Practical Evaluation of Contraction and Abduction Algorithm in Description Logic

Master Thesis

submitted in partial fulfillment of the requirements for the degree of Master of Science in Information and Media Technologies.

Submitted by

Yonny Sutanto

January 2006

Supervisor:

Prof. Dr. Ralf Möller

Prof. Dr. Friedrich H.Vogt

Technical University of Hamburg Harburg

Department of Software, Technology & System (STS)

Declaration

Hereby, I declare that this master thesis with the subject “Practical Evaluation of Contraction and Abduction Algorithm in Description Logic”, has been prepared by myself. All literally or content related quotations from other sources have been pointed out and no other sources than declared have been used.

Hamburg, January 2006

Yonny Sutanto

Abstract

Description Logics have been recognized as a general purpose language for knowledge representation. Some application domains in which Description Logics have shown their strengths are medical applications, libraries and information systems, configuration tasks and also software engineering. This Thesis will evaluate the non-standard reasoning services in Description Logics, contraction and abduction algorithm, and implement the given algorithm in an Image Retrieval System.

Acknowledgements

First of all, I would like to thank Prof. Dr. Ralf Möller from Software, Technology and Systems (STS) of Technical University of Hamburg-Harburg for giving me an opportunity to work on this topic of Master Thesis.

I would like to thank Michael Wessel for all his guidance and support from the time I started my project work until I finish my Master Thesis.

Finally, I would like to thank the Racer System Team (http://www.racer-systems.com) for providing an educational license of RacerPro.

Contents

List of Figures

Chapter 1

Introduction

1.1. Background

As the impact of digital technology, huge amount of image collections are made available through the World Wide Web for almost everyone. There are also enormous numbers of images exist in many type of organizations and institutions such as museums, libraries, archives, information centers, hospitals, educational institutions, newspapers as well as in personal archives [11].

Some image collections may exist in well-defined domains like architecture or medical, while some others exist in a very general domain like collections from stock photography. For image collections with ten thousands of images, an automatic indexing is a very good solution. The Content-Based Image Retrieval (CBIR) method is the answer for auto-indexing system but this is worth only in a well-defined domain, i.e. in a narrow field where the images in the collection have similar shapes, colors or textures.

The human-annotated images are still the best solution for most of image retrieval systems with broad collections of images and some studies have shown that users prefer to search by higher-level concepts [18]. Despite of its popularity among the users, Annotation-Based Image Retrieval (ABIR, [23]) has some limitations. Some of the main problems of manual annotation by human are:

- A keyword in a document does not necessarily mean that the document is relevant, and relevant document may not contain the explicit word [20, 21].
- The meaning of an image or what message it delivers depends on the purpose of the image [17].
- Lower recall rate of synonyms and lower precision rate of homonyms (precision and recall in information retrieval will be discussed in section 3.3.1.), also semantic relations such as hyponymy, meronymy and antonymy [22] are not exploited [19].
- Manual image annotation is time-consuming, thus it is costly [24].
- Human annotation is subjective [24].
- Some images could not be (human-) annotated since their content is difficult to describe using words [12].

There are many ways to overcome the problems mentioned above, some of the common approaches are by using thesaurus like WordNet [28] and by using classification systems like ICONCLASS [27] and Art and Architecture Thesaurus (AAT) [26]. A few researches have been using description logic based approaches to find the solution.

The research in this Thesis will also use a description logic based approach by implementing concept contraction and concept abduction algorithm given in [4].

1.2. Objectives

The main objective of this Thesis is to evaluate the concept contraction and concept abduction algorithm for matchmaking given by Cali et.al. [4], and implement it in an image retrieval system using Java programming language and RacerPro as the inference service.

The system should overcome the common problems in traditional image retrieval systems which are normally text- / metadata-based as mentioned in section 1.1. Other goals would be:

- To achieve better precision and recall rates.

- To give users a logical ranking system.
- Higher user satisfaction and lower user frustration.

1.3. Structure of this Thesis

Chapter 1: Introduction

This chapter gives the idea what is the background of the research, some objectives and structure of this document.

Chapter 2: Description Logics

The definition of description logic and its architecture will be discussed in this chapter. Also, the standard reasoning services, satisfiability and subsumption, and especially the non-standard concept contraction and concept abduction which are used in the matchmaking algorithm. There is also an overview of some known description logic systems at the end of this chapter.

Chapter 3:

Chapter 3 focuses on image management, annotation and retrieval, what constraints they have, the current methods and techniques in annotation and retrieval system. It will discuss some important issues in information retrieval and image retrieval. The description logic approach for image retrieval systems is explained in this chapter with the algorithm which will be implemented. How the algorithm works and how to represent image profiles are also the main topics in chapter 3.

Chapter 4:

This part is dedicated to analysis and design process. It begins from knowledge domain and problem analysis. After that, a brief explanation on the knowledge base design and then the programming design. The ontology can be found in this chapter.

Chapter 5:

Chapter 5 shows how the implemented system works from annotation process to the retrieving process and calculating the penalties. Some demonstrations can be found at the end of this chapter.

Chapter 6:

This is the last chapter of the document. It presents some conclusions of the work, advantages and disadvantages of image retrieval system using concept contraction and concept abduction. Some thoughts about what can be or should be done to improve the system for future works are given as ending of this Thesis.

Chapter 2
Description Logics

2.1. What are Description Logics?

Description Logics (DLs) are family of knowledge representation (KR) formalisms which can be used to represent the knowledge of an application domain in a structured and formally well-understood way. The first step is defining the relevant concepts of the domain (its terminology), and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description).

The main characteristic of these languages which distinguishes it to its predecessors is that, they are equipped with a formal logic-based semantics. Another distinguished feature is the emphasis on reasoning as a central service.
Here, reasoning can be interpreted as a process of making inferences through logical thinking. Reasoning allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base.

Knowledge representation itself is the study of how knowledge about the world can be represented and what kinds of reasoning can be done with that knowledge.
Description Logic languages are then viewed as the core of knowledge represen-

tation systems, considering both the structure of a DL knowledge base and its associated reasoning services.

2.2. Architecture of Description Logics

Figure 2.1. shows the architecture of a knowledge representation system based on Description Logics [1].

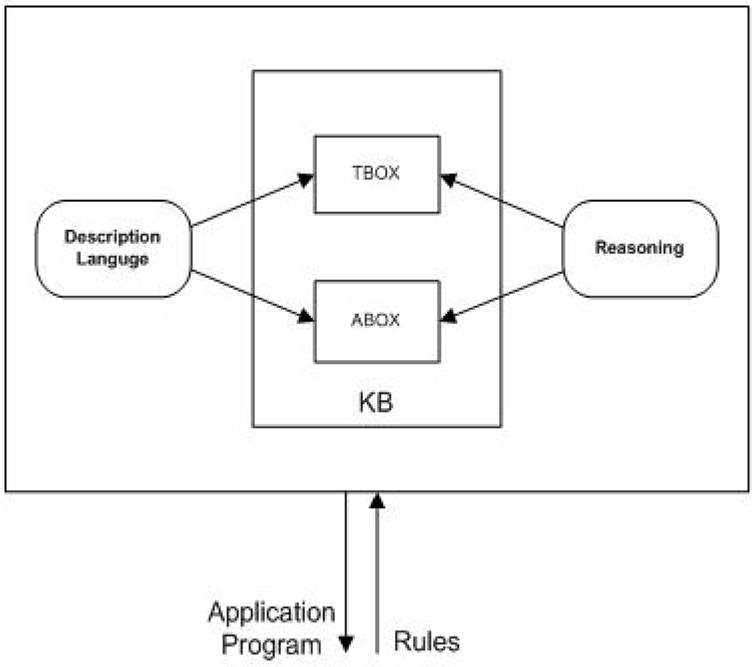


Figure 2.1. Architecture of a knowledge representation system based on Description Logics.

A knowledge representation system based on Description Logics provides facilities to set up knowledge bases, to reason about their content, and to manipulate them.

Two components of a knowledge base (KB) as seen in Figure 2.1. are:

1. the Terminological Box (TBox), which introduces the *terminology*, that is, the vocabulary of an application domain.

2. the Assertional Box (ABox), which contains *assertions* about named individuals in terms of this vocabulary.

The vocabulary comprises two components:

- *concepts*, denote sets of individuals, and
- *roles*, denote binary relationships between those individuals

It is also allowed for the users of all DL systems to build complex descriptions of concepts and roles, in addition to atomic concepts and roles (concept and role names), by using the TBox to assign names to complex descriptions. The language for building descriptions is a characteristic of each DL system, and different systems are distinguished by their description languages.

2.2.1. Description Languages

As stated before, complex descriptions can be built from elementary descriptions, that is *atomic concepts* and *atomic roles*, inductively with *concept constructors*.

The letters A and B will be used for atomic concepts, the letter R for atomic roles, and the letters C and D for concept descriptions The basic description language $\mathcal{AL}$ (attributive language) will be used for a brief explanation since other languages of this family are extensions of $\mathcal{AL}$**.**

Concept descriptions in $\mathcal{AL}$ are formed according to the following syntax rule [1]:

$$\begin{array}{lll}
C, D \longrightarrow & A \mid & \text{(atomic concept)} \\
& \top \mid & \text{(universal concept)} \\
& \bot \mid & \text{(bottom concept)} \\
& \neg A \mid & \text{(atomic negation)} \\
& C \sqcap D \mid & \text{(intersection)} \\
& \forall R.C \mid & \text{(value restriction)} \\
& \exists R.\top & \text{(limited existential quantification)}
\end{array}$$

In attributive language $\mathcal{AL}$, negation can only be applied to atomic concepts, and only the top concept is allowed in the scope of an existential quantification over a role.

For a brief example, let's assume that Person and Male are atomic concepts. An $\mathcal{AL}$ concept Person⊓Male is describing persons that are male while its negation Person⊓¬Male is describing persons that are not male. Next, let's suppose that hasChild is an atomic role, we can use it together with the atomic concepts to compose the concepts [1]:

$$\text{Person} \sqcap \exists\text{hasChild}.\top$$

and

$$\text{Person} \sqcap \exists\text{hasChild.Female}$$

the first denoting those persons that have a child, while the second denoting those persons, all of whose children are female. Persons without a child can be described by the concept Person ⊓ ∀hasChild.⊥. Here, the bottom concept (⊥) is used.

In $\mathcal{AL}$, an Interpretation I consists of a non-empty set Δ^I (the domain of the interpretation) and an interpretation function, that assigns to:

- each concept name A, a subset A^I of Δ^I $(\mathrm{A}^I \subseteq \Delta^I)$
- each role name R, a binary relation R^I over Δ^I $(R^I \subseteq \Delta^I \times \Delta^I)$
- each feature name f, associated with the concrete domain D, a partial function $f^I : \Delta^I \longrightarrow \mathcal{D}$.

The interpretation function is extended to concepts descriptions by following inductive definitions:

$$\begin{aligned} \top^I &= \Delta^I \\ \bot^I &= \varnothing \\ (\neg A)^I &= \Delta^I \setminus A^I \\ (C \sqcap D)^I &= C^I \cap D^I \end{aligned}$$

$$(\forall R.C)^{I} \;=\; \{a \in \Delta^{I} \mid \forall b.(a,b) \in R^{I} \longrightarrow b \in C^{I}\}$$
$$(\exists R.\top)^{I} \;=\; \{a \in \Delta^{I} \mid \exists b.(a,b) \in R^{I}\}$$

For more expressive languages, we can add further constructors to $\mathcal{AL}$, as following:

- The *union* of concepts, written as $C \sqcup D$ and interpreted as:

$$(C \sqcup D)^{I} = C^{I} \sqcup D^{I}$$

- *Full existential quantification*, written as $\exists R.C$ and interpreted as:

$$(\exists R.C)^{I} = \{a \in \Delta^{I} \mid \forall b.(a,b) \in R^{I} \wedge b \in C^{I}\}$$

- *Number restrictions*, written as $\geqslant nR$ (at-least restriction) and as $\leq nR$ (at-most restriction), where n ranges over the nonnegative integers. They are interpreted as:

$$(\geqslant nR)^{I} = \{\, a \in \Delta^{I} \mid |\{b \mid (a,b) \in R^{I}\}| \geqslant n \,\},$$

and

$$(\leqslant nR)^{I} = \{\, a \in \Delta^{I} \mid |\{b \mid (a,b) \in R^{I}\}| \leqslant n \,\},$$

respectively, where “| . |” denotes the cardinality of a set.

- The *negation* of arbitrary concepts, written as $\neg C$ and interpreted as:

$$(\neg C)^{I} = \Delta^{I} \setminus C^{I}.$$

With those additional constructors, now we can describe, for instance, persons that have no more than 2 children or at least three children, one of which is male:

$$\text{Person} \sqcap (\leqslant 2\ \text{hasChild} \sqcup (\geqslant 3\ \text{hasChild} \sqcup \exists \text{hasChild.Male}))$$

2.2.2. Terminological Box (TBox)

After discussing how to form complex descriptions of concepts to describe classes of objects, now e look at terminological axioms which make statements about how concepts or roles are related to each other.

In most general situation, terminological axioms have the forms either inclusions or equalities. The inclusions are denoted as $C \sqsubseteq D$ or $R \sqsubseteq S$, and the equalities are denoted as $C = D$ or $R = S$, where C, D are concepts and R, S are roles.

The basic form of declaration in a TBox is a concept definition, that is, the definition of a new concept in terms of other previously defined concepts, or an equality hose left hand side is an atomic concept. Definitions are used to introduce symbolic names for complex descriptions. For example, a woman can be defined as a female person by the following axiom:

$$\text{Woman} = \text{Person} \sqcap \text{Female}$$

The above declaration provides such sufficient and necessary conditions for classifying an individual as a woman. Another example which is using role

$$\text{Mother} = \text{Woman} \sqcap \exists \text{hasChild.Person}$$

has gave an association to the description of the right hand side the name Mother. If Father is defined analogously to Mother, we can define Parent as

$$\text{Parent} = \text{Mother} \sqcup \text{Father}$$

There are some important common assumptions usually made about DL terminologies:

- only one definition for a concept name is allowed
- definitions are acyclic in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that directly refer to them.

This kind of restriction is common to many DL knowledge bases and implies that every defined concept can be expanded in a unique way into a complex expression containing only atomic concepts by replacing every defined concept with the right-hand side of its definition.

$$\begin{aligned} \text{Woman} &= \text{Person} \sqcap \text{Female} \\ \text{Man} &= \text{Person} \sqcap \neg\text{Woman} \end{aligned}$$

Mother	=	Woman ⊓ ∃hasChild.Person
Father	=	Man ⊓ ∃hasChild.Person
Parent	=	Mother ⊔ Father
Brother	=	Man ⊓ $\geqslant_1$ hasSibling
Sister	=	Woman ⊓ $\geqslant_1$ hasSibling
Grandmother	=	Mother ⊓ ∃hasChild.Parent
Wife	=	Woman ⊓ ∃hasSpouse.Man

Figure 2.2. A simple TBox with concepts about family relationships.

Below are the replacements of every defined concepts in Figure 2.2. to its definition:

Woman	=	Person ⊓ Female
Man	=	Person ⊓ ¬ (Person ⊓ Female)
Mother	=	(Person ⊓ Female) ⊓ ∃hasChild.Person
Father	=	(Person ⊓ ¬ (Person ⊓ Female)) ⊓ ∃hasChild.Person
Parent	=	((Person ⊓ Female) ⊓ ∃hasChild.Person) ⊔ (¬ (Person ⊓ Female) ⊓ ∃hasChild.Person))
Brother	=	¬ (Person ⊓ Female) ⊓ $\geq$1 hasSibling
Sister	=	(Person ⊓ Female) ⊓ $\geq$1 hasSibling
Grandmother	=	((Person ⊓ Female) ⊓ ∃hasChild.Person) ⊓ ∃hasChild.(((Person ⊓ ¬ (Person ⊓ Female)) ∃hasChild.Person) ⊔ ((Person ⊓ Female) ∃hasChild.Person))
Wife	=	(Person ⊓ Female) ⊓ ∃hasSpouse.(¬ (Person ⊓ Female))

Figure 2.3. The expansion of Family TBox

2.2.3. Assertional Box (ABox)

The ABox, which is also known as world description, contains assertions about individuals. Two kinds of assertions in an ABox are

$$C(a) \quad \text{and} \quad R(b,c)$$

where a, b, c are individual names with C as concepts and R as roles. In concept assertions $C(a)$, one states that a belong to C. And in role assertions $R(b,c)$, one states that c is the filler of the role R for b. For example,

Mother(EMMA)

states that the individual EMMA is a mother, and

hasChild(EMMA, SUSAN)

describes that EMMA has SUSAN as a child.

A DL system not only stores terminologies and assertions, but also offers services that *reason* about them. The basic reasoning services of a DL system will be discussed in the next chapter.

2.2.4. Reasoning Services

2.2.4.1. Standard Reasoning Services

The basic reasoning services on concept expression are concept subsumption and concept satisfiability.

If supply is denoted by the concept C and demand by the concept D, unsatisfiability of $C \sqcap D$ identifies the incompatible between supply and demand, and satisfiability identifies potential partners between them.

Determining subsumption (typically written as $C \sqsubseteq D$) is the problem whether the concept D is more general than the concept C. In other words, a concept C is subsumed by a concept D if in every model of $\mathcal{T}$, the set denoted by C is the subset of the set denoted by D.

For example, w.r.t. Family TBox in Figure 2.2., it can be verified whether Woman $\sqsubseteq$ Wife (Woman is subsumed by Wife), or the other way around Wife $\sqsubseteq$ Woman (Wife is subsumed by Woman).

The basic reasoning mechanism provided by DL systems can check the subsumption of concepts. Hence, it is also sufficient to implement the other inferences, one of them is reduction to subsumption as the following [1]:

For concepts C, D, w.r.t. a TBox, we have:

- *C is unsatisfiable* $\Leftrightarrow$ *C is subsumed by* $\perp$
- *C and D are equivalent* $\Leftrightarrow$ *C is subsumed by D and D is subsumed by C*
- *C and D are disjoint* $\Leftrightarrow$ $C \sqcap D$ *is subsumed by* $\perp$

There also exists other relationships between concepts which can be reduced to subsumption and (un)satisfiability, equivalence and disjointness. The formal definitions of these properties are as follows: [1]

Let $\mathcal{T}$ be a TBox

- Satisfiability: A concept C is satisfiable with respect to $\mathcal{T}$ if there exists a model I of $\mathcal{T}$ such that C^I is nonempty. In this case we say also that I is a model of C.
- Subsumption: A concept C is subsumed by a concept D with respect to $\mathcal{T}$ if $C^I \subseteq D^I$ for every model I of $\mathcal{T}$. In this case we write $C \sqsubseteq_{\mathcal{T}} D$ or $\mathcal{T} \models C \sqsubseteq D$.
- Equivalence: Two concepts C and D are equivalent with respect to $\mathcal{T}$ if $C^I = D^I$ for every model I of $\mathcal{T}$. In this case we write $C \equiv_{\mathcal{T}} D$ or $\mathcal{T} \models C \equiv D$.
- Disjointness: Two concepts C and D are equivalent with respect to $\mathcal{T}$ if $C^I \cap D^I = \emptyset$ for every model I of $\mathcal{T}$.

For instances, in TBox the concept expression:

$$\text{Woman} \sqcap \text{Mother}$$

is satisfiable with regards to the Family TBox defined in Figure 2, and

$$\neg\text{Woman} \sqcap \text{Mother}$$

is unsatisfiable, since if we unfold this:

$$\begin{gathered}\neg\text{Woman} \sqcap \text{Mother} \equiv \\ \neg(\text{Person} \sqcap \neg\text{Man}) \sqcap (\text{Woman} \sqcap \exists\text{hasChild.Person}) \equiv \\ (\neg\text{Person} \sqcup \text{Man}) \sqcap \text{Person} \sqcap \neg\text{Man} \sqcap \exists\text{hasChild.Person} \equiv \\ \mathbf{Man} \sqcap \neg\mathbf{Man} \sqcap \exists\text{hasChild.Person} \equiv \bot\end{gathered}$$

We have the bottom/empty concept as a result ($\bot$).

2.2.4.2. Non-Standard Reasoning Services

Colucci et.al. have defined non-standards reasoning services (inferences) in [3], named contraction and abduction.

For matchmaking services in the WWW, ranking of potential counteroffers is very critical to make the service useful for its users. Since image retrieval system does just the same thing with such a system, ranking of potential images which match the user's demand has become a very significant task.

With the standard reasoning services for TBox which have been mentioned before, concept subsumption and satisfiability, it is not possible to perform the ranking task. Therefore the non-standard reasoning services for concepts which have this capability are needed. Those reasoning services are known as concept abduction and concept contraction.

Concept Contraction

Concept contraction extends concept satisfiability. We have concept *C* as supply, concept *D* as demand and the demander is the one who is actively starting the search, if their conjunction $C \sqcap D$ is unsatisfiable in the TBox $\mathcal{T}$, the aim is to retract requirements in D to obtain a concept K (for Keep) such that $K \sqcap C$ is satisfiable in $\mathcal{T}$.

The demander is weakening or even removing his requests to investigate whether what is left of the original request is still worth an interest. A user is interested in what he must trade to initiate the transaction; a concept G (for Give Up) such that D was made by G and K, that is, $S \equiv G \sqcap K$.

Definition 1: *Let $\mathcal{L}$ be a DL. C, D, be two concepts in $\mathcal{L}$, and $\mathcal{T}$ be a set of axioms in $\mathcal{L}$, where both C and D are satisfiable in $\mathcal{T}$. A Concept Contraction Problem (CCP), identified by $(\mathcal{L}, \mathcal{C}, \mathcal{D}, \mathcal{T})$, is finding a pair of concepts $(G, K) \in \mathcal{L} \times \mathcal{L}$ such that $\mathcal{T} \models C \equiv G \sqcap K$, and $K \sqcap D$ is satisfiable in $\mathcal{T}$. We call K a contraction of C according to D and T.* [3]

The symbol $\mathcal{Q}$ is used for a CCP, and the set of all solutions to a CCP c is denoted by $SOLCCP(\mathcal{Q})$. Note that there is always the trivial solutions $(G, K) = (C, \top)$ to a CCP. This solution is the most drastic contraction that gives up everything of C. On the other hand, when $C \sqcap D$ is satisfiable in $\mathcal{T}$, the best possible solution is $(\top, C)$, that is, give up nothing if possible. Since one wants to give up as little as possible, some minimality in the contraction must be defined. The subsumption relation between concepts w.r.t. a TBox T is denoted by $\sqsubseteq_{\mathcal{T}}$.

Definition 2: *Let $\mathcal{Q} = (\mathcal{L}, \mathcal{C}, \mathcal{D}, \mathcal{T})$ be a CCP. The set $SOLCCP_{\sqsubseteq}(\mathcal{Q})$ is the subset of solution (G, K) in $SOLCCP(\mathcal{Q})$ such that G is maximal under . The set $SOLCCP_{\leq}(\mathcal{Q})$ is the subset of $SOLCCP(\mathcal{Q})$ such that G has minimum length.* [3]

Even if contraction has been performed and the consistency between supply and demand has been recovered, partial specifications still need to be solved. It could be in such a situation that the supply does not explicitly imply the demand, although they are compatible. Hence, it is necessary to assess what should be hypothesized in the supply in order to start the transaction with the demand. Therefore we need the other non-standard inference service, concept abduction which will be discussed in the next section.

Concept Abduction

Concept abduction extends concept subsumption in particular, by providing new concept *H* when *C* is not subsumed by *D*.

Definition 3: *Let $\mathcal{L}$ be a DL. C, D, be two concepts in $\mathcal{L}$, and $\mathcal{T}$ be a set of axioms in $\mathcal{L}$, where both C and D are satisfiable in $\mathcal{T}$. A Concept Abduction Problem (CAP), identified by ($\mathcal{L}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{T}$), is finding a concept $H \in \mathcal{L}$ such that $\mathcal{T} \models C \sqcap H \sqsubseteq D$, and moreover $C \sqcap H$ is satisfiable in $\mathcal{T}$. We call H a hypothesis about C according to D and T.* [3]

The set of all solutions to a CAP $\mathcal{P}$ is denoted with *SOLCAP*($\mathcal{P}$) where $\mathcal{P}$ is a symbol for a CAP. Note that in the definition it is limited to satisfiable *C* and *D* since *C* unsatisfiable implies that the CAP has no solution at all, while *D* unsatisfiable leads to counterintuitive results ($\neg C$ would be a solution in that case).

In CAP, there is no distinction between manifestations and hypotheses, which is very common when using abduction for diagnosis.

2.3. Description Logic Systems

A DL system is a reasoner for knowledge base; a DL system which is expressive and efficient is considered to be a good one although there are still some conditions that should be taken into account like security, size (should be small and simple) and the user friendliness. All DL systems provide subsumption and satisfiability as standard inference services. Some known Description Logic Systems are:

- CLASSIC - A description logic from AT&T Laboratories implemented first in Lisp and later in C and C++. It has aimed to balance expressive power and computational complexity and is one of the less expressive implemented systems. It is the basis of a number of configuration and data mining applications, notably PROSE from AT&T and Lucent and the Management Discovery Tool from NCR.

- DLP
- FaCT
- FLEX - A description logic from the Technical University of Berlin. It is the basis of a very large natural language application.
- KRIS - An expressive description logic originally from DFKI (the German Institute for Artificial Intelligence).
- Loom - A highly expressive description logic-based system from University of California - Information Sciences Institute. It is the basis of many applications funded by ARPA.
- RacerPro - Renamed ABox and Concept Expression Reasoner.

DLP, FaCT and RacerPro belong to the new optimized generation of very expressive but sound and complete DL systems. RacerPro is used as a reasoner for this Thesis' research.

2.3.1. More about RacerPro

Some key features of the RacerPro are:

- compiling the source code for various operating systems
- versions for large-scale applications (i.e., on 64bit computer systems)
- developing specific versions tailored to various ways description logic systems are used in applications (e.g., for processing OWL documents and processing specific patterns of queries).
- integration of RacerPro into the user's computational environment
- importing data from relational databases

Chapter 3

A Description Logic Approach for Image Retrieval Systems

This Thesis, like the title says, will try to evaluate the non-standard reasoning services in DLs, concept contraction and concept abduction, using the algorithm given by Cali et.al. (2004) which can be found in their paper [4]. The algorithm for matching user profiles in [4] is tailored for dating services but it can be modified to meet this Thesis' needs.

This chapter discusses the image retrieval and image annotation and the algorithm given in [4], how to modify and implement it in an image retrieval system.

3.1. Overview

As the old adage goes, a picture says more than a thousand words, so with that in human life, from the prehistoric time where humans made paintings on cave walls up to now, where millions of pictures are produced every single day.

It was impossible to transport cave-wall paintings so it can be seen by others. But since humans have made writings and paintings on transportable materials like woods, stones, canvas and papers, it is made possible to move it from one place to another and even to reproduce it.

Another human invention which had a great impact on humankind is the computer. Together with the evolving of photography, they have a great product named digital camera. Since then, reproducing an image can be done within a second.

The born of World Wide Web had made the world "wider", it does not have boundaries anymore. The World Wide Web is a world without borders. There is no easier way to distribute photographs, pictures, images (or whatever you named it) other than through World Wide Web. But that's not the only way where people search for images, exchange pictures or buy and sell them.

One may ask, why do people search for images? The answers for it may be as followings:

- To make a clear perception of something, examples for this are images using for illustration on magazines / newspapers, images of two species of orchids which are totally difference but have a same appearance.
- To collect certain images, a dog lover who collects dog pictures is a good example for it.
- To satisfy one's curiousness, for example, someone just wants to know how the new wife of Prince Charles looks like.

Those are just a few reasons from many others, but in general, there are three main entities what people normally search for:

- Object
- Event (happening)
- Location

The next questions would be these HOWs:

- How to retrieve the right images?
- How to deliver an appropriate number of results?

- How to satisfy the user?
- How not to make the user frustrating?
- Altogether, it's just one question remaining: how to design a good image retrieval system.

Before we go further to discuss about image retrieval, let's take a look at how can one query images. According to Eakins and Graham [5], there are three characteristics of image queries:

- *Level 1* comprises retrieval by *primitive* features such as color, texture, shape or the spatial location of image elements. Examples of such queries might be like "find images with bluish background", "find images containing red brown corners" or a most general one like "find more images like this". This level of image retrieval is often called content-based image retrieval.
- *Level 2* comprises retrieval by *derived* (sometimes known as *logical*) features, involving some degree of logical inference about the identity of the objects depicted in the image. It can usefully be divided further into retrieval of objects of a given type (e.g. "find sunset images") and retrieval of individual objects or persons ("find images of Liberty Statue").
- *Level 3* comprises retrieval by *abstract* attributes, involving a significant amount of high-level reasoning about the meaning and purpose of the objects or scenes depicted. This kind of retrieval can be named events or types of activity (e.g. "find images of a wedding ceremony") or emotional significance ("find images depicting happiness").

There is a significant gap between level 1 and level 2. Some authors like Gudivada and Raghavan [10] refers to level 2 and 3 together as semantic image retrieval, and thus the gap between levels 1 and 2 as semantic gap [5].

Inoue in his paper [23] has defined two types of image retrieval as Query-by-Text (QbT) and Query-by-Example (QbE). The QbT is a cross-medium retrieval since queries are texts and targets are images, as opposed to QbE, a mono-medium retrieval where queries and targets are images.

3.2. Image Retrieval Systems: Current Methods and Techniques

3.2.1. Low-Level Retrieval

Regarding to the three characteristics of image queries by Eakins and Graham [5] (in section 3.1.), the first level of image query is the primitive level which comprises retrieval by primitive features such as color, shape and texture. This kind of image retrieval is called content-based image retrieval (usually abbreviated as CBIR), also known as query by image content (QBIC) and content-based visual information retrieval (CBVIR).

There is no or only a little human intervention needed in the annotation process of this kind of image retrieval. In CBIR, an image is represented by its signature, which is composed of features derived from its physical contents (i.e. pixel values) [11]. Users of these systems "naively expect to search for a specific object or person", but in reality can only search for images with a similar distribution of image properties [12]. Some known CBIR are:

- CIRES (http://amazon.ece.utexas.edu/~qasim/research.htm)
- SIMPLIcity and ALIP (http://wang.ist.psu.edu/IMAGE/)
- GIFT, The GNU Image Finding Tool (http://www.gnu.org/software/gift/)
- SIMBA (http://simba.informatik.uni-freiburg.de/)
- imgSeek (http://www.imgseek.net/)
- Cortina (http://cortina.ece.ucsb.edu/)
- Octagon (http://users.utu.fi/jukvii/octagon)

The CBIR method will not be discussed more in great detail since this Thesis' research will be focused on semantic level of image queries.

The next level of image queries is the semantic level, which comprises retrieval by logical features and abstract attributes.

3.2.2. High(er)-Level Retrieval

As an opposed to CBIR, Inoue refers the high-level retrieval as Annotation-Based Image Retrieval or ABIR [23]. Currently more researches have been conducted in CBIR than ABIR. Researchers working on CBIR claim that ABIR has limitations [23]. Brahmi and Ziou have mentioned some of these limitations [24]: manual image annotation is time consuming, thus it is costly and annotation made by human is very subjective. Other authors like Sclaroff et.al. mentioned that some images are difficult to annotate since their content cannot be described with words [12]. Examples for the last could be a radiology image or a satellite picture.

In spite of the fact that CBIR is a more popular topic among the researchers, the CBIR community is becoming aware of this: “It is becoming clear in the image retrieval community that content-based Image Retrieval is not a replacement of, but rather a complementary component to, the text-based Image Retrieval. Only the integration of the two can result in satisfactory retrieval performance” [29]. This awareness has stimulated research into the addition of other text-based techniques to image retrieval system [11]. Jörgensen has done a summary of several methods and techniques used in high(er)-level image retrieval [11], some of them are:

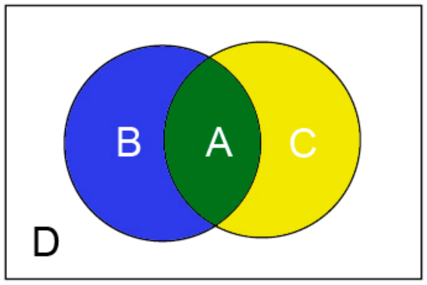
- Methods for exploiting linguistic context in image interpretation, supplementing feature-based approach. The associated text is used as a set of constraints to identify the (physical) content of an image. The author of PICTION*) used the information obtained from the associated caption to identify human faces in newspaper photographs.
- Using a technique called Latent Semantic Analysis (LSA), LSA works by using statistical techniques to associate words to the “semantic” concept of a given documents and it assumes that there is an underlying or “latent” structure in the patterns of words usage across “documents” (text, paragraphs or sentences).

- Semiautomatic methods have been experimented by the authors of WebSEEk$^{*)}$ system. The system uses text derived from image addresses and HTML tags

3.3. Important Issues in Image Retrieval

3.3.1. Information Retrieval: Precision and Recall

In information retrieval, there are two basic measures in evaluating search effectiveness called precision and recall. Precision is the proportion of relevant documents retrieved to the total numbers of retrieved documents and recall is the proportion of the relevant documents retrieved to the number of all relevant documents.



	Relevant	Not Relevant
Retrieved	**A**	**C**
Not Retrieved	**B**	**D**

Figure 3.1. Precision and Recall

A = relevant documents - retrieved

B = relevant documents – not retrieved

C = irrelevant documents – retrieved

D = irrelevant documents – not retrieved

A + B = all relevant documents in the database

A + C = all retrieved documents

$$\text{Precision} = \frac{A}{A+B} \text{ x } 100\% \qquad \text{Recall} = \frac{A}{A+C} \text{ x } 100\%$$

There will be perfection if precision and recall are both 100%, it means all retrieved documents are relevant documents, but it is very hard (if not impossible) to achieve. In reality, we can only achieve a good balance between precision and recall. Figure 3.2. shows two venn diagrams, one depicts perfection (precision = recall = 100%), while the other depicts bad precision.

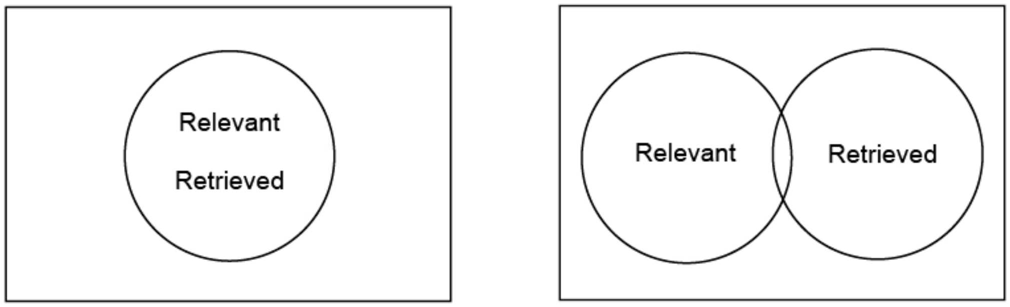


Figure 3.2. Perfection in information retrieval is depicted in the left diagram while bad precision is depicted in the right diagram.

3.3.2. User's Perspective: Satisfaction and Frustration

Two main processes in an image retrieval system are image annotation and image retrieval. In a traditional image retrieval systems which are metadata-based, the first stands for the process of creating the metadata in order to describe the images while the later stands for the process of finding images through the metadata. Nevertheless, image annotation is a key of success of such systems where the success is defined by its user satisfaction.

Poorly annotated images give the user small number of results, thus it can cause user dissatisfaction. But excessively annotated images, its popular term "keywords spamming", do not guarantee that the user will then be satisfied with a large amount of results he gets. Figure 3.x depicts the correlation between number of results and user satisfaction.

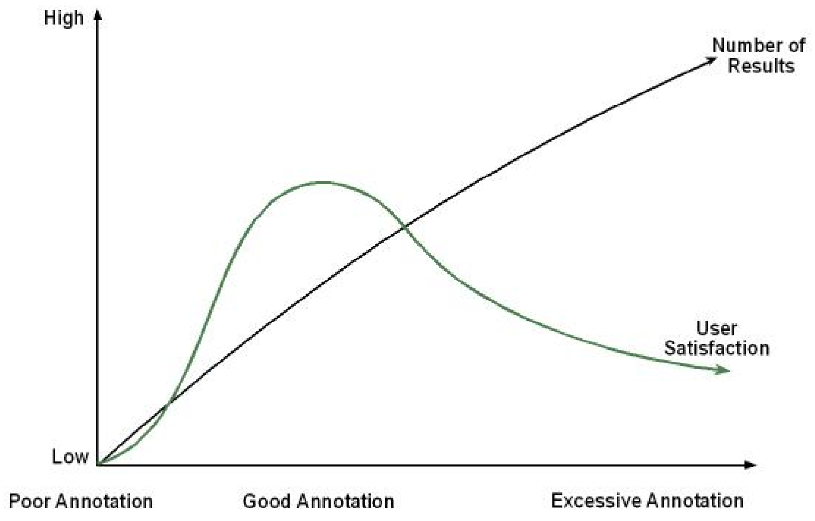


Figure 3.3. The correlation between number of results and user satisfaction with the annotation.

On the other hand, excessively annotated images give the user a large number of results though, but it causes high user frustration since the user gets what he doesn't want. As we can see in figure 3.x., it shows the correlation between number of results and the level of user frustration.

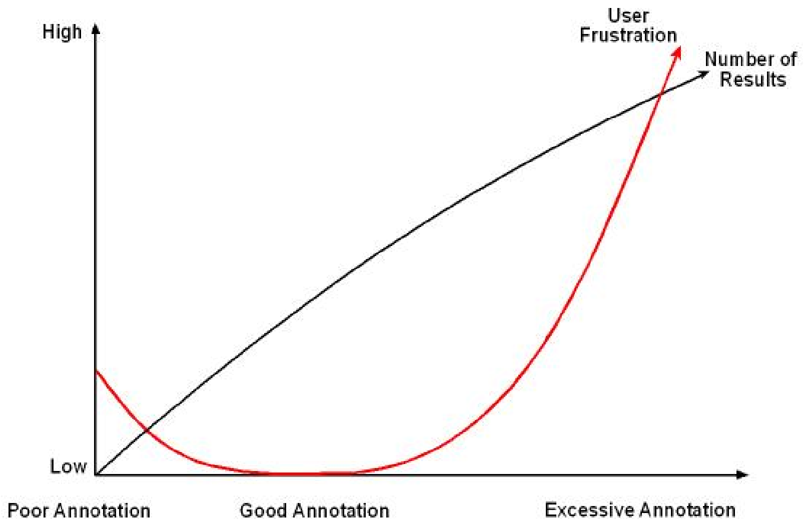


Figure 3.4. The correlation between number of results and user frustration with the annotation.

If we merge both diagrams of figure 3.x. and figure 3.x., we have a complete illustration of how a good image retrieval system should be. Figure 3.x. shows us the complete illustration.

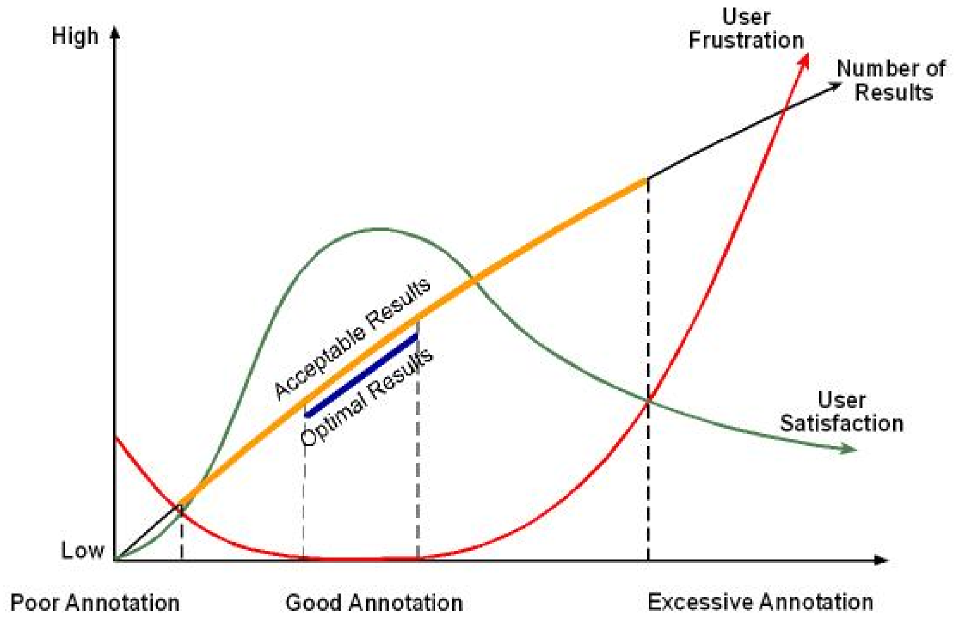


Figure 3.5. A merge of Figure 3.3. and Figure 3.4.

From the above diagram we have the so called "acceptable results" as shown by the yellow line and "optimal results" as shown by the blue line. The yellow line shows a range of results where user satisfaction is equal or greater than user frustration and the blue line shows a smaller range of results where the level of user frustration is equal or almost zero. The diagram also indicates that the highest level of user satisfaction is in the range of blue line.

	Poor Annotation	Good Annotation	Excessive Annotation
Numbers of results	**Low**	**Low – Medium**	**High**
User satisfaction	**Low**	**Medium – High**	**Medium – Low**

User frustration	**Medium – Low**	**Low**	**Medium - High**

Although good annotation does not provide user with large number of results, it delivers user with high accuracy and hence it satisfies the user.

Now let's take a look at some examples of image annotation below:

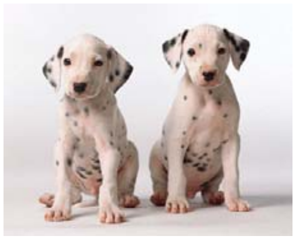
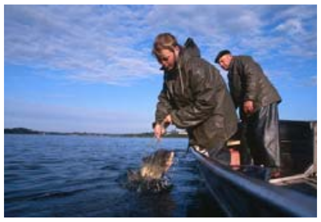
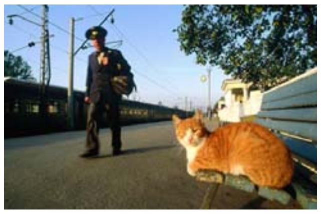
	Poor Annotation	Good Annotation	Excessive Annotation
	dogs	dalmatian dogs	dogs, dalmatian dogs, pet, cute dogs, domestic dogs, standing dogs
	man, fish	fishermen, small boat, fishing, lake	man, fishermen, fish, big fish, fishing, fishing boat, lake, lake tahoe
	man, cat, train	policeman, cat, running train	policeman, postman, clerk, officer, train, cat, wild cat, brown cat, brown wild cat

Figure 3.6. Some examples of poor, good and excessive annotation.

A good image retrieval system must have a good annotation which really describes what an image is or consists of and a reliable back-end which can retrieve images based on the retrievers' (users') needs with the possibility to rank the result based on .

3.4. Contraction and Abduction Algorithm for Matchmaking in an Image Retrieval System

3.4.1. The Matchmaking Algorithm

Cali et.al. [4] have designed an algorithm based on description logic for matching user profiles. The algorithm is using the non-standard reasoning services, concept contraction and concept abduction explained in section 2.2.4.2. The user profiles are tailored for dating service, though with small modifications the same framework can be used for different applications, for example finding job and in classified ads. The term for this problem domain is known as *matchmaking*. In this case, the algorithm will be modified for matching image profiles in image retrieval system.

The original algorithm can be found in **Appendix A**, below is the modified one used in this Thesis' research.

```
---------------------------------------- Begin Algorithm ----------------------------------------
Algorithm CalculatePenalty
  Input demand profile $P_d$, supply profile $P_s$, concept Hierarchy $\mathcal{H}$
  Output real value penalty $\geqslant 0$
  penalty := 0;
  // Contraction
  foreach $A_d \in Names(P_d)$ do
    if there exists $A_s \in Names(P_s)$
      such that $\mathcal{H} \models A_d \sqsubseteq \neg A_s$
    then remove $A_d$ from $P_d$
        penalty := penalty + $\Pi_c(A_d)$
  foreach $p_d(f) \in Features(P_{d+})$ do
    if there exists $p_s(f) \in Features(P_s)$
      such that $\exists x.\, p_d(x) \wedge p_s(x)$ is unsatisfiable in the domain associated to $f$
    then remove $p_d(f)$ from $P_d$
        penalty := penalty + $\Pi_{cf}(p_d(f), p_s(f))$
```

```
foreach $\exists$has-segment.$(C_d \sqcap \geqslant x_d$ (level)$) \in$ Segments( $P_d$ ) do
  foreach $\forall$has-segment.$(\neg C_s \sqcup \leqslant x_s$ (level)$) \in$ NoSegments( $P_s$ ) do
    if $\mathcal{H} \models C_d \sqsubseteq C_s$ and $x_d \geqslant x_s$
    then replace $\exists$has-segment.$(C_d \sqcap \geqslant x_d$ (level)$)$ in $P_d$
            with $\exists$has-segment.$(C_d \sqcap \geqslant x_s$ (level)$)$
          penalty := penalty + $\Pi_{cl}(x_d, x_s)$
foreach $\forall$has-segment.$(\neg C_d \sqcup \leqslant x_d$ (level)$) \in$ NoSegments( $P_d$ ) do
  foreach $\exists$has-segment.$(C_s - \geqslant x_s$ (level)$) \in$ Segments( $P_s$ ) do
    if $\mathcal{H} \models C_s \sqsubseteq C_d$ and $x_d \leqslant x_s$
    then replace $\forall$has-segment.$(\neg C_d \sqcup \leqslant x_d$ (level)$)$ in $P_d$
            with $\forall$has-segment.$(\neg C_d \sqcup \leqslant x_s$ (level)$)$
          penalty := penalty + $\Pi_{cl}(x_s, x_d)$

// Abduction
foreach $A_d \in$ Names( $P_d$ ) do
  if there does not exist $A_s \in$ Names( $P_s$ ) such that $\mathcal{H} \models A_s \sqsubseteq A_d$
  then add $A_d$ to $P_s$
        penalty := penalty + $\Pi_a( A_d )$
  foreach $p_d(f) \in$ Features( $P_d$ ) do
    if there exists $p_s(f) \in$ Features( $P_s$ ) . $p_d(x) \wedge p_s(x)$
    then if $\forall x. p_s(x) \Rightarrow p_d(x)$ is false in the domain associated to $f$
          then add $p_d(f)$ to $P_s$
                penalty := penalty + $\Pi_{af}(p_d(f), p_s(f))$
    else add $p_d(f)$ to $P_s$
          penalty := penalty + $\Pi_{af}(p_d(f), \top(f))$
foreach $\exists$has-segment.$(C_d \sqcap \geqslant x_d$ (level)$) \in$ Segments( $P_d$ ) do
  if there does not exist $\exists$has-segment.$(C_s \sqcap \geqslant x_s$ (level)$) \in$ Segments( $P_s$ )
    such that $\mathcal{H} \models C_s \sqsubseteq C_d$ and $x_s \geqslant x_d$
    then if there exists ∃has-segment.($C_s \sqcap \geqslant x_s$ (level)) ∈ Segments( $P_s$ )
            such that $\mathcal{H} \models C_s \sqsubseteq C_d$
        then let ∃has-segment.($C_s \sqcap \geqslant x_s$ (level)) be the concept in Segments( $P_s$ )
                with maximum $x_s$ among those for which $\mathcal{H} \models C_s \sqsubseteq C_d$ holds
            penalty := penalty + $\Pi_{al}$ ($x_d$, $x_s$)
        else penalty := penalty + $\Pi_a$ (∃has-segment.($C_d \sqcap \geqslant x_d$ (level)))
        add ∃has-segment.($C_d \sqcap \geqslant x_d$ (level)) to $P_s$
  foreach ∀has-segment.($\neg C_d \sqcup \leqslant x_d$ (level)) ∈ NoSegments( $P_d$ ) do
    if there does not exist ∀has-segment.($\neg C_s \sqcup \leqslant x_s$ (level)) ∈ NoSegments( $P_d$ )
      such that $\mathcal{H} \models C_d \sqsubseteq C_s$ and $x_d \geqslant x_s$
    then if there exists ∀has-segment.($\neg C_s \sqcup \leqslant x_s$ (level)) ∈ NoSegments( $P_d$ )
            such that $\mathcal{H} \models C_d \sqsubseteq C_s$
        then let ∀has-segment.($\neg C_s \sqcup \leqslant x_s$ (level)) be the concept in Segments( $P_s$ )
                with minimum $x_s$ among those for which $\mathcal{H} \models C_d \sqsubseteq C_s$ holds
            penalty := penalty + $\Pi_{al}$ ($x_s$, $x_d$)
        else penalty := penalty + $\Pi_a$ (∀has-segment.($\neg C_d \sqcup \leqslant x_d$ (level)))
        add ∀has-segment.($\neg C_d \sqcup \leqslant x_d$ (level)) to $P_s$
    return penalty
------------------------------------------- End Algorithm -------------------------------------------
```

3.4.2. Representing Image Profiles to Fit the Algorithm

The conjunction of an image profile P is composed of [4]:

- A conjunction of atomic concepts to represent atomic properties, denoted as <u>*Names(P)*</u>.

  For examples: `film-image, digital-image, bw-image.`

- A conjunction of concepts which have form *p(f)* represent physical characteristics. The predicate *p* can be one of $=_\ell(.)$, $\geq_\ell(.)$, and $\leq_\ell(.)$, l is the value of the concrete domain associated to *f*.

For examples: ISO value and image resolution, for brevity, only one feature will be used that is ISO value.

- A conjunction of concepts which have form $\exists R.(C \sqcap \geq_x (level))$, where R is a role, in this case, the role `has-segment`, C is a conjunction of a concept names and $0 \leq x \leq 1$. A concept in this form represents a category in a concept C with level at least x, and the set of such concepts is denoted with *Segments(P)*.

  For example: $\exists$`has-segment`.(`DOG` $\sqcap \geqslant_{0.75}$ (`level`))
- A conjunction of concepts which have form $\forall R.(\neg C \sqcup \leq_x (level))$, where R is a role, in this case, the role `has-segment`, C is a conjunction of a concept names and $0 \leq x \leq 1$. A concept in this form represents a category in a concept C with level at least x, and the set of such concepts is denoted with *NoSegments(P)*

  For example: $\forall$`has-segment`.($\neg$`CAT` $\sqcap \leqslant_{0.30}$ (`level`))

3.4.3. Contraction Part

In contraction phase, if $P_d \sqcap P_s$ is not satisfiable in Hierarchy $\mathcal{H}$, then it removes or weakens conjuncts from P_d so as to make $P_d \sqcap P_s$ satisfiable in $\mathcal{H}$ and adds a penalty. In other words, if the supplier has something that the demander does not like, the demander gives up or weakens his request in order to make the profile match.

3.4.4. Abduction Part

In the abduction phase, when the demander wants something that the supplier does not provide explicitly, it assumes that the supplier may or may not satisfy the demander's request, it then adds or strengthens conjuncts in P_s to make $\mathcal{H} \models P_s \sqsubseteq P_d$.

3.4.5. Penalty Functions

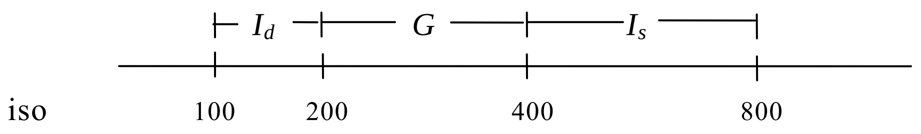
The penalty functions given in [4]:

In the contraction phase, the penalty function for the predicate restriction $p_d(f)$ and $p_s(f)$ is:

$$\Pi_{cf}\ (p_d(f), p_s(f)) = \frac{|G|}{|I_d \cup I_s \cup G|}$$

where I_d and are I_s are the intervals associated to p_d and p_s respectively, and G is the gap between them.

For instance, a supply P_s with range predicate ($p_s(f)$) $(\geq 400 \wedge \ \leq 800)$ (`iso`) which does not match the demand P_d for images with range predicate ($p_d(f)$) $(\geq 100 \wedge \ \leq 200)$ (`iso`) is shown in the diagram below:



I_d = 100; G = 200; I_s = 400

Then we have $\Pi_{cf}\ ((\geq 100 \wedge \leq 200), (\geq 400 \wedge \leq 800)) = \frac{100}{100+400+200} = \frac{1}{7}$

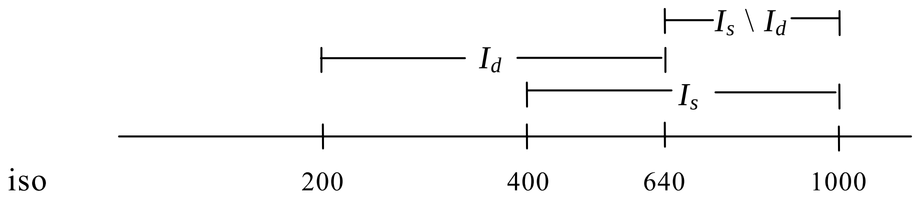
Note that if both $p_d(f)$ and $p_s(f)$ are equality predicates instead of range predicates as in the above example, then we get a highest penalty, since I_d and I_s equal 0 (means no interval), therefore:

$$\Pi_{cf}\ (p_d(f), p_s(f)) = \frac{|G|}{|G|} = 1$$

Also note that there is no gap G in the abduction phase and since $\forall x . p_s(x) \Rightarrow p_d(x)$ is false in the domain associated to f, we have $|I_s| > 0$. The penalty function for the predicate restriction $p_d(f)$ and $p_s(f)$ in the abduction phase is as follows:

$$\Pi_{af}\ (p_d(f), p_s(f)) = \frac{|I_s \setminus I_d|}{|I_s|}$$

For instance, a supply ($p_s(f)$) $(\geq 400 \wedge \leq 1000)$ (`iso`) which does not explicitly satisfy the demand ($p_d(f)$) $(\geq 200 \wedge \leq 640)$ (`iso`) is shown in the diagram below:



$I_s = 600;\ I_s \setminus I_d = 360$

The penalty is calculated as follows:

$$\Pi_{af}\ ((\geq 200 \wedge \leq 640), (\geq 400 \wedge \leq 1000)) = \frac{360}{600} = \frac{3}{5}$$

Given $x_d, x_s \in [0,1]$, $\Pi_{cl}(x_d, x_s) = x_d - x_s$ and $\Pi_{al}(x_d, x_s) = \frac{x_d - x_s}{1 - x_s}$

For $C_d = \sqcap_{i=1}^{n} A_i$

$$\Pi_a(\exists \text{hasCategory}.(C_d \sqcap \geq x_d \text{ (level)})) = x_d.\sum_{i=1}^{n} \Pi_a(A_i)$$

$$\Pi_a(\forall \text{hasCategory}.(\neg C_d \sqcap \leq x_d \text{ (level)})) = \frac{1 - x_d}{\sum_{i=1}^{n} \frac{1}{\Pi_a(A_i)}}$$

The penalty functions for atomic concepts A_d, $\Pi_c(A_d)$ and Π_a (A_d) depend merely from domain knowledge [4].

There are some codes in section 5.2.3. which calculate $\Pi_c(A_d)$ and Π_a (A_d) in general (using the path of A_d) using logarithm.

Chapter 4

ContAb Image Retrieval System: Analysis and Design

4.1. Knowledge Domain and Problem Analysis

Starting from this chapter, the term “image” or “images” will be narrowed and refer to photograph images which are produced by photo camera either digital or film. Also the term “film camera” is used rather than “analog camera” as an opposed to digital camera since the term “analog” in this case will be misleading.

If we talk about images (once again, photograph images), what comes in mind? One has been mentioned in the first paragraph of this chapter, the camera. Next question would be: where to save images? Before digital technology affects our life, the answer for this might be only film. But now there is digital film or more precisely digital image sensor. Both film and digital image sensor (like CCD, CMOS, etc) are the most common recording media for images. And the most important is: what does an image have in it? Of course it has colors, and it has a meaning. “A meaning” of an image here means what can one say about an image. One may say: “It is an image of a dog” while the other says “It is an image of a sitting dog on the beach”. There would be different interpretations of an image from two persons. But they agreed that there is a dog on the image. The dog on the image is a logical segment which needs human interpretation to describe it. Now we have four related entities of an image, camera, recording media, color and segment.

The picture below shows all entities and their relationship of the ContAb Image Retrieval System. The name "ContAb" has been chosen since it uses concept **Cont**raction and concept **Ab**duction for its "engine".

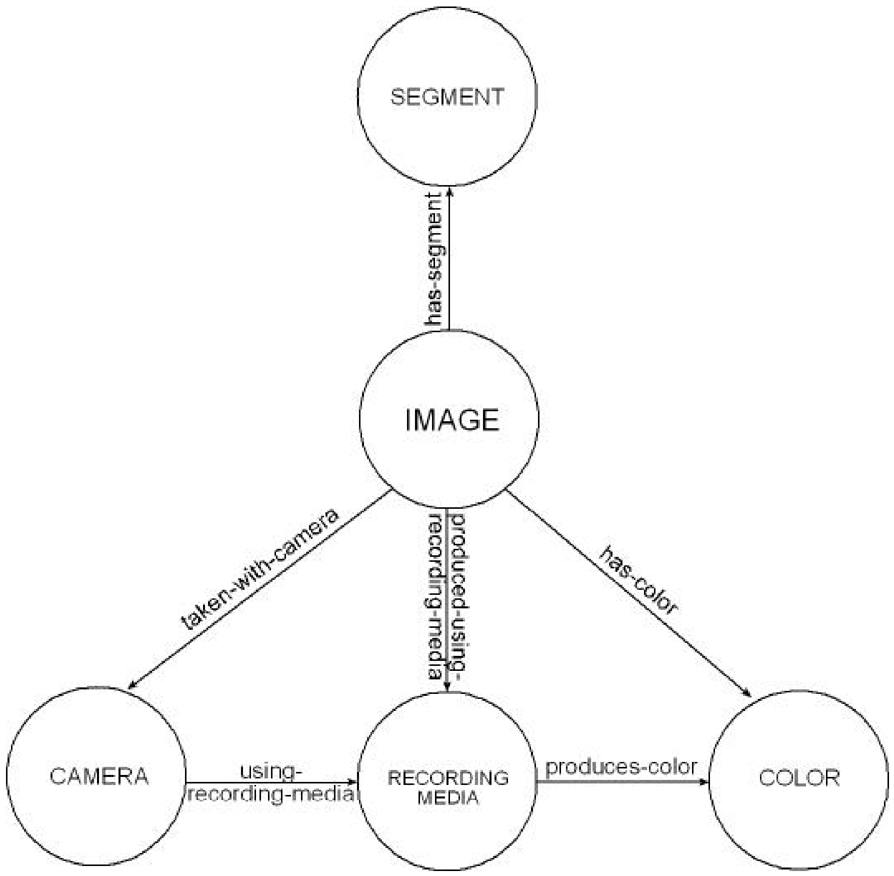


Figure 4.1. The main entities in ContAb Image Retrieval System.

The above picture shows the `IMAGE` entity and its relationship to other entities and also the relationship between `CAMERA` and `RECORDING-MEDIA` as well as relationship between `RECORDING-MEDIA` and `COLOR`.

As discussed earlier, CAMERA can be DIGITAL-CAMERA or FILM-CAMERA with the related RECORDING-MEDIA which can be DIGITAL-IMAGE-SENSOR or FILM. The color of an image is normally spread into two categories, black and white and color image. But if it comes to the general term of color in photography, black and white image refers to image with GRAYSCALE-COLOR and color image refers to image with all possible COLOR including GRAYSCALE-COLOR and NON-GRAYSCALE-COLOR.

The categorization of SEGMENT is a significant process in designing the system, since this is where the ontological categorization should be applied. Depends on the image collections, SEGMENT can be very specific, moderate or very general. For instance, image collections from a historical museum have a very specific SEGMENT, and image collections from stock photography (like Getty Image or Corbis) have a general SEGMENT. ContAb image retrieval system is designed more likely for the last one.

4.2. Knowledge Base Design

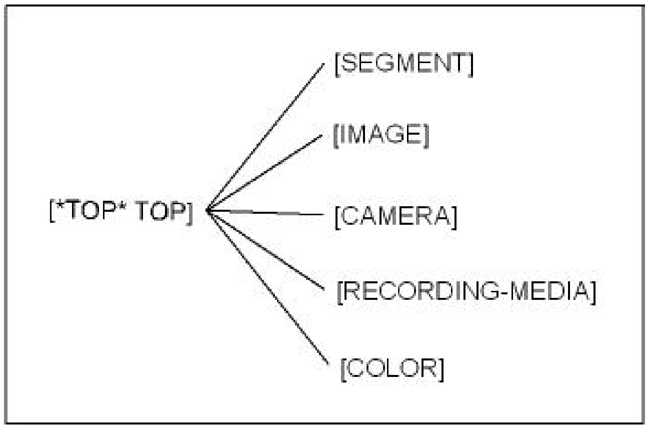
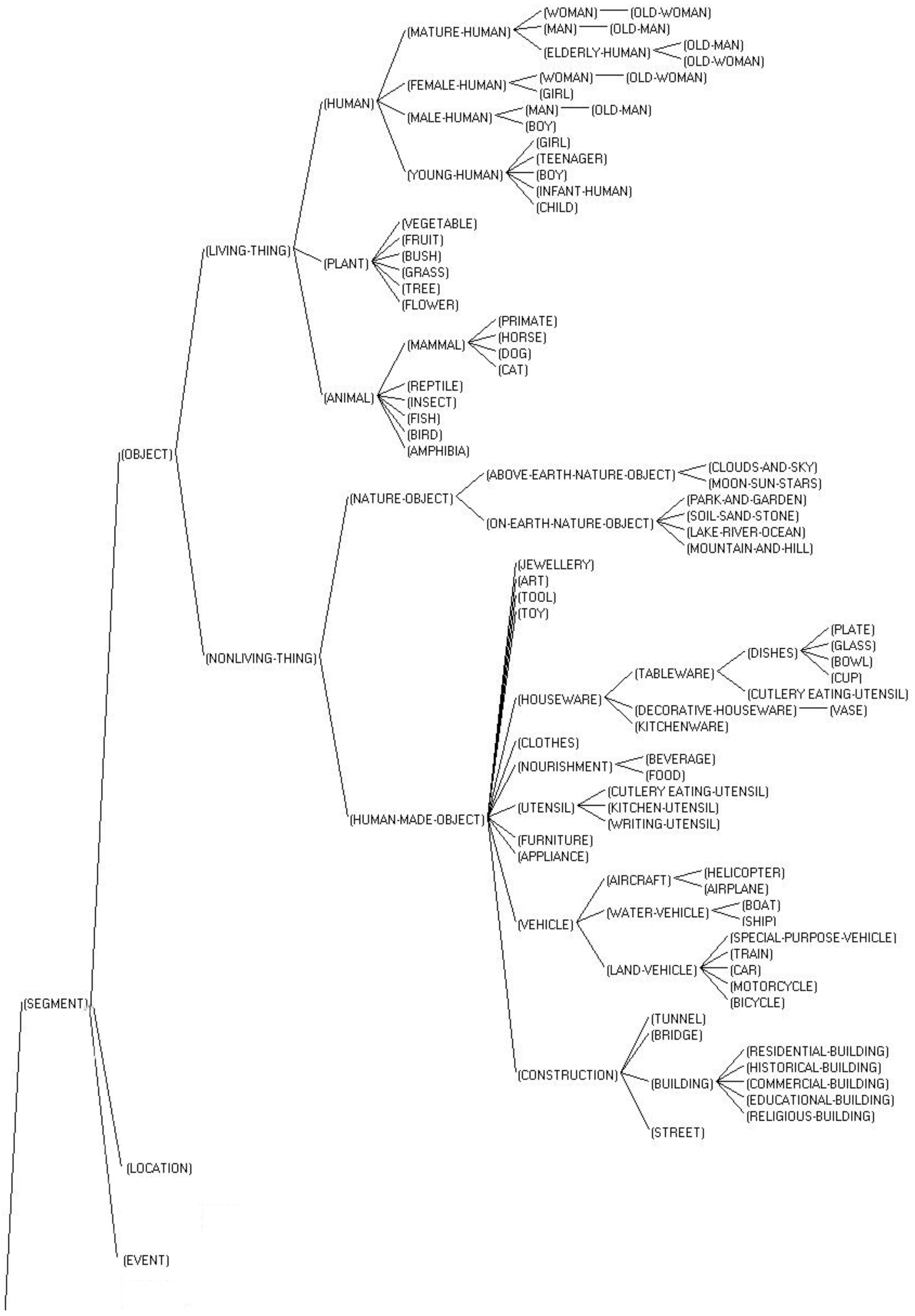


Figure 4.2. All main entities as upper levels in the taxonomy.

According to [7], the word ontology comes from the Greek ων = being and λόγος = word/speech. In philosophy, it is the study of being or existence as well as the basic categories thereof--trying to find out what entities and what types of entities exist. Ontology has strong implications for the conceptions of reality. In computer science, an ontology is the attempt to formulate an exhaustive and rigorous conceptual schema within a given domain, a typically hierarchical data structure containing all the relevant entities and their relationships and rules (theorems, regulations) within that domain.

The categorization of images in this Thesis has three general entities below the most general entity (root/top) which are `OBJECT`, `EVENT` and `LOCATION`. Each entity is divided to the more specific entities until it reaches its most specific entities. For example, the `OBJECT` entity has `LIVING-THING` and `NONLIVING-THING` as successors, the `NONLIVING-THING` has `HUMAN`, `ANIMAL` and `PLANT` as successors, and so on. For brevity, only `OBJECT` entity will be used for examples thtoughout this paper.


(SEGMENT)
(OBJECT)
(LIVING-THING)
(HUMAN)
(MATURE-HUMAN)
(WOMAN)
(OLD-WOMAN)
(MAN)
(OLD-MAN)
(ELDERLY-HUMAN)
(OLD-MAN)
(OLD-WOMAN)
(FEMALE-HUMAN)
(WOMAN)
(OLD-WOMAN)
(GIRL)
(MALE-HUMAN)
(MAN)
(OLD-MAN)
(BOY)
(YOUNG-HUMAN)
(GIRL)
(TEENAGER)
(BOY)
(INFANT-HUMAN)
(CHILD)
(PLANT)
(VEGETABLE)
(FRUIT)
(BUSH)
(GRASS)
(TREE)
(FLOWER)
(ANIMAL)
(MAMMAL)
(PRIMATE)
(HORSE)
(DOG)
(CAT)
(REPTILE)
(INSECT)
(FISH)
(BIRD)
(AMPHIBIA)
(NONLIVING-THING)
(NATURE-OBJECT)
(ABOVE-EARTH-NATURE-OBJECT)
(CLOUDS-AND-SKY)
(MOON-SUN-STARS)
(ON-EARTH-NATURE-OBJECT)
(PARK-AND-GARDEN)
(SOIL-SAND-STONE)
(LAKE-RIVER-OCEAN)
(MOUNTAIN-AND-HILL)
(HUMAN-MADE-OBJECT)
(JEWELLERY)
(ART)
(TOOL)
(TOY)
(HOUSEWARE)
(TABLEWARE)
(DISHES)
(PLATE)
(GLASS)
(BOWL)
(CUP)
(CUTLERY EATING-UTENSIL)
(DECORATIVE-HOUSEWARE)
(VASE)
(KITCHENWARE)
(CLOTHES)
(NOURISHMENT)
(BEVERAGE)
(FOOD)
(UTENSIL)
(CUTLERY EATING-UTENSIL)
(KITCHEN-UTENSIL)
(WRITING-UTENSIL)
(FURNITURE)
(APPLIANCE)
(VEHICLE)
(AIRCRAFT)
(HELICOPTER)
(AIRPLANE)
(WATER-VEHICLE)
(BOAT)
(SHIP)
(LAND-VEHICLE)
(SPECIAL-PURPOSE-VEHICLE)
(TRAIN)
(CAR)
(MOTORCYCLE)
(BICYCLE)
(CONSTRUCTION)
(TUNNEL)
(BRIDGE)
(BUILDING)
(RESIDENTIAL-BUILDING)
(HISTORICAL-BUILDING)
(COMMERCIAL-BUILDING)
(EDUCATIONAL-BUILDING)
(RELIGIOUS-BUILDING)
(STREET)
(LOCATION)
(EVENT)

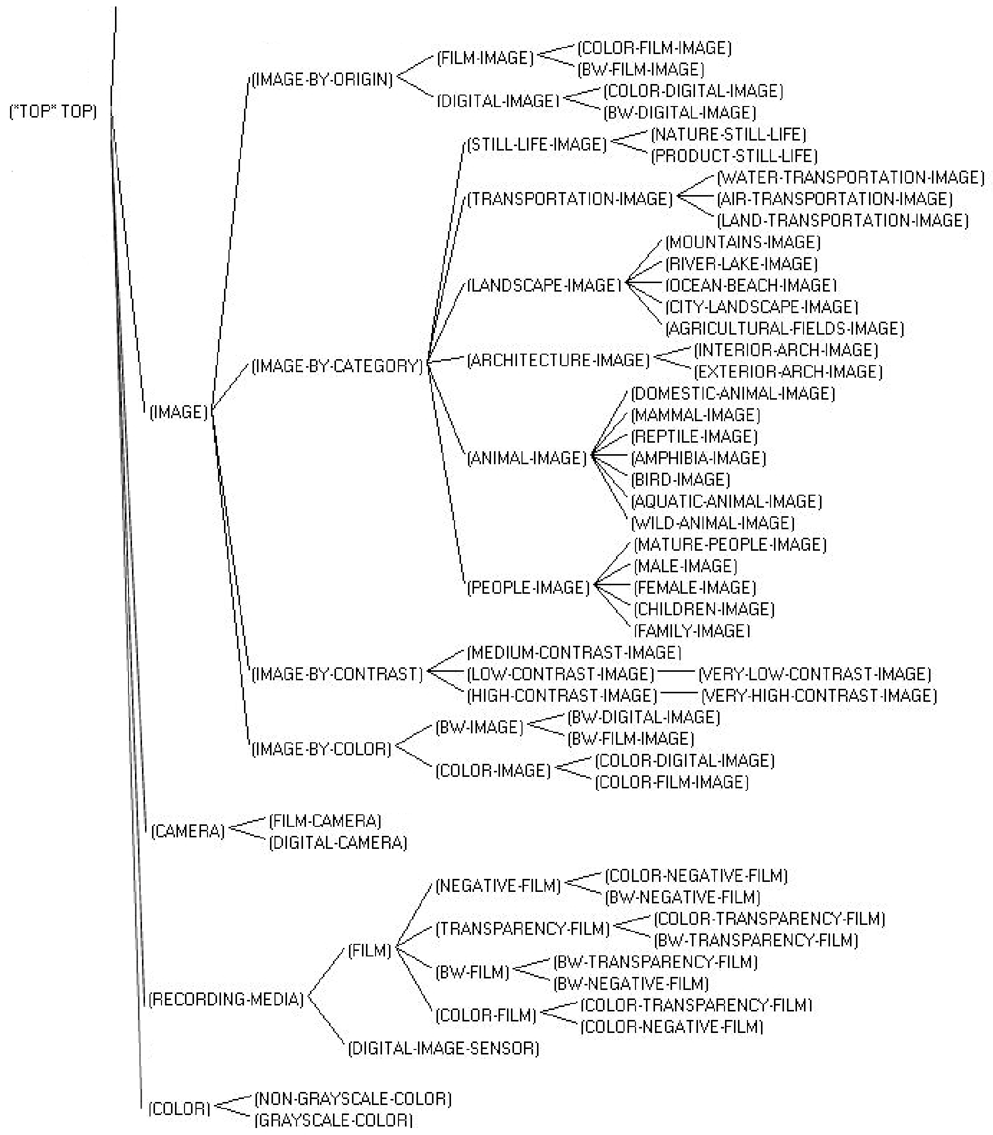


Figure 4.3. The ontology of ContAb Image Retrieval System.

4.3. System Design

As mentioned earlier, the implementation will be using Java as the programming language and RacerPro as the description logic reasoner. Derb database will also be used to save the annotated data before they are loaded in the ABox.

4.3.1. Overall Architecture

The main components architecture of ContAb Image Retrieval System is illustrated below:

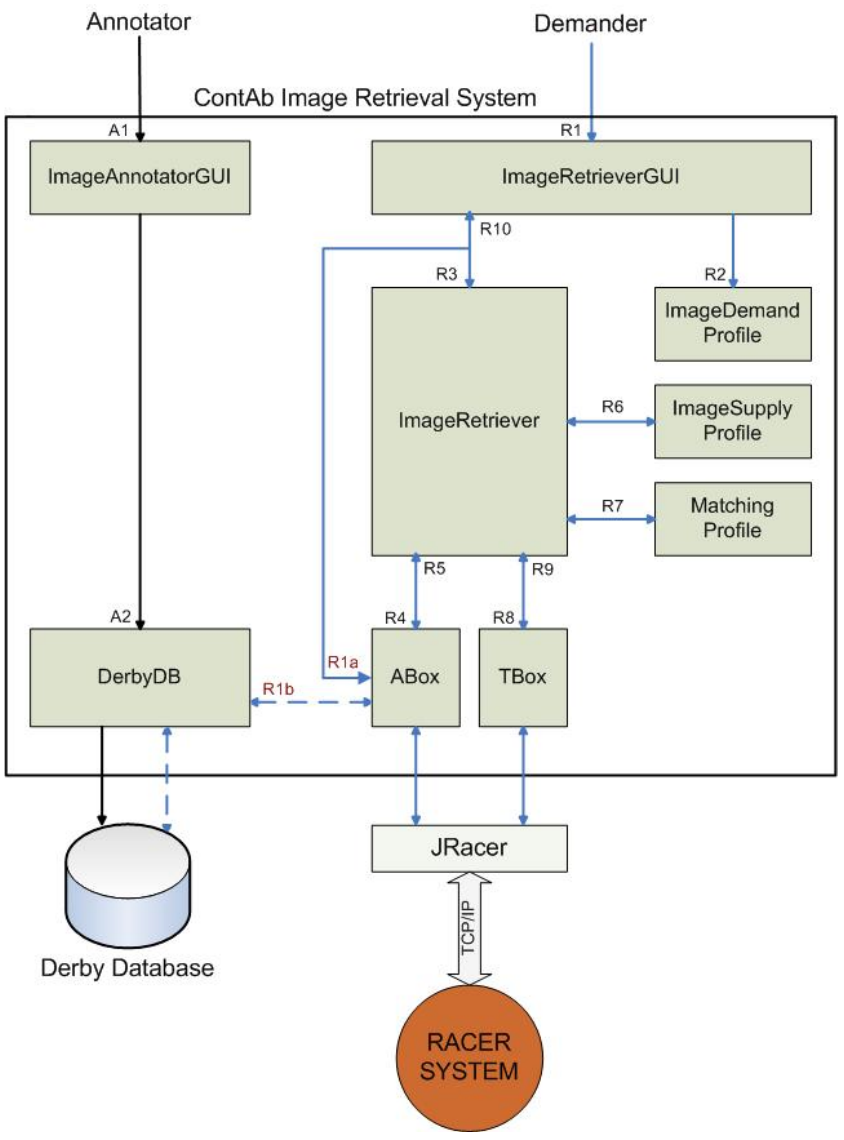


Figure 4.4. The Main Components Architecture of ContAb Image Retrieval System

In the above figure, there are two main processes shown:

- Annotation process shown by the black arrowed lines and numbers prefixed with "A" (A1 – A2).
- Retrieval process shown by the blue arrowed lines and numbers prefixed with "R" (R1 - R10).

The annotation process is quiet simple:

A1: The annotator puts the image data through the user interface and then submits it.

A2: The user interface class (ImageAnnotatorGUI) passes the data to DerbyDB class to be saved in the database. The DerbyDB class checks first whether the data exist, if yes, the data will be updated, otherwise the data will be inserted.

The summary of retrieval process is as following:

R1: The demander interacts with the user interface, puts some details about images he wants (and optional precision level) and then submits them.

R2: After the demander submitted the form, image demand profile is built.

R3: The ImageRetriever now handles the process.

R4: After demand profile is built, the ImageRetriever asks the ABox class to find images with / without the corresponding positive / negative keywords by querying the Racer System using JRacer as the interface.

R5: The query results are sent to the ImageRetriever.

R6: Image supply profiles are built by making an instance of ImageSupplyProfile class.

R7: Matching profiles are built by making an instance of MatchingProfile class. Each pair of matching profile consists of the image demand, one matching image supply and a corresponding penalty with initial value equal to zero.

R8: The ImageRetriever class will now perform the CalculatePenalty algorithm for every pair of matching profile. The TBox class is responsible for sending the query to the Racer System, starting from contraction phase for atomic concepts to the end of the algorithm.

R9: The answers from the Racer System are sent to the ImageRetriever class and the penalty calculation will be performed if it's necessary. After all matching profiles are evaluated against the algorithm; the results will be ranked based on penalty values and (optional) precision level defined by the demander.

R10: The ranked matching profiles are sent to the user interface to be shown to the demander.

4.3.2. Graphical User Interface Design

Graphical User Interface (GUI) is a critical part of a computer application. Galitz has mentioned some benefits of a good GUI design as following [30]:

1. Lower training costs, because the screen data is displayed in a more intuitive and self-explanatory manner.
2. Less user stress, because the interface helps rather than impedes users
3. Better user satisfaction

This section explains how the represented image profile will be implemented in Annotation-GUI and Retrieval-GUI.

The Annotation-GUI will have two mode options, simple mode and detail mode. The difference between them is, in simple mode all concept names can be selected straight forward, where in detail mode the origin of an image will be inferred from the camera used. This will be explained more clearly in demonstration (section 5.3.1.).

Furthermore, both Annotation- and Retrieval-GUI will follow these rules:

- The items will be placed in order from most simple and important to more complex and less important (except for keywords and segment annotation, since this needs a wider space than others).
- The use of drop down menu is only for item with more than 3 options.
- What is not needed will be made invisible.

4.3.3. UML Class Diagram

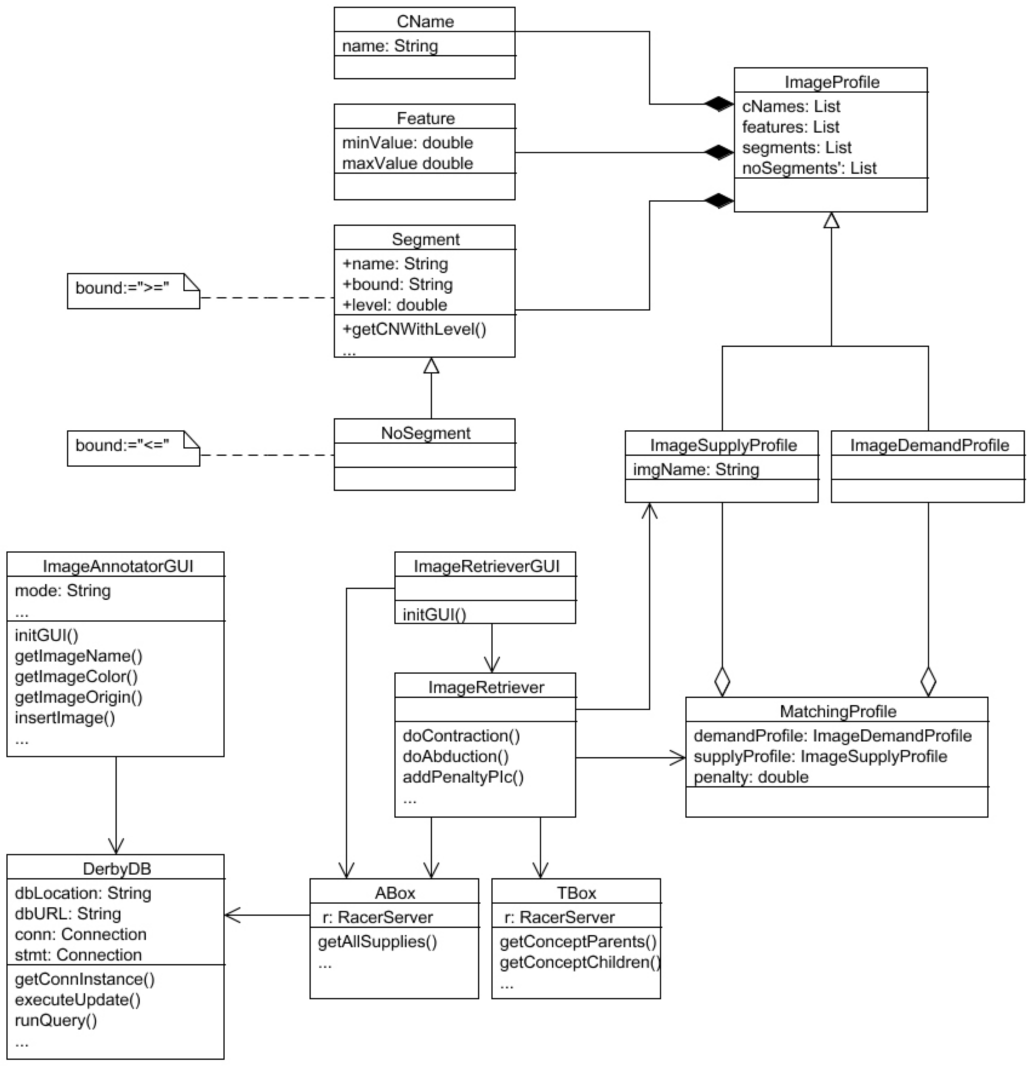


Figure 4.5. The class diagram.

Chapter 5

Implementation

5.1. Annotation Process

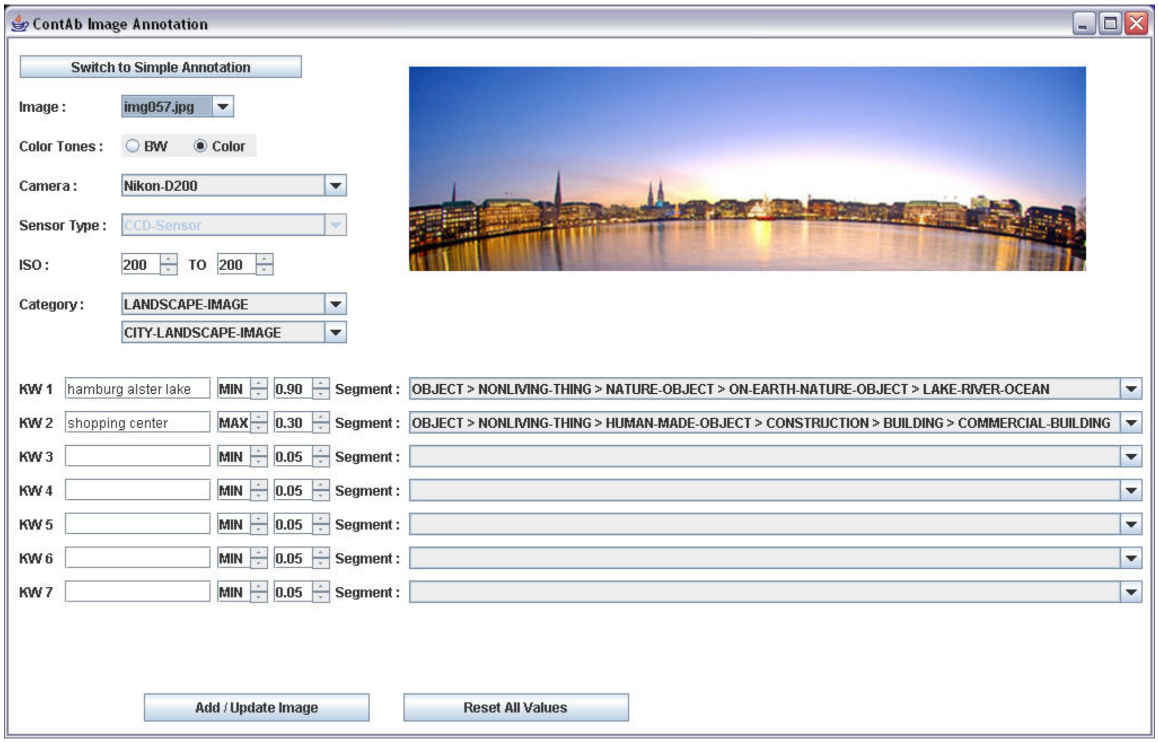
Since Annotation is not the main point of this Thesis, this part is implemented in an easy way, that is, all tasks are put in the ImageAnnotatorGUI class except the tasks for storing data in the database.

5.1.1. Annotation-GUI

The ImageAnnotationGUI class extends JFrame in javax.swing package.

```
public class ImageAnnotatorGUI extends javax.swing.JFrame {
    private JDesktopPane jDesktopPane1;
    private JLabel imageLabel;
    private JComboBox imageComboBox;
    private JLabel showImageLabel;
    private JLabel keyLabel1;
    private JTextField keyTextField1;
    private JButton modeButton;
    private JButton addImageButton;
    private JButton resetButton;
    private JSpinner isoLowSpinner;
    private JSpinner isoHiSpinner;
    …
    …

}
```


ContAb Image Annotation

Switch to Simple Annotation

Image : img057.jpg

Color Tones : BW [✓ Color]

Camera : Nikon-D200

Sensor Type : CCD-Sensor

ISO : 200 TO 200

Category : LANDSCAPE-IMAGE

CITY-LANDSCAPE-IMAGE

KW 1	hamburg alster lake	MIN	0.90	Segment :	OBJECT > NONLIVING-THING > NATURE-OBJECT > ON-EARTH-NATURE-OBJECT > LAKE-RIVER-OCEAN
KW 2	shopping center	MAX	0.30	Segment :	OBJECT > NONLIVING-THING > HUMAN-MADE-OBJECT > CONSTRUCTION > BUILDING > COMMERCIAL-BUILDING
KW 3		MIN	0.05	Segment :	
KW 4		MIN	0.05	Segment :	
KW 5		MIN	0.05	Segment :	
KW 6		MIN	0.05	Segment :	
KW 7		MIN	0.05	Segment :	

Add / Update Image

Reset All Values

Figure 5.1. ContAb Image Annotation Graphical User Interface.

First of all, the annotator must select an image to annotate. Since all images are found in one single directory, the image names will be read from that directory using `listNames()` method:

```
public static String[] listNames (String directory) {
    File dir = new File(directory);
    FilenameFilter filter = new FilenameFilter() {
        public boolean accept(File dir, String name) {
            return name.endsWith(".jpg");
        }
    };
    String[] f = dir.list(filter);
    return f;
}
```

The `listNames()` method uses `FilenameFilter` to read files with .jpg extension only since all images are stored in this format.

Since all individuals will be loaded to ABox in the beginning of retrieval process and it assumes that the annotation process happens before the retrieval process, the individuals will be taken from the database. Those individuals are of `camera` and `recording-media`. The DerbyDB class does the job with the methods `getCameras()` and `getRecordingMedia()`.

It assumes that the knowledge base is already loaded to TBox at the very beginning, so the system can query the TBox for image categories and segments. To get all categories, the ImageAnnotatorGUI class calls the getConceptChildren method in the TBox class with `"IMAGE-BY-CATEGORY"` as the argument since all categories are concept children of `"IMAGE-BY-CATEGORY"` concept.

```
Tbox.getConceptChildren("IMAGE-BY-CATEGORY");
```

Below is the getConceptChildren method:

```
public static String[] getConceptChildren(String parent) {
    String que, res;
    que = "(concept-children " + parent + ")";
    res = r.getRacerOutput(que);
    String[] child = res.split(" ", 0);
    return child;
}
```

where “r” is an instance of the RacerServer class. Some important parts of RacerServer class can be found in **Appendix C**.

The segments are displayed as a tree, for example:
`OBJECT > LIVING-THING > HUMAN.`
The easiest way to get this is through iteration from the most general concept to most specific concepts. The method in TBox which is responsible for this is the `getConceptDescendantsTree` method.

```
public static List getConceptDescendantsTree(String ancestor) {
    String[] tree;
    String branch;
    List t = new ArrayList();
    String[] child = getConceptChildren(ancestor);

    for ( int i = 0; i < child.length; i++ ) {
        if ( !child[i].contains("BOTTOM") ) {
            t.add(child[i]);

            String[] gChild = getConceptChildren(child[i]);
            for ( int j = 0; j < gChild.length; j++ ) {
                if ( !gChild[j].contains("BOTTOM") ) {
                    branch = child[i] + " > " + gChild[j];
                    t.add(branch);

// there are total 7 iterations
// to the deepest path of the segment.

}
```

The keywords fields and the corresponding level-bound, level-value and segment fields are limited to seven rows.

5.1.2. Data Collection and Storing

Once the annotator put all the data for an image and submitted them, the system checks whether the image and its data exist in the database. If they exist, the old data will be deleted first before inserting the new data, otherwise it just does inserting the data. This way is chosen for the sake of simplicity.

The `insertImage()` method which will be performed at the end of a single annotation process is as following:

```
private void insertImage() {
    String img = getImageName();
    String col = getImageColor();
    String orig = getImageOrigin();
    String cam = getCameraModel();
    String med = getRecordingMedia();
    String[] iso = getISOValue();
    String isoLow = iso[0];
    String isoHi = iso[1];
    String cat = getSelectedCategory();

    String sql = "INSERT INTO APP.IMAGES VALUES " +
                 ('"+img+"','"+cat+"','"+col+"','"+orig+"','"+
                 cam+"','"+med+"','"+isoLow+"','"+isoHi+
                 "','"+this.mode+"')";
    if ( DerbyDB.executeUpdate(conn, sql) > 0 ) {
        List v = getKeywordsProperties();

        for ( ListIterator i = v.listIterator(); i.hasNext(); ) {
            String kp = i.next().toString();
            String[] prop = kp.split(":");

            double level = Double.parseDouble(prop[2]);

            String sql2 = "INSERT INTO APP.KEYWORDS " +
               "(IMAGE_NAME,KEYWORD,LEVEL_BOUND,LEVEL,SEGMENT) "+
               "VALUES ('" + img + "','"+prop[0]+"','"+prop[1]+
               "',"+level+",'"+prop[3]+"')";

            DerbyDB.executeUpdate(conn, sql2);
        }
    }
}
```

As can be seen in the `insertImage()` method above, all the annotated data will be gathered by these getter methods:

- `getImageName()`, it returns the image name only, the extension will be omitted.
- `getImageColor()`, returns either `bw-image` or `color-image`.
- `getImageOrigin()`, returns either film-image or digital-image, this method is for simple mode annotation only.
- `getCameraModel()`, returns the selected camera, in detail mode only.
- `getRecordingMedia()`, returns the selected recording media, detail mode only.
- `getISOValue()`, returns ISO Value as string array in format { low-bound, high-bound }
- `getSelectedCategory()`, returns the selected category, if a subcategory selected, it returns only that subcategory.

```
private String getSelectedCategory() {
    String cat =
       catComboBox.getModel().getSelectedItem().toString();
    if ( !cat.equals("IMAGE-BY-CATEGORY") ) {
        String subCat =
           subCatComboBox.getModel().getSelectedItem().toString();
        if ( !subCat.equals("ANY-SUBCATEGORY") ) cat = subCat;
    }
    return cat;
}
```

- `getKeywordsProperties()` collects all positive / negative keywords and their corresponding level-bound, level-values and segments. It returns the properties as a list. This method will first check whether a keyword has a minimum length of 3 characters or not, if not, the keyword and its properties will be ignored. The minimum length of a keyword can be changed by giving the constant variable `MIN_KEY_LENGTH` another value.

```
private List getKeywordsProperties() {
    String key, bound, level, segment, prop;
    List keyProps = new ArrayList();

    if ( keyTextField1.getText().length() >= MIN_KEY_LENGTH ) {
        key = keyTextField1.getText();
        segment = getMostSpecificSegment(egmentComboBox1.
                      getModel().getSelectedItem().toString());
        level = levelSpinner1.getModel().getValue().toString();
        bound = boundSpinner1.getModel().getValue().toString();
        prop = key + ":" +  bound + ":" +  level + ":" +  segment;
        keyProps.add(prop);
    }
    if ( keyTextField2.getText().length() >= MIN_KEY_LENGTH ) {
    …
    }
    …
    // if ( keyTextField3 … up to keyTextField7

    return keyProps;
}
```

5.2. Retrieval Process

5.2.1. Retrieval-GUI

Same as the implementation of the annotation-GUI, the retrieval GUI also extends javax.swing.JFrame.

```
public class ImageRetrieverGUI extends javax.swing.JFrame {
    private JDesktopPane jDesktopPane1;
    private JLabel originLabel;
    private JRadioButton originRadioButton1;
    private JLabel colorLabel;
    …
}
```

5.2.2. Image Demand Profiling

The image demand profile is built directly in the `ImageRetrieverGUI` class as soon as the demander submits the image query.

```
private void findButtonActionPerformed(ActionEvent evt) {

    ImageDemandProfile d = buildDemandProfile();
    …
}
```

But before the demand profile is built, the following condition must be met:

1. If a keyword belongs to two segments which have a subsumption relation then the more general segment will be taken. For example, if the system found the keyword "pretty woman" is an instance of segment `FEMALE-HUMAN` and also `WOMAN`, then only the segment `FEMALE-HUMAN` will be taken. This is done by the method `findMostGeneralSegments()` in the class TBox.

```
private List findMostGeneralSegments(List s) {
    List discardSegments = new ArrayList();

    for ( ListIterator li = s.listIterator(); li.hasNext(); ) {
        String seg = (String) li.next();
        String[] desc = Tbox.getConceptDescendants(seg);

        for ( int i = 0; i < desc.length; i++ ) {
            discardSegments.add(desc[i]);
        }
    }
    s.removeAll(discardSegments);
    return s;
}
```

2. If a segment query subsumes another one then the subsumee (the one which is more specific) will be discarded. This may happen since the system assumes that the demander has no knowledge about how the system works. For example, consider the following scenario, a demander is searching for:
   “white dog” ⩾0.70 (level), and
   “pet” ⩾0.80 (level)
   In ABox we have: “white dog” as an instance of concept `DOG` and “pet” as an instance of concept `ANIMAL`.
   In TBox there is an axiom: `DOG` implies `ANIMAL` (`DOG` $\sqsubseteq$ `ANIMAL`).
   So, (`ANIMAL` ⩾0.80 (level)) subsumes (`DOG` ⩾0.70 (level)).
   Thus, the query “white dog” ⩾0.70 (level) will be discarded.
   The method `hasSubsumptionRelation()` and `isSubsumedBy()` in class TBox is used for querying the RacerPro in this case. It returns true if a conjunction of concept subsumes another one, otherwise false.

```
public static boolean hasSubsumptionRelation(String cn1, String
cn2) {

    boolean subsumption = false;

    if (isSubsumedBy(cn1,cn2) || isSubsumedBy(cn2, cn1) )
       subsumption = true;

    return subsumption;
}

public static boolean isSubsumedBy(String cn1, String cn2) {
    boolean isSubsumed = false;
    String que, res;
    que = "(concept-subsumes? " + cn2 + " " + cn1 + ")";
    res = r.getRacerOutput(que);

    if ( res.equals("T") ) isSubsumed = true;

    return isSubsumed;
}
```

Below is the `buildDemandProfile()` method which builds the image demand profile:

```
private ImageDemandProfile buildDemandProfile() {
  List posSegments = new ArrayList();
  List negSegments = new ArrayList();
  List keyProps = getKeywordsProperties();
  List posKeyList = new ArrayList();
  List negKeyList = new ArrayList();

  for (ListIterator k=keyProps.listIterator(); k.hasNext(); ) {
     String[] prop = (String[]) k.next();
     String key = prop[0];
     String bnd = prop[1];
     double lvl = Double.parseDouble(prop[2]);
     List seg = findMostGeneralSegments( findSegments(key) );

     if ( bnd.equals(">=") ) posKeyList.add(key);
     else negKeyList.add(key);

     for (ListIterator i = seg.listIterator(); i.hasNext(); ) {
        String sg = (String) i.next();
        Segment s = new Segment(sg, bnd, lvl);

        if ( bnd.equals(">=") ) posSegments.add(s);
        else negSegments.add(s);
     }
  }

  List demandSegments =fixDemandSegments(posSegments,negSegments);

  List cNames = buildCNames();
  List features = buildFeatures();
  List segments = getSegments(demandSegments);
  List noSegments = getNoSegments(demandSegments);

  ImageDemandProfile d = new ImageDemandProfile(cNames, features,
                                            segments, noSegments);

  d.posKeyList = posKeyList;
  d.negKeyList = negKeyList;

  return d;
}
```

5.2.3. Finding the Matching Supplies

A decision on how to retrieve the image supplies have to be made before. It depends on the amount of the collections and how dispersed they are. A dispersed collection is for example a collection which has images from dog to UFO (Unidentified Flying Object) and from beach to shoes. Thus, these scenarios can be applied:

- For small and dispersed collections, it's good to retrieve all available images and evaluate them one by one, since finding the exact matches in this case is considerably hard or the possible exact matches are too little.
- For big or moderate and concentrated collections, it is better to retrieve only the exact matches and rank the result from the most promising one to the less promising. An example of this kind of collections would be a collection from an architecture image database with hundreds or thousands of images.
- For other kind of image collections between first point and second point, it depends on what goal the system designer wants to achieve. When precision matters, the second approach should be taken into account, otherwise the first approach.

In the case of ContAb Image Retrieval System, the first approach will be taken since there are less than one hundred images in its collection and they are dispersed.

The method `getAllSupplies()` in class ABox returns a list of all image supplies as ImageSupplyProfile instances.

```
public static List getAllSupplies() {

  List supplies = new ArrayList();

  String[] sup = r.getRacerOutput("(concept-instances image)").split("
               ");
  for ( int i = 0; i < 10; i++ ) {
     String img = sup[i];
     List cn = findCNames(img);
     List f = DerbyDB.findFeatures(img);
     List s = DerbyDB.findSegments(img);
     List ns = DerbyDB.findNoSegments(img);

     ImageSupplyProfile is = new ImageSupplyProfile(img, cn, f, s, ns);
        supplies.add(is);
  }

  return supplies;
}
```

For the sake of simplicity, `Features`, `Segments` and `NoSegments` are taken from the database as can bee seen on the code snippets above.

5.2.4. Do Contraction

The method `doContraction()` in class `ImageRetriever` evaluates the image demand profile against every image supply profile in matching profile. Just like the given algorithm, it starts from evaluating every `CNames` in demand profile and ends with `NoSegments`.

```
private static void doContraction(MatchingProfile mp) {

  ImageDemandProfile d = mp.demandProfile;
  ImageSupplyProfile s = mp.supplyProfile;

  // CNames(Pd)
  for ( ListIterator i1 = d.cNames.listIterator(); i1.hasNext(); ) {
    String dName = (String) i1.next();

    for ( ListIterator i11 = s.cNames.listIterator(); i11.hasNext(); ) {
      String sName = (String) i11.next();
      if ( TBox.isDisjoint(dName, sName) ) {
        d.cNames.remove(dName);
        mp.penalty += addPenaltyPIc(dName);
      }
    }
  }

  // Features(Pd)
  for ( ListIterator i2 = d.features.listIterator(); i2.hasNext(); ) {
    Feature dFeature = (Feature) i2.next();
    String featName = dFeature.name;
    double dFeatMin = dFeature.minValue;
    double dFeatMax = dFeature.maxValue;
    for (ListIterator i21 = s.features.listIterator(); i21.hasNext(); ) {
      Feature sFeature = (Feature) i21.next();
      double sFeatMin = sFeature.minValue;
      double sFeatMax = sFeature.maxValue;

      if ( sFeature.name.equalsIgnoreCase(featName) ) {
        String pdF = "(and (>= " + featName + " " + dFeatMin + ") (<= " +
                     featName + " " + dFeatMax + "))";
        String psF = "(and (>= " + featName + " " + sFeatMin + ") (<= " +
                     featName + " " + sFeatMax + "))";
        if ( !TBox.isSatisfiable(pdF, psF) ) {
          d.features.remove(dFeature);
          mp.penalty += addPenaltyPIcf(dFeature, sFeature);
        }
      }
    }
  }

  // Segments(Pd)
  for ( ListIterator i3 = d.segments.listIterator(); i3.hasNext(); ) {
    Segment dSeg = (Segment) i3.next();
    …
    …
  }

  // NoSegments(Pd)
  for ( ListIterator i4 = d.noSegments.listIterator(); i4.hasNext(); ) {
    NoSegment dNoSeg = (NoSegment) i4.next();
    …
    …
  }
}
```

The code snippet below:

```
  for ( ListIterator i1 = d.cNames.listIterator(); i1.hasNext(); ) {
    String dName = (String) i1.next();

    for ( ListIterator i11 = s.cNames.listIterator(); i11.hasNext(); ) {
      String sName = (String) i11.next();
      if ( TBox.isDisjoint(dName, sName) ) {
        d.cNames.remove(dName);
        mp.penalty += addPenaltyPIc(dName);
      }
    }
  }
```

is the implementation this algorithm part:

```
foreach $A_d \in Names(P_d)$ do
  if there exists $A_s \in Names(P_s)$
    such that $\mathcal{H} \models A_d \sqsubseteq \neg A_s$
  then remove $A_d$ from $P_d$
      penalty := penalty + $\Pi_c$ ( $A_d$ )
```

And these codes:

```
for ( ListIterator i2 = d.features.listIterator(); i2.hasNext(); ) {
  Feature dFeature = (Feature) i2.next();
  String featName = dFeature.name;
  double dFeatMin = dFeature.minValue;
  double dFeatMax = dFeature.maxValue;

  for (ListIterator i21 = s.features.listIterator(); i21.hasNext(); ) {
    Feature sFeature = (Feature) i21.next();
    double sFeatMin = sFeature.minValue;
    double sFeatMax = sFeature.maxValue;

    if ( sFeature.name.equalsIgnoreCase(featName) ) {
      String pdF = "(and (>= " + featName + " " + dFeatMin + ") (<= " +
                   featName + " " + dFeatMax + "))";
      String psF = "(and (>= " + featName + " " + sFeatMin + ") (<= " +
                   featName + " " + sFeatMax + "))";
      if ( !TBox.isSatisfiable(pdF, psF) ) {
        d.features.remove(dFeature);
        mp.penalty += addPenaltyPIcf(dFeature, sFeature);
      }
    }
  }
}
```

are the implementation of the following algorithm part:

```
foreach $p_d(f) \in$ Features( $P_d$ ) do
  if there exists $p_s(f) \in$ Features( $P_s$ )
    such that $\exists x.\ p_d(x) \wedge p_s(x)$ is unsatisfiable in the domain associated to $f$
  then remove $p_d(f)$ from $P_d$
      penalty := penalty + $\Pi_{cf}$ ($p_d(f)$, $p_s(f)$)
```

5.2.5. Do Abduction

The method `doAbduction()` also implements the abduction algorithm in the same order as in the algorithm, first it evaluates the concept names:

```
private static void doAbduction(MatchingProfile mp) {

 ImageDemandProfile d = mp.demandProfile;
 ImageSupplyProfile s = mp.supplyProfile;

 // CNames(Pd)
 for (ListIterator i1 = d.cNames.listIterator();i1.hasNext(); ) {
   String dName = (String) i1.next();
   boolean abducted = true;

   for (ListIterator i11 = s.cNames.listIterator();
                                                  i11.hasNext();){
     String sName = (String) i11.next();
     if ( TBox.isSubsumedBy(sName, dName) ) abducted = false;
   }
   if ( abducted ) {
     s.cNames.add(dName);
     mp.penalty += addPenaltyPIa(dName);
   }
 }

 …
 …
```

The codes above are the implementation of this algorithm part:

```
foreach $A_d \in Names(P_d)$ do
  if there does not exist $A_s \in Names(P_s)$ such that $\mathcal{H} \models A_s \sqsubseteq A_d$
  then add $A_d$ to $P_s$
        penalty := penalty + $\Pi_a(A_d)$
```

Evaluation of `Features` in abduction phase:

```
for (ListIterator i2 = d.features.listIterator(); i2.hasNext();) {
  Feature dFeature = (Feature) i2.next();
  String featName = dFeature.name;
  double dFeatMin = dFeature.minValue;
  double dFeatMax = dFeature.maxValue;
  boolean featureExists = false;

´ for (ListIterator i21 = s.features.listIterator();
                                                  i21.hasNext();){
    Feature sFeature = (Feature) i21.next();
    double sFeatMin = sFeature.minValue;
    double sFeatMax = sFeature.maxValue;
    if ( sFeature.name.equalsIgnoreCase(featName) ) {
      featureExists = true;
      String pdF = "(and (>= "+ featName +" " +dFeatMin+") (<= "+
                                 featName + " " + dFeatMax + "))";
      String psF = "(and (>= "+featName+ " " +sFeatMin+ ") (<= " +
                                 featName + " " + sFeatMax + "))";
      if ( !TBox.isSubsumedBy(psF, pdF) ) {
        s.features.remove(sFeature);
        s.features.add(dFeature);
        mp.penalty += addPenaltyPIaf(dFeature, sFeature);
      }
    }
  }
  if (!featureExists) mp.penalty += addPenaltyPIaf(dFeature, "T");
}
```

From the algorithm part:

```
    **foreach** $p_d(f) \in Features( P_d )$ **do**
    **if** there exists $p_s(f) \in Features( P_s ) \ . p_d(x) \wedge p_s(x)$
    **then if** $\forall x.p_s(x) \Rightarrow p_d(x)$ is false in the domain associated to $f$
        **then** add $p_d(f)$ to $P_s$
            penalty := penalty + $\Pi_{af} (p_d(f), p_s(f))$
    **else** add $p_d(f)$ to $P_s$
        penalty := penalty + $\Pi_{af} (p_d(f), \top(f))$
```

5.2.6. Penalty Calculations and Ranking

Only three implementations of penalty functions will be explained here, $\Pi_c (A_d)$, $\Pi_a (A_d)$ and $\Pi_{cf} (p_d(f), p_s(f))$ since

- $\Pi_{cl} (x_d, x_s)$ and $\Pi_{al} (x_d, x_s)$ are clear, and
- $\Pi_a (\exists \text{has-segment}.(C_d \sqcap \geqslant x_d \text{(level)}))$ and
  $\Pi_a (\forall \text{has-segment}.(\neg C_d \sqcup \leqslant x_d \text{(level)}))$ are based on $\Pi_a (A_d)$,

The implementation of penalty function $\Pi_c (A_d)$ is as following:

```
private static double addPenaltyPIc(String dName, String sName) {
  double nPath = (double) TBox.getDeepestPathInTaxonomy(dName);
  double posAd = (double) TBox.getPositionInHierarchy(dName);
  double posDis = (double) TBox.getDisjointPositionInHierarchy(dName,
                                                               sName);
  double logAd = logBaseOf(nPath, posAd);
  double logDis = logBaseOf(nPath, posDis);

  double pen = logAd - (logAd*logDis) + Math.pow(logDis, 2);
  return pen;
}
```

There are three methods from TBox class involved:

- `getDeepestPathInTaxonomy()` gets the deepest path in `SEGMENT` or `IMAGE` taxonomy. It returns a value of type double which is used as the base for logarithm in the penalty calculation.
- `getPositionInHierarchy()` gets the position / path of a concept in `SEGMENT` or `IMAGE` taxonomy.
- `getDisjointPositionInHierarchy()` gets the position of the disjoint axioms in the hierarchy.

The implementation of penalty function $\Pi_a\ (A_d)$ is as following:

```
private static double addPenaltyPIa(String dName, String sName) {
    double nPath = (double) TBox.getDeepestPathInTaxonomy(dName);
    double posAd = (double) TBox.getPositionInHierarchy(dName);
    double posAs = (double) TBox.getPositionInHierarchy(sName);

    double logAd = logBaseOf(nPath, posAd);
    double logAs = logBaseOf(nPath, posAs);

    double pen = logAd - logAs;
    return pen;
}
```

The penalty function $\Pi_a\ (A_d)$ above is simpler than its counterpart in contraction phase, $\Pi_c\ (A_d)$. It only calculates the difference between the two paths, A_d and A_s.

The `logBaseOf(double b, double n)` method does the calculation of logarithm base on b of n.

```
public static double logBaseOf(double base, double n) {
    return Math.log(n)/Math.log(base);
}
```

The implementation of penalty function

$$\Pi_{cf}\ (p_d(f), p_s(f)) = \frac{|G|}{|I_d \cup I_s \cup \mathrm{G}|}$$

is as following:

```
private static double addPenaltyPIcf(Feature pdf, Feature psf) {
    double pdfMin = pdf.minValue;
    double pdfMax = pdf.maxValue;
    double psfMin = psf.minValue;
    double psfMax = psf.maxValue;
    double dInterval = pdfMax - pdfMin;
    double sInterval = psfMax - psfMin;
    double gap = 0;

    if ( pdfMax < psfMin ) gap = psfMin - pdfMax;
    if ( psfMax < pdfMin ) gap = pdfMin - psfMax;

    double pen = gap / (dInterval + sInterval + gap);
    return pen;
}
```

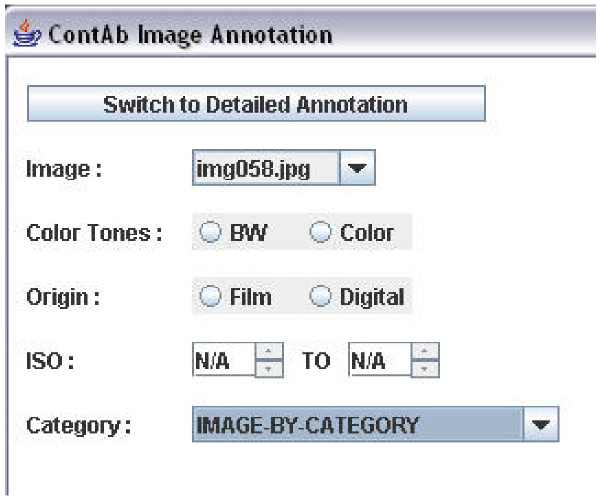
where gap = G, dInterval = Id, and sInterval = Is.

After all matching profiles are evaluated and penalty functions are calculated, the method `doRanking()` will perform the sorting task. It compares the previous penalty value of a matching profile with the next value; if the previous value is greater then it swaps the position with the next value and so on. At the end, the matching profiles are sorted from the smaller penalty value to the bigger. With these results and the precision level given by the demander, it is easy to adjust how many images the system should provide to the demander. Higher precision level means lower number of results, lower precision level means higher number of results.

5.3. Demonstrations

5.3.1. Image Annotation

There are two annotation modes, simple and detail annotation. The picture below shows a part of image annotation in simple mode.



ContAb Image Annotation

Switch to Detailed Annotation

Image : img058.jpg

Color Tones : ○ BW ○ Color

Origin : ○ Film ○ Digital

ISO : N/A TO N/A

Category : IMAGE-BY-CATEGORY

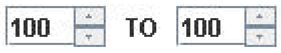
Figure 5.2. Image annotation in simple mode.

In simple mode, the annotation process is straight forward. There are only direct (atomic) concepts for TBox:

1. `IMAGE-BY-COLOR`: `BW-IMAGE` or `COLOR-IMAGE`
2. `IMAGE-BY-ORIGIN`: `FILM-IMAGE` or `DIGITAL-IMAGE`
3. `IMAGE-BY-CATEGORY`: category or its subcategory

It may seem that an overlapping occurs between categories / subcategories (subset of `IMAGE-BY-CATEGORY`) and `SEGMENT`. In fact, they do have the same idea but play different roles, `SEGMENT` must have all "entities" of images in the collections while `IMAGE-BY-CATEGORY` must not. In other words, `SEGMENT` gives the user specificity and `IMAGE-BY-CATEGORY` gives the user generality.

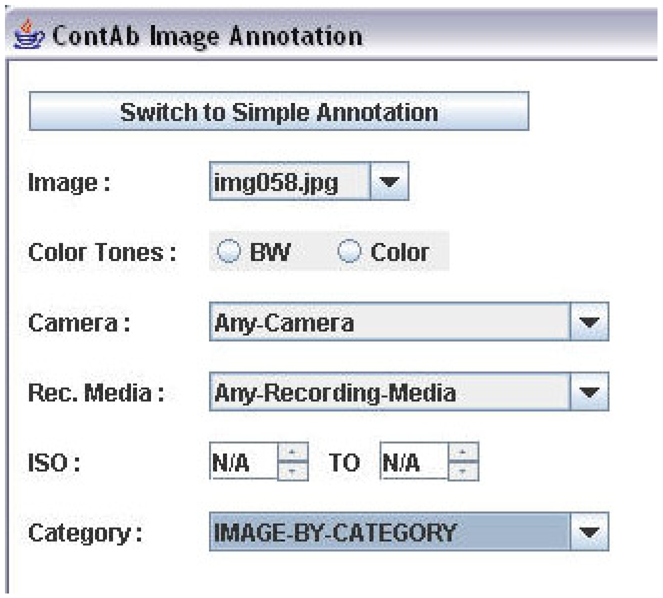
The ISO feature can be annotated in 5 ways:

1. Single value, for example: 

100 TO 100

2. A range of values, for example: 200 TO 400
3. A range of values without lower bound, for example: N/A TO 200
   In this case, a lowest possible value will be given automatically as a lower bound, which is 25 (lowest ISO).
4. A range of values without upper bound, for example: 800 TO N/A
   In this case, a highest possible value will be given automatically as a higher bound, which is 3200 (highest ISO).
5. No values: N/A TO N/A
   In this case, no values will be given and the system assumes this as missing information.

In detail mode, the annotator has the possibility to add some metadata like camera model and film used.


ContAb Image Annotation

Switch to Simple Annotation

Image : img058.jpg

Color Tones : BW Color

Camera : Any-Camera

Rec. Media : Any-Recording-Media

ISO : N/A TO N/A

Category : IMAGE-BY-CATEGORY

Figure 5.3. Image annotation in detail mode.

In detail annotation, the system will know the origin of the image through the reasoning service (in this case RacerPro). Here is a brief explanation of how it works:

In TBox:

DL notation: `digital-image` = `image` ∧ ∀`taken-with-camera.digital-camera`

Racer syntax: (equivalent digital-image (and image (all taken-with-camera digital-camera)))

In natural language: "All images taken with digital camera are digital images."

In Abox:

Racer syntax: (instance any-digital-camera digital-camera)

Racer syntax: (related img001 camera-x has-been-taken-with-camera)

It means: "img001 has been taken with camera x which is a digital camera."

Therefore the system knows that img001 is a `DIGITAL-IMAGE`.

Next step in annotation process is *keywording*, choosing the right segment for the keyword(s) and weighting the keyword for the selected segment. These are some examples:

`IMG021`

Keywords	Level	Segment
fish sticks	MIN 0.90	`FOOD`
white plate	MAX 0.20	`PLATE`

`IMG026`

Keywords	Level	Segment
black cat	MIN 0.65	`CAT`
gold fish	MIN 0.65	`FISH`

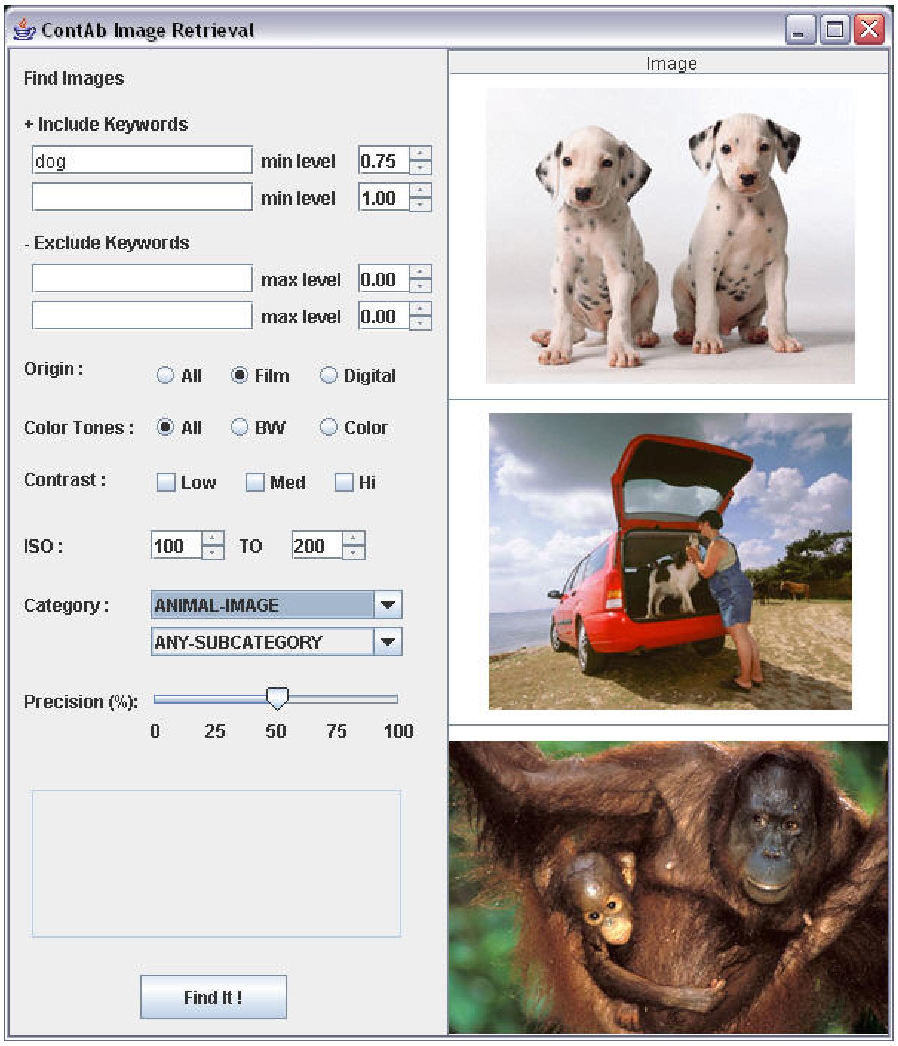
Keywords	Level	Segment
agricultural landscape	MIN 0.70	`ON-EARTH-NATURE-OBJECT`

IMG033	sunflower field	MIN 0.50	FLOWER
	hanging clouds	MIN 0.45	CLOUDS-AND-SKY

Figure 5.4. Some examples of annotated images.

5.3.2. Image Retrieval

This section provides a simple example of image retrieval process: a demander looks for dog images.


ContAb Image Retrieval

Find Images

+ Include Keywords

dog min level 0.75

min level 1.00

- Exclude Keywords

max level 0.00

max level 0.00

Origin : All [✓ Film] Digital

Color Tones : [✓ All] BW Color

Contrast : Low Med Hi

ISO : 100 TO 200

Category : ANIMAL-IMAGE

ANY-SUBCATEGORY

Precision (%): 0 25 50 75 100

Find It !

Image

Figure 5.5. ContAb Image Retrieval Graphical User Interface.

As can be seen in the Figure 5.5., the demander has a demand profile as following: $\exists\texttt{has-segment}.(\texttt{DOG} \sqcap \geqslant_{0.75} (\texttt{level})) \sqcap \text{film-image} \sqcap (\geqslant_{100}(\texttt{iso}) \sqcap \leqslant_{200} (\texttt{iso}))$.
The results indicate that there are only 2 exact matching for query DOG. The third result is shown as "most promising result", since it still has an ANIMAL segment. It shows a good performance of contraction and abduction algorithm, in particular when it is combined with the precision option.

Chapter 6
Conclusions & Future Work

The practical evaluation of contraction and abduction algorithm has shown that contraction gives a higher precision rate since conflicting information has to be given up and abduction gives the demander more results by adding the missing information to the domain knowledge.

This system gives the user (demander) a transparent idea about how the results are ranked. Also with this precision level, user satisfaction is self-adjustable but, like user frustration, it depends on the image collections and the quality of the annotation. As long as there is a human involved in the annotation process, it will always be subjective and it can be only minimized.

For future work, using thesaurus and a more expressive description language for the system would be a great challenge since these offer a great improvement.

References

[1] F. Baader and W. Nutt. Basic Description Logic. In *The Description Logic Handbook*, Chapter 2, Cambridge University Press, 2003.

[2] D. Nardi, R. J. Brachman. “An Introduction to Description Logics”. In *The Description Logic Handbook*, Cambridge University Press, 2003, pages 5-20.

[3] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. Concept Abduction and Contraction in Description Logics. In *Proc. of DL 2003*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-81/, 2003.

[4] A. Cali, D. Calvanese, S. Colucci, T. Di Noia and F. M. Donini. “A Description Logic Based Approach for Matching User Profiles”. In *Proc. of the 2004 Int. Workshop on Description Logics (DL 2004)*, 2004.

[5] J. P. Eakins and M. E. Graham. “Content-based image retrieval: A report to the JISC Technology Applications Programme”, January 1999. http://www.unn.ac.uk/iidr/report.html

[6] I. Horrocks and P. F. Patel-Schneider. “DL Systems Comparison”. http://www.ceur-ws.org/Vol-11/

[7] Wikipedia, http://en.wikipedia.org/

[8] Description Logic Systems, http://www.ida.liu.se/labs/iislab/people/patla/DL/systems.html

[9] Racer System, http://www.racer-systems.com/

[10] V.N. Gudivada and V.V. Raghavan. “Content-Based Image Retrieval Systems”, IEEE Computer Society (Vol.28 No.9) in Eakins and Graham [5], section 2.3.

[11] C. Jörgensen. “Image Retrieval, Theory and Research”. The Scarecrow Press, Inc, 2003, Chapter 1 – 4.

[12] S. Sclaroff, M. L. Cascia, S. Sethi and L. Taycher. “Unifying Textual and Visual Cues for Content-Based Image Retrieval on the World Wide Web.” Computer Vision and Image Understanding, 75(1-2):86-98, 1999.

[13] C. Shirky. “Ontology is Overrated: Categories, Links, and Tags”, http://www.shirky.com/writings/ontology_overrated.html

[14] R. Datta, J. Li, and J.Z. Wang. “Content-Based Image Retrieval - Approaches and Trends in the New Age“, http://www-db.stanford.edu/~wangz/project/imsearch/ review/ACM05/.

[15] H. Chuge. “Semantic-based Web Image Retrieval”, http://www2003.org/cdrom/papers/poster/p172/p172-zhuge/p172-zhuge.htm

[16] K. Mahesh. “Text Retrieval Quality: A Primer”, Oracle Corporation, http://www.oracle.com/technology/products/text/htdocs/imt_quality.htm?_template =/ocom/ocom_item_templates/print

[17] H. Evans. “Practical Picture Research: A Guide to Current Practice, Procedure, Techniques and Resources”, Blueprint, London, 1996.

[18] S.F. Chang, J.R. Smith, M. Beigi and A. Benitez. “Visual Information Retrieval from Large Distributed Online Repositories.” Communications of the ACM 40, no.12 (1997), in [11].

[19] E. Hyvönen, A.Styrman, S. Saarela. “Ontology-Based Image Retrieval.“ http://www.cs.helsinki.fi/u/eahyvone/publications/yomuseum.pdf

[20] D.Fensel (ed.). “The Semantic Web and Its Languages.” IEEE Intelligence Systems, Nov / Dec 2000, in [19].

[21] E. Hyvönen, K. Viljanen and A.Hätinen. “Yellow Pages on the Semantic Web.“ Number 2002-03, in [19].

[22] C. Fellbaum, (ed.). “WordNet. An Electronic Lexical Database.” The MIT Press, Cambridge, Massachusetts, 2001.

[23] M. Inoue. “On the Need for Annotation-Based Image Retrieval.” http://research.nii.ac.jp/~m-inoue/paper/inoue04irix.pdf

[24] D. Brahmi and D. Ziou. “Improving CBIR Systems by Integrating Semantic Features.” In *Proc. RIAO*, pages 291-305, Vaucluse, France, April 2004.

[25] Measuring Search Effectiveness. http://www.hsl.creighton.edu/hsl/Searching/Recall-Precision.html

[26] T. Peterson. Introduction to the Art and Architecture Thesaurus, 1994. http://shiva.pub.getty.edu.

[27] J. van den Berg. Subject Retrieval in Pictorial Information Systems. In *Proceedings of the 18th international congress of historical sciences, Montreal, Canada*, pages 21-29, 1995. http://www.iconclass.nl/texts/hixtory05.html.

[28] WordNet. *A Lexical Database for English Language*, Princeton University, Princeton, New Jersey. http://wordnet.princeton.edu/

[29] Y. Rui, T. S. Huang and S. F. Chang. "Image Retrieval: Current Techniques, Promising Directions, and Open Issues." *Journal of Visual Communication and Image Representation* 10 no.4 (1999): 39-62.

[30] W.O. Galitz. "The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques." Second Edition. Wiley Computer Publishing, 2002.

Appendix A

```
Algorithm CalculatePenalty
  Input demand profile $P_d$, supply profile $P_s$, concept Hierarchy $\mathcal{H}$
  Output real value penalty $\geqslant 0$
  penalty := 0;
  // Contraction
  foreach $A_d \in Names(P_d)$ do
    if there exists $A_s \in Names(P_s)$
      such that $\mathcal{H} \models A_d \sqsubseteq \neg A_s$
    then remove $A_d$ from $P_d$
          penalty := penalty + $\Pi_c(A_d)$
  foreach $p_d(f) \in Features(P_d)$ do
    if there exists $p_s(f) \in Features(P_s)$
      such that $\exists x.\ p_d(x) \wedge p_s(x)$ is unsatisfiable in the domain associated to $f$
    then remove $p_d(f)$ from $P_d$
          penalty := penalty + $\Pi_{cf}(p_d(f), p_s(f))$
  foreach $\exists$hasInterest.($C_d \sqcap \geqslant x_d$ (level)) $\in Interests(P_d)$ do
    foreach $\forall$hasInterest.($\neg C_s \sqcup \leqslant x_s$ (level)) $\in NoInterests(P_s)$ do
      if $\mathcal{H} \models C_d \sqsubseteq C_s$ and $x_d \geqslant x_s$
      then replace $\exists$hasInterest.($C_d \sqcap \geqslant x_d$ (level)) in $P_d$
              with $\exists$hasInterest.($C_d \sqcap \geqslant x_s$ (level))
            penalty := penalty + $\Pi_{cl}(x_d, x_s)$
  foreach $\forall$hasInterest.($\neg C_d \sqcup \leqslant x_d$ (level)) $\in NoInterests(P_d)$ do
    foreach $\exists$hasInterest.($C_s - \geqslant x_s$ (level)) $\in Interests(P_s)$ do
      if $\mathcal{H} \models C_s \sqsubseteq C_d$ and $x_d \leqslant x_s$
      then replace $\forall$hasInterest.($\neg C_d \sqcup \leqslant x_d$ (level)) in $P_d$
              with $\forall$hasInterest.($\neg C_d \sqcup \leqslant x_s$ (level))
            penalty := penalty + $\Pi_{cl}$ ($x_s$, $x_d$)

  // Abduction
  foreach $A_d \in$ Names( $P_d$ ) do
    if there does not exist $A_s \in$ Names( $P_s$ ) such that $\mathcal{H} \models A_s \sqsubseteq A_d$
    then add $A_d$ to $P_s$
          penalty := penalty + $\Pi_a$ ( $A_d$ )
    foreach $p_d(f) \in$ Features( $P_d$ ) do
      if there exists $p_s(f) \in$ Features( $P_s$ ) . $p_d(x) \wedge p_s(x)$
      then if $\forall x.p_s(x) \Rightarrow p_d(x)$ is false in the domain associated to $f$
            then add $p_d(f)$ to $P_s$
                  penalty := penalty + $\Pi_{af}$ ($p_d(f)$, $p_s(f)$)
      else add $p_d(f)$ to $P_s$
            penalty := penalty + $\Pi_{af}$ ($p_d(f)$, $\top(f)$)
  foreach $\exists$hasInterest.($C_d \sqcap \geqslant x_d$ (level)) $\in$ Interests( $P_d$ ) do
    if there does not exist $\exists$hasInterest.($C_s \sqcap \geqslant x_s$ (level)) $\in$ Interests( $P_s$ )
      such that $\mathcal{H} \models C_s \sqsubseteq C_d$ and $x_s \geqslant x_d$
    then if there exists $\exists$hasInterest.($C_s \sqcap \geqslant x_s$ (level)) $\in$ Interests( $P_s$ )
            such that $\mathcal{H} \models C_s \sqsubseteq C_d$
          then let $\exists$hasInterest.($C_s \sqcap \geqslant x_s$ (level)) be the concept in Interests( $P_s$ )
                  with maximum $x_s$ among those for which $\mathcal{H} \models C_s \sqsubseteq C_d$ holds
                penalty := penalty + $\Pi_{al}$ ($x_d$, $x_s$)
          else penalty := penalty + $\Pi_a$ ($\exists$hasInterest.($C_d \sqcap \geqslant x_d$ (level)))
          add $\exists$hasInterest.($C_d \sqcap \geqslant x_d$ (level)) to $P_s$
  foreach $\forall$hasInterest.($\neg C_d \sqcup \leqslant x_d$ (level)) $\in$ NoInterests( $P_d$ ) do
    if there does not exist $\forall$hasInterest.($\neg C_s \sqcup \leqslant x_s$ (level)) $\in$ NoInterests( $P_d$ )
```

```
   such that $\mathcal{H} \vDash C_d \sqsubseteq C_s$ and $x_d \geqslant x_s$
then if there exists $\forall$hasInterest.$(\neg C_s \sqcup \leqslant x_s$ (level)) $\in$ NoInterests( $P_d$ )
         such that $\mathcal{H} \vDash C_d \sqsubseteq C_s$
      then let $\forall$hasInterest.$(\neg C_s \sqcup \leqslant x_s$ (level)) be the concept in Interests( $P_s$ )
               with minimum $x_s$ among those for which $\mathcal{H} \vDash C_d \sqsubseteq C_s$ holds
            penalty := penalty + $\Pi_{al}$ ($x_s$, $x_d$)
      else penalty := penalty + $\Pi_a$ ($\forall$hasInterest.$(\neg C_d \sqcup \leqslant x_d$ (level)))
      add $\forall$hasInterest.$(\neg C_d \sqcup \leqslant x_d$ (level)) to $P_s$
return penalty
```

Appendix B

```
(full-reset)

(in-knowledge-base image-tbox image-abox)

; ATTRIBUTE

(define-primitive-role taken-with-camera :domain image :range camera :feature-p t)

(define-primitive-role produced-using-recording-media :domain image :range recording-media :Feature-p t)

(define-primitive-role produces-color :domain recording-media :range color)

(define-primitive-role has-color :domain image :range color)

(define-primitive-role using-recording-media :domain camera :range recording-media)

(define-primitive-role has-segment :domain image :range segment)

; CONCRETE DOMAIN

(define-concrete-domain-attribute iso :type real)

; TOP

(disjoint image camera recording-media color segment)

; IMAGE

(implies (or image-by-color image-by-contrast image-by-category image-by-origin) image)

; IMAGE BY COLOR

(disjoint bw-image color-image)

(implies (or bw-image color-image) image-by-color)

(implies (some produced-using-recording-media bw-film) bw-image)

(implies bw-image (all has-color grayscale-color))

(implies (some produced-using-recording-media color-film) color-image)

(implies color-image (some has-color color))

; IMAGE BY ORIGIN

(disjoint film-image digital-image)
```

```
(implies (or film-image digital-image) image-by-origin)

(equivalent film-image (and image (all taken-with-camera film-camera)))

(equivalent digital-image (and image (all taken-with-camera digital-camera)))

(equivalent bw-film-image (and bw-image film-image))

(equivalent bw-digital-image (and bw-image digital-image))

(equivalent color-film-image (and color-image film-image))

(equivalent color-digital-image (and color-image digital-image))

; IMAGE BY CONTRAST

(implies (or low-contrast-image medium-contrast-image high-contrast-image) image-by-contrast)

(disjoint low-contrast-image medium-contrast-image high-contrast-image)

(implies (<= iso 50) very-low-contrast-image)

(implies (<= iso 100) low-contrast-image)

(implies (and (> iso 100) (< iso 400)) medium-contrast-image)

(implies (>= iso 400) high-contrast-image)

(implies (>= iso 800) very-high-contrast-image)

(implies very-low-contrast-image low-contrast-image)

(implies very-high-contrast-image high-contrast-image)

; IMAGE BY CATEGORY

(implies (or people-image animal-image architecture-image landscape-image transportation-image still-life-image) image-by-category)

(disjoint people-image animal-image architecture-image landscape-image transportation-image still-life-image)

(implies (or mature-people-image male-image female-image children-image family-image) people-image)

(disjoint mature-people-image children-image)

(disjoint male-image female-image)

(implies (or domestic-animal-image mammal-image reptile-image amphibia-image bird-image aquatic-animal-image wild-animal-image) animal-image)

(disjoint domestic-animal-image wild-animal-image)
```

```
(disjoint mammal-image reptile-image amphibia-image bird-image aquatic-animal-image)

(implies (or interior-arch-image exterior-arch-image) architecture-image)

(disjoint interior-arch-image exterior-arch-image)

(implies (or mountains-image river-lake-image ocean-beach-image city-landscape-image agricultural-
fields-image) landscape-image)

(disjoint mountains-image river-lake-image ocean-beach-image city-landscape-image agricultural-fields-
image)

(implies (or water-transportation-image air-transportation-image land-transportation-image) transportation-
image)

(disjoint water-transportation-image air-transportation-image land-transportation-image)

(implies (or nature-still-life product-still-life) still-life-image)

; SEGMENT

(disjoint event location object)

(implies (or event location object) segment)

(disjoint living-thing nonliving-thing)

(implies (or living-thing nonliving-thing) object)

(disjoint animal plant human)

(implies (or animal plant human) living-thing)

(implies (or bird fish amphibia insect reptile mammal) animal)

(disjoint bird fish amphibia insect reptile mammal)

(implies (or primate horse dog cat) mammal)

(disjoint primate horse dog cat)

(implies (or tree bush grass flower fruit vegetable) plant)

(disjoint tree flower grass fruit vegetable)

(disjoint tree flower bush fruit)

(implies (or male-human female-human young-human mature-human) human)

(disjoint male-human female-human)

(disjoint young-human mature-human)
```

(implies (or infant-human child teenager) young-human)

(disjoint infant-human teenager)

(implies elderly-human m3ature-human)

(equivalent man (and male-human mature-human))

(equivalent boy (and male-human young-human))

(equivalent woman (and female-human mature-human))

(equivalent girl (and female-human young-human))

(equivalent old-woman (and woman elderly-human))

(equivalent old-man (and man elderly-human))

(implies (or human-made-object nature-object) nonliving-thing)

(disjoint human-made-object nature-object)

(implies (or construction vehicle appliance furniture utensil nourishment clothes houseware toy tool art jewellery) human-made-object)

(disjoint construction vehicle appliance furniture utensil nourishment clothes toy tool art jewellery)

(disjoint construction vehicle appliance furniture nourishment clothes houseware toy tool art jewellery)

(implies (or bridge street building tunnel) construction)

(implies (or residential-building historical-building commercial-building educational-building religious-building) building)

(implies (or water-vehicle aircraft land-vehicle) vehicle)

(disjoint water-vehicle aircraft land-vehicle)

(implies (or helicopter airplane) aircraft)

(implies (or boat ship) water-vehicle)

(implies (or special-purpose-vehicle train car motorcycle bicycle) land-vehicle)

(disjoint special-purpose-vehicle train car motorcycle bicycle)

(implies (or writing-utensil kitchen-utensil eating-utensil) utensil)

(disjoint writing-utensil kitchen-utensil)

(disjoint writing-utensil eating-utensil)

(implies (or food beverage) nourishment)

```
(disjoint food beverage)

(implies (or tableware decorative-houseware kitchenware) houseware)

(implies (or dishes cutlery) tableware)

(disjoint dishes cutlery)

(equivalent cutlery eating-utensil)

(implies (or plate glass bowl cup) dishes)

(disjoint plate glass bowl cup)

(implies (or on-earth-nature-object above-earth-nature-object) nature-object)

(disjoint on-earth-nature-object above-earth-nature-object)

(implies (or clouds-and-sky moon-sun-stars) above-earth-nature-object)

(disjoint clouds-and-sky moon-sun-stars)

(implies (or park-and-garden soil-sand-stone lake-river-ocean mountain-and-hill) on-earth-nature-object)

(disjoint park-and-garden soil-sand-stone lake-river-ocean mountain-and-hill)

(implies (or antarctica europe asia africa america australasia) location)

(equivalent eurasia (and europe asia))

(implies (or australia new-zealand melanesia micronesia polynesia) australasia)

(disjoint asia africa america australasia)

(disjoint europe africa america australasia)

(disjoint antarctica asia africa)

(implies (or north-america central-america south-america) america)

(disjoint north-america central-america south-america)

(implies (or northern-europe western-europe eastern-europe southern-europe) europe)

(disjoint northern-europe western-europe eastern-europe southern-europe)

(implies (or northern-africa western-africa middle-africa eastern-africa southern-africa) africa)

(disjoint northern-africa western-africa middle-africa eastern-africa southern-africa)

(implies (or western-asia central-asia eastern-asia southern-asia southeastern-asia) asia)
```

```
(disjoint western-asia central-asia eastern-asia southern-asia southeastern-asia)

; COLOR

(disjoint grayscale-color non-grayscale-color)

(implies (or grayscale-color non-grayscale-color) color)

; CAMERA

(disjoint film-camera digital-camera)

(implies (or film-camera digital-camera) camera)

(equivalent digital-camera (and camera (all using-recording-media digital-image-sensor)))

(equivalent film-camera (and camera (all using-recording-media film)))

(equivalent (and image (all produced-using-recording-media film)) (and image (all taken-with-camera film-
camera)))

(equivalent (and image (all produced-using-recording-media digital-image-sensor)) (and image (all taken-
with-camera digital-camera)))

; RECORDING MEDIA

(disjoint bw-film color-film)

(disjoint negative-film transparency-film)

(implies (or film digital-image-sensor) recording-media)

(implies (or bw-film color-film negative-film transparency-film) film)

(equivalent bw-negative-film (and bw-film negative-film))

(equivalent bw-transparency-film (and bw-film transparency-film))

(equivalent color-negative-film (and color-film negative-film))

(equivalent color-transparency-film (and color-film transparency-film))

(implies bw-film (all produces-color grayscale-color))

(implies color-film (some produces-color color))
```

Appendix C

```
package jracer;

import java.io.*;
import java.net.*;
import java.util.*;


/** This class implements a racer client with a plain interface. It allows to establish a
connection with the RACER server, and send to it plain string messages.
*/

public class RacerServer {

   /** The socket that enables communication with the racer server. */
   private Socket racerSocket;

   /** The input stream from the RACER server socket. */
   private InputStream racerInputStream;

   /** The output stream to the RACER server socket. */
   private PrintStream racerOutputStream;

   /** The IP location where the racer server is located. */
   private String racerServerIP;

   /** The port used by the racer server. */
   private int racerServerPort;

   /** This is the string for letting know the racer server the process has ended. */
   private static final String SERVER_END_STRING = ":eof";

/** This method builds a new racer client. */

public RacerServer(String ip,int port) {
   racerServerIP=ip;
   racerServerPort=port;
}

/** This method tries to establish a connection with the racer server. If there is any
problem, an IOException is thrown. */

public void openConnection() throws IOException {
   racerSocket = new Socket(racerServerIP,racerServerPort);
   racerInputStream = racerSocket.getInputStream();
   OutputStream out = racerSocket.getOutputStream();
   racerOutputStream = new PrintStream(out,true);
}
```

```
/** This method checks if a connection is already opened. */

public boolean isOpened() {
   if ( (racerOutputStream != null) && (racerSocket != null) ) return true;
   else return false;
}

public void closeConnection() throws IOException {
   if ( (racerOutputStream == null) && (racerSocket == null) )
      System.out.println("No opened connection or connection is already closed.");
   else {
      if (racerOutputStream!=null) {
         racerOutputStream.print(SERVER_END_STRING);
         racerOutputStream.flush();
         racerInputStream.close();
         racerOutputStream.close();
         racerOutputStream=null;
      }
      if (racerSocket!=null) {
         racerSocket.close();
         racerSocket=null;
      }
      //System.out.println("Connection is closed.");
   }
}

/** This method reads a string from the racer socket connection. */

private static String readFromSocket(InputStream in) throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    int c=in.read();
    baos.write(c);
    while (c!=10) {
                c=in.read();
                if (c!=10)  {
                     baos.write(c);
                    }

          }
   return baos.toString();

}

/** This method sends a command to the RACER server and returns a string with the
answer. If the racer
   server returns an ":ok" message, the answer is the null String; if the racer server
returns an
   ":answer" message, the returned value is the String corresponding to the answer;
and finally, if the
   racer server returns an ":error" message, a RacerException is thrown. */

public String send(String command) throws RacerException, IOException {
   racerOutputStream.println(command);
```

```
    String result=readFromSocket(racerInputStream);
    return parseResult(command,result);
}

public String sendRacerQuery(String query) {
    String result = "Error";
    try {
        if ( !isOpened() ) openConnection();
        result = send(query);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    return result;
}

public String getRacerOutput(String query) {
    String rOutput = "";

    try {
        if ( !isOpened() ) openConnection();
        rOutput = send(query);
        rOutput = rOutput.replace("(","");
        rOutput = rOutput.replace(")","");
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    return rOutput;
}
```

Notes: