents returned in response to an initial query. The user peruses the content of each document in this set, assesses it for relevance and marks those that best meet his information need. The limitations in providing increasingly better ranked results based solely on the initial query, and the resultant need for query modification has already been identified. Relevance feedback systems automatically resubmit the initial query, expanding it by using terms taken from the documents marked relevant by the user. In practice, relevance feedback can be very effective but it relies on the user assessing the relevance of documents and indicating to the system which documents contain relevant information. In a real scenario, the user may be unwilling to browse to documents web pages to gauge their relevance. Such a task imposes an increased burden and increased cognitive load. Documents may be lengthy or complex, the user may have time restrictions or the initial query may have retrieved a poor set of results.

Implicit relevance feedback [105], [108] is a method to discover a user's preferences from the information that can be derived from his *usual* interaction with the retrieval system. It means that this information suggests implicitly what the user prefers, i.e. what he is exactly searching for. We expand and extend these methods according to the different nature of our ontology-based querying process.

In this thesis we focus on two types of interactions: query construction and browsing of results.

Query Construction

A user's query is usually constructed in several subsequent query refinement steps that constitute a so-called query session. Despite the fact that in most IR systems, users do not quantify the importance of a descriptor (predicate) in a query, the user does have his local preferences regarding these constraints, i.e. some constraints in a user's request are more relevant for his need than other. For example, the constraint that the color of the car should be *DarkBlue* can be less important for the user's need than the constraint stating that the car should use *Diesel*.

Such an importance of an attribute is defined on the level of a *constraint* the user has set in his request. The goal is to discover which of the constraints that the user has posted so far are more relevant for his need. Therefore, we define a parameter *Importance* of a constraint as the combination of its interpretation in the underlying user's request (*Interpretability*) and its relevance for the actual user's need (*Actuality*). Formally,

Definition 3.11: *Importance*

$$Importance(descriptor, Q, Q_s) = a*Interpretability(descriptor, Q) + b*Actuality(descriptor, Q_s),$$

where:
a , b are weighting parameters (by default: $a = b = 1$) and Q and Q_s are the user's query and current user's search session, respectively. The meaning of *descriptor* is described above.

Interpretability

The general assumption is that the user forms a query according to his current information need, i.e. that all parts of the query correspond, to some extent, to his need. However, it is possible that a query is interpreted in a repository in an unexpected manner, i.e. the execution of a query can be interpreted ambiguously: either as a more general or a more specific query.

The main indicator of misinterpretability of a query is the existence of equivalent queries for that query. The most important ones are:

minimal equivalent query:

$$min\_equal\_query(Q) = \{Q' \mid equ(Q, Q') \wedge (\neg\exists Q'' \; equ(Q, Q'') \wedge part(Q', Q'')),$$

where:
$equ(a, b)$ is true if a and b are equivalent queries
$part(a, b)$ is true if query a contains all predicates contained in query b

maximal equivalent query:

$$max\_equal\_query(Q) = \{Q' \mid equ(Q, Q') \wedge (\neg\exists Q'' \; equ(Q, Q'') \wedge part(Q'', Q')).$$

For example, if we assume that queries "*Cabriolet*" and "*Cabriolet, Metallic*" are equivalent and the user makes the query "*Cabriolet*", then the constraint *Metallic* is interpreted as an additional constraint in the user's query, but it might not be aligned to his need. Consequently, the importance of that parameter for refining the query should be reduced. Therefore, if a query constraint is a part of the *maximal equal query* of a user's query, but not of the query, then its *Interpretability* is very low (= 0).

Otherwise, if the user makes the query "*Cabriolet, Metallic*", then the system will retrieve the same results as for query "*Cabriolet*". It can be interpreted as the "ignorance" of the constraint *Metallic* in the search process. Consequently, the importance of this constraint should be increased. Therefore, by considering the constraints from the initial query that are

not contained in the *minimal equivalent query* we get a set of constraints with a high *Interpretability* (=1). The following definition summarizes this discussion.

Definition 3.12: *Interpretability*

$$Interpretability(c,Q) = \begin{cases} 0, & c \in Q_x \wedge Q_x \in max\_equal\_query(Q) \wedge c \notin Q \\ 1, & \forall Q_x \in min\_equal\_query(Q) \;\; c \notin Q_x \wedge c \in Q, \\ & 0.5, \;\; others \end{cases}$$

where c is a constraint (descriptor) from the user's query Q.

Actuality

The *Actuality* parameter reflects the phenomena, accounted in the IR research [107], that the user may change the criteria about the relevance of a query term, when encountering newly retrieved results. In other words, the constraints most recently introduced in a user's query are more indicative of what the user currently finds relevant for his need. We model it by using the analogy to the ostensive relevance proposed in [21] using the following definition.

Definition 3.13: *Actuality*

$$Actuality(c,Qs) = \frac{1}{num\_steps(c,Qs)},$$

where:
c is a query constraint, Qs is the current query session and $num\_steps(c, Qs)$ is the number of refinement steps, which the constraint c is involved in as a part of the query. Note that $num\_steps(c, Qs) > 0$.

Browsing of Results

The theory about the implicit relevance feedback postulates that if the user selects a resource from the list of retrieved results, then this entry corresponds, to some extent, to his information need. As a list of results we consider a list of resources or a list of query refinements. However, a click on a particular result in the list cannot be treated as an absolute relevance judgement, since the user typically scans only the top l ranked ($l \approx 10$) results. For example, maybe a result ranked much lower in the list was much more relevant, but the user spotted it sooner. It appears that the user clicks on the (relatively) most promising results in the top l, independently of their absolute relevance. However, if we assume that the user scans the list of results from top to bottom, the relative relevance is evident: all non-clicked-on results placed above a clicked-on result are less relevant than the clicked-on result. If a result is an information resource, the relevance is related to some features that are contained in the clicked-on resource and not contained in non-clicked-on resources. It means that by analysing the commonalities in the attributes of results the user clicked/not clicked on, we can infer more information about the intension of the user in the current query session. In order to achieve this, we define the relation *preferred ranking* $<_{rQ^*}$ as:

$$R_i <_{rQ^*} R_j \text{ for all pairs } 1 \le i < j, \text{ with } j \in C \text{ and } i \notin C,$$

where:
$(R_1, R_2, R_3, \ldots)$ is a ranked list of results,
set C contains the ranks of the clicked-on results and
Q is the posted query.

ImplicitRelevance

By analysing the difference between the features (attributes, constraints) of the clicked-on and non-clicked-on resources we get a set of so called *Preferred* constraints for query Q in the following manner:

$$Preferred(Q) = \cup_j Preferred_j(Q),$$

where:

$Preferred_j(Q) = \{el \mid el \in Attr(R_j) \setminus \cup_i Attr(R_i), \forall i\ R_i <_r Q^*\ R_j\}$,

$$Attr(R_x) = \begin{cases} R_x & \text{if } R_x \text{ is a query refinement} \\ \text{set of constraints (attributes) that are defined for } R_x, \text{ if } R_x \text{ is an information resource n} \end{cases}$$

The set of constraints that seems to be relevant for the user in query Q_s can be calculated using the following definition.

Definition 3.14: *Implicit Relevance*

$$Impl Rel(c, Qs) = \begin{cases} 0, & c \notin Preferred(Qs) \\ \frac{1}{k} \sum_{i=1,k} \frac{1}{n - step(c, Q_S, i)}, & c \in Preferred(Qs) \end{cases} \quad (1)$$

where:

k is the number of refinement steps in which constraint c is involved as a preferred constraint in the current session Qs,

$step(c, Qs, i)$ is the function that returns in which refinement step (counting from the original query, i.e. 1, 2, …) constraint c has appeared for the i-th time,

n is the total count of refinement steps in Qs. Note $n > steps(c, Qs, i)$.

In this way we decrease the likelihood that a preferred constraint is still relevant if it was suggested as relevant in a previous refinement step, but was not selected by the user as relevant in the subsequent refinement step (see (1)). Therefore, our approach has the self-improvement nature – it learns from its failures.

ImplicitIrrelevance. Since the recommended refinements are presented to the user in decreasing order of relevance, one can assume that if the user has selected the n-th ranked result, then the first n-1 ranked results (constraints) are wrongly ranked on the top of the list of the refinements. We call these constraints *implicit irrelevance*. They are calculated in a similar manner as implicit relevant constraints.

Definition 3.15: *Implicit Irrelevance*

$$ImplIrrel(c, Qs) = \begin{cases} 0, & c \notin Nonpreferred(Qs) \\ \frac{1}{k} \sum_{i=1,k} \frac{1}{n - step(c, Q_S, i)}, & c \in Nonpreferred(Qs) \end{cases}, \quad (2)$$

where:

$NonPreferred(Q) = \cup_j NonPreferred(Q)j$

$NonPreferred_j(Q) = \{el \mid el \in \cup_i Attr(R_i) \setminus Attr(R_j), \forall i\ R_i <_r Q^*\ R_j\}$.

The definition of $step(c, Qs, i)$ is analogue to (1), but regarding implicit irrelevance.

Similar to (1), formula (2) enables the correction of false assumptions (regarding the preferences of the current user) made in the ranking process.

Formulas (1) and (2) ensure the self-adaptivity of the ranking system, e.g. they do not allow that the system repeatedly ranks a non-interesting refinement highly. Finally, the calculation of the relevance of refinement *c* is given in the following definition.

Definition 3.16: *Feedback Relevance*

$$Feedback\ Relevance(c, Qs) = \frac{Impl\ Rel(c, Qs) + 1}{Impl Irrel(c, Qs) + 1}$$

where *c* is a refinement in the current session *Qs*.

Finally, the Personal relevance is calculated as a combination of the *Importance* of a query term for a refinement and *FeedbackRelevance* of that refinement.

Definition 3.17: *Personal Relevance*

$$PersonalRelevance(ref, Q, Q_s) = \Sigma_{\forall term \in Refinement(ref)} a*Importance(term, Q, Q_s) + b*FeedbackRelevance(ref, Q_s),$$

where:
Refinement(*c*) is a function that retrieves terms which a new refinement *c* depends on. This function shows which query terms are caused by refinement *c* and *a* and *b* are weighting parameters (by default: $a = b = 1$).

The total relevance of a refinement is calculated as the combination of semantic and personal relevance.

Definition 3.18: *(Total) Relevance*

The total relevance of refinement *ref* for query *Q* in query session *Qs* looks like:

$$(SemanticRelevance(ref, Q) + \lambda*PersonalRelevance(ref, Q, Q_s))/(\lambda+1)$$

where:
λ is a forgetfulness coefficient that models the impact on the past user's behavior on the ranking process:
$\lambda = 0$ - the past is forgotten,
$\lambda < 1$ - the past carries less weight than the present, usually $\lambda = 1/2$.

3.2.3 Three Types of Ontology-based Query Refinement Approaches

So far we have defined a generic query refinement process (the so-called librarian agent process), depicted in Figure 3.8, without specifying in detail the role of an ontology in each phase of the process. Indeed, depending on the nature of the information repository, the ontology can play different roles in the query refinement process and these roles will be discussed in the next chapters in detail.

In principle, the role of an ontology is to support conceptual processing of all the factors that influence the query refinement process: the ambiguity should be measured on the ontological level, the generation of the query neighborhood should be driven by the ontology and the analysis of the implicit user feedback should be ontology-based.

In the scope of this thesis we have explored three different types of query refinement approaches, depending on the level of the formalization of the query language and information repository. Figure 3.16 depicts these situations:

1) **Full-fledged ontology-based Query Refinement** deals with the ontology-based queries against a knowledge base, according to the ontology definitions given in the previous

chapter. This refinement exploits the full potential of an ontology-based structure for the query refinement. It is described in the next chapter.

2) **Ontology-supported keyword-based Query Refinement** treats the keyword-based queries against an information repository whose resources are annotated (indexed) with terms from an ontology. The query refinement profits from the well-defined annotation structure provided by an ontology. This is the topic of Chapter 5.
3) **Conceptual Query Refinement** possesses no special requests on the retrieval structure: queries are keyword-based and the information repository is not additionally indexed. In such a situation, our query refinement approach derives semantics from the plain text and builds an ad-hoc conceptual structure that is used in the refinement process. This type of refinement is described in Chapter 6.

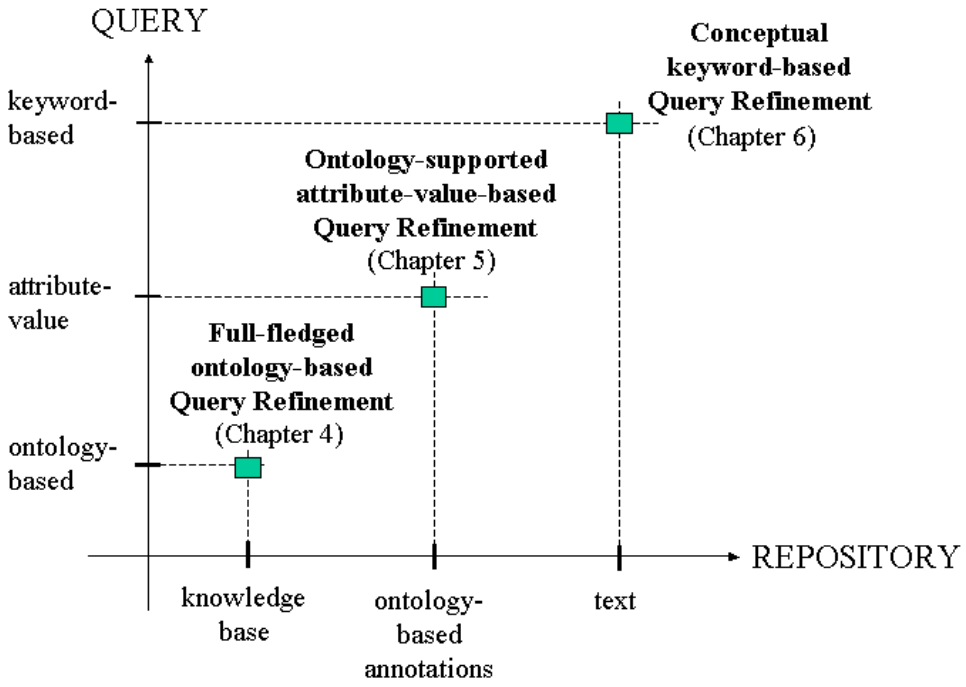


Figure 3.16: The query refinement space

3.3 Conclusion

In this chapter we discussed the importance of an incremental query refinement process for realizing an efficient information retrieval system, especially from a non-experienced user's point of view (which is usually the case in traditional IR systems). We investigated cognitive, affective and situational factors that affect the development of a user's information. Moreover, we have developed an ontology-based query refinement process, the so-called Librarian Agent Query Refinement Process. In this three-phased process we (i) detect what can be a problem in a query (i.e. we determine the ambiguity of a query), (ii) generate a minimal and complete set of refinements that resolves these problems and (iii) rank the refinements such that the user can easily develop his information need further (i.e. he can recognize easily which refinement is most relevant for his particular need). One of the main advantages is the possibility to realize some more advanced models of refinement, like cooperative answering in which the system cooperates with a user in resolving his information need.

This general model will be instantiated in the next three chapters for the three most general information retrieval cases, respectively:

- the full-fledged ontology-based refinement, required for search on the Semantic Web,

- attribute-value ontology-based refinement, suitable for product-catalog applications and
- conceptual keyword-based refinement, applicable for traditional keyword-based search.

4 Full-fledged Ontology-based Query Refinement

4.1 Motivating Example

In this section we are giving a motivating example in order to demonstrate the specific requirements for query refinement in the case of the ontology-based information retrieval as defined in Chapter 2, section 2.4. Table 4.1 illustrates the Car ontology, which is considered in this example. For example, a vehicle can be either a car or a motorbike and a car can have several characteristics (e.g. `hasType`). A visual representation of this ontology is given in Figure 4.1. Table 4.2 represents a set of statements (i.e. a part of a knowledge base) regarding our motivating example. For example, the car c5 is a `FamilyCar` and has a `GPS` system as a `hasFeature` property.

Table 4.1: Car ontology

Ontology	

`isA(Car, Vehicle)`
`isA(Motorbike, Vehicle)`
 	
`hasType(Car,CarType)`
`hasFeature(Car, Feature)`
 	
`isA(SportsCar, CarType)`
`isA(FamilyCar, CarType)`
`isA(MiniCar, CarType)`
 	
`isA(Color, Feature)`[29]
`isA(Luxury, Feature)`
`isA(Cabriolet, Luxury)`
`isA(Metallic, Luxury)`
`isA(Automatic, Luxury)`
`isA(GPS, Luxury)`
`isA(Petrol, Feature)`
`isA(MerchantFeature, Feature)`
`isA(Price, MerchantFeature)`
	`hasColorValue(Color,ColorValue)`[30]
	`hasColorType(Color,ColorType)`
	`sub(ColorValue, ColorValue)`[31]
	
	`hasLuxuryQuality(Luxury, LuxuryQuality)`
	
	`hasLanguage(GPS, Language)`
	`hasGPSType(GPS, GPSType)`
	`hasLevel(Automatic, Level),`
	`hasAutomaticType(Automatic,AutomaticType)`
	
	`hasValue(Price, Value)`
	`spendLitters(Petrol, Value)`
	`hasType(Petrol,PetrolType)`

Example 1)

Let us consider that a user is interested in buying a car, but he is not sure which features of a car to select. He wants to inspect which cars are available at all in that very moment and makes quite a general request `"?Car"`[32]. This is a classical example of an ill-defined user's need, where the user should be advised (by a shop assistant) to specify his need incrementally [128]. Since there are several cars (`c1` – `c14`), the user needs some help in refining his query. The traditional approach in the refinement of such a request is to ask the user about the values of all remaining features [117], [144], e.g. regarding this example, about values for the features `CarType` and `Luxury`[33]. Therefore, the user will be provided with six options (i.e.

[29] It means that a Color is a type of Feature(s), i.e. there is a hierarchical relationship between them.
[30] It means that the concept Color has a property hasColorValue whose range is ColorValue.
[31] This relation enables modeling a hierarchy between instances of the concept ColorValue.
[32] Due to simplicity we do not present an ontology-based representation of queries in this subsection. However, the translation is straightforward. E.g. the query for a car that has a luxury feature, depicted as "? `Car + Luxury`", corresponds to "`forall x,y • Car(x) and Luxury(y) and hasFeature(x, y).`"
[33] Note that only the Luxury hierarchy is used for describing features of cars.

`SportsCar`, `FamilyCar`, `Metallic`, `Cabriolet`, `Automatic`, `GPS` – probably visually grouped in two clusters, `CarType`[34] and `Luxury`) from which he should select the most relevant one.

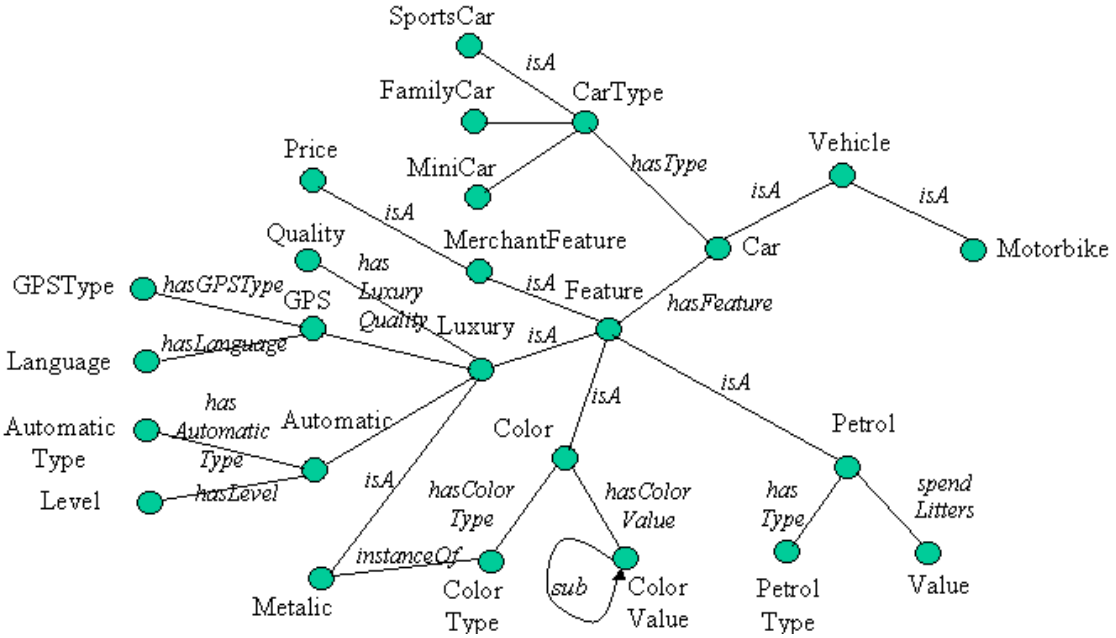


Figure 4.1: Graphical representation of the (part of) ontology given in Table 4.1

However, such an approach has the drawback that the user does not get a more complex interpretation of these refinements in order to fulfil his need in the most appropriate manner. Indeed, it is possible that between the feature values there exist some inclusion relations, such as those in the given example: (i) all `Cabriolet` cars are `Sportscar`, (ii) all cars that have `GPS` are of the type `FamilyCar`, or (iii) a car with the `Cabriolet` and `GPS` features does not exist. An efficient system should take these dependencies into account. Let us assume that the user prefers a `Cabriolet` to a `SportsCar`. In the traditional approach the user can select the value `Cabriolet` but such a value causes the automatic selection of the (non-preferred) feature `SportsCar`. Therefore, the system has failed to help the user inspect the repository in a step-by-step manner that will avoid retrieving non-preferred or zero results. Note that repairing from failing queries can be a very time consuming activity [48].

In order to avoid this problem for the given dataset, in the first refinement step the user should be provided with a list of only two values for the `Luxury` (instead of all four), i.e. `Metallic` and `Automatic` as well as two values of the `CarType` of the car. All other values for `Luxury` are clustered in one of these four clusters (consider Figure 4.2 for a better understanding). This clustering of queries should be driven by an internal conceptual structure of the information repository being searched (the so-called lattice-based clustering – see Figure 4.2) in order to avoid drawbacks introduced by hierarchical clustering [24]. Now, the above mentioned user (who prefers `Cabriolet` to `SportsCar`) can adapt the satisfaction of his need more appropriately to the content of the repository: he does not "see" the feature `Cabriolet` in the first refinement step and tries to satisfy his next preference (i.e. not a `SportsCar`) by selecting `FamilyCar` value. Therefore, the user should be provided with a minimal list of values that he should consider in the first step. Obviously, it is possible that by

[34] The knowledge base does not contain any `MiniCar` car.

such a selection the user changes his need when he considers the alternatives (new products or new features), since the user is not aware of all preferences until he sees them violated[35] [51].

Table 4.2: A simple dataset used in motivating example

Knowledge Base
Car(c1),hasType(c1,SportsCar[36]), hasFeature(c1, Cabriolet)
Car(c2),hasType(c2,SportsCar), hasFeature(c2,Metallic)
Car(c3),hasType(c3,SportsCar), hasFeature(c3,Metallic)
Car(c4),hasType(c4,SportsCar), hasFeature(c4,Metallic)
Car(c5),hasType(c5,FamilyCar), hasFeature(c5,GPS)
Car(c6),hasType(c6,FamilyCar), hasFeature(c6,Automatic)
Car(c7),hasType(c7,FamilyCar), hasFeature(c7,Automatic)
Car(c8),hasType(c8,FamilyCar), hasFeature(c8,Automatic)
Car(c9),hasType(c9,FamilyCar), hasFeature(c9,Automatic)
Car(c10),hasType(c10,FamilyCar), hasFeature(c10,Metallic)
Car(c11),hasType(c11,FamilyCar), hasFeature(c11,Metallic)
Car(c12),hasType(c12,FamilyCar), hasFeature(c12,Metallic)
Car(c13),hasType(c13,FamilyCar), hasFeature(c13,Metallic)
Car(c14),hasFeature(c14,Automatic)
LuxuryQuality(“High”), LuxuryQuality(“Low”)
hasLuxuryQuality(Metallic,“High”[37]), hasLuxuryQuality(Cabriolet,“High”)
hasLuxuryQuality(Automatic,“Low”), hasLuxuryQuality(GPS,“Low”)
ColorValue(“BlueColor”)[38], ColorValue(“DarkBlue”),
ColorValue(“WhiteBlue”) , ColorValue(“GreenColor”)
sub(“DarkBlue”,“BlueColor”) , sub(“WhiteBlue”,“BlueColor”)
ColorType(“Metallic”)[39], ColorType(“Standard”), ColorType(“Protected”)
LuxuryType(“Cabriolet”), PetrolType(“Diesel”)

Therefore, in such a system the user can develop his information need in a step-by-step manner, which is a very successful strategy when the user searches through an information repository which he is not familiar with [21]. Indeed, the user often assumes wrongly the existence of some constraints in the repository. A good query refinement system should inform the user about such “false” assumptions and should present to him all possible *interpretation*s of his query regarding the repository.

Example 2)

Let us consider another example of the refinement of the query `"?Car"`: let us assume that the user who has made the query is not yet sure about which concrete type of `Luxury` to select, but he has preferences about some properties of the `Luxury`, e.g. the vendor, the quality or the price of a luxury equipment. Therefore, instead of selecting a concrete type of the feature `Luxury` (i.e. `Metallic`, `Cabriolet`, `Automatic`, `GPS`), the user prefers to select a property of a `Luxury` (in the given example the `hasLuxuryQuality` property that models the quality of the luxury). However, in the modern product catalogue systems such a possibility is completely missing since the relations between features of a product are not given explicitly. If we assume the usage of an ontology for modelling product data, we can reason about the features of the products. Considering the ontology in Table 4.1 and data in Table 4.2 and the above mentioned lattice-based clustering, in the very first refinement step the user should choose only between `"High"` and `"Low"` quality of a `Luxury`. One can imagine that for such a question the user has a concrete answer, e.g. it might be that he wants a `SportsCar` with the

[35] For example, the user does not think of stating a preference for any intermediate airport until a solution proposes to change airplanes in a place that he dislikes.
[36] Note that meta-modeling is applied.
[37] Note that default-values are used.
[38] It means that one instantiation of the concept ColorValue is “BlueColor”.
[39] Note that the Metallic concept defined as a subconcept of the Luxury concept and the “Metallic” instance defined as an individual of the ColorType concept are different entities in the Car ontology.

best quality of any equipment he selects, but he has failed to express such a need in his original query. In that case (i.e. the high quality of a luxury), the additional refinements for `Automatic` and `GPS` are irrelevant, since the quality of these features is not a high one (consider Table 4.2). In other words, the refinement system should provide the user with different views on the product data in order to enable him to satisfy his need in the most appropriate manner.

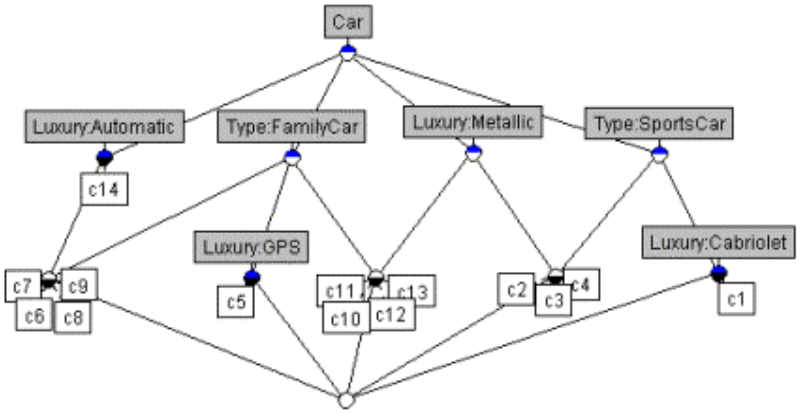


Figure 4.2: Lattice-based clustering of dependencies between product' features for the knowledge base depicted in Table 4.2. Note that a node encompasses all resources from its children nodes and all attributes from its parent nodes. E.g., node {(`Luxury:GPS`), (`c5`)} contains the attribute (`Type:FamilyCar`) as well.

Example 3)

Let us assume a user who is interested in a family car, but does not have other preferences. An efficient system should help the user inspect the repository (i.e. cars `c5 – c13`) efficiently in order to find the most appropriate result. It is clear that there are three possible refinements of the feature `FamilyCar`: `GPS`, `Automatic` and `Metallic` (see Figure 4.2). However, the system should interpret these refinements in order to suggest to the user the most appropriate one, as follows:

Since there is just one `GPS`-car (of nine retrieved `FamilyCar` cars) this refinement is not the most appropriate for a step-by-step refinement. Regarding `Automatic` and `Metallic` there are four cars in each refinement. Therefore, if the user wants to see which properties are *found in most* of `FamilyCar` cars, he should select one of these two refinements. However, an efficient system should be even more cooperative: it should recognise that the feature `Automatic` *is more characteristic* for `FamilyCar` cars than the feature `Metallic`, since 4/5 cars that possess feature `Automatic` are `FamilyCars` and only 4/7 of the cars with feature `Metallic` are `FamilyCars` (another 3/7 are `SportsCars`). It might mean that `Automatic` is a feature designed exclusively for this type of cars. Therefore, if the user is interested in `FamilyCar` it makes sense to recommend him first a feature that can be found (almost) only in that type of cars.

An efficient query refinement system should rank the refinements according to these comments. Moreover, the system should learn the user's preferences by observing his search behavior (e.g. which resources the user has selected for viewing in the current query session), developing in that way a so-called short-term user's profile [34], which is used for the personalization of the query refinement process.

Hence, a query refinement system should go beyond an ad-hoc generation of the candidates (or a particular candidate) for the refinement. Moreover, it should be treated as a (derivation) process that generates a set of refinements in a sound and complete way and ranks them according to their relevance for achieving the user's need. In the next section we formalize the

requirements for such a query refinement process. As we have already mentioned in the previous chapter, we focus on the query expansion type of the refinement.

4.2 Requirements

As we have already explained in section 3.2.1, one way of defining a *query expansion operator* is in terms of a relationship between the original query and the transformed ("expanded") one:

Definition 4.1: *An expanded query* $<Q', A>$ expands $<Q, A>$ (in the notation $<Q, A> \rightarrow_{ext} <Q' A>$, or as a shorthand[40] $Q \rightarrow_{ext} Q'$), if in the context of the given knowledge base *KB* and the ontology *O* it follows: $F(Q') \subseteq F(Q)$.

However, in a refinement process it is possible that the expanded query *Q'* introduces a new variable which has to be substituted in the $F(Q')$, like, regarding Table 4.1 and Table 4.2, Q = "`forall x <- SportsCar(x)`" and Q' = "`forall x, y <- SportsCar(x) and Luxury(y) and hasFeature(x, y)`". In that case, set inclusion "$\subseteq$" between results does not hold. Therefore, we need a partial interpretation of the subset relation, which we realize by projecting the set of results of Q and Q' on a set of common variables (F) in notation F_P. P is usually a set of variables used in original query Q (the projection is then depicted as F_Q). In the rest of this section we implicitly assume the usage of such a projection (i.e. instead of F_Q we write F). In an information retrieval scenario (e.g. search a product catalogue) it is possible to define a "carrier" of the information need of the user, i.e. which type of resources (i.e. concept) the user is searching for. (Note that other query variables define the characteristics of that resource). In that case, each query contains a variable which corresponds to that concept. In the previous query Q it is variable x. In fact, this variable is called the *key* variable and the corresponding projection is depicted as F_{key}. This assumption decreases the complexity of the calculation significantly. In the rest of the text for each statement $F(Q)$ we assume its projection on the key variable, i.e. $F_{key}(Q)$.

We take a convention that the first mentioned variable in a query is interpreted as its *key* variable. The usage of a *key* variable can be found in similar information retrieval tasks. We mention just two examples:

- The formal approach for answering conjunctive queries on Semantic Web [47], [60] is based on (i) translating a query into an equivalent concept expression (e.g. in Description Logic notation), (ii) classifying this new concept and (iii) using standard inference methods to check whether an object is instance of a concept. Obviously, this concept corresponds to our key variable.
- Some recent approaches in the frequent pattern discovery [40] use the key parameter in order to specify what has to be counted, i.e. they focus "counting" on just one variable (e.g. either customers or employees, etc.).

Therefore, a refinement operator $\rightarrow_{ext}$ derives a set of extensions for a query Q, or more formally: $Q \rightarrow_{ext} \{Q' | F(Q') \subseteq F(Q)\}$. Consequently, in each step the user is provided with *a set of expansion Q'* which enables him to search in a step-by-step manner.

In order to define the usefulness of a set of derived refinements more formally, we introduce several properties of an efficient query refinement process which are derived from the general discussion about step-by-step refinement we have given in the previous chapter. All examples are related to Table 4.1, Table 4.2 and Figure 4.2.

[40] In the rest of this thesis we will *implicitly* assume that each refinement is related to an *Ambiguity Test*

1) *Completeness w.r.t. relevant results*. Since in a query refinement the number of results for query decreases, the process that supports query refinement should ensure that the results that are relevant for the user will not be discarded in the refinement. It means that each answer *a* from the $F(Q)$ will be retrieved for at least one refinement *Q'*.

$$\forall a \in F(Q)\ \exists Q'(Q \rightarrow_{ext} Q') \wedge a \in F(Q').$$

For example, if we consider query "`forall x <- FamilyCar(x)`" and the set of its refinements: {(`FamilyCar(x) and hasFeature(x, Automatic)`", "(`FamilyCar(x) and hasFeature(x,Metallic)`"}, then the result `c5` is not available in the refinements although it is a result of the initial query and, therefore, this set of refinements is not complete. To make the set of refinements complete, we add `FamilyCar(x) and hasFeature(x, GPS)`.

2) *Soundness w.r.t. relevant results*. Additionally, in order to avoid providing more irrelevant resources than those retrieved for the original query, only the answers relevant for the original query should be generated in a query refinement. In that way, we ensure that in each refinement step the precision of the retrieval system will not decrease.

$$\forall Q'(Q \rightarrow_{ext} Q') \wedge \forall a \in F(Q') \rightarrow a \in F(Q).$$

Note, that the definition of the refinement process assumes this soundness requirement. However, this definition allows the generation of an unsatisfiable refinement that retrieves an empty set of results (so-called *trivial* refinements). Therefore, we introduce the following extension in order to ensure that each refinement will lead the user to fulfill his information need.

3) *Soundness w.r.t. relevant refinements*. From the refinements point of view, only relevant refinements, i.e. refinements which lead to a non-failing query, should be provided to the user, i.e.:

$$\neg\exists Q'(Q \rightarrow_{ext} Q') \wedge F(Q') = \{\}.$$

For example, if we consider query "`forall x <- FamilyCar(x)`", the refinement {(`FamilyCar(x) and hasFeature(x,Cabriolet)`)} returns zero results. Therefore, it is a *trivial* refinement.

4) *Maximally general relevant refinements*. As we have elaborated in the motivating example, the user should be provided with most general refinements in order to support fine-tuning of his needs. Therefore, we should support the so-called *step-by-step* query refinement, which generates only maximally general relevant refinements in a step:

$$\forall Q'(Q \rightarrow_{ext} Q') \neg\exists Q''(Q \rightarrow_{ext} Q'') \wedge (Q' \rightarrow_{ext} Q'').$$

In that way, we ensure that the user will be provided with the minimal set of refinements in a refinement's step.

For example, the following set of the refinements of the query "`forall x <- Car(x)`":

{ (`Car(x) and hasFeature(x,Automatic)`),
(`Car(x) and hasFeature(x,Metallic)`),
(`Car(x) and hasFeature(x,GPS)`,
(`Car(x) and hasType(x,FamilyCar)`,
(`Car(x) and hasType(x,SportsCar)` }

is not minimal since the third refinement (`GPS`) is subsumed by the fourth one (`FamilyCar`).

5) *Ranking of refinements*. It is possible to define relevance for each refinement. If the query refinements are ranked according to their relevancies for the user's current need, he can navigate through the query space in a more efficient way. Therefore,

   $\exists\eta:(\Omega, \Omega)\rightarrow\Re$,

   where Ω is a set of all queries. In other words,

   $\forall Q'(Q\rightarrow_{ext} Q') \rightarrow 1 \geq \eta(Q',Q) > 0$.

   For an example consider Example 3) from the previous section.
6) *Query Neighbourhood*. In order to realize step-by-step query refinement we have to define a subsumption relation that induces a lattice structure. The main request is that this relation is a quasi-order relation on the set of queries.

In the next section we present an instantiation of the Librarian Agent query refinement process, given in the previous chapter, which fulfils these requirements.

4.3 Full-fledged Librarian Agent Query Refinement Process

According to the Librarian Agent process (see Figure 3.8), there are three phases in a query refinement process, which we describe in the next subsections.

4.3.1 Phase 1: Ambiguity Discovery

Expansion is a query modification operator that strengthens the query. Therefore, the set of answers generated by the transformed query is a subset of the answers produced by the original query. The problem in the expansion is to define an *Ambiguity Test* (see section 3.2.1) precisely, since this test corresponds to that part of a user's information need, which he did not succeed in expressing in the query in a clear way. For example, it is possible that the user wants a car that has some infrequently fabricated features[41] and his acceptance criteria can be, for instance, that if fewer types of cars have such a feature, then the car types possessing that feature are more acceptable. Therefore, the ambiguity test should be related to the comparison between several options and a refinement system has to enable the user to "inspect" several queries which are similar (slightly refined) to the original query.

We find two main factors that cause the ambiguity of a query:

a) **the ontology**:
   E.g., if in an ontology, the concept `CarType` related to the concept `Car` through the property `hasType`, is modelled through three subconcepts `SportsCar`, `FamilyCar` and `MiniCar`, then the query for a car, i.e. `"forall x <- Car(x) and hasType(x,y)"` can be (mis)interpreted as a user's need for a (i) Sport-, (ii) Family- or (iii) Mini-car.
b) **the information repository**:
   E.g., if in an ontology-based information repository each resource that has the characteristic `Cabriolet` has also the characteristic `Metallic` and vice versa (it means that adding the constraint `Metallic` in the query `"?Cabriolet"` does not change the list of results), then the user's query for resources that are `Cabriolet` can be (mis)interpreted, regarding query results, as a query for: (i) `Metallic` and `Cabriolet`, (ii) `Metallic` or (iii) `Cabriolet`.

Therefore, as we already mentioned in section 3.2.2.1, we define two types of the ambiguity that can arise in interpreting a query: (i) the *semantic ambiguity*, as the characteristic of the used ontology and (ii) the *content-related ambiguity*, as the characteristic of the repository. In the next two subsections we are giving more details on them.

[41] For an example consider the discussion in Example 3) in section 4.1.

4.3.1.1 Semantic Ambiguity

The goal of an ontology-based query is to retrieve the set of all instances which fulfill all the constraints given in that query. In such a logic query the constraints are applied to the *query variables*. For example, in the query:

`"forall x <- Car(x) and hasType(x, FamilyCar)",`

`x` is a *query variable* and `hasType(x,FamilyCar)` is a *query constraint*. The stronger these constraints are (by assuming that all of them correspond to the user's need), the more relevant the retrieved instances are for the user's information need. Indeed, since a query is just an approximation of the user's need, an unambiguous query can be treated as a very good approximation of that need. Therefore, in order to define the semantic ambiguity of a query, we need a measure to estimate how strongly each of the query variables is constrained. It can be interpreted as how ambiguously the user's information need is represented when using that query variable.

Moreover, query variables can be treated as conceptual descriptors of the user's information need. For example, the query:

`"forall x, y <- Car(x) and hasFeature(x, y) and Luxury(y)"`

is about cars (variable x) and the feature luxury (variable y). By measuring the ambiguities of these variables we can determine if the user should be asked to specify more details about the general characteristics of the car or about the characteristics of the luxury.

Since an instance in an ontology is described through (i) the concept it belongs to and (ii) the relations to other instances, we see two factors which determine the semantic ambiguity of a query variable:

- the concept hierarchy: How general is the concept the variable belongs to (how many interpretations does it have)? E.g. the query "`forall x <- CarType(x)`" is more ambiguous than the query "`forall x <- SportsCar(x)`". Note that all the presented examples are related to Table 4.1 and Figure 4.1.
- the relation-instantiation: How descriptive/strong are constraints applied to that variable. E.g. the query "`forall x <- Car(x) and hasType(x,FamilyCar)`" is more ambiguous than the query "`forall x <- Car(x) and hasType(x,FamilyCar) and hasFeature(x,Metallic)`".

In other words, to define the semantic ambiguity of a query, we check how many constraints are applied to each query variable and how specific each of these constraints is. Consequently, we define the following two parameters in order to estimate these values:

Definition 4.2: *VariableGenerality*

$$\mathit{VariableGenerality}(X) = \mathit{Subconcepts}(\mathit{Type}(X)) + 1,$$

where *Type*(*X*) is the concept C the variable *X* belongs to, *Subconcepts*(*C*) is the number of subconcepts of the concept *C*. For example, for the query "`forall x <- CarType(x)`", *VariableGenerality*(*X*) = 4, in the case that there are three subconcepts of the concept `CarType`.

Definition 4.3: *VariableCompactness*

$$\mathit{VariableCompactness}(X,Q) = \frac{|\mathit{Relation}(\mathit{Type}(X))|+1}{|\mathit{AssignedRelations}(\mathit{Type}(X),Q)|+1} \cdot \frac{1}{|\mathit{AssignedConstraints}(X,Q)| - |\mathit{AssignedRelations}(\mathit{Type}(X),Q)|+1}, \quad (1)$$

where:

Relation(*C*) is the set of all relations defined for the concept *C* in the ontology,
AssignedRelations(*C,Q*) is the set of all relations defined in the set *Relation(C)* and which appear in the query *Q*. Note *AssignedRelations*(*C,Q*) ≥ 0,
AssignedConstraints(*X,Q*) is the set of all constraints related to the variable *X* that appear in the query *Q*. Note *AssignedConstraints*(*X,Q*) ≥ 1.

Note that *VariableCompactness*(*X, Q*) > 0 and that smaller values indicate less ambiguity in interpreting that variable; e.g. for the query `"forall x <- Car(x) and hasType(x, FamilyCar)"` we have *VariableCompactness* (*X*)=(3/2)*(1/2), by assuming that there are two relations defined for the concept `Car`, i.e. |*Relation*(`Car`)|=2. Note, `Car(x)` is included in the set *AssignedConstraints*(*X,Q*).

The second factor in (1) is introduced in order to model the influence of multivalued relations. For example for the query `"forall x <- Car(x) and hasFeature(x,Metallic) and hasFeature(x, Cabriolet)"` we have *VariableCompactness*(*X*)=(3/2) * (1/3).

The total ambiguity of a variable is directly proportional to the compactness and indirectly proportional to the generality of a variable ambiguity. Note that the second parameter is greater than 1. We now define the ambiguity as follows:

$$Ambiguity(X,Q) = \frac{VariableCompactness(X,Q)}{VariableGenerality(X)} \quad (2)$$

Finally, the *Semantic Ambiguity* for the query Q is calculated using the following definition.

Definition 4.4: *Semantic Ambiguity*

$$SemanticAmbiguity(Q) = \frac{1}{|Var(Q)|} \sum_{x \in Var(Q)} Ambiguity(x,Q) \ ,$$

where:
Var(*Q*) represents the set of variables that appear in the query *Q*.
|*a*| represents the cardinality of the set *a*.

By analysing these ambiguity parameters it is possible to discover which of the query variables introduces the highest ambiguity in a query. Consequently, this variable should be refined in the query refinement phase. However, the previous analysis is based only on the structure of the ontology and the assumption that every structure in the ontology is instantiated (e.g. for each concept there is at least one instance). Indeed, it is possible that according to the structure of a query, a query variable should be further constrained, but such constraints do not exist (are not instantiated) in the information repository. For example, in the query `"forall x <- Color(x) and hasColorValue(x, "BlueColor")"`, the variable `x` has a high value for *VariableGenerality* and should be further constrained. This can be done by constraining its generality via adding another relation which is defined for the concept `Color`, for example `hasColorType(x, "Metallic")`. However, in the case that there are no cars which have `"BlueColor"` and have `"Metallic" color type`, it should be avoided to recommend such a "no-effect" refinement to the user. Therefore, in order to recommend a user only refinements which are useful, we have to measure the ambiguity of the query regarding the content of the information repository, which is the topic of the next subsection.

4.3.1.2 Content-Related Ambiguity

As we have seen in the last example, the ontology structure is just one component for determining candidates which can replace an ambiguous query in order to close the gap between that query and the user's real information need. One can say that the semantic

ambiguity estimates the "theoretical" ambiguity of a query. Moreover, it enables us to quantify that ambiguity very precisely and to recommend some refinements by determining which parts of the query are more ambiguous than other parts. However, an ontology defines just a model how the entities from a real domain should be structured. If there is a part of that model, which is not instantiated in that domain, then that part of the model cannot be used for calculating ambiguity. Therefore, we should use the content of the information repository to prune results from the ontology-related analyses of a user's query.

From the content point of view, the results of a query can be used for defining potential ambiguities which arise in the query process. For example, if two queries have the same list of results, then that list of results can be treated as an ambiguous entity – it can be (mis)interpreted as a result of two different queries. However, since the user posts a query and wants to refine it without changing directly the list of results, we will interpret all the content-related ambiguities on the level of the user's query. Regarding the previous example, two queries that return the same list of results are treated as equivalent queries. In that case, after posting a query, a list of equivalent queries can be presented to the user as an indicator of the content-related ambiguity of his query. Therefore, the content-related ambiguity of a query can be measured by comparing the results of the given query with the results of other queries.

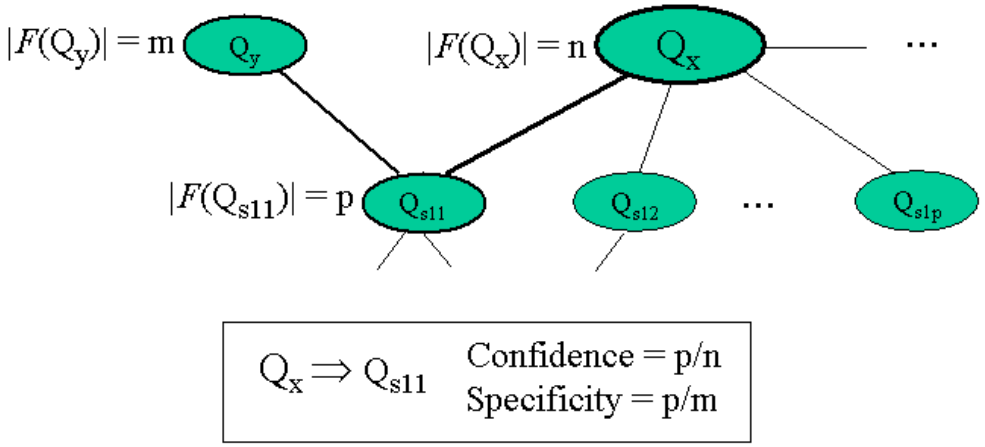
Moreover, one possible interpretation of the relations between neighbors in a lattice structure is through association rules [119], as depicted in Figure 4.3.
In section 3.2.2.1.2 we have introduced two parameters for estimating this ambiguity: *Confidence* and *Specificity*. These parameters enable the resolution of the relevance problem we have mentioned in example 3) in our motivating example.

The *Confidence* has a similar meaning as in the traditional association rule analysis, i.e. as the confidence in association rules[42]. Therefore, this value is calculated as the ratio between number of results in query *Q* and query *Q'* as given in the following definition.

Definition 4.5: *Confidence*

$$Confidence(Q', Q) = F(Q)/F(Q')$$

This parameter shows how strong the refinement *Q*' is. In other words, it is the estimation of how likely the user, who makes query *Q*, would refine the query into *Q*'.



[42] Association rules are rules of the type $X_k \Rightarrow Y_k$, ($X_k \cap Y_k = \{\}$), so that measures for support and confidence exceed user-defined thresholds. Thereby, support of a rule $X_k \Rightarrow Y_k$ is the percentage of transactions that contain $X_k \cup Y_k$ as a subset, and confidence for $X_k \Rightarrow Y_k$ is defined as the percentage of transactions that Y_k is seen when X_k appears in a transaction. More details can be found in [119].

Figure 4.3: An illustration of expressing association rules in a lattice. $Q_x \Rightarrow Q_{s11}$ is a rule, whose characteristics are described through *confidence* and *specificity*[43]. |x| depicts the number of elements in the set x.

However, the *Confidence* parameter says little about the refinement *Q*' itself, e.g. how this refinement is specific for the initial query *Q*. In order to measure this property we introduce the parameter *Specificity* as the ratio between the number of results for a query that contains only the new constraints and the original query, as given in the following definition:

Definition 4.6: *Specificity*

$$Specificity(Q', Q) = F(Q' - Q)/F(Q'),$$

where "-" represents the difference between the set of constraints given in the query *Q* and its refinement *Q*'.

See Figure 4.3 for an illustration.

In other words, it shows the extent of strength of the new constraints introduced by *Q'*, related to the original query *Q*. The specificity corresponds in some sense to the parameter deviation that can be found in the frequent pattern discovery [40].

Note that $0 < Confidence, Specificity < 1$, whereas larger values indicate larger suitability of a constraint to be used for refining a given query. In that way, it is ensured that the refined query will retrieve "enough" resources to the user.

A method how to calculate these parameters efficiently is given in section 4.3.2.2.

4.3.2 Phase 2: Refinement Generation

If we consider query *Q* as a logical formula, then the results of query, $F(Q)$ can be treated as an interpretation *I* which is a model for formula *Q*. Further, condition $F(Q') \subseteq F(Q)$ from Definition 4.1 can be read as "the interpretations (models) that make *Q'* true are subsets of the interpretations that make *Q* true", which is in accordance with the model-theoretic interpretation, a definition of the logic implication, i.e. query *Q*' implies another query *Q* (*Q* logically entails *Q*'). Since the ontology and the knowledge base are the only constraints used for driving the refinement process, then

$$(Q \rightarrow_{ext} Q') \equiv (KB, O \vdash Q' \rightarrow Q), \qquad (3)$$

where $\vdash$ depicts the derivation (inference) process.

Therefore, the process of query refinement can be mapped onto subsumption reasoning, i.e. for a query *Q* we can find/calculate all queries which are logically implied by *Q* – this set is a set of valid query refinements.

We are, in short, giving another interpretation of this implication process:

As mentioned in Chapter 2, the general form of the logic-based IR is $K \vdash r \rightarrow q$. It means that domain theory *K* is used for driving this logic implication. However, users tend to refine their queries, so that a retrieval process additionally hosts the process of transforming the initial query *q* in *q*', i.e. $q' \rightarrow q$. By using the transitivity of classical logical implication we get from the above implications:

[43] The presented calculation assumes that entire relevant information is presented in Figure 4.3.

$$K \vdash (r \rightarrow q' \wedge q' \rightarrow q) \mid\!\!- r \rightarrow q.$$

It means: if there is a new query q' so that the new query implies the original query and that the new query is implied (satisfied) by a resource, then we can say that the original query is also satisfied by the resource. This statement shows that the process of transforming the original query should be driven by domain theory K as well, i.e.

$$K \vdash q' \rightarrow q.$$

As q' may be any query expression, we can re-write the above implication like:

$$\vee_{q'}(r \rightarrow q' \wedge q' \rightarrow q) \mid\!\!- r \rightarrow q.$$

In other words, the task of query refinement is to find all transformations of the original query which can lead to a resource relevant for the query given by the user. Consequently, query refinement *can be considered as an ordinary IR task* in a repository of queries, i.e. it is the task of finding all relevant refinements for a given query. It means that the quality of a query refinement can be measured in the traditional IR manner: by using precision and recall, i.e. the refinement process should return *all and only relevant* queries[44], as we have elaborated in previous section.

However, due to the undecidability of subsumption reasoning in the general case [76], we need an alternative subsumption order, which introduces more tractability by the minimal loss in the quality of the subsumption. We choose the θ-subsumption [93], a frequently used subsumption order in inductive logic programming tasks.

Definition 4.7: Given two queries Q_a and Q_b we say Q_a θ-*subsumes* Q_b if and only if there exists a substitution θ so that all the atoms in $Q_b\theta$ occur in Q_a .

Example: the query "`forall x <- FamilyCar(x) and hasFeature(x,Metallic)`" θ-subsumes the query "`forall x, y <- FamilyCar(x) and hasFeature(x,y)`". The valid substitution is {`y /Metallic`}.

The θ-subsumption condition is stronger than - though not equivalent to – the logical implication; θ-subsumption is, in turn, weaker than the subset relation. In theory testing θ-subsumption is NP complete, but in some practical cases as discussed in [38], θ-subsumption can be tested efficiently.

The main reasons for this choice are the strength of θ-subsumption (i.e. good approximation) and the existence of well-developed *clause induction* frameworks for a definite database[45] [40], [41], which we briefly interpret in the context of the query refinement in the next subsection.

4.3.2.1 Query Refinement as a Step in the Clause Induction Process

4.3.2.1.1 Terminology

A first order alphabet is a set of predicate symbols, constant symbols and functor symbols. A clause is a formula of the form $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ where the A_i and B_i are logical atoms. An atom $p(t_1, \ldots, t_n)$ is a predicate symbol p followed by a bracketed n-tuple of terms t_i. A term t is a variable V or a function symbol $f(t_1, \ldots, t_k)$ immediately followed by a bracketed n-tuple of terms t_i or constants which are function symbols of arity 0. Functor-free clauses are clauses that contain only variables as terms. All variables in clauses are universally classified, although this is not explicitly written. Extending the usual convention for definite clauses

[44] Interpreting in the logic-based manner, the logic implication $K \vdash q' \rightarrow q$ has to be sound and complete.
[45] A Datalog database is a definite database in which terms are either variables or constants, but not functions.

(where $m = 1$), we call $A_1, \ldots, A_m$ the *head* of the clause and $B_1, \ldots, B_n$ the *body* of the clause. A fact is a definite clause with an empty body ($m = 1$, $n = 0$).

A Herbrand interpretation over a first order alphabet is a set of ground facts constructed with the predicate, constant and functor symbols in the alphabet. A Herbrandt interpretation I is a model for a clause c if and only if for all grounding substitutions θ of c: $body(c)\theta \subset I \rightarrow head(c)\theta \cap I \neq \varnothing$. It means that c is true in I or c makes the interpretation I true. A Herbrand Interpretation I is a model for a clausal theory T if and only if it is a model for all clauses in T. Roughly speaking, the truth of a clause c in an interpretation I can be determined by running the query ?- $body(c)$, not $head(c)$ on a dataset containing I using a theorem prover. If the query succeeds, the clause is false in I. If it finitely fails, the clause is true. This method is designed for range restricted clauses. A clause c is range-restricted if and only if the set of variables that appear in the head of the clause are subsumed by the set of variables that appear in the body of the clause.

The least Herbrand interpretation of a definite clause theory is the set of all ground facts (using the predicates, functors and constants of the definite clause theory) that are logically entailed by the definite clause theory. $M(T)$ will be the notion for the least Herbrand interpretation of a clause theory T.

4.3.2.1.2 Logical Framework for Induction: Closed Observation

There are several formalisms of induction in the clausal logic and we base our work on the characterizing induction from closed observations. The basic ideas are:

1) all observations (data, examples) are completely specified and,
2) the hypothesis (refined clauses) should reflect what is in the data.

The first principle is implemented by representing the observations as Herbrand interpretations, with the consequence that all examples are assumed to be completely specified (like the closed world assumption). The second principle is enforced by requiring all hypotheses to be true in all of the observations. The term characterizing induction comes from the fact that in the framework there is only one type of observations and the aim is to characterize the observations.

Characterizing induction from closed observation can be defined as follows:

Definition 4.8: Let O be a set of observations, B a background theory, L a set of clauses. $H \subset L$ is a solution if and only if H is a logically maximally general valid hypothesis. A hypothesis H is valid if and only if for all $o_i \in O$, H is true in all minimal Herbrand models of $B \cup o_i$.

Since the search space of the clausal logic is large (and even infinite), the definition of some syntactic restrictions on the space of hypothesis is of crucial importance in the clausal induction.

Definition 4.9: (*Language assumption*): The alphabet of the hypothesis language will only contain constants or predicate symbols that occur in one of the observations or in the background theory.

In order to illustrate the previous definitions, let us analyse an example:

Imagine we are observing different collections of cars and the background theory B consists of:

$$Vehicle(x) \leftarrow Car(x)$$
$$Vehicle(x) \leftarrow Motorbike(x)$$

and we observe two different collections:

$$o_1 = \{Car(V1), Motorbike(V5)\}$$
$$o_2 = \{Car(V2), Motorbike(V6), Motorbike(V7)\}$$

If L is restricted to range-restricted and constant-free clauses a solution is:
(1) $Vehicle(x) \leftarrow Car(x)$
(2) $Vehicle\ (x) \leftarrow Motorbike(x)$
(3) $Car(x) \vee Motorbike(x) \leftarrow\ Vehicle(x)$
(4) $\leftarrow Car(x) \vee Motorbike(x)$
This is a solution because all clauses (1-4) are true in both minimal Herbrand models $M(B \cup o_1)$ and $M(B \cup o_2)$.

The framework derives hypotheses that are valid in the model. Intuitively, validity means that the hypothesis holds on the data, i.e. that the induced hypothesis postulates true regularities present in the model.

4.3.2.1.3 Comparison to Other Induction Frameworks

The main issue for comparison to other induction frameworks is that this framework is specifically tailored towards the discovery of regularities that hold in a set of (unclassified) observations or that characterize the closed observations. Within the inductive logic programming (ILP) and other forms of machine learning, people have classically focused on learning rules that discriminate positive observations from negative ones. Within ILP that is captured in the following definition of explanatory induction [93].

Definition 4.10: (*Explanatory induction*) Let P be a set of true observations, N be a set of false observations, B a background theory. $H \subset L$ is an explanatory solution if and only if H is complete with regard to the positive observations and consistent with regard to the negative observations. A hypothesis H is complete with regard to P and B if and only if $B \cup H \models P$; H is consistent with regard to N and B if and only if $B \cup H \cup N \not\models \emptyset$.

A difference between the frameworks is that the closed observation induction assumes complete information about the examples and does not allow inductive leaps on the models, i.e. applying the induced hypotheses on the observations will not result in postulating new facts. The closed observation induction makes inductive leaps of a different kind, in the sense that it postulates that the induced hypotheses will be valid on unseen observations.

Another difference is that the aim in characterizing induction is to find a most informative hypothesis, i.e. the hypothesis that covers the minimal number of examples. When working with closed observations "most informative" means "logically maximally general". The reason is that the logically more general hypotheses have the least number of models; hence they cover the minimal number of observations. In this case a hypothesis covers an example if the example is valid in the hypothesis. In contrast, when working with explanatory induction, "most informative" means "logically maximally specific", as these hypotheses cover the least observations. In this case a hypothesis covers an example if the hypothesis entails the example.

4.3.2.1.4 Query Refinement Induction

Since the task of this framework for clause induction is to learn hypotheses (clauses) which explain a set of observations (data) for the given background theory, we have to adapt its results to our problem setting expressed in (3): for a given ontology (O), its knowledge base (KB) and a query (Q) derive a set of subsumed queries. The mapping of elements is straightforward: The set of observations corresponds to our knowledge base; the background theory represents an ontology, and a hypothesis or a clause is actually treated as a query. Obviously, our *query refinement induction* problem is a simplification of the clause induction framework since:

(1) a query can be treated as a clause with an empty head, and,

(2) we are interested in only one induction step (due to a step-by-step refinement).
Therefore, our task can be treated as a step in the incremental clause induction. Consequently, we can reuse the principle of "incremental learning" in the clause induction [84]: If clause *C* (θ-) subsumes clause *D*, there exists a refinement chain from *C* to *D* that uses only elementary substitutions and adding literals. In other words, we have two elementary transformations (substitutions and adding literals) which can be applied to query *Q* in order to derive refinements *Q'*.

Therefore, true to the "Generalization as Search" tradition in machine learning [81], we can cast the query implication in logic as a search process through a quasi-ordered query space defined by a generalization relation.

However, several issues have to be resolved in order to make this search process reliable, e.g. how to find a minimal but complete set of substitutions and literals which can be applied/added in a refinement step.

First, even if we choose search space *L* to be finite, it is in most cases impractical to define *L* extensionally. In order to complete the induction process we need a so-called *declarative language bias* formalism to formulate an intensional syntactic definition of *L*.

Second, a naive strategy for search can be to inspect all the elements from space *L* one by one (ad infinitum). If the patterns are totally independent as far as their quality is concerned, there is no alternative to such an exhaustive search. If however, the patterns are dependent w.r.t. a quality criterion, a more intelligent strategy is applicable: on the basis of the patterns seen so far, the doomed patterns can be removed or "pruned" from the queue. We then have to define an ordering on the patterns keeping in mind that this order will determine the amount of pruning.

Third, two steps are required to make this idea practical: (i) we do not want to manually sort the queue, so we need some computable principle to *structure the space*; (ii) we do not want to generate first the whole search space and then to sort it, especially since we are interested in the neighborhood of a query only. Therefore we need a technique to *explore the space* efficiently, i.e. to compute the successors of a given query and generate the search space on the fly.

In the following subsections we discuss these issues regarding the query refinement task.

4.3.2.2 Defining the Search Space: Declarative Language Bias

The notion bias, generally circumscribed as a tendency to show prejudice against one group and favoritism toward another, has been adapted to the field of computational inductive reasoning to become a generic term for "any basis for choosing one generalization over another, other than strict consistency with instances" [80].

With the traditional association rule induction [119], the definition of a language bias is straightforward: *L* is simply 2^I, where *I* is the set of items. Given a set *Atoms* of ground atoms, the language *L* consists of 2^{Atoms}, i.e., of all possible combinations of the atoms. In ILP on the other hand this issue has been studied extensively, which is motivated by the huge, often infinite search space that requires a tight specification of interesting patterns. When variables are allowed in *L*, the power set idea can be extended to a set of literals [158]. However, this solution is shown to be inconvenient for many reasons [38].

Several formalisms have been proposed for adding language bias information in a declarative manner to the search process. These can be classified into two main families: those based on templates and those based on type and mode declarations [38]. The latter one is more relevant for our approach. In general, this formalism constrains the search space through the

constraints posed on candidates for the refinement. We mention here the constraints from the best known approach for clause induction WARMR [38]:

- mode

Non-ground atoms in set *Atoms* are allowed to occur multiple times in the query, as long as their variables obey the so-called mode constraints. These are declared for each variable argument of each atom by means of three mode-labels +, - and ± , where:

- + the variable is strictly input, i.e. bound before the atom is called,
- - the variable is strictly output, i.e. bound by the atom, and,
- ± the variable can be both input and/or output, i.e. anything goes.

Finally, a query is then mode-conforming if an ordering of atoms exists so that every input variable occurs in one of the previous atoms, and no output variable does.

- typing

Additional constraints on the sharing of variable names can be imposed via type declarations. The convention is to append these to the mode declarations in *Atoms*. A query is then type-conforming if and only if the arguments that share a variable name have identical types or at least one of them is untyped.

Finally, the clause induction task requires the specification of a **key** atom which is obligatory in all queries. This notation can be done with *key = KeyAtom*, where *KeyAtom* is a mode and type declaration as defined above. Obviously, the key atom declaration should not contain any + mode.

Ontology-based Declarative Language Bias - ontoDLB

Since we are using an ontology as the background theory, the *mode* and *typing* constraints could be read from the ontology and the knowledge base directly. Indeed, for each relation *rel*(*a*, *b*) in an ontology, the domain and range are formally specified, so that the typing of arguments is driven by this information. It means that for a query `"forall x, y <- Car(x) and CarType(y) and hasType(x,y)"` extension `hasFeature(y,Metallic)` cannot be valid since relation `hasFeature(a,b)` has as the domain concept `Car` that is represented through variable `x` in the query.

Since we are using this background information, the classification of variables through the mode constraint seems to be obsolete: for the above mentioned query, in extension `hasFeature(x,Metallic)` variable `x` is the output variable, but in `hasFeature(x,z)and hasLuxuryQuality(z,"High")` variable `z` is an input variable.

The *key* constraint corresponds to the definition of the *Key* we have given in section 4.2.

Additionally to the possibility to define declarative language bias automatically, we are introducing some new parameters for constraining the search space as follows.

Proposition 4.1: The contraposition of the interpretation of the logical implication given at the beginning of this section, states that, if a model *M* is not a model for *Q* then *M* will not be a model for any logical refinement *Q'* of *Q*. In the context of *our Language assumption*, it shows that large parts of the search space can be pruned, i.e. that only literals (i.e. substitutions) that can be considered as relevant for the original query can be proposed as candidates for the expansion.

Therefore, it is sufficient to analyse the answers of query *Q* in order to find all the constraints that are relevant for the refinement. Let us assume that $X = F(Q)$. Then for each variable from *Q*, we get a set of instances related to variable *x* and retrieved for query *Q* as:

$$RelevantSet(x,Q) = \bigcup_{a \in F(Q)} ExtractVar(x,a),$$

where *ExtractVar* retrieves the instance that corresponds to variable *x* from an answer tuple *a*.

For example, for query *Q* = `"forall x <- Car(x) and hasType(x, FamilyCar)"` against the knowledge base represented in Table 4.2, we get:

$$RelevantSet(x, Q) = \{\texttt{c5}, \texttt{c6}, \texttt{c7}, \texttt{c8}, \texttt{c9}, \texttt{c10}, \texttt{c11}, \texttt{c12}, \texttt{c13}\}. \quad (4)$$

From each of these relevant elements a new pattern set is generated by replacing each element with all its relation instances from the knowledge base (see Definition 2.6):

$$PatternSet(x,Q) = \{ r(a,b) \mid a \in RelevantSet(x,Q), r(a,b) = true \}.$$

Regarding (4):

PatternSet(*x*, *Q*) = {
`hasFeature(c5,GPS), hasFeature(c6,Automatic),` (5)
`hasFeature(c7,Automatic),hasFeature(c8,Automatic),`
`hasFeature(c9, Automatic),hasFeature(c10, Metallic),`
`hasFeature(c11,Metallic),hasFeature(c12,Metallic),`
`hasFeature(c13,Metallic),hasType(c5,FamilyCar),`
`…` `,hasType(c13,FamilyCar)` }

Finally, this set is analysed in order to find a frequent pattern that can be used for the refinement.

We find two types of patterns:

- *Substitutions*(*x*, *Q*) by counting only those relation instances from *PatternSet* whose predicate matches with a constraint from the query. Regarding (5), such a relation instance is `hasType(c5,FamilyCar)` and the corresponding pattern is `hasType(x,FamilyCar)`.
- *Literals*(*x*, *Q*) by treating the rest of the *PatternSet*.

For each pattern *pat* from these sets we define its *Confidence* and *Specificity*, as characteristics of the content ambiguity, in the following manner:

$$Confidence(pat,Q) = \frac{\left|\{ l \mid l \in PatternSet(x,Q) \wedge Unify(l, pat) = true \}\right|}{\left|RelevantSet(x,Q)\right|}, \quad (6)$$

where *Unify*(*a*, *b*) returns true if there is a substitution in KB that unifies predicates *a* and *b*. For example, *Unify*(`hasFeature(c5,GPS), hasFeature(x,GPS)`) = true, since there is a substitution {`x/c5`}.

$$Specificity(pat,Q) = \frac{\left|F(Query(pat))\right|}{\{ l \mid l \in PatternSet(x,Q) \wedge Unify(l, pat) = true \}},$$

where *Query*(*pat*) is an elementary query that contains only the constraint *pat*.

Regarding (5),

Literals(*x*, *Q*) = {`hasFeature(x,GPS), hasFeature(x,Automatic), hasFeature(x, Metallic)`}.

Moreover, for example, considering Figure 4.2:

- *Confidence*(`hasFeature(x,GPS)`, *Q*) = 1/9 (there are 9 `FamilyCars` and only one has `GPS`) and

- *Specificity*(`hasFeature(x,Automatic)`, Q) = 4/5 (4 of 5 cars that have luxury feature `Automatic` are `FamilyCars`).

Note that these two parameters corresponding to the two parameters we have mentioned in example 3 in the motivating example are very important for refinements' ranking.

In this way we get a set of substitutions and predicates which are guaranteed to be relevant for the query refinement process. Moreover, based on the above given contraposition to the logical implication, we conclude that this set is a complete one.

Therefore, our formalism for defining language bias generates two sets of candidates, *Substitutions* and *Literals*, from the initial query and its set of results. Due to its ontology-based structure, it is called *ontoDLB*.

4.3.2.3 Structuring the Search Space

As we have already stated: The θ-subsumption condition is stronger than, though not equivalent to, the logical implication; θ-subsumption is, in turn, weaker than the subset relation.

Proposition 4.2: For all queries Q_1 and Q_2:

$$Q_1 \supseteq Q_2 \Rightarrow Q_1 \text{ θ-subsumes } Q_2$$
$$Q_1 \text{ θ-subsumes } Q_2 \Rightarrow Q_2 \models Q_1$$

For a proof see [93].

Definition 4.11: Let Q and R be two queries, then

$$Q \rightarrow_{ext} R \Leftrightarrow R \text{ θ-subsumes } Q.$$

We say "query Q is more general than R" and "query R is more specific than Q", or "query R is an extension of the query Q".

The following property follows immediately from the previous definition and from the reflexivity and transitivity of θ-subsumption:

> The $(\Omega(O), \rightarrow_{ext})$ is a quasi-ordered set, where $\Omega(O)$ represents a set of queries in the ontology O.

Proposition 4.3: The generality order $\rightarrow_{ext}$ is monotonous with respect to query results. Formally, given an ontology O and two queries Q_1 and Q_2 :

$$Q_1 \rightarrow_{ext} Q_2 \quad \Rightarrow F(Q_2) \subseteq F(Q_1)$$

Proof :

According to Definition 4.11:

$$Q_1 \rightarrow_{ext} Q_2 \quad \Rightarrow_{def} Q_2 \text{ θ-subsumes } Q_1$$

According to the Proposition 4.2:

$$\Rightarrow_{def} Q_1 \models Q_2$$

According to the definition of the implication

$$\Rightarrow_{def} \text{if } Q_2 \text{ matches } r \text{ then } Q_1 \text{ matches } r$$
$$\Rightarrow |\{r \mid Q_1 \text{ matches } r\}| \geq |\{r \mid Q_2 \text{ matches } r\}|$$
$$\Rightarrow F(Q_2) \subseteq F(Q_1) \square$$

Therefore, the generality under the θ-subsumption relation induces a lattice structure on a set of queries, that is needed for building a query neighborhood as we have already explained in

section 3.2.2.2.2. By now we can apply the classical machine learning principles to obtain an algorithm for characterizing induction from closed observations. First, the machine learning principles state that induction is a search process through a partially ordered space induced by the generalisation relation. Second, given the generality order under θ-subsumption, two main strategies are available for exploring the lattice. One can either go general-to-specific, i.e. from the top towards the bottom of the lattice or vice versa. Theoretically, there may, however, be a problem when search specific-to-general as one should start from the most specific hypothesis which could be an infinite one. Therefore, we will only consider general-to-specific search.

4.3.2.4 Exploring the Search Space

A refinement operator ρ (with transitive closure ρ*) for language L is a mapping L to L^2 so that:

1. $\forall Q \in L$: $\rho(Q) \subseteq \{Q' \in L \mid Q' \neq Q$ is a maximally general specialization of Q under θ-subsumption$\}$ and
2. ρ is complete, i.e., $\rho(T) = L$, where T is the most general element in L.

Completeness means that all the elements of the language can be generated using ρ.

A refinement operator ρ is *optimal* if and only if

$$\forall c, c_1, c_2 \in L : c \in \rho^*(c_1) \text{ and } c \in \rho^*(c_2) \rightarrow c_1 \in \rho^*(c_2) \text{ or } c_2 \in \rho^*(c_1).$$

Optimal refinement operators are more efficient than classical refinement operators because they generate each candidate clause exactly once. A known problem with classical refinement operators is that they generate candidate clauses (and their refinements) more than once, making the search intractable. Optimality is thus desirable for efficiency reasons.

Refinement operator for ontoDLB

Since the *ontoDLB* notation belongs to type and mode declaration of the declarative language bias it is less restrictive. This is reflected in the complexity of the refinement operator, which is essentially based on generating new queries by substituting a variable or adding a literal and then testing whether these new queries are valid.

Definition 4.12: *Refinement operator* (ρ_{closed}):

Assume the *ontoDLB*'s output for the results of query Q is {*Substitutions*, *Literals*} as described in section 4.3.2.2. The closed-refinement of query Q, denoted $\rho_{closed}(Q$, *Substitutions*, *Literals*) is a query obtained in one of the following ways:

(a) $\rho_{closed}(Q$, *Substitutions*, *Literals*$) = Q\theta$, where θ is a substitution from the set *Substitutions* in the case that θ contains a substitution pair related to variable x from Q;

(b) $\rho_{closed}(Q$, *Substitutions*, *Literals*$) = (Q \wedge L)$, where L is a ground literal contained in the set *Literals* for any variable x from query Q;

Therefore, our starting problem ($Q \rightarrow_{ext} Q'$) can be reformulated as calculating $\rho_{closed}(Q, a, b)$, where a and b represent two sets generated from *ontoDLB*. For refinement Q' we can write $Q \geq_{closed} Q'$. Furthermore, the ρ_{closed} refinement operator is a weakly complete operator [4]. The incompleteness is the consequence of *uncovered infinite ascending chains* $Q \geq_{closed} ... \geq_{closed} E_{i+1} \geq_{closed} E_i \geq_{closed} ... E1$ (for which there exists no maximal element $E \geq_{closed} E_i$ for all i, so that $Q \geq_{closed} E_i$) [4].

An important corollary of the less restrictive nature of the *ontoDLB* formalism is its non-optimality. The order in which literals are drawn from the type and mode specifications in Atoms is not fixed. As a consequence, if a pattern is generated, many of its logically

equivalent permutations are likely to be generated as well. These variations can be filtered in the algorithm in which the refinement operator is embedded.

Finally, we show that the query refinement process driven by the ρ_{closed} refinement operator satisfies properties we have specified in section 4.2:

1) Completeness w.r.t. relevant results:
   It can be easily shown using weakly completeness of ρ_{closed}-refinement operator.

2) Soundness w.r.t. relevant results:
   Let us assume that there is a refinement Q_x which contains a result a that cannot be retrieved for the query Q (a is then an irrelevant result). i.e.
   $$\exists Q_x, a\ (Q \geq_{closed} Q_1) \wedge a \in F(Q_x) \wedge a \notin F(Q) \quad (7)$$
   However, from Definition 4.4 it follows $Q \geq_{closed} Q_x \Rightarrow F(Q) \supseteq F(Q_x)$,
   i.e. $\neg\exists a\ a \in F(Q_x) \wedge a \notin F(Q)$. This statement implies a contradiction in (7).

3) Soundness w.r.t. relevant refinements:
   It is obvious from Definition 4.12 and the discussion above it.

4) Maximally general relevant refinements:
   It is obvious from Definition 4.12 ρ_{closed} -refinement contains only so-called downward covers[46] [4].

5) Ranking:
   The parameters *Confidence* and *Specificity* are used for the ranking: a higher value for *Confidence/Specificity* means a higher rank of a refinement. More details are given in section 4.3.3.

6) Query Neighbourhood:
   We have already shown (section 4.3.2.3) that the relation $\rightarrow_{ext}$ can be used for generating a lattice of refinements.

4.3.2.5 Query Refinement Induction Algorithm

In general, induction approaches follow two-phased search that can be found in the levelwise algorithm [38]. The levelwise algorithm is based on a breadth-first search in the lattice spanned by a specialization relation between patterns. The method looks at a level of the lattice at a time, starting from the most general patterns. The method iterates between *candidate generation* and *candidate evaluation* phases: in the candidate generation, the lattice structure is used for pruning non-frequent patterns from the next level; in the candidate evaluation phase, frequencies of candidates are computed with respect to the database. Pruning is based on the monotonicity of the specialization relation with respect to frequency: if a pattern is not frequent then none of its specializations are frequent. So while generating candidates for the next level, all patterns that are specializations of infrequent pattern can be pruned. For instance, in the APRIORI [119] algorithm for frequent items, candidates are generated so that all their subsets (i.e. generalizations) are frequent.

1) Assuming all the candidates of a level are tested in a single database pass, the database is scanned at most d +1 times, where d is the maximum level (size) of a frequent pattern. This is an important factor when mining large databases.
2) The time complexity is in practice linear in the product of the size of the result times the number of examples, assuming matching patterns against the data is fast.

As we have already elaborated, our approach follows only one induction step that is represented below:

[46] D is downward cover of C iff $C \geq_{closed} D$ and no E satisfies $C \geq_{closed} E \geq_{closed} D$.

Function Query Refinement Induction

Inputs: Knowledge base K; *ontoDLB* language L, query Q, refinement operator ρ_{closed}
Outputs: *Te*, list of all equivalent queries and *Ts*, list of all subsumed queries of the query Q

1. Initialize *Te*:=∅
2. Initialize *Ts*:=∅
3. Derive sets *Substitutions* and *Literals* as outputs of *ontoDLB*(K, Q, $F(Q)$)
4. Apply operator ρ_{closed} to find candidates for extension
   For each query $Q' \in \rho_{closed}(Q, Substitutions, Literals)$
   Add Q' to *Ts* unless:
   (a) $\exists A \in Ts$: Q' θ-subsumes A. It means that A is a non-maximal refinement
   (b) $\exists B \in Te$: Q' θ-subsumes B and B θ-subsumes Q'. It means that Q' is an equivalent refinement
   In the case (b): Add Q' to *Te*
5. Return *Ts* and *Te*

Therefore, we are using the refinement operator ρ_{closed} to compute extensions (Step 4). However, since the search space induced by the operator is a lattice and we are interested in the first level only (due to the request for step-by-step refinement and Proposition 4.3), the refinements that belong to the other layers should be filtered. The current list of subsumed queries *Ts* is scanned for the query that θ-subsumes the candidate query Q' (Step 4a). If such a query is found, Q' is discarded.

Finally, the current list of equivalent queries *Te* is scanned for a query that is logically equivalent under θ-subsumption to candidate query Q' (Step 4b). If such a query is found, Q' is again discarded from the list of subsumed queries *Ts*. However, it is added to the list of equivalent queries *Te*. This is where the algorithm compensates for the non-optimality of the refinement operator ρ_{closed}.

Moreover, the implementation could be extended with many extra features that provide more expressive power and allow more condensed descriptions. One general mechanism is to make the presence of literals conditional on the presence or absence of other literals. For example: the query `"forall x <- Car(x)"` can be extended with the literal `hasFeature(x, y)` only in combination with an atom that provides more information on the luxury, such as `hasLuxuryQuality(y, "High").`

4.3.3 Phase 3: Ranking of Refinements

As we have already mentioned in section 3.2.2.3 there are two types of relevance: *Semantic* and *Personal*.

For the *Personal* relevance we presented in section 3.2.2.3 a general model that can be directly applied for the interaction in this case, i.e. (see Definition 3.17)

$$PersonalRelevance(ref, Q, Q_s) = \Sigma_{\forall term \in Refinement(ref)} a*Importance(term, Q, Q_s) + b*FeedbackRelevance(ref, Q_s)$$

where:
a and b are weighting parameters (by default: $a = b = 1$).

Regarding *Semantic* relevance we use an instantiation of the general formula given in section 3.2.2.3.1. We combine both relevancies we mentioned in section 4.3.1 as follows (see Definition 3.7):

SemanticRelevance(*ref*, *Q*) =

a * (*Confidence*(*Q'*, *Q*)**Specificity*(*Q'*, *Q*)* *SemanticAmbiguity*(*Q'*)) +

b * *Relatedness*(*ref*, *Q*)

where:
Q' is an extension of the query *Q* with the refinement *ref*,
Relatedness(*ref*, *Q*) = $\Sigma_{\forall term \in Q}$ Rel^{max}(*ref*, *term*) (see Definition 3.10) , and,
a and *b* are weighting parameters (by default: *a* = *b* =1).

In other words, the relevance of a refinement *ref* is greater if this refinement is more likely to be performed (*Confidence*), if it is more tailored for the original query (*Specificity*) and if the variables that are further constrained in that refinement are very ambiguous (*SemanticAmbiguity*). Moreover, the refinement should be strongly related to the initial query (*Relatedness*). This formula ensures that the user will be provided with the refinements that are very related to the initial query, that decrease the ambiguity of the query and which are consequently more suitable for user's information need.

Relatedness. The only parameter that should be additionally defined is the matrix Rel^{max}, i.e. the relatedness matrix *Rel*. This matrix defines the closeness between query constraints as follows:

$$\forall a,b \in \Omega(O)\ domain(a) = domain(b) \vee range(a) = range(b) \rightarrow relatedness(a, b),$$

where:

a and *b* are elementary constraints (*predicates* in the form *rel*(*par1*, *par2*)),

domain(*a*) is the functions that retrieves the domain of the constraint a, e.g. *domain*(*rel*(*par1*, *par2*)) = *par1*, according to the Definition 2.3 and

range(*a*) is the functions that retrieves the range of the constraint a, e.g. *range*(*rel*(*par1*, *par2*)) = *par2*.

Finally, the total relevance of a refinement *ref* for the query *Q* in the query session *Qs* is calculated as (see Definition 3.18):

$$(SemanticRelevance(ref, Q) + \lambda * PersonalRelevance\ (ref, Q, Q_s))/(\lambda+1)$$

where λ is a forgetfulness coefficient that models the impact of the past user behavior on the ranking process: $\lambda = 0$ - the past is forgotten, $\lambda < 1$ - the past carries less weight then the present, usually $\lambda = 1/2$.

4.3.4 Comprehensiveness of the Approach

Our learning setting can be seen as learning from multiple relations (more widely – interpretations) [13], whereas an interpretation corresponds to a *PatternSet*, see (5). However, our goal is not to learn the entire theory, but rather just one or several refinement(s) of a query. In other words, we are interested in a (complete) set of single changes that can be applied to a query. Thus we can overcome the problem implying that, while learning from interpretations, only information about the example that is to be classified is used. Indeed, in a refinement step, we calculate independently the refinements regarding each variable (i.e. its *PatternSet*) and the user selects one of them as the refinement. In the next refinement step, the user is again provided with the list of refinements regarding each variable, so that he can select another variable. In this way, we allow "learning" of refinements like "a SportsCar,

which has a GPS system that is manufactured by SONY" (of course in a step-by-step manner). Note that very complicated patterns like "a SportsCar that has a GPS system and a CD-player manufactured by the same vendor" can be learned as well. Finally, mapping to the (*w*)*rmode* refinement operator *rmode*(*n*: (A_1, ...,A_n) [39] is straightforward: n =1 and A_is are atoms taken from *PatternSet*. Note that our approach differs from the learning of frequent query expansions presented in [40], by learning step-by-step query refinements, which means that we add only one change in a query in a refinement step.

One of the advantages of our approach is the possibility to define various views on the knowledge base and generate different views on refinements. Regarding the example 2) from the motivating example, by selecting one of two possibilities to generate refinement candidates (substitution or add literal), our approach enables two types of refinements. We discuss that case briefly:

Query: `"forall x,y <- Car(x) and hasFeature(x, y)"`
List of Refinements 1 (using substitution):

```
1. "forall x,y <- Car(x) and hasFeature(x, Metallic)"
2. "forall x,y <- Car(x) and hasFeature(x, Automatic)"
3. "forall x,y <- Car(x) and hasFeature(x, GPS)"
4. "forall x,y <- Car(x) and hasFeature(x, Cabriolet)"
```

List of Refinements 2 (by adding literal):

```
1. "forall x,y <-
      Car(x) and hasFeature(x,y) and hasLuxuryQuality(y,"High")"
2. "forall x,y <-
      Car(x) and hasFeature(x,y) and hasLuxuryQuality(y,"Low")"
```

Figure 4.4 illustrates this case.
Therefore, the user can control the granularity that will be used in generating the next refinement steps.

From the conceptual point of view the presented approach is quite general: it casts the query expansion problem to a machine learning problem of search for frequent patterns. By defining the corresponding refinement operator (and all required structures, like declarative language bias) the approach can be applied to other ontology languages, for example languages based on DL. Indeed, in the case of OWL the prenex conjunctive normal forms (PCNF) with existential variables should be considered [85].

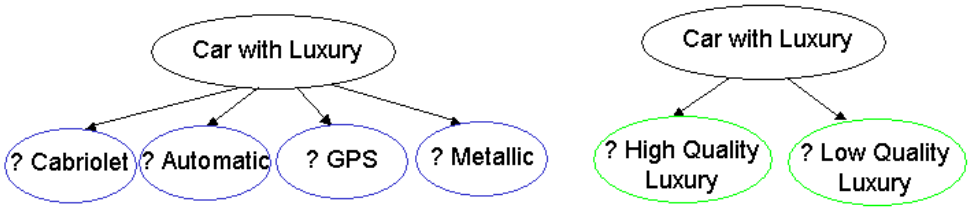


Figure 4.4: An illustration of the different refinement views of a query

4.3.5 Complexity Issue

Although the discussion about the complexity of calculating θ-subsumptions can be long [40], we are mentioning here only the complexity issue related to our setting. Moreover, as we have already mentioned, we have made a lot of simplifications in a standard clause induction setting (e.g. by considering queries only). Finally, we are using step-by-step refinement which means that we generate just one step in the induction process.

Let us assume that a knowledge base is characterized through:

- ***c*** – number of concepts,

- ***l*** – average number of relations of a concept, and,
- ***g*** –average number of the instantiations of a relation (multi-value relations).

If we consider query ***q*** with ***n*** results and ***k*** variables then ***k*n*l*g*** is the number of elements in the whole *PatternSet*, i.e. the number of candidates for the refinement. This is the maximal requirement regarding the space complexity. Since ***k***, ***l***, ***g*** << ***n*** it depends linearly on the number of results: *SpaceComp* = **Ω**(n).

Each of them is compared to a query ***q*** w.r.t. θ-subsumption, which is a low complex operation regarding the time. Therefore, the time complexity of this comparison is **Ω**(***k*n*l*g***).

Due to non-optimality of the operator, each of the candidate refinements will be tested on θ-subsumption with other refinements, which requires 3*(***k*n*l*g***)*(***k*n*l*g-1***)/2 operation. The multiplicator 3 is used since for testing equality the comparison has to be done in both directions, i.e. two times for each candidate for equality + one time for subsumption. Therefore, *TimeComp* = **Ω**(***k*n*l*g***) + **Ω(**3*(***k*n*l*g***)*(***k*n*l*g-1***)/2) = **Ω(**n^2). In other words, the time complexity depends only on the square of the number of results.

4.4 Evaluation

As we have already stated, a very important advantage of modelling a query refinement process in the logic-based manner is the possibility to reason about various features of a model. This is becoming increasingly important because the conventional evaluating methods such as experimentally measuring precision and recall are sometimes insufficient. We have performed two evaluation studies in order to (i) prove the strength of our approach regarding existing methods and (ii) prove the (formal) correctness of the approach itself.

4.4.1 First evaluation study

Since the goal of our research is to model the query refinement support for the ontology-based information retrieval (like on the Semantic Web), our evaluation study concerns the comparison in the effectiveness (regarding refinement) between a traditional refinement method and a system, which uses advanced ontology-based technologies (so called Semantic Portals).

We developed a small car-feature ontology which reflects information that can be found in popular portals for on-line used cars shopping (e.g. www.autocsout24.de), which is partially presented in Table 4.2. It contains about 80 ontology entities. We selected 200 cars from the actual offer from a car shopping portal (www.autocsout24.de) as instances in the repository. The dataset seems to be rather small but the complexity of the resulting lattice structure is rather high. Note that this complexity does not decrease the performance of our approach, since we process that structure in a step-by-step manner. Moreover, the goal of our experiment was to prove only the recommendations for navigation (query refinement) which the system proposes in a navigation step.

We selected eight users with medium knowledge about the cars domain (i.e. all of them have a driver licence). They should navigate through the given repository with the Librarian Agent's support. The task in a navigation session was to select a car that best matches three predefined car-features (e.g. Colour, Luxury). The precondition was to select at most one value for a feature in a navigation step, since we wanted to test the suitability of automatically generated query refinement recommendations. Each user had 10 navigation tasks. In each navigation step we measured the relevance of the system's recommendations for further navigation (query refinement) generated in the Query Neighbourhood. Briefly, in each step a user was provided with six suggestions for query refinement, three of them were the three best

ranked refinements calculated using our approach and three were the best ranked refinements calculated by the traditional query refinement technique presented in [43] (IQE). As known to the author an approach for the refinement of ontology-based queries does not exists. Users were not aware about which particular method they use for refinement.

Users were asked to evaluate the relevance of the first 10 refinements on the following scale: very relevant (4), medium (3), low (2), not at all relevant (1). The results are presented in Table 4.3.

Table 4.3: Results of the first evaluation study, first experiment

Method	Average relevance per a refinement step (max is 4)	Average relevance per best ranked refinement
LibrarianAgent	3.2	3.8
IQE	2.4	2.6

Discussion:

Our approach performs better than a standard query refinement technique. This was expected since our earlier experiments [130] showed that traditional query refinement methods miss the processing of the user's information need, which is the main advantage of our method. However, the best result is achieved regarding best refinements in each step (second column in Table 4.3): almost in each step our approach suggested a very relevant refinement. Regarding users confidence in our method, we can be satisfied: our method always generates useful results for a user. However, improving visual user interfaces will enable a more intuitive user-system dialogue. It should help a user to better understand the recommendations suggested by the system and consequently to have more confidence in the system.

As we already mentioned in the evaluation studies performed in the Chapter 2, our complex notion of relevance is the main factor that enables very reliable ranking of relevant results. In order to define which of the two factors *SemanticRelevance* or *PersonalRelevance* (see section 4.3.3and Definition 3.17) is more important for the refinement, we vary the factor λ (=0, =0.9) for the total relevance in the previous experiment (where $\lambda = 0.5$), such that the impact of the *PersonalRelevance* is controlled. Table 4.4 presents results.

Table 4.4: Results of the first evaluation study, second experiment

Setting	Average relevance per a refinement step (max is 4)	Average relevance per best ranked refinement
λ =0 (without influence of user's behaviour)	2.8	3.4
λ =0.9 (strong influence of user's behaviour)	3.1	3.85

Discussion:

The results show that the usage of the information about a user's behavior is very important for a more personalized retrieval, especially for the relevance of the top-ranked refinements. Interestingly, the relevance of top-ranked results depends very strongly on the user's preferences, such that the results are even better than in the first experiment, as the column 3 for λ =0.9 in Table 4.4 shows.

In order to define how factors that determine *Semantic Relevance* (i.e. *Ambiguity* and *Relatedness*) influence the quality of the refinement process, we have performed one additional experiment (see Table 4.5) in the same setting, by taking $\lambda = 0$ (in order to eliminate the effect of the user's behavior) and varying parameters *a* and *b* in the general formula for calculating *Semantic Relevance* (see section 4.3.3 and Definition 3.7).

Table 4.5: Results of the first evaluation study, third experiment

Setting	Average relevance per a refinement step (max is 4)	Average relevance per best ranked refinement
a =0, b=1 (only Ambiguity)	2.6	3.1
a =1, b=0 (only Relatedness)	2.4	3.0

Discussion:

Since the difference is not statistically significant ($p > 0.001$), both factors have to be taken into account in a query refinement approach. However, the results show that ambiguity is a very important factor in the query refinement, i.e. the effect of decreasing ambiguity of a query seems to be more important for a user than just recommending a refinement that is related to the original query. It illustrates the complexity of the refinement process in which a user not only expects a relevant refinement, but moreover a refinement that can help him to clarify the meaning of the query (i.e. to decrease its ambiguity). Otherwise, he seems to be lost on the information space.

4.4.2 Second evaluation study

In the second study we have compared *formal* properties of the portals, i.e. their query refinement subsystems, according to section 4.2. The experiment should show how these formal properties are fulfilled in both portals. More clearly, we want to check the comprehensiveness on the refinement (regarding properties 1. – 4. section 4.2) provided by the portal to the user in a navigation step. We assume that the user makes a general query and then tries to specify it in several navigation steps which form a navigation session. The experimental setting was the same as in the first experiment. The results are given in Table 4.6.

Table 4.6: Results of the second evaluation study

Method for navigation	Completeness of results in a step (average)	Soundness of results in a step (average)	Completeness of questions in a step (average)	Minimality of questions (average)
IQE	70%	80%	46%	55%
Librarian Agent	100%	100%	100%	100%

Discussion:

In order to simplify calculation (but without effecting the generality/validity of the experiment) we have made a relative measurement, i.e. we have put the parameters of a portal in the context of another. For example, for parameter 3, we have compared the set of questions provided by both the portals. 100% means that this method for that parameter includes all values produced by the other portal. The main finding is that in traditional, syntax-based query refinement approaches a user is missing some very important insights

about the refinement process, like does he get all possible refinements, are some refinements redundant, that can help him much in a better understanding of the repository and in the further development of his information need.

4.5 Related Work

Using lattices for a query refinement process is not new, as some lattice representations were used in early IR [121] and even more recently [122] for refining queries containing Boolean operators. However, as these approaches typically rely on a Boolean lattice formalization of the query, the number of proposed refinements may grow too large even for a very limited number of terms and they may easily become semantically meaningless to the user. These limitations can be overcome by using concept lattices. In [23] the authors described an approach, named REFINER, to combine the Boolean information retrieval and the content-based navigation with concept lattices. For a Boolean query the 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 [11], 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.

Conceptually, the most similar approach to our query refinement system is the Query By Navigation [18], an approach for the navigation through a hyperindex of query terms. The hyperindex search engine [19] directs the users to add, delete or substitute a term from the initial query by providing the minimal query refinements/enlargements. It is designed specifically to (i) help the user express a precise description of his or her information need and (ii) reduce the information overload by presenting the search result at a higher level of abstraction. Moreover, in [54] the analogy between the lithoid, a crystalline structure which organizes document descriptions (and may be used to support searchers in formulating their information demands via query by navigation) and the formal concept lattice is shown and used in the phrase search.

However, all of presented approaches are related to Boolean queries.

In terms of the formal framework, Chaudhuri [28] has proposed an elegant one to describe query modification, and especially query generalization, for the relational model. He defined extended queries which express additional constraints on the answer set. Several query modification operators, mainly based on the structure of a query, are defined in order to model constraints which can be added to a query. However, the goal is not to support the refinement of a user's need, but just the extension of the query. Therefore, a generalization contains only one way of modifying the query. Beside the difference in defining modification operators, we enable a step-by-step modification in which the user himself can define which modification can be relevant for his need. An extension of [28] for the case of XML datasets can be found in [73].

Recently, a framework for the refinement of SQL queries in multimedia databases has been proposed [90]. Query refinement is achieved through relevance feedback where the user judges individual result tuples and the system adapts and restructures the query to better reflect the user's information need. In that way a kind of similarity search is achieved. However, the approach does not treat the refinement process formally, but rather as a set of heuristics (like predicate addition or removal) described as query refinement strategies. Moreover, it does not generate a set of refinements which can support the user in developing ill-defined information needs.

Regarding search in product catalogues the most similar approach is presented in [101]. It is an extension of a mediator architecture that supports the relaxation or tightening of query

constraints when no or too many results are retrieved from the catalogue. The query language is a type of Boolean queries suitable for the (web) form based querying against product catalogues. The query tightening is enabled when the cardinality of the resulted set has reached a predefined threshold and it is realized by selecting the most informative, not yet constrained product features. The information content of a feature is defined by measuring its entropy. Like the previous one, this approach does not treat the problem of query refinement on an ontology-based level.

Finally, our approach can be seen as a method for interactive query refinement for the case of logic-based information retrieval. In that sense, our recommendations can be treated as a combination of subject thesauri and co-occurrence term lists [112].

4.6 Conclusion

In this Chapter we presented an instantiation of the librarian agent query refinement process for the case of the full-fledged ontology-based retrieval, in which the query as well as the information resources are represented using an ontology. We consider query refinement as a θ-subsumption problem and define a logic-based refinement operator for efficient traversing the query refinement space. In that way we get a formal query refinement method that ensures the minimality and completeness of the set of refinements, as well as their suitability for the user information need. One of the main advantages of the approach is the possibility to vary the granularity (generalization level, using ontology) on which the refinements are generated, such that various views on the information resources are possible. In that way, not only the particular refinements of the query, but moreover the particular user's-need specific views on the refinement problem can be realized.

Moreover, the evaluation studies have shown experimentally that a comprehensive model of relevance for query refinement is needed, since in a refinement process a user does not only expect a relevant refinement, but moreover a refinement that can help him to clarify the meaning of the query (i.e. to decrease its ambiguity). Otherwise, he is lost on the information space, as experienced in traditional search approaches.

5 Ontology-supported Attribute-value-based Query Refinement

5.1 Introduction

One of the common pitfalls of the formal logical systems is the problem of their acceptance by the users who are unfamiliar with the underlying formalism, or even more problematic, by the users who are unfamiliar with computer science in general. Indeed, the very formal syntax of an ontology-based query can be seen as a bottleneck in the usage of ontology-based information retrieval systems, although the advantages of these systems, regarding retrieval quality are obvious.

Since the main advantage of using ontology-based queries is their well-defined structure[47] that enables the user to express his intention more precisely, the usage of semi-formal structured queries seems to be a promising alternative for the expressiveness of the queries. For example, the query:

```
forall x, y <-
  Car(x) ∧ hasType(x, SportsCar) ∧ hasFeature(x, y) ∧
  Color(y) ∧ hasColorValue(y, "BlueColor").                    (1)
```

seems to be too complex for a non-experienced user. On the other side, the meaning of the query is very simple: it is a request for a car that is of the type sports car and has blue color. The information that a car has a feature is implicitly assumed.

Therefore, a part of the well-defined query semantics can be simplified, by eliminating the information that can be implicitly assumed. However, the main question is which part of the semantics of an ontology-based query can be abstracted. For example, the following query represents an equivalent query to (1):

```
forall x, y <-
  hasType(x,SportsCar) ∧ hasFeature(x, y) ∧ hasColorValue(y, "BlueColor").
```

Indeed, if we assume that the property `hasType` is defined for cars and the property `hasColorValue` is defined for colors, two explicit statements (`Car`(x) and `Color`(y)) from (1) can be omitted.

In the same way, if we assume that there is one relation defined between cars and colors, the predicate `hasFeature` can be omitted as well. Therefore, what remains are the predicates that convey meaning: `hasType`(`x, SportsCar`) ∧ `hasColorValue(y, "BlueColor"`). Since there is no predicate that connects the variables, they can be omitted as well. Finally, a set of predicate (attribute)-value pairs remains as a description of the meaning of the query, whereas other relevant information is implicitly assumed. Thus, the attribute-value pairs

$$\{\texttt{(CarType, SportsCar), (ColorValue, "BlueColor")}\}$$

can estimate the need expressed in query (1) quite well.

Indeed, the queries in the form of attribute-value pairs are the most frequently used forms of expressing the user's information need in the so-called product catalogue applications: a user (potential customer) is provided with a list of (carefully selected) questions to provide values for some characteristics (attributes) of the desired product. Such a list of attribute-value pairs is usually interpreted as a query against a database.

[47] Note that this does not solve the problem of the ambiguity of the queries since we define the ambiguity on the level of a user's information need.

However, due to the abstraction process we described above, the predefined values for the attributes provided by such a product catalogue system represent only an iceberg of the domain model that underlies the given problem. Consequently, the opportunities for defining and interpreting a user's information need decreases significantly. Indeed[48],

(1) the user is usually forced to provide the most specialized values for each of the attributes (e.g. they cannot specify that he wants a car with any type of luxury, but rather he has to specify a concrete one), and,

(2) the relationships between selected features are not represented explicitly (e.g. the user does not know that some types of colors are treated as a luxury)

Figure 5.1 illustrates the part of the domain model represented in Figure 4.1 that is visible to the user in a traditional product catalogue application.

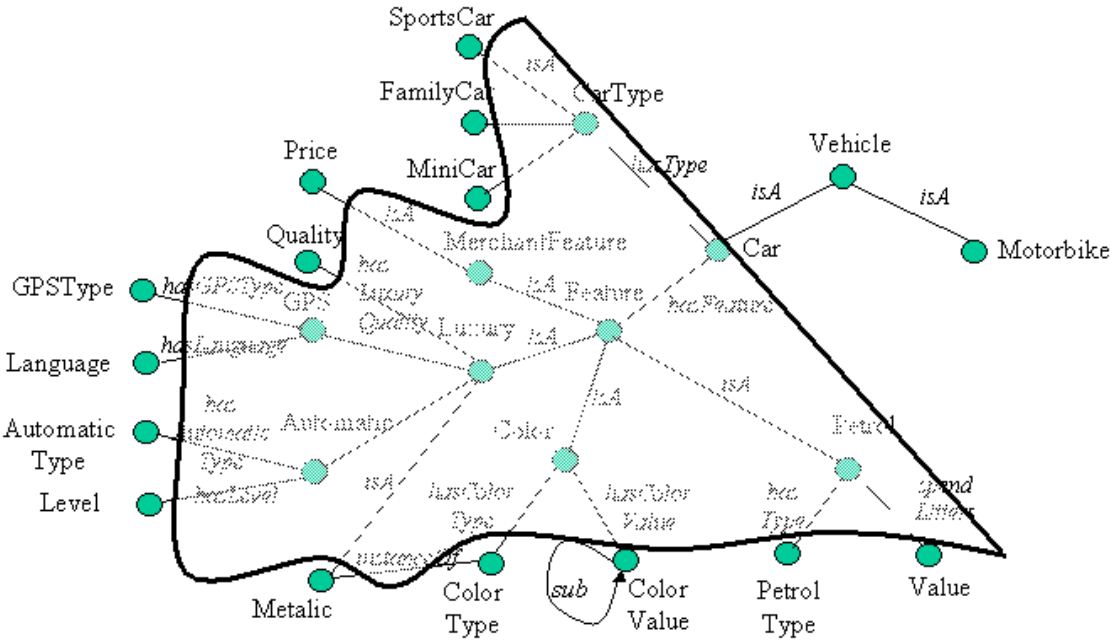


Figure 5.1: Iceberg of the domain model visible in traditional product catalogue applications

Therefore, the main drawback is that the user sees only the surface area of the information space that can be useful for defining his information need. Consequently, the need for efficient approaches that can support the user in refining his initial request is inevitable. It should enable the user to inspect other alternatives for his need, i.e. to enable an efficient navigation through the entire information space. However, a recent analysis of the behavior of on-line shoppers [101] has shown that 55,2% of shoppers miss (i.e. they need) such an assistance in the buying process. We discuss that issue briefly.

The most frequently applied topologies of a product catalogue are (i) a hierarchical, (ii) a product-feature oriented, or (iii) a shopper's need oriented organization [45] which do not support an efficient exploitation of the information space, since the underlying domain model is very simple. Even in the case that the topology of a product catalogue corresponds to the decision tree generated from that product database[49] [144], or that the questions for gathering the user's preferences are elicited from a domain expert [125], the user can find a relevant product efficiently only when he knows exactly what he is searching for. Indeed, the existing

[48] The given examples are related to the domain model represented in Figure 4.1.
[49] In that case the navigation paths are optimised regarding the number of questions which are asked by the system.

product catalogue systems only help the user find the exact product he is searching for, but not in suggesting alternatives which can be probably more relevant for his information need. In other words, a system just executes the user's request instead of interpreting the meaning of that request. In the latter case, the system can understand what the user's information need is (and not only which concrete product the user has in mind in that very moment), so that it could support the user in browsing the information space regarding that need. For example, the system can estimate the importance of each constraint put in the request for the user's need and anticipate additional features of a product that can be relevant for the user. This is especially important in the case, when the user forms a request that does not return any result. In such cases, some unimportant constraints can be removed in order to support the user in finding products most similar to the initially requested one. As far as the author knows, this feature is completely absent from the existing product catalogue systems.

5.2 Requirements for Query Refinement

Let us analyze a small knowledge base (product catalogue) presented in Table 5.1 that is related to the ontology given in Figure 4.1. Each row represents the features assigned to a product (a car), e.g. product P8 is a cabriolet, its color is green metallic and it has an automatic gear changing system. The features are organized in a hierarchy (through the property `sub`), for example, the feature `"BlueColor"` has two specializations `"DarkBlue"` and `"WhiteBlue"` which means that a dark or white blue car is also a blue color car.

Let us suppose that a user wants to buy a car but he is not sure which one. The only feature he is sure about is that he wants a cabriolet. An "ideal" system enables him to make such a query: `Cabriolet`; it moves the user in the corresponding position in the structure[50] of a product-catalogue, i.e. in the position which enables the navigation to all products which have the feature `Cabriolet`. The first problem in traditional approaches [144] is that they do not enable a comprehensive overview of the product-catalogue's structure. Indeed, they provide a set of predefined, hierarchically organized questions which do not enable the user to express an ill-defined information need. The user has to follow questions whose ordering is predefined by the system's developer and cannot be changed on-demand. For example, in a solution which is based on the decision tree generated from the product database presented in Table 5.1, there are several nodes which can correspond to the value "`Cabriolet=true`". It means that the user needs to determine the values of all other features which are placed higher in the decision tree in order to find a starting point for further navigation. However, the problem is that the user knows only the value "`Cabriolet=true`" and the system should enable direct processing of such a need, without demanding the user to first answer the questions about other features.

This leads us to **the first requirement** for the query refinement: to enable ***querying*** the product catalogue so that the right and only one starting point for navigating the product catalogue regarding the user's need can be reached automatically. In the case that the constraints given in a query cannot be fulfilled in the underlying product database, the system should propose a relaxation of that query, which should impose minimally loss in the original user's need.

Let us assume that the system has found the right "position" for the user's need in the product-catalogue structure. Since the user did not start navigation from the root of the product-catalogue structure, the "ideal" system should provide all information which is relevant for the current searching position, in order to orient the user in the product-feature space. For example, according to the Table 5.1, the system should inform the user, who is

[50] For example, a product-catalogue structure is a tree in the case that the catalogue is formed from a decision tree analysis of the product database.

searching for a cabriolet, that (i) all cabriolets are metallic, i.e. that the car he wants has to be metallic and (ii) a cabriolet cannot be a diesel[51]. Further, the system should provide the information that there are three products (P1, P5, P8), which fulfill the current features selected by the user and that all of them have some additional "interesting" features which should be specified. To continue navigation the user can specify one of these features or he can go backward through the product navigation structure and choose another product feature.

Table 5.1: The dataset (product catalogue) used through Chapter 5. In order to achieve better readability of the example, we use a shorthand notation for car features in comparison to the original definitions given in section 5.3. For example, the sign "X" in the cell P1-BlueColor corresponds to the following facts from the underlying knowledge base: *hasFeature*(P1, col1) ∧ *hasColorValue*(col1, "BlueColor")

	Blue Color	Dark Blue	White Blue	Green Color	Metallic	Cabriolet	Automatic	Diesel	Price
P1	X	X			X	X			10 k
P2	X	X					X		8 k
P3	X	X							7 k
P4	X		X					X	8 k
P5	X		X		X	X	X		11 k
P6	X		X				X		8 k
P7				X					5 k
P8				X	X	X	X		11 k

This leads us to **the second requirement**: ***informativeness*** of the search – the system should provide as many as possible pieces of information relevant for the search which can be implied from the search context. This information should help the user to orient himself in the search space and decide how to continue the navigation. For each of the recommended navigations paths the system should explain the reasons behind suggesting it. This feature is completely missing in existing product catalogue systems [125].

By continuing the navigation, the user should be provided with the minimal set of refinements which lead to all products with the selected feature (in this case "`Cabriolet=true`"). The minimal set of refinements means that the user will not be asked for the information which can be deduced from other questions (i.e. no redundant questions are allowed). An example of a redundant question is asking whether the car has `"BlueColor"` after knowing that the car is `"DarkBlue"` (note that each `"DarkBlue"` car has `"BlueColor"`, since these features are organized in a hierarchy through the property `sub`).

On the other hand, the set of refinements (questions) has to be complete, i.e. each product with the selected feature has to be reachable by going via at least one of the given refinements. For example, if the system "forgets" to provide refinement "Automatic=true" from the current position, products P5 and P8 cannot be reached in further navigation, since these products have feature `Automatic` and are relevant for the user's request since they are cabriolets.

[51] In order to understand the dependencies between data presented in Figure 5.1, consider Figure 5.7.

Moreover, all the questions have to be relevant for the current search context, namely, the system must not ask questions which lead nowhere. An example is asking, in the current situation, whether the car can be `Diesel` when it is known (see Table 5.1) that it is impossible to have a `Diesel` car which is a `Cabriolet`.

This leads us to **the third requirement**: the system should provide a ***minimal and complete set of relevant refinements***. This feature is not explicitly presented in the current systems, i.e. the author is not aware of any methodological approach in these systems that ensures minimality and completeness of refinements. Hence, search often provides many results that seem unrelated to the user's request. It was reported that problems in navigating through the product catalogue are the second[52] most cited reason for refusing on-line shopping.

Let us continue our imaginary scenario: the user continues the navigation through the product catalogue. He is interested in the car P1 and he clicks on it in order to get more information. However, he is not ready to buy and wants to refine his needs further. The "ideal" system should analyze the user's past activities in order to anticipate his needs. For example, by analyzing the "clicked" product the system can discover that refinement `"BlueColor"` can be more relevant for the user's need since car P1 has feature `"BlueColor"` but not feature `Automatic`. In order to enable more efficient search, the system should rank the possible refinements (i.e. further navigations) according to their relevance for the user' need.

It can be considered as **the fourth requirement** – exploiting the ***user's relevance feedback***: the system has to discover the user's needs by analyzing his behavior. Learning from the behavior of users is present in a special class of product-catalog applications, so-called recommender systems [7] which learn from the past behavior of usere (i.e. collaborative systems), or use the user's profile to filter the interesting products (i.e. content-based recommenders).

Let us assume that the user chooses refinement "`BlueColor=true`" and selects product P1 for buying.

This is the situation in which the system should provide the information about the relative quality of the selected product (in comparison to other similar products) in order to ensure the user that he has found the most suitable product for his need. First, the system should provide the information, how clearly the user's need is formulated. For example, the user is in a "node" with feature `"BlueColor"` and he should be informed that this feature can be further refined either in `"DarkBlue"` or `"WhiteBlue"`, i.e. the user's need is very general and can (should) be further specified.

Next, the user can be informed that the product he selected does not fit his need perfectly, since he has expressed the need as the color `"BlueColor"`, but he wants now to choose a product with `"DarkBlue"` color. In that case the user's need can be further refined before making the decision about buying.

Last, the user has to be provided with the information about alternative cars which should have slightly different features including the merchant's features (e.g. price, warranty). An efficient query refinement system should take into account all interactions the user has made with the system, in order to elicit what the user's preferences are, i.e. what the user is searching for. This information can be very useful since it suggests to the user which refinements of his query can lead to the retrieval of even more relevant results. For example, the system can "recognize" that an intermediate need was `"BlueColor"` and recommend to the user car P5, which is a better offer than car P1 and which is very relevant for his need. Indeed, car P5 is very similar to P1 (feature `"WhiteBlue"` instead `"DarkBlue"`), it

[52] The first reason concerns security.

approximates well the user's need (the user's preference was "BlueColor") and it has one feature more (i.e. Automatic) for just a slightly higher price (only 10% higher price).

Therefore, **the fifth requirement** is the support for ***estimating the quality*** of the selected product (regarding the user's need and similar products) and ***finding alternative products*** that can be (more) relevant for the user's need.

Finally, let us analyze a case of a failing query. Let us suppose that the user is interested in a Cabriolet car that has an Automatic gear changing system and has "DarkBlue" color. However, in the underlying product catalogue there is no car which fulfils these constraints. It means that the search retrieves no result. An efficient query refinement system should help the user in resolving such situations, first of all by explaining the causes of the problem. For example[53], the system can discover that by eliminating each one of the given three constraints in the query, each resulting subquery retrieves some results, i.e. ?"Cabriolet"+"Automatic":{P5,P8}[54], ?"Cabriolet"+"DarkBlue":{P1}, ?"Automatic"+"DarkBlue":{P2}. Since sub-query "Cabriolet"+"Automatic" returns more objects than the other queries, it is the most appropriate one for the step-by-step expansion. However, since the subquery does not correspond to the initial user's need perfectly (in the case of subquery "Cabriolet"+"Automatic", the feature "DarkBlue" is missing), an efficient query refinement system should recommend to the user the non-failing subquery which least degrades the user's need. Moreover, a new feature that can approximate the missing part of the original query should be recommended to the user. In the given example, the "DarkBlue" feature can be approximated with the feature "WhiteBlue". Indeed, the request "Cabriolet" + "Automatic" + "WhiteBlue" returns results (P5) and approximates well the user's initial need (only the value of the color is replaced with a very similar one). Note that this procedure corresponds to the algorithm for repairing from failing queries, we gave in section 3.2.2.1.

In the next section we present an approach for the refinement of attribute value queries, based on the librarian agent query refinement process, that realizes the requirements elaborated in this section.

5.3 An approach for Ontology-supported Attribute-value-based Query Refinement

The ontology-supported attribute-value querying assumes the existence of an ontology that is used for the description of the information resources' properties. Moreover, the queries are related to the ontology in the following way:

Definition 5.1: *Ontology-based information repository*

An ontology-based information repository *IR* is a structure (*R, O, ann*), where:

- *R* is a set of elements r_i that are called resources, $R=\{r_i\}$, $1 \le i \le n$;

- *O* is an ontology, which defines the vocabulary used for annotating these resources. We say that the information repository is annotated with ontology *O* and knowledge base *KB*(*O*);

- *ann* is a binary relation between a set of resources and a set of facts from knowledge base *KB*(*O*), $ann \subseteq KB(O) \times R$. We write $ann(k_i, r)$, meaning that fact k_i is assigned to resource *r* (i.e. resource *r* is annotated with fact k_i). This relation is called a *context relation*.

[53] Consider Figure 5.7 for the better understanding of this example.

[54] ?"Cabriolet"+"Automatic":{P5, P8} means that for the query ?"Cabriolet"+"Automatic" there are results P5 and P8.

For example, the product catalogue dataset presented in Table 5.1 can be represented as repository $IR_{catalogue}$= (*R, O, ann*),
where:
R={*P1, P2, P3, P4, P5, P6, P7, P8*},
O is the ontology represented in Figure 4.1, and, (2)
ann is the products-features mapping presented in Table 5.1.

Definition 5.2: *Attribute-value query against an ontology-based repository*

An attribute-value (conjunctive) query is of the form or can be rewritten into the form:

$$QAV(O) = \overline{Att}(\overline{Val})$$

with $\overline{Att}$ being a vector of conjoined concepts from the ontology *O* and $\overline{Val}$ being a vector of constants, that corresponds to the instantiation of the concepts used in $\overline{Att}$ vector.

$\overline{Att}$ can be a vector of disjoined concepts; in this case we are talking about a disjunctive attribute-value query. However, in this research we assume that a user makes a conjunctive attribute-value query. This assumption corresponds well to the situation one can account in real systems. However, for some processing (like in Definition 5.15) the disjunctive queries are needed.

For example, considering the ontology given in Figure 4.1, for query:
{`(ColorValue, "BlueColor"), (PetrolType, "Diesel")`} we have:
$\overline{Val}$:= `("BlueColor", "Diesel")`,
$\overline{Att}$:= (Att_1, Att_2), $Att_1(a, b)$:= `ColorValue`(a), $Att_2(a, b)$:= `PetrolType`(b).

Due to better readability we will use a shorthand notion for representing attribute-value queries, obtained by eliminating the name of attributes from a query, like:
`{(ColorValue,"BlueColor"),(PetrolType, "Diesel")} ≡ {"BlueColor","Diesel"}`.

Since the attributes correspond to the concepts that a value "belongs" to, the recovery of the original meaning is straightforward. Moreover, we assume that the values are unique or can be transformed to be unique.

In the next three subsections we shall present the Librarian Agent Query Refinement Process for the attribute-value-based retrieval process.

5.3.1 Phase 1: Ambiguity Discovery

As we have already elaborated in section 3.2.2.1, two main types of query ambiguity arise from the domain model and the information repository, so-called semantic- and content-related ambiguity, respectively.

5.3.1.1 Semantic Ambiguity

As already discussed in section 3.2.2.1 we have found two kinds of tests that can be used for evaluating semantic ambiguity:

- *Generality*: how clear (specific) is a query, and,
- *Compactness*: how complete is the information space covered by a query.

These measures represent strong indicators that there is a possibility to clarify the meaning of a query.

Generality

The *Generality* factor represents uncertainty in determining the user's interest in the given query. For example, when the user makes a query using concept `Vehicle` which contains two

subconcepts `Car` and `Motorbike`, it could be a matter of discussion whether he is interested in the general concept or in one of its subconcepts. Anyway, he has failed to express it in a clear manner.

The *Generality* of an attribute-value query is calculated as the combination of the generality of the corresponding attributes and values.

The *Generality* of an attribute from an attribute-value pair is calculated similarly to the *VariableGenerality* we have described in section 4.3.1.1.

Definition 5.3: *Generality of an attribute:*

$$GeneralityAtt(A_i) = |Subconcepts(A_i)| + 1$$

where:
Subconcepts(C) is the set of subconcepts of concept C. More formally: *Subconcepts*(C) = $\{Cx|H(C, Cx)\}$. See Definition 2.2.

Definition 5.4: *Generality of a value*

$$GeneralityVal(V_i) = Subvalues(V_i) + 1$$

where:
Subvalues(V_i) is the number of entities that can be found in a hierarchy of value V_i. Note that the hierarchy can be any type of relation (not necessary a taxonomical relation). The only request is the transitivity of that relation. An example is the hierarchy introduced by relation `sub` between `ColorValues`, i.e. *Subvalues*(`"BlueColor"`) = 2, since there are two entities (`"DarkBlue"` and `"WhiteBlue"`) in the `sub` hierarchy of `"BlueColor"`, according to the dataset given in Table 4.2.

The *Generality* of an attribute-value pair (A_i, V_i) is calculated as the product of these two parameters, in order to model uniformly the directly proportional effect of both parameters to the ambiguity:

$$GeneralityPair((A_i, V_i)) = GeneralityAtt(A_i) * GeneralityVal(V_i).$$

Finally, the *Generality* of a query is inversely proportional to the *Generality* of its attribute-value pairs:

$$Generality(Q) = (1/m) * \sum_{i, (Ai, Vi) \in Q} (1/GeneralityPair((A_i, V_i))),$$

where m is the number of attribute value pairs in query Q.

Compactness

The compactness factor expresses how a user's information need corresponds to the complexity of the information space. This factor helps us determine how "globally" the user has defined his need, or in other words, how completely the need covers the information space. For example, whether the need covers just a local part of the space (high *Compactness*), or is the need distributed over the whole information space quite uniformly.

Obviously, a well-defined query should exploit the whole information space, since this can ensure that a user's need for an information resource is built with respect to the whole domain model and not just to the part that the user is aware or familiar with. Note that the goal of the query refinement is to "move" the query toward that goal.

The compactness of a query depends on the compactness between each two attribute-value pairs. It is calculated as follows.

Definition 5.5: *Compactness* between two attribute-value pairs (A_i, V_i) and (A_j, V_j) w.r.t. the ontology O:

$$CompactnessPair((A_i, V_i), (A_j, V_j), O) = OM(A_i, A_j, O),$$

where $OM(A_i, A_j, O)$ represents *Object Match* we have defined in section 2.4.2.1 and introduced in [126], and, O is the given ontology.

For example, regarding Figure 4.1 $OM($`ColorType`, `PetrolType`, $O) = 1/2$ since there are just two concepts (i.e. `Color`, `Petrol`) which are different in the least common hierarchy (i.e. concept `Feature`) of the `ColorType` and `PetrolType`.

Since the whole approach is based on the existence of an ontology, in the rest of the text we will use the short version of the previous formula, without parameter O.

Finally, the compactness of the query is the mean value of compactness between each two pairs in the query:

$$Compactness(Q) = (1/n)\sum_{i,\,j(Ai,\,Vi)\,\in Q,\,(Aj,\,Vj)\in Q} (CompactnessPair((A_i, V_i), (A_j, V_j))).$$

where n is the number of pairs of attribute value pairs in query Q. Obviously, if a query contains m attribute-value pairs, $n = \binom{m}{2}$.

Finally, the ambiguity of a query is

$$SemanticAmbiguity(Q) = (1/(a+b))*(a*Generality(Q) + b*Compactness(Q)),$$

where a and b are parameters that enable us to prioritize any of these two factors (per default $a = b = 1$).

5.3.1.2 Content-related Ambiguity

The *content-related ambiguity* of a query depends on the characteristics of the information repository, especially the availability of the relevant resources for the user's query. Since this feature determines a list of results for a query, the content-related ambiguity of the 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 the queries in order to estimate this type of ambiguity.

5.3.1.2.1 Structuring Query Space

Definition 5.6: *Context*

A *Context* in an ontology-based information repository $IR=(R, O, ann)$ (see Definition 5.1) is a tuple:

$$(Q_i, R_i, ann)$$

where $Q_i \subseteq I(O)$, $R_i \subseteq R$ and $ann(Q_i, R_i)$ (see Definition 2.6 for the description of $I(O)$).
In order to define *Context*, context relation *ann* is overloaded to cover set arguments in the following way:

$$ann(q, R_i) \equiv \forall_{r\in Ri}[ann(q, r)],$$
$$ann(Q_i, r) \equiv \forall_{q\in Qi}[ann(q, r)],$$
$$ann(Q_i, R_i) \equiv \forall_{r\in Ri,\,q\in Qi}[ann(q, r)],$$

where $q \in I(O)$, $r \in R$.

5.3.1.3 Properties of the Context

Using the context relation a classification of resources and their attributes (facts from the ontology according to Definition 5.1) can be generated so that each class can be seen as a concept in terms of the properties of associated resources and attributes. In our interpretation, resources and attributes assign meanings to each other via the context relation. Sharing the

meanings of the resources thus can be seen as sharing attributes. We introduce two functions that express this sharing:

The **common attributes** of a set of resources are found by function *ComAttr*: $2^R \rightarrow 2^{I(O)}$ as follows:

$$ComAttr(R_i) = \{q \mid q \subseteq I(O) \wedge ann(q, R_i)\}.$$

The **resources sharing attributes** are captured by function *ComRes*: $2^{I(O)} \rightarrow 2^R$ as follows:

$$ComRes(Q_i) = \{r \mid r \subseteq R \wedge ann(Q_i, r)\}.$$

Definition 5.7: *Request*

A request is a context $\varsigma = (Q', R', ann)$ for which it follows:

$$ComRes(Q') = R'.$$

$Q' \subseteq KB(O)$, is called a set of *ς_attributes*,

$R' \subseteq R$, is called a set of *ς_resources*.

An example for the request regarding the dataset presented in Table 5.1 and (2) is

$$\varsigma = (\{\text{"BlueColor"}, \texttt{Metallic}\}, \{P1, P5\}, ann), \quad (3)$$

where *ann* is the mapping between resources and attributes defined in Table 5.1.
This statement should be read as "{*P1, P5*} is the set of all cars that have `Metallic-Blue Color`".

Note that a request corresponds to the user's request (initial query). In that case, *ς_attributes* are features of a resource the user is interested in and *ς_resources* are resources that fulfill that request. For short, we will represent a request as a pair (Q', R').

Let C be a context. We will write $\rho(C)$ to denote its extensionality R_i and $\alpha(C)$ for its intention Q_i.

In the following we define different relations between the *user's requests* in order to measure the content-related ambiguity of a query.

Definition 5.8: *Extensional equivalence* (=) between two user's requests ς_1, ς_2, is defined by:

$$(Q_1', R_1') = (Q_2', R_2') \leftrightarrow R_1' = R_2', \text{ or}$$
$$(Q_1', R_1') = (Q_2', R_2') \leftrightarrow \text{ComRes}(Q_1') = \text{ComRes}(Q_2')$$

It means that two of the user's requests are structurally equivalent if their sets of result resources (*ς_resources*) are equivalent.

Regarding the product catalogue presented in Table 5.1 and (3) we have:

$$(\{\texttt{Metallic}\}, \{P1, P5, P8\}) = (\{\texttt{Cabriolet}\}, \{P1, P5, P8\}).$$

For a better understanding of this equivalence consider Figure 5.7.

Definition 5.9: *Query concept*

The set of all the extensionally equivalent user's requests forms a *Query concept*:

$$\Delta = (Q_x, R_y), \text{ where}$$

$Q_x \subseteq KB(O)$, Q_x is called a set of *Δ_attributes* (attribute set) and contains the union of attributes of all the requests that result in set R_y. For user's request ς_u it is calculated in the following manner:

$$Q_x = \{q \in KB(O) \mid \forall r \in R_y \wedge ann(q, r)\}.$$

It holds:

$$Q_x = \{ \bigcup_{\forall i \;\; \zeta_i = \zeta_u} \zeta_i \_ attributes \}.$$

$R_y \subseteq R$, R_y is called a set of Δ_*resources* (resource set) and is equal to the ς_*resources* set of query Q_x. Formally:

$$R_y = \{ r \in R \mid \forall q \in Q_x \wedge ann(q,r) \}.$$

Note that the set of resources is equal in all (mutually) equivalent queries.

This context (query concept) is a special situation when the duality of meaning between set Q_x of resources and set R_y of attributes is symmetric:

$$ComRes(Q_x) = R_y$$
$$ComAttr(R_y) = Q_x$$

Let us define further:
$ResClass(D) = ComRes(ComAttr(D))$ and
$AttrClass(A) = ComAttr(ComRes(A))$.

Obviously not every set of resources (attributes) forms a concept. But when it does, at most one concept can be associated with it. So, a concept is uniquely identified by its set of resources or its set of attributes.

An example of a *Query concept* regarding the product catalogue presented in Table 5.1 is:

({`Cabriolet`, `Metallic`}, {*P1, P5, P8*}).

Note that for concept $\Delta = (Q_x, R_y)$, $\rho(\Delta)$ denotes its extensionality R_y and $\alpha(\Delta)$ its intention Q_x.

For each resource *r* in the repository the smallest Query concept containing this resource is called the base concept associated with that resource, denoted as $Base(r)$, or $\Delta_{base}(r)$. The base concept for an attribute (or a set of attributes) is introduced analogously. For a request, the base concept is obtained as the base concept of the set of attributes of that request, i.e.

$$\Delta_{base}(\varsigma) = \Delta_{base}(\alpha(\varsigma)).$$

The *Query concept* which contains all existing resources in *IR* (i.e. a cluster for which $\rho(\Delta)=R$) is called the *root concept*.

The set of all *Query concepts* Δ is denoted by $\Delta(IR)$. We define the following relations on this set:

Definition 5.10: *Extensional subsumption* (parent-child relation) (<) between two query concepts is defined by:

$$\Delta_1 < \Delta_2 \equiv \rho(\Delta_1) \subset \rho(\Delta_2).$$

Having a restricted extensional meaning is equivalent to having an augmented intentional meaning:

$$\Delta_1 < \Delta_2 \leftrightarrow \alpha(\Delta_2) \subset \alpha(\Delta_1).$$

For a single query concept we define a special subsumption relation ($<_{dir}$) on the set of Query concepts as:

$$\Delta_1 <_{dir} \Delta_2 \leftrightarrow \Delta_1 < \Delta_2 \wedge \neg\exists \Delta_i, \Delta_1 < \Delta_i < \Delta_2.$$

In that case we call Δ_2 a direct_parent concept of Δ_1 and Δ_1 a direct_child concept of Δ_2. The sets of direct_parent and direct_child clusters of a *Query concept* are calculated in the following way:

$$DirectParents(\Delta_1) = \{\Delta_i \in \Delta(IR) \mid \rho(\Delta_1) \subset \rho(\Delta_i) \wedge \neg\exists\, \Delta_j \; \rho(\Delta_1) \subset \rho(\Delta_j) \subset \rho(\Delta_i)\}$$

and

$$DirectChildren(\Delta_1) = \{\Delta_i \in \Delta(IR) | \alpha(\Delta_i) \subset \alpha(\Delta_1) \wedge \neg\exists\, \Delta_j\, \alpha(\Delta_i) \subset \alpha(\Delta_j) \subset \alpha(\Delta_1)\}$$

Note that a parent-child relation is irreflexive, anti-symmetric and transitive. Therefore, this relation introduces a partial ordering in the set of concepts.

Let C be a set of concepts. A lower bound of C is a *common subconcept*. If there exists a greatest element in the set of lower bounds of C, then this element is called the *greatest lower bound* and is denoted as $\wedge(C)$. Likewise, the smallest element in the set of upper bounds is called the *smallest upper bound*, denoted as $\vee(C)$.

Lemma 5.1:
It can be proven that for each set of concepts C:

$$\rho(\wedge(C)) = \cap_{c\in C}\rho(c)$$
$$\alpha(\vee(C)) = \cap_{c\in C}\alpha(c)$$

Proof:
We will prove this lemma only for the upper bound of a set of concepts C.

$\alpha(\vee(C))$
$= ComAttr(\cup_{c\in C}\rho(c))$ //from the common attribute property of a context
$= \cap_{c\in C} ComAttr(\rho(c))$ //since ComAttr calculates common elements
$= \cap_{c\in C}\alpha(c)$.

Theorem 5.1: A *Query concept* inherits all common Δ_resources from all its direct_parent concepts:

$$\Delta_1 = (Q_{x1}, R_{y1}) = (Q_{x1}, \bigcap_{\Delta_1 <_{dir} \Delta_i} R_{yi}).$$

Proof:
We show that it holds:

$$R_{y1} = \bigcap_{\Delta_1 <_{dir} \Delta_i} R_{yi} \qquad (4)$$

Let us assume that query concept Δ_1 has *n-1* direct_parents, $\Delta_i, i = 2..n$, $n > 2$. Regarding Definition 5.10, for each of these direct parents (Δ_i) it follows:

$$(Q_{x1}, R_{y1}) <_{dir} (Q_{xi}, R_{yi}) \leftrightarrow R_{y1} \subset R_{yi} \qquad i = 2..n \qquad (5)$$

Further,

$$R_{y1} \subset R_{y2} \wedge R_{y1} \subset R_{y3} \ldots \wedge R_{y1} \subset R_{yn} \Rightarrow R_{y1} \subseteq (R_{y2} \cap R_{y3} \ldots \cap R_{yn}) \qquad (6)$$

We show that in (6) $R_{y1} = (R_{y2} \cap R_{y3} \ldots \cap R_{yn})$ holds. In the case that there is an object which is not part of R_{y1} and it is an element of $(R_{y2} \cap R_{y3} \ldots \cap R_{yn})$, then

$$\exists R_w \in (R_{y2} \cap \ldots \cap R_{yn}) \wedge R_w \notin R_{y1} \Rightarrow \exists \Delta_k = (Q_k, R_1 \cup \{R_w\}) \wedge \neg(\Delta_1 \leq_{dir} \Delta_k) \qquad (7)$$

For Δ_k it holds that $R_w \in R_{yk} \subseteq (R_{y2} \cap \ldots \cap R_{yn})$. It means that:

$R_{y1} \subseteq R_{yk}$, i.e. $\Delta_1 < \Delta_k$ (8)

Now, from (7) and (8) we have that $\neg(\Delta_1 \leq_{dir} \Delta_k)$ and $\Delta_y < \Delta_k$, which means that between Δ_y and Δ_k there is another query, i.e. $(\Delta_1 \leq_{dir} \Delta_i \leq_{dir} \Delta_k)$, which is impossible since Δ_y and Δ_k differ only in one object (R_w). Hence $R_{y1} = (R_{y2} \cap R_{y3} \ldots \cap R_{yn})$, i.e. (4) is true.

Corollary 5.1: For two *Query concepts*, Δ_1, Δ_2, there is a *greatest lower bound* Δ_{comm_child}. Moreover, it is calculated as the concept whose set of resource is: $R_{ycomm\_child} = (R_{y1} \cap R_{y2})$.

The validity of this statement can be proved similar to Theorem 5.1.

Theorem 5.2: A set of common attributes for a set of *Query concepts* is the intersection of sets of their attributes (Δ_attributes), i.e. $CommonAttr(\Delta_1, \Delta_2, ..., \Delta_m) = \bigcap_{i=1,m} Q_{xi}$

The proof can be done in the similar way as in the case of Theorem 5.1.

Corollary 5.2: For two *Query concepts*, Δ_1, Δ_2, there is a unique *smallest upper bound* Δ_{comm_parent}. Moreover, it is calculated as the concept whose set of attributes is: $Q_{xcomm\_parent} = (Q_{x1} \cap Q_{x2})$.

The existence of Δ_{comm_parent} follows directly from Theorem 5.2. The uniqueness can be shown in a similar way as for the concept lattice structure, given in [52] as "the basic theorem on concept lattices."

Corollary 5.3: For each attribute-value pair Aa there is a Query concept Δa which is the unique common parent (not obligatory a direct_parent) for all other Query concepts containing element Aa in their Δ_attributes.

Proof: Let's assume that there are *m* Query concepts, $\Delta_1, ..., \Delta_m$ and that all of them contain the element Aa in their Δ_attributes (i.e. $Aa \in Q_{xi}$, i=1..m). Let's divide these clusters in *k* pairs ($k = \left\lceil \frac{m}{2} \right\rceil$), so that each group is contained in at least one pair. According to the Corollary 5.2, for each pair of groups (Δ_1, Δ_2) there is a unique common parent Δ_{parent} which fulfills $Q_{xparent} = (Q_{x1} \cap Q_{x2}) \supset A_a$. After this step, there are *k* clusters which are common parents of $\Delta_1, ..., \Delta_m$ and contain element Aa in their Δ_attributes. By repeating the step maximal $\left\lceil \frac{k}{2} \right\rceil$ times, cluster, Δ_a, remains, which is the unique common parent of all *m* clusters, $\Delta_1, ..., \Delta_m$ and which contains element Aa in its Δ_attributes.□

Moreover, this unique common parent cluster, Δ_a, fulfills the following condition:

$$\neg\exists \Delta_i, \Delta_a <_{dir} \Delta_i \wedge Q_{xi} \subset \{Aa\} \wedge Q_{xa} \subset \{Aa\},$$

i.e. there is no parent cluster of Δ_a which contains Aa in its Δ_attributes.

Note that Corollary 5.3 is valid not only for a single attribute, but for a set of attributes, i.e. there is a unique common parent for all Query concepts clusters which contain a set of attribute-value pairs. The proof is straightforward.

This Corollary is very important since it ensures that for each set of attribute-value pairs (*Aa*) expressed in a user's request there is a unique Query concept, $\Delta_{base}(Aa)$, which is the starting point for the navigation through the dataset (i.e. product catalogue, see section 5.4), as required in the motivating example given in section 5.2.

Therefore, since each set of concepts has a unique lower and upper bound, the resulting lattice structure is a complete lattice, as illustrated in Figure 5.2.

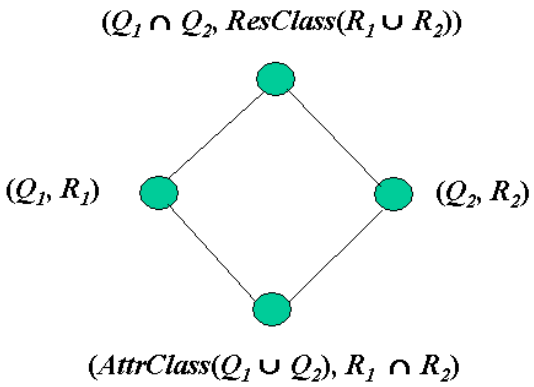


Figure 5.2: Lattice organization of query concepts

Moreover, the partial order of *Query concepts* induces many properties that are important for an efficient manipulation of the user's queries:

Definition 5.11: *Similarity* (siblings relation) (~)

We define two kinds of structural similarity between two different Query concepts Δ_1 and Δ_2:

- common_parent ($\sim_{parent}$):

$$\Delta_1 \sim_{parent} \Delta_2 \leftrightarrow \exists \Delta_i \Delta_1 <_{dir} \Delta_i \wedge \Delta_2 <_{dir} \Delta_i ,$$

- common_child ($\sim_{child}$):

$$\Delta_1 \sim_{child} \Delta_2 \leftrightarrow \exists \Delta_i \Delta_i <_{dir} \Delta_1 \wedge \Delta_i <_{dir} \Delta_2 .$$

Query concepts that are in a similarity relation are called *Similar Concepts*.

Consider Figure 5.6 for an illustration of the similarity relation.

Definition 5.12: *Disjoint Query concepts* ($<>$) are concepts which have no resources in common, i.e.

$$\Delta_1 <> \Delta_2 \equiv \rho(\Delta_1) \cap \rho(\Delta_2) = \{\}.$$

5.3.1.3.1 Role of an Ontology - Semantic Lattice

We see two advantages of using an ontology for organizing a lattice structure:

(i) the creation of a so-called ***semantic neighborhood***, in which the relations between the query concepts are defined according to the relations from the ontology. Moreover, the relevance of these relations is determined semantically, and,

(ii) ***semantic grouping*** of query concepts that belong to the direct neighborhood of a query concept according to the structure of an ontology.

In the rest of this subsection we discuss these issues.

Semantic Neighborhood

The extensional subsumption between queries is defined on the set of resources, i.e. a query is subsumed by the other query if the results of the first query are subsumed by the set of results of the second query. However, we have already mentioned in Definition 5.10 that this relation can be interpreted on the level of the attributes of a query. Since, according to the definition of an attribute-value query, these attributes represent the concepts from the ontology, so that the relations between the query concepts can be ranked semantically.

First, we define two special types of direct children that reflect these semantic relations between the queries.

Definition 5.13: *Specialisation/generalisation* $<_{spec}, >_{gen}$ by:

A query is the specialization of another query, when:

$$\Delta_1 <_{spec} \Delta_2 \leftrightarrow \Delta_1 <_{dir} \Delta_2 \wedge \exists q_{t1} \in Q_{x1} \quad \exists q_{t2} \in Q_{x2} \quad \exists h \in H \quad h(q_{t1}, q_{t2}),$$

where H is a hierarchical relation as we have described in Definition 5.4. For example, it can be a concept hierarchy (consider Definition 2.2).

A query is the generalization of another query, when:

$$\Delta_1 >_{gen} \Delta_2 \leftrightarrow \Delta_2 <_{dir} \Delta_1 \wedge \quad \exists q_{t1} \in Q_{x1} \quad \exists q_{t2} \in Q_{x2} \quad \exists h \in H \quad h(q_{t2}, q_{t1})$$

$<_{spec}$ and $>_{gen}$ are transitive and mutually inverse relations.

The following two theorems help with determining candidates for the specialisation/generalisation of a query. The proofs are straightforward using Definition 2.2.

Theorem 5.3: If a direct_child is the specialisation of the given query concept, then there is another direct_parent of that direct_child that contains an attribute which is in a hierarchical relation with an attribute from the given query.

$$\Delta_a <_{spec} \Delta_b \rightarrow (\Delta_a <_{dir} \Delta_b) \wedge \exists \Delta_i, \Delta_a <_{dir} \Delta_i \wedge \exists x \in Q_{xi} \wedge \exists y \in Q_{xb} \wedge \exists h \in H \quad h(x, y))$$

Theorem 5.4: If a direct_parent is the generalisation of the given query concept, then there is another direct_child of that direct_parent that contains an attribute which is in a hierarchical relation with an attribute from the given query.

$$\Delta_a >_{gen} \Delta_b \rightarrow (\Delta_b <_{dir} \Delta_a) \wedge \exists \Delta_i, \Delta_i <_{dir} \Delta_a \wedge \exists x \in Q_{xi} \wedge \exists y \in Q_{xb} \wedge \exists h \in H \quad h(y, x))$$

Moreover, considering an ontology we can define the semantic relations between siblings:

Definition 5.14: *Sibling specialization* (real sibling) ($\sim_{spec}$) by:

$$\Delta_1 \sim_{spec} \Delta_2 \leftrightarrow \Delta_1 \sim_{parent} \Delta_2 \wedge \exists x \in Q_{x1}, y \in Q_{x2} \wedge \quad \exists h \in H, z \quad h(x, z) \wedge h(y, z)$$

The following theorem helps with calculating candidates for the real sibling. The proof is straightforward using Definition 2.2.

Theorem 5.5: If two queries, Δ_a, Δ_b, have only one and the same direct_parent they are real siblings.

$$(\Delta_a <_{dir} \Delta_c) \wedge (\Delta_b <_{dir} \Delta_c) \wedge \neg \exists \Delta_d, \Delta_e \neq \Delta_c \wedge (\Delta_a <_{dir} \Delta_d) \wedge (\Delta_b <_{dir} \Delta_e) \rightarrow \Delta_a \sim_{spec} \Delta_b$$

Note that in the case that the ontology allows the expression of disjoint relations, semantically disjoint concepts can be defined in an analogous way.

Finally, the ontology is used for ranking subsumed queries, which will be described in detail in section 5.3.3, where we describe our approach for the ranking of refinements.

Consider Figure 5.6 for an illustration of the relations presented in this section.

Semantic Grouping

Since the query part of a query concept is in the form of a set of attribute-value pairs, it is possible that several query concepts from the same neighborhood (e.g. direct_children) contain different values for the same attribute. Such concepts are grouped in a virtual concept.

For example, it is possible that two direct children contain, among others, attribute values `"BlueColor"` and `"GreenColor"`, respectively, which belong to the same concept in the ontology. In that case, these two concepts should be grouped in a (virtual) concept in order to enable the user to select one of ontology-related attribute values at once.

Definition 5.15: *Virtual Concept*

Virtual concept is of the form

$$\Delta v = (Q_x, V, R_y),$$

where:
V represents a disjunctive attribute-value query (see Definition 5.2), and,
Q_x and R_y have the same meaning as in Definition 5.9.

Definition 5.16: *Grouping Operator* (ν)

$$\nu: 2^{dc(x)} \times \Delta_x \rightarrow 2^{\Delta v(IR)},$$

where:

$dc(x)$ corresponds to the set $DirectChildren(\Delta_x)$, and,

$\Delta v(IR)$ represents the set of all virtual concepts.

The operator is defined as follows:

$$\nu(\Delta_{x1},\ldots,\Delta_{xk},\Delta_x) = \{(Q_{xx}, \vee_{p,\ Attribute(refxp)=Attribute(ref)} ref_{xp}, \cup_{p,\ Attribute(refxp)=Attribute(ref)} R_{yxp}) \mid \forall ref \in \cup_{i\in\{1,\ldots,k\}} Q_{xi}\ \exists ref_i\ _{i\in\{1,\ldots,k\}}\ Attribute(ref) = Attribute(ref_i) \wedge Value(ref) \neq Value(ref_i) \wedge ref \neq ref_i\}$$

where:
Δ_{xi} is a direct_child of $\Delta_x = (Q_{xx}, R_{yx})$ in the form $(Q_{xx} \wedge ref_{xi}, R_{yxi})$,
$Attribute(ref)$ returns the attribute from the attribute-value pair *ref*, and,
$Value(ref)$ returns the value from the attribute-value pair *ref*.

Therefore, several child_concepts can be joined in a virtual concept, if they have some identical attributes, but with different values of these attributes.

Figure 5.3 a) illustrates the process of calculating a Virtual concept and 5.3 b) presents an example of it

Note that various levels of granularity can be achieved by using an ontology-based hierarchical structure in defining the equality between two concepts in an ontology.
Note that the goal of the virtual grouping is only the visual presentation of the query concepts. In the query refinement process, the user selects a concrete value from the (virtual) concept so that the lattice structure remains intact for further calculation, i.e. further calculation is done independently of the existence of virtual concepts.

5.3.1.3.2 Quantifying Content-related Ambiguity

As we have mentioned in section 3.2.2.2 there are two general types of content-related ambiguity: *Confidence* and *Specificity*. In section 4.3.1.2 we described how they can be calculated in a lattice structure (see Figure 4.3). Here we show how that calculation can be realized using a query concept- lattice structure.

Confidence

Confidence between a query *q* and a query *qc'* where *qc'* is the refinement of *q* with attribute *c* that is calculated as:

$$Confidence(qc', q) = \frac{|\rho(\Delta_c)|}{|\rho(\Delta_{base}(q))|},$$

where:
$|\rho(\Delta_x)|$ depicts the number of resources contained in concept Δ_x. It can be easily calculated by using the *Covering* parameter (see below),
$\Delta_{base}(q)$ is the base concept that corresponds to the given query, and,

Δ_c is a direct_child concept whose set of attributes contains *c*, that can be a candidate for the query refinement of *q* and therefore is not contained in concept $\Delta_{base}(q)$.

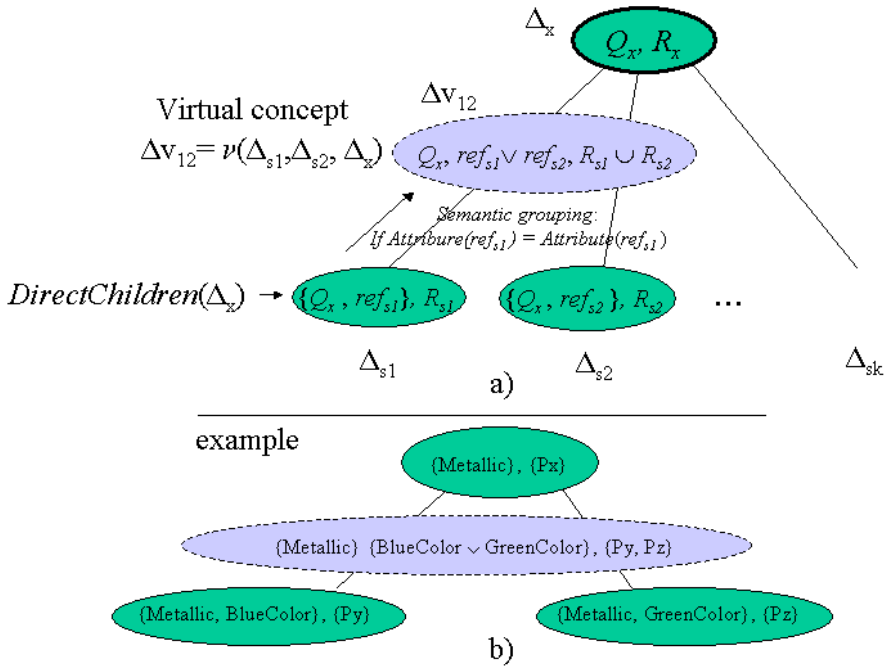


Figure 5.3: Semantic grouping process: a) The process of calculating a virtual concept , b) An example of virtual concept

Specificity

Similarly, the *Specificity* of the refinement *c* for a query *q* can be calculated as

$$Specificity(qc', q) = \frac{|\rho(\Delta_c)|}{|\rho(\Delta_{sibling}(q))|},$$

where:

qc' that corresponds to the query that is refinement of the *q* with attribute *c*,

$\Delta_{sibling}(q) \sim_{child} \Delta_{base}(q)$ and $\Delta_c <_{dir} \Delta_{sibling}(q) \wedge \Delta_c <_{dir} \Delta_{base}(q)$.

$\Delta_{base}(q)$ is the base concept that corresponds to the user's request and Δ_c is a direct_child concept whose set of attributes contains *c*.

Note that 0 < *Confidence*, *Specificity* < 1. A larger value indicates larger suitability of a constraint to be used for refining a given query. In this way it is ensured that the refined query will retrieve "enough" resources to the user.

However, in order to make the user navigate through the query space more easily, we have defined several additional parameters for estimating content ambiguity: *Max_equal_request*, *Min_equal_request*, *NonUniquenessOfAnAttribute*, *NecessarySetOfResources*, *DisjointTerms*, *Covering* and *CoveringAttributes*.

In the rest of this section we define these parameters[55]. Figure 5.4 and Figure 5.5 give a graphical representation of these parameters.

Max_equal_request

[55] Consider Figure 5.7 for the better understanding of examples.

Max_equal_request of a user's request ς_a is a set of attributes found in its largest (regarding the set of attributes) extensionally equivalent request, $\varsigma_{a\,max}$. $\varsigma_{a\,max}$ is equal to the base concept *Base*(ς_a), or Δ_a as a shorthand here. This concept is calculated as the smallest concept that contains all the attributes from the given user's request, i.e.:

$$\Delta_a = (ComAtr(ComRes(\alpha(\varsigma_a))), ComRes(\alpha(\varsigma_a))$$

Therefore,

$$Max\_equal\_request(\varsigma_a) = ComAtr(ComRes(\alpha(\varsigma_a))).$$

For example, regarding the product catalogue presented in Table 5.1:
Max_equal_request(({`Metallic`}, {*P1, P5, P8*})) = {`Cabriolet`, `Metallic`}. (9)

Min_equal_request

Min_equal_request is a set of attributes found in the smallest (regarding the set of attributes) equivalent request for the given user's request ς_a, $\varsigma_{a\,min}$. There can be several such requests and the *Min_equal_request* is calculated as the cross product of these, in the following way:

$$Max\_equal\_request(\varsigma_a) = \{(\times(\alpha(\Delta_a) \cap \alpha(\Delta_x))| \forall x\ \Delta_a <_{dir} \Delta_x\},$$

where $\Delta_a = \Delta_{base}(\varsigma_a)$.

For example (consider Table 5.1):
Min_equal_request(({`Cabriolet`, `Metallic`, `Automatic`}, {*P5*, *P8*}))
= {{`Cabriolet`, `Automatic`},{`Metallic`, `Automatic`}}.

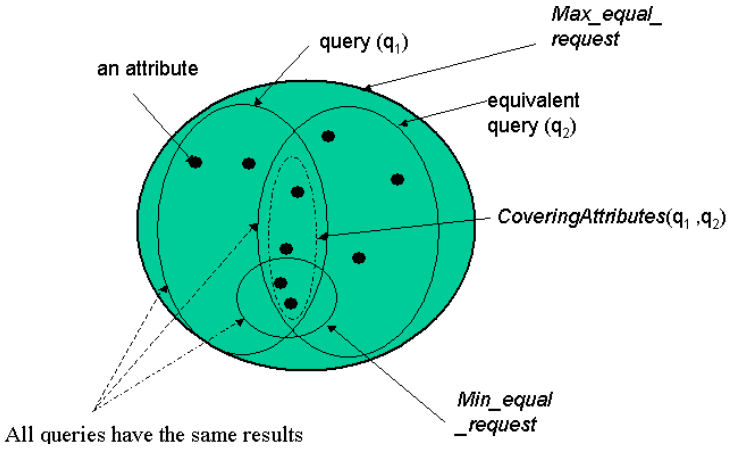


Figure 5.4: An illustration of content-related parameters (part I)

NonUniquenessOfAnAttribute

For each attribute *q* from request ς_a it is possible to define its uniqueness for that request by accounting the number of parent requests that contain this attribute. The less the number of such parents is, the higher the uniqueness of that attribute. If the given request corresponds to the base concept of that attribute, its nonuniqueness is minimal.

NonUniquenessOfAnAttribute(ς_a, *q*)=

$$|\{x|\ \Delta_a <_{dir} \Delta_x \wedge q \in \alpha(\Delta_x) \wedge \exists e \in \alpha(\Delta_x) \wedge e \neq q\}|\ /\ |\{x|\ \Delta_a <_{dir} \Delta_x\}|,$$

where $\Delta_a = \Delta_{base}(\varsigma_a)$.

For example, regarding the product catalogue presented in Table 5.1:
NonUniquenessOfAnAttribute(
({`Automatic`, `Metallic`, `BlueColor`}, {*P5*}), {`Automatic`}) = 2/3.

NecessarySetOfResources

A set of resources which have to be contained in $\varsigma\_resources$ of a query is called the *NecessarySetOfResources* of that query. It means that by excluding these resources from the $\varsigma\_resources$ set, that query becomes equivalent to one of the direct_child queries. For a query there can be several sets of *NecessarySetOfResources*, which are calculated as follows:

$$NecessarySetOfResources(\varsigma_a) = \{\rho(\Delta_a) \setminus \{\cup \rho(\Delta_x) | \forall x\ \Delta_x <_{dir} \Delta_a\}\},$$

where $\Delta_a = \Delta_{base}(\varsigma_a)$.

For example, regarding product catalogue presented in Table 5.1:
NecessarySetOfResources(({`GreenColor`}, {*P7*, *P8*})) = {*P7*}.

DisjointTerms

DisjointTerms is a set of Δ_attributes which cannot be found in the current concept, i.e. these attributes are disjoint with the current set of attributes. In other words, this is a set of Δ_attributes that are contained in *Disjoint* concepts. It can be calculated as:

$$DisjointTerms(\Delta_a) = \{Q_b \mid Q_b \in Q_{yi}, \Delta_i <> \Delta_a\}.$$

Covering and CoveringAttributes

Covering and *CoveringAttributes* are measures which define the percentage of identical resources and attributes, respectively, in two queries. More formally, for two queries ς_a and ς_b we define:

$$Covering(\varsigma_a, \varsigma_b) = |\rho(\varsigma_a) \cap \rho(\varsigma_b)| \ / \ \max(|\rho(\varsigma_a)|, |\rho(\varsigma_b)|)$$

$$CoveringAttributes(\varsigma_a, \varsigma_b) = |\alpha(\varsigma_a) \cap \alpha(\varsigma_b)| \ / \ \max(|\alpha(\varsigma_a)|, |\alpha(\varsigma_b)|)$$

For example, regarding product catalogue presented in Table 5.1:
Covering(({`Cabriolet`}, {*P1*, *P5*, *P8*}), ({`Cabriolet, DarkBlue`}, {*P1*})) = 1/3.

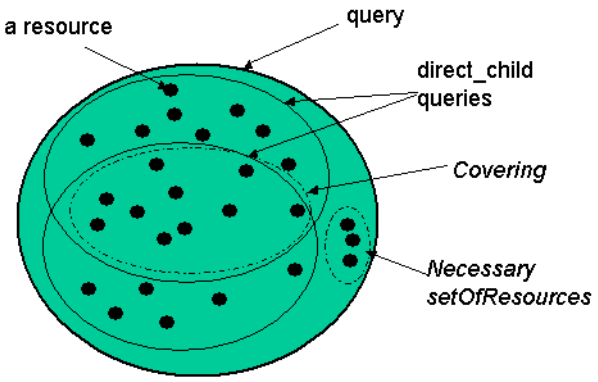


Figure 5.5: An illustration of content-related parameters (part II)

It is clear that the calculation of all the above mentioned parameters could be time-consuming. In order to make this calculation more effective we use formal concept analysis (FCA) [52] for organizing data in so-called concept lattices, which correspond to multi-inheritance hierarchical clusters. This is elaborated in section 5.4.

These parameters define the ambiguity parameters realized in a query compass, illustrated in Figure 3.10. Table 5.2 summarizes the role of these parameters in the navigation process.

5.3.2 Phase 2: Refinement Generation

As we have already mentioned in section 3.2.2.2, there are three refinements tactics that can be applied in the refinement phase: *Refinement*, *Query Neighborhood* and *Cooperative Answering* tactics. In the next three subsections we describe them in details.

5.3.2.1 Refinement Tactics: Generating Minimal and Complete Set of Refinements

The goal of the query refinement process is to support the disambiguation of a query by adding some constraints to it. In that way the misinterpretations of a query regarding a user's information need should be reduced and the list of results will be more relevant. However, the problem is that the search space for the query refinements is very large. Theoretically, any constraint that does not produce a failing query can be added to the initial query since any such query can be treated as an appropriate one for the given information need. Indeed, the set of candidates for the refinement of query ς_a looks like:

$$Candidate(\varsigma_a) = \{q' \mid \rho(query(q', \varsigma_a)) \neq \{\}\}$$

where $query(q, \varsigma_a)$ represents a refinement of request ς_a with constraint q.

Table 5.2: The suggestions for the query refinement, which are based on the analysis of the ambiguity parameters

Value	Meaning	Action
Big difference between *Min_equal_request* 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 *Max_equal_request* 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.
Large *NonUniquenessOfAn Attribute*	Query is not specific enough w.r.t. the parent queries	Move the query toward child_queries in order to narrow the scope of the query.
Few elements in the *NecessarySet OfResources*	The query shares almost all results with other queries	If more specific results are needed, then replace the query with some neighborhood query.
Large *Covering/ CoveringTerms*	The query gives similar results as a query from the neighborhood	If more results are needed, then move the query in the direction of that "similar" query.

A naive strategy is to generate all elements from this set as candidates for the refinement. If these candidates are totally independent as far as the quality is concerned, there is no alternative to such an exhaustive generation (search for refinements).

If, however, the candidates are dependent w.r.t. a quality criterion, a more intelligent strategy is applicable: only the candidates that do not depend on other candidates should be generated and presented to the user. We define dependency between the constraints q, q' in request ς_a as follows:

$$dependsOn(q, q', \varsigma_a) \equiv \alpha(\Delta_{base}(query(q', \varsigma_a))) \subseteq \alpha(\Delta_{base}(query(q, \varsigma_a)))$$

Lemma 5.1:
If the user is not interested in a refinement, he is not interested in all refinements that depend on it.

Proof:

Let us assume that there is in refinement x of request ς_a a constraint q_x that the user is not interested in. Since all the refinements that depend on x contain that constraint q_x, it follows that none of them will be interesting for the user □.

Therefore, the task of the refinement generation phase is to find the set of all independent (simple) refinements. In that way, we ensure that the user can be provided with enough information to decide whether a refinement corresponds to his information need or not. This type of refinement is called *step-by-step refinement* since the user develops his query in the so-called step-by-step fashion, as introduced in Chapter 3.

Definition 5.17: *Simple constraint*

A refinement constraint is a simple one for a given query if adding that constraint to the query does not imply adding some other query constraints, except in the case that those constraints belong to an equivalent request. More formally:

$$SimpleConstraint(q, \varsigma_a) = true \rightarrow$$
$$\neg\exists\, q'\, q' \in \alpha(\Delta_{base}(query(q, \varsigma_a))) \wedge q \notin \alpha(\Delta_{base}(query(q', \varsigma_a)))$$
$$\equiv \forall q' \in \alpha(\Delta_{base}(query(q, \varsigma_a))) \rightarrow q \in \alpha(\Delta_{base}(query(q', \varsigma_a))),$$

where $query(q, \varsigma_a)$ represents a refinement of request ς_a with constraint q.

Secondly, the set of refinements that are provided to the user has to be complete, which means that every possible information need that is approximated in the given user's request has to be fulfilled from this refinement set. This is expressed in the following definition.

Definition 5.18: *Complete Set of Refinements*

$$CompleteSet(set, \varsigma_a) = true \rightarrow$$
$$\forall\, q'\, q' \notin set \wedge \rho(query(q', \varsigma_a)) \neq \{\}\ \exists q \in set\ \ \alpha(query(q, \varsigma_a)) \supset \alpha(query(q', \varsigma_a))$$

where ς_a is a user's request and *set* is the set in question.

If each element in the refinement set is simple, we call this set minimal. More formally:

$$MinimalSet(set, \varsigma_a) = true \rightarrow$$
$$\forall q \in set\ SimpleConstraint(q, \varsigma_a) = true$$

Lemma 5.2:
The set of constraints that represents the difference between the constraints that can be found in the union of direct children concepts of the given user's request and the constraints found in the user's request forms a minimal and complete set of refinements. More formally,

$$Candidate(\varsigma_a) = \{\cup\alpha(\Delta_x) \setminus \alpha(\Delta_a) |\ \forall x\ \Delta_x <_{dir} \Delta_a\},$$

where $\Delta_a = \Delta_{base}(\varsigma_a)$.
It follows:

$$MinimalSet(Candidate(\varsigma_a)\ ,\ \varsigma_a) = true$$
$$CompleteSet(Candidate(\varsigma_a)\ ,\ \varsigma_a) = true$$

Proof:

a) Minimality

We use the contradiction:
Let us assume that there is constraint q' that belongs to $Candidate(\varsigma_a)$ and it is not a simple refinement for ς_a. It means that this constraint can be omitted from $Candidate(\varsigma_a)$ since it can be obtained in the next refinement step.

Since we assume that *q'* is not a simple refinement, it follows that there is a constraint $q_k \in Candidate(\varsigma_a)$ for which:

$$q_k \in \alpha(\Delta_{base}(query(q', \varsigma_a))) \wedge q' \notin \alpha(\Delta_{base}(query(q_k, \varsigma_a)))$$

Since $q' \in \alpha(\Delta_{base}(query(q', \varsigma_a)))$ it follows:

$$\alpha(\Delta_{base}(query(q', \varsigma_a))) \subset \alpha(\Delta_{base}(query(q_k, \varsigma_a)))$$

and further:

$$\Delta_{base}(query(q_k, \varsigma_a)) < \Delta_{base}(query(q', \varsigma_a)). \quad (10)$$

Since $q_k \in Candidate(\varsigma_a)$ it follows that $\Delta_{base}(query(q_k, \varsigma_a))$ is a direct child cluster of ς_a.

According to the definition of a direct_child cluster and (5) $\Delta_{base}(query(q', \varsigma_a))$ is not a direct child cluster, i.e. *q'* does not belong to a direct child cluster.

Consequently $q' \notin Candidate(\varsigma_a)$ which is contradictory to the initial assumption.

b) Completeness

We use the contradiction:
Let us assume that there is a constraint *q'* that does not belong to $Candidate(\varsigma_a)$ and whose base cluster for ς_a cannot be obtained as a refinement of any direct child clusters.

$$\exists q' \notin Candidate(\varsigma_a) \wedge \delta(query(q', \varsigma_a)) \neq \{\} \wedge \neg\exists$$
$$\Delta_x <_{dir} \Delta_{base}(\varsigma_a), \text{ such that } \Delta_{base}(query(q', \varsigma_a)) < \Delta_x$$

Due to the monotonicity of the retrieval function, $\delta(\varsigma_a) \subseteq \delta(query(q', \varsigma_a))$. Consequently, $\Delta_{base}(query(q', \varsigma_a)) < \Delta_{base}(\varsigma_a)$. Since there is no direct child cluster that is subsumed by $\Delta_{base}(query(q', \varsigma_a))$ it follows that $\Delta_{base}(query(q', \varsigma_a)) <_{dir} \Delta_{base}(\varsigma_a)$.

Consequently, $q' \in Candidate(\varsigma_a)$ which is contradictory to the initial assumption.

Therefore, by considering the constraints contained in the direct child concepts of a *Query concept*, a set of minimal and complete set of refinements for that *Query concept* can be generated. It means that each information need that is approximated in the user's request can be refined in the direct child clusters of that request.

5.3.2.2 Query Neighborhood Tactic

We define here only the basic query neighborhood structure. The weighted structure that is used in ranking will be described in section 5.3.3.

Definition 5.19: *Query Neighborhood*

The Neighborhood of the user's request ς_u is the structure

$$N := (E, P, C, \gamma, \eta),$$

where:

- $E := \{\varsigma_i \mid \varsigma_i = \varsigma_u\}$, i.e. the set of all user's requests equivalent to ς_u. Consequently, all the requests from *E* form the base concept for ς_u, i.e. $E = Base(\varsigma_u)$ in the notion $\Delta_{base}(\varsigma_u)$,
- $P := \{\Delta_i \mid \Delta_{base}(\varsigma_u) <_{dir} \Delta_i\}$, i.e. the set of all direct_parent clusters of $\Delta_{base}(\varsigma_u)$,
- $C := \{\Delta_i \mid \Delta_i <_{dir} \Delta_{base}(\varsigma_u)\}$, i.e. the set of all direct_child clusters of $\Delta_{base}(\varsigma_u)$,
- $\gamma : P \rightarrow \Re$, is the relevance function for direct_parent clusters (it is used for ranking direct_parent clusters), and,
- $\eta : C \rightarrow \Re$, is the relevance function for direct_child clusters (it is used for ranking direct_child clusters) ($\Re$ denotes the set of real numbers)

The ranking issues are discussed in section 5.3.3.

Definition 5.20: *Extended Query Neighborhood*

The extended query neighborhood extends the Neighborhood of a request with similar query concepts, i.e.

$$extN = (E, P, C, S, \gamma, \eta, \kappa),$$

where:

- $S := \{\Delta_i \mid \Delta_{base}(\varsigma_u) \sim \Delta_i\}$, i.e. the set of all similar clusters of $\Delta_{base}(\varsigma_u)$, and,
- $\kappa: S \rightarrow \Re$, is the relevance function for similar clusters.

Figure 5.6 presents a schematic description of the extended query neighborhood.

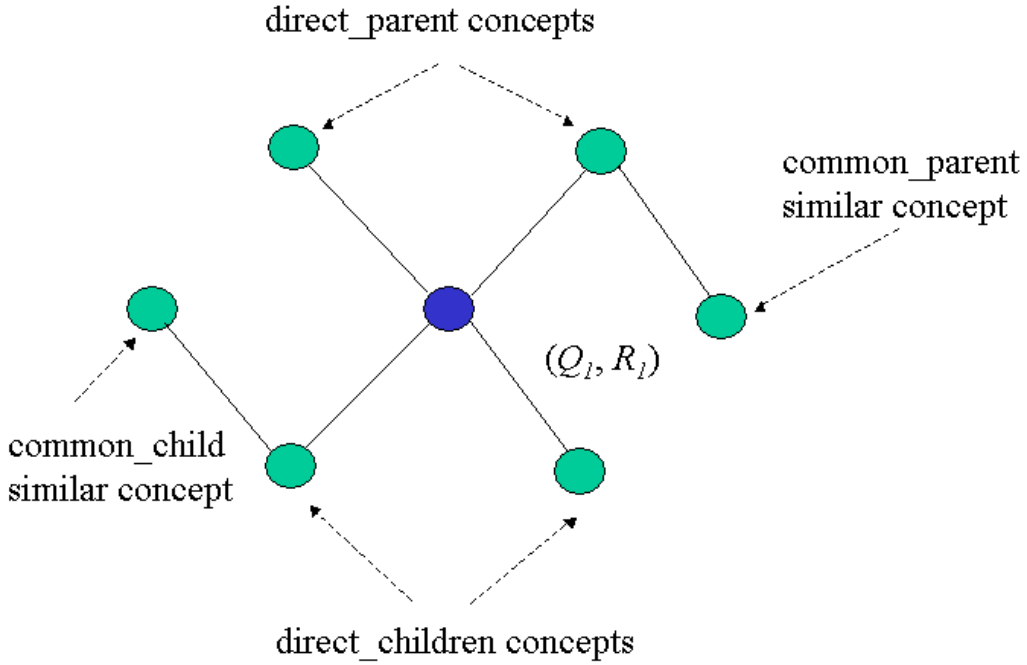


Figure 5.6: Extended query neighborhood

Figure 5.7 gives a visual presentation of the relations between users' requests regarding the product catalogue presented in Table 5.1. Each node represents a query concept, whereas each user's query is represented in its short-hand notation (see Table 5.1). Considering Figure 5.7, concept 1 is a direct_parent and concept 3 is a direct_child concept of the concept 2 (i.e. 1 and 3 are a part of the neighborhood of the query 2). Moreover, concept 3 is a specialization of the concept 2, since "DarkBlue" is in a hierarchical relation with the "BlueColor" (relation "sub").

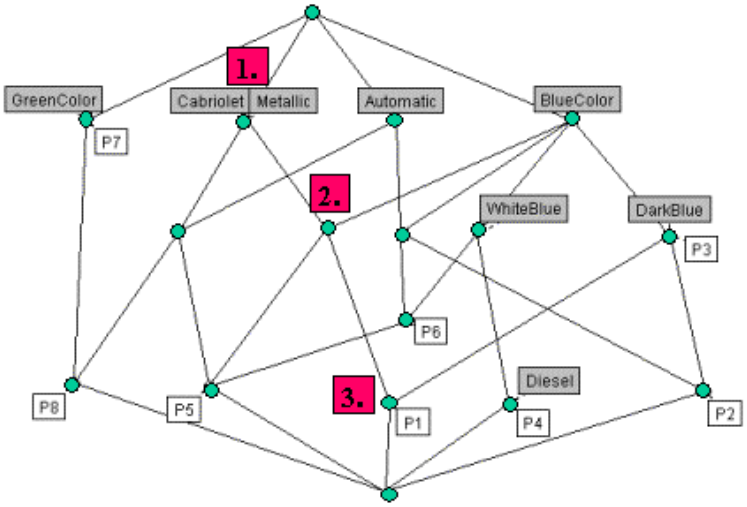


Figure 5.7: Query neighborhoods for the dataset presented in Table 5.1. Note that the nodes represented in the lattice should be read as follows: e.g. the foremost left node corresponds to the query concept ({*GreenColor*}, {*P7, P8*}). Also, the concept encompasses all resources from its children concepts and all attributes from its parent concept.

5.3.2.3 Cooperative Answering

We describe here two types of cooperative answering: *finding alternative results* and *resolving failing queries*.

5.3.2.3.1 Finding Alternative Results

An alternative result for a query is a result that does not fulfill all the constraints from the query perfectly, but has (many) suitable features from the user's point of view (e.g. usability). In the domain of e-commerce such features are often called merchant characteristics (like price of a product, warranty, time of delivery). However, the problem is how to find which characteristics can be relaxed in the user's query and which not. Our approach supports this decision by analyzing implicitly the discovered user's need. The query neighborhood is used as a snapshot of the repository where relevant alternatives can be found. Briefly, we define function *alternatives*(R_p, M, *level*), which finds all resources from the neighborhood of *query concept* in which the user selected resource R_p for which the merchant characteristic M is in the offset *level* of the value for resource R_p. The system checks all the resources contained in sibling *query concepts* which satisfy the offset of the selected merchant characteristic. The user should set offset according to his preferences. The default value is 0,1. Formally,

$$Alternatives(R_p, M, level) = \{R_{pi} \mid R_{pi} \subset \bigcup_{\forall \Delta_i,\ \Delta_i \sim \Delta_p} R_{yi} \ \wedge \left| M(R_p) - M(R_{pi}) \right| < level * M(R_p)\}, \quad (11)$$

where Δ_p is the current *Query concept* that contains product R_p selected by the user. The alternative results are ranked according to the relevance to the current elicitated user's need. Moreover, using the parameter *importance* (see section 3.2.2.3.2) our approach "knows" which attributes are relevant for the user's need and tries firstly to find alternative products that retain these attributes.

Finding alternate products can be a very useful service for the user in an e-commerce application. First, a lot of on-line buyers report the problem concerning the exhaustivity of their search when trying to buy a product: they are not sure that there is no better product for

their need. Secondly, it is possible that the user makes a constraint (e.g. the price is below 10000 EUR), which reduces the quality of other product's features very much (e.g. there is no car with an air conditioning system and a navigation system for that price). In the case that by relaxing that constraint slightly (e.g. the price is below 11000) a much better offer can be found (e.g. a car that has all the features relevant for the user plus a navigation system and the metallic color). Our service can help the user in both situations.

5.3.2.3.2 Resolving Failing Queries

As we have already mentioned in section 3.2.2.2.3, the main challenge is that it is possible to have a huge number of (independent) causes of a failure in a query. Consequently, finding all of them can require exponential time in the worst case [51]. Therefore, an efficient "repairing" system has to select only the most relevant *repairs* of a failing query. In our approach the relevance of a repair is calculated according to the loss in information that is caused by replacing/eliminating a query constraint. In that way, we ensure that the interpretation of a new (satisfiable) query is close to the user's information need expressed in the original query.

The degradation of a user's information need caused by eliminating constraint C_i from his initial (failing) request ς_x, is proportional to the ambiguity introduced by this constraint and inverse proportional to the number of query's constraints that are related to C_i:

$$InfContent(C_i, \varsigma_x) = \sum_{\forall \varsigma_x' \in AttributePairs(C_i)} SemanticAmbiguity(\varsigma_x') \cdot \frac{1}{|\{ f \mid f \in Q_x \ \wedge \ f \in Related(\varsigma_x') \}|} \quad (12)$$

where:

ς_x is a user's request in the form $\{Q_x, R_x\}$, and,

AttributePairs(ς_x) is the function that returns all attribute-value pairs from request ς_x, i.e. the set of its attributes Q_x.

Related(c) is a function that retrieves constraints that are related to c regarding the underlying ontology. This function will be discussed in the next section.

The following procedure searches for the most suitable repairs regarding a user's query. It is a specialization of the procedure we have given in section 3.2.2.2.3.

Let assume that a query ς has n constraints.

Step 1 Create n subqueries by eliminating (each) one constraint from the query. Evaluate these queries.

If one of these subqueries does not fail *Then* go to Step2.

Else Calculate the *InfContent* for each constraint and select the subquery that corresponds to (i.e. misses) the least informative constraint. Repeat Step1 starting with that subquery.

Step 2 *If* more then one subquery did not fail,

Then Calculate the *InfContent* of the missing constraint in each subquery and select that subquery which corresponds to the least informative one (it is most close to the original query), e.g. let us denote it as ς_i.

Calculate the starting concept $\Delta_{start}(\varsigma_i)$ and determine all direct_child concepts of that concept.

If one of these direct_child concepts (e.g. Δ_k) contains an attribute (constraint), that is different from $\Delta_{start}(\varsigma_i)$ but is in relation to the constraint eliminated by generating ς_i, then that cluster is a candidate for query repair. Formally,

$RepairCandidates(\varsigma, \varsigma_i) = \{x \mid x \in Q_a \wedge x \notin \varsigma \wedge \exists R_a \{Q_a, R_a\} \in DirectChildren(\Delta_{start}(\varsigma_i)) \wedge x \in Rel(y) \wedge y \in \{\varsigma \setminus \varsigma_i\}\}$.

These candidates are ranked according to the ontology-based similarity (function *Rel*, see 5.3.3.1.1) between the candidate and the eliminated constraint.

Example for Figure 5.6: Let us suppose that the user is interested in a `Cabriolet` car that has an `Automatic` gear changing system and has `"DarkBlue"` color. However, in the underlying product catalogue, there is no car which fulfills these constraints. According to the previous algorithm: ς = `?"Cabriolet" +"DarkBlue" + "Automatic"`, ς_i = `"Cabriolet" + "Automatic"` and $RepairCandidates(\varsigma, \varsigma_i)$ = {`"WhiteBlue"`} since `"WhiteBlue"` and the eliminated constraint, `"DarkBlue"`, are *related* (i.e. have a common parent). Therefore, the new query is `?"Cabriolet" +"WhiteBlue" + "Automatic"`. It conveys the initial user's need very well.

Since our approach finds only a few most relevant repairs, the complexity of the approach is low. We perform a depth-first search so that in at most n steps (n is the number of constraints) we find a non-failing subquery. In the i-th step from the beginning we perform $(n - i)$ tests in order to find the most suitable subquery. Therefore, the worst time-complexity of Step 1 is $O(n^2)$. The complexity of Step 2 is equal to the complexity of finding direct_child clusters of a query that is discussed in the next section.

5.3.3 Phase 3: Ranking of Refinements

Although the set of refinements is minimal and complete, the user has to inspect the whole list of refinements in order to find the relevant one. Since a list of refinements can be long, they should be ranked according to their relevance for a user's information need.

As we have mentioned in section 3.2.2.3 there are two dimensions of the relevance in a query refinement process:

- semantic dimension, that expresses how suitable a refinement for the initial query is, and,

- personal dimension, that evaluates how a refinement is tailored to the user's behavior.

In the next two subsections we describe these dimensions in more details.

5.3.3.1 Semantic Relevance

5.3.3.1.1 Basic Model

Regarding *Semantic* relevance we use an instantiation of the general formula given in section 3.2.2.3.1. We combine both the relevancies we have mentioned in section 5.3.1.1 and 5.3.1.2 as follows:

$SemanticRelevance(ref,Q) = a*(Confidence(Qref', Q)* Specificity(Qref',Q)*SemanticAmbiguity(Qref'))+ b* Relatedness(ref, Q)$
where:
Qref' is an extension of query Q with refinement *ref* (see section 5.3.1.2)
$Relatedness(ref, Q) = \Sigma_{\forall pair \in AttributeValue(Q)} Rel^{max}(ref, pair)$ (see Definition 3.10)
$AttributeValue(Q)$ is a set of variable-attribute pairs contained in query Q,
a and b are weighting parameters (by default: $a = b = 1$).

In other words, the relevance of refinement *ref* is greater if this refinement is more likely to be performed (*Confidence*), if it is more tailored for the original query (*Specificity*) and if the variables that are further constrained in that refinement are very ambiguous (*SemanticAmbiguity*). Moreover, the refinement should be strongly related to the initial query

(*Relatedness*). This formula ensures that the user will be provided with the refinements which are highly related to the initial query, which decrease the ambiguity of the query, and which are consequently more suitable for a user's information need. Note that this formula is analog to the formula we gave in section 4.3.3.

A parameter that should be additionally defined is the matrix Rel^{max} i.e. the relatedness matrix *Rel*.

Relatedness. The *Rel* matrix, introduced in section 3.2.2.3.1, defines the closeness between the query constraints as follows:

1) $\forall v_1,v_2 \in KB(O)\ \exists a,b\ H(a, b) \wedge l_C(a, v_1) \wedge l_C(b, v_2) \rightarrow relatedness(v_1, v_2)$

where:

H is a hierarchical relation as we have described in Definition 5.2. For example, it can be a concept hierarchy (consider Definition 2.2), and,

l_C is the concept instantiation function (see Definition 2.6).

2) $\forall v_1,v_2 \in KB(O)\ \exists r_1,r_2,c,a,b\ \ c = domain(r_1) \wedge c = domain(r_2) \wedge a = range(r_1) \wedge b = range(r_1) \wedge l_C(a, v_1) \wedge l_C(b, v_2) \wedge \exists x\ r_1(x, v_1) \wedge r_2(x, v_2) \rightarrow relatedness(v_1, v_2)$.

where[56]:

domain(*a*) is the function that retrieves the domain of constraint a, e.g. *domain*(*rel*(*par1*, *par2*)) = *par1*, according to Definition 2.3, and,

range(*a*) is the function that retrieves the range of constraint a, e.g. *range*(*rel*(*par1*, *par2*)) = *par2*.

3) $\forall v_1,v_2 \in KB(O)\ \exists r_1,r_2,c,a,b\ \ c = range(r_1) \wedge c = range(r_2) \wedge a = domain(r_1) \wedge b = domain(r_2) \wedge l_C(a, v_1) \wedge l_C(b, v_2) \wedge \exists x\ r_1(v_1, x) \wedge r_2(v_2, x) \rightarrow relatedness(v_1, v_2)$.

5.3.3.1.2 Extension of the basic model: Informativeness

The problem of determining the relevance of an attribute for a user's information need can be seen as a part of a classification problem in which for the given set of relevant information resources the importance of each attribute for the classification (relevant/non-relevant) is determined.

Indeed, in classification learning one of the crucial steps is how to determine the suitability of an attribute for the classification. We illustrate (see Figure 5.8) this step on a small example regarding the ontology represented in Table 4.1.

If we imagine the task to ask the user about only one attribute (`CarType`, `GPSType`, `Language`, `AutomaticType`, `Level`, cf. Figure 5.8) in order to suggest to him a value as much relevant as possible for the model of the car he wants buy, then the question for the value of the `GPSType` should be more suitable than the question for `AutomaticType` since for `GPS` there are 50% cars with the value "semi" (i.e. c2, c3, c5) and 50% with the value "auto". On the other hand, regarding `AutomaticType`, there are 66,6% cars with the value "type1" (i.e. c1, c2, c4, c5) and 33,3% with the value "type2". Therefore, the distribution of resources having an attribute-value plays a very important role for the determination of the relevance of that attribute.[57]

[56] For used terminology consider Definition 2.3 and Definition 2.6.

[57] A more intuitive example is the task to discover a number between 1 and 100 that the test person is thinking about. The question "is the number odd" is more likely to lead to the solution than the question "is the number prime". However, regarding the distribution of values, the most suitable question is "is the number greater than 50".

Car	CarType	GPSType	Language	AutomaticType	Level
c1	FamilyCar	auto	eng	type1	low
c2	SportsCar	semi	ger	type1	low
c3	SportsCar	semi	fra	type2	high
c4	SportsCar	auto	fra	type1	high
c5	FamilyCar	semi	fra	type1	high
c6	FamilyCar	auto	eng	type2	high

Figure 5.8: A small example regarding the calculation of the Informativeness of attributes

Therefore, a good quantitative measure of the worth of an attribute is needed. In information theory, *Entropy* is a very frequently used measure for the characterization of the (im)purity of an arbitrary collection of examples [162]. We give here the basic formulas. More details can be found in [162].

Entropy measures the impurity of *S* as follows:

$$Entropy(S) = - p_+ * \log_2 p_+ \; - p_- * \log_2 p_-,$$

where:

S is a sample of training examples,

p_+ is the proportion of positive examples in *S* regarding a class and

p_- is the proportion of negative examples in *S* regarding the given class.

Entropy(*S*) represents the expected number of bits needed to encode the class of a randomly drawn member of *S* under the optimal, shortest-length code, since according to information theory, optimal length code assigns - $\log_2 p$ bits to a message having probability *p*. Therefore, *Entropy* represents the minimum number of bits of information needed to classify an arbitrary member of *S*. Figure 5.9 illustrates this calculation.

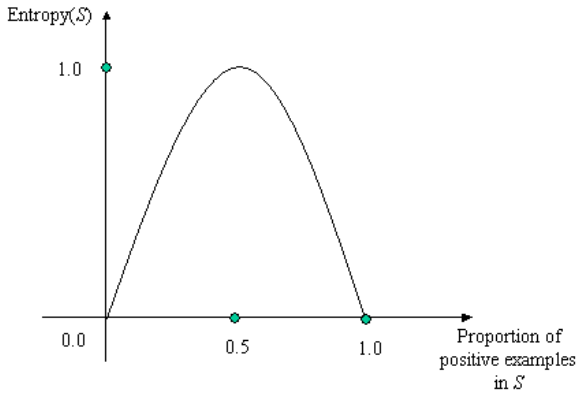


Figure 5.9: Entropy function

In the general case, for *c* classes and p_i as the proportion of class *i* in examples in *S*:

$$Entropy(S) = \Sigma_{i=1,\,c}(- p_i \log_2 (p_i))$$

If all examples belong to the same category, entropy is 0.

If the examples are equally mixed (1/c examples of each class), entropy is maximal at 1.0.

The information gain of an attribute *A*, *Gain*(*S*, *A*) is the expected reduction in entropy caused by partitioning this attribute, i.e. due to sorting *S* on *A*:

$$Gain(S, A) = Entropy(S) - \Sigma_{v \in Values(A)}(|S_v|/|S|) \cdot Entropy(S_v),$$

where S_v is the subset of S for which attribute A has value v.

The attribute A with the maximum information gain is the most appropriate attribute for asking the user for its value.

Note that for numeric features, the values have to be discretized, for instance in equi-depth buckets [162].

Information gain has the disadvantage that it prefers attributes with large number of values that split the data into small pure subsets. If one attribute has many values compared to the others, it will be selected according to *Gain* measure. *GainRatio* is an alternative metric, which resolves the problem of multivalued attributes:

$$GainRatio(S, A) = Gain(S, A)/SplitInfo(S, A), \quad (13)$$

where: $SplitInfo(S, A) = -\sum_{i=1,n} \frac{|S_i|}{|S|} * log_2(\frac{|S_i|}{|S|})$ and S_i is a subset of S in which attribute A has its ith value.

Therefore, *SplitInfo* measures the amount of information provided by an attribute that is not specific to the category. It is actually the entropy of S w.r.t the attribute A.

However, in an ontology-based product catalogue application, these measures have to be adapted/expanded in order to reflect the complexity of the domain knowledge. We give here only a very intuitive example regarding Figure 5.8: although attributes `CarType` and `GPSType` have the same values for the information gain (both have the same distribution of values across the dataset, i.e. 50% of examples belong to a value), it is clear that, from the user's information need point of view, the importance of these attributes is quite different. Indeed, *type of car* is much more relevant than the *type* characteristic of the feature `GPS`.

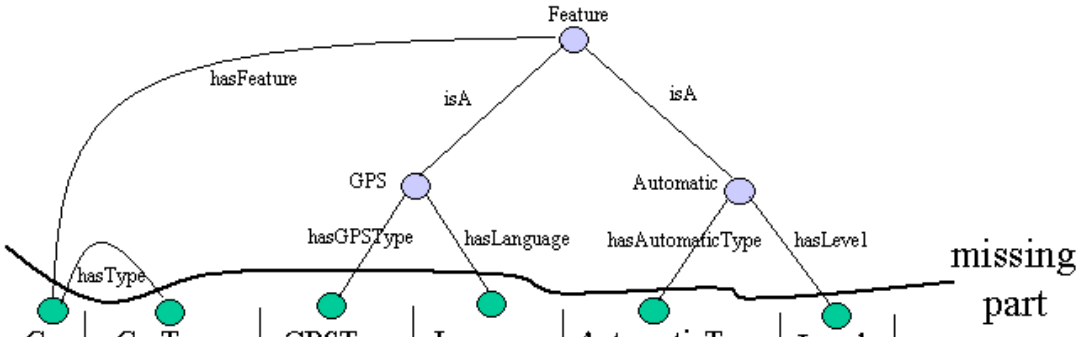
The problem lies in the vague approximation of the product-feature space in a traditional product-catalogue scenario, as we have illustrated in Figure 5.1. Indeed, if we include the underlying domain model (Figure 4.1) in the dataset presented in Figure 5.8, we get a "better feeling" about the relevance of attributes, as presented in Figure 5.10.

Therefore, the relevance of an attribute depends not only on its information content but also on its semantic meaning with regard to the underlying domain ontology.

Moreover, an ontology introduces subspaces in the feature-space since some features are semantically more related, i.e. they belong to the same concept. For example, `GPSType` and `Language` belong to the concept `GPS`. Further, it means that a more abstract level of determining the suitability of an attribute for a user's information need is required. Indeed, the user should be firstly asked if he is interested in the `GPS` at all, before being asked about the type of the `Language` in the `GPS` device. Figure 5.11 represents such an abstract view on the dataset given in Figure 5.8.

There are two problems in trying to apply directly information theory for selecting the most informative attribute in the ontology-based querying that we are discussing in the rest of this subsection:

1. how to determine the most suitable concept for the refinement, and,
2. how to determine the most suitable attribute (of that concept) for the refinement.



Car	CarType	GPSType	Language	AutomaticType	Level
c1	FamilyCar	auto	eng	type1	low
c2	SportsCar	semi	ger	type1	low
c3	SportsCar	semi	fra	type2	high
c4	SportsCar	auto	fra	type1	high
c5	FamilyCar	semi	fra	type1	high
c6	FamilyCar	auto	eng	type2	high

Figure 5.10: Ontology-enriched product dataset presented in Figure 5.8

Car	CarType	GPS	Automatic
c1	FamilyCar	g1	a1
c2	SportsCar	g2	a1
c3	SportsCar	g3	a2
c4	SportsCar	g4	a3
c5	FamilyCar	g3	a3
c6	FamilyCar	g1	a2

GPS	GPSType	Language
g1	auto	eng
g2	semi	ger
g3	semi	fra
g4	auto	fra

Automatic	AutomaticType	Level
a1	type1	low
a2	type2	high
a3	type1	high

Figure 5.11: Ontology-based interpretation of the product dataset presented in Figure 5.7

1. Since in an ontology-based query an attribute is assigned to a concept (i.e. variable), before determining the informativeness of an attribute (using entropy), the informativeness of the concept has to be determined. For example, for the query[58]:

Q = "(`CarType, SportsCar`)", (14)

the question is which of the concepts `Car`, `GPS` or `Automatic` is the most suitable for further refinement.

[58] The query is about cars that are of the type SportsCar.

In order to include the importance of an attribute from the ontology point of view, we reuse the cost-based calculation of the *Gain*, given in [148]:

$$Gain(S, A) = Gain(S, A)^2/Cost(A).$$

In section 5.3.1 we have already discussed the calculation of the semantic ambiguity of an attribute, *GeneralityAtt*(*Ax*), that can be treated as an inverse cost in the calculation of the *Gain*,

$$Gain(S, A) = Gain(S, A)^2 * GeneralityAtt(A).$$

Indeed, if the concept is more general, the importance of its further refinement with an attribute that belongs to it increases (a general constraint seems to be a too broad indicator what a user is searching for).

Finally, we define the suitability of each of these concepts in the following manner:

$$Suitability(Ax, Q) = GainRatio(l_c(Ax), Ax) \quad (15)$$

where:
Ax is a concept that can be used as a refinement of the query *Q*,
$l_c(Ax)$ is the set of instances (values) for the concept *Ax* (see Definition 2.6). Note that the set of results are abstracted using ontology as presented in the upper table in Figure 5.11,
GainRatio is defined in (13).
For example *Suitability*(`GPS`, "(`CarType, SportsCar`)") = *GainRatio*({g1, g2, g3, g4}, `GPS`), (consider Figure 5.11).

For the given problem domain we define only two classes that an example (instance) can belong to: *Relevant* and *Irrelevant*, representing a set of relevant and irrelevant examples regarding the user's need (i.e. query). The set of relevant examples for (14) is depicted by the rectangle in the first table in Figure 5.11 (all `SportsCars`).

Further, for the set *S* of examples given in Figure 5.11, *S* = {c1, c2, c3, c4, c5, c6}

Entropy(*S*) = E(3/6, 3/6)
(there are 3 relevant and 3 irrelevant, regarding user's query, cars)

Gain(*S*, `GPS`) = ((2/6)*E(0,1) + (1/6)*E(1,0) + (2/6)*E(1/2,1/2) + (1/6)*E(1,0))
(correspond to the values g1, g2, g3, g4 of `GPS` respectively)

SplitInfo(*S*, `GPS`) = ((2/6)*log(2/6) + (1/6)*log(1/6) + (2/6)*log(2/6)+ (1/6)*log(1/6))

By applying (15) to concepts `Car`, `GPS` or `Automatic` we get that concept `GPS` is the most suitable for further refinement.

2. Structuring the feature space by an ontology introduces several sets of training examples, as presented in Figure 5.11. For example, in order to determine which of the `GPS` attributes (i.e. `GPSType` or `Language`) is more suitable for splitting the example set, the set of training examples g1-g4 (left bottom table in Figure 5.11) should be analyzed. On the other hand, the relevancies are given on the set of concept identifiers, as shown in the top table in Figure 5.11. It causes the problem that a training example can be classified in several classes, in this case as relevant and non-relevant. For example, by considering the left-bottom table in Figure 5.11 it is difficult to say if the example g_3 is relevant or irrelevant, since, according to the query (14) and the set of training examples given in the top table, it can be treated as both of them. Indeed, g_3 is assigned to sportscars (car c3) as well as familycars (car c5).

Therefore, we need a notation of fuzzy entropy, which we introduce in the following manner:

For each example x_i (instance) we introduce a partial membership to a category j $p_j(x_i)$ that is calculated from set S. For example $p_{relevant}(g1)=0$, $p_{relevant}(g3)=0.5$.

Further, the calculation of *Entropy* includes this fuzzy membership:

$$FEntropy(W) = -\sum_{i \in Category} \frac{\sum_{j=1,k} p_i(w_j)}{k} * log(\frac{\sum_{j=1,k} p_i(w_j)}{k}),$$

where:
W is a set of examples, ($W = \{w_1, ..., w_k\}$),
Category is a set of all categories for the classification,
$p_i(w_j)$ is the distribution of example w_j regarding category *i,* and,
k is the number of examples that belong to the category.

For example, regarding attribute `GPSType` from Figure 5.10:

FEntropy({g2, g3}) = -((1+1/2)/2)*log(1+1/2)/2) + (0+1/2)/2)*log(0+1/2)/2).

Further,

$$Gain(S, A) = FEntropy(S) - \Sigma_{v \in Values(A)}(|S_v|/|S|) \cdot FEntropy(S_v)$$

Formula (13) is used to determine the *GainRatio* of an attribute, that is used in following definition for *Informativeness*.

Definition 5.21: *Informativeness*

Informativeness of an attribute Att_x for the query Q, is calculated as the product of the suitability of the concept that is described (constrained) using that attribute (e.g. `GPSType` for concept `GPS`) (function *ConstraintOf*(Att_x)) and the information gain of that attribute, i.e.:

$$Informativeness(Att_x,Q)=Suitability(ConstraintOf(Att_x), Q))*GainRatio(S, Att_x),$$

where S represents set of training examples relevant for the query Q.

For example, the informativeness of the attribute `GPSType` for the query (14) looks like:

$$Informativeness(\texttt{GPSType},Q)=Suitability(\texttt{GPS},Q)) *GainRatio(S, \texttt{GPSType}).$$

The *Informativeness* is used for calculating the relevance of a refinement (an attribute-value pair) by defining the relevance of the corresponding attribute.

Therefore, we can extend the basic model for the relevance given in 5.3.3.1.1 with this factor, as follows:

SemanticRelevance(*ref*,*Q*) =(1/(a + b + c))*(*a**(*Confidence*(*Qref'*, *Q*)* *Specificity*(*Qref'*,*Q*)**SemanticAmbiguity*(*Qref'*)) + *b** *Relatedness*(*ref*, *Q*) + *c** *Informativeness*(*Attribute*(*ref*),*Q*))
where c is a weighting factor (per default, $c = 1$).

5.3.3.2 Personal Relevance

Since the model for calculating Personal relevance given in section 3.2.2.3.2 is defined on the conceptual level and does not depend on the query language, it can be directly applied to the case of attribute-value querying (see Definition 3.17):

PersonalRelevance(*ref*, *Q*, Q_s) =

$\Sigma_{\forall term \in Refinement(ref)} a*Importance(term, Q, Q_s) + b*FeedbackRelevance(ref, Q_s)$

where a and b are weighting parameters (by default: $a = b = 1$).

Finally, the total relevance of a refinement *ref* for the query *Q* in the query session *Qs* is calculated as (see Definition 3.18)

$$(SemanticRelevance(ref, Q) + \lambda * PersonalRelevance(ref, Q, Q_s))/(\lambda+1)$$

where λ is a forgetfulness coefficient that models the impact on the past behavior of the user on the ranking process: $\lambda = 0$ - the past is forgotten, $\lambda < 1$ - the past carries less weight then the present, usually $\lambda = 1/2$.

5.4 A Way to Calculate the Query's Neighborhood and the Complexity of the Approach

In this section we discuss the complexity of the calculations that underlie the approach. The most expensive calculations are (i) the generation of the Query Neighborhood (Definition 5.19) since it represents a minimal and complete set of queries similar to the given one and (ii) the calculation of the content-related ambiguity parameters since they require an analysis of the underlying repository. The semantic ambiguity parameters are based on the underlying ontology and can be pre-calculated.

We discuss here how a mathematical lattice theory called Formal Concept Analysis [52] can be used for supporting expensive calculations in the presented approach. FCA is a technique derived from lattice theory that has been successfully used for various analytical purposes. We mention only the main concepts needed for an understanding of our approach. A complete description can be found in [52].

By using the formal concept analysis an annotated Information Repository *aIR=(R, O, ann*) can be treated as the formal context, where *R* represents a set of objects, *O* a set of attributes and *ann* is a binary relation between the objects and the attributes (G, M and I in the FCA terminology, respectively).

A formal concept of a formal context (G, M, I) is a pair (A, B) where:

$A \subseteq G$, $B \subseteq M$,

$A = B'$[59]$= \{g \in G \mid \forall m \in B: (g,m) \in I\}$ and

$B = A' = \{m \in M \mid \forall g \in A: (g,m) \in I\}$.

From Definition 5.9 it is clear that a *Query concept* satisfies these conditions, i.e. for a concept $\Delta = (Q_x, R_y)$ is valid:

$$R_y = Q_x' = \{ r \in R \mid \forall q \in Q_x \quad ann(r,q) \} \text{ and}$$

$$Q_x = R_y' = \{ q \in Q \mid \forall r \in R_y \quad ann(r,q) \}.$$

It means that a *Query concept* from our lattice structure corresponds to a *formal concept* in a concept lattice generated from that dataset. Moreover, the partial order between formal concepts introduced by the relation "$\leq$" as:

$$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 (\Leftrightarrow B_2 \subseteq B_1),$$

corresponds to the subsumption relation we have defined on the set of query concepts (see Definition 5.10). Therefore, in order to calculate relations and query parameters proposed in sections 5.3 efficiently, we can reuse all the results accumulated in the last twenty years in the FCA theory, as we will demonstrate below. Efficient algorithms for calculating some FCA

[59] X' is the standard notion from FCA and is not related to the usage of ' we performed earlier.

parameters, which we use in interpreting ambiguity, can be found in the systems TOSCANA and Anaconada [44].

Several algorithms have been developed for building the concept lattice of an input context (G, M, I), e.g. [22]. Usually, the efficiency of such algorithms critically depends on the number of concepts present in the lattice.

The best theoretical worst time complexity is O(|I||M|(|G| + |M|)) [87]. In practice, the behavior may significantly vary depending on a number of factors including the relative size of G and M, the size of I and the density of the context, i.e. the size of I relative to the product |G||M|. An experimental comparison is given in [69].

In the information retrieval process, due to the large number of information resources in an information repository, the full concept lattice may be constructed only for small to medium size collections, usually up to thousands of information resources. For larger test collections, such as those containing millions of documents, it is infeasible to build the whole associated concept lattice.

However, for the query refinement task we are interested only in a very small portion of the lattice, actually: a focus concept (i.e. query concept) and its neighborhood [11].

The problem of generating all the nearest neighbors of a given concept has been addressed in order to build a full lattice and to find that portion of the lattice which is centered on that concept [23]. Since our procedure for calculating a query neighborhood extends these results, we describe here the essential ideas. In the rest of the section we will present only the results for the lower neighbors (Direct Children) in detail since the determination of the upper neighborhood is a dual problem and can be solved easily by adopting the presented algorithms.

The main idea is to find suitable candidates for the lower neighborhood. The candidate children are all the nodes whose extent is more specific than the extent of the given concept. Given a concept (X, Y) it is sufficient to generate, for each attribute *m* that does not belong to Y, the set of attributes Z = Y ∪ {m} and then find

$$Z' = \{g \in G \mid (\forall m \in Z)\ gIm\} \text{ and } Z'' = \{m \in M \mid (\forall g \in Z')\ gIm\}$$

The set of children is then obtained from the set of nodes (Z', Z'') generated in this way, by selecting the most specific of them (i.e., those whose intent is not contained in the intent of some other node in the same set).

Using query concept lattice, the basic calculations for defining neighbors of Δ_a (without postprocessing the set of neighbors in order to find the most specific) can be formalized as:

$$\Delta_a <_{dir} \Delta_b \rightarrow \quad \Delta_b \in \{((R_{ya} \cup \{g\})', (R_{ya} \cup \{g\})'' \mid g \in R / R_{ya}\} \text{ and}$$

$$\Delta_b <_{dir} \Delta_a \rightarrow \quad \Delta_b \in \{((Q_{xa} \cup \{m\})'', (Q_{xa} \cup \{m\})' \mid m \in \Omega / Q_{xa}\}$$

Note that $\Delta_a = (Q_{xa}, R_{ya})$.

We have defined the following algorithm to calculate the lower neighborhood:

Find Lower Neighbors Algorithm

Input: Context (G, M, I), concept(X, Y) of that context

Output: The set of lower neighbors of (X, Y) in the concept lattice of (G, M, I)

1. lowerNeighbours:={} // the set of lower neighbors

```
2.          INCandidate:={}                    // the set of candidates
3.          for each m∈M\Y
4.                  X1:= X ∩ {m}'              // operator (') is described earlier
5.          Y1:=X1'
6.          if (X1, Y1)∉INCandidate
                    then
7.                          Add(X1, Y1) to INCandidate
8.                          count(X1, Y1):=0
                    else
9.                          count(X1, Y1):=count(X1, Y1) + 1
10.     if (|Y1| - |Y|) = count(X1, Y1) then
11.         Add (X1, Y1) to lowerNeighbours
```

The theoretical time complexity of the computation of the lower neighbors is $O(|G||M|^2)$. The time complexity of the algorithm for finding both the lower and upper neighbors is O $O(|G||M|(|G|+|M|))$.

One of the bottlenecks in the calculation is step 3 in the previous algorithm since all possible attributes are inspected. This is usually not necessary since a lot of attributes are irrelevant as extensions of a query, i.e. they are not at all related to the user's information need that is expressed in the original query.

Therefore, due to the closed world assumption we are dealing with, we can assume that the candidate attributes (element *m* from step 3.) for a given concept (Qx, Ry) belong to one of the information resources Ry. This assumption is valid due to the monotonicity of the retrieval: an extended query always returns a subset of results retrieved for the original query. Therefore, the presented algorithm is extended with step 3a as follows:

3a. $Mm = \{\cup\, t' \mid \forall\, t \in Ry\}$ // find the set of all relevant attributes

3b. for each $m \in Mm$

Since for each resource just a small subset of all possible attribute-values is defined, this improvement reduces time complexity of the calculation significantly, i.e. for this ratio. Theoretically, the worst case remains $O(|G||M|^2)$ as given above. However, for the query refinement task practical improvements lie between 30% -50 %.

According to Definition 5.11 the sibling relation can be calculated by using a set of direct_parents and direct_children so that the efficient methods from FCA can be used in this case as well.

As we have already demonstrated, all the content-related ambiguity parameters can be calculated by using the neighborhood of the given query concept, so that the complexity of that calculation is the same as presented above.

5.5 Evaluation: SMART – An Ontology-based Shop Assistant

As a proof of the research concepts presented in this chapter, we have developed a prototype shop assistant system called SMART, SeMAnticaly enRiched e-shop assistanT, as an extension of the Librarian Agent framework we have presented in Chapter 3. In this section we are giving an overview of the procedural model of the system and presenting an evaluation study that shows the advantages of applying ontologies in the e-commerce domain. We perform two types of evaluations: (i) A conceptual one where we compare functionalities (i.e.

potential) provided by our approach and found in traditional systems and (ii) an experimental one, where we compare the performance of our system with the performance of a traditional one. Moreover, we discuss how ontologies lead to a completely new, more human-like, interpretation of an e-shopping assistant.

5.5.1 Procedural Schema

The model presented in section 5.3 is the conceptual backbone of the SMART shop assistant. In order to explain the advantages of our approach, in this section we are presenting an informal workflow that explains how SMART supports search through a product catalogue.Figure 5.12 illustrates the algorithm.

1. Input:

The user selects some attributes he finds important for search and enters their values. An attribute-value query $Q_1, \ldots, Q_d$ is formed.

2. Finding Starting Cluster:

The most general concept, $\Delta_a = \{Q_{xa}, R_{ya}\}$, which contains the attribute-value queries $Q_1,...,Q_d$ (i.e. $Q_{xa} \subseteq \{Q_1,...,Q_d\}$) from the user's request is found. There is only one such a cluster (see Corollary 5.3) – it is the base concept for a set of attribute-value pairs from the user's query.

3. Presenting Query Concept Info:

The user is presented with the information about the "quality" of the concept which corresponds to the selected attributes. Particularly, the list of products and the following set of semantic and content ambiguity parameters are presented to the user: *Importance*, *Ambiguity, Max_equal_request*, *Min_equal_request* and *DisjointTerms* (see 5.3.1.1 and 5.3.1.2.3).

4. Presenting Cluster Neighborhood:

The list of query concepts from *DirectChildren*(Δ_a) and *DirectParents*(Δ_a), with the ambiguity information of each concept, is presented to the user. Actually, the Neighborhood of the user's request is presented to the user (see Definition 5.19).

5. Tracking User's Interaction:

The system logs all activities the user has made, e.g. the user has viewed more info about a product P_p. That information is used as the user relevance feedback [108]. There are two types of actions that affect the workflow:

Refinement -	Continue Flow (Step 6)
Select Product for Buying -	**GOTO** Step 9

6. Refinement:

The user continues his search by selecting one of the neighborhood refinements.

Moreover, the user can enter an attribute as valid on its own (independently of the presented refinements). It is called *free-querying*. In that case the algorithm jumps back to Step 2.

7. Finding Next Concept:

If the user selects one of the direct_child refinements, the new starting concept is already found, i.e. no additional calculation is required. The new concept is treated as the current concept: Δ_a. **GOTO** Step 3.

In the case of free-querying the most common concept which contains selected attribute-value pairs should be found. **GOTO** Step 2.

8. Repeat steps until product, P_b is selected for buying in Step 5.

9. Estimating the quality of the solution and finding the alternatives for buying: The user is presented with an estimation how general for his needs (expressed through the set of selected attributes) is the query concept in which he selected the product for buying. This information is derived from the *Ambiguity* parameters of the current concept. In order to evaluate his choice, the user can inspect the most suitable alternative products regarding a merchant attribute and the level of the similarity, as we have explained in section 5.3.2.2. **GOTO** Step 3.

10. Repeat until finishing the buying.

5.5.2 Soft Navigation in SMART

In this section we are presenting how the model introduced in section 5.3 can be used for the efficient search through a product catalogue, i.e. how the requirements derived from motivating example (see section 5.2) are fulfilled in the SMART approach.

First requirement: To enable querying the product catalogue so that the right (only one) starting point for an information need can be found automatically.

According to Corollary 5.3 (see the comment on it) for each set of attributes $\{Q_1,...,Q_d\} \in Q_{xa}$ there is a *Query concept* Δa which is the unique common parent for all other *Query concepts* which subsume that set in their Δ_attributes. It means that for any query posted by the user there is exactly one concept that serves as its "starting point" – its *base* concept. That concept, $\Delta_a = \{Q_{xa}, R_{ya}\}$, fulfills the following condition:

$$Q_{xa} \subset \{Q_1,...,Q_d\} \wedge \neg\exists \Delta_i, \Delta_a <_{dir} \Delta_i, Q_{xi} \subset \{Q_1,...,Q_d\}.$$

In the case that the user's query returns no result, the SMART shop assistant uses the method described in section 5.3.2.2 for resolving failing queries.

Second requirement: Informativeness of the search – the system should provide as many pieces of information as possible which are relevant for the search and can be implied from the search context.

The ambiguity parameters which are presented to the user are already discussed in section 5.3.1. We emphasize here only two pieces of information which our approach derives from the hierarchical ordering of concepts: for the set of attributes selected by the user, SMART determines (i) the set of all attributes that are automatically fulfilled (*Max_equal_request* set) and (ii) the set of attributes which cannot be satisfied at all (*DisjointTerms*). Since an average user is not familiar with a product catalogue in advance and cannot predict which attributes are common or disjoint, we find this information as very valuable for fast navigation through the catalogue. Regarding the motivating example, if the user selects attribute `Diesel`, then the system informs him that such a car cannot be `Cabriolet`. Moreover, regarding informativeness in order to make the user confident with the suggested refinements, the system provides explanations why a particular refinement is highly ranked.

Third requirement: The system has to provide a minimal set of relevant refinements, ranked regarding their relevance for a user's need.

In Lemma 5.2 we have already shown that the difference between the constraints that can be found in the union of direct children concepts of the given user's request and the constraints found in the user's request represents a minimal and complete set of refinements. However, the main goal is to enable the user to navigate efficiently through a product dataset, i.e. to ensure that the user always has a minimal interface (i.e. set of refinements) to all relevant **products**. The following theorem proves this property.

Theorem 5.6: A set of direct_child elements of a concept is the minimal set of the expansions of the query (attributes) which correspond to that concept. It means that none of these expansions is redundant (i.e. cannot be excluded without at the same time preventing the finding of a relevant product) and each relevant product can be reached by one of the following expansions (i.e. no refinement has to be added in order to find a relevant product). The relevance of a product is determined with the current set of attributes, i.e. considering concept Δ_a a product is relevant if it contains all attributes from A_{xa}.

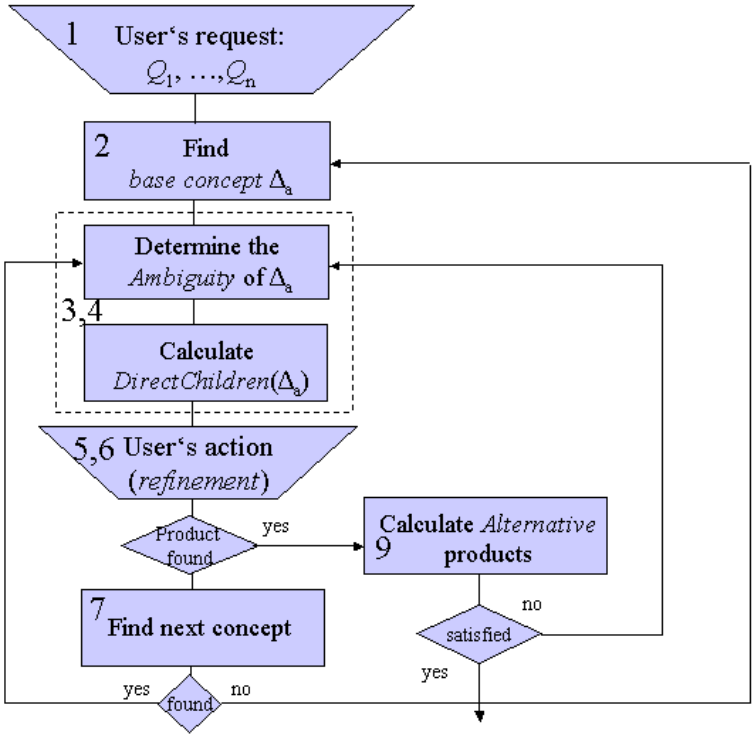


Figure 5.12: Procedural schema of the SMART approach

Proof:

1) *None of the expansions is redundant*:

Let us assume that for concept Δ_a there is a direct_child concept, e.g. Δ_{ac}, which can be excluded but all relevant products can still be reached using expansions to the remaining direct_child concepts. Formally:

$$\forall R_p \in R_{yac} \wedge \Delta_{ac} <_{dir} \Delta_a \rightarrow \exists \Delta_{ai} \neq \Delta_{ac} \wedge R_p \in R_{yai} \wedge \Delta_{ai} <_{dir} \Delta_a .$$

Let us assume that concepts Δ_{a1}, Δ_{a2}, ..., Δ_{ak} (all direct_child concepts of Δ_a) cover all products contained in Δ_{ac}. It means that Δ_attribute of Δ_{ac} has to subsume all attributes found in Δ_attributes of concepts Δ_{a1}, Δ_{a2}, ..., Δ_{ak}, i.e. $Q_{xac} \subset Q_{xa1}, Q_{xac} \subset Q_{xa2} ..., Q_{xac} \subset Q_{xak}$. According to Definition 5.10 Δ_{ac} is a direct_child concept of concepts Δ_{a1}, Δ_{a2}, ..., Δ_{ak}. Since Δ_{a1}, Δ_{a2}, ..., Δ_{ak} are direct_child concepts of Δ_a, it follows that between Δ_{ac} and Δ_a there is/are another concept(s). However, according to Definition 5.10, there cannot be a concept between a concept and its direct_child concept. Therefore Δ_{ac} cannot be a direct_child of Δ_a. It follows that the initial assumption was wrong.

2) *None of the relevant expansions is missing*:

Let us assume that for the concept Δ_a there is a product P_p which is not contained in one of direct_child concepts of Δ_a, but that product can be found in a subsumed concept e.g. Δ_{ac}.

Formally: $\exists R_p \in R_{ya}, \exists \Delta_{ac}, \Delta_{ac} < \Delta_a \wedge \neg \Delta_{ac} <_{dir} \Delta_a \wedge R_p \in R_{yac} \wedge R_p \notin \bigcup_{\forall \Delta_i, \Delta_i <_{dir} \Delta_a} R_{yi}$. Since Δ_{ac} is not a direct_child of Δ_a there exists a direct_parent of Δ_{ac}, Δ_{acp}, which is subsumed by Δ_a, i.e. $\Delta_{acp} < \Delta_a$. According to Definition 5.10, the concept contains all the products of its direct_child concepts so that it follows that product R_p is contained in the concept Δ_{acp}, i.e. $R_p \in R_{yacp}$. If Δ_{acp} is a direct_child of the Δ_a, then the initial assumption is wrong. Otherwise, the previous discussion can be repeated for Δ_{acp}: there is a direct_parent of Δ_{acp} which contains R_p and it is subsumed by Δ_a etc. After a finite number of repetition, the given direct_parent has to be a direct_child of Δ_a. Since it contains R_p, a contradiction with the initial assumption arises. □

Fourth requirement: The system should discover (anticipate) a user's need by analyzing his behavior.

This is one of the main strengths of our approach, elaborated in detail in section 5.3.3.

Fifth requirement: The system should provide an estimation of the quality of a selected product.

SMART provides two ways of examining the product selected for buying: regarding (i) user's needs and (ii) alternative products. See Step 9 in the previous section for details.

Since the user tends to inspect all possible alternatives for buying, our system has to prove that the set of alternatives it finds is the minimal according to a given criterion. We have introduced the matching between attributes specified as relevant for the user and the attributes of a product as the validation criterion. According to this criterion, a set of alternatives found in *Alternatives*(P_b, *M*, *level*) is a minimal one, since the number of common attributes between a given product and any product from sibling concepts (which are treated in the function *Alternatives*) is greater than the number of common attributes between that product and a product which is not in a direct (parent, child and sibling) relation with the given product. This is a property inherited from the lattice structure of the product database.

Using Figure 5.7 we can briefly reinterpret the search process presented in the motivating example. The user makes the query (`Cabriolet`) which moves him to the node (i.e. concept) ({`Cabriolet`, `Metallic`}, {*P1, P5, P8*}) depicted as 1 in Figure 5.7. This is the starting concept for the user's need. By analyzing the neighborhood of that concept, SMART "recognizes" that all `Cabriolet`-cars are `Metallic` and that a `Cabriolet` cannot be a `Diesel`. Next, the system provides only two refinement possibilities to the user: `Automatic` and `BlueColor` since these correspond to only two direct_child concepts of the current concept. By choosing the refinement `BlueColor` the user is moved to the concept ({`Cabriolet`, `Metallic`, `BlueColor`}, {*P1, P5*}) depicted as 2 in Figure 5.7. He selects product *P1* for buying. SMART informs the user that the current concept might be too general for this decision since he has required the `BlueColor` and there are several "`subtypes`" of that attributes. Moreover, by inspecting siblings of the concept that corresponds to product *P1* (depicted as 3 in Figure 5.7), the system suggests buying product *P5* as an alternative, since *P5* is very similar to *P1* (feature `WhiteBlue` instead `DarkBlue`) but it has one feature more (`Automatic`) for a slightly higher price (only 10%). Due to simplicity the values for products' prices are omitted in Figure 5.7.

5.5.3 Conceptual Evaluation: Knowledge Level of an Shop Assistant

So far we have explained the logical and procedural part of an ontology-based product catalogue application. However, a product catalogue application replaces a human shop assistant, so that there is another level on which the discussion about the SMART approach can be performed: the knowledge level. This is the topic of the following section.

5.5.3.1 Introduction

A lot of effort has been spent in the last decade in replicating real-world shopping experience in e-commerce sites. Particularly, a number of models have been proposed to describe a real-world customer-buying process [59], [117] and several recommendation strategies have been developed to represent the background knowledge and experience of a shop assistant [7]. Most of them introduce some plausible heuristics about a user's behavior (e.g. a user should select the most preferable product among several alternatives) and in an intensive software engineering process they implement such a solution. However, the buying process can be considered as a decision-making process in which the user "searches", regarding a certain problem (formulated as an inquiry/query), for a solution (represented as a relevant product). Therefore, one can abstract particular e-commerce scenarios and consider the on-line shopping problem on the *knowledge level* [82]. In such a view the goal of problem solving is not just to select one of the possible actions, but rather to construct a model of part of the world that allows the problem-solver to conclude eventually that its goals have been achieved [154]. In other words, regarding the shopping domain, instead of interviewing an experienced shop assistant about concrete questions which should be given to a customer in a particular shopping situation (for a particular request of a user) [12] one should define a model regarding the goals that a shop assistant would achieve by asking such questions, i.e. why he would make a question. A more abstract (knowledgeable) model means more flexible, extendable problem solving. For example, the above-mentioned models of customer-buying behavior [59], [117] define (only) a heuristic/workflow how a shop assistant resolves a request in a particular situation. Such an approach does not support either an easy evaluation (since the process is not defined on an abstract level) or the maintenance (in the case that a new method appears) of the system.

In this section we present an approach to modeling the behavior of an on-line shop assistant on the knowledge level, using generic problem solving methods (PSM) [111], [161]. Particularly, from the knowledge level point of view the problem-solving used in the e-shopping domain might be seen as a method that searches for a set of products relevant for a set of features (properties) given by the user and that refines that set (i.e. rules out some products) by introducing new features that are relevant for the user. It corresponds to the *cover and differentiate* PSM (in the rest of the text abbreviated as *c&d*) [75], very successfully applied in various diagnosis and classification tasks, in which the *cover-task* takes a set of symptoms and produces a set of explanations that seem applicable, whereas the *differentiate-task* tries to rule out elements of this set. More precisely, we use *c&d* as the model which underlies the problem solving process of a shop assistant, whereas the knowledge used for problem-solving in *c&d* is used as a guideline for the process of eliciting users' needs. In this way we define the goal an on-line shop assistant would achieve by asking the user some questions, which enables us to generate more useful questions. Consequently, we can (i) model such a conversation with the user so that a minimal and complete set of questions will be generated and (ii) design new support processes for on-line shopping (e.g. comparing a product with several alternate products) which will help an on-line shop assistant "achieve" his goal in a more efficient manner. Moreover, we can show that some popular buying models can be easily interpreted by using our generic problem-solving method.

5.5.3.2 Modeling E-shopping Problem-solving on the Knowledge Level

5.5.3.2.1 Knowledge Level

The knowledge level [82] provides the means of 'rationalizing' the behavior of a system from the standpoint of an external observer. This observer treats the system as a 'black box' but maintains that it acts 'as if' it possesses certain knowledge about the world and uses this knowledge in a perfectly rational way in order to reach its goals (*principle of rationality*[60]). There are three different perspectives on the knowledge level: a *domain model* and *a task model,* that talk in a precise and systematic way about domain knowledge and goals of the system, respectively and a *problem-solving method* that relates the task and domain models in order to accomplish goals. In the meantime, a lot of such generic inference patterns, called *problem-solving methods (PSM*) [161], have been identified [74]: cover and differentiate for diagnosis, propose and revise for parametric design, skeletal-plan-refinement for hierarchical planning, etc.

5.5.3.2.2 E -shopping as a Problem-solving

As has already been mentioned in previous section, a buying process can be considered as a problem solving process, in which a shop assistant tries to resolve a user's "problem" regarding shopping, i.e. the user selects several product's features for which the shop assistant tries to find the most relevant ones. In order to resolve the problem in the most efficient manner, the shop assistant narrows down the search space by eliciting some additional knowledge (e.g. more relevant features of a product) from the user, by asking him directly or by observing his behavior. Obviously, the quality of that elicitation process affects the quality of the buying process, e.g. a customer may feel distrust towards the shop assistant who posts irrelevant questions [117]. However, in most on-line shop systems the communication is initiated either by an anthropomorphic shopping agent for the given domain [12], who *transfer*s his knowledge into the set of questions, or by an automatic analysis of product data, e.g. using some data-mining algorithms like ID3 [144]. The drawbacks of the first approach are well-known in the knowledge acquisition community, namely, a highly expensive hard-coding of the expert knowledge disables its reusability in similar situations. In the second case the expert background knowledge is completely missing, so that the flexibility of the solution is lost.

Fortunately, from the knowledge level point of view the solution for an effective communication seems to be very simple: if we understand the rationale *why* knowledge is needed, we can understand *what* knowledge should be elicited. Indeed, by analyzing existing e-shop portals and their "conversations" with customers, we have extracted the common behavior (rationale) of different shop assistants, which we formulate in a simplified form like:

"In each action an e-shop assistant performs, he tries to eliminate as many as possible irrelevant products that are offered to a user. "

Consequently, in the elicitation process (e.g. by questioning) the shop assistant tries to acquire as much as possible "eliminating" knowledge - the knowledge that can be used for efficient elimination of products irrelevant for the current user. Finally, we can abstract this behavior to a generic inference pattern, which (1) for a set of symptoms proposes a set of explanations and then (2) seek information to eliminate irrelevant explanations. By analyzing available libraries of PSMs [161], we have found a very suitable inference pattern - *cover and differentiate* PSM, developed for supporting diagnosis task [74].

[60] The agent will select an action that according to its knowledge leads to the achievement of one of his goals.

5.5.3.2.3 Cover-and-differentiate PSM

c&d is a role limiting method that implements a form of heuristic classification [31]. We present here the most important details regarding our research. A complete description of *c&d* method can be found in [74]. The method resolves a problem by first proposing candidates that will cover or explain the symptoms or complaints specified by the user and then by seeking the information that will differentiate the candidates. The search method is divided into a covering and a differentiate task. These tasks are abstractly defined as follows: the cover task takes a set of features (symptoms) and produces a set of candidates (explanations) that seem applicable; the differentiate task tries to rule out elements of this set. In order to achieve these goals each task uses corresponding knowledge (covering or differentiating). Two constraints guide and confine the search in the *c&d* method towards its goal of producing a consistent explanation [74]: *exhaustivity* - if a symptom has at least one potential explanation, the final diagnosis must include at least one of the potential explanations and *exclusivity* - each symptom can be explained only by one final explanation.

In Figure 5.13 we are presenting the structural decomposition of the method. This *c&d* process is iterative since some of the symptoms used to differentiate the candidates often need to be explained in a subsequent step.

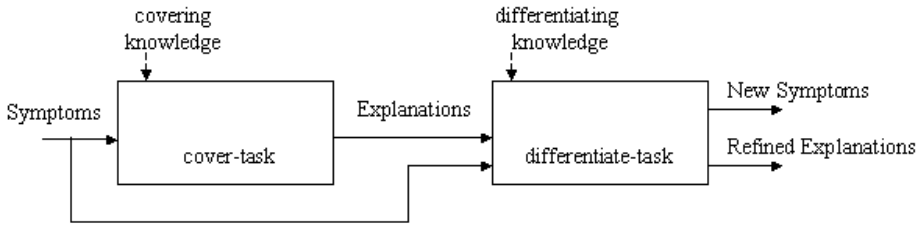


Figure 5.13: Structure of the *c&d* PSM

The domain knowledge used in the method should be represented as a causal network, which is the main source of the covering knowledge.

5.5.3.2.4 Using c&d for E-shop Problem-solving

If we consider a buying scenario as the process in which a shop assistant tries to find suitable candidates (products) which satisfy (explain) a set of features the user prefers, the mapping to *c&d* domain is straightforward: features are symptoms and products are explanations. Moreover, it has been shown that *c&d* can be applied to case-based assessments [42], where the task is to find the most specific decision in a decision taxonomy that fits a specific case. This scenario is very similar to a buying scenario. Both of the above-mentioned constraints for *c&d* search are valid in the e-shopping domain, since the user wants to find at least one relevant product and each relevant product has to explain all the requested features.

Therefore, from a structural point of view, we can use the generic *c&d* inference pattern as the problem-solving method in a shopping portal. However, the main problem is how to define covering and differentiating knowledge (relevant for *c&d*) in the e-shopping problem solving.

Covering Knowledge

First of all, *c&d* requires a causal network as the covering knowledge, which is not a preferred knowledge representation paradigm in the shopping domain. Note that we post minimal requirements on the structure of the domain knowledge in an e-shop scenario and try to prepare it for the *c&d* –based processing.

Basically, the causality in *c&d* can be expressed as:

If *cover*(S, E) then *cover*(S', E'),

where
cover(S, E) means that a set of symptoms (S) can be explained with set of explanations (E),
S and S' are sets of symptoms,
E and E' are sets of explanations and $S \subseteq S'$ and $E \subseteq E'$.
In this case we consider that symptoms $\{S' \setminus S\}$ are caused by symptoms S.

According to the *c&d* interpretation of the e-shopping scenario, this condition can be rewritten as:

If *cover*(F, P) then *cover*(F', P'),

where F, F' are sets of features and P, P' are sets of products and $F \subseteq F'$ and $P' \subseteq P$ and *cover*(F, P) means that all products from P have all features from F. In such a causal case we consider that features $\{F' \setminus F\}$ are caused by features F. For the example given in Table 5.1, one can conclude that feature `BlueColor` is caused by feature `Metallic`.

Therefore, we need a partial order between feature-products pairs in order to "simulate" a causal network for a whole product dataset. Comparing other e-shop applications this is a very important difference – we organize products in a causal network in the first place, whereas most of other approaches use a decision tree topology.

Since we have already introduced the relation subsumption as a partial order (see Definition 5.10) in the product-feature space, the lattice of *Query concepts* can be used as the required causal network. In other words, it defines covering knowledge for *c&d* problem-solving in an e-shopping domain. This covering knowledge is used in the cover-task of the problem-solving in order to find a set of relevant explanations (covering). Theoretically, the entire causal knowledge can be calculated, but since we are considering a step-by-step refinement of a user's request, the neighborhood (direct children) of a concept is required only. The complexity of this task was discussed in section 5.4.

Differentiating Knowledge
The differentiate-task uses differentiating knowledge in order to eliminate some covering explanations generated in the cover-task. Obviously, the more explanations are eliminated by using a differentiating knowledge, the greater usability of that differentiating knowledge is. In an ideal case, after applying this knowledge, only one explanation should remain. In the *c&d* method such knowledge is elicited from experts in a highly interactive process of refining the knowledge base [74].

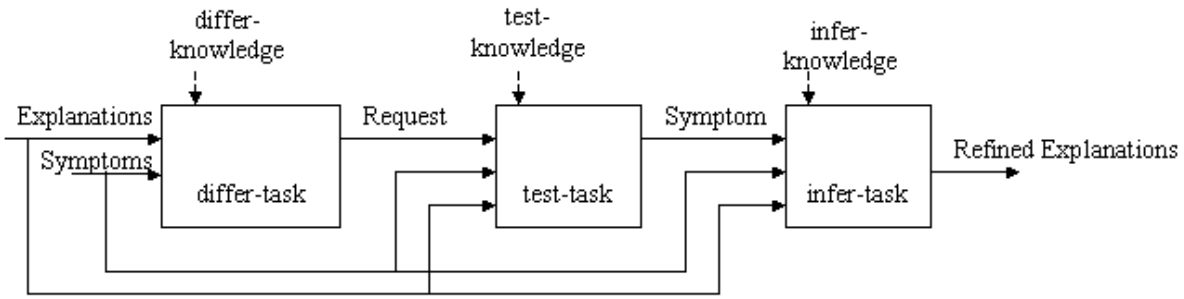


Figure 5.14: Decomposition of the differentiate task of *c&d* PSM

Following the analogy to the *c&d* method, the differentiate-task in the e-shopping scenario consists of three subtasks represented in Figure 5.14. The main problem is how to obtain the knowledge employed in these subtasks, which we discuss in next three subsections.

a. Differ knowledge

This subtask finds out which new symptoms (features) should be tested (e.g. is the value of the symptom X equal Y). In a problem-solving system this task can be seen as the crucial one: a system seems to be more intelligent if it makes as few as possible tests/questions in order to conclude something.

From the knowledge level point of view, the realization of this subtask should be driven by the "principle of rationality" of the agent – in the case of a shop agent, the goal is to eliminate as many irrelevant candidates as possible. In other words, the selection of the features for testing should be done in such a way that the results of tests would enable maximal restriction of the search space. It is clear that the selection of features for testing has to be *fair* (*complete)* – each candidate (relevant product) has a chance to "survive". Moreover, it is clear that the number of tests should be *minimal* since we can theoretically ask for the availability of each feature. In the e-shop domain the principle of the minimality is important due to the need to develop a buyer's information need incrementally, that is, in the so-called step-by-step manner. Briefly, on-line buyers often have a vague idea of what to buy (due to the unfamiliarity with the content of the product database) or how to express their request (due to the unfamiliarity with the used vocabulary). In that case the shop assistant should ask only for features that do not imply another feature not yet considered by the user. For example, if all the cars that are "metallic" have blue color (by assuming that there are "non-metallic" blue cars) and the user has not set any of these features yet, then a "non-minimal" test is to ask: "Should the car be metallic?" An explanation is that in the case that the user does not want blue color, the question about "metallic" feature is irrelevant. Moreover, since the shop agent tries to eliminate as many products as possible by using a test, his asking whether the car should be blue (instead of the previous question) will cover all "blue-colored" cars, including all metallic cars.

A solution for this problem is to ask an expert which features should be tested in which situation. However, it can be a very expensive process and cannot guarantee a fair and minimal testing.

Another possibility is to reuse knowledge employed in the differ-task in the *c&d* method in e-shopping problem-solving. Basically, the *c&d* differ-knowledge compares *competing explanations* for a symptom directly, i.e. it compares each two explanations which cover the same set of symptoms. Therefore, for a set of initial symptoms *c&d* differ-knowledge calculates the set of competing explanations which cover all the symptoms and then tries to eliminate some of them by asking for their availability [96]. We use the same idea: for a set of product's features Q_{init} and the set of relevant products R_{init} we calculate the set of possible competing features Q_{com}, i.e. the set of features which can be found in relevant products. Using the notation introduced in previous sections, we can formalize this calculation as

$$Q_{com} = \Big(\bigcup_{a, \Delta_a <_{dir} \Delta_l} R_a \Big) \setminus R_{init}, \qquad (16)$$

where Δ_l is the largest equivalent node (*len*) for the node (Q_{init}, R_{init}).

The largest equivalent node is calculated analog to the maximal equivalent request (see section 5.3.1.2). For a node Δ_{init} it is $\Delta_l = (Q_{xl}, R_{yl})$ such that: $Q_{xl} = \{ \bigcup Q_{xi} \mid \forall i \quad \Delta_i = \Delta_l \}$.

Moreover, we have already shown in section 5.5.2 that this strategy selects a *minimal* set of features that ensures *complete* testing, i.e. each relevant and no irrelevant (regarding current user's need) product can be obtained/tested.

b. Test knowledge

The task of this knowledge is to perform tests on the features selected in the differ-task. In the simplest case the user is asked for the values of selected features. However, there are several methods that can be used to decide which products to eliminate (see next section). For example, several products can be compared, or the features of a product should be compared with each other. In all these tasks the test knowledge is used in order to perform tests in the most appropriate manner (from the user's point of view).

Note that our approach is based on testing feature-value pairs. It means that we do not ask the user to choose between all values of a feature in the case that some of these values depend on the values of some other features.

For example, if the color of a car can be red, green and blue, but all red cars are metallic, the system asks the user to select only between:
- the green and blue color and
- whether the car should be metallic.

In that way we ensure minimal testing. Note that most of the other methods for generating product catalogues do not treat such dependencies between features.

c. Infer knowledge

This kind of knowledge enables the interpretation of test results in order to eliminate irrelevant results. Since the infer-task "understands" what the goal of the whole method is, it can interpret the test results differently depending on which strategy for elimination is selected. See next section for more details.

5.5.3.2.5 Extending Original c*&*d *PSM*

The subtasks in a generic task can be seen as placeholders for various strategies that can be used for the same type of reasoning (i.e. for the same type of inputs and outputs and the same goal). In the previous section we have described the "default" strategy for reasoning about how to eliminate irrelevant products, which we borrow from the *c&d* method and its applications for diagnosis tasks. Since we assume (claim) that we have found a generic reasoning pattern for "diagnosis" in the e-shopping domain, we should be able to map existing solutions for e-shopping into this pattern. Moreover, we should be able to process – by using the proposed framework - all the available information relevant for problem-solving in this domain. Finally, since we consider the problem from the knowledge level, we should be able to develop new services for more efficient e-shopping, which would be based on the proposed generic pattern. In the following we discuss these three challenges.

Mapping Existing Solutions
A popular strategy for eliciting the buyer's preferences in a shopping scenario is to propose to the user several products and to ask him to select the most suitable or unsuitable one [117]. The buyer should also explain his decision. According to his answers, the shop assistant selects new products for recommendation and repeats the procedure.

It is very easy to recognize the generic *cover and differentiate* pattern in this scenario: in the differ task several products are selected for comparison, in the test task they are compared and in the infer task these results are used for generating new relevant products that cover the user's need.

However, the problem we have considered in the "traditional" solutions is that they are performed on the symbolic level, so that they implement only a particular heuristic of a shop

assistant, like [117]: in the case of proposing *exactly* three candidate products to the user, the first candidate is the product closest to the centre point[61] of the set of relevant products, the second one is positioned most distantly from the centre point and the third one is the product positioned most distantly from the second one.

It is clear that this heuristic cannot be directly reused in the case of an arbitrary number of products that should be proposed to the user. The drawback is that the knowledge level analysis is missing: since it is not clear how the knowledge about comparison will be used, it is difficult to elicit this knowledge, i.e. to define a heuristic which products to compare in an arbitrary case.

On the other side, if we consider the comparison between products as a differ task in *c&d*, then the goal is defined very clearly: select products for comparison in such a way that as much as possible information relevant for eliminating products can be obtained. In that case the selection of the *n*-products that should be proposed to the user is very simple regarding the lattice structure in the product-feature space: for the most general[62] node which covers features the user has selected as relevant in the current search session, we should rank the directly subsumed nodes (i.e. direct_child nodes) according to the number of features and select the first n top ranked nodes. From each of these nodes we should pick-up one product, which is used as a representative of the entire node. More formally, if we assume that the user has selected the features Q_{init} as relevant ones (i.e. his query is mapped into the node (Q_{init}, R_{init})), then the following function calculates the *k* clusters, which product should be presented to the user from:

$$how\_to\_user((Q_{init}, R_{init}), k) = \{n_x | n_x \in max(\{(Q_t, R_t) | (Q_t, R_t) <_{dir} len((Q_{init}, R_{init}))\}, |Q_t|, k)\},$$

where:
$max(A, c, b)$ is the function that retrieves *b*-top ranked elements from set *A* regarding the condition *c*,
$|S|$ depicts the number of elements in set *S*, and,
len is Largest equivalent node, see (16).

Interpreting Relevant Information

In addition to using explicit user feedback about the relevance of the retrieved results, the buyer's preferences can be derived from the so-called implicit relevance feedback [108], where each action of the buyer is captured as relevant information about his preferences. For example, if the buyer selects a product in order to get more information about it, it is assumed that this product is somehow relevant for him. By using the *c&d* method the information about the user's feedback can be interpreted as a type of differentiating knowledge, so-called *enabling connection knowledge* [74]. Briefly, for each connection (i.e. a link in the feature-product lattice) a symptom (feature) can be assigned, which defines an external condition when a causal connection can be established. Its task is to role out (eliminate) all connections which are not enabled in a particular situation. In that way our approach can take into account the user's profile by generating questions to that user, but, of course, only if such a profile exists.

The main problem is how to guess which features of that product the buyer actually likes and which are not so important for the buyer. However, similarly to the discussion in the previous subsection, the analysis on the knowledge level enables us to benefit maximally from the user's feedback: we interpret the feedback information so that as many products as possible will be eliminated. We give here only an example. Let us assume that for a user's query three

[61] Here we do not discuss how these parameters are calculated.
[62] It means that this node is not subsumed by another node which covers the given set of features.

results are retrieved: p1 that has features (red, metallic, automatic), p2 with (green, metallic, non-automatic) and p3 with (blue, non-metallic, automatic) and the user selects the first one, i.e. p1. Using the e-shop agent "principle of rationality", the most relevant feature for the user's need is the first one (color) since in that case only one product remains relevant (p1). In the case of assuming that the user has selected the first product while the feature metallic is the most relevant feature for him, two products can be considered as relevant (p1 and p2). A similar discussion is valid for the case of assuming that the feature automatic is the most relevant one. Moreover, another cognitive explanation why the color is the most relevant one is also possible: by choosing the red color, the user contrasted that value with two different ones. For the features metallic (and automatic) there is only one different value.

Note that the general model of the implicit relevance feedback we have introduced in Chapter 3 corresponds to this interpretation. Therefore, that model can be explained on the knowledge level of the buyer.

Developing New Services
The knowledge level analysis provides a consistent and abstract view of problem solving. In the case of e-shopping that view looks like: Do something so that as many irrelevant products as possible are eliminated. If we apply that principle to the situation when the user has selected a product as a very relevant one (e.g. he is ready to buy it), we get a new service which can help the user buy the most relevant product. Briefly, when the user selects a product to buy, we can try to eliminate that product by comparing it to some other products that seem very interesting/relevant for the user in order to be sure that the selected product is exactly the one that the user needs. The interpretation using the *c&d* method looks like: Relevant candidates are found in the differ task, the comparison is part of the test task and introducing a new product for buying is done in the infer task.

Obviously, this service can be implemented as one of the types of cooperative answering: *Finding alternative results* (section 5.3.2.3.1). Therefore, we have shown that finding alternative results is not just an effective procedure for presenting results to the user, but, moreover, a very useful service that corresponds very well to the knowledge level of the shopping process.

5.5.4 Experimental Evaluation

Since the goal of our research is to model the query refinement support for the ontology-based information retrieval (like on the Semantic Web), our evaluation study concerns the comparison in the effectiveness (regarding search) between a traditional web search system and a system, which uses advanced ontology-based technologies (so called semantic portals).

In order to prove the advantages of using our approach in a real-world domain, we performed an evaluation study to compare our approach with a traditional approach. In fact, we compare search for relevant products (holiday trips) in two tourist portals, traditional (http://demo.dwm.uni-hildesheim.de/ecommerce/) and semantic one (based on SMART approach), that are based on the same product data. However, the semantic portal uses an ontology as the backbone for the search process and applies the logic-based query refinement approach as the method for navigating through the portal. We avoid here the details about the structures of portals. We note only that the traditional portal is a very advanced one – it combines a traditional decision tree approach with case-based reasoning techniques. The dataset contains a snapshot of offers acquired from a travel agency (about 15000 offers, each described with about 15 properties, like the place, price, etc.). The semantic portal is based on an ontology that was derived from the given database schema using the approach described in [127].

We have performed two evaluation studies that are described below.

5.5.4.1 First evaluation study

We have compared *formal* properties of the portals, i.e. their query refinement subsystems, according to the factors introduced in section 4.2. Note that these requirements are based on the general discussion about step-by-step refinement we have given in Chapter 3 and are therefore valid for any step-by-step query refinement approach.

The experiment should show how these formal properties are fulfilled in both portals. More clearly, we want to check the comprehensiveness on the refinement (regarding properties 1. – 4. section 4.2) provided by the portal to the user in a navigation step. We assume that the user makes a general query and then tries to specify it in several navigation steps, which forms a navigation session. We compare the navigation structure for 100 queries posted against both the portals. The queries have been selected by 10 participants (10 queries per candidate) who have actually performed a search for a holiday trip. The participants are graduate students and no additional instructions have been given to them. In order to ensure a fair comparison, half of the tasks (search), for each participant, have been performed on each of the portals.

Table 5.3: Results of the first evaluation study

Method for navigation	Completeness of results in a step (average)	Soundness of results in a step (average)	Completeness of questions in a step (average)	Minimality of questions (average)
Traditional	85%	100%	50%	60%
Our approach	100%	100%	100%	100%

We have done a post festum analysis of the support for the query refinement provided by a portal, by measuring parameters 1. - 4. for each navigation step in each navigation session. It means that we have "traversed" off-line all the navigation paths given by the users and calculated (per hand) parameters in each step. Table 5.3 summarizes the results. In order to simplify calculation (but without effecting the generality/validity of the experiment) we have made a relative measurement, i.e. we have put the parameters of a portal in the context of another. For example, for the parameter 3, we have compared the set of questions provided by both the portals. 100% means that this portal for that parameter includes all values produced by the other portal. We have not included any significant time-delay in calculating/presenting refinements in the semantic portal, comparing to the traditional portal.

Discussion:

We give only the average value for all query sessions for all parameters because we just want to illustrate the nature of results without going into any detailed analysis. It is clear that in each refinement step the user can expect only relevant results (column 3: Soundness of results = 100%) even in the "traditional" portal. However, in the "traditional" portal some of the relevant results are missing (column 1: about 15%), due to problems in modeling hierarchically organized data in a standard relational database. On the other side, in the semantic portal the transitivity axiom (from the ontology) ensures the completeness of the answers. Moreover, 50% of the relevant refinements that should be provided to the user (questions) are missing (column 4) as can be expected since the refinement structure in a traditional portal is generated in an ad-hoc manner. Better results for the completeness of the results can be explained by the fact that some products are placed in several refinements, so the user can find a product using several refinements. Finally, ad-hoc generation of refinements in traditional portals disables fine-tuning of the user's needs in a step-by-step manner in about 40% of cases (column 5), i.e. in 40% of refinements the user is provided with

sub-optimal recommendations for a refinement (e.g. the user is asked for a value of a product's feature which can be derived from other features).

5.5.4.2 Second evaluation study

The second experiment is a classical user-driven study in which we want to prove the user's acceptance of the proposed approach. The set up of the experiment is similar to the previous one. In order to avoid bias and to generalize results we have taken another group of 10 participants. They have got 10 queries which they should resolve in one of the portals. Again, each of the portals has been used in half of the tasks for each user. We have measured the *length of a navigation path*, the *duration of a navigation session* and the *confidence* of the user in the selected product. The confidence describes the user's sureness that his decision is the best possible one (i.e. that there is no better product for his need). It is measured on the scale 1 - 4, with 4 meaning maximal confidence.

Note that the user interfaces (GUI) in the portals are different, but the structure of the information provided by the portals is the same (a list of refinements and a list of products). We find that "syntax" differences in the GUI do not influence (strongly) the results of the experiment. The main difference is the "semantics" of the refinement process.

The results of the second experiment are presented in Table 5.4.

Table 5.4: Results of the second evaluation study (Note: SD means standard deviation)

Method for navigation	Duration of a session (in sec.)	Length of a path	Confidence
Our approach	69,31 (SD = 12,283)	5,63 (SD = 1,125)	3.32 (SD = 0,469)
Traditional	75,30 (SD = 10,689)	5,77 (SD = 1,355)	2.76 (SD = 0,698)

Discussion:

Our approach requires less time (column 2), fewer navigation steps for a task (column 3) and the users are more confident in a product selected in the proposed portal (column 4). This can be interpreted as the better quality of the questions provided by our approach, i.e. the Logic-based approach asks questions but they are carefully selected (more useful for a user), since the user does not spend much time in a navigation step. Finally, the Logic-based approach covers a large part of the search space with such questions, so that the user is very confident with a selected product, i.e. he has feeling that lots of alternatives are taken into account in the navigation process. This is a very important feature for recommender applications – the user should have trust in the recommendation process. To see if these differences can be considered statistically significant we have performed a paired t-test for each measure. The test has revealed no effect of the method on the *length of a path* ($p = 0{,}1873$). It has, though, revealed the superiority of our approach with respect to search time ($p < 0.0001$) and the user's confidence ($p < 0.001$).

Our experiment has shown that a model-based comparison of two query refinement methods is possible – one can "reason" about the characteristics of two methods and not only to "measure statistically" their effectiveness, which always implies the problem of the users, settings, etc. Moreover, using the logic-based model of the query refinement one can predict what the user can expect from the system. For example, for a refinement setting, it is possible that the user "looses" some refinements in order to speed-up the search process. Finally, this predictability enables us to (easily) generate query refinement systems tailored to the user's needs.

Moreover, by comparing a traditional portal with an "ideal" approach, we can locate the

weakest points in it and try to improve the performance. For example, regarding the last evaluation study, we have found out that a big problem in the considered traditional portal arises when the user does not select the destination for a trip, since the offers are finely clustered regarding to the property "destination" and the traditional portal does not take such clustering into account. In that case the user misses a lot of offers that can be very relevant for his request.

5.6 Related Work

Product Catalogue

Regarding search in product catalogues the most similar approach is presented in [101]. It is an extension of a mediator architecture that supports the relaxation or tightening of query constraints when no or too many results are retrieved from the catalogue. The query language is a type of Boolean query suitable for the (web) form based querying against product catalogues. The query tightening is enabled when the cardinality of the resulted set has reached a predefined threshold and it is realized by selecting the most informative, not yet constrained product features. The information content of a feature is defined by measuring its entropy. However, this approach does not treat the problem of query refinement on an ontology-based level.

An important success factor for on-line shops is the topology of the product catalogue, which can be organized in a (i) hierarchical, (ii) product-feature oriented or (iii) shopper's need oriented fashion [45]. Even in the case of optimal cases so that the topology of a product catalogue corresponds to the decision tree generated from that product database[1] [144], or that the questions for gathering the user's preferences are elicited from domain experts [125], the user can find a relevant product efficiently only when he knows exactly what he is searching for.

However, there are shopping portals that recommend to the user some products according to his preferences, e.g. the products, which "similar" users are interested in, or the products that are similar to the products that the user was interested in in previous purchases, i.e. collaborative- and content- based filtering, respectively [7]. Although these recommendations look like the suggestions a real shop assistant give to a regular customer, the problem is that such recommendation requires the explicitisation of users' preferences, as well as a large amount of previous similar cases in order to provide useful recommendations, which leads to the so-called cold-start problem. Nevertheless, the reluctance of users to provide explicit information about their preferences decreases the efficiency of a recommender system. Consequently, a non-personalized user can benefit from a recommender very rarely. Even in the case that the user's profile is well defined it is possible that the "current" search requires more specialized information than that provided by the long-term user profile, i.e. the system has to discover the "current" need of the user, i.e. his need in the current search session [34]. Moreover, in some product areas (such as appliances or home electronic equipment) the buyer's interests typically only relate to a single buying session, and the profile information is typically not re-used in later sessions. This is so-called *ephemeral personalization* [113].

Finally, the existing on-line shopping systems weakly support the direct comparison between the products the user can be interested in [125]. In addition to the problem that a list of compared products can be very large, a table-based comparison can be of limited help if sorting the list according to a single product feature (such as price) is not sufficient because the buyer's soft preferences consider more than one feature [124]. Moreover, such a comparison is not integrated in the search process, which means that the user has to break the

[1] In that case the navigation paths are optimised regarding the number of questions which are asked by the system.

navigation process in order to find alternative products on his own [72]. The main problem is to anticipate which products, except the products which are explicitly returned for his request, can be interesting for the user. By knowing the user's "hidden" preferences in actual search, the system can find alternatively relevant products, which do not match the user's request perfectly.

The previous discussion leads us to the conclusion that the crucial point in an on-line shopping system is to discover as much as possible information about what the user is actually searching for. Indeed, most of the above-mentioned drawbacks result from relying only on the request (query) the user has made, since it is less informative than the original user's need [110]. Once the user's real information need is discovered, for example from his behavior in the portal, the on-line shopping system can support the user in finding products (resources) which are highly relevant for his need. For example, the products that can serve as alternatives to the selected product can be found easily.

As we have already explained in the motivating example, our approach resolves exactly these issues.

Information Need

Here we discuss some models which deal with the weakly defined information need.

The concept of a weakly-defined information need can be found in some earlier works on information needs [9]. In [95] this type of search is called opportunistic information seeking, characterized by the fact "that users are initially unable to articulate their needs clearly".

Another interesting model is the ostensive model of developing information needs [21] which recognizes ".... that information-needs are developing, inaccessible, and observable only retrospectively through their external physical effects". The use of the path information as a surrogate of the information-need (in fact, as the physical manifestation of the development of the information need) is formalized in this model.

Our approach can be explained through information foraging theory [29] which analyses how user strategies and technologies for information seeking, gathering, and consumption are adapted to the flux of information in the environment.

Last, but not least, our approach can be treated as an extension of the interactive query expansion [43] in which expansion terms, extracted from retrieved documents, are somehow grouped/clustered and suggested to the user in a suitable visual metaphor.

However, none of these approaches (except the interactive query refinement) is focused on querying an information repository, but on guiding the users in the ill-defined navigation process. Consequently, the ordering of the search space and several concepts proposed in our approach, like query ambiguity and query neighborhood, are completely missing in other approaches.

Query Refinement

The existing systems for query refinement are based on expanding a query with new terms, which are obtained by analyzing the results retrieved for the initial query, for example by finding the most common terms in the retrieved documents [5]. In the so-called local-context analysis expansion terms are extracted from the relevant documents, whereas the relevance can be determined by user feedback [108] or by assuming the top-ranked documents to be relevant (pseudo-relevance feedback). Since in a real search context users are usually reluctant to provide such relevance feedback information, the pseudo-relevance feedback is commonly accepted in the IR. This is a very suitable heuristic in the case that the initial query perfectly expresses the user‘s information need, which is not always true, especially in the case of using a controlled vocabulary for forming user queries. However, when the user posts

a non-ideal (for the refinement) initial query, the extensions of the query can diverge from the user information need more than the initial query did, i.e. the query refinements lead to more inappropriate results than the initial query. Moreover, a query can reflect several information needs thus, providing only one expansion seems to be a non-optimal solution [62]. Therefore, the user needs several possibilities for the refinement of his query. From these possibilities he selects the most suitable one for his information need. In other words, the possible refinements should be clustered with respect to the user's information needs. These clusters define the neighborhood of a query – each elementary change of the query converts the query in one of the queries from its neighborhood. Consequently, one of these neighbors corresponds to the information need of a concrete user. Hence, in the refinement process the user needs a system that looks at the queries around the initial query in order to decide how, if at all, to change the initial query. In other words, a map of the query neighborhood should be provided to the user.

Another problem we find in the existing systems is that they present only a modified list of results for a refinement, without assessing the refinement process. However, the user should be provided with the details about what is achieved (refined) in the refinement process. For example, a query can be extended with more than 50 terms [36], which leads to a more focused search, but disables the possibility to check in reality whether the extended query terms correspond to the user's information need [43]. A trivial case is that the list of extension terms contains a term which the user might find inappropriate. Or the user might be interested in the percentage of the common results in the initial query and the refinement in order to reason about the quality of the refinement. Therefore, the user needs a method that enables the characterization of the query refinement process regarding his information needs, e.g. how far that refinement is from the initial query or how ambiguous it is. In the case of several possible refinements provided to the users that we have mentioned above, a compass for moving through the query neighborhood is needed.

Regarding query expansion word sense disambiguation (WSD) 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 [104]. In [155], the set of experiments using lexical relations from WordNet for the query expansion is described. However, the conclusion is that more careful strategies for applying background knowledge in the query expansion process are needed. Indeed, the "trivial" approach to extend a query term with all its synonyms taken from WordNet usually results in a significant decrease in the precision of the retrieval system.

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. *Query expansion* aims at helping the user make a better query, i.e. it attempts to improve retrieval effectiveness by replacing or adding extra terms to an initial query. *Interactive query expansion* supports such an expansion task by suggesting candidate expansion terms to the user, usually based on hyper-index [19] or concept-hierarchies [70] automatically constructed from the document repository. In [159] the model of the query-document space is used for an interactive query expansion. *Recommendation systems* [7] try to recommend the items similar to those a given user has liked in the past (content-based recommendation), or try to identify the users whose tastes are similar to those of the given user, and recommend items they have liked (collaborative recommendation). Personalized web agents, e.g. WebWatcher [64] track 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 a rich domain model for the refinement of a query, i.e. the reasons for doing a refinement are not based on a deep understanding of the structure of a query, or a deep exploration 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.

Query by navigation [19] can be seen as a very similar concept to query refinement approach. However, we extend the navigation concept with the notion of query ambiguity. Moreover, since the existing approaches for query by navigation treat each query the user posts separately, the variety of the analyses, especially regarding the ambiguity of the query and the query equivalence, is missing. Finally, the query transformations regarding the query's siblings are not explicitly supported.

Query Ambiguity

The determination of ambiguity in a query as well as the sources of such an ambiguity, is the prerequisite for an efficient search for information. Although some work has recently been done in quantifying the query ambiguity based on the language model of the knowledge repository [94] the IR research community has not explored the problem of using a rich domain model in modeling the query process.

The most interesting approach to quantifying query ambiguity is proposed in [35]. The authors have developed a measure of a query with respect to a collection of documents with the aim of quantifying the query's ambiguity with respect to those documents. This measure, the clarity score, is the relative entropy between a query language model (i.e. a language model representing the collection word usage that is associated with the query) and the corresponding collection language model. However, our approach is based on using a controlled vocabulary, i.e. an ontology, for annotating documents, so that the intended usage of a term is already defined with the structure of the ontology

Some very important results in the query analysis can be found in the deductive database community [28], namely semantic query optimization. That approach, although revolutionary for using the domain knowledge for an optimal compilation of the queries, does not consider the ambiguity of the query regarding the user's information need at all.

5.7 Conclusion

In this chapter we have instantiated the librarian agent query refinement process for the case of the attribute-value ontology-based retrieval, in which the query is structured in the form of ontology-based attribute-value pairs. We have defined a comprehensive model for the lattice-based organization of such a query space. The formal concept analysis methods are applied for the efficient calculation of the query space.

We have performed a very extensive conceptual and experimental evaluation in order to demonstrate (i) the advanced functionalities that can be achieved by semantic modeling of a product catalog application domain and (ii) the increase in a user's confidence in a system that performs reasonably (i.e. according to some rational expectations). Indeed, the traditional product catalog systems do not succeed in persuading a user that the recommendations they generate are the most reasonable one, by not providing any reasonable "evidence" for that. On the other hand, our logic-based approach ensures that the generated refinements are optimal according to some reasonable criteria, like non-redundancy and usefulness of a refinement.

6 Conceptual Keyword-Based Query Refinement

6.1 Motivation

From a user's point of view, the simplest way to describe his information need is by using just a set of keywords that are related to what he is searching for. Indeed, such a query formulation does not put forward any assumptions about the syntax of the query nor about the user's background knowledge, e. g. to be familiar with the terminology used in the domain of interest. On the other hand, a set of keywords is a very natural way to easily express an information need. Ease of expression usually is the crucial requirement in the real-world search. For example, the information need we have mentioned in section 5.1, expressed as an attribute-value query:

$$\{(\texttt{CarType}, \texttt{"SportsCar"}), (\texttt{ColorValue}, \texttt{"BlueColor"})\}$$

can be represented as the following keyword-based query:

$$(\texttt{SportsCar}, \texttt{BlueColor}).$$

It is clear that a part of the query semantics is lost and that several interpretations of such a query are possible. For instance, one can assume that the user is interested in the sports car whose model is called "BlueColor", or in a general information repository; likewise, one could assume that Mr. Blue is the designer of that sports car. Therefore, a query can be mapped into several information needs whenever there is no other evidence that can be of any help in this disambiguation process, i.e. all these assumptions should to be taken into account.

Such a line of reasoning is the crucial reason for the inefficiency of traditional (web) search engines. Figure 6.1 illustrates the problems which arise when the user makes a very short query ("jaguar") and the retrieval system returns all the information resources that are related to any context in which the symbol "jaguar" appears, like jaguar-as-car, jaguar-as-animal and jaguar-as-operating system.

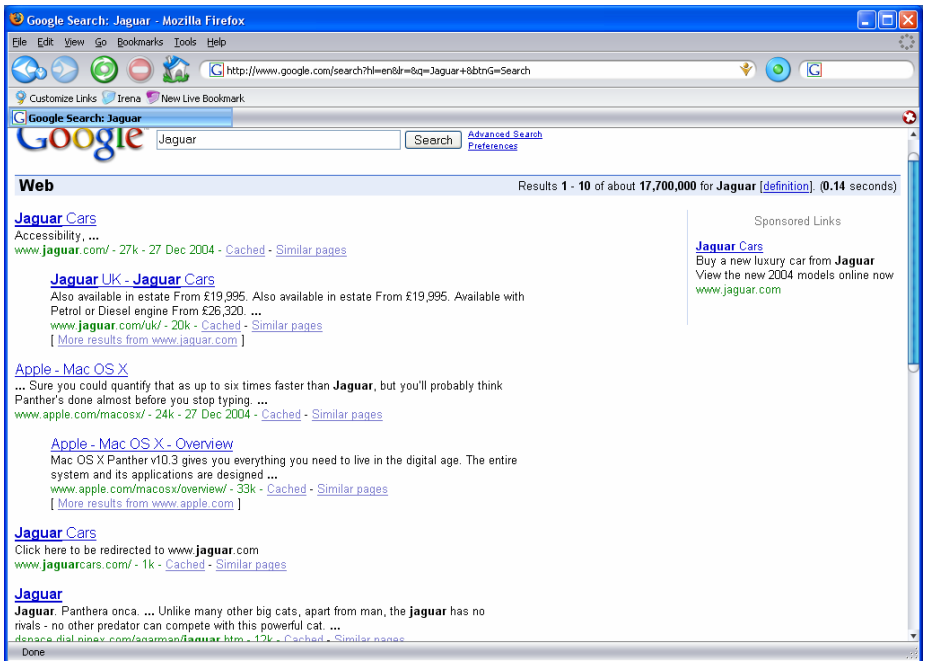


Figure 6.1: Results from Google web search engine for the query "jaguar". First, second and fifth results are about cars, third and fourth are about computers and sixth is about animals

It is clear that on the level of symbols (keyword-based query) it is not possible to determine uniquely the information need that drives a query. What is missing is a conceptualization of the set of symbols in a meaningful structure that can constrain the search space. Such a process can be explained by the meaning triangle [88], a structure for defining the meaning of words whose origin lies in semiotic research. The meaning triangle illustrates the fact that, although a word as a symbol cannot completely capture the essence of a reference (= concept) or of a referent (= thing), there is a correspondence between them. The relationship between a word and a thing is indirect. The correct linkage can only be accomplished when an interpreter processes the word invoking a corresponding concept and establishing the proper linkage between the concept and its respective thing in the world. In other words, the general context of communication is described by the meaning triangle that defines an interaction between symbols, concepts and things of the world. It assumes the existence of an additional conceptual layer, which the set of symbols is mapped into, in order to define the meaning of a query.

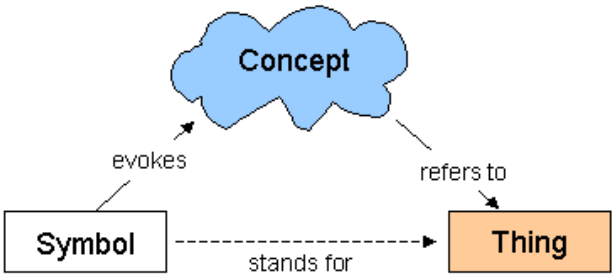


Figure 6.2: Meaning triangle

Figure 6.3 illustrates the role of the meaning triangle (see Figure 6.2) in the above example regarding the query "jaguar". In this example, it filters mappings to the things in the real world whose conceptualization is not related to cars.

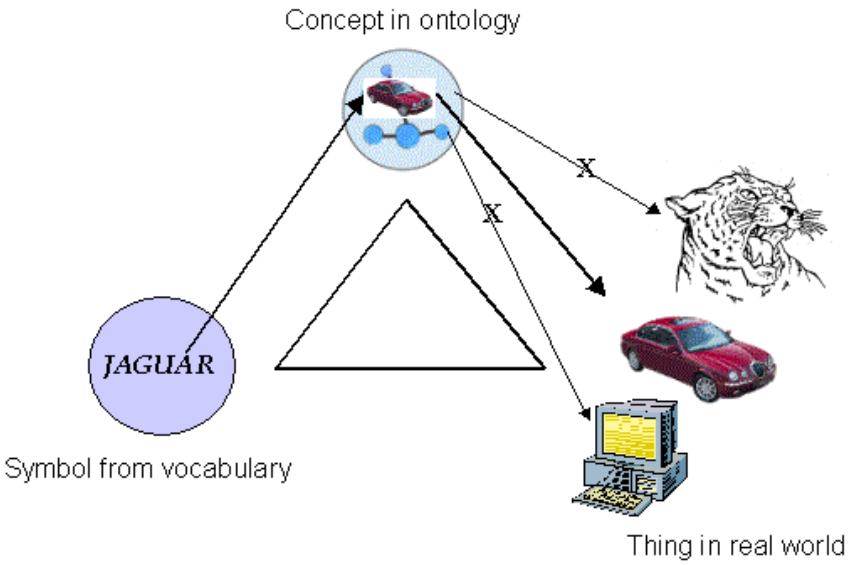


Figure 6.3: The role of the meaning triangle in the disambiguation of the query "Jaguar"

However, in the context of keyword-based queries the assumptions about the existence of predefined conceptual structures are too strong, and, therefore, this conceptualization process should be performed (semi-) automatically. Two crucial problems arise in this automation process, namely:

1) How to construct the conceptual/meaning layer, which a query should be mapped into:

   Since it is not realistic to assume that for each domain a fine-grained ontology will be available, there should be a possibility to develop automatically a conceptualization that is needed for this disambiguation. Note that a full description of the ontology is not needed, but rather only that part related to the query disambiguation process.

2) How to perform that mapping (query -> conceptual layer) automatically:

   As already mentioned, users are reluctant to provide additional information about their queries. Consequently, the meaning of a keyword-based query should be built automatically, as far as possible.

In the ontology-based querying, an ontology plays the role of the meaning layer, i.e. the user formulates the query by using the background knowledge about the domain (i.e. the domain ontology). In other words, by using the keyword "jaguar" only, it is not possible to discover what the user is searching for, but it is possible to determine which interpretations are possible and to ask the user (explicitly or implicitly) which of them is the most suitable one. Note that the usage of a predefined domain ontology enables direct questioning about the most suitable interpretation. For example, for the query "jaguar" and the ontology in which jaguar can be a car, an animal or a plane, the system can ask the user to choose one of these interpretations. However, the existence of a full ontology is a too strong assumption for keyword-based search systems. Moreover, although the role of corporate taxonomies in the enterprise search process has increased recently, the taxonomies cannot be treated as fixed structures. Indeed, in order to reflect the dynamics of businesses, the information (semantics) that emerges from the retrieval process has to be taken into account in the query conceptualization task.

Secondly, the retrieval system has to respond to a user's query in such a manner that the user gets a picture of what the most plausible interpretations of his query are and he can select one of them. In the case of the query "jaguar", the system should recognize different contexts in which the term "jaguar" appears, like "jaguar" and "food", or "jaguar" and "design", or "jaguar" and "capacity" which could help in distinguishing various meanings of the word "jaguar". It is clear that such an explanation cannot directly point to the car, animal and plane, but could indicate one of these meanings. For example, the user can avoid the context "jaguar" and "food" very easily if he is not interested in the interpretation of the word "jaguar" in the context of animals.

Moreover, such an approach opens a palette of new possibilities for a disambiguation process, since it contributes directly to clarifying the meaning of the query term. For example, the additional descriptions for the term "jaguar" can be combined with a general, incomplete ontology in order to get a bigger picture regarding the meaning of the query. For example, by considering the extensions "jaguar" + "food" and "jaguar" + "legs", one can conclude that the query "jaguar" is about an animal, if there is a domain ontology in which the characteristics "food" and "legs" are assigned to the concept "animal", although the term "jaguar" is not present. Therefore, the system can conclude that "jaguar" could be an animal and this, as has been mentioned above, is a crucial task for query refinement. Finally, such an approach can be used for the incremental population of an ontology.

Therefore, both conceptualization problems mentioned above are related to the process of emerging semantics from an information retrieval process, which can be done in a query refinement process. Indeed, a query refinement process can be seen as a process of the conceptualization of a query since the refinements (i.e. added terms) represent the part of the domain that is strongly related to the query and, consequently, can be used for the disambiguation purposes. Secondly, we consider the query refinement as a process in which the meaning of a query is refined (discovered) that corresponds actually to the mapping of

symbols into a conceptual layer. This type of refinement we will call *conceptual query refinement*, since it resolves the refinement problem on the level of the meaning of a query (i.e. on the conceptual level): in each refinement step a direct clarification of the meaning of the query, not just a constraining of its scope, is performed.

Note, that the existing methods for the refinement of keyword-based queries seem to be inadequate for such a conceptualization [21], since they usually return a long list of refinements that is not semantically related to the query. Indeed, recent experimental studies of an interactive query refinement have shown that only one third of the terms derived from the document relevance feedback are identified by users as useful for refining their queries [43]. In other words, users are overloaded with refinement information, similarly to an overload with search results[63] in an information retrieval task. The main cause of the problem is a weak definition of the notion of relevance: a term is considered relevant for the refinement if it appears frequently in relevant documents. Obviously, such a definition covers only the syntactical level of relevance since the context in which these terms appear (i.e. their meaning) is not at all treated. Consequently, a lot of irrelevant refinements for the particular user's information need will be generated since it is possible that a candidate term appears very frequently in the relevant documents, but not in the context of the query terms. In other words, the existing query refinement methods take only the document-centered view on the problem, without taking into account query characteristics, i.e. semantic relationships between refinement terms and the user's need (i.e. the user's query).

Thus, the main challenge in the conceptual query refinement process is to discover the right meaning of a query which is usually extracted from the information contained/produced in a retrieval process. This is the topic of the next section.

6.2 Emerging Semantics from an Information Retrieval Process

As we have already elaborated in Chapter 2, relevance is one of the crucial questions in an information retrieval process. In the traditional systems relevance is usually interpreted as an estimation of statistical information, i.e. the results that fulfill some requirements from the statistical point of view are treated as relevant (e.g. if the frequency of a term in a document is greater than a threshold, then this document is relevant for the query containing that term).

It is clear that such information cannot be used for a semantic-based processing of a query, as required by a conceptual query refinement approach. Indeed, statistics shows that something (e.g. a keyword) appears more frequently than other artifacts. On the other hand, semantics tells us that an artifact (e.g. a relation between two terms) has a meaning.. It follows that the artifacts that represent a semantic unit (i.e. meaningful artifacts) appear more frequently than meaningless ones. Moreover, the greater the frequency of the artifact's appearance, the greater the probability that that artifact has a meaning is. Therefore, there is some semantics in statistical information. However, the problem still remains concerning which kind of semantics can be extracted from statistics, or, more precisely, which statistical information can be used for deriving semantics.

In an ontology, semantics is defined through some structures:

> a) a *relation* between two concepts means that in a concrete domain there is a clear connection (dependency) between instances which belong to them, and,
>
> b) a *hierarchical* relation between the two concepts means that the scope of the subconcept is subsumed by the scope of the superconcept, i.e. that the subconcept always appears in the context of its superconcept.

[63] Paradoxically, the query refinement should help the user in resolving an information overload.

By using these two structures we can define two meaningful patterns for discovering semantics from statistical information, namely:

> a) if two terms appear frequently in a "dependency" relation in the list of results, then it can be concluded that there is an ontology relation between them in the given domain. The "dependency" relation means that they clearly affect one another,
>
> b) if two terms appear frequently as one logical (interpretation) unit (e.g. "personalized workflow"), then that unit can be treated as a subconcept of the term that conveys meaning (usually the most right term of the unit).

Therefore, if we include some linguistic structures (e.g. noun phrases) in processing statistic information about the results of a query, we can derive some semantic information from those results. In section 6.4.1.1 we present how this is done.

In order to carry it out, we have to consider in the refinement process not only the results retrieved for a query, but also the query itself. In other words, we see a query and the corresponding set of documents as two (equally) relevant factors for the calculation of relevance. It is important to note that the traditional query refinement methods do not directly take into account the effects of the initial query since they process just a list of relevant results (i.e. they assume that this list incorporates the effect of the query). Moreover, by neglecting the direct influence of the given query, the processing of a document is independent of the context for which it was found relevant. In other words, a document retrieved for two different queries would generate the same refinement terms in both refinement processes that would not be the best solution in the general case. Recently, several approaches which take into account a user's query in the refinement process have been proposed [20]. However, they are based on the naive statistical processing of the resulting documents by extracting the terms that co-occur with the query terms. In that way the terms which appear in the context of the query term can be identified, but the effect between these terms cannot be semantically qualified. The problem is that the query is just taken as a bag of words, so that the potential refinements of the query are not semantic, but rather its pure syntactical extensions/refinements. Consequently, the role which each of the refinements plays for the refinement of the query's meaning is not clear at all. Moreover, a lot of irrelevant refinements are generated, especially in the case of the queries that consist of three and more query terms. The missing part in these refinement approaches is that the query represents a user's information need and as such a notion, it has to have a clear interpretation. In other words, for the refinement's purposes the query cannot be considered as a bag of words. Moreover, each potential refinement should clarify the meaning of the query as a whole, i.e. it should be interpreted in a common context of (possibly) all query terms. For example, if a user makes the query "workflow and personalization", then the refinement "process" can be considered as a highly relevant one only if there is a common context in which all three terms can be interpreted. On the other hand, the refinement "mining" that appears in the context of the term "workflow" but does not appear in any context with the term "personalization," will not be treated as (highly) relevant since there is no clear meaning of such a query as a whole, regarding the given information repository. Note, that it does not mean that it is not reliable that the user might have such an information need (like e.g. "personalized workflow mining"), but that, in the given repository, such an interpretation is not a common one and consequently it should be low ranked. As already mentioned, we treat the information retrieval as an exploratory process: the user should explore the information repository and align his information need to the content of the repository.

In other words, the crucial problem is the mapping between the user's cognitive space and the information space, as illustrated in Figure 6.4. The user has an information need that represents a logical unit (in Figure 6.4 represented as a meaningful graphical form in the

cognitive space). However, due to the very simple query syntax, the user represents an approximation of that need in the form of a set of keywords. On the other hand, due to a simple statistic-based retrieval model, this set of keywords can be mapped into different logical units in the information space and lots of them are completely different from the user's original information need, as illustrated in Figure 6.4. However, if we introduce more information from the query that determines the user's original information need, then the mapping into the information space can be performed more precisely. This additional information, depicted as "indicators" in Figure 6.4, is usually related to the structure of the query, like the order of the query terms. Finally, by combing statistical and structural information a very reliable interpretation (semantics) of the query can be discovered.

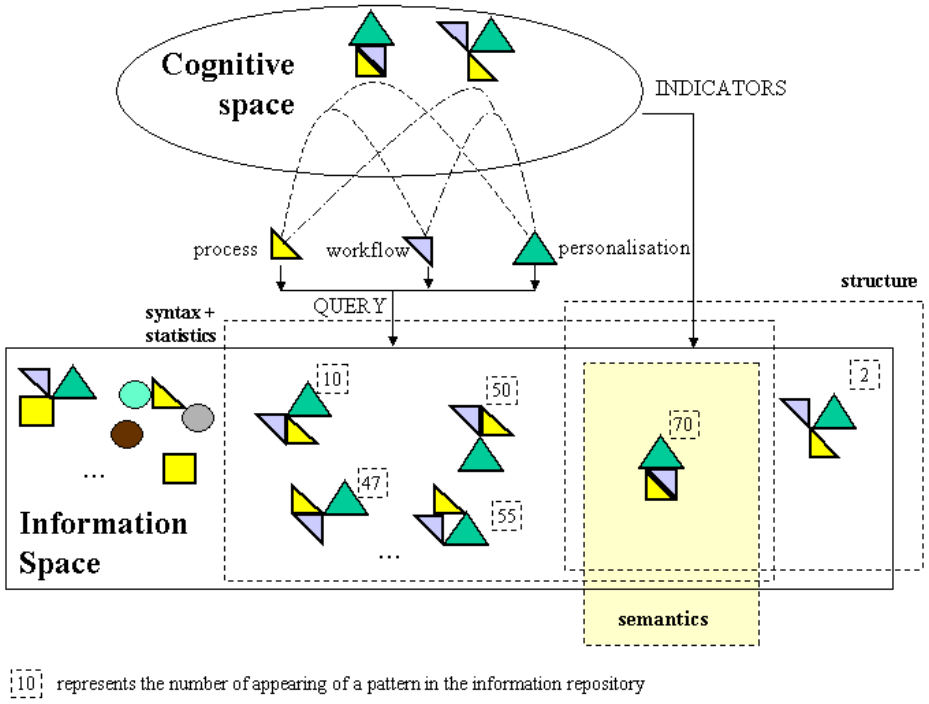


Figure 6.4: Closing the gap between the user's cognitive space and the information space by using the structure-based indicators of the user's information need

In order to realize the presented scenario the meaning of a query should be built out of the information repository. As we have already elaborated, the main problem is that a naive statistical approach cannot derive semantics from a text. On the other hand, we can consider the interpretation of a query from a more conceptual point of view. Indeed, if we consider a query as an approximation of the user's need, then we can imagine a conceptual extension of that query which will result in a description more similar to the given information need. This is very similar to the process man performs by analyzing the relevance of a text for his query. For example, in a text, a common context in which the query terms appear should be found; the smaller the context, the better the understanding of that common context is. Further, the role a common term (e.g. adjective or an object) in the context of query terms can indicate its meaning for the query. If we relate these "structural" elements to the statistical information regarding the frequency of these structures appearance, we might be able to generate the semantic meaning of a query as well as the role the query refinement can play in it.

For example, regarding the query mentioned above, if there are a lot of linguistic contexts in which the term "process" appears as an adjective for the word "workflow" and as an object for the word "personalization", it can be concluded that in that domain a subtype of workflow exists, which is described as a "process workflow" that can be personalized. It means that "process workflow and personalization" can be treated as a very suitable refinement.

Therefore, a semantic-based refinement of the initial query (i.e. semantics) is derived from the frequency of the appearance (i.e. a statistics) of a linguistic pattern (i.e. a structure).

In the above example, the usage of structure (in this case a linguistic pattern) represents the crucial difference between a traditional term- co-occurrence-based query refinement approach [6] and a more conceptual one that we present here. Regarding the above example, the traditional approach could find the term "process" as a possible refinement, but it cannot define its role in clearing the meaning of the query, i.e. the refined query looks like "process and workflow and personalization". Note that in a more conceptual case, the difference between "process workflow and personalization" and "workflow and personalization process" can be very useful.

This section shows the power of conceptual query refinement: If the query refinement retrieves the information that is semantically related to a query term, then the disambiguation process can combine that information on an abstract level, i.e. conceptually, in order to build the meaning of the query term. Since there are two elementary descriptors of an entity in an ontology: (1) a hierarchy the entity belongs to and (2) the relations with other entities, a query refinement should generate at least these two types of information in order to alleviate the disambiguation process. Obviously, a third type is needed in order to associate all other entities that are not directly (i.e. in a hierarchy or a direct relation) related to an entity.

6.3 Conceptual Query Model

6.3.1 Model

The main problem in analyzing a Boolean (i.e. keyword-based) query is the lack of a conceptualization in which the meaning of the query can be interpreted. For example, if the user posts the query "knowledge and management", then there are several interpretations of the symbol "knowledge" regarding the user's information need. For example, the user might be interested in: (i) knowledge management (as a subtype of management, i.e. "management of knowledge"), (ii) management knowledge (as a subtype of knowledge, i.e. "knowledge about management"), (iii) management and knowledge (as an arbitrary relation between management and knowledge). In order to define the problem of query ambiguity more formally, we have defined a conceptual model for describing the interpretation(s) of a query. It is presented in the form of an ontology in Figure 6.5. The model can be treated as an extension of the meaning triangle. The main task of the model is to represent the possible intensions (regarding information search) the user might have by specifying such a query.

The interpretation of a query depends on the interpretations of individual query terms. The interpretation of a query term can be defined through its relations with other terms (cf. Figure 6.5, concept *Relation*). The query's interpretation is then defined through the interpretation of these relations in various contexts (cf. Figure 6.5, concept *Context*) whereas these contexts are organized hierarchically (cf. Figure 6.5, relation *isPartOf*). Since the query's meaning is built hierarchically, the relationships should be established between *Relation* concepts as well (cf. Figure 6.5, *fromRelation*, *toRelation*). Therefore, the query's meaning is represented as a context that encompasses (hierarchical) relations between the query terms.

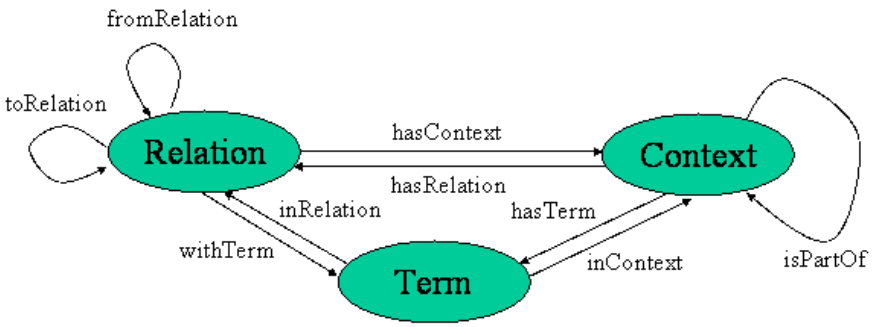


Figure 6.5: Conceptual model for query interpretation

The formal description of the model in F-Logic looks like:

```
Term::Object[                    Relation::Object[                    Context::Object[
        inRelation=>>Relation;           withTerm=>>Term;                     hasTerm=>>Term;
        inContext=>>Context].            fromRelation=>>Relation;             hasRelation=>>Relation;
                                         toRelation=>>Relation;               isPartOf=>>Context].
                                         hasContext=>>Context].
```

The query's meaning represents an information need that can be assigned to the given query. For example, if the user posts the query $query_1$= "knowledge, management, quality" a possible meaning "quality *regarding*[64] knowledge management" can be represented in the given model as a set of statements:

Context($query_1$), *Relation*(rel_1), *Term*("knowledge"), *Term*("management"), *Term*("quality"), *inRelation*("knowledge", rel_1), *withTerm*(rel_1, "management"), *Relation*(rel_2), *inRelation*("quality", rel_2), *toRelation*(rel_1, rel_2).

Such a meaning can be represented in short as: rel_2("quality", rel_1("knowledge", "management")). This is called a query model. Note that a query might have several meanings, i.e. it can be mapped into several query models.

By introducing this conceptual model of a keyword-based query, the query refinement process can be seen as the refinement of the query model. Indeed, for a query (q_1, q_2, … q_n), where q_i i=1, n are query terms, several query models can be built and for each of them several refinements can be generated. Figure 6.6 illustrates this process. Therefore, by using a conceptual representation of the query, a refinement is not represented just as a bag of words, but rather as a structure with an implicitly represented meaning. It will enable the user to better understand the refinements and to focus on semantic-relevant ones.

6.3.2 Modeling Relations and Contexts

A lot of relations can be established between two terms, for example between the terms "*process*" and "*workflow*": "a *process* is executed by a *workflow*" or "a *process* optimizes a *workflow*" or "a *process workflow* is a type of *workflow*". From the point of view of defining the meaning of a term, the exact naming of relations can be replaced by introducing types of relations which a term belongs to. Indeed, from the conceptual point of view, two terms are either in a *specialization* relation (*Specialize*) (i.e. a term specializes the meaning of another term, like "process + workflow = process workflow"), in a *modification* relation (*Modify*) (i.e. a term modifies the meaning of another term, like a "process executed by a workflow") or in a

[64] The word "regarding" is just a placeholder for a connection between the term "quality" and the phrase "knowledge management". The crucial point in building the query's meaning is that instead of treating the query as a bag of words, some relations between the query terms are established. In this case, the terms "knowledge" and "management" are related to each other and as a whole to the term "quality".

co-occurrence relation (*Co-occurrence*) (i.e. two terms just appear together in a context, without a direct influence on each other, like "… The process is described using a model, which can be found in the literature about workflows"). Note that this classification corresponds to the analyses regarding the frequently used query refinement "patterns", which show that 90% of the changes users make in subsequent refinements of a query are specializations or modifications (in the sense we have defined above) of the query [19]. The remaining 10% can be treated as adding just co-occurring terms. Therefore, our conceptual model reflects the users' refinement behavior quite well.

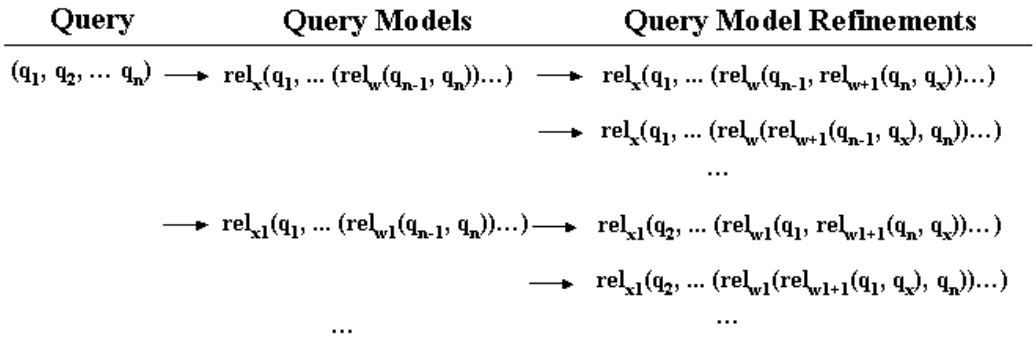


Figure 6.6: Basics of conceptual query refinement: (i) Query is interpreted as a conceptual model and (ii) refinement is performed on these models

It is clear that in an information repository there are lots of meanings in which a query term can appear. In the case that a query contains several query terms, only terms that appear together (regarding the conceptual model: in the same *Context*) can be treated as relevant ones. The structure of the contexts is determined by the syntactic organization of documents in the information repository, i.e. a Context can be a noun phrase[65], a sentence, or a paragraph, which corresponds to the way one captures the relatedness between two terms. It means that two semantically related query terms should appear together in the context of two noun phrases, sentences or paragraphs that are located close to each other, respectively. Figure 6.7 presents the above mentioned extensions of the model: There are three types of relations: *Specialize*, *Modify* and *Cooccurrence*. There are three types of contexts: *NounPhrase*, *Sentence* and *Paragraph* which are related to the *isPartOf* relation. The relationship *neighbour* between two contexts describes their collocation. A *Term* can play the role of a *head* in a noun phrase.

Finally, by using typing of relations a conceptual model of a query $(q_1, q_2, \dots q_n)$ is represented as a statement in the form $\mathbf{rel_{x1}}(q_1, \dots (\mathbf{rel_{xl}}(q_{n-1}, q_n))\dots)$, where $\mathbf{rel_{xi}} \in$ {*Specialize*, *Modify*, *Cooccurrence*}. For example, for the query (business, knowledge, management), the model *Modify*("business", *Specialize*("knowledge", "management")) represents an information need for "business *regarding* knowledge management", whereas the model *Specialize*(*Specialize*("business", "knowledge"), "management")) represents a need for "business knowledge management", as the management of the business knowledge.

Note that the aim of this conceptual model is not to generate a natural language expression of a query, but rather to define some relations between the query terms that should help the user specify the meaning of the query more precisely. The query refinement process will be based upon these relations.

[65] In linguistics a noun phrase is a phrase whose Head is a noun. For example, in the sentence "Semantic knowledge management is a new topic", the phrase "semantic knowledge management" is a noun phrase. The term "management" plays the role of the Head.

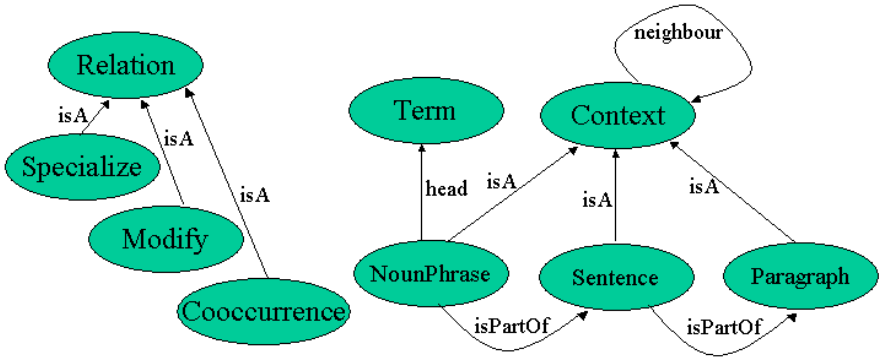


Figure 6.7: Entities of the conceptual model used in this research

6.3.3 Modeling Knowledge about the Refinement Process

The main advantage of using an ontology for modeling a domain is the possibility to express knowledge about the domain in the form of axioms. When used in the reasoning process, these axioms support the derivation of new knowledge regarding the particular state of affairs. This knowledge can be used (1) for eliminating inconsistent information (consistency checking), (2) for the derivation of the conceptual model and (3) for further reasoning about the problem at hand. In order to support these three processes we define three types of axioms:

- Consistency checking axioms, presented in Figure 6.8,
- Model derivation axioms used for the derivation of the conceptual model of a query which are described in section 6.4.1.1,
- Refinement generation axioms used for the derivation of the refinement which are described in section 6.4.2.

Rules regarding inverse relations:
$\forall c_1,r_1\ hasContext(r_1, c_1) \leftarrow Relation(r_1) \wedge Context(c_1) \wedge hasRelation(c_1, r_1)$.
$\forall c_1,t_1\ inContext(t_1, c_1) \leftarrow Term(t_1) \wedge Context(c_1) \wedge hasTerm(c_1, t_1)$.

Rules regarding transitivity of *isPartOf* relation:
$\forall c_1,c_2,\ c_3\ \ isPartOf(c_1,\ c_3) \leftarrow Context(c_1) \wedge Context(c_2) \wedge Context(c_3) \wedge isPartOf(c_1,\ c_2) \wedge isPartOf(c_2,\ c_3)$
$\forall t_1,r_1,c_1\ \ inContext(t_1,\ c_1) \leftarrow Relation(r_1) \wedge Term(t_1) \wedge Context(c_1) \wedge inRelation(t_1,\ r_1) \wedge hasContext(r_1,\ c_1)$
$\forall t_1,c_1,c_2\ \ inContext(t_1,\ c_2) \leftarrow Term(t_1) \wedge Context(c_1) \wedge Context(c_2) \wedge inContext(t_1,\ c_1) \wedge isPartOf(c_1,\ c_2)$
$\forall r_1,c_1,c_2\ \ hasContext(r_1,\ c_2) \leftarrow Relation(r_1) \wedge Context(c_1) \wedge Context(c_2) \wedge hasContext(r_1,\ c_1) \wedge isPartOf(c_1,\ c_2)$

Validation rules:
$\forall r_1,c_1\ NounPhrase(c_1) \leftarrow Specialize(r_1) \wedge hasContext(r_1, c_1)$
$\forall r_1,c_1\ Sentence(c_1) \leftarrow Modify(r_1) \wedge hasContext(r_1, c_1)$

Figure 6.8: Set of consistency checking axioms

These axioms are used in the process of building (instantiating) the conceptual model for a given query. As mentioned above, this model supports the interpretations of the meaning of the query, by establishing relations between the query terms and other terms from the information repository. Since the queries are typically very short (in average 2-4 terms), it is possible that there are several interpretations of a query regarding the given information repository. The presented model enables the determination of not only how ambiguous a query is, but moreover, which part(s) of the query contribute most to this ambiguity, e.g. the query term that is completely isolated from the rest of the query terms. Further, the model supports an analysis of the requirements for the disambiguation of the query, e.g. how to find links between an isolated part of the query and the rest of the query. Therefore, the model serves as a backbone for the query refinement process, which is the topic of the next section.

6.4 Conceptual Query Refinement Process

As a general workflow for performing query refinement we use the librarian agent query refinement process described in Chapter 3. The process consists of three phases: potential ambiguities (i.e. misinterpretations) of the initial query are firstly discovered and assessed (cf. the so-called *query ambiguity discovery* phase). Next, the suitable query refinements are generated in order to reduce the accounted ambiguities (cf. the so-called *refinements derivation* phase). Finally, the recommendations for refining the given query are ranked according to their relevance for fulfilling the user's information need and according to the possibility to disambiguate the meaning of the query (cf. the so-called *refinements ranking* phase). In the next three subsections we describe these three phases for our conceptual query refinement approach.

6.4.1 Phase 1: Query Ambiguity Discovery

Since the goal of the query refinement process is to reduce misinterpretations of a query, the first step in this phase is to discover interpretations (query models) that can be assigned to the query. The second step is to assess them in order to determine which of them can be treated as misinterpretations.

This phase corresponds to the query ambiguity phase in librarian agent query refinement process (see section 3.2.2.1). However, since an ontology for structuring the domain is not available, the content-related ambiguity cannot be calculated (see section 3.2.2.1.2). Moreover, instead of *generality* and *compactness* of a query (which are ontology-related categories), we calculate the *interpretation* of a query as the characteristic of the semantic ambiguity.

6.4.1.1 Query Interpretations

As we have already mentioned, the meaning of a query (i.e. an interpretation) is defined through the relations that can be established between query terms, i.e. for a query (q_1, q_2, …, q_n) the meaning is defined as

$\mathbf{rel_{x1}}(q_1, \ldots (\mathbf{rel_{xk}}(q_{n-1}, q_n))\ldots)$, whereas $\mathbf{rel_{xi}} \in \{Specialize, Modify, Cooccurrence\}$.

Since the user posts a query against an information repository in order to satisfy his information need, an interpretation of the query should emerge from that repository. However, the information repository contains plain text, which represents, in the first instance, linguistic information[66], like: there is a sentence, there is a verb, there is a noun phrase, etc. Therefore, in order to build the meaning of the query, one has to process this information and derive conceptual relations between the query terms.

[66] Deriving meaning of sentences goes beyond this information and requires some other types of knowledge.

The main problem in processing a text is that a lot of information about the relations between the terms can be generated, if one tries to apply all possible procedures for generating relations. For example, we can use the following two procedures:

1) all the terms in a noun phrase define the *Specialize* relation with the head of the noun phrase, and,

2) the terms from two consequent noun phrases are in the *Modify* relation.

For example, for the sentence "*A search algorithm for a minimal solution subgraph in AND/OR graphs with cycles is described*[67]." there are seven relations that can be defined for the terms "algorithm" and "subgraph", which represent the heads for the noun phrases "search algorithm" and "minimal solution subgraph", respectively: *Specialize*("search", "algorithm"), *Specialize*("minimal", "subgraph"), *Specialize*("solution", "subgraph"), *Modify*("algorithm", "subgraph"), *Modify*("search", "subgraph"), *Modify*("algorithm", "minimal"), *Modify*("algorithm", "solution").

It is clear that such an approach leads to an explosion of relations, whereas some of them can be useless for a particular situation, e.g. *Modify*("algorithm", "minimal").

On the other side, it is possible that some relations are missing due to the inflexibility in the definition of the above-mentioned procedures. For example, the relation between terms "algorithm" and "graph" will not be discovered since the noun phrases, which they are belonging to, are not direct neighbors. Therefore, some new procedures should be added, which produce a new amount of (mostly useless) relations. The useless relations cause severe problems due to the memory and time constraints, since query refinement is an activity that is performed at query-time. Therefore, a procedural approach of generating all possible relations between the query terms and other terms from a relevant text seems to be not adequate for the query refinement task.

In order to resolve this problem we use the formal, logic-based nature of the conceptual model and generate relations between terms on-demand. It means, that if it is necessary to find relations between two terms then an inference process should start in order to generate these relations. Therefore, instead to prove if one of the already generated relations satisfies a given condition, the relations that satisfy this condition are generated in the inference process. Figure 6.9 presents the difference between the procedural and the inference-based approach.

Therefore, we can formalize procedural knowledge needed for the extraction of the conceptual relations in a set of axioms. The axioms that drive this inference process specify *declaratively* the procedural knowledge used in the above mentioned traditional approach. It means that several axioms are defined in order to describe the knowledge about the refinement process. In the following we list these axioms for the query model derivation.

Axiom1.1: Deriving relation *Specialize* between two terms:

$\forall t_1,t_2,r_1,c_1\ Specialize(r_1) \wedge inRelation(t_1, r_1) \wedge withTerm(r_1, t_2) \wedge hasContext(r_1, c_1)$

$\leftarrow Term(t_1) \wedge Term(t_2) \wedge inContext(t_1, c_1) \wedge inContext(t_2, c_1) \wedge NounPhrase(c_1) \wedge head(c_1,t_2)$,

where predicate *NounPhrase(a)* denotes that *a* is a noun phrase

predicate *head(a, b)* indicates that b is the head of the noun phrase *a*.

This axiom means that each term in a noun phrase is the specialization of the head of that noun phrase. However, during the execution of this axiom, a set of other axioms (e.g. consistency checking axioms) can be evaluated and new information will be derived, so that

[67] The examples used in this chapter are taken from the *CompuScience* dataset, http://www.fiz-informationsdienste.de/en/DB/compusci/index. html.

more relevant results will be produced. Beside the compactness of the knowledge representation, this is another advantage of the inference-based approach.

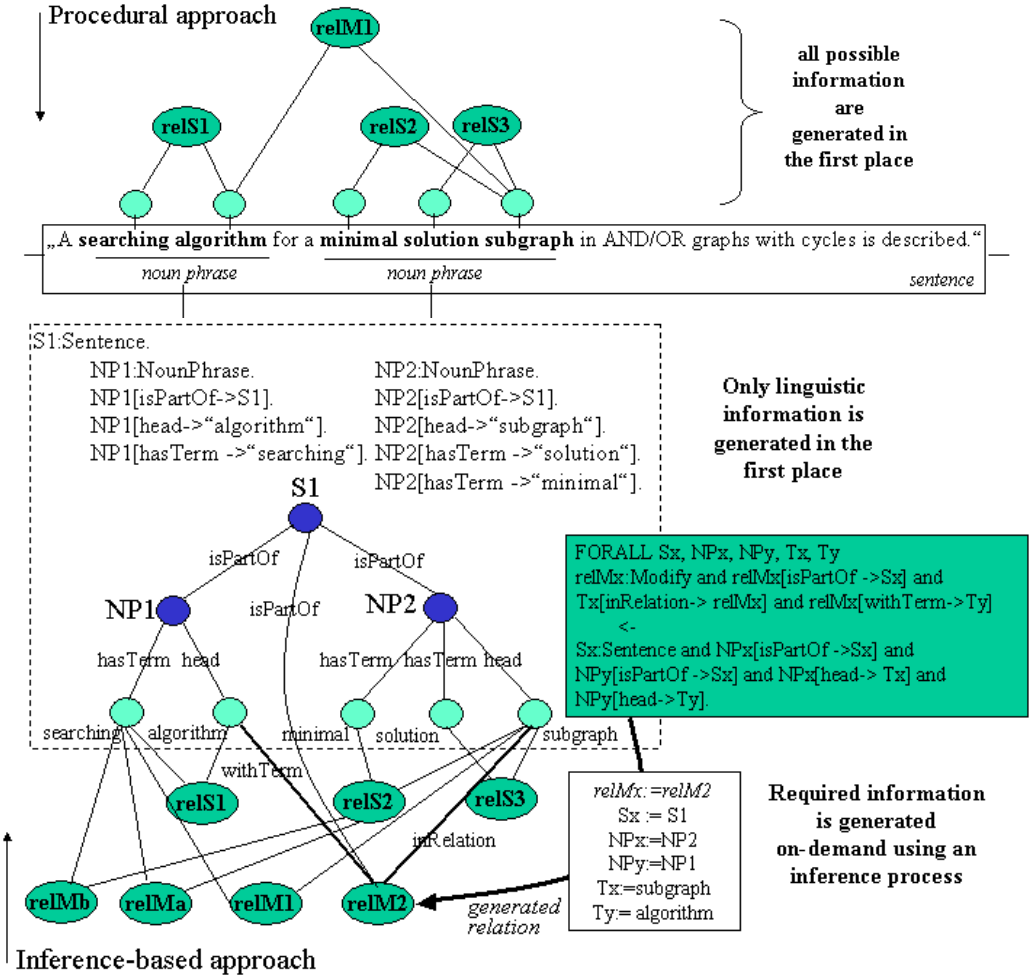


Figure 6.9: Difference between procedural- and inference-based approaches for deriving a conceptual query model. The upper part illustrates the procedural approach, whereas the bottom part illustrates the inference-based approach for generating relations between terms. The axiom presented in the right part enables the derivation of the relation relM2. All other relations (relXY) are generated in an analogous way. As a logical language we use F-Logic

Axiom1.2: Deriving relation *Specialize* between a relation and a term:
$\forall t_1,r_1,r_2,c_1$ $Specialize(r_2) \wedge inRelation(t_1, r_2) \wedge toRelation(r_2, r_1) \wedge hasContext(r_2, c_1) \leftarrow Term(t_1) \wedge Specialize(r_1) \wedge hasContext(r_1, c_1) \wedge hasContext(r_2, c_1) \wedge \exists t_2\ inRelation(t_2, r_2) \wedge \exists r_4\ Specialize(r_4) \wedge ((inRelation(t_1, r_4) \wedge withTerm(r_4, t_2)) \vee (inRelation(t_2, r_4) \wedge withTerm(r_4, t_1))) \wedge hasContext(r_4, c_1)$

Axiom1.3: Deriving relation *Specialize* between a term and a relation:
$\forall t_1,r_1,r_2,c_1$ $Specialize(r_2) \wedge withTerm(r_2, t_1) \wedge fromRelation(r_1, r_2) \wedge hasContext(r_2, c_1) \leftarrow Term(t_1) \wedge Specialize(r_1) \wedge hasContext(r_1, c_1) \wedge hasContext(r_2, c_1) \wedge \exists t_2\ withTerm(r_2, t_2) \wedge \exists r_4\ Specialize(r_4) \wedge ((inRelation(t_1, r_4) \wedge withTerm(r_4, t_2)) \vee (inRelation(t_2, r_4) \wedge withTerm(r_4, t_1))) \wedge hasContext(r_4, c_1)$

Axiom1.4: Deriving relation *Specialize* between two relations:
$\forall r_1,r_2,r_3,c_1$ $Specialize(r_3) \wedge fromRelation(r_1, r_3) \wedge toRelation(r_3, r_2) \wedge hasContext(r_3, c_1) \leftarrow Specialize(r_1) \wedge Specialize(r_2) \wedge hasContext(r_1, c_1) \wedge hasContext(r_2, c_1) \wedge \exists t_1,t_2$

$(inRelation(t_1, r_1) \vee withTerm(r_1, t_1)) \wedge (inRelation(t_2, r_2) \vee withTerm(r_2, t_2)) \wedge \exists r_4$ $Specialize(r_4) \wedge inRelation(t_1, r_4) \wedge withTerm(r_4, t_2) \wedge hasContext(r_4, c_1)$

Axiom1.5: Deriving relation *Specialize* between two relations:
$\forall r_1,r_2,r_3,c_1$ $Specialize(r_3) \wedge fromRelation(r_2, r_3) \wedge toRelation(r_3, r_1) \wedge hasContext(r_3, c_1) \leftarrow$ $Specialize(r_1) \wedge Specialize(r_2) \wedge hasContext(r_1, c_1) \wedge hasContext(r_2, c_1) \wedge \exists t_1,t_2$ $(inRelation(t_1, r_1) \vee withTerm(r_1, t_1)) \wedge (inRelation(t_2, r_2) \vee withTerm(r_2, t_2)) \wedge \exists r_4$ $Specialize(r_4) \wedge inRelation(t_1, r_4) \wedge withTerm(r_4, t_2) \wedge hasContext(r_4, c_1)$

Axiom 2.1: Deriving relation *Modify* between two terms:
$\forall t_1,t_2,r_1,c_1,c_2,c_3$ $Modify(r_1) \wedge inRelation(t_1, r_1) \wedge withTerm(r_1, t_2) \wedge hasContext(r_1, c_3) \leftarrow$ $Term(t_1) \wedge Term(t_2) \wedge inContext(t_1, c_1) \wedge inContext(t_2, c_2) \wedge NounPhrase(c_1) \wedge$ $NounPhrase(c_2) \wedge Sentence(c_3) \wedge isPartOf(c_1, c_3) \wedge isPartOf(c_2, c_3) \wedge$ neighbor(c_1,c_2),
where neighbor(x, y) is a predicate that returns true if in the given context noun phrase x appears next to y or vice versa.

Axiom 2.2: Deriving relation *Modify* between a relation and a term:
$\forall t_1,r_1,r_2,c_1,c_2,c_3$ $Modify(r_2) \wedge inRelation(t_1, r_2) \wedge toRelation(r_2, r_1) \wedge hasContext(r_2, c_1) \leftarrow$ $Term(t_1) \wedge (Specialize(r_1) \vee Modify(r_1)) \wedge inContext(t_1, c_1) \wedge hasContext(r_1, c_1) \wedge \exists t_2$ $inRelation(t_2, r_2) \wedge \exists r_4$ $Modify(r_4) \wedge ((inRelation(t_1, r_4) \wedge withTerm(r_4, t_2)) \vee (inRelation(t_2,$ $r_4) \wedge withTerm(r_4, t_1))) \wedge hasContext(r_4, c_1)$

Axiom2.3: Deriving relation *Modify* between a term and a relation:
$\forall t_1,r_1,r_2,c_1,c_2,c_3$ $Modify(r_2) \wedge withTerm(r_2, t_1) \wedge fromRelation(r_1, r_2) \wedge hasContext(r_2, c_1) \leftarrow$ $Term(t_1) \wedge (Specialize(r_1) \vee Modify(r_1)) \wedge inContext(t_1, c_1) \wedge hasContext(r_1, c_2) \wedge \exists t_2$ $withTerm(r_2, t_2) \wedge \exists r_4$ $Modify(r_4) \wedge ((inRelation(t_1, r_4) \wedge withTerm(r_4, t_2)) \vee (inRelation(t_2,$ $r_4) \wedge withTerm(r_4, t_1))) \wedge hasContext(r_4, c_1)$

Axiom2.4: Deriving relation *Modify* between two relations:
$\forall r_1,r_2,r_3,c_1,c_2,c_3$ $Modify(r_3) \wedge fromRelation(r_1, r_3) \wedge toRelation(r_3, r_2) \wedge hasContext(r_3, c_1)$ $\leftarrow Relation(r_1) \wedge Relation(r_2) \wedge \neg Cooccurence(r_1) \wedge \neg Cooccurence(r_2) \wedge hasContext(r_1, c_1)$ $\wedge hasContext(r_2, c_1) \wedge \exists t_1,t_2$ $(inRelation(t_1, r_1) \vee withTerm(r_1, t_1)) \wedge (inRelation(t_2, r_2) \vee$ $withTerm(r_2, t_2)) \wedge \exists r_4$ $Modify(r_4) \wedge inRelation(t_1, r_4) \wedge withTerm(r_4, t_2) \wedge hasContext(r_4, c_1)$

Axiom2.5: Deriving relation *Modify* between two relations:
$\forall r_1,r_2,r_3,c_1,c_2,c_3$ $Modify(r_3) \wedge fromRelation(r_2, r_3) \wedge toRelation(r_3, r_1) \wedge hasContext(r_3, c_1)$ $\leftarrow Relation(r_1) \wedge Relation(r_2) \wedge \neg Cooccurence(r_1) \wedge \neg Cooccurence(r_2) \wedge hasContext(r_1, c_1)$ $\wedge hasContext(r_2, c_1) \wedge \exists t_1,t_2$ $(inRelation(t_1, r_1) \vee withTerm(r_1, t_1)) \wedge (inRelation(t_2, r_2) \vee$ $withTerm(r_2, t_2)) \wedge \exists r_4$ $Modify(r_4) \wedge inRelation(t_2, r_4) \wedge withTerm(r_4, t_1) \wedge hasContext(r_4, c_1)$

Axiom 3.1: Deriving relation *Coocurence* between terms:
$\forall t_1,t_2,c_1,c_2,c_3,r_1$ $Cooccurence(r_1) \wedge inRelation(t_1, r_1) \wedge withTerm(r_1, t_2) \wedge hasContext(r_1, c_3)$ $\leftarrow Term(t_1) \wedge Term(t_2) \wedge inContext(t_1, c_1) \wedge inContext(t_2, c_2) \wedge NounPhrase(c_1) \wedge$ $NounPhrase(c_2) \wedge Sentence(c_3) \wedge isPartOf(c_1, c_3) \wedge isPartOf(c_2, c_3) \wedge \neg$neighbor$(c_1,c_2)$,

Axiom 3.2: Deriving relation *Coocurence* between a relation and a term:
$\forall t_1,r_1,c_1,c_2,,r_2$ $Cooccurence(r_2) \wedge withTerm(r_2, t_1) \wedge fromRelation(r_2, r_1) \wedge hasContext(r_2, c_1)$ $\leftarrow Term(t_1) \wedge Relation(r_1) \wedge \exists c_1$ $inContext(t_1, c_1) \wedge hasContext(r_1, c_1) \wedge \exists t_2$ $withTerm(r_2, t_2)$

$\wedge\ \exists\ r_4\ \ Cooccurence(r_4) \wedge ((inRelation(t_1, r_4) \wedge withTerm(r_4, t_2)) \vee (inRelation(t_2, r_4) \wedge withTerm(r_4, t_1))) \wedge hasContext(r_4, c_1)$

Axiom 3.3: Deriving relation *Coocurence* between a term and a relation:
$\forall t_1,r_1,c_1,c_2,r_2\ Cooccurence(r_2) \wedge inRelation(t_1, r_2) \wedge toRelation(r_2, r_1) \wedge hasContext(r_2, c_1) \leftarrow Term(t_1) \wedge Relation(r_1) \wedge \exists c_1\ inContext(t_1, c_1) \wedge hasContext(r_1, c_1) \wedge \exists\ t_2 \wedge inRelation(t_2, r_2) \wedge \exists\ r_4\ \ Cooccurence(r_4) \wedge ((inRelation(t_1, r_4) \wedge withTerm(r_4, t_2)) \vee (inRelation(t_2, r_4) \wedge withTerm(r_4, t_1))) \wedge hasContext(r_4, c_1)$

Axiom 3.4: Deriving relation *Coocurence* between two relations:
$\forall r_1,r_2,c_1,c_2,c_3,r_3\ \ Cooccurrence(r_3) \wedge fromRelation(r_1, r_3) \wedge toRelation(r_3, r_2) \wedge hasContext(r_3, c_1) \leftarrow Relation(r_1) \wedge Relation(r_2) \wedge \exists c_1\ hasContext(r_1, c_1) \wedge hasContext(r_2, c_1) \wedge \exists\ t_1,t_2\ (inRelation(t_1, r_1) \vee withTerm(r_1, t_1)) \wedge (inRelation(t_2, r_2) \vee withTerm(r_2, t_2)) \wedge \exists\ r_4\ Cooccurence(r_4) \wedge inRelation(t_1, r_4) \wedge withTerm(r_4, t_2) \wedge hasContext(r_4, c_1)$

Axiom 3.5: Deriving relation *Coocurence* between two relations:
$\forall r_1,r_2,c_1,c_2,c_3,r_3\ \ Cooccurrence(r_3) \wedge fromRelation(r_2, r_3) \wedge toRelation(r_3, r_1) \wedge hasContext(r_3, c_1) \leftarrow Relation(r_1) \wedge Relation(r_2) \wedge \exists c_1\ hasContext(r_1, c_1) \wedge hasContext(r_2, c_1) \wedge \exists\ t_1,t_2\ (inRelation(t_1, r_1) \vee withTerm(r_1, t_1)) \wedge (inRelation(t_2, r_2) \vee withTerm(r_2, t_2)) \wedge \exists\ r_4\ Cooccurence(r_4) \wedge inRelation(t_2, r_4) \wedge withTerm(r_4, t_1) \wedge hasContext(r_4, c_1)$

These axioms (1.1 – 3.5) formalize the procedural knowledge needed for extracting these three types of relations between terms. Note that we are discovering only the relations that are related to the query terms without doing the formalization of all sentences in all documents.

The very first phase in generating a query's interpretation is the shallow NLP[68] processing of the results of the query. It results in a set of noun phrases, related to the query terms, which are placed in a particular context (sentence, paragraph). Moreover, the relations (order) between noun phrases are discovered as well. Therefore, the linguistic processing creates the initial fact base that is used as a knowledge base in the inference process, that is, the second phase in the interpretation process. Figure 6.9 illustrates this process.

The query models are built using the mentioned set of axioms. Since the task of this processing is to determine the most plausible interpretation of the query, a complete set of query models is generated. However, our logic-based approach enables the automation of this process. Indeed, we can define logic queries for generating abstract query models that will result in concrete instances of the model. It means that we need one logic query to derive a query model.

We illustrate this process for a query with three query terms:

For a query (q_1, q_2, q_3) the generic set of models is as follows: {$\mathbf{rel_x}(q_1, \mathbf{rel_y}(q_2, q_3))$, $\mathbf{rel_x}(q_2, \mathbf{rel_y}(q_1, q_3))$, $\mathbf{rel_x}(q_3, \mathbf{rel_y}(q_1, q_2))$}, where $\mathbf{rel_x}$, $\mathbf{rel_y} \in$ {*Specialize*, *Modify*, *Cooccurrence*}.

In F-Logic, the logic query for deriving the model $\mathbf{rel_x}(q_1, \mathbf{rel_y}(q_2, q_3))$ looks like:

> FORALL $\mathbf{rel_{xx}}$, $\mathbf{rel_{yy}}$ <- q_1[inRelation -> $\mathbf{rel_{xx}}$] and q_2[inRelation -> $\mathbf{rel_{yy}}$] and $\mathbf{rel_{yy}}$[withTerm -> q_3] and $\mathbf{rel_{xx}}$[toRelation -> $\mathbf{rel_{yy}}$] and $\mathbf{rel_{xx}}$:$\mathbf{rel_x}$ and $\mathbf{rel_{yy}}$:$\mathbf{rel_y}$. (1)

[68] We use the existing NLP open source packages for this purpose, like Text2Onto (text2onto.semanticweb.org)

This query produces all instances of query model $\mathbf{rel_x}(q_1, \mathbf{rel_y}(q_2, q_3))$. Note that performing this logic query causes the evaluation of several axioms for deriving relations we have given above.

The main advantage of the inference process is that it introduces an abstraction level that enables focusing only on important terms, i.e. only information relevant for the refinement will be generated. Figure 6.10 illustrates such an abstraction process in building a query model.

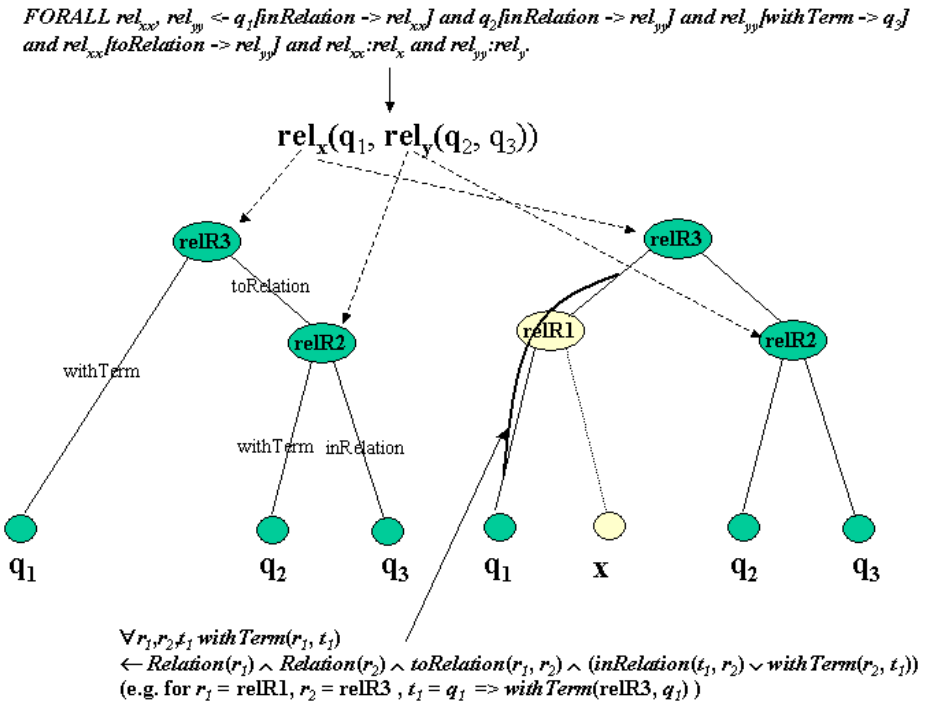


Figure 6.10: Abstraction process driven by transitivity axioms. The statement on the top is a F-Logic query used for inferring query model $\mathbf{rel_x}(q_1, \mathbf{rel_y}(q_2, q_3))$ (see (1)). The statement on the bottom is a transitivity axiom that is evaluated in order to generate, in the right-hand part of the Figure, a direct connection between relations between q_1 and a relation of q_2 and q_3 as required by (1). In the left-part such an "abstraction" process is not necessary.

Regarding complexity, for a query with n-terms there are (n!)/2 queries in the form: $\mathbf{rel_x}(q_1, \mathbf{rel_y}(q_2, \ldots \mathbf{rel_z}(q_{n-1}, q_n)\ldots))$. However, as we have already mentioned, the average length of web queries is between 2 and 4, which makes the approach efficient for a realistic search scenario. For example, for n = 4, there are in total 48 possible query model forms.

6.4.1.2 Ambiguity Calculation

Each query model generated in the previous step represents a possible interpretation of the query.

For example, for the query ("algorithm", "solution", "subgraph"):

Modify(algorithm, *Specialize*("solution", "subgraph")) means
„algorithm regarding solution subgraph"
Cooccurrence("algorithm", *Specialize*("solution", "subgraph")) means
„algorithm and solution subgraph"
Cooccurrence ("solution", *Modify* ("algorithm", "subgraph")) means
„solution and algorithm regarding subgraph".

However, beside the interpretation, each of the models has a certain level of support that indicates the reliability of a model. For example, the model *Modify*("algorithm", *Specialize*("solution", "subgraph")) can have a larger support than the model *Cooccurrence*("solution", *Modify*("algorithm", "subgraph")), which means that the first model reflects the user's information need in a more reliable manner. The basis for such a measurement is a statistical evaluation of the particular relations, like *Specialize*("solution", "subgraph") or *Modify*("algorithm", "subgraph"). Obviously, if there are a lot of instances of a relation, then the relation can be treated as a reliable interpretation of the relation between the given terms. Further, the model that includes that relation can be treated as a more reliable one, i.e. as a less ambiguous (misinterpreted) query model.

It is clear that this measurement has to take into account various parameters, which are topics of the next subsections. Since a query consists of several query terms, we will start with the discussion about the ambiguity of a query term.

6.4.1.2.1 Ambiguity of a Query Term

As we have already mentioned, the meaning (interpretation) of a query term can be defined through the relations with other terms. It is clear that the number of such relations indicates the diversity in the interpretation of a query term. We define two functions in order to assess this interpretability. Note that, since a domain ontology is not available here, *interpretability* represents the notion that replaces *generality* and *compactness* we introduced in section 3.2.2.1.1 as the two characteristics of the *semantic ambiguity*.

Each of these functions has two forms since the used relations are not symmetric, i.e. the meaning of the relation *rel*(a,b) is different from *rel*(b,a) for $rel \in \{Specialize, Modify, Cooccurrence\}$. For example, the relations *Specialize*("knowledge", "management") and *Specialize*("management", "knowledge") have different interpretations, as we have already illustrated at the beginning of this section.

$$
\begin{aligned}
&TermInterpretability(t, term_x, r\_type, first) = \\
&\quad \{x \mid r\_type(x) \wedge inRelation(t, x) \wedge withTerm(x, term_x)\} \\
&TermInterpretability(t, term_x, r\_type, second) = \\
&\quad \{x \mid r\_type(x) \wedge inRelation(term_x, x) \wedge withTerm(x, t)\} \\
&TermInterpretabilityTotal(t, r\_type, first) = \\
&\quad \{x \mid \exists y \in T\ r\_type(x) \wedge inRelation(t, x) \wedge withTerm(x, y)\} \\
&TermInterpretabilityTotal(t, r\_type, second) = \\
&\quad \{x \mid \exists y \in T\ r\_type(x) \wedge inRelation(y, x) \wedge withTerm(x, t)\},
\end{aligned}
\tag{2}
$$

where:

$r\_type \in \{Specialize, Modify, Cooccurrence\}$

$t \in Q_x$ (the set of query terms),

$term_x \in T$ (the set of all terms that can be found in the given repository)

$r\_type(x)$ means that relation x is of the type r_type.

The first function (*TermInterpretability*) calculates all relations of type *r_type* between *t* and *term*. The second function (*TermInterpretabilityTotal*) calculates all relations of type *r_type* related to *t*.

For example, regarding Figure 6.9, *TermInterpretabilityTotal*(subgraph, *Specialize, first*) = {relS2, relS3}, if we assume that relSx depicts a *Specialize* relation.

Figure 6.11 illustrates the calculation of these functions.

In order to ensure the completeness of the calculation (i.e. to find all possible relations of the given type), each of these formula is described declaratively and calculated using an inference

process. For example, the set of r_type_x relations between terms t and t_x can be calculated using the following F-Logic query:

TermInterpretability(t, t_x, r_type_x, *first*):= (3)
FORALL $\mathbf{rel_x}$ <- $\mathbf{rel_x}$:r_type_x and t[inRelation -> $\mathbf{rel_x}$] and $\mathbf{rel_x}$[withTerm -> t_x].

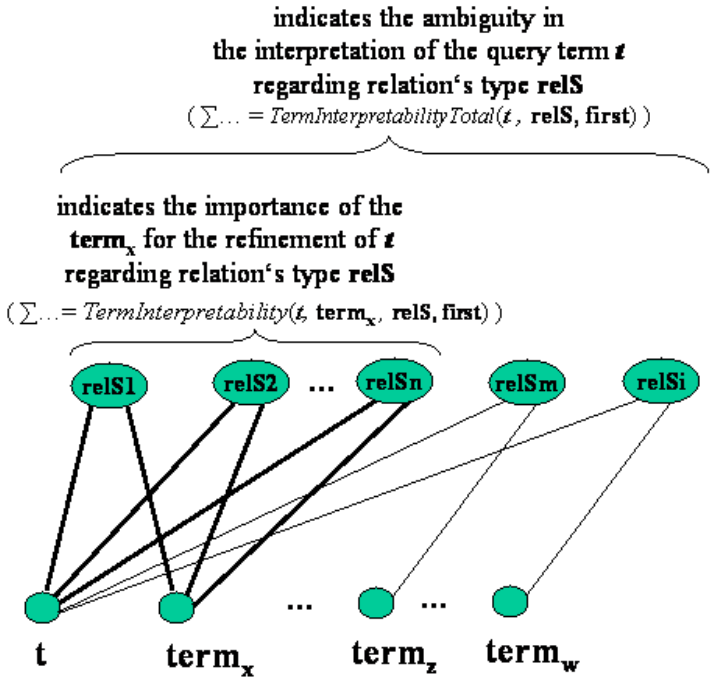


Figure 6.11: Illustration of the calculations related to the term interpretability. *TermInterpretability*(t, $term_x$, *relS, first*) calculates all relations of type *relS* between t and $term_x$. *TermInterpretabilityTotal*(t, *relS, first*) calculates all relations of type *relS* where t is the first argument. *relSx* are descriptions of the relations of type *relS*∈ {*Specialize*, *Modify*, *Cooccurrence*}.

6.4.1.2.2 Ambiguity of a Query

Since we assume that a query represents a user's information need, the interpretation of a query is related to a plausible meaning that can be assigned to the query as a whole. Consequently, the query's ambiguity is defined through the number of interpretations of a query.

However, the crucial question is how to define a "plausible meaning" that can be assigned to the query. Since the query's meaning is captured by the relations between the query terms, we define the "plausibility" as the support that this meaning "collects" in the information repository (i.e. how many instances of this relation exist in the repository). In order to calculate that support, we define the function *RelationSet*($\mathbf{rel_x}$(a, b)) that returns the set of relations of type $\mathbf{rel_x}$ established between a and b, as follows:

RelationSet($\mathbf{rel_x}$) ={

if $\mathbf{rel_x}$:= $r\_type_x(q_p, q_n)$	*TermInterpretability*(q_p, q_n, r_type_x, *first*)
if $\mathbf{rel_x}$:= $rel_x(t, rel_z)$	$\{x \mid \exists y \in RelationSet(rel_z)\ rel_x(x) \wedge inRelation(t, x) \wedge toRelation(x, y)\}$
if $\mathbf{rel_x}$:= $rel_x(rel_z, t)$	$\{x \mid \exists y \in RelationSet(rel_z)\ rel_x(x) \wedge fromRelation(t, x) \wedge withTerm(x, y)\}$

if $\mathbf{rel_x} := rel_x(rel_y, rel_z)$ $\{x| \exists y \in RelationSet(rel_y) \exists z \in RelationSet(rel_z) rel_x(x) \wedge fromRelation(y, x) \wedge toRelation(x, z)\}$

where:

$\mathbf{rel_x} := rel_x(t, rel_z)$ depicts that relation $\mathbf{rel_x}$ is defined between a term t and a relation rel_z

$\mathbf{rel_x} := rel_x(rel_y, rel_z)$ depicts that relation $\mathbf{rel_x}$ is defined between two relations.

For example, regarding Figure 6.9:
RelationSet(*Specialize*("solution", "subgraph")) =
TermInterpretability("solution", "subgraph", *Specialize*, first) = {relS3}

and

RelationSet(*Modify*(search, (*Specialize*("minimal", "subgraph")) = {relMa, relMb}.

The task of this measure is to compare the interpretability that a query term has in the information repository and the interpretation that is assigned to it in a particular query model. Figure 6.12 illustrates a mismatch that can arise between these two interpretations.

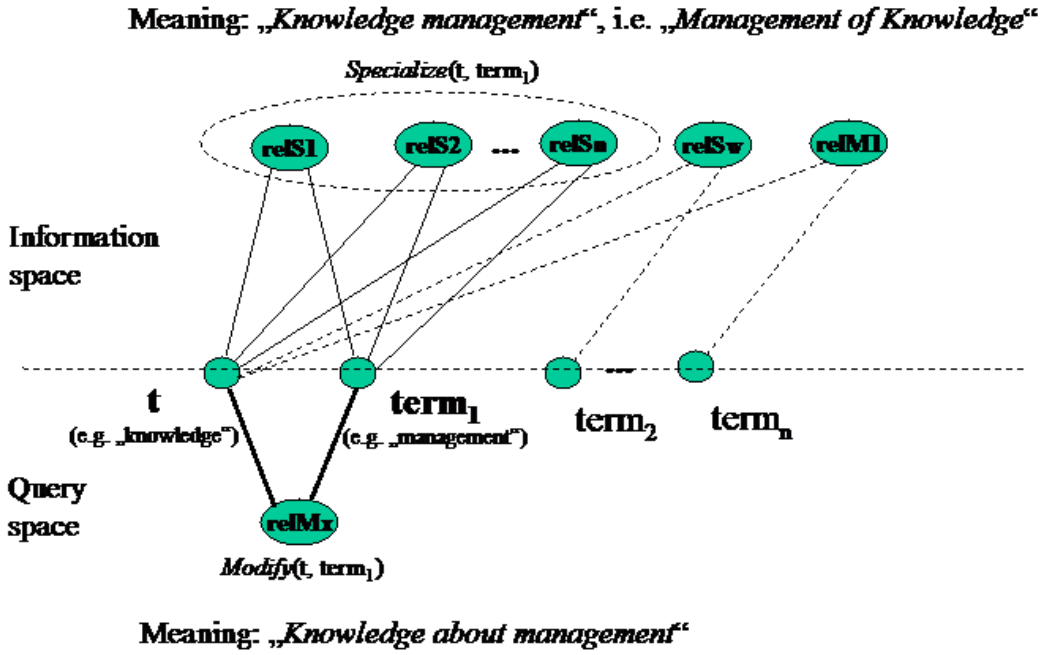


Figure 6.12: Illustration of a mismatch between two interpretations. In the information repository space term t is in various relations of type *Specialize* with other terms, whereas the relation to term $term_1$ is especially strong, which means that *Specialize*(t, $term_1$) is a very plausible meaning. If we assume that the user's need is related to a relation of type *Modify* between t and term $term_1$, i.e. *Modify*(t, $term_1$), the mismatch in meaning is obvious.

Note that *RelationSet*($\mathbf{rel_x}(a, b)$) represents a set of relations of type $\mathbf{rel_x}$ established between a and b. Since a relation can be treated as a composition of other relations, i.e. $rel_x = rel_x(q, rel_y)$, the calculation of the *RelationSet* can be performed recursively, as illustrated in Figure 6.13.

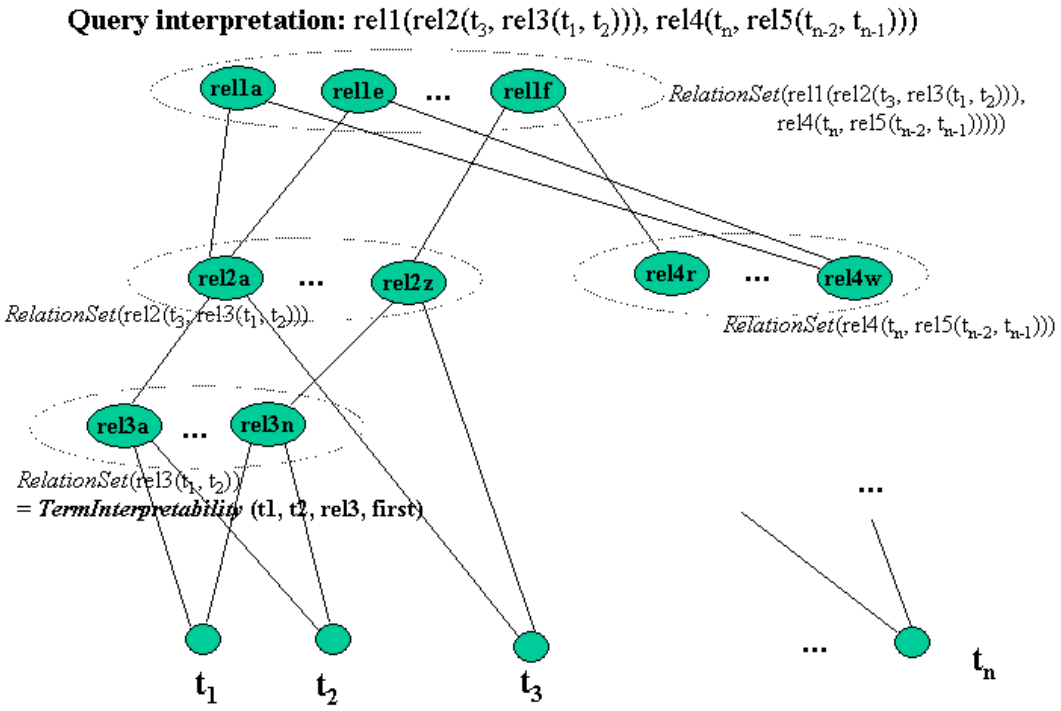


Figure 6.13: Illustration of the iterative calculation of function *RelationSet*. The query terms, given at the bottom of Figure, are pairwise grouped in several subsequent steps. In each step all possible relations of the given type (e.g. rel3a-rel3n for type rel3∈{*Specialize*, *Modify*, *Cooccurrence*}) are counted. That set represents the value for the corresponding *RelationSet*

Like a query term, a relation term can be interpreted in various ways regarding a given query term:

$RelationInterpretability(rel1, term_x, r\_type, first) =$
$\{x|\exists r \in RelationSet(rel1)\ r\_type(x) \wedge fromRelation(r, x) \wedge withTerm(x, term_x)\}$
$RelationInterpretability(rel1, term_x, r\_type, second) =$
$\{x|\ \exists r \in RelationSet(rel1)\ r\_type(x) \wedge inRelation(term_x, x) \wedge toRelation(x, r)\}$
$RelationInterpretabilityTotal(rel1, r\_type, first) =$
$\{x|\exists y \in T\ x \in RelationInterpretability(rel1, y, r\_type, first)\}$
$RelationInterpretabilityTotal(rel1, r\_type, second) =$
$\{x|\ \exists y \in T\ x \in RelationInterpretability(rel1, y, r\_type, second)\}$

where:
$r\_type \in \{ Specialize, Modify, Cooccurrence\}$,
$term_x \in T$ (set of all the terms that can be found in the given repository),
parameters first and second paly the same role we desribed in section 6.4.1.2.1.

The first function (*RelationInterpretability*) calculates all relations of type *r_type* between *rel1* and *term*. The second function (*RelationInterpretabilityTotal*) calculates all relations of type *r_type* related to *rel1*. An illustration of these functions is depicted in Figure 6.14.

In order to ensure the completeness of the calculation (i.e. to find all possible relations of the given type), each of these formula is described declaratively and calculated using an inference process. For example, the set of r_type_x relations between relation *rel1* and t_x can be calculated using the following F-Logic query:

$RelationInterpretability(rel1, t_x, r\_type_x, first) :=$

FORALL $\mathbf{rel_x}$ <- $\mathbf{rel_x}$: r_type_x and $\mathbf{rel_x}$[fromRelation -> *rel1*] and $\mathbf{rel_x}$[withTerm -> t_x].

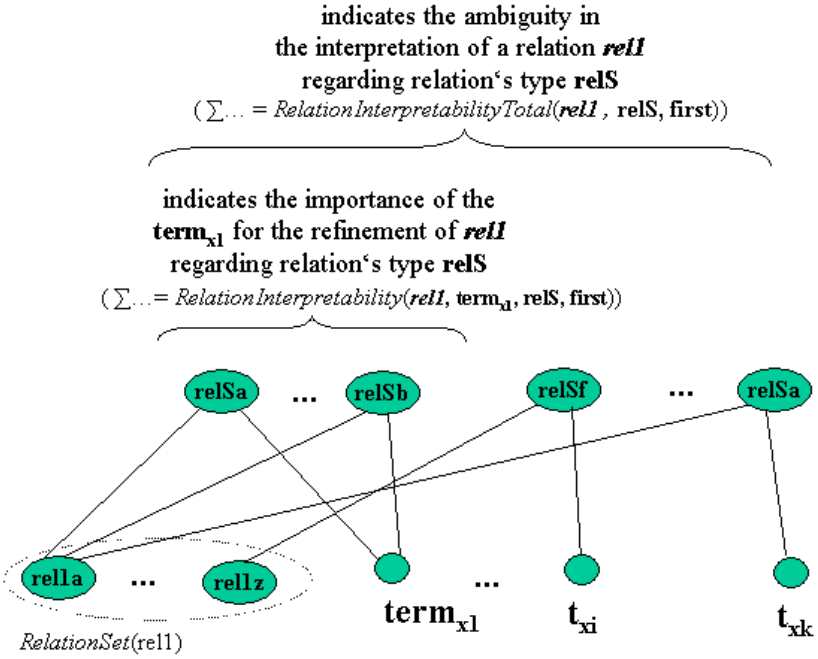


Figure 6.14: Illustration of the calculations related to the query interpretability. *RelationInterpretability*(*rel1*, $term_{x1}$, *relS, first*) calculates all relations of the type *relS* between relation *rel1* and $term_{x1}$. *RelationInterpretabilityTotal*(*rel1*, *relS, first*) calculates all relations of type *relS* where *rel1* is the first argument. Note that *rel1a* - *rel1z* represents elements of the set *RelationSet*(*rel1*).

So far, in order to discover the meaning of a query, we have analyzed only query terms. However, if we consider the query as an approximation of a user's information need, it seems sound to assume that the structure of the query is a valuable source for determining the meaning of the query. Indeed, a query is produced in a cognitive process that should be (implicitly) reflected in the structure of the query. We use the following heuristic in order to reflect this process:

"*A user does not order query terms arbitrarily, but rather in a meaningful way*".

Indeed, the order of query terms can indicate the meaning. For example, the need that caused the query "quality system reengineering"(e.g. quality regarding system reengineering) seems to be different from the need that caused the query "system quality reengineering"(e.g. system for quality reengineering).

In order to take this into account we have defined a new function *OrderImportance*, as a combination of the distance between terms in query *Q* and the order between them regarding a relation $\mathbf{rel_x}$, as follows:

$$
\begin{aligned}
&1/|dist(q_p, q_n, Q)|*1/(1\text{-}sign(dist(q_p, q_n, Q)), \text{ if } \mathbf{rel_x} := rel_x(q_p, q_n)\\
&1/|dist(t, termL(rel_z), Q)|*1/(1\text{-}sign(dist(t, termL(rel_z), Q)),\\
OrderImportance(\mathbf{rel_x},Q) = \{ &\qquad\qquad \text{if } \mathbf{rel_x} := rel_x(t, rel_z)\\
&1/|dist(t, termR(rel_z), Q)|*1/(1\text{-}sign(dist(t, termL(rel_z), Q)),\\
&\qquad\qquad \text{if } \mathbf{rel_x} := rel_x(rel_z, t)\\
&1/|dist(termR(rel_y), termL(rel_z), Q)|*1/(1\text{-}sign(dist(termR(rel_y),\\
&termL(rel_z), Q)) \qquad\qquad \text{if } \mathbf{rel_x} := rel_x(rel_y, rel_z)
\end{aligned}
$$

where:

$\mathbf{rel_x} := rel_x(q_p, q_n)$ depicts that relation $\mathbf{rel_x}$ is defined between two query terms,
$dist(q_p, q_n, Q)$ represents the distance between q_p and q_n in query Q. E.g. *dist*("system", "reengineering", "system quality reengineering") = 2. Note that *dist*(a, b, Q) > 0 for a<>b.

$$termL(rel_z) = \begin{cases} q & \text{if } rel_z := rel_x(q, rel_y) \\ termL(rel_v) & \text{if } rel_z := rel_x(rel_y, q) \text{ or } rel_z := rel_x(rel_v, rel_y) \end{cases}$$

$$termR(rel_z) = \begin{cases} q & \text{if } rel_z := rel_x(rel_y, q) \\ termR(rel_y) & \text{if } rel_z := rel_x(rel_v, rel_y) \text{ or } rel_z := rel_x(q, rel_y) \end{cases}$$

$$\text{sign}(a) = \begin{cases} 0, & \text{for } a \geq 0 \\ -1 & \text{for } a < 0 \end{cases}$$

For example, *OrderImportance*(*Specialize*("reengineering", "system"), "system quality reengineering") = (1/2)*(1/(1-(-1))) =1/4, whereas the first factor (1/2) corresponds to the distance between terms and the second factor (1/(1-(-1))) express the inverse order of query terms in the relation with respect to the query.

Note that, regarding complexity, this operation can be very easily performed.

Finally, by taking into account all the above mentioned factors the plausibility of an interpretation (query model) is a combination of the (i) support (number of instances) of its relations (i.e. all parts of the query model), (ii) the type of relations and (iii) the order-importance for each relation:

$$Plausibility(\mathbf{rel_1}(q_1, \mathbf{rel_2}(q_2, \ldots \mathbf{rel_n}(q_n, q_{n+1})\ldots)), Q) = \prod_{i=1,n} Card(RelationSet(\mathbf{rel_i}))*RelImportance(\text{Type}(\mathbf{rel_i}))*OrderImportance(\mathbf{rel_i}, Q),$$

where:

Card(a) is the function that retrieves the number of elements in set *a*,
RelImportance represents the strength of defining a meaning by using a type of relations. Obviously, the *Specialize* relation is the strongest one.

$$RelImportance(\text{Type}(\mathbf{rel_i})) = \begin{cases} p & \text{for Type}(\mathbf{rel_i}) = Specialize \\ q & \text{for Type}(\mathbf{rel_i}) = Modify \\ r & \text{for Type}(\mathbf{rel_i}) = Cooccurrence \end{cases}$$

where:
$p > q > r$. In experiments we use p =3, q=2, r=1
Type($\mathbf{rel_i}$) returns the type ({*Specialize*, *Modify*, *Cooccurrence*}) of the relation $\mathbf{rel_i}$

Therefore, *Plausibility* represents a kind of estimate that a query model corresponds to the user's information need expressed in a query. Moreover, *Plausibility* can be used for assessing the gap between the user's information need and the corresponding query: if the query model that corresponds to the information need has low plausibility (relatively to other query models from the same query) then the user fails to express his need in a clear manner in the given query. It means that a lot of irrelevant resources are retrieved in the retrieval process, i.e. the precision of the retrieval is rather low. In this situation, the system should make suggestions for modifying parts of the query in order to meet the user's information need.

Note, that plausibility includes the structure of the query and the nature of the relations as well. For example, if a query model is based on the *Cooccurrence* relation, it can be less plausible than the one based on the *Specialize* relation, even if it has a higher number of appearances. Therefore, a kind of semantics arises from this combination of statistic and structural information.

The set of query models for a query is generated automatically, whereas the level of the plausibility is defined for each of them. Regarding complexity, note that just one logic query of the above mentioned form (1), is required to determine all query models for a given order of variables and the plausibility values for all of them. Therefore, for a query of 4 query terms, 48 queries in total are required to complete this calculation.

Note that, in order to decrease the number of queries, some further parameterization of the query is possible. However, since we are dealing with short queries, such optimizations are not really necessary.

6.4.1.3 Advantages of Using the Conceptual Query Model

The conceptual query model enables several additional analyses of the retrieval process, which we summarize briefly:

1) Quality of a query:

- Although for a query some results are retrieved, the "quality" of the query can be estimated as being "low" and this can trigger a refinement process. For example, the ambiguity of the query can be estimated very easily: The number of query model determines the ambiguity.

2) Quality of results:

- This approach enables automatic and on-the-fly evaluation of the retrieval process. If the intended query model has a low *Plausibility* then this querying process has a low precision regarding the user's information need - i.e. lots of results are irrelevant for the user's need. It means that is very likely that the top ranked results, actually the results that will be most likely viewed by the user, are not highly relevant for his need. Consequently, the user should modify his query and the refinement system should give some hints. Note, that in the traditional approaches such an evaluation at query-time is impossible, since there is no additional layer (the conceptual layer) where the evaluation should take place – a search engine retrieves all the results it finds relevant so that the user has to evaluate the quality of results on his own. Since our conceptual model reflects the semantic relevance of a query, this human-like evaluation can be simulated.

3) Inclusion of the user's cognitive space:

- Query refinement is a very personalized process since it is related directly to the discovery of the user's information need[69]. It means that the process should take into account as many indicators as possible of that need. In all the existing query refinement approaches the refinement process is oriented towards inspecting the information space in order to find what the most plausible refinement can be. In that way the user's cognitive space is completely neglected. The conceptual query model exactly reflects the cognitive space. It gives hints about the way in which the user's cognitive space is structured in the domain of the given query. This is the most

[69] Information retrieval deals indirectly with the user's information need, since the search process returns resources that are relevant, in the first sense, to a query. On the other hand, the query must not be highly relevant for the corresponding user's information need.

important information for achieving a high precision in a retrieval process, since it helps us avoid misinterpretations of a query. Information that can be also discussed is why the user has chosen a particular query term instead of a similar one. This information can be used as a negative example.

6.4.2 Phase 2: Refinements Derivation

The main advantage of building the query's conceptual model is the possibility to reason about the refinement process, i.e. to define refinements not on the basis of the (syntactic) co-occurrence between terms, but rather on their semantic co-acting. In that way, the refinement can be defined as an improvement of the query's meaning and not only as a pure syntactical change in the query.

The main assumption is that the user is not familiar with the information repository and he needs help to explore it. Note that even if the user exactly knows that he is interested in "solution subgraphs", the definition of such a request is on the syntax level, i.e. the traditional keyword-based query "solution-subgraph", does not enable precise search since the resources about "solution direct subgraph" will not be found. By deriving the meaning and representing it conceptually in the refinement process (i.e. *Specialize*(solution, subgraph)) such syntactical varieties will be abstracted. This is one of the very important advantages of using conceptual query models.

Moreover, the quantification of the plausibility of query models enables reasoning not only on the possible refinements, but moreover on the role that these refinements might have with respect to the meaning of the query. It means that each added term has well-defined relations to other query terms. Consequently, each possible refinement can be quantified according to the "increase" in the meaning of the query, i.e. how that refinement contributes to reducing the ambiguity of the query.

One of the main problems for the traditional query refinement approaches is that they produce a lot of refinements of which a (large) part is completely irrelevant for the query. In order to resolve this problem, our approach uses a reasoning process in order to guarantee the relevance of the generated refinements, i.e. for each refinement there is a logical explanation why it is generated and which role it plays for clarifying the meaning of the query.

Since the elementary building block for defining meaning is a relation between two terms $\mathbf{rel_x}(q_a, q_c)$, we define the refinement process on relations. More specifically, we have defined two inference patterns that drive the refinement process: *termRefinement* and *relationRefinement*, whereas *termRefinement* refines a term as an argument of a relation and *relationRefinement* refines a relation as a whole (i.e. $\mathbf{rel_x}(q_a, q_c)$).

For the first pattern, *termRefinement*, there are two refinement forms, where each of them corresponds to a combination of the term and a new query term:

$$termRefinement(q_a): \quad 1.\ \mathbf{rel_{xx}}(q_a, XX) \qquad 2.\ \mathbf{rel_{xx}}(XX, q_a)\ ,$$

where $\mathbf{rel_{xx}}$ is a new relation, XX is a new query term.

This pattern represents narrowing the interpretation of a query by specifying more details about the context given by query terms, like

Modify("algorithm", "subgraph") ---->

Modify("algorithm", *Specialize*("solution", "subgraph")).

The natural language interpretation is: "algorithm *regarding* subgraph" is refined into "algorithm *regarding* solution subgraph".

For the second pattern, *relationRefinement*, there are two refinement forms, whereas each of them corresponds to a combination of the relation and a new query term:

relationRefinement($\mathbf{rel_x}(q_a, q_c)$): 1. $\mathbf{rel_{xx}}(XX, \mathbf{rel_x}(q_a, q_c))$ 2. $\mathbf{rel_{xx}}(\mathbf{rel_x}(q_a, q_c), XX)$.

It represents narrowing the interpretation of a query by extending the context in which the query terms are considered, like the following refinement:

Specialize("solution", "subgraph") --->

Modify("algorithm", *Specialize*("solution", "subgraph")).

The natural language interpretation is: "solution subgraph" is refined into "algorithm *regarding* solution subgraph".

The formal description of these refinement patterns looks like:

termRefinement(q_a):=
FORALL q_x, $\mathbf{rel_{xx}}$ <- (q_a[inRelation -> $\mathbf{rel_{xx}}$] and $\mathbf{rel_{xx}}$[withTerms -> q_x]) or (q_x[inRelation -> $\mathbf{rel_{xx}}$] and $\mathbf{rel_{xx}}$[withTerms -> q_a])
relationRefinement($\mathbf{rel_x}$):=
FORALL q_x, $\mathbf{rel_{xx}}$ <- ($\mathbf{rel_{xx}}$[fromRelation -> $\mathbf{rel_x}$] and $\mathbf{rel_{xx}}$[withTerms -> q_x]) or ($\mathbf{rel_{xx}}$[toRelation -> $\mathbf{rel_x}$] and q_x[inRelation-> $\mathbf{rel_{xx}}$])

Therefore, for an elementary relation $\mathbf{rel_x}(q_a, q_c)$ there are six refinements (two for each entity (two query terms + one relation term) = six refinements).

$\mathbf{rel_x}(q_a, q_c)$ -> 1. $\mathbf{rel_x}(\mathbf{rel_{xx}}(q_a, XX), q_c)$ 2. $\mathbf{rel_x}(q_a, \mathbf{rel_{xx}}(q_c, XX))$ 3. $\mathbf{rel_x}(\mathbf{rel_{xx}}(XX, q_a), q_c)$
4. $\mathbf{rel_x}(q_a, \mathbf{rel_{xx}}(XX, q_c))$ 5. $\mathbf{rel_{xx}}(XX, \mathbf{rel_x}(q_a, q_c))$ 6. $\mathbf{rel_{xx}}(\mathbf{rel_x}(q_a, q_c), XX)$

Moreover, for a query that consists of n query terms and m relations there are (m+n)*2 refinements in total. Since the number of query terms in a real scenario is small, the complexity of this process is not critical.

Finally, if a refined query term (e.g. q_a) or relation term (i.e. $\mathbf{rel_x}$) are in the scope of an other relation (i.e. $\mathbf{rel_y}(q_y, \mathbf{rel_x}(q_a, q_b))$) then the newly generated relation (i.e. $\mathbf{rel_{xx}}$) should be connected to the existing relation that includes the refined term as an argument. For the logic/declarative expression it means that one additional statement has to be generated for each refined term. For the term x_a which in the original query plays the role of an argument of the relation $\mathbf{rel_y}$, we distinguish two cases of such connectors:

if x_a plays the role of the first argument in $\mathbf{rel_y}$ i.e. ($\mathbf{rel_y}(x_a, X)$): $\mathbf{rel_y}$[fromRelation -> $\mathbf{rel_{xx}}$]
if x_a plays the role of the second argument in $\mathbf{rel_y}$ i.e. ($\mathbf{rel_y}(X, x_a)$): $\mathbf{rel_y}$[toRelation -> $\mathbf{rel_{xx}}$]

The complete query for inferring all possible refinements for the query model $\mathbf{rel_x}(q_1, q_2)$ looks like:

FORALL $\mathbf{rel_{xx}}$, $\mathbf{rel_{x1}}$, $\mathbf{rel_{x2}}$, $\mathbf{rel_{y1}}$, $\mathbf{rel_{y2}}$, $\mathbf{rel_{z1}}$, $\mathbf{rel_{z2}}$, q_{x1}, q_{x2}, q_{y1}, q_{y2}, q_{z1}, q_{z2}, <-
(q_2[inRelation -> $\mathbf{rel_{xx}}$] and $\mathbf{rel_{xx}}$[withTerm -> q_1] and $\mathbf{rel_{xx}}$:$\mathbf{rel_x}$) and
(($\mathbf{rel_{x1}}$[withTerm -> q_{x1}] and $\mathbf{rel_{x1}}$[toRelation -> $\mathbf{rel_{xx}}$]) or
($\mathbf{rel_{x2}}$[fromRelation -> $\mathbf{rel_{xx}}$] and q_{x2}[inRelation -> $\mathbf{rel_{x2}}$]) or (4)
($\mathbf{rel_{xx}}$[fromRelation -> $\mathbf{rel_{y1}}$] and $\mathbf{rel_{y1}}$[withTerm -> q_{y1}] and q_1[inRelation -> $\mathbf{rel_{y1}}$) or
($\mathbf{rel_{xx}}$[fromRelation -> $\mathbf{rel_{y2}}$] and $\mathbf{rel_{y2}}$[withTerm -> q_1] and q_{y2}[inRelation -> $\mathbf{rel_{y2}}$]) or
($\mathbf{rel_{xx}}$[toRelation -> $\mathbf{rel_{z2}}$] and $\mathbf{rel_{z2}}$[withTerm -> q_2] and q_{z2}[inRelation -> $\mathbf{rel_{z2}}$]) or
($\mathbf{rel_{xx}}$[toRelation -> $\mathbf{rel_{z1}}$] and $\mathbf{rel_{z1}}$[withTerm -> q_{z1}] and q_2[inRelation -> $\mathbf{rel_{z1}}$])).

Figure 6.15 illustrates this refinement process. The structure represented in Figure 6.15 can be treated as a derivation pattern that will be multiplicated in the queries with more than two query terms.

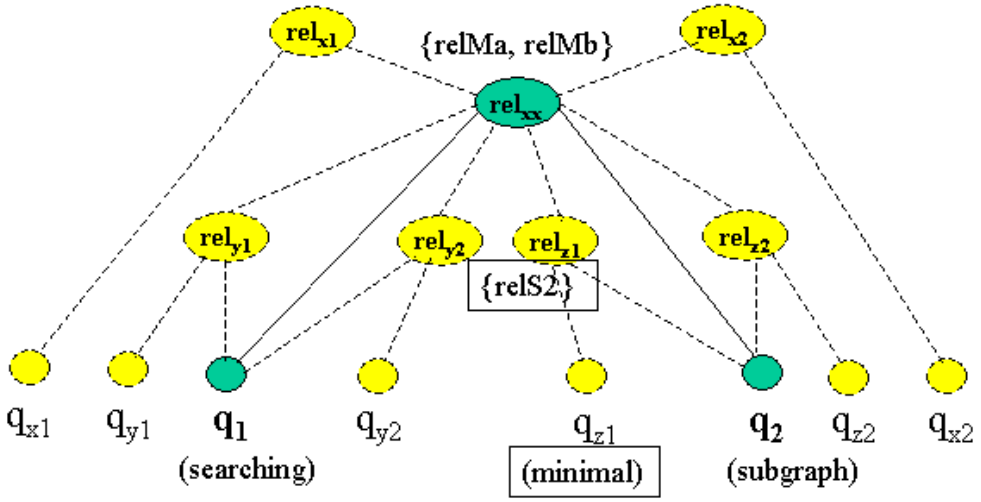


Figure 6.15: A complete set of refinements for query model $rel_x(q_1, q_2)$. The process of the derivation of the refinement "minimal" for the query ("search", "subgraph") for the situation represented in Figure 6.9. The relation relS2 and concept "minimally" satisfy the last condition (i.e. last row) in (4).

In order to demonstrate the pattern-based nature of the refinements, Figure 6.16 represents the complete set of refinements for query model $\mathbf{rel_x}(q_1, \mathbf{rel_y}(q_2, q_3))$. Since the query model consists of two relation terms and three query terms there are ten, (2+3)*2, refinements in total.

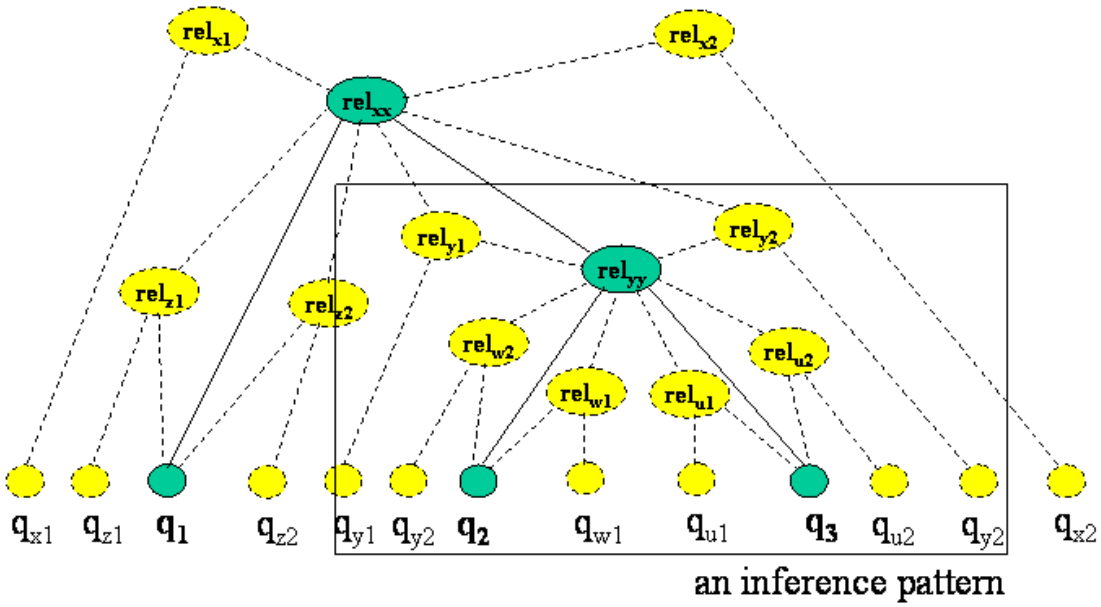


Figure 6.16: A complete set of refinements for query model $rel_x(q_1, rel_y(q_2, q_3))$. Inference pattern represents the basic calculation for two terms that is subsequently repeated for more query terms.

6.4.3 Phase 3: Ranking of Refinements

As we have already mentioned, the user's query can be mapped into several query models and each query model can produce a lot of refinements. Therefore, ranking of the query models and the refinements should be performed in order to increase the effectiveness of the proposed

method. Indeed, as we have already described, for each query model a level of the confidence with the user's information need is expressed through the *Plausibility* factor.

In Chapter 3 we have already introduced the notion of *Relevanc*e, that consists of *Semantic* and *Personal* relevance. Since *Personal* relevance is related to a user's interaction with the results, which we described for a general case in section 3.2.2.3.2, we give here only the calculation of *Semantic* Relevance.

Note that as in the previous section we distinguish between the term and the relation refinement.

As described in section 3.2.2.3.1, *Semantic* Relevance depends on (a) the level of the disambiguation of the meaning of a query that is achieved by introducing a refinement and (b) the relatedness between a refinement and the rest of the query:

(a) In this case the disambiguation is calculated as the increase in the disambiguation of the term/relation that is directly refined. This parameter (*TermInterpretabilityRatio*, *RelationInterpretabilityRatio*, corresponds to the term and relation refinement, respectively) represents the ratio between the number of relations of a type that include a query term (i.e. relation) and the number of these relations when the query refinement term is added. Obviously, the larger the ratio, the larger the increase in meaning is.

(b) The relatedness between a refinement and the rest of the query depends on the:

(i) plausibility of the given query model (evaluated through the function *Plausability*),

(ii) the strength of the relation that defines the minimal context for the refinement (evaluated trough *RelationSet*),

(iii) the type of the relation that is refined (measured using *RelImportance*) and

(iv) the type of the relation in which a new term is added (measured using *RelImportance*).

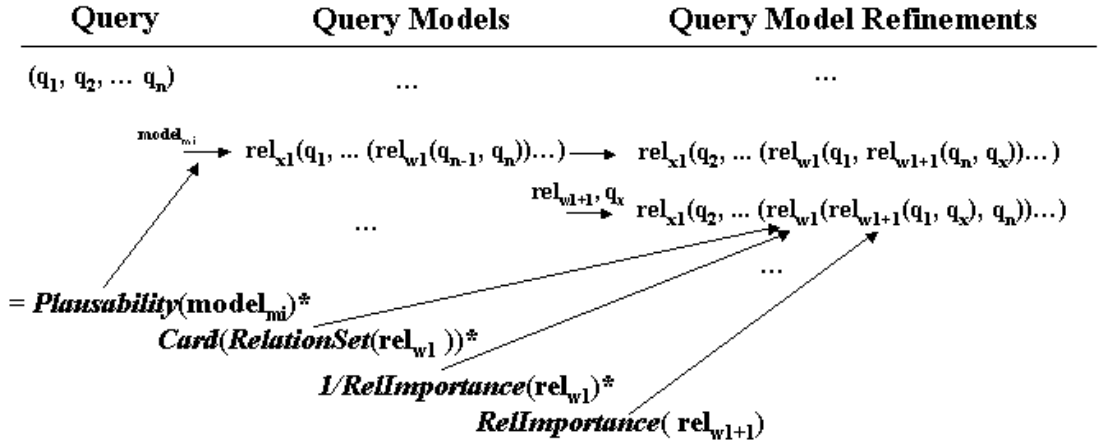


Figure 6.17: Factors that influence the relatedness of a refinement to the query.

Formally, the ***Relevance*** of a query model qm_i:

for refinement $\mathbf{q_x}$,
that has **pos** position,
in a relation type $\mathbf{rel_{xt}} \in \{Specialize, Modify, Cooccurrence\}$,
to relation $\mathbf{rel_{ia}}$ or term q_{ia},
which is an argument of relation $\mathbf{rel_i}$,

is assessed through improving/clarifying the meaning of a query in the following way:

1) For term q_{ia} (term refinement)

SemanticRelevance(qm_i, $\mathbf{rel_i}$, $\mathbf{rel_{xt}}$,q_{ia}, q_x, pos) =

a**TermInterpretabilityRatio*(q_{ia}, q_x, $\mathbf{rel_i}$, $\mathbf{rel_{xt}}$, pos)

+ b**Plausability*(qm_i)*(*Card*(*RelationSet*($\mathbf{rel_i}$))*

(*1/RelImportance*(Type($\mathbf{rel_i}$)))* *RelImportance*($\mathbf{rel_{xt}}$))

where:

pos ∈{first, second} defines the position refinement q_x has in the new relation generated in the query,

a and *b* are weighting parameters (by default: *a* = *b* =1).

TermInterpretabilityRatio(q_{ia}, q_x, $\mathbf{rel_i}$, $\mathbf{rel_{xt}}$, pos) =

TermInterpretabilityInRelation(q_{ia}, q_x, $\mathbf{rel_i}$, $\mathbf{rel_{xt}}$, pos)/

TermInterpretabilityInRelationTotal(q_{ia}, $\mathbf{rel_i}$, $\mathbf{rel_{xt}}$, pos)

TermInterpretabilityInRelation(t, $term_x$, rel_x, *r_type*, pos) = |{x| x∈*TermInterpretability*(t, $term_x$, *r_type*, pos) ∧ ∃ r∈ *RelationSet*(rel_x) ∧ (*inRelation*($term_x$, r) ∨ *withTerm*(r, $term_x$))}|

TermInterpretabilityInRelationTotal(t, rel_x, *r_type*, pos) = |{x| x∈*TermInterpretabilityTotal*(t, *r_type,* pos) ∧ ∀y∈T ∃r∈*RelationSet*(rel_x) ∧ (*inRelation*(y, r) ∨ *withTerm*(r, y))}|

TermInterpretabilityInRelation(t_1, t_{x1}, rel_x, rel_1, pos) represents the number of the relations of type rel_x between t_1 and t_{x1} that are in the scope of relation rel_1 that exists in the original query. *TermInterpretabilityInRelationTotal*(t_1, rel_x, rel_1, pos) is the total number of the relations of type rel_x between t_1 and *any other term* that are in the "scope" of relation rel_1.

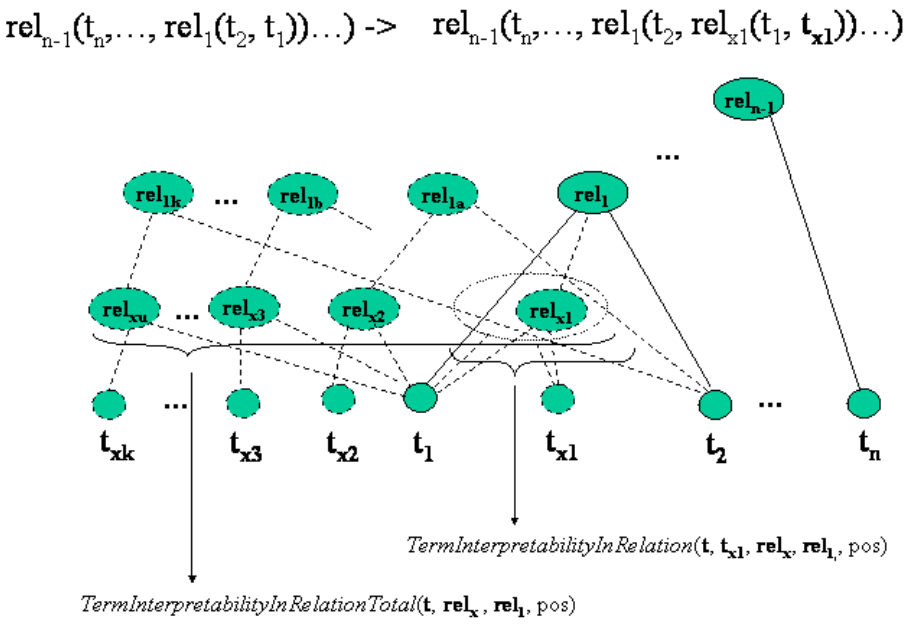


Figure 6.18: An illustration of the calculation of the relevance of a query term for refinement: $\mathbf{rel_{n-1}}(t_n,\ldots, \mathbf{rel_1}(t_2, t_1))\ldots)$ into $\mathbf{rel_{n-1}}(t_n,\ldots, \mathbf{rel_1}(t_2, \mathbf{rel_{x1}}(t_1, t_{x1}))\ldots)$. Note that t_{x1} and rel_{x1} are new added entities in the refined query model.

This value shows how the introduction of a new term constrains the ambiguity of an existing relation. For example, if the query model is *Modify*("knowledge", "graph"), then for the

refinement *Modify*("knowledge", *Specialize*("solution", "graph")), the *TermInterpretabilityRatio* represents the ratio between the frequency of the appearance of the "solution-graphs" in the context of "knowledge" and the frequency of the appearance of all kinds of graphs (e.g. "problem", "cyclic", …) in the context of "knowledge". In that way we can measure how specific "management of solution subgraphs" w.r.t. the "management of other types of graphs" is. This specificity is used as one of the factors for determining the relevance of the proposed refinement. Figure 6.18 illustrates these calculations.

2) For relation $\mathbf{rel_{ia}}$ (relation refinement):

$SemanticRelevance(qm_i, \mathbf{rel_i}, \mathbf{rel_{xt}}, \mathbf{rel_{ia}}, q_x, pos) =$

$a*RelationInterpretabilityRatio(\mathbf{rel_{ia}}, q_x, \mathbf{rel_i}, \mathbf{rel_{xt}}, pos)$
$+ \quad b*Plausability(qm_i)*(Card(RelationSet(\mathbf{rel_i}))* \quad (1/RelImportance(\text{Type}(\mathbf{rel_i})))*$
$(RelImportance(\mathbf{rel_{xt}}))$

where:

$RelationInterpretabilityRatio(\mathbf{rel_{ia}},q_x, \mathbf{rel_i}, \mathbf{rel_{xt}}, pos) =$
$RelationInterpretabilityInRelation(\mathbf{rel_i},q_x,\mathbf{rel_i}, \mathbf{rel_{xt}}, pos)/$
$RelationInterpretabilityInRelationTotal(\mathbf{rel_{ia}}, \mathbf{rel_i}, \mathbf{rel_{xt}}, pos)$

$RelationInterpretabilityInRelation(rel_{ia}, term_x, rel_x, r\_type, pos) =$
$|\{x| \; x \in RelationInterpretability(rel_{ia}, term_x, r\_type, pos) \wedge \exists \; r \in RelationSet(rel_x) \wedge$
$(inRelation(term_x, r) \vee withTerm(r, term_x))\}|$

$RelationInterpretabilityInRelationTotal(rel_{ia}, rel_x, r\_type, pos) =$
$|\{x| \; x \in TermInterpretabilityTotal(rel_{ia}, r\_type, pos) \wedge \forall y \in T \; \exists \; r \in RelationSet(rel_x) \wedge$
$(inRelation(y, r) \vee withTerm(r, y))\}|$

The explanation of these functions is very similar to the one presented above. Figure 6.19 illustrates these functions.

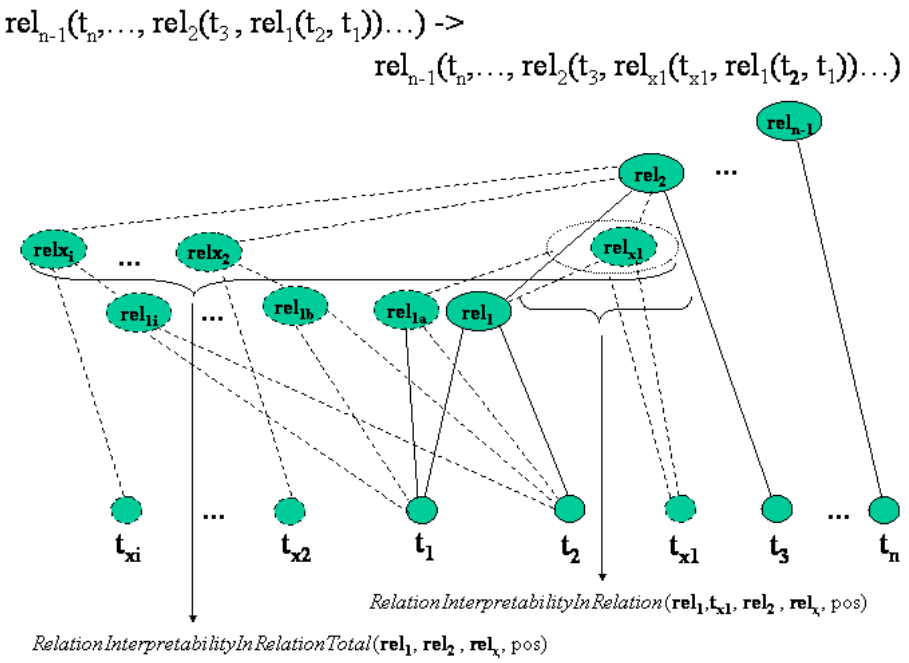


Figure 6.19: An illustration of the calculation of the relevance of a relation for the refinement

Regarding complexity, note that in this step no inference is performed. Moreover, all data is already pre-calculated in the previous two phases of the refinement process. In order to make this calculation even more feasible, we take only the first n (n<10) query models that correspond to the most plausible interpretations of the query, i.e. that have the biggest values for the *Plausibility* parameter.

Note that beside the entire query model refinements, individual refinement terms (i.e. just a list of terms) can be presented to the user. The relevance of a refinement is calculated as the sum of all relevancies of that refinement in all possible query models. Although in that case the interpretation of the query is lost, the semantics of the query refinement process remains, i.e. only semantically related refinements are generated. Moreover, the minimal number of elements of the list can be ensured as well, i.e. only refinements that are not subsumed by another refinement are presented to the user, generating the neighborhood of the query, as already explained in Chapter 3.

6.4.4 Iterative Refinement

The presented refinement process is organized as a step-by-step refinement process, which means that in each step one additional query term is added. Moreover, the process is iterative, whereas the next refinement step takes the query context from the previous one. It means that the particular query model that is refined in the previous step is favored in the next refinement step. For example, if the query ("quality", "system") was refined with the refinement *Modify*(*Specialize*("quality", "system"), "improvement"), then for this new query ("quality" "system", "improvement") the model *Modify*(*Specialize*("quality", "system"), "improvement") is treated as the most plausible one. Since this refinement process should be suitable for any traditional, keyword-based search engine, the retrieval process cannot incorporate the query model directly. Moreover, since it is possible that the user changes slightly his information need during the search process, we retain this possibility by generating all possible query models for the new query, whereas the ranking of these models (i.e. calculating plausibility) includes the information about the query model that is the output from the previous refinement step. Figure 6.20 presents this situation.

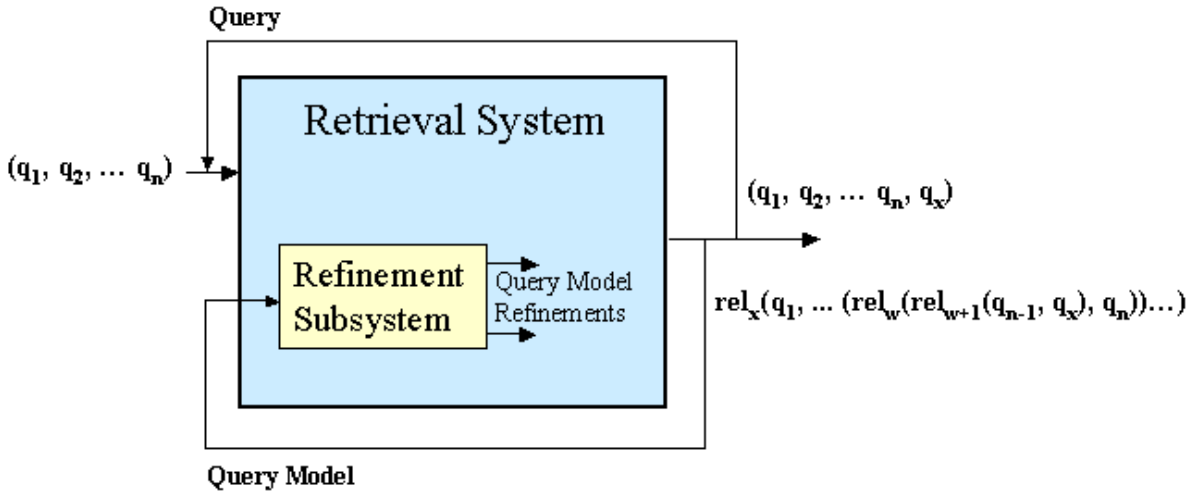


Figure 6.20: Schema of the iterative refinement process

Note that information about the user's feedback can be very easily incorporated in the retrieval process by using the methods we have described in Chapter 3.

6.5 Adding Lexical Knowledge to the Refinement Process

6.5.1 Introduction

The main idea in the conceptual query refinement is to derive (infer) semantics from statistical and structural information, whereas the structure is determined through a shallow NLP. However, due to the usage of various vocabularies (e.g. usage of different descriptions for the same real thing), valuable statistical information is lost. For example, regarding query refinement, the terms "personalization" and "customization" could be treated as identical, so that the refinements derived from both of them can be aggregated, which consequently increases the plausibility of a refinement. A valuable source for such relations can be, e.g. Wordnet[70].

However, past research regarding the usage of lexical information in the query refinement has shown that a "trivial" usage of Wordnet fails to achieve significant improvements (if any) [155], mainly due to a decrease in the retrieval's precision. We will call such an approach *syntax-based*. For example, if each term from the query is expanded with all known synonyms regarding Wordnet, it is possible that some of the synonyms cause the so-called meaning drift and consequently the precision decreases. The main reason for such results is a syntactical usage of semantic structures. Indeed, a synonym from the Wordnet can be used as a substitute for the original query term only, if it plays a similar role, from the semantics point of view, in the user's query.

Since our logic-based approach operates on the level of the meaning of a query, the selection of the synonyms can be performed on that level as well. In that way only the lexical varieties that are relevant for the user's information need will be included in the statistics regarding a query model, i.e. taken into consideration for emerging (deriving) semantics. Such an approach we will call *semantic-based* lexical approach.

We see two possibilities to include synonyms into the conceptual query refinement: in the process of building query models and in the process of inferring refinements. In the next two subsections we elaborate more on these extensions.

6.5.1.1 Term Extensions

The meaning (interpretation) of a query term can be defined through the relations with other terms. It is clear that the number of such relations indicates diversity in the interpretation of a query term. In section 6.4.1.2.1 we defined the functions, *TermInterpretability* and *TermInterpretabilityTotal* to assess this interpretability.

However, as we have already mentioned, the relatedness between two terms can be defined on a more abstract level, i.e. through the definition of the lexicographic relations between terms independently of the concrete repository. Indeed, on the level of a concrete language (e.g. English) several relations that indicate common interpretations of two terms can be established, like synonym, homonym, and hyponym. From the query refinement point of view, the synonym relation seems to be the most useful one, since it denotes a similar (identical) meaning for two terms. We present here the extension of formulas for calculation of the term interpretability, given in section 6.4.1.2.1, with the treatment of synonyms as follows:

$$TermInterpretability(t, term_x, r\_type, first) = \{x \mid r\_type(x) \wedge (inRelation(t, x) \vee (\exists s \in \text{Synonym}(t) \wedge inRelation(s, x))) \wedge ((withTerm(x, term_x) \vee (\exists term_s \in \text{Synonym}(term_x) \wedge withTerm(x, term_s)))\} \quad (5)$$

[70] http://www.cogsci.princeton.edu/~wn/

$TermInterpretability(t, term_x, r\_type, second) = \{x \mid r\_type(x) \wedge (withTerm(x, t) \vee (\exists s \in \text{Synonym}(t)) \wedge withTerm(x,s))) \wedge (inRelation(term_x, x) \vee (\exists term_s \in \text{Synonym}(term_x) inRelation(term_s, x)))\}$

where the function *Synonym*(*a*) retrieves a list of synonyms of the term *a* (using Wordnet).

Formulas for calculating the total interoperability can be extended in an analogous way.

Figure 6.21 illustrates this calculation. It is clear that this approach can be very easily extended for other linguistic relations (e.g. hyponym).

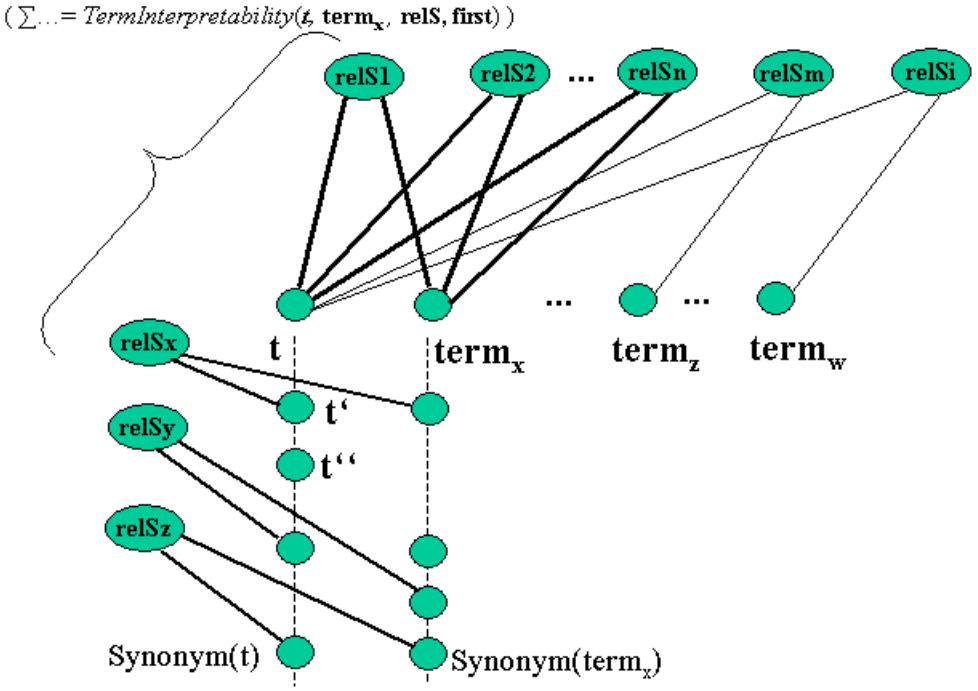


Figure 6.21: Extension of the calculation related to the interpretation of a term (in this case term *t*). The main issue is to substitute a term with its synonyms that are in similar relations with other query terms. In that way a drift of meaning is avoided.

The goal of such a calculation is to aggregate evidence for similar interpretations of a query. Consequently, such interpretations will appear as the most reliable interpretations of the meaning of the query, which affects the process of deriving refinements. Indeed, the refinements that correspond to the most reliable meaning of the query (i.e. the refinements that refine these meanings) are treated as the most reliable refinements, i.e. they are closest to the initial user's information need.

The crucial issue in our approach is that we use linguistic knowledge to support the reasoning on the conceptual level. In other words, we are not interested in the synonyms per se, but in their interpretations regarding the concrete information repository. If the interpretations of a query term drift from the meaning of other synonyms, these interpretations will be treated as a minority and will not influence the refinement process. In that way the drift of the query meaning is avoided.

Consequently, the synonyms of a query term that do not appear in the context of other query terms will be treated as irrelevant, i.e. they will not satisfy the conditions required by (1). Regarding Figure 6.21, if we assume that all relations between synonyms are presented in the Figure, then synonym **t''** of the term t will be treated as irrelevant for the user's information need, since it does not appear in the context, in which term **t** appears in the query. Note that the usage of synonyms that are irrelevant for the query is a crucial problem in the traditional

approaches, as we have mentioned above. Our reasoning on the conceptual level enables a very elegant elimination of these useless synonyms.

6.5.1.2 Refinements Aggregation

The process of deriving refinements is based on a set of logical axioms that explore the search space in order to find terms (candidates) that can improve the meaning of a query.

However, problems arise if some of the candidates derived in the refinement process have a related meaning (are synonyms) and play the same role in a refinement. For example, regarding Figure 6.22, the refinement "smallest" is obtained in the same way (for the same variable: q_{z1}) as the refinement "minimal" and by assuming that these two terms are synonyms (like in Wordnet), their effects on the refinement process can be aggregated. Consequently, such a refinement has a higher support and can be of a higher rank.

The complete query for inferring all possible refinements for the query model $\mathbf{rel_x}(q_1, q_2)$, including the synonym relation, is:

FORALL $\mathbf{rel_{xx}}$, $\mathbf{rel_{x1}}$, $\mathbf{rel_{x2}}$, $\mathbf{rel_{y1}}$, $\mathbf{rel_{y2}}$, $\mathbf{rel_{z1}}$, $\mathbf{rel_{z2}}$, q_{x1}, q_{x2}, q_{y1}, q_{y2}, q_{z1}, q_{z2}, <-

(q_1[inRelation -> $\mathbf{rel_{xx}}$] and $\mathbf{rel_{xx}}$[withTerm -> q_2] and $\mathbf{rel_{xx}}$:$\mathbf{rel_x}$) and

((q_{x1}[inRelation -> $\mathbf{rel_{x1}}$] or (exist q_s and *belongs*(q_s, Synonym(q_{x1})) and q_s[inRelation -> $\mathbf{rel_{x1}}$]) and $\mathbf{rel_{x1}}$[toRelation -> $\mathbf{rel_{xx}}$]) or

($\mathbf{rel_{x2}}$[fromRelation -> $\mathbf{rel_{xx}}$] and (exist q_s and *belongs*(q_s, Synonym(q_{x2})) and q_s[inRelation -> $\mathbf{rel_{x2}}$]) or $\mathbf{rel_{x2}}$[withTerm -> q_{x2}]) or (6)

($\mathbf{rel_{xx}}$[fromRelation -> $\mathbf{rel_{y1}}$] and (q_{y1}[inRelation -> $\mathbf{rel_{y1}}$] or (exist q_s and *belongs*(q_s, Synonym(q_{y1})) and q_s[inRelation -> $\mathbf{rel_{y1}}$]) and $\mathbf{rel_{y1}}$[withTerm -> q_1]) or

($\mathbf{rel_{xx}}$[fromRelation -> $\mathbf{rel_{y2}}$] and q_1[inRelation -> $\mathbf{rel_{y2}}$] and ($\mathbf{rel_{y2}}$[withTerm -> q_{y2}]) or (exist q_s and *belongs*(q_s, Synonym(q_{y2})) and q_s[inRelation -> $\mathbf{rel_{x1}}$]) or

($\mathbf{rel_{xx}}$[toRelation -> $\mathbf{rel_{z2}}$] and (q_1[inRelation -> $\mathbf{rel_{z2}}$] and ($\mathbf{rel_{z2}}$[withTerm -> q_{z2}] or (exist q_s and *belongs*(q_s, Synonym(q_{z2})) and q_s[inRelation -> $\mathbf{rel_{z2}}$])) or

($\mathbf{rel_{xx}}$[toRelation -> $\mathbf{rel_{z1}}$] and (q_{z1}[inRelation -> $\mathbf{rel_{z1}}$] or (exist q_s and *belongs*(q_s, Synonym(q_{z1})) and q_s[inRelation -> $\mathbf{rel_{z1}}$]) and $\mathbf{rel_{z1}}$[withTerm -> q_1]).

where *belongs*(*el*, *Set*) is a function that returns true if *el* is an element of set *Set*. In the logical implementation of the system Ontobroker, which we have used in this research, this predicate is implemented as a built-in. An important factor in using inference mechanism to calculate these refinements is that the relation Synonym is defined as a transitive relation.

Note that it is not enough that two refinements' candidates are synonyms in order to aggregate their impacts on the refinement. Moreover, they have to play the same role in that refinement, i.e. they should clarify the meaning of the query in the same way. For example, regarding Figure 6.22 and (6), the refinement candidate "minimum" can be treated as a synonym of the candidate "minimal", but is derived as the refinement q_{y1} (i.e. it clarifies the meaning of the query term "search" as presented in Figure 6.22). It does not interfere with the refinement "minimal" in that context, since "minimal" clarifies the meaning of the query term "subgraph". In other words, the query models that are derived in these two refinements are different.

Finally, the usage of the lexical knowledge can be very effective in cooperative answering [131], especially in the case when the user is not familiar with the problem domain and creates "non-optimal" queries, in which some of the query terms should be replaced in order to express the user's information need in the right way.

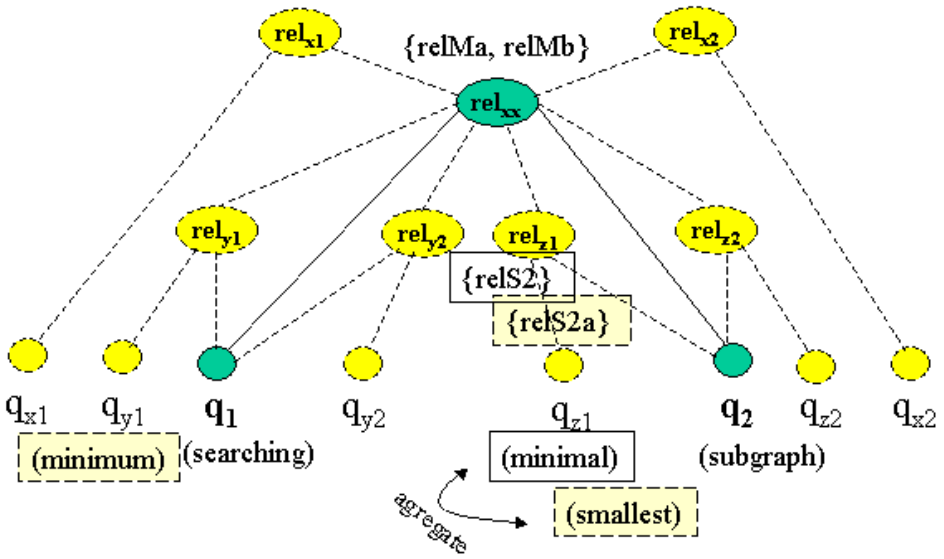


Figure 6.22: Usage of synonyms for the derivation of refinements.

Figure 6.22 illustrates the usage of synonyms. It is related to Figure 6.9 and the results from Wordnet show that the terms "minimal", "smallest" and "minimum" are synonyms. Since "minimal" and "smallest" play the same role in the model (specialization of the term "subgraph"), they are treated as one refinement, i.e. their interpretation results are aggregated. The term "minimum" plays a different role (i.e. a specialization of the term "search") and is not treated as a synonym regarding the refinement process.

6.6 Support for Cooperative Answering

The introduction of a conceptual layer in the query refinement process enables much more powerful reasoning services than just the derivation of refinements. Indeed, the conceptual layer explicates the meaning of a query and formalizes relations between the query and terms from the repository, so that various aspects of the refinement process can be formally analyzed, like (i) which part of the query has a "clear" meaning and which is the most ambiguous one, (ii) the explanation why a refinement is generated, and, (iii) which part of the query is best clarified by a refinement, to name but a few. Consequently, it enables moving the focus of the refinement process from modifying just a query (e.g. by adding a new query term) to modifying the meaning of the query (by extending the conceptual/semantic model of the query).

Therefore, this modeling paradigm enables us to go one step forward and to treat the refinement problem more deeply, namely at the level of a user's information need. Indeed, as we have already elaborated, a query is just an approximation of an information need and, therefore, a refinement process should take a more flexible view on a query in order to enable the user to find more relevant results for his information need. Since our query refinement approach operates on the conceptual query model that represents the meaning of a query (and as such, it models the user's information need), we can talk, instead of the query refinement, about the *refinement of the user's information need*. Such a service that helps the user to change his need according to a particular situation in an information repository can be treated as Cooperative Answering, which we have described in Chapter 3. Two types of cooperative answers have been defined there:

1) Failing query resolution.

> An empty answer is surprising to the user since the user expects that there exist answers to the query. When a query fails, a system could be more cooperative by

helping to trace the reason for the query's failure, or at least to pinpoint to the failure.

2) Alternative results.

An alternative result for a query is a result that does not fulfill all the constraints from the query perfectly (but fulfills some of them), but has some suitable features from the user's need point of view. However, the problem is how to find those constraints that can be relaxed in the user's query and those that cannot.

Since for resolving from failing queries we have defined in Chapter 3 a general procedure that can be directly applied in this case, in this section we elaborate on deriving alternative results only.

A task regarding a real world situation (e.g. a business process) can be interpreted and conceptualized in the user's cognitive space in various ways, depending on, e.g. the user's background knowledge. Therefore, it is possible that a query, which results from such a task's conceptualization, is not "well prepared" for the underlying information repository, whereas "not well prepared" means that there are some indicators about the existence of a better conceptualization of the task. Note that it does not mean that the query can be better defined (e.g. to add some new terms), but rather that the information need could be better defined, i.e. a new, semantically modified query could be generated. Obviously, this modified query has to correspond to the task specification.

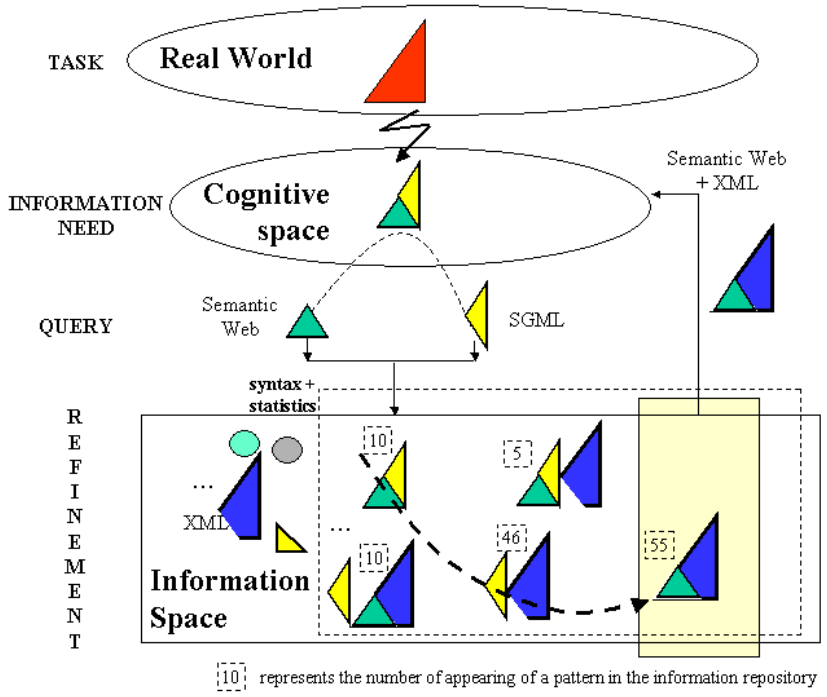


Figure 6.23: Conceptual description of the refinement of an information need

We have found two strong indicators that the user has made a non-optimal conceptualization of his task even though the system has retrieved some results:

1) An increase in the number of the interpretations, i.e. when the replacement of a query term with another term, by retaining the model of the query, increases the number of interpretations.

Let us illustrate this situation on the example of the query "Semantic Web, SGML", presented in Figure 6.23. If we assume that the user's task at hand is related to the markup of Semantic Web documents, a possible interpretation of the problem can be a need for information

resources about "Semantic Web" and SGML. However, let us assume that the relationship between these two concepts is not very strong in the underlying information repository, i.e. there are just a few instances of this relation. On the other hand, there is a strong relation between the concepts "Semantic Web" and XML that builds a similar interpretation of the task, i.e. the term XML appears in a similar context, regarding the term Semantic Web, as the term SGML. Therefore, it is possible that the user, due to, for example, a lack of background knowledge, has generated an inappropriate information need and he specifies a suboptimal query.

Therefore, the main indicator is that the most likely interpretations of a query have a weak plausibility. Since it is hard to define weak plausibility, we use an experimentally derived heuristic that a particular query model is weakly plausible if the number of interpretations of that query model is less than fifty percent of the average number of interpretation of a query model for that repository.

2) The second indicator comes from the cognitive space: If the most plausible query models do not correspond to the order of the original query terms, then this might be an indicator that the user's (need) query is misinterpreted in the information repository.

For example, if the user specifies a query "quality system expert" and the most plausible models are about "expert system" and "quality", which obviously does not correspond to the order of query terms, then some suggestions could be given to the user about how to modify his need, like "quality improvement expert". Obviously, the term "improvement" has to appear in similar contexts as the terms "quality" and "expert".

Further, the search space for the corresponding refinements can be very large, i.e. it is difficult to determine which of the query terms to refine. We constrain this space by focusing on query terms that introduce a non-confidence that the information need corresponds well to the real world situation (i.e. the task at hand). However, since a query term appears always in a relation with other terms or relations, the main task is to find the relation that has a very low support. The following function, that measures the change in the support of each relation in a query model, (Card(*RelationSet*($\mathbf{rel_x}$)) - Card(*RelationSet*($\mathbf{rel_y}$))), is used for determining the weak points (relations) in a query model:

$$
\textit{Support}(\mathbf{rel_x}) = \begin{cases} \textit{Card}(\textit{RelationSet}(\mathbf{rel_x})) & \text{if } \mathbf{rel_x} := rel_x(q_y, q_z) \\ \textit{Card}(\textit{RelationSet}(\mathbf{rel_x})) - \textit{Card}(\textit{RelationSet}(\mathbf{rel_y})) & \text{if } \mathbf{rel_x} := \mathbf{rel_x}(q_x, \mathbf{rel_y}) \text{ or } \mathbf{rel_x}(\mathbf{rel_y}, q_x) \\ \textit{Min}(\textit{Card}(\textit{RelationSet}(\mathbf{rel_x})) - \textit{Card}(\textit{RelationSet}(\mathbf{rel_y})), \textit{Card}(\textit{RelationSet}(\mathbf{rel_x})) - \textit{Card}(\textit{RelationSet}(\mathbf{rel_z}))) & \text{if } \mathbf{rel_x} := rel_x(rel_y, rel_z) \end{cases}
$$

Finally, in query model $\mathbf{rel_1}(q_1, \mathbf{rel_2}(q_2, \ldots \mathbf{rel_i}(\mathbf{rel_{i+1}}(q_i, q_{i+1}), \mathbf{rel_{i+2}}(q_{i+2}, \ldots, \mathbf{rel_n}(q_n, q_{n+1})\ldots))$, all relations $\mathbf{rel_x}$ for which *Support*($\mathbf{rel_x}$) < threshold (it is set manually) should be processed for refinement.

The set of possible refinements for relation $\mathbf{rel_x}$, from this set can be inferred by using the following rules:

for $\mathbf{rel_x}$**:=** $rel_x(q_y, q_z)$,

$$\forall\, x\ \exists \mathbf{rel_{1x}}, \mathbf{rel_{1x}}{:}\text{Type}(\mathbf{rel_1}) \wedge \ldots \wedge \mathbf{rel_{xx}} \wedge \mathbf{rel_{xx}}{:}\text{Type}(\mathbf{rel_x}) \wedge (\mathbf{rel_{xx}}(q_y, x) \vee \mathbf{rel_{xx}}(x, q_z)) \wedge \mathbf{rel_{(x-1)x}}(q_{(x-1)x}, \mathbf{rel_{xx}}) \wedge \ldots \wedge \mathbf{rel_{1x}}(q_1, \mathbf{rel_{2x}})$$

for $\mathbf{rel_x}$**:=** $rel_x(rel_{x+1}, q_z)$

$\forall$ x $\exists$**rel$_{1x}$**, **rel$_{1x}$**:Type(**rel$_1$**) $\wedge$ … $\wedge$ **rel$_{xx}$** $\wedge$ **rel$_{xx}$**:Type(**rel$_x$**) $\wedge$ **rel$_{xx}$**(**rel$_{x+1}$**, x) $\wedge$ **rel$_{(x-1)x}$**($q_{(x-1)x}$, **rel$_{xx}$**) $\wedge$ …$\wedge$ **rel$_{1x}$**(q_1, **rel$_{2x}$**)

for **rel$_x$:=** $rel_x(rel_{x+1}, rel_{x+2})$

$\forall$ x $\exists$**rel$_{1x}$**, **rel$_{1x}$**:Type(**rel$_1$**) $\wedge$ … $\wedge$ **rel$_{(x+2)x}$** $\wedge$ **rel$_{(x+2)x}$**:Type(**rel$_{x+2}$**) $\wedge$ **rel$_{(x+2)x}$**(x, **rel$_{x+3}$**) $\wedge$ **rel$_{xx}$**(**rel$_{(x+2)x}$**, **rel$_{x+1}$**) $\wedge$ … $\wedge$ **rel$_{1x}$**(q_1, **rel$_{2x}$**)

and Figure 6.25 illustrate the first and second calculations, respectively. On the top of Figure 6.24 is the original query, whereas at the bottom is the alternative query. Term x can replace query term q_n, since it has the same types of relations with q_{n+1} and q_{n-1} like q_n. If the support for these relations is strong, then x can be considered as a candidate for the query refinement, i.e. for the generation of an alternative answer. "Type" represents the type of a relation, that can be {*Specialize, Modify, Cooccurrence*}.

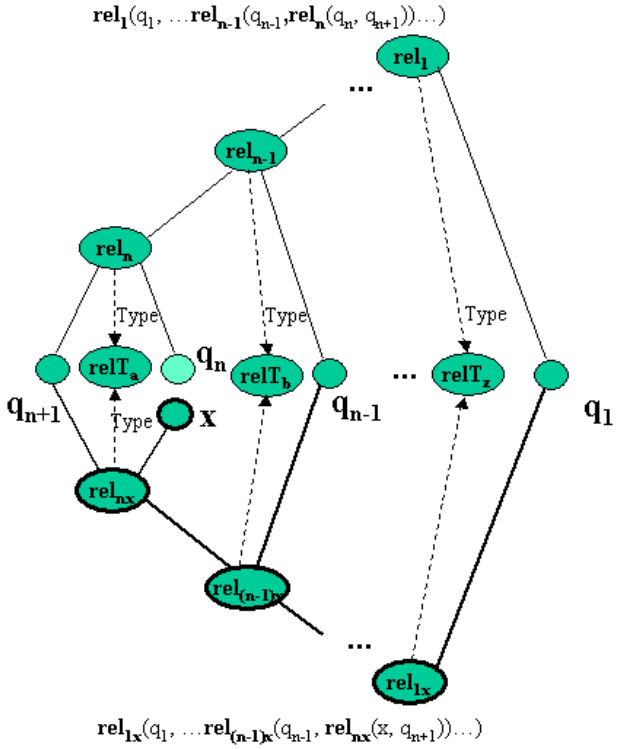


Figure 6.24: An example of calculating alternative queries

Actually, the main drawback that can be found in traditional systems is: They evaluate the query as a black box. If we assume that the user can clarify the relation between two terms in an easier way than in a more complex structure (e.g. three terms), then a direct relation with a very large support can be an indicator of the user's intention, independently of the support for the whole query. Indeed, what if, regarding the given example, the user is interested in "quality improvement expert" but the term "system" in the query "quality system expert" was a wrong choice? The system should mark that as a potential explanation of the query's ambiguity and as a possible refinement for generating an alternative query. Therefore, in order to take into account this issue, all parts of a query model should be included in estimating the plausibility of the query's meaning.

This service can be treated as a semantic "Did You Mean" service[71] in which the user's query is semantically analyzed in order to find possible improvements w.r.t. the user's information need. Therefore, since this service "changes" the information need of the user, we call it a

[71] "Did you mean" is the service provided by the Google web search engine, which enables the correction of the lexicographical (syntax) errors, like that of "Smantic Web" into "Semantic Web".

"Do You Need" service. As our case study illustrates, this service is very useful in situations where, due to unfamiliarity with the retrieval task, the user specifies a "wrong" query, so that the traditional query expansion methods only introduce an additional confusion into the retrieval process.

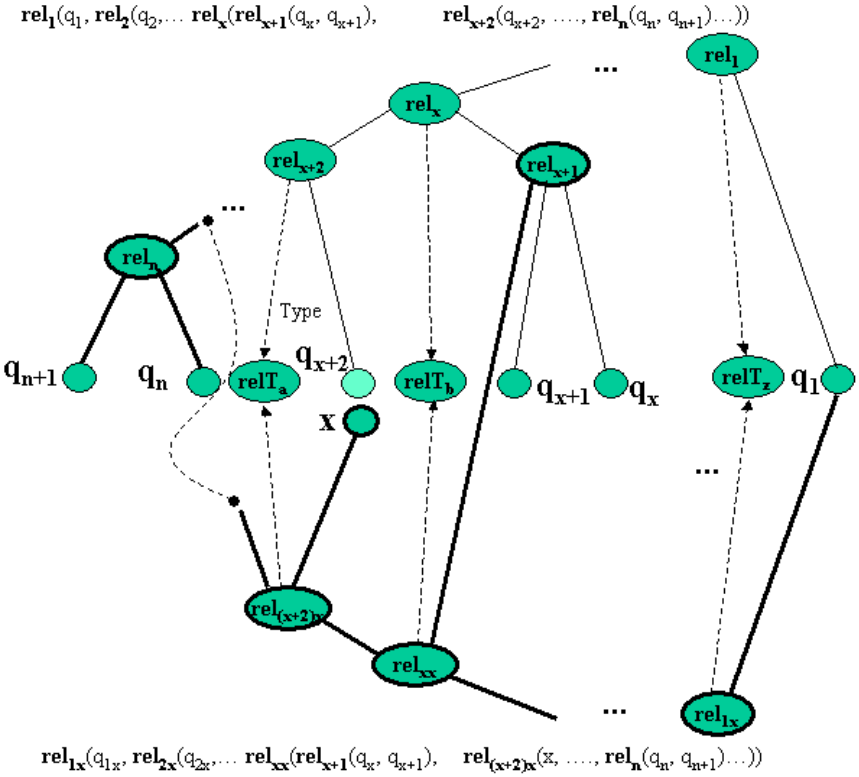


Figure 6.25: An example of calculating alternative queries.Term x replaces query term q_{x+2} whereas the structure of the query, i.e. the relations between all other query terms, remains

Note that this service includes both the dimensions of an information retrieval process:

a) By analyzing the information space it can discover that the user's need cannot be reflected well in the information repository.

b) By analyzing the cognitive space it can discover that the user's need is misinterpreted in the querying process.

Query refinement is a very personalized process since it is related directly to the discovery of the user's information need. It means that the process should take into account as many indicators of that need as possible. In all the existing query refinement approaches the refinement process is oriented towards inspecting the information space in order to find what the most plausible refinement can be. In that way the user's cognitive space is completely neglected. The two issues mentioned above reflect the cognitive space. They give hints about how the user's cognitive space is structured in the domain of the given query. This is the most important information for achieving a high precision in the retrieval process, since it helps to avoid misinterpretations of a query.

6.7 Case Study

In this section we present a case study regarding the bibliographic search we have done in the scope of the ***SemIPort***[72] project. *CompuScience*[73] is a bibliographic database covering

[72] SemIPort (http://km.aifb.uni-karlsruhe.de/semiport/) is a Semantic Web related project, funded by the BMBF, German Ministry for Education and Science, whose task is the development of semantic methods for information portals.
[73] http://www.fiz-informationsdienste.de/en/DB/compusci/index.html

literature in the field of computer science, information- and communication technology, information management and science with about 160.000 citations. Citations are in English and contain bibliographic information and indexing terms. Many records also include an abstract. The citations are classified according to the Computing Reviews Classification Scheme of the ACM.

6.7.1 Architecture

Figure 6.26 represents the conceptual architecture of the retrieval system used in the case study.

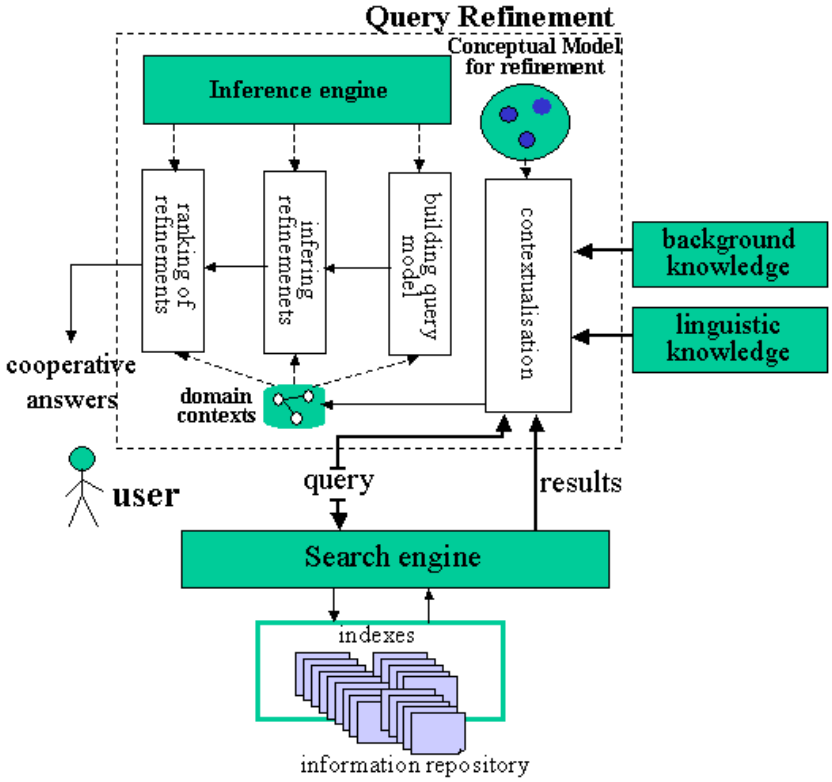


Figure 6.26: Conceptual architecture of the query refinement system

The building blocks correspond to three phases in the librarian agent query refinement process. The contextualization module has the task to process results retrieved by the search engine and to build an initial set of facts (i.e. the initial domain context) that are used in the refinement process. In particular, as a linguistic parser we use Text2Onto, and Ontobroker is used as an inference engine. The background linguistic knowledge is taken from WordNet. For performing search the full-text search engine Lucene[74] is used. Note that we have already given some details about the complexity of the approach. In the case of short queries more than 75% of the computing time goes to the natural language processing, in particular search for noun phrases.

Figure 6.27 is a screenshot from an official Information Portal for Computer Science in Germany (www.io-port.net) where the presented approach is used for query refinement ("Verfeinerte Anfragen" – in German). The refinement process can be applied subsequently, whereas the next refinement is based upon the discovered query model in the current refinement step. The modifications and specializations lines in Figure 6.27 contain the refinement derived from the *Modify* and *Specialize* relations respectively. Note that the

[74] http://jakarta.apache.org/lucene/docs/index.html

semantics of both lines is different. For example in modifications there is a refinement "code problem", which is interpreted as "problems *regarding* code". In specializations, there is a refinement "code binary", which is interpreted as "binary code"

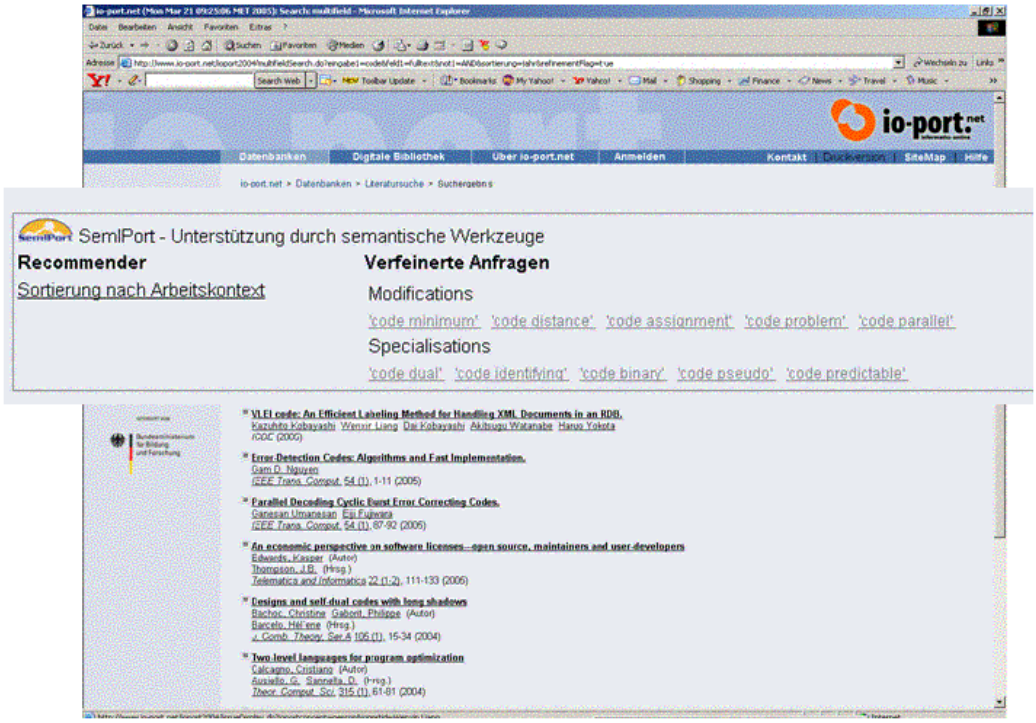


Figure 6.27: A screenshot from a running system where the presented approach is applied

6.7.2 Evaluation

In order to prove the usability and validity of our approach we have performed several evaluation studies (1) proving the quality of the presented approach, (2) its comparison with the standard approaches for query refinement and (3) evaluation of the role of the lexical knowledge on the quality of the retrieval process (including time and correctness).

Figure 6.28 presents a screenshot from the system used in the evaluation.

6.7.2.1 First evaluation study

The main strength of our approach is the quality of the provided refinements - the refinements we provide should be very relevant for the user's current information need. Here we present one of the evaluation studies we have performed in order to prove this claim.

Although the standard approach to inspect usefulness of generated refinements is to rely on the user's judging about the relevance of provided refinements, in order to be as objective as possible, we have performed a user-driven study in which the relevance of the list of refinements is determined objectively. Namely, at the end of each query session (the user has selected a relevant document) we have compared a list of refinements our approach has generated with the content of the document(s) selected by a user. The comparison is performed by three domain experts, who have evaluated the similarity between the first 10 ranked results and the corresponding document. For each refinement the mark is composed as *level_of_relevance/ranking_level*, where *level_of_relevance* is the value an expert assigned to a refinement (1-10) and *ranking_level* represents the position in the refinement list.

We have selected 15 participants, who were PhD students in Computer Science. Each of them has made 10 queries related to the research he is familiar with. No additional instructions are given to them.

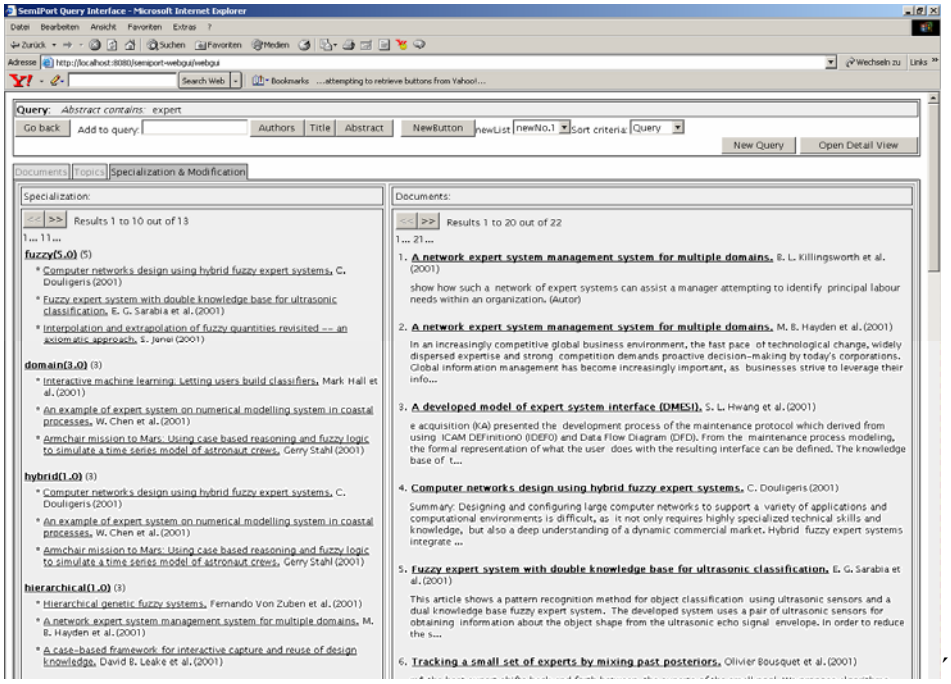


Figure 6.28: A screenshot from a demo system where the presented approach is applied. The *Modify* and *Specialize* types of the refinement are presented in the left pane (only specialization is visible). In the right pane the list of result for the given query is presented

The results are presented in the Table 6.1. They demonstrate a high relevance, regarding the user's information need, for the most relevant refinements.

Table 6.1: Results of the first evaluation study. Precision@x denotes that the first x ranked results are considered. The results are normalized on the interval [0,1]

Precision@X	Precision@10	Precision@5	Precision@3
Value	0.85	0.925	0.98

The complex combination of parameters used for determining relevance ensures high precision of the generated refinements. Similar to the analyses we gave in Chapter 4, we performed an evaluation of the importance of the relevance factors we have defined in section 6.4.3 for the total relevance. The main finding is that the plausability of a query model determines very strongly the quality of the ranking process. It can be expected, by considering the fact that plausability defines, in fact, the meaning of a user's information need. Table 6.2 summarizes these results.

Table 6.2: Results of the relevance evaluation study. Values in Table represent the decrease in the Presicion@x by eliminating a parameter

Parameter	Plausability	RelationSet	RelImportance	Interpretability
Presicion@10	20%	11%	12%	15%
Presicion@5	13%	3%	5%	8%
Presicion@3	7%	2%	2%	4%

6.7.2.2 Second evaluation study

The second experiment is a classical user-driven study, in which we have compared our approach with the traditional query refinement approach, namely, an interactive query expansion (IQE) where the refinements are calculated by standard F_4 term weighting function [43]. Since it is difficult to define in advance a set of relevant resources for a query regarding the given repository, we have set-up an experiment in which each participant has to perform an unsupervised search process. More precisely, each participant (the same setting as in the previous case) has to choose ten tasks on his own and to perform half of them by using the IQE query interface and another half by using our query refinement support. For each task a participant is expected to find five relevant results[75]. In order to take into account the quality of the selected relevant results, each participant has to express his *confidence* in these results. The confidence describes the participant's certainty that the selected results are the best possible ones (i.e. that there is no a better result for his need). This is measured on the scale 1 - 4, whereas 4 means maximal confidence. Besides confidence, we have measured the *length of a query session* (number of querying steps) and the *duration of a query session* for each query.

The results of the second experiment are presented in Table 6.3.

Table 6.3: Results of the second evaluation study

Querying Method	Session Duration (in sec.)	Session Length (num. of steps)	Confidence
Our approach	37,28	5,58	3.23
IQE	57,30	3,87	2.26

Discussion:

Although the search supported by our approach requires more querying steps for a task (column 3) it is performed faster (column 2). Moreover, the participants are more confident in results found by using the query refinement support in querying (column 4). This means that our approach provides refinements that are very useful (relevant) for a current query, since the user does not spend much time in a querying step. Finally, our approach covers a large part of the search space with such refinements, so that the user is very confident with the results selected as relevant, i.e. he has the feeling that lots of alternatives are taken into account in the querying process. This is a very important feature in the recommender applications – the user should trust the recommendation process. To see if these differences can be considered statistically significant, we have performed a paired t-test for each measure. It does reveal the superiority of our approach with respect to all three parameters ($p < 0.0001$).

6.7.2.3 Third evaluation study

The third evaluation study encompasses several experiments in which we test the impact that lexical knowledge has on the query refinement process.

In the first experiment we compare the approach presented in section 6.5 (Lexical Approach) with the basic conceptual query refinement approach (Basic Approach). Since it is difficult to define in advance a set of relevant resources for a query regarding the given repository, we performed an experiment with the identical setting as the previous experiment (see Table 6.3). Therefore, each participant (the same setting as in the previous case) has to choose ten tasks on his own and to perform half of them by using Lexical Approach and another half by using the Basic Approach. The results of this experiment are presented in Table 6.4.

[75] Each participant determines the relevance in each searching session on his own.

Note that in the case of short queries more than 75% of computation time goes to the NLP pre-processing of results, in particular the search for noun phrases.

Table 6.4: Results of the third evaluation study, first experiment

Querying Method	Session Duration (in sec.)	Session Length (num. of steps)	Confidence
Lexical Approach	43,30	5,58	3.88
Basic Approach	40,80	5,87	3.25

Discussion:

To see if these differences can be considered statistically significant, we have performed a paired t-test for each measure. It does reveal the superiority of our approach (Lexical) with respect to the third, but not for the first and second parameters ($p > 0.0001$). It means that a careful usage of Wordnet results in a better precision, but not in any significant increase in the processing time.

In the second experiment we test the quality of the refinements regarding the way the lexical knowledge is used (see section 6.5.1): a) in the context of other query term as presented in this chapter (semantic-based lexical approach) or b) in isolation (syntax-based lexical approach), as used in the traditional approaches. The setting for the experiment is the same as in the previous experiment. The results of the second experiment are presented in Table 6.5.

Table 6.5: Results of the third evaluation study, second experiment

Querying Method	Session Duration (in sec.)	Session Length (num. of steps)	Confidence
Semantic-based Lexical Approach	46,30	5,88	3.74
Syntax-based Lexical Approach	51,80	6,55	2.85

Discussion:

The difference in the first and third parameter is statistically significant. The confidence in the results (syntax w.r.t. semantic approach) decreases due to a drift in the query meaning. Therefore, any usage of lexical knowledge does not necessary contribute to a more efficient search. It should be semantic-based in order to ensure benefits in the retrieval process.

6.8 Related Work

The use of co-occurrence statistics to automate the discovery of semantic relationships among terms has a long history in information retrieval and computational linguistic [156]. Ruge [106] has applied a corpus of linguistic techniques to the automatic construction of thesauri for the use in information retrieval. Strzalowski [143] has utilized a broader notion of dispersion in a formula to compare the specificity of semantically related terms for the automatic construction of lexical domain maps. The I^3R [150] system provides assistance to the user by suggesting paths that should lead to relevant information. Recommendation is given to guide and not to restrict the user's options. This is reflected by the displays on the neighborhood and context maps, which consist of nodes representing concepts, documents

and journal issues connected by links. More recently, the terminological feedback is used for improving performance of a query refinement system. The Paraphrase Search Assistant [1] exploits the tendency for the key domain concepts within results sets to participate in families of semantically related lexical compounds.

However, our approach differs from the above presented ones by introducing the conceptual level, which is used for an analysis of the user's query. It enables a more efficient refinement process. Moreover, our inference-like search is a unique feature, as well as the cooperativeness enabled by it. Although a lot of research is dedicated to improving the cooperativeness of a database access, almost all of them are focused on resolving the problem of failing queries. If there is not a cause per se for the query's failure, it is then worthwhile to report the part of the query which has failed [53]. Further, some types of query's relaxations [73] are proposed for weakening the user's query in order to allow him to find some relevant results. Our approach operates on the conceptual level so that an alternative querying is possible and very reliable.

The existing methods for query refinement seem to be inadequate for the discovery of the meaning of a query [21], since they usually return a long list of refinements, which is hard to process manually. Indeed, recent experimental study in the interactive query refinement has shown that only one third of the terms derived from document relevance feedback are identified by users as useful for refining their searches [43]. In other words, users are overloaded with refinement information, just as they are overloaded with search results in an information retrieval task. The main cause of the problem is a vague definition of the notion of relevance: it is based on a statistically low chance regarding a set of documents, e.g. a term is relevant for the refinement if it appears frequently in relevant documents. Obviously, such a definition covers only the syntax level of the relevance, since the context in which these terms appears (i.e. their meaning) is not at all treated. Consequently, a lot of irrelevant refinements for the particular user's information need will be generated, since it is possible that a term appears very frequently in relevant documents but not in the context of the query terms. Indeed, the existing query refinement methods take only the document-centered view on the problem, without taking into account query characteristics, i.e. semantic relationships between refinement terms and the user's need (i.e. query). The main advantage of our approach is exactly based on these findings: an explicit inclusion of the user's cognitive space in the refinement process can increase the quality of the refinements drastically.

Note that this principle differs very much from the existing (successful, commercial) approaches that are based on document clustering, where the description of a cluster does not need to be directly related to the meaning of the query. Indeed, the description of a cluster is generated from the text of the retrieved results, without taking explicitly into account the query word. Figure 6.29 represents such a situation for the Velocity meta-search engine (www.vivisimo.com), the most successful clustering engine based on the award-winning Vivísimo clustering technology. For example, for the query "jaguar" we have analyzed the cluster described as "Mark Webber" (6 results): in none of the documents is the word "Mark Webber" frequently related to the concept jaguar directly, but only to Webber's racing performance. In fact, he is a very successful formula one (F1) driver, so that these documents are related to the "jaguar" as well. Therefore, the semantics of the cluster should be not "Mark Weber", but something more general that is related to the "formula one race" in order to help the user develop his information need. Obviously, there could be some users who are interested in Mark Webber's performance, but it is hard to believe that they will start the query with the keyword "jaguar" (especially because he is now racing for BMW).

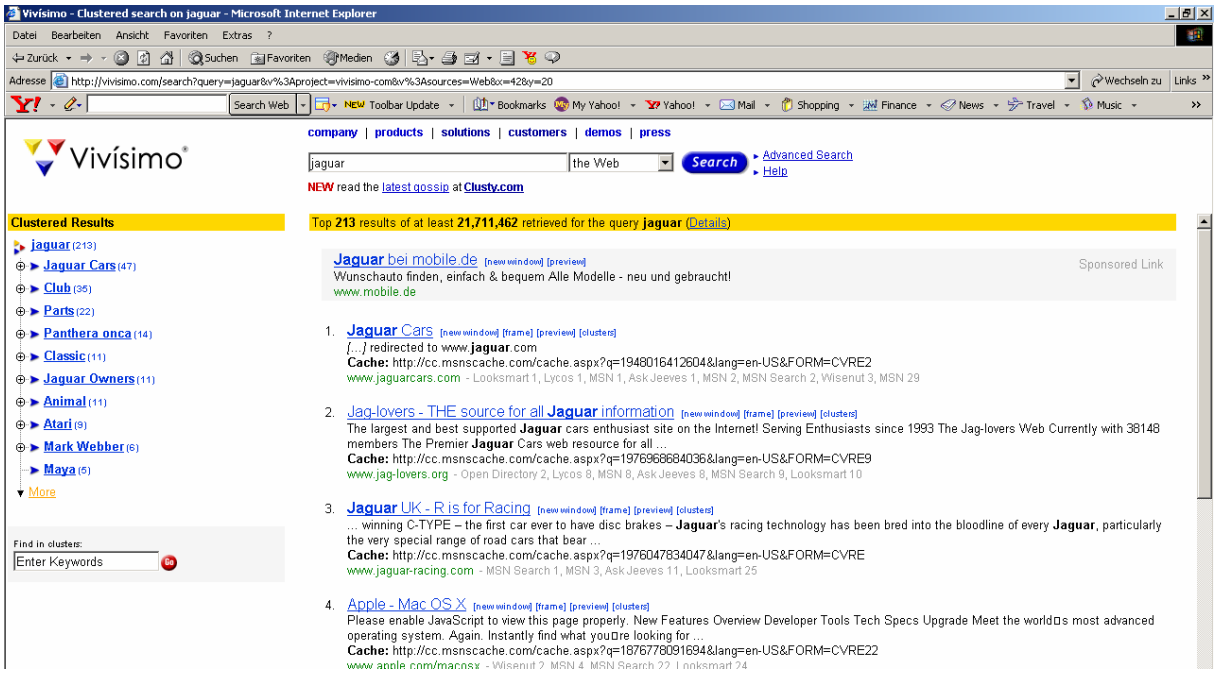


Figure 6.29: Vivisimo's response to the query "Jaguar"

Moreover, due to the low precision of standard web IR methods, some of these results are totally irrelevant for the initial query (i.e. jaguar). Consequently, the clustering methods which do not consider the "value" of the query for performing clustering produce some irrelevant clusters as in the case of Vivisimo.

Although the clustering is done well (but not perfectly since, e.g. the interpretation "airplane" is missing), further clarification of the meaning of the query is purely syntactical. Let us assume that the user is interested in animals – he can click on the cluster "Animal" and extend the query with the term "animal". It means that the query is now "animal and jaguar", but the user's information need is not clarified. Indeed, the query "animal and jaguar" depicts just a constraining of the search space, but not a real clarification of the meaning of the query. Figure 6.30 illustrates this case: the subcluster "Yahooligans" does not tell anything about the relation between animal and jaguar. Indeed, the main problem in existing approaches is that they do not guarantee the meaningfulness of all provided solutions, so that the confidence in the entire approach decreases.

The situation is more illustrative for a query with more than one query term. Let us assume that the user is interested in food that is consumed by jaguars and makes the query "jaguar food". As presented in Figure 6.31, the very first cluster is related to cars. Therefore, clustering does not support discovery of meaning, since the query is not interpreted as a meaningful entity, but rather as a syntactical constraint on the search space.

This is the main drawback of traditional IR: the retrieved documents are about query terms but not about the query as a whole [6].

Finally, the difference between the two interpretations of the query "jaguar food", namely, between (1) the need for resources about what jaguars consumes as food and (2) the need for resources about jaguars as food, is completely impossible.

By designing our refinement system we have taken into account all the above-mentioned problems.

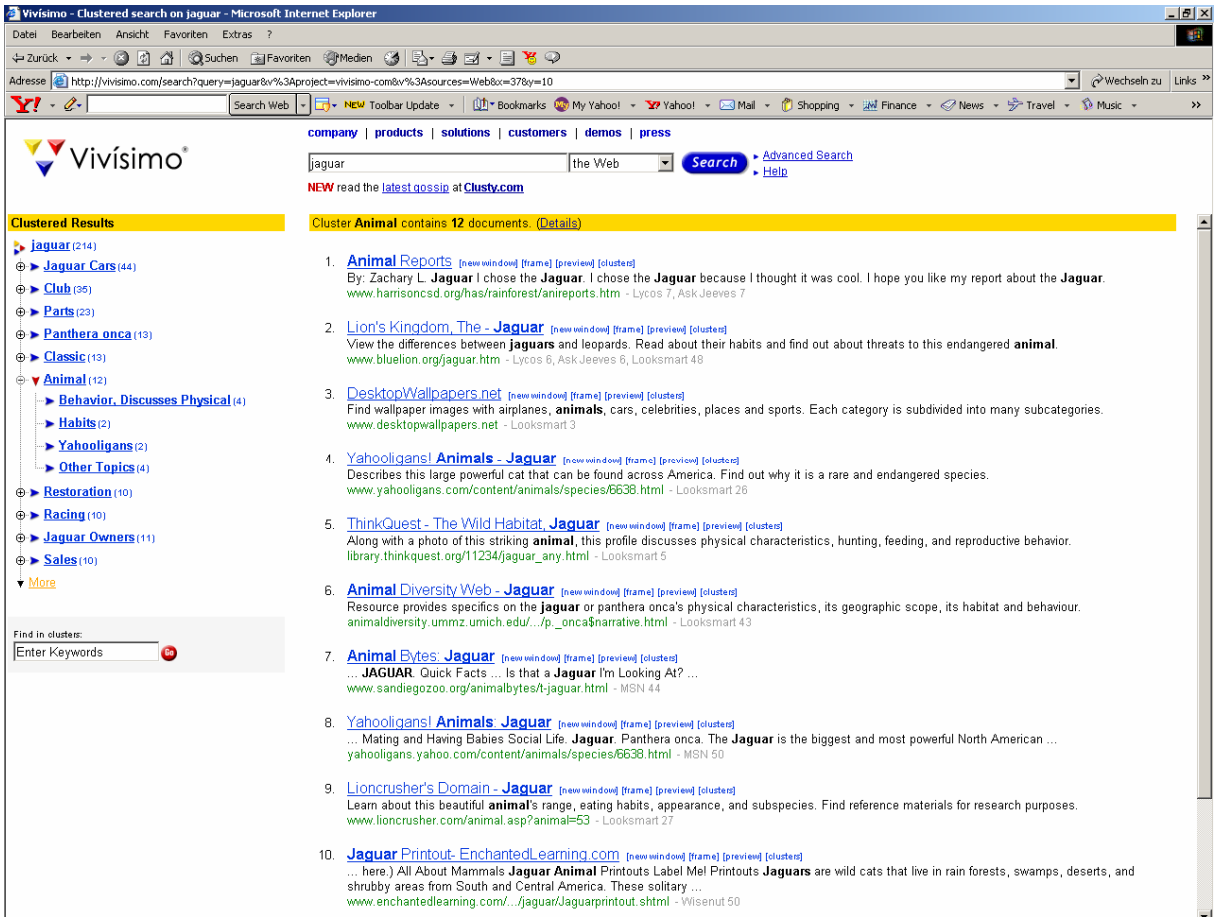


Figure 6.30: Subclusters defined for the cluster "animal" regarding the query "jaguar" (www.vivisiomo.com)

6.9 Conclusion

This chapter presents a novel approach to the refinement of boolean queries by using semantic technologies. We have introduced a conceptual model for defining user's queries, which enables the resolution of query disambiguation on the semantic level, i.e. on the level of the meaning of a query. Consequently, query refinement is performed as an inference process in order to guarantee the relevance of the generated refinements. The main advantage for the user is that the refinements' set contains only refinements whose role in clarifying the meaning of the query is clear, e.g. for each refinement there is a logical explanation why it is generated. Moreover, due their conceptual nature, the refinements are represented as meaningful patterns that can be directly compared to the user's information need. Finally, the approach is based not only on inspecting the mapping of a query into an information repository, but moreover on analyzing the cognitive context of the query, which enables the development of services that go beyond the query refinement and support the refinement of the user's information need. The presented method has been implemented in a running information portal, which illustrates the scalability of the approach.

Figure 6.32 gives an overview of the roles specific for the contextual information (depicted as spaces in Figure), namely, the roles played by real world, cognitive space, information space, linguistic space and formal space in the presented refinement approach.

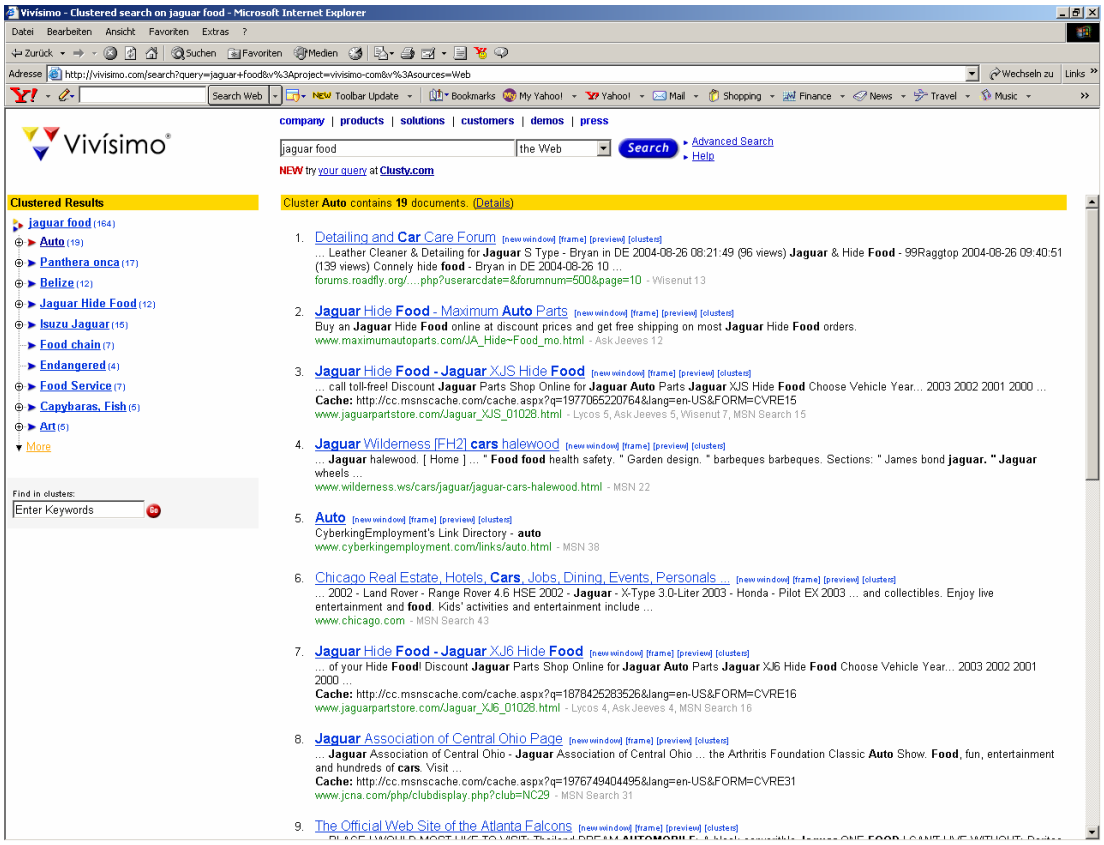


Figure 6.31: Vivisimo's response for the query "Jaguar and Food"

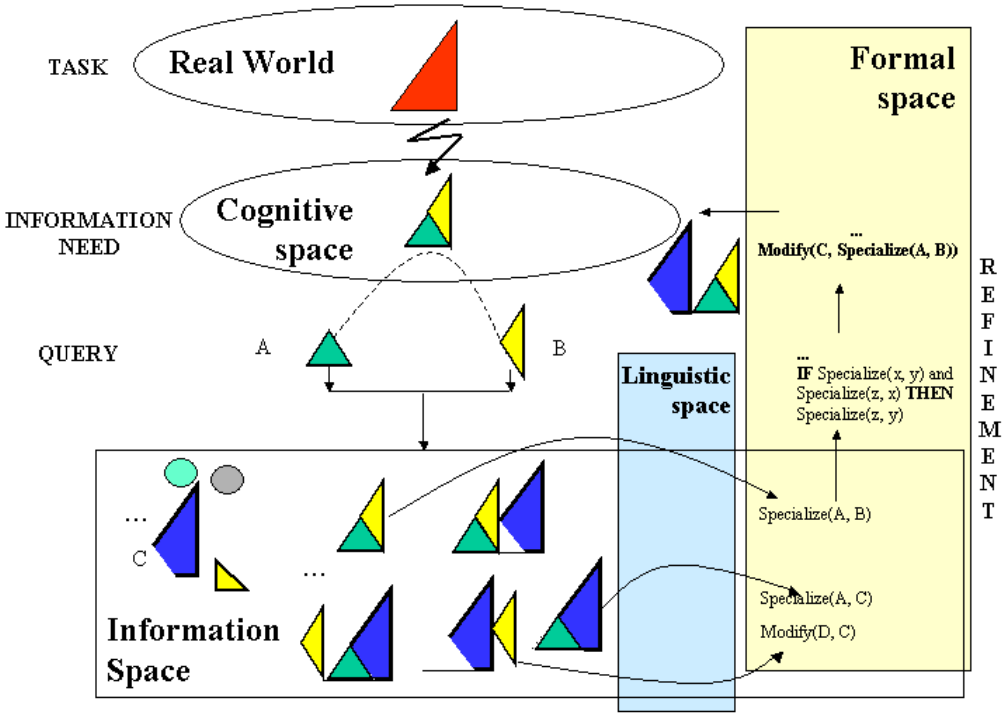


Figure 6.32: The complete view on the conceptual query refinement

The main question in performing research studies like this one concerns the real benefits to be gained from applying more advanced calculation methods. Usually the improvements can be

measured in the % of improvements with respect to some gold standards. However, this research proposes not only a better or more efficient method for calculating refinement, but, moreover, it offers, due to its conceptual orientation, a new perspective to the treatment of the entire problem domain (i.e. query refinement).

First, by introducing a conceptual layer for the interpretation of a query, the query refinement problem stops being a problem of search for the most suitable term that can be added to the query; instead, it focuses upon finding the most suitable interpretation (meaning) of the query. Indeed, our approach adds, not only terms, but, moreover, the structural relations that build the meaning of the query. In that way we can talk, instead of about the refinement of a query, about the refinement of the query's meaning.

Secondly, a set of new refinement services can be defined. A concrete consequence of the new conceptualization of the query refinement problem is the possibility to perform the refinement not on the level of the query, but on the level of the user's information need. Indeed, a query is just an approximation of the user's information need [110] and one of the goals of a query refinement process should be to support the user in redefining what he is actually searching for.

Finally, the introduced conceptual model enables a more efficient (natural) treatment of the lexical knowledge in the query refinement process, as the evaluation study has shown. Indeed, synonyms of a query term are not treated in isolation, but, instead, only in the context of other query terms. In that way we ensure that an unwanted query meaning drift is avoided which is in itself a big challenge for the traditional methods.

This work (re)opens an issue concerning the need for a more semantic-based indexing of documents in traditional information retrieval in order to improve the overall retrieval performance. Our experimental studies have shown that the semantics which has emerged from text documents has enough quality to enable the building of the query refinement methods that outperform the most advanced syntax-driven methods. Consequently, one can imagine a more emergent-semantic based indexing process (partially based on the methods described in this chapter) that will retain the autonomy of web data, but will significantly increase the precision of the retrieval process performed by a traditional search engine.

On the other hand, from the Semantic Web point of view, this work (re)answers the question of the quality/usability of domain ontologies that can be learned from (web) data. From the information retrieval point of view, such an approach is very promising.

7 Conclusion

Due to the proliferation of content production (enterprise- and world-wide), the main concern in the search for information will be how to focus the attention of a user on the most valuable information for his need. Moreover, one very important issue is how information delivery can be embedded into the day-to-day work (and life[76]), especially of knowledge workers[77]. The amount of this "information capital" in organizations has grown at an astronomical rate over the last few years as new digital content is being created and older content is being digitized and saved. Given the lack of time and attention that many knowledge workers face today, no one can expect that they will browse or search through repositories for a while without being quite sure that they will find what they are looking for.

On the other hand, a very competitive business environment pushes the informativeness as one of the main factors for organizational survival. It is clear that we are entering now a knowledge era, which emphasizes intangible assets, especially the role of implicit knowledge. However, it is more than ever clear that possessing the knowledge per se is not enough to ensure competitiveness in long or even medium terms. At least the ever shorter periods of the innovation cycles demand continuous knowledge production and, more importantly, knowledge production in the right business context.

Finally, the efficient search for information has a clear business case. Most enterprises have already recognized the strategic importance of connecting employees to the content they need, as it directly affects their costs, revenues, productivity, and even competitiveness. For example, an employee spends about 2 hours per day in search for information, whereas for the knowledge intensive activities, the spent time and the importance of the (very) relevant information increase exponentially. It is easy to calculate the ROI of just small improvements in speed of search process.

However, the tools to handle this "information overload" have lagged behind. One can say[78] "that traditional enterprise search engines attempt to find a needle in a haystack. They try to guess user's intent from one- or two-word queries and return the best matches for them. Different types of results are mixed together, making the job more difficult. Traditional search is like leaving a user in a room with books piled randomly on the floor."

Although in the last decade the information retrieval research community contributed with lots of interesting methods for search enhancement (like the logic model of information retrieval we have presented in Chapter 2), the principles of indexing and search deployed in real systems did not change substantially over time: resources are indexed with a flat set of keywords, a user makes a keyword-based query and gets a list of results, whereas the presentation is enriched by showing (automatically produced) summaries of results, or relevant passages from results, or even (syntactically defined) clusters of results. Question answering is another alternative for search, but, however, the current state of practice is far away from the results that an average user is expecting[79].

[76] Consider e.g. the role of information delivery in tourism

[77] A 2003 Service & Support Professionals Association study reported that the most common customer complaint (71%) is the time to resolve an issue. The study revealed that during an average 12-minute support call, support engineers often spend as much as 70% of their time in problem diagnosis and knowledge search, but only 30% in problem resolution (www.vivisimo.com).

[78] Adapted from www.vivisimo.com

[79] It does not mean that existing approaches are disappointing, but only that the expectations are too high.

Consequently, the result is that the same problems regarding information overload accounted several years ago are still present and even more critical (due to the increase in the content volume and its heterogeneity).

Therefore, it seems to be that traditional thinking about information search has achieved its limits regarding the quality of results. Otherwise, taking into account how much money has been invested nowadays in the search-engine industry[80], we should expect substantial improvements in traditional, keyword-based search engines (at least in Google), comparing to their performance from several years ago.

As pointed out in this thesis, a crucial problem is a too simplified approach to understand what a user is searching for and what a document is about: neither a user's information need nor the content of a document can be well (i.e. understandable) represented as a bag of words. It is practically impossible to consider only isolated words and to expect to get only meaningful results, independently how much one does believe in the ranking mechanism. Regarding super-player Google, this is especially valid for enterprise search where the hyperlink structure of documents, as an additional source of relevance, does not exist.

Therefore, some more powerful, but scalable, methods for the description of the meaning of a query and the content of an information resource are needed. This thesis is a journey to discover such solutions.

In this thesis we promote ontologies as a promising technology for the improvement of the search process. According to Gartner „corporate taxonomies are recognized as the key to understanding the scope and content of 'what an organisation knows'„. Ideally, an information repository should be represented using ontology-based statements and a user has to be familiar with ontology-based querying in order to discover the full potential of the ontology-based search. In the last six years such a vision emerges from the Semantic Web initiative for a machine-readable and understandable Web, which will enable an unified exchange of heterogeneous and distributed information and direct communication between systems and devices even if they talk different languages.

Moreover, independently of the efforts that should be put to develop and continually evolve an ontology, the efficient usage of ontologies in information retrieval opens some important questions.

One problem we discussed in detail in this work is modeling the relevance in the ontology-based information retrieval, considering the situation that almost all ontologies that can be found in Semantic Web are non-probabilistic and that the relevance is one of the main issues in information retrieval. Another problem is, for example, the confidence in the metadata that requires very efficient solutions for trust management.

A totally different problem is the concrete role that an ontology can play in an information retrieval task:

- An ontology is a powerful retrieval model, under the assumption that a user will make an "ideal" query and an efficient inference process will return "ideal" results,
- An ontology is an efficient conceptualization model, under the assumption that a user has a problem to define his information need and the main task is to precisely define what he is searching for.

[80] Search business is in an extreme expansion now (e.g. Google trade value is $226 per share comparing to $26 per share for Microsoft).

Correspondingly, the focus of the ontology-based information retrieval research can be on the retrieval or on the conceptualization phase[81] of an information retrieval process.

By supporting the second issue[82], we see a big importance to include a user (actually his judgment) more actively in the search process, since he usually makes a short query but he can judge about the relevance of other terms for his need in an "I know it, when I see it" style. However, users are often very reluctant to provide more information about themselves, such that implicit feedback should be counted as the only source for discovering the user's search context. Moreover, explicit profiles are found as not so reliable due to a drift in preferences or due to sharing computer equipment. Finally, "personalization" is not just about asking a user to explicitely formulate what he likes, but moreover to discover what this particular user is searching for. Note that search is a problem-solving activity (not only in an enterprise context), which means that a user has a task in mind that is usually a meaningful description[83] (and not just a bag of words). Therefore, the user's task in combination with his background knowledge (including his knowledge gap), plus an affective context in which a user behaves, define an information need that results in a query.

In this thesis we comprehended all these factors that influence a user's need in a complex notion of relevance and have realized a query refinement process that enables an incremental development of his need.

This query refinement approach involves a user in a search process more actively and, like in a brick-and-mortar environment, enables the retrieval system to cooperate with the user in resolving his information need by providing additional hints or similar results.

In this work we have extended the notion of query refinement in three ways:

1. First, we see query refinement as a process in which the meaning of the query is clarified, i.e. as a process of discovering a user's information need, at least that part that is related to his cognitive gap. The information about affective context can be concluded from the user's search behavior and can be reflected in the number of presented results or even in the way how results are presented. For example, the number of clicks and the time spent on a result page (e.g. that contains ten results) can indicate the user's current mood regarding search.

2. Second, we treat refinement as a clustering task, where the set of clusters is minimal and complete regarding the set of results. In that way a user can be sure that by inspecting generated refinements (i.e. clusters) a complete list of results (or at least a complete part, e.g. first 500 results) is covered. Moreover, the meaning of the clusters regarding the given query is clear. This is the main drawback in the current clustering methods that use only syntactical processing of results in order to generate clusters.

3. Third, due to a user's unfamiliarity with the information repository or the ill-definedness of his information need, he can be lost in the information space and he creates a query that does not correspond well to his task. In that way a traditional query refinement approach will try to refine the query by adding new query terms, which can lead to a drift in the meaning of the query. On the other hand, the methods presented in this thesis exploit the possibility that a query is a wrong interpretation of the user's task and provide the user with alternative, more reliable, queries. In that way we can talk about the refinement of the user's information need, instead of the refinement of just a query.

[81] Google recently announced research regarding the role an ontology can play in resolving the ambiguity in a user's query.

[82] Certainly, the combination of both issues, i.e. improving the inference mechanism and the incremental conceptualization, is the best option. However in this research we "start" from an existing inference engine.

[83] In fact we can imagine a task as an information resource that should be matched with information resurces from the given information repository.

Moreover, we have performed several evaluation studies as proofs of the concepts we proposed in this thesis. The main finding is that if more semantics is introduced in the model of a retrieval system, then the precision of the system increases, and in addition the system can provide new functionalities that cannot be realized efficiently with traditional, syntax-based systems. Further, from the user's point of view, additional functionalities mean new ways to express an information need, or new views on the content that can be combined in order to get new insights about it. Moreover, the evaluation studies have shown experimentally that a comprehensive, semantically-based modelling of relevance in a query refinement process is needed, since in a refinement process a user does not expect only a relevant refinement, but moreover a refinement that can help him to clarify the meaning of the query (i.e. to decrease its ambiguity). Otherwise, he is lost in the information space, as experienced in traditional search approaches.

However, the crucial problem is, where does a usable semantic model come from, since the ontology development remains a bottleneck in building ontology-based application, including information retrieval systems. In order to make our methods more applicable for real-world problems, in this thesis we have proposed a method to learn a semantic model suitable for IR tasks from the result at query time.

As we already mentioned, this thesis is a journey to a cooperative, user-driven search paradigm that will give a user more confidence in the search process. How are we sure that the recommendation we get from an (human) librarian are "ideal"? In fact, after explaining what we need, we are just confident that he has enough knowledge to deliver us a few of the most relevant books for our need, or to give some hints we were not aware about. However, he should ask only "meaningful" questions, since otherwise the confidence in his recommendation, and consequently in the search process, will decrease. Obviously, this is as far and unpredictable a journey as the automation of decision-making processes (e.g. design of expert systems) was. Surely, we can reuse some lessons learned from that story. For example, that not all knowledge can be acquired, that it is dynamic and has a context-dependent relevance, that it should be represented in a reconfigurable form and that this reconfiguration should be task-driven. This is the light we pack for our next journey.

8 Appendix A - The conceptual models of the solutions provided in each chapter

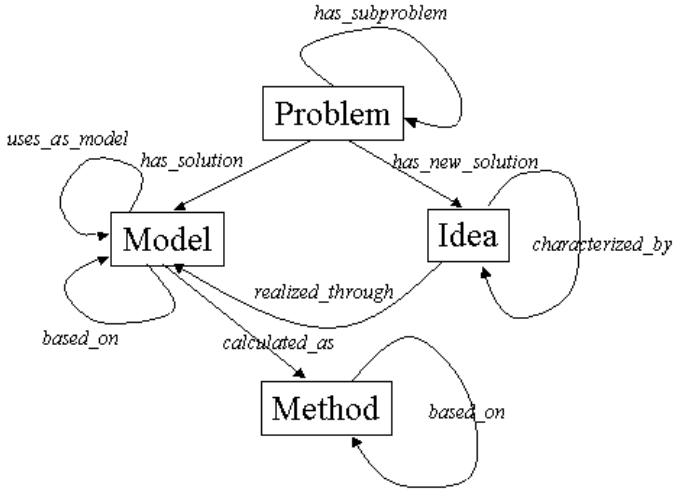


Figure A1: The conceptual model of a problem-solving activity

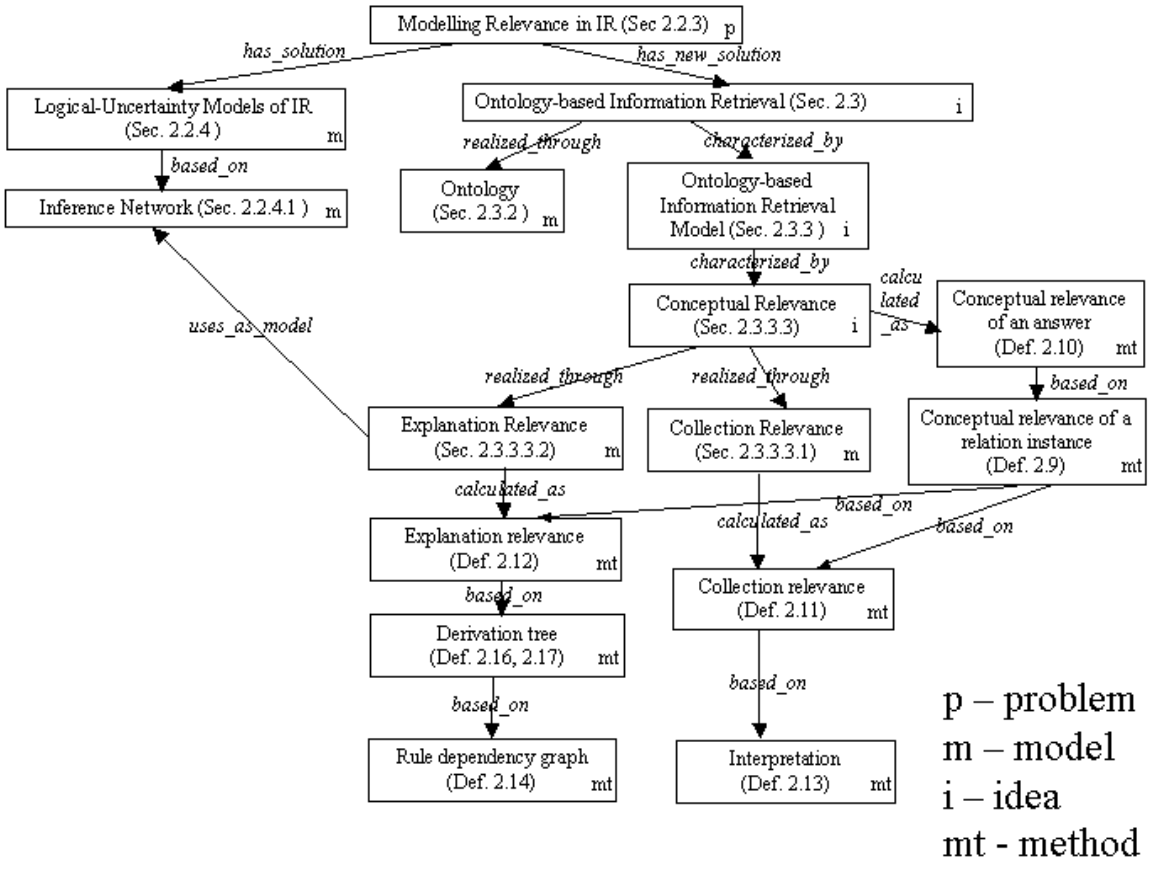


Figure A2: The conceptual model of Chapter 2

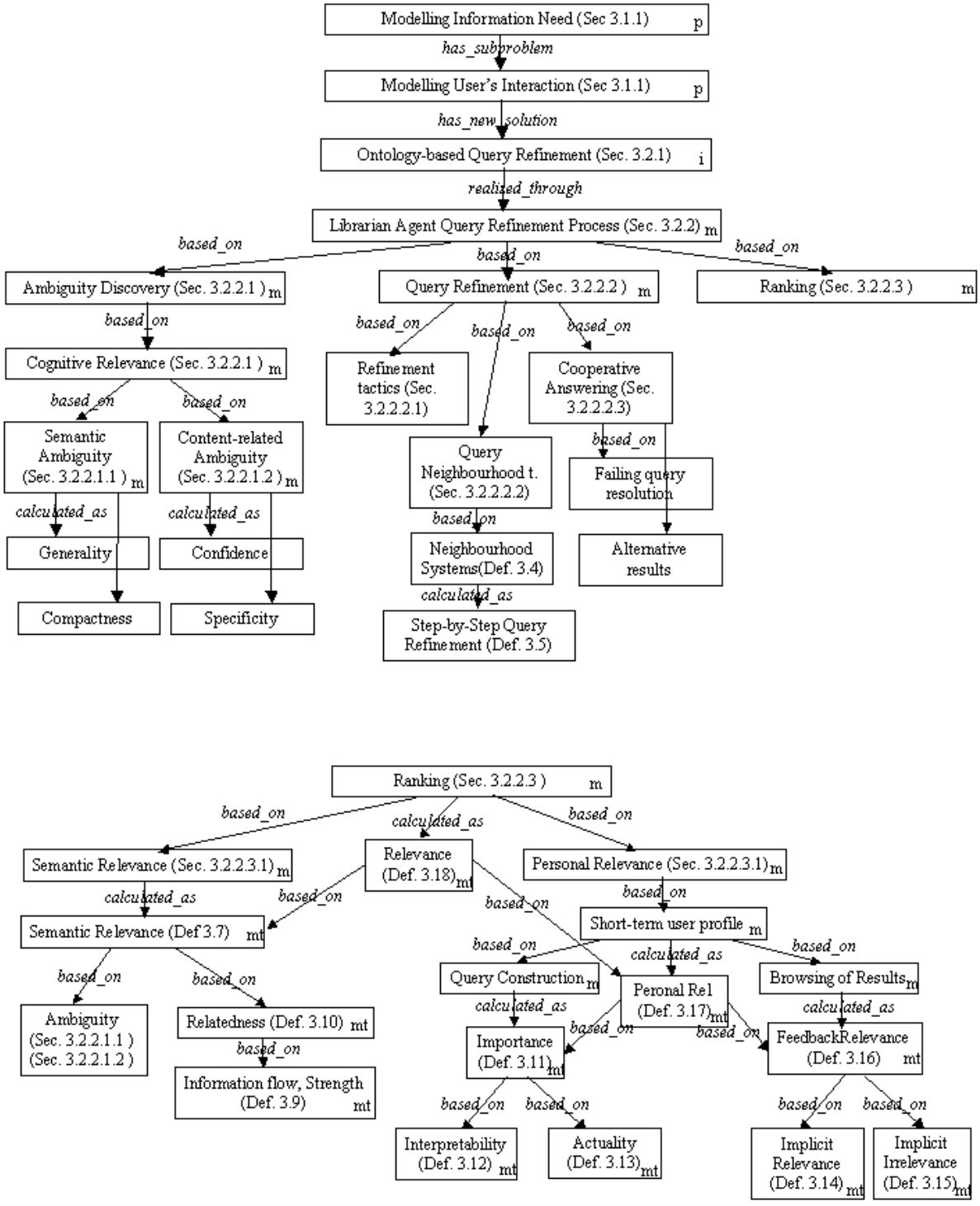


Figure A3: The conceptual model of Chapter 3

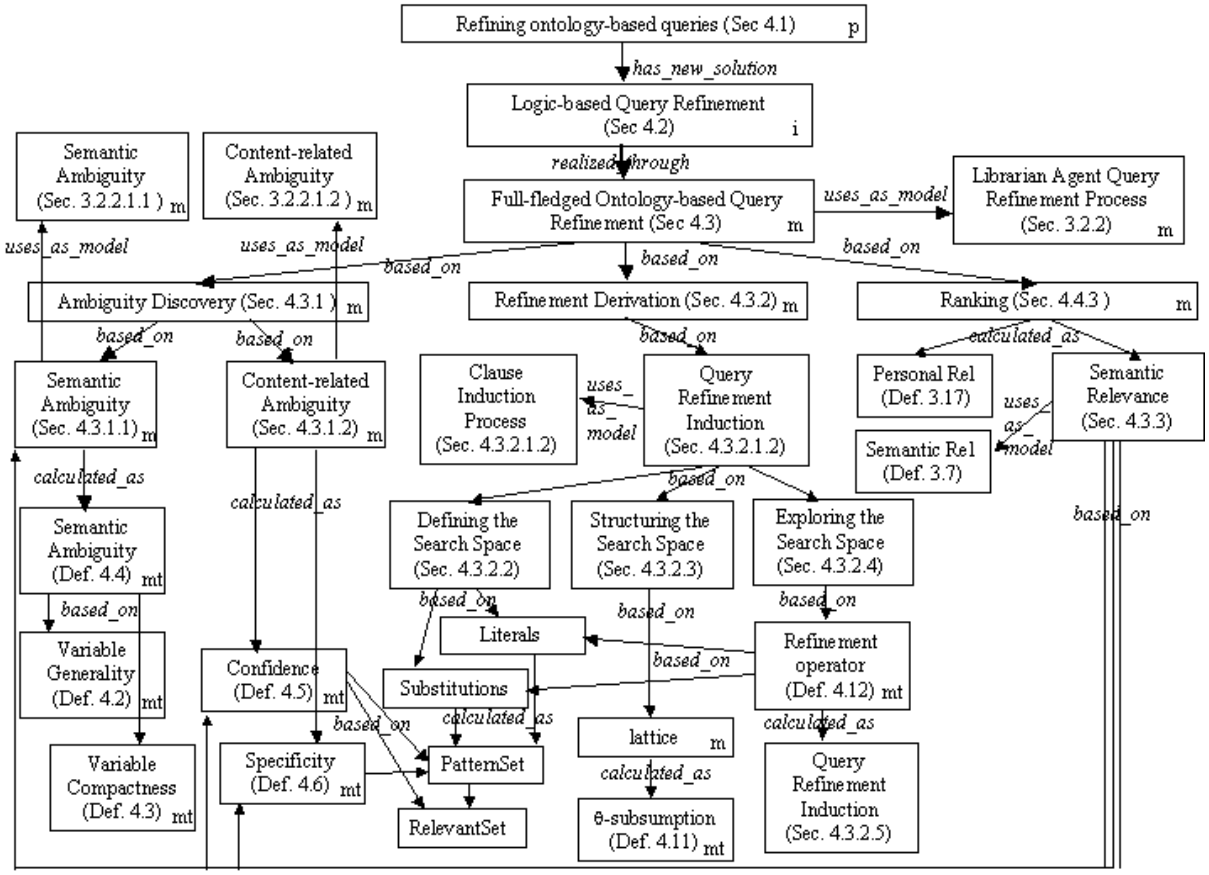


Figure A4: The conceptual model of Chapter 4

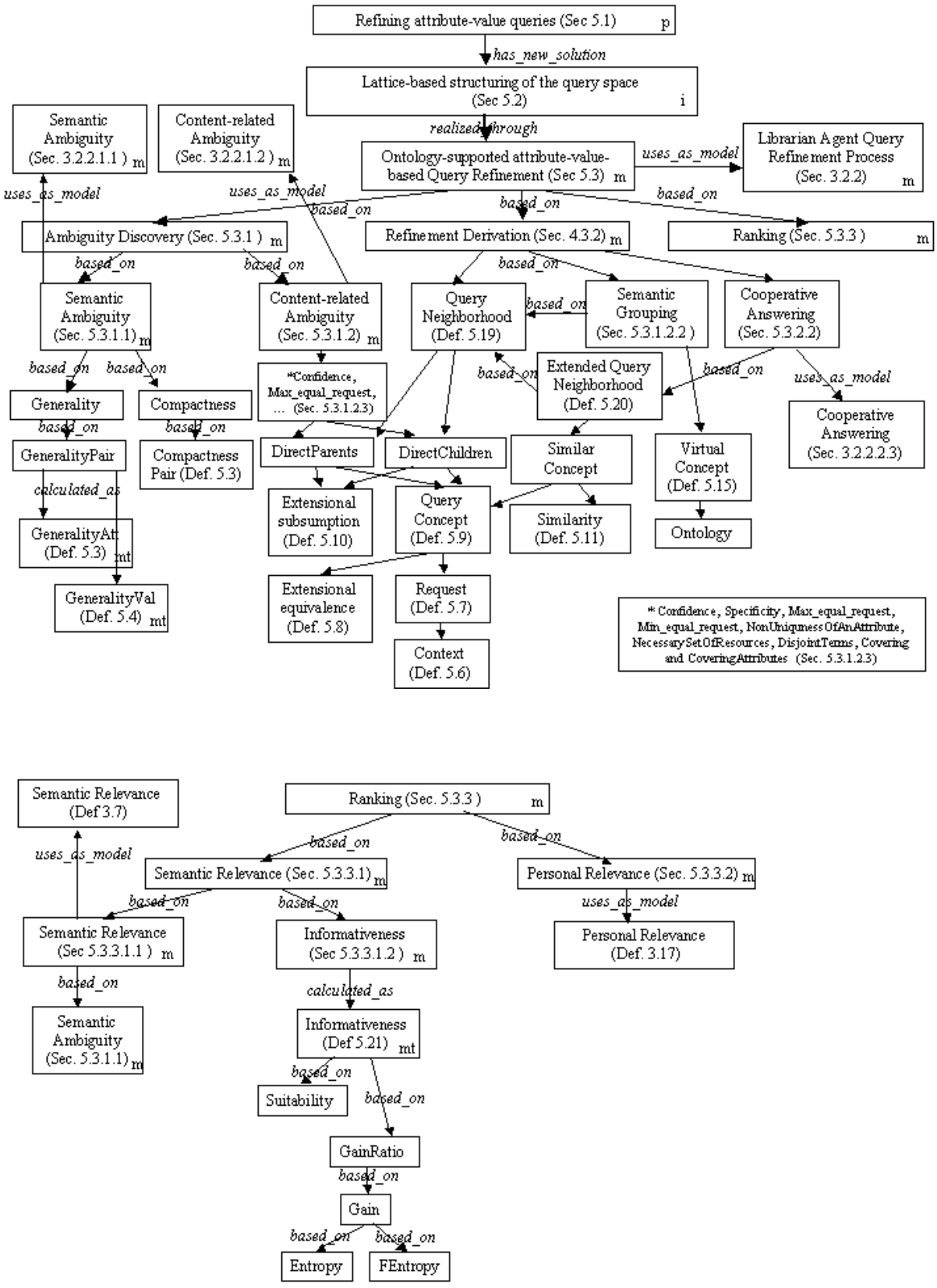


Figure A5: The conceptual model of Chapter 5

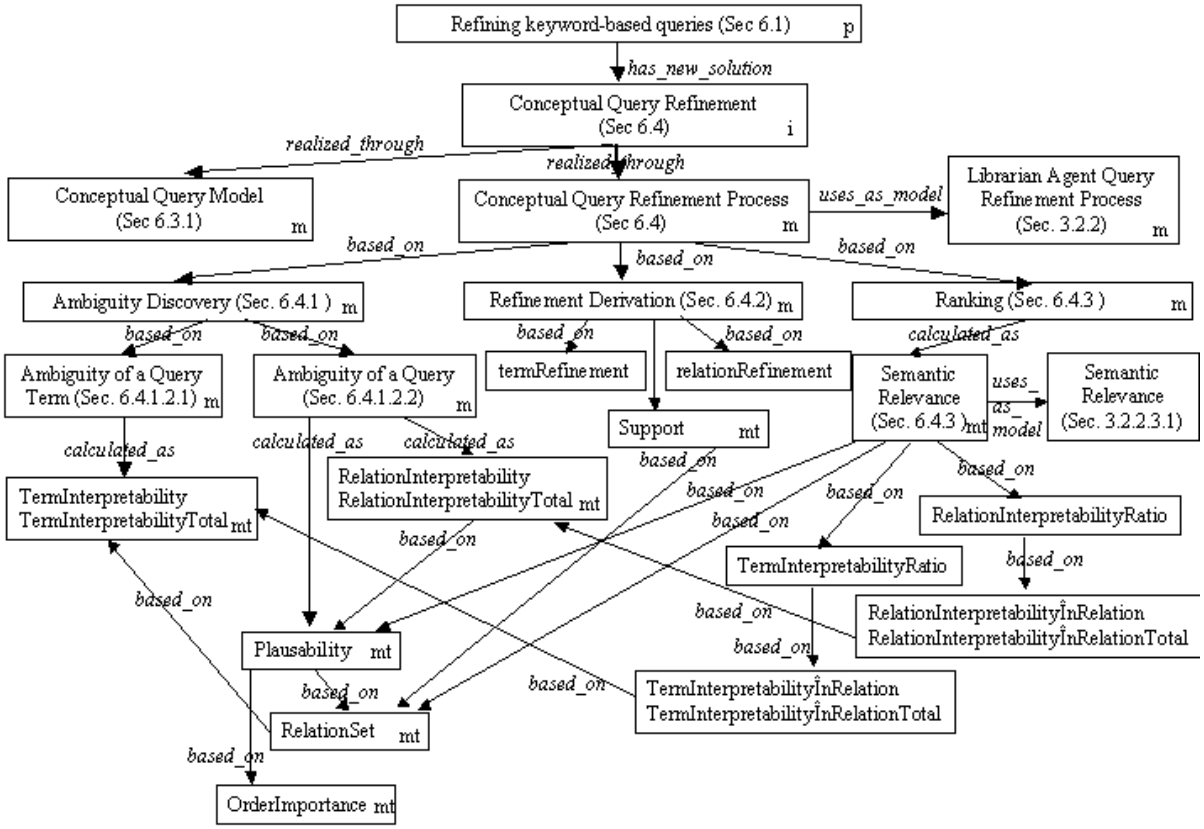


Figure A6: The conceptual model of Chapter 6

9 References

[1] P.G.Anick, S. Tipirneni, *The Paraphrase Search Assistant: Terminological Feedback for Iterative Information Seeking*, In Proceedings of SIGIR 1999, pp. 153-159, 1999.

[2] T. Arora, R. Ramakrishnan, W.G. Roth, P. Seshadri, D. Srivastava, *Explaining program execution in deductive systems*, In S. Ceri, K. Tanaka, S. Tsur (Eds.): Deductive and Object-Oriented Databases, in Proceedings of Third International Conference (DOOD'93), LNCS 760, Springer-Verlag, pp. 101-119 ,1993.

[3] I.J. Aalbersberg, *A document retrieval model based on term frequency ranks*, In Proceedings of the 17th International Conference on Research and Development in Information Retrieval (ACM SIGIR '94), Dublin, Ireland, ACM, pp. 163-172, 1994.

[4] L. Badea, M. Stanciu, *Refinement Operators can be (weakly) perfect*, In S. Dzeroski, P. Flach (Eds.): Inductive Logic Programming, In Proceedings of 9th International Workshop (ILP-99), LNCS 1634, Bled, Slovenia, pp. 21-32, 1999.

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

[6] B. Bakel, *Modern classical document indexing: a linguistic contribution to knowledge-based*, in Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '98), ACM, pp. 333-334. 1998.

[7] M. Balabanovic, Y. Shoham, *Content-Based, Collaborative Recommendation*, in Communications of the ACM (CACM), Volume 40, Number 3, pp. 66-72, 1997.

[8] J. Barwise, J. Seligman, *Information Flow: The Logic of Distributed Systems*, Cambridge Tracts in Theoretical Computer Science, 1997.

[9] M. J. Bates*, The design of browsing and berrypicking techniques for the online search interface*, Online Review, 13(5), pp. 407-424, 1989.

[10] M. J. Bates, *Where should the person stop and the information search interface start?*, in Information Processing and Management, Volume26, Number 5, pp. 575-591, 1990.

[11] P. Becker, P. Eklund, *Prospects for Document Retrieval using Formal Concept Analysis*, in Proceedings of the Sixth Australasian Document Computing Symposium, Coffs Harbour, Australia, http://www.kvocentral.org/kvopapers/beckeradcs01.pdf, 2001.

[12] B. Berendt, *Using site semantics to analyze, visualize, and support navigation*, in Data Mining and Knowledge Discovery, Volume 6, Number 1, pp. 37-59, 2002.

[13] H. Blockeel, L. de Raedt, N. Jacobs, B. Demoen, *Scaling up inductive logic programming by learning from interpretations*, in Data Mining and Knowledge Discovery, Volume 3, Number 1, pp. 59-93, 1999.

[14] A. Bookstein, S.T. Klein, T. Raita, *Clumping properties of content-bearing words*, in Journal of the American Society for Information Science (JASIS), Volume 49, Number 2, pp. 102-114, 1998.

[15] P. Borlund, *The IIR evaluation model: A framework for evaluation of interactive information retrieval systems*, in Information Research, Volume 8, Number 3, http://informationr.net/ir/8-3/paper152.html, 2003.

[16] A. Botafogo, E. Rivlin, B. Shneiderman, *Structural Analysis of Hypertexts: Identifying Hierarchies and useful Metrics*, in ACM Transactions on Information Systems (TOIS), Volume 10, Number 2, pp. 142 - 180, 1992.

[17] S. G. Bradshaw, *Reference directed indexing: Attention to descriptions people use for information*, Master's thesis, The University of Chicago, Chicago, 1998.

[18] P. Bruza, T. van der Weide, *Stratified Hypermedia Structures for Information Disclosure*, in The Computer Journal, Volume 35, Number 3, pp. 208-220, 1992.

[19] P.D. Bruza, S. Dennis, *Query Reformulation on the Internet: Empirical Data and the Hyperindex Search Engine*, in Proceedings of the 5TH RIAO Conference on Computer-Assisted Information Retrieval (RIAO97) - Computer-Assisted Information Searching on Internet, pp. 488-499, 1997.

[20] P. Bruza, D. Song, *Inferring query models by computing information flow*, Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2002), pp. 260-269, 2002.

[21] I. Campbell, *The ostensive model of developing information needs*, Ph.D. Thesis, University of Glasgow, 2000.

[22] C. Carpineto, G. Romano, *A order-theoretic approach to conceptual clustering*, In Proceedings of the 10th International Conference on Machine Learning, pp. 33-40, USA, 1993

[23] C. Carpineto, G. Romano, *Effective reformulation of boolean queries with concept lattices*, In Proceedings of the 3rd International Conference on Flexible Query-Answering Systems (FQAS 1998), Denmark, pp. 83-94, 1998.

[24] C. Carpineto, G. Romano, *Order-theoretical ranking*, in Journal of the American Sociery for Information Science (JASIS), Volume 51, Number 7, pp. 587-601, 2001.

[25] W. W. Chu, K. Chiang, C.-C. Hsu, H. Yau, *Error-based Conceptual Clustering Method for Providing Approximate Query Answers*, in Communication of the ACM (CACM), Volume 39, Number 12, pp. 216–230, 1996.

[26] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, J. Rice, *OKBC: A Programmatic Foundation for Knowledge Base Interoperability*, in Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 98), pp. 600-607, 1998.

[27] E. Charniak, *Passing markers: A theory of contextual influence in language comprehension*, In Cognitive Science, Volume 7, pp. 171-190, 1983.

[28] S. Chaudhuri, *Generalization and a Framework for Query Modification*, In Proceedings of the Sixth International Conference on Data Engineering (ICDE 1990), IEEE Computer Society, Los Angeles, CA, pp. 138-145, 1990.

[29] E.H. Chi, P. Piroli, K. Chen, J.E. Pitkow, *Using information scent to model user information needs and actions and the Web*, in Proceedings of the SIG-CHI on Human factors in computing systems (CHI 2001), pp. 490-497, 2001.

[30] C.W. Choo, B. Detlor, D. Tunrnball, *Web Work: Information Seeking and Knowledge Work on the World Wide Web*, Kluwer, 2000.

[31] W.J. Clancey, *The Knowledge Level Reinterpreted: Modeling How Systems Interact*, In Machine Learning, Volume 4, pp. 285-291, 1989.

[32] W.S. Cooper, *A definition of relevance for Information Retrieval*, in Information Storage and Retrieval, Volume 7, pp. 19-37, 1971.

[33] F. Creatsni, M. Lalmas, *Logic and Uncertainty in Information Retrieval*, in M. Agosti, F. Crestani, G. Pasi (Eds.): Lectures on Information Retrieval, Third European Summer-School (ESSIR 2000), Varenna, Italy, Revised Lectures, LNCS 1980, Springer, pp. 179-206, 2000.

[34] W.B. Croft, S. Cronen-Townsend, V. Lavrenko, *Relevance Feedback and Personalization: A Language Modeling Perspective*, in Proceedings of the Second DELOS Network of Excellence Workshop on "Personalisation and Recommender Systems in Digital Libraries", http://www.ercim.org/publication/ws-proceedings/DelNoe02/croft-delos.pdf, 2001.

[35] S. Cronen-Townsend, W.B. Croft, *Quantifying query ambiguity*, in Proceedings of Human Language Technology Conference (HLT 2002), San Diego, CA, pp. 94-98, 2002.

[36] H. Cui, J.R. Wen, J.Y. Nie, W.Y. Ma, *Probabilistic Query Expansion Using Query Logs*, in Proceeding of the Eleventh World Wide Web Conference (WWW 2002), Honolulu, Hawaii, pp. 325-332, 2002.

[37] S. Decker, M. Erdmann, D. Fensel, R. Studer, *Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information*, In R. Meersman, Z. Tari, S. Stevens (Eds.): Database Semantics - Semantic Issues in Multimedia Systems, IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics (DS-8), Kluwer Academic Publisher, pp. 351–369, 1999.

[38] L. Dehaspe, *Frequent pattern discovery in first-order logic*, PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, http://www.cs.kuleuven.ac.be/~ldh, 1998.

[39] L. Dehaspe, H. Toivonen, *Discovery of frequent datalog patterns*, in Data Mining and Knowledge Discovery, Volume 3, Number 1, pp. 7-36, 1999.

[40] L. Dehaspe, H. Toivonen, *Discovery of Relational Association Rules*, In N. Lavrac, S. Dzeroski (Eds.): in Relational Data Mining, Springer-Verlag, pp. 189 – 212, 2001.

[41] L. De Raedt, L. Dehaspe, *Clausal Discovery*, in Machine Learning, Volume 26, Numbers 2-3, pp. 99-146, 1997.

[42] C. Duursma, *Role Limiting methods for the Concept Model Library*, ESPRIT Project P5248, http://arti.vub.ac.be/kads/CH/CH.html, 1993.

[43] E.N. Efthimiadis*, Interactive Query Expansion: a user-based evaluation in a relevance feedback environment*, in Journal of the American Society for Information Science, Volume 51, Number 11, pp. 989-1003, 2000.

[44] P.Eklund, B. Groh, G. Stumme, R. Wille, *A Contextual-Logic Extension of TOSCANA*, B.Ganter, G. Mineau (Eds.): Conceptual Structures: Logical, Linguistic, and Computational Issues, 8th International Conference on Conceptual Structures (ICCS 2000), Darmstadt, Germany, LNCS 1867, Springer Verlag, pp.453-467, 2000.

[45] D. Felix, C. Niederberger, P. Steiger, and M. Stolze, *Feature-oriented vs. Needs-oriented Product Access for Non-Expert Online Shoppers*, in Proceedings first IFIP Conference on e-commerce, e-business, and e-government (I3E), pp 399-406, 2001.

[46] R. Fikes, T. Kehler, *The Role of Frame-Based Representation in Reasoning*, in Communication of ACM (CACM), Volume 28, Number 9, pp. 904-920,1985.

[47] R. Fikes, P. Hayes, I. Horrocks, *OWL-QL - A Language for Deductive Query Answering on the Semantic Web*, in Journal of Web Semantics, Volume 2, Number 1, pp. 19-29, 2004.

[48] C. Fluit, M. Sabou, F. van Harmelen, *Ontology-based Information Visualisation*, In V. Geroimenko, C. Chen (Eds.): Visualizing the Semantic Web, Springer, pp. 36-48, 2002.

[49] N. Fuhr, K. Großjohann, *XIRQL: A Query Language for Information Retrieval in XML Documents*, W. B. Croft, D. J. Harper, D. H. Kraft, J. Zobel (Eds.): SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, pp. 172-180, 2001.

[50] R. Fung, B. Del Favero, *Applying bayesian networks to Information Retrieval*, In Communications of the ACM, Volume 38, Number 3, pp. 42-48, 1995.

[51] T. Gaasterland, P. Godfrey, J. Minker, *An overview of cooperative answering*, in Journal of Intelligent Information Systems, Volume 1, Number 2, pp. 123--157, 1992

[52] B. Ganter, R. Wille, *Formal Concept Analysis: Mathematical Foundations*, Springer Verlag, 1999.

[53] P. Godfrey, *Minimization in cooperative response to failing database queries*, in International Journal of Cooperative Information Systems (IJCIS), Volume 6, Number 2, pp. 95-149, 1997.

[54] F. Grootjen, *Employing semantical issues in syntactical navigation*, in Proceedings of BCS-IRSG 2000 Colloquium on Information Retrieval Research, http://www.niii.ru.nl/F.Grootjen/publications/irsg00.pdf, 2000.

[55] T. R. Gruber, *Towards principles for the design of ontologies used for knowledge sharing*, in International Journal of Human-Computer Studies, Volume 43, Number 5-6, pp. 907-928, 1995.

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

[57] N. Guarino, C. Masolo, G. Vetere, *OntoSeek: Content-Based Access to the Web*, in IEEE Intelligent Systems, Volume 14, Number 3, pp. 70-80, 1999.

[58] S. Henninger, *Using Iterative Refinement to Find Reusable Software*, in IEEE Software, Volume 11, Number 5, pp. 48-59, 1994.

[59] J. Howard, J.N.Sheth*, The theory of buyer behaviour*, Willy, 1994.

[60] I. Horrocks, S. Tessaris, *Querying the semantic web: a formal approach*, In Ian Horrocks and James Hendler (Eds.): The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, LNCS 2342, Springer-Verlag, pp. 177-191, 2002.

[61] T.W.C. Huibers, B. van Linder, *Formalising Intelligent Information Retrieval Agents,* In Proceedings of the 18th British Computer Society Information Retrieval Colloqium, UK, pp. 125-143, 1996.

[62] C. Janiszewski, *The Influence of Display Characteristics on Visual Exploratory Search Behavior*, in Journal of Consumer Research, Volume 25, Number 3, pp. 290-301, 1998.

[63] M. Jarke. J. Koch, *Query Optimization in Database Systems*, ACM Comp. Survey, Volume 16, Number 2, pp. 111–152, 1984.

[64] T. Joachims, D. Freitag, T. Mitchell, *Webwatcher: A tour guide for the World Wide Web*. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, (IJCAI 97), Nagoya, JapanMorgan Kaufmann, pp. 770-777, 1997.

[65] T. Joachims, *Evaluating Retrieval Performance Using Clickthrough Data*, in Proceedings of the SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval, 2002.

[66] H. Joho, C. Coverson, M. Sanderson, M. Beaulieu, *Hierarchical presentation of expansion terms*, in Proceedings of the 2002 ACM Symposium on Applied Computing (SAC), ACM, pp. 645-649, 2002.

[67] S. J. Kaplan, *Cooperative Responses from a Portable Natural Language Query System*, In Artificial Intelligence, Volume 19, Number 2, pp. 165-187, 1982.

[68] P. D. Karp, *The design space of frame knowledge representation systems*, Technical Report 520, SRI International AI Center; available on line as ftp://www.ai.sri.com/pub/papers/karp-freview.ps.Z, 1992.

[69] S.O. Kuznetsov, S.A. Obiedkov, *Comparing performance of algorithms for generating concept lattice*, in Journal of Experimental and Theoretical Artificial Intelligence, Volume 14, Number 2-3, pp. 189-216, 2002.

[70] M. Kifer, G. Lausen, J. Wu, Logical Foundations of Object-Oriented and Frame-Based Languages, in Journal of ACM, Volume 42, Number 4, pp. 741–843, 1995.

[71] T. Lau, Y. Sure, *Introducing Ontology-based Skills Management at a large Insurance Company*, M. Glinz, G. Mueller-Luschnat (Eds.): Modellierung 2002, Modellierung in der Praxis - Modellierung für die Praxis, Arbeitstagung der GI, Tutzing, LNI 12, pp. 123-134, 2002.

[72] J. Lee, H. S. Lee, P. Wang, *Analytical Product Selection Using a Highly-Dense Interface for Online Product Catalogs*, IBM Inst. Advan. Comm., http://www.research.ibm.com/iac/papers/VOPC.pdf , 2001.

[73] D. Lee, *Query Relaxation for XML Model*, PhD Thesis, University of California, Los Angeles, June 2002.

[74] A. Maedche, S. Staab, N. Stojanovic, R. Studer, Y. Sure, *SEmantic portAL: The SEAL Approach*, Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential, MIT Press 2003, ISBN 0-262-06232-1, pp. 317-359, 2003.

[75] S. Marcus, *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers, 1988.

[76] J. Marcinowski, L. Pacholski, *Undecidability of the Horn-clause implication problem*, In Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1992), pp. 354-362, 1992.

[77] C. Martin, *Direct memory access parsing*, Tech. Rep. No. CS93-07, The University of Chicago, Department of Computer Science, 1993.

[78] G. A. Miller, R. Beckwith, C. Felbaum, D. Gross, K. Miller, *Introduction to WordNet : An On-line Lexical Database*, in International Journal of Lexicography, Volume 3, Number 4, pp. 235 – 244, 1990.

[79] M. Minsky, *A Framework for Representing Knowledge*, in Patrick Henry Winston (ed.), The Psychology of Computer Vision, McGraw-Hill, New York, 1975.

[80] T.M. Mitchell, *The need for biases in learning generalizations*, Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, 1980.

[81] T.M. Mitchell, *Generalization as search*, in Artificial Intelligence, Volume 18, Number 2, pp.203-226, 1982.

[82] A. Newell, *The knowledge level*, in Artificial Intelligence, Volume 18, pp. 87-127, 1982.

[83] J. Y. Nie, F. Lepage, M. Brisebois, *Information retrieval as counterfactual*, The Computer Journal, Volume 38, Number 8, pp. 643–657, 1996.

[84] S.H. Nienhuys-Cheng, W. Van Laer, J. Ramon, De Raedt, *Generalizing Refinement Operators to Learn Prenex Conjunctive Normal Forms*, In S. Dzeroski, P. Flach (Eds.): Inductive Logic Programming, In Proceedings of 9th International Workshop (ILP-99), LNCS 1634, Bled, Slovenia, pp. 245-256, 1999.

[85] S. Nienhuys, W. Laer, J. Ramon, *Generalizing refinement operators to learn prenex conjunctive normal forms*, ERIM Report Series, ERS-2000-39-LIS, 2000.

[86] P. Norvig, *From A Unified Theory of Inference for Text Understanding*, UC Berkeley Computer Science Technical Report CSD-87-339, 1987.

[87] L. Nourine, O. Raynaud, *A fast algorithm for building lattice*, in Information Processing Letters, Volume 71, pp. 199-204, 1999.

[88] C.K. Odgen, I.A. Richards, *The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolis*m, Routledge & Kegan Paul Ltd., London, 10 edition, 1923.

[89] K. O'Hara, H. Alani, N. Shadbolt, *Identifying Communities of Practice*, in R. Traunmueller (Ed.): Information Systems: The e-Business Challenge, IFIP 17th World Computer Congress - TC8 Stream on Information Systems: The e-Business Challenge, IFIP Conference Proceedings 2002, Kluwer, pp. 89-102, 2002.

[90] M. Ortega-Binderberger, K. Chakrabarti, S. Mehrotra, *An Approach to Integrating Query Refinement in SQL*, in C. Jensen, K. Jeffery, J. Pokorny, S. Saltenis, E. Bertino, K. Boehm, M. Jarke (Eds.): Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, LNCS 2287, pp. 15-33, March, 2002.

[91] M. Quillian, *The teachable language comprehender*, in Communications of the ACM, Volume 12, Number 8, pp. 459-476, 1969.

[92] L. Page, *PageRank: Bringing order to the web*, Stanford Digital Libraries, http://hci.stanford.edu/ page/papers/pagerank, 1997.

[93] G.D. Plotkin, *A note on inductive generalization*, In B. Meltzer, D. Michie, (Eds.): Machine Intelligence, American Elsevier, Volume 5, Number 8, pp. 153 – 163, 1970.

[94] J. Ponte, W.B. Croft, *A language modeling approach to information retrieval*, in Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '98), Melbourne, Australia. pp. 275-281, 1998.

[95] P. Pu, P. Janecek, *Visual Interfaces for Opportunistic Information Seeking*, In Proceedings of the 10th International Conference on Human - Computer Interaction (HCII'03), Crete, Greece, pp. 1131-1135, 2003.

[96] F. Puppe, *Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods*, Springer Verlag, Berlin, 1993.

[97] R. Rada, H. Mili, E. Bicknell, M. Blettner, *Development and application of a metric on semantic nets*, in IEEE Transaction on Systems, Man, and Cybernetics, Volume 19, Number 1, pp. 17–30 ,1989.

[98] P. Resnik, *Using information content to evaluate semantic similarity in a taxonomy*, In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 95), Montréal, Canada, pp. 448-453,1995.

[99] C. Riesbeck, *Knowledge reorganization and reasoning style*, Tech. Rep. No. 270, New Haven, CT: Yale University, Department of Computer Science, 1983.

[100] R. Richardson, A.F. Smeaton, J. Murphy, *Using Wordnet as knowledge base for measuring semantic similarity between word*s, Technical Report CA-1294, Dublin City University, School of Computer Applications, 1994.

[101] F. Ricci, A. Venturini, D. Cavada, N. Mirzadeh, D. Blaas, M. Nones, *Product Recommendation with Interactive Query Management and Twofold Similairty*, In Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR 2003), pp. 479-493, 2003.

[102] C. J. van Rijsbergen, *A new theoretical framework for information retrieval*, In Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'86), Pisa, Italy, pp. 194–200, 1986.

[103] C. J. van Rijsbergen, *A non-classical logic for information retrieval*, in The Computer Journal, Volume 29, Number 6, pp. 481-485, 1986.

[104] M. Rila, *The Use of WordNet in information retrieval*, in Proceedings of the ACL Workshop on the Usage of WordNet in Natural Language Processing Systems, pp. 31-37, 1998.

[105] J.J. Rocchio, *Relevance feedback in information retrieval*, in the SMART Retrieval System - Experiments in Automatic Document Processing, Englewood Cliffs, NJ, Prentice Hall, Inc. pp. 313-323, 1971.

[106] G. Ruge, *Experiments on Linguistic based Term Associations*, RIAO91, 1991.

[107] I.Ruthven, M.Lalmas, C.J. van Rijsbergen, *Incorporating user search behaviour into relevance feedback*, in Journal of the American Society for Information Science and Technology, Volume 54, Number 6, pp. 528-548, 2003.

[108] G. Salton, C. Buckley, *Improving retrieval performance by relevance feedback*, in Journal of the American Society for Information Science, Volume 41, Number 4, pp. 288-297, 1990.

[109] G. Salton, C. Buckley, *Automatic text structuring and retrieval - Experiments in automatic encyclopedia searching*, in Proceedings of the 14th International Conference on Research and Development in Information Retrieval (ACM SIGIR '91), Chicago, pp. 21-30, 1991.

[110] T. Saracevic, *Relevance: A Review of and a framework for the thinking on the notion in information science*, in Journal of the American Society for Information Science, Volume 26, Number 6, pp. 321-343, 1975.

[111] A.Th. Schreiber, B.J. Wielinga, R. de Hoog, H. Akkermans, W. van de Velde, *CommonKADS: A Comprehensive Methodology for KBS Development*, In IEEE Expert, Volume 9, Number 6, pp. 28-37, 1994.

[112] B. R. Schatz, E.H. Johnson, P.A. Cochrane, H. Chen, *Interactive Term Suggestion for Users of Digital Libraries: Using Subject Thesauri and Co-occurrence Lists for*

Information Retrieval, in Proceedings of the 1st ACM International Conference on Digital Libraries (Digital Libraries 1996), pp. 26-133, 1996.

[113] J.B. Schafer, J.A.Konstan, J.Riedl, *E-Commerce Recommendation Applications*, in Data Mining and Knowledge Discovery, Volume 5, Number 1/2, pp. 115-153, 2001.

[114] W. Sierpenski, C. Krieger, *General Topology*, University of Toronto Press, 1956.

[115] F. Sebastiani, *On the role of logic in information retrieval*, in Information Processing and Management, Volume 34, Number 1, pp.1-18, 1998.

[116] U. Shah, T. Finin, A. Joshi, C. Cost, J. Mayfield, *Information Retrieval on the Semantic Web*, in Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2002), McLean, VA, USA, pp. 461-468, 2002.

[117] H. Shimazu, *ExpertClerk: Navigating Shoppers Buying Process with the Combination of Asking and Propossing*, in B. Nebel (Ed.): Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001), Seattle, Washington, USA, Morgan Kaufmann, pp.1443- 1450, 2001.

[118] C. Silverstein, M. Henzinger, H. Marais, M. Moricz, *Analysis of a Very Large Alta Vista Query Log* SRC Technical Note 1998-014, http://gatekeeper.dec.com /pub/DEC/SRC/technical-notes/abstracts/src-tn-1998-014.html, 1998.

[119] R. Srikant, R. Agrawal, *Mining Generalized Association Rules*, in U. Dayal, P. Gray, S. Hishio: Proceedings of 21th International Conference on Very Large Data Bases (VLDB'95), Zurich, Switzerland. Morgan Kaufmann, pp. 407-419, 1995.

[120] I. Soboroff, C. Nicholas, P. Cahan, *Ranking retrieval systems without relevance judgements*, In Proceeding of the Annual ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR01), pp. 66–73, 2001.

[121] D. Soergel, *Mathematical analysis of documentation systems*, in Information storage and retrieval, Volume 3, pp. 129-173, 1967.

[122] A. Spoerri, *Infocrystal: Integrating exact and partial matching approaches through visualization*, In Proceedings of Intelligent Multimedia Information Retrieval Systems and Management (RIAO 94), New York, pp. 687-696, 1994.

[123] A. Spink, T. Saracevic, *Interactive information retrieval: Sources and effectiveness of search terms during mediated online searching*, in Journal of the American Society for Information Science, Volume 48, Number 8, pp. 741-761, 1997.

[124] M. Stolze, *Soft Navigation in Product Catalogs*, in Proceedings of the 2nd European Conference on Research and Advanced Technology for Digital Libraries (ECDL 1998), Heraklion, GR, pp. 385-396, 1998.

[125] M. Stolze, W. Rjaibi, *Towards Scalable Scoring for Preferece-based Item Recommendation*, in IEEE Data Engineering Bulletin, Volume 24, Number 3, pp. 42-49, 2001.

[126] N. Stojanovic, A. Maedche, S. Staab, R. Studer, Y. Sure, *SEAL: A framework for developing SEmantic PortALs*, in Proceedings of the First International Conference on Knowledge Capture (K-CAP 2001), Victoria, BC, pp. 155-162, 2001.

[127] L. Stojanovic, N. Stojanovic, R. Volz, *Migrating data-intensive Web Sites into the Semantic Web*, In Proceedings of the ACM Symposium on Applied Computing (SAC 20002), Madrid, Spain, pp. 1100-1107, 2002.

[128] N. Stojanovic, *On Analysing Query Ambiguity for Query Refinement: The Librarian Agent Approach*, in Proceedings of 22nd International Conference on Conceptual Modelling (ER 2003), LNCS 2813, Chicago, IL, US, pp. 490-505, 2003.

[129] N. Stojanovic, *On the role of a Librarian Agent in ontology-based Knowledge Management Systems*, J.UCS Special Issue WM 2003, Volume 9 / Issue 7, pp. 697-718, July 2003.

[130] N. Stojanovic, *Information-Need Driven Query Refinement*, in Proceedings of 2003 IEEE / WIC International Conference on Web Intelligence (WI 2003), Halifax, Canada, IEEE Press, pp. 388-395, 2003.

[131] N. Stojanovic, L. Stojanovic, *On Modelling Cooperative Retrieval Using an Ontology-Based Query Refinement Process*, in Proceedings of 23rd International Conference on Conceptual Modelling (ER 2004), LNCS 3288, Shanghai, China, pp. 434-449, 2004.

[132] N. Stojanovic, *A Logic-Based Approach for Query Refinement*, in Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004), 20-24 September 2004, Beijing, China, pp. 477-480, 2004.

[133] N. Stojanovic, *On Using Query Neighbourhood for Better Navigation through a Product Catalog: SMART Approach*, in Proceedings of 2004 IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE 04), ISBN 0-7695-2073-1, Taipei, Taiwan, pp. 405-412, 2004.

[134] N. Stojanovic, *Information-need Driven Query Refinement*, Web Intelligence and Agent Systems, An International Journal, *IOS* Press ISSN: 1570-1263, planned for Vol. 3, No 3, 2005.

[135] N. Stojanovic, *On the Query Refinement in the Ontology-based Searching for Information*, Information Systems, Elsevier Science, ISSN 0306-4379, accepted for publishing, 2005.

[136] N. Stojanovic, *On the Conceptualization of the Query Refinement Task*, in Library Management, Emerald Group Publishing, Volume 26, Number 4/5, pp. 281 – 294, 2005.

[137] N. Stojanovic, *On the Query Refinement in the Ontology-based Searching for Information*, Information Systems, Elsevier Science, ISSN 0306-4379, accepted for publishing, 2005.

[138] N. Stojanovic, *On Ranking Refinements in the Step-by-Step Searching through a Product Catalogue*, in Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), Brighton, UK, pp. 527-530, 2004.

[139] N. Stojanovic, R. Studer, L. Stojanovic, *An Approach for Step-By-Step Query Refinement in the Ontology-Based Information Retrieval*, in Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004), 20-24 September 2004, Beijing,China, pp. 36-43, 2004.

[140] N. Stojanovic, *On the Role of Query Refinement in Searching for Information: The Librarian Agent Query Refinement Process*, in Proceedings of 4th International Conference on Web Information Systems Engineering (WISE 2003), Rome, Italy, pp. 41-52, 2003.

[141] N. Stojanovic, R. Studer, L. Stojanovic, *An Approach for the Ranking of Query Results in the Semantic Web*, In Proceedings of the International Semantic Web Conference (ISWC 2003), Sanibel Island, FL, USA, pp. 500-516, 2003.

[142] L. Stojanovic, *Methods and Tools for Ontology Evolution*, PhD Thesis, University of Karlsruhe, http://www.ubka.uni-karlsruhe.de/vvv/2004/wiwi/10/10.pdf, 2004.

[143] T. Strzalkowski, *Building a lexical domain map from text corpora*, COLING94, 1994.

[144] W.K. Sung, D. Yang, S.M. Yiu, D.W. Cheung, W.S. Ho, T.W. Lam, S.D. Lee, *Automatic Construction of Online Catalog Topologies*, in IEE Transactions on Systems, Man and Cybernetics – Part C, IEEE Publication, Volume 32, Number 4, pp. 382-391, 2002.

[145] U. Straccia, *Foundations of a Logic based approach to Multimedia Document Retrieval*, PhD Thesis, Department of Computer Science, University of Dortmund, June 1999.

[146] G. Stumme, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, N. Stojanovic, R. Studer, Y. Sure, R. Volz, V. Zacharias, *The Karlsruhe view on ontologies*, Technical report, University of Karlsruhe, Institute AIFB, 2003.

[147] M. Sussna, *Word Sense Disambiguation for Free-text Indexing Using a Massive Semantic Network*, in Proceedings of the Second International Conference on Information and Knowledge Management (CIKM 1993), Washington, pp. 67-74, 1993.

[148] M. Tan, J. C. Schlimmer, *Two case studies in costsensitive concept acquisition*, In Proceedings of 8th National Conference on Artificial Intelligence (AAAI-90), Boston, MA, pp. 854-860, 1990.

[149] A. Theobald, G. Weikum, *Adding relevance to XML*, in Proceedings of the 3rd International Workshop on the Web and Databases (WebDB 2000), Texas, USA, pp. 35-40, 2000.

[150] R.H. Thompson, W.B. Croft, *Support for Browsing in an Intelligent Text Retrieval System*, International Journal of Man-Machine Studies; 30: 639-668. ISSN: 0020-7373, 1989.

[151] K.S. Trividi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[152] H. Turtle, B. Croft, *Evaluation of an Inference Network-Based Retrieval Model*, in ACM Transaction of Information Systems (TOIS), Volume 9, Number 3, pp. 187-222, 1991.

[153] J. D. Ullman, *Principles of Database and Knowledge-based Systems*, Computer Science Press, Rockville, 1990.

[154] W. Van de Velde, *Issues in Knowledge Level Modelling*, in J.M. David, J.P. Krivine, R. Simmons, (Eds.): Second Generation Expert Systems, Springer Verlag, pp. 211-231, 1993.

[155] E. Voorhees, *Query expansion using lexical-semantic relations*, In W. B. Croft, C.J. van Rijsbergen (Eds.): Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1994), Dublin, Ireland, Special Issue of the SIGIR Forum, pp. 61-69, 1994.

[156] X. Jinxi, W. B. Croft, *Query expansion using local and global document analysis*, in Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval, pp.4-11, Zurich, Switzerland, 1996.

[157] Y.Y. Yao, *Neighborhood Systems and Approximate Retrieval*, Technical Report 03/2000, University of Regina ISBN 0-7731-00398-8, 2000.

[158] I. Weber, *Discovery of first-order regularities in a relational database using offline candidate determination*, In Proceedings of the Seventh International Workshop on Inductive Logic Programming (ILP 1997), Springer, Volume 1297, LNCS 1297, pp. 288-295, 1997.

[159] J.R. Wen, J.Y. Nie, H.J. Zhang, *Clustering User Queries of a Search Engine*, in Proceedings of the Tenth International World Wide Web Conference (WWW 10), Hong Kong, pp. 162-168, 2001.

[160] R.W. White, J.M. Jose, I. Ruthven, *An implicit feedback approach for interactive information retrieval*, in Information Processing and Management, Elsevier, http://umiacs.umd.edu/~ryen/papers/WhiteIPM2004.pdf , 2004.

[161] B. J. Wielinga, A. Th. Schreiber, J. A. Breuker, *KADS: A modelling approach to knowledge engineering*, in Knowledge Acquisition, Volume 4, Number 1, Special issue "The KADS approach to knowledge engineering", pp. 5-53, 1992.

[162] I.H. Witten, E. Frank, *Data Mining*, Morgan Kaufmann Publisher, 2000.

[163] S. Wong, Y. Yao, *On modeling information retrieval with probabilistic inference*, in ACM Transactions on Information Systems, Volume 13, Number 1, pp. 38–68, 1995.