POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Master of Science in Computer Engineering

Human-enhanced Multimedia Analysis and Search

Supervisor: Alessandro Bozzon Ph.D.
Assistant Supervisor: Prof. Piero Fraternali

Master Graduation Thesis by:
Nicolò Aquilini, **Student Id. number:** 754915

Academic Year 2011-2012

POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Corso di Laurea Specialistica in Ingegneria Informatica

Human-enhanced Multimedia Analysis and Search

Relatore: Alessandro Bozzon Ph.D.
Correlatore: Prof. Piero Fraternali

Tesi di Laurea di:
Nicolò Aquilini, **matricola:** 754915

Anno Accademico 2011-2012

To my family...
...and to all the friends who shared with me
these unforgettable years of my life.

Acknowledgements

I would like to thank all the people who helped and supported me, during these years and that made possible this important achievement.
First of all, my deepest thank goes to my supervisor Alessandro Bozzon, Ph.D., who guided, supported and inspired me since the beginning of this thesis; his courtesy, his patience and his always prompt assistance made this thesis a precious formative experience.
I would like to thank Prof. Piero Fraternali, which great experience, ideas and suggestions had been an added value to this work. In addition, I would like to express my gratitude to him for having allowed me to participate to the Plenary Meeting of the CUbRIK project, held in London during May 2012: it was a really nice and invaluable experience.
Special thanks to all the Polimi team working on the CUbRIK project, in particular: Prof. Marco Tagliasacchi, Eleonora Ciceri, Ilio Catallo, Andrea Mauri, Chiara Pasini and Luca Tettamanti. Without their help and support this thesis would not have been possible.
I wish also to thank Lorenzo Eccher from Engineering, his help, support and experience were precious allies in the achievement of the goals of this thesis.
Many thanks to the Homeria team, especially to Miguel Ángel Preciado Rodríguez and Javier Garcia Morón, who helped me during the last part of this work.

This achievement would not have been possible without the constant support of my family: my infinite gratitude goes to my parents, my brothers Matteo and Emiliano, my sister-in-law Giada, my little nephew Emanuele, my uncles Ornella and Giuseppe and my cousin Roberta.
I'm also grateful to all the people who shared with me these years at Politecnico and made this life experience really unforgettable, in particular: Mattia, Davide, Claudio, all the Matteo$_s$ and all the Andrea$_s$, Antonella, Sara, Masoumeh, Parandis, Francesco, Luca, Manuele and many others.
Last but not least, a massive thank to all my friends, especially to Elisa and Erika, Fabrizio, Emanuele, Niccolò and Lorenzo: your sincere friendship is a precious part of my life.

Sommario

Negli ultimi anni si è registrata una esplosione nella produzione di contenuti multimediali digitali, specialmente sul Web, con la conseguente necessita di strumenti e metodi sempre più efficienti per la gestione di dati multimediali. *Multimedia Information Retrieval* è ormai una disciplina consolidata, che studia i problemi legati al processamento e alla analisi di contenuti multimediali. Nonostante i progressi scientifici degli ultimi anni, i calcolatori mostrano limitazioni e difficoltà nell'analisi di contenuti multimediali, principalmente dovute a problemi come il cosiddetto "*gap semantico*".
L'analisi e l'interpretazione di contenuti multimediali (come ad esempio immagini e filmati) è invece un'esperienza quotidiana per ogni essere umano. Grazie ai sensi e alle potenzialità del cervello, gli umani sono in grado di avere prestazioni migliori rispetto alle macchine nello svolgere questi compiti. Questa constatazione ha portato ad un rinnovato interesse per l'utilizzo della "*computazione umana*" (in inglese *Human Computation*), cioè l'idea di utilizzare il lavoro umano per eseguire operazioni che i calcolatori non sono ancora in grado di eseguire in modo efficiente, al fianco della computazione automatica.

Questa tesi presenta un approccio per la progettazione e lo sviluppo di "*human-enhanced* Search-based Applications" (hSBA), una nuova classe di sistemi di gestione dati, che considera la computazione umana come elemento fondamentale per i processi di analisi e interrogazione di collezioni di documenti multimediali.
All'interno dell'elaborato, vengono presentati i requisiti che vengono richiesti dalla presenza di esseri umani all'interno dell'architettura della hSBA. Vengono poi discussi i modelli necessari per descrivere i dati e i processi gestiti da una hSBA. Viene inoltre presentato un caso d'uso, sviluppato all'interno di un progetto di ricerca europeo, che esemplifica l'utilizzo dei modelli presentati e la sua implementazione in un'applicazione reale e funzionante. Infine, viene proposta una valutazione sull'implementazione del caso d'uso. I primi risultati sperimentali mostrano che l'introduzione della computazione umana nell'architettura del sistema porta ad un miglioramento non trascurabile delle prestazioni di ricerca del sistema stesso.

Abstract

Recent years witnessed an explosion in the production of digital multimedia contents, especially on the Web, calling for efficient methods and tools for multimedia data management. *Multimedia Information Retrieval* is a mature discipline, devoted to study of the problems of multimedia processing and analysis. Despite the scientific advancements achieved in the last years, the automatic analysis of multimedia contents exhibits several limitations, mainly due to problems such as the semantic gap.
On the other hand, the analysis and interpretation of multimedia contents (i.e., images, videos, audios, etc.) is well developed skills of human beings which are typically able to out-perform machines in several multimedia analysis tasks (e.g., object recognition). This observation led to renewed interest in exploiting *Human Computation*, i.e. the idea of using human effort to perform tasks that computers cannot yet perform efficiently and correctly, to complement machines in their analysis tasks.

This thesis presents a framework for the design and the development of *human-enhanced* Search-Based Applications (hSBA), a new class of multimedia data management systems that involves humans in the content analysis and search processes.
First, we discuss the requirements introduced by the addition of humans in the analysis and search processes. Then, we present a modeling framework to describe the information and the processes that a hSBA should handle. We exemplify the usage of the framework by means of a use case developed within an European research project, describing its implementation in an industrial-strength framework for search based applications. Finally, we provide an evaluation on the application. Preliminary results show that the inclusion of humans in the loop contributes to a non-negligible improvement in the retrieval performances of the application.

Table of Contents

List of Tables

List of Figures

1 Introduction

In the last decades, the realm of *Multimedia Information Retrieval* (MIR) has gained an increasing importance due to the incremented availability of multimedia contents. Automated processing and analysis of multimedia contents have become fundamental tasks that MIR systems have to address. New methods and technologies has been developed in order to allow machines to inspect, infer and characterize the actual content of multimedia items (i.e., videos, pictures, audios, etc.), in terms of semantic concepts, objects, body parts, moods, etc.. Despite the techniques and the technologies introduced, many real world *multimedia retrieval* applications still require the manual production of multimedia objects' metadata, an activity usually performed by experts rather than by machines. *Retrieval* applications, most often referred as *Search-Based Applications* (SBAs), are complex data management systems, in which search engines, even though being fundamental for the application, are just a part of them. SBAs include further components as, for instance, heterogeneous data source integration, content analysis and user interfaces.
A limitation that machines exhibit when dealing with multimedia contents is the low precision of the analysis outcomes that often they produce. Recent studies such as [Bozzon et al., 2012c], show that the uncertainty on the quality of the results, produced by automated multimedia content analysis tools, has a sensible impact on the performances of the overall SBA.
Conversely, the analysis and interpretation of multimedia contents (i.e., images, videos, audios, etc.) is a common skill for every human being. Human beings are typically able to out-perform machines in several multimedia analysis tasks (e.g., object recognition). This simple observation led to renewed interest in exploiting human capabilities among machine computation. The idea of solving the difficult artificial intelligence problems through human power, assumes the name of *Human Computation.*
The inclusion of human performers in the analysis and search processes, introduces new challenges in the design of applications. Humans are not fungible as machines, so the recruitment and the motivation of the workers, are the main challenges to address.
The last decade has seen also the definitive explosion of the *World Wide Web*

and its evolution centered on the users (i.e., Web 2.0) that brought the crowds to be constantly on-line, producing, consuming and sharing contents. Furthermore, the growth of new platforms, such as *Social Networks* (e.g., *Facebook*, *Twitter*), brought users to share on the Web their personal information, their tastes and preferences, as well as their relationships. Hence, on Web 2.0 we interact with *persons*, even before that with just users. Relationship (or *friendship* according to the *Facebook* vocabulary) is a powerful concept that allow to link a single person to other persons, and, recursively, to a huge crowd.
Studies, such as [von Ahn and Dabbish, 2008], show that these crowd spend plenty of time on-line performing many different activities, especially entertainment ones, such as gaming.
Thus, nowadays the Web embeds an enormous potential labour force that could be exploited for several tasks, including multimedia content analysis. Games With A Purpose [von Ahn and Dabbish, 2008] are an effective and appealing example on how to gather and channel this potential into useful work.
But entertainment is not the only way to recruit and motivate workers on the Web. The idea of *out-source* tasks to the crowds on the Web, takes the name of *crowdsourcing*. In the field of *Crowdsourcing* the power of *Human Computation* is exploited by assign the tasks to an "undefined, generally large group of people in the form of an open call" [Howe, 2006]. *Amazon's Mechanical Turk*[1] is an interesting example of an online crowdsourcing platform. It is an online market for small task, in which businesses and developers can access to an on demand and scalable workforce and workers are rewarded with monetary payment.
Thus, today the Web offers new powerful means to easily access and exploit *Human Computation*. Crowds can provide the needed knowledge and workforce to overcome some of the limitations that search-based applications shows, especially when dealing with multimedia contents.
The idea of exploit crowdsourcing in SBAs, takes the name of *crowdsearching*. *Crowdsearching* is a very recent trend in Information Retrieval and can be defined as the promotion of individual and social participation to search-based applications and improve the performance of information retrieval algorithms with the calibrated contribution of humans [Bozzon et al., 2012c].
Within a SBA, *Human Computation* and *crowdsourcing* could be exploited to perform tasks such as object recognition, image classification and audio genre recognition. While, other kinds of tasks, such the ones which include huge numerical calculations, could be still assigned to machines, which are definitively better and more reliable than humans in fulfilling those tasks. So, such observations lead us to the idea that humans' and machines' skills could be conveyed within a hybrid system, for multimedia analysis and search.
In our work we aim to propose a new framework in which humans' efforts can be used not as a replacement of machines, but as an *improvement* and an *enhancement* to the machine work. We define a *human-enhanced* Search-Based Application (hSBA) as the addition of *Human Computation* tasks to a common search-based application.

[1]http://www.mturk.com

1.1 Original Contribution

Our work embraces different research areas: *Multimedia Information Retrieval*, *Search-based applications*, *Human Computation* and *Crowdsearching*, to which we provide the following original contributions:

- a discussion on the characteristics and the requirements that a multimedia search-based application should fulfil in order to support human computation.
- the definition of reference models (i.e., data models and process models) to address the problem of the introduction of human computation and crowdsourcing in a default search-based application.
- the design of a reference multimedia search-based application architecture that enables the support of crowdsourced tasks among automatic ones.
- a demonstration, through a use case, of the benefits that crowdsourcing introduces in the quality performances of a multimedia search-based application.

1.2 Thesis Organization

The thesis is organized as follows:

- **Chapter 2** introduces the context of our thesis, providing the needed concepts that we will use throughout the whole discussion on *Multimedia Information Retrieval*, *Human Computation*, *Crowdsearching* and *Search-based applications.*
- **Chapter 3** highlights the requirements that drive the design of a multimedia search-based application, that exploit human computation and crowdsourcing tasks.
- **Chapter 4** introduce a framework for the design and the development of a human-enhanced search-based application.
- **Chapter 5** presents the use case of a trademark logo detection application that materializes the concepts and the models introduced in the proposed framework.
- **Chapter 6** describes the implementation of the trademark logo detection application and provides an evaluation of its performances.
- **Chapter 7** draws the conclusions of our work and presents the future works directions of our study.

2 Background

In this chapter we present the main research areas in which a *Human-Enhanced Multimedia Search-based Application* (hSBA) falls, introducing terms and concepts that we will encounter further throughout our thesis.
In Section 2.1 we introduce the main concepts of *Multimedia Information Retrieval*, starting from the classical theoretical definitions to the main state-of-the-art techniques used in real systems.
Section 2.2 gives a definition of *Human Computation*, introduces its concepts and provides an overview on the existing Multimedia Information Retrieval systems that exploit humans and crowds in their processes.
Then, in Section 2.3 we present the novel concept of *Crowdsearching*, reporting recent studies on a framework to exploit crowd tasks in Multimedia Information Retrieval and on a reference model for *crowdsearching* search.
Finally, in Section 2.4, we present the steps in the design of a *Search-based application*, introducing the reference architecture, the reference processes and the dimensions to be considered.

2.1 Multimedia Information Retrieval

2.1.1 Basic Concepts

Information retrieval deals with the representation, storage, organization of, and access to information items [Baeza-Yates and Ribeiro-Neto, 1999]. As stated in [Manning et al., 2008] Information retrieval (IR): "*is finding material (usually documents) of an unstructured nature (usually text) that satisfy an information need from within large collections (usually stored on computers).*"
Understanding the user information need is not an easy task and according to the state-of-the-art the user cannot specify the information need using the natural language. Instead, the user must first translate this information need into a query which can be processed by the IR system. In its most common form, this translation yields a set of keywords (or index terms) which summarizes the description of the user information need.

Given a user query, a good IR system should retrieve all the information which might be relevant to the user, and rank them according to a degree of relevance [Baeza-Yates and Ribeiro-Neto, 1999]. This concept of *relevance* is crucial in IR, but relevancy might be subjective with respect to different users.
Multimedia Information Retrieval (MIR) is a branch of the classical IR, in which information items are multimedia items, like images, audios or videos. Thus, MIR has to deal with more complex data with respect to the classical IR.

2.1.2 Information Retrieval Models

In classical Information Retrieval documents are represented by a set of keywords, called *index terms*. *Index terms* are used to index and summarize the content of a document. The relevance of an *index term* within a document is captured by a numerical weight, assigned to the index term itself. Thus, a document d_j can be represented an *index term* vector $[w_1j, w_2j, \dots, w_Mj]^T$ in which each w_ij is the numerical weight > 0, associated to the term t_i and the document d_j.
In IR there are several models in order to build the document representation, the main ones are:

- the *Boolean model*
- the *Vector Space model*

The *Boolean model* is the simplest IR model, the weight w_ij is equal to 1 if the term t_i is contained in the document d_j, otherwise it is equal to 0. Queries are defined as Boolean expressions over the *index terms*. Each document is represented as a set of word and is independent by the frequency or the position of the terms. Despite its simplicity, this model holds some drawbacks: it does not allow any ranking on the retrieved documents and the translation of an information need in a Boolean expression is not a simple task.

((*text* OR *information*) AND *retrieval* AND NOT *theory*)

An example of Boolean expression

The *Vector Space model* (VSM) represents documents and queries as vectors in the term space. The *index term* relevance in a document is represented by a real valued numerical weight. Documents that are close to each other in the vector space are similar to each other. In the *Vector Space model* a similarity SC(q, d_j) is computed between the query and each document, and results can be ranked according to this similarity. There are several measures that can be used to compute the similarity between documents, the most widely used ones are *Euclidean distance* and *Cosine similarity.*

$$\textbf{Euclidean distance:} \quad SC(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

$$\textbf{Cosine similarity:} \quad SC(\mathbf{p}, \mathbf{q}) = \cos(\alpha) = \frac{\mathbf{q}^T \mathbf{p}}{||\mathbf{q}|| ||\mathbf{p}||}$$

The weighting of the index terms should be proportional to the importance of the of the term in the document and in the document collection. The *tf*idf* (*term frequency – inverse document frequency*) weight is often used as weighting factor in IR. The weight value increases proportionally to the number of times a term appears in a document, but decreases with the frequency of the term in the document collection.

$$tf * idf(t,d,D) = tf(t,d) * idf(t,D)$$

where

$$tf(t,d) = \frac{tc(t,d)}{max\{tc(w,d) : w \in d\}}$$

$$idf(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

$tc(t,d)$: term count of t in d

$max\{tc(w,d) : w \in d\}$: maximum term count of d

$|D|$: the total number of documents in the corpus

$|\{d \in D : t \in d\}|$: number of documents where the term t appears

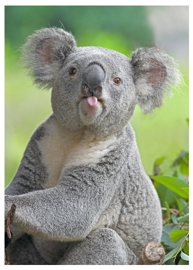
2.1.3 From text to multimedia

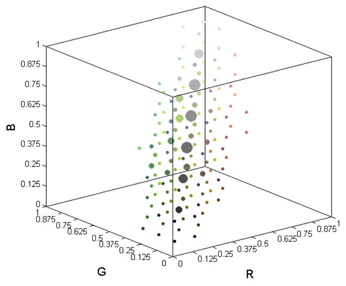
Classical Information Retrieval deals with terms, documents and vocabularies, in other words it deals with text. The advent of the Web and the explosive growth of digital media lead the user information needs to involve also multimedia resources (images, videos, etc.). Multimedia Information Retrieval aims to apply the known and widely studied methods of IR in the realm of multimedia. Over the years, different methods and approaches to MIR has been proposed, the main ones are: *Content-based MIR* and *Concept-based MIR*.
In *Content-based MIR*, the information within a multimedia object is inferred from the content of the object itself (e.g., colors and textures within an image, pitches in a audio stream, etc.). On the contrary, *Concept-based MIR* relies on a set of textual metadata attached to the multimedia item, that characterize it. Figure 2.1 depicts two possible representations of the same sample image in Figure 2.1(a): Figure 2.1(b) draws the color histogram as *Content-based* representation of the image, while Figure 2.1(c) provides a set of concepts (a.k.a. *tags*) that can be associated to the image in a *Concept-based* perspective.
According to the scope of this thesis, we will focus just on the *Content-based* approach.

As in IR a document is represented by a set of terms, in *Content-based MIR*, a media can be represented by its features. For the sake of clarity, let us consider just the case of Image Retrieval, which was one of the first techniques in the field of MIR.

2.1.4 Content-based Image Retrieval

Content-based Image Retrieval CBIR deals with images and relies on the concept of *visual features*. A visual feature is an image property such as color, texture, and shape. Those features should have some properties, like the invariance to scaling, cropping and rotations or the robustness to illumination conditions, viewpoint change and occlusions. Figure 2.2 depicts a set of transformation applied to an image.


(a) A koala

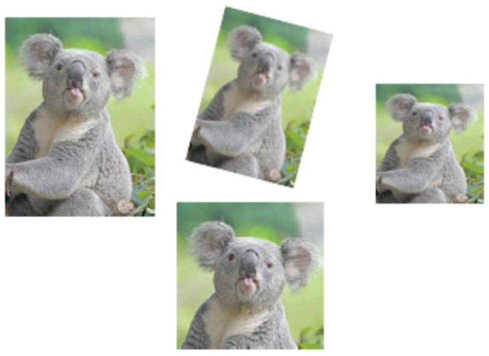


(b) The color histogram of the image

koala, marsupial, animal, Australia, ...

(c) A set of concepts related to the image

Figure 2.1: *Content-based* and *Concept-based* representations of an image


(a) Rotation, scaling and cropping transformations

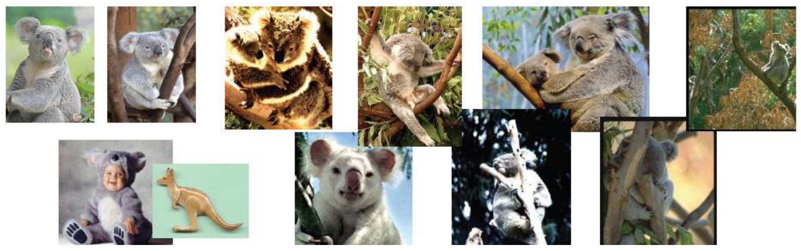

(b) Occlusion, illumination and viewpoint transformations

Figure 2.2: Example of image transformations [Yan and Hsu, 2008]

Visual features may be global, if they summarize properties of the whole image or local, if they refer to properties of particular structures within the image. Figure 2.3 draws an example of visual feature representation of an image.

As for textual documents in *Vector Space Model*, the final representation of an image is a vector that can be plotted onto a feature space. Similar images are represented by vectors that are close in the feature space; with respect to

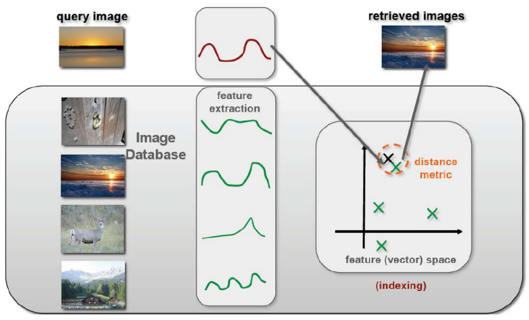


Figure 2.3: The *visual features* approach [Yan and Hsu, 2008]

documents the metric used to establish the closeness of two vectors is the *Cosine similarity*. Such metric, according to [Tan et al., 2005], has been judged to be most reliable in Image Retrieval.
In *Content-based Image Retrieval* an image can be described either from its overall properties (e.g., color, textures, etc.) or by the properties of some particular structures within it (e.g., corners, peculiar structures, etc.). In the next section we introduce the *global features* and *local features* approaches; the latter one allow us to present some of the image processing and description techniques that we will use in our multimedia search-based application.

2.1.4.1 Global features

Thank to its robustness against image size and rotation, color is one of the most widely used features in Image Retrieval.
Given a discrete color model defined by some color axes (e.g., RGB, HSV, etc.), the *color histogram* Swain and Ballard [1991] is obtained by discretizing the image colors and counting the number of times each discrete color occurs in the image array. *Color histograms* are invariant to translation and rotation about the viewing axis, and change slowly under the change of angle of view, change in scale and occlusion. The value of a color bin in the histogram is proportional to the number of pixel having that color, like term frequency within a document.
Another global feature used to characterize images, is the *texture*. *Texture*

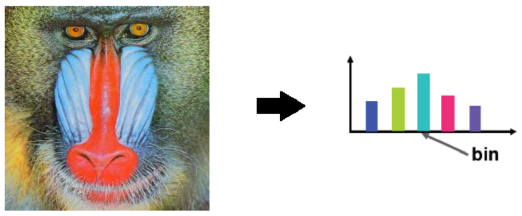


Figure 2.4: An example of color histogram

refers to the visual patterns that have properties of homogeneity that do not result from the presence of only a single color or intensity. *Texture* carries important information about the structural arrangement of surfaces and their

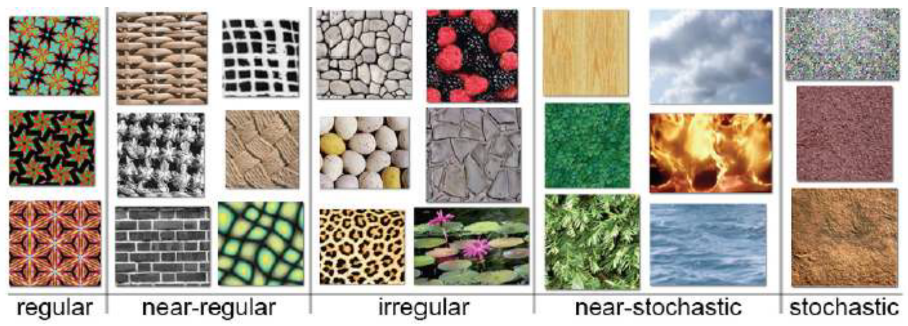


Figure 2.5: A classification of textures

relationship to the surrounding environment. Figure 2.5 report a classification of textures that we can find within images. There are several methods in order to compute *texture features*:

- *Structural*
- *Statistical*
- *Spectral*

Structural methods describes the arrangement of the texture elements within the images. Indeed, *statistical* methods characterize a texture in terms of its statistical features. Finally, *spectral* methods, analyze the textures in the spatial-frequency domain.
One of the most popular set of *statistical* texture features are the ones presented in [Tamura et al., 1978], selected according to psychovisual experiments. The so-called *Tamura's features* are a set of six features: *coarseness*, *contrast*, *directionality*, *lineliness*, *regularity* and *roughness*.
In *Spectral* methods, the *spectrum* of an image, computed through the *2D discrete Fourier Transform* (i.e., DFT), is analyzed to extract information on the texture characterizing the image. A lack of the *Fourier transform*-based spectrum analysis is the loss of spatial information. A local spatial-frequency analysis can be performed using *Gabor filters* [Manjunath and Ma, 1996].

2.1.4.2 Local features

Key-point detection In the *local features* approach, the goal is to characterize an image through features extracted from local regions of the image itself. The position of these regions is determined by a set of *interest points*. The number of interest points may vary depending on the image. Thus, in order to compute local features we need to first detect the *interest point* (or *key-point*), then for each interest point computed a descriptor of the surrounding image region. Figure 2.6 depicts the key-points detected within an image.
One of the first methods proposed to perform feature detection is the *Harris corner detection* algorithm. This algorithm is based on the observation that around a corner point, the image intensity will change greatly when the window is shifted in an arbitrary direction. The *Harris interest point detector* is based on the difference between a patch with patches shifted by a small amount in

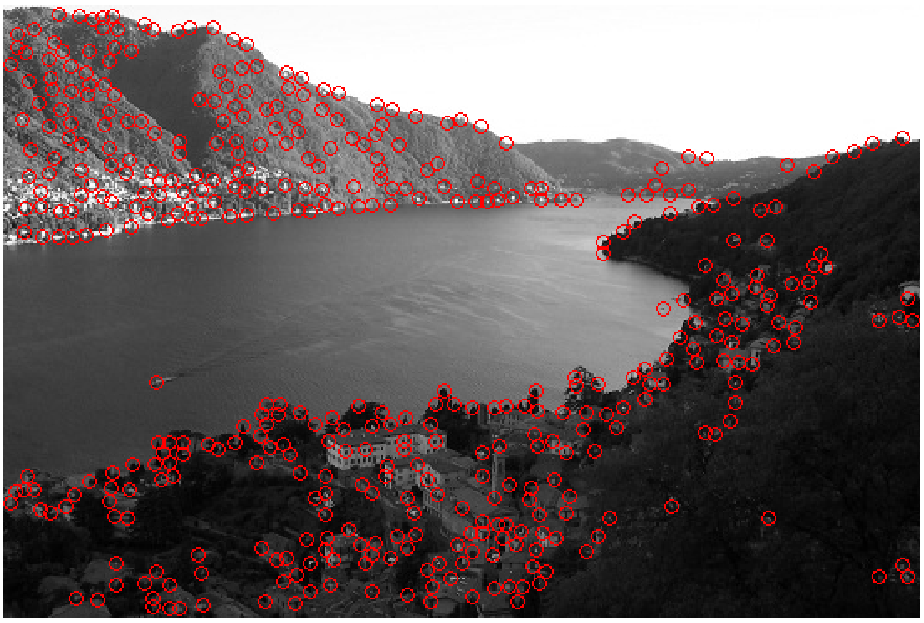


Figure 2.6: An example of the key-points detected within an image

different directions. For each point within the image these differences are computed, and a matrix capturing the intensity structure of local neighbor can be derived. Analyzing the eigenvalues of this matrix, we can classify the each point and establish whether a point is a constant intensity one, an edge or a corner.
The main drawback of the *Harris detector* is that is not invariant to scale and affine transformations. To overcome the limitations of the *Harris detector*, scale invariant detectors have been proposed [Lowe, 2004]. The concept behind a scale invariant detector is that what the detector identifies is not just the position of the key-point, but also the scale at which the point is a key-point.
Scale invariant detector rely on the assumption that scale changes are the same in every direction, although they exhibit robustness to weak affine transformations. Such detectors search for local extrema not just in the (x, y) space of the image, but in a 3D space, where the third axis is the scale coefficient. Such tridimensional image space is called the scale-space representation of the image. The scale space of an image is a function L(x, y, σ) produced by the convolution of a variable-scale Gaussian G(x, y, σ), with the input image I(x, y). In order to seek key-points in the scale-space several methods have been proposed:

- *Laplacian of Gaussians* (LoG)
- *Hessian-Laplace*
- *Harris-Laplace*
- *Difference of Gaussians* (DoG)

The latter one is an efficient solution to detect stable key-points locations [Lowe, 2004]. In DoG, the input image is convolved with the *Difference of Gaussian* function D(x, y, σ), such function can be computed the difference of two nearby scales separated by a constant multiplicative factor k. Local minima and maxima of the D(x, y, σ) are detected by comparing each point with all its neighbor along x, y, and scale axes. A point is considered a key-point only if its D(x, y, σ) value is larger than all of these neighbors or smaller than all of them. Figure 2.7 draws the DoG scale space.

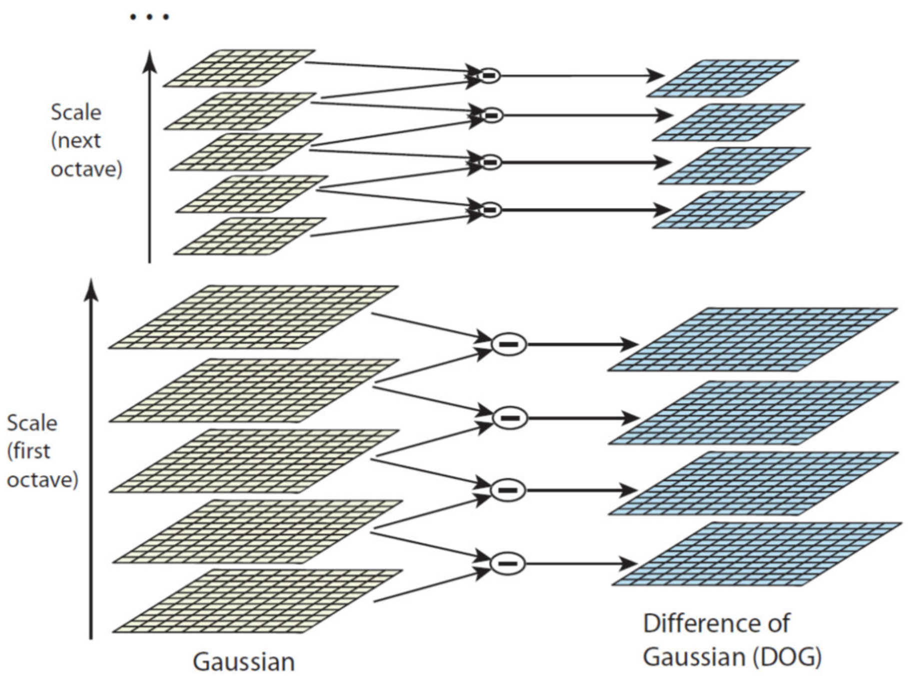


Figure 2.7: The computation of the DoG scale space [Lowe, 2004]

In order to achieve rotation invariance to each key-point detected an orientation coefficient is computed, based on local properties of the image. Figure 2.8 shows the dominant orientations computed for the key-points of a sample image.

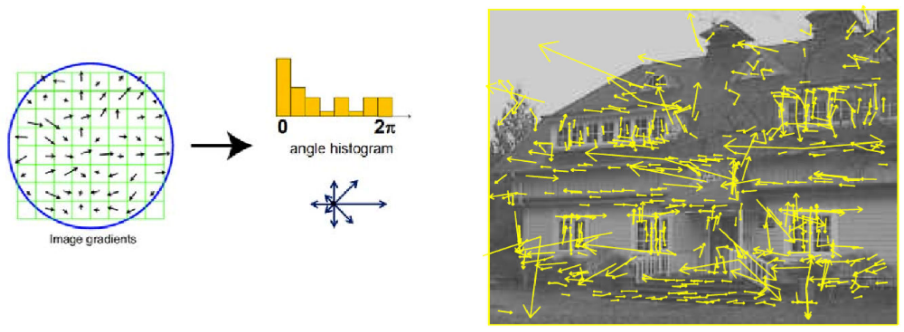


Figure 2.8: The dominant orientations of the image key-points [Lowe, 2004]

Key-point descriptors The location of the key-points within an image, the output of the feature detection, is not enough to achieve *Content-based Image Retrieval* goals, we need to characterize those peculiar structures within the image, computing the so-called feature descriptors.
A widely used technique to compute feature descriptors is the *SIFT* (*Scale Invariant Feature Transform*) [Lowe, 2004]. *SIFT* is based on the *DoG* feature detector and characterize each key-point attaching to it information about the orientations histograms of its surrounding regions (a.k.a. feature vectors or key-point descriptors). For each key-point a 128 dimensional vector is computed, that is formed by 16 8-bins orientation histograms related to a 4x4 region around the key-point. *SIFT* is scale invariant thank to the use of the DoG detector, it is rotation invariant if the histograms are computed with respect to the key-point orientation and is robust to illumination changes if the feature vector is normalized. Figure 2.9 shows a visual representation of the SIFT descriptors.

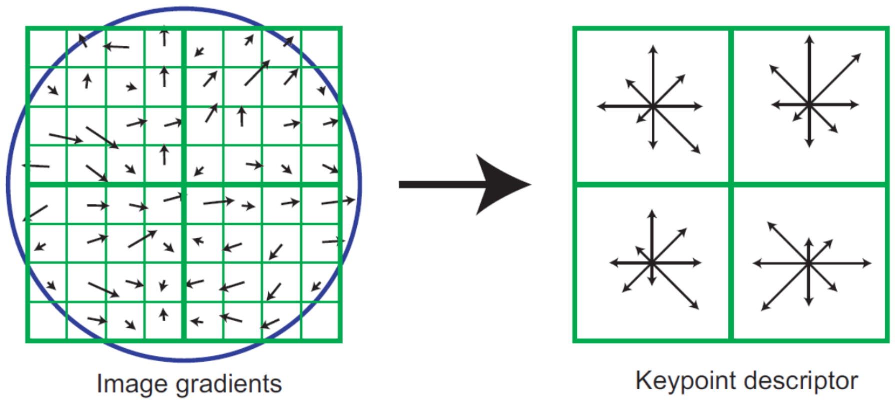


Figure 2.9: The SIFT local descriptors [Lowe, 2004]

An alternative to *SIFT* is *SURF* (*Speeded-Up Robust Features*) [Bay et al., 2006] which uses a more efficient detector and compute a 64 dimensional feature vector based as in *SIFT* on orientations in the key-point local neighborhood, but more robust to noise within the image.

Matching Once feature vectors have been computed for a set of images, we should be able to find the correspondences between key-points from different images. A way to perform the key-point matching have been proposed in [Lowe, 2004]: for each key-point descriptor in an image A we find the closest descriptor of image B (e.g., the one that minimize the *cosine similarity*). We consider a match correct if the ratio of the distance from the closest neighbor to the second closest is above a given threshold.
As the matching process is subject to errors, verification techniques are needed in order to find robust key-points correspondences. The key-points of the same objects in two different images are related by a geometrical transformation (e.g., affine, perspective, etc.). Within a set of observed data (i.e., the set of key-point matches) there are inliers and outliers. *Spatial verification* is based on the obser-

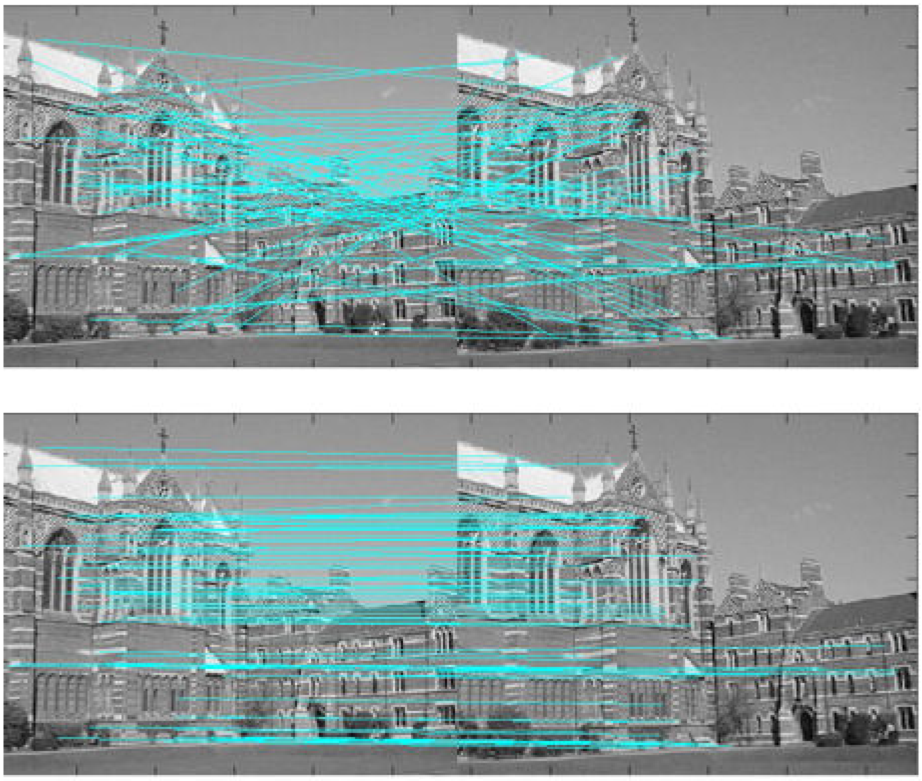


Figure 2.10: The key-points correspondences prior to (above) and after (below) *spatial verification*

vation that outliers do not comply with the geometrical transformation. Figure 2.10 draws an example of the key-point correspondences prior and after *spatial verification.* The goal of spatial verification is to estimate the parameters of such transformation. Once the transformation parameters are known, a model can be built and according to the model correspondences can be refined.

Random Sample Consensus (*RANSAC*) [Fischler and Bolles, 1981] is the typical algorithm used to perform *spatial verification.* In *RANSAC* algorithm the parameters of the geometrical transformation are estimated by random sampling.

Algorithm

1. *Randomly select a sample of s data points from S and instantiate a model from this subset;*

2. *Determine the set of data points S_i which are within a distance threshold t of the model. The set S_i, is the consensus set of the sample and defines the inliers of S.*

3. IF *the size of S_i (i.e., number of inliers) is greater than some threshold T:*
   re-estimate the model using all the points in S_i and terminate,

```
OTHERWISE
    select a new subset and repeat the above;
```

4. *After N trials the largest consensus set S_i is selected, and the model is re-estimated using all the points in the subset S_i.*

2.1.4.3 Visual words

Similar to terms in a text document, an images has local interest points or key-points defined as salient image patches (i.e., small regions) that contain rich local information of the image. Images can be represented by sets of key-point descriptors, but the sets vary in cardinality and lack meaningful ordering. This creates difficulties for learning methods (e.g., classifiers) that require feature vectors of fixed dimension as input [Yang et al., 2007].
A solution is to perform clustering in the feature space, and group descriptors vectors in clusters. Each cluster represents a *visual word*. Thus, for each image the histogram of *visual words* can be computed. Once those histogram have been computed, we can treat *visual words* and images as they were terms and documents in classical Information Retrieval and apply the related models (e.g., *Vector Space Model*) and techniques (e.g., $TF * IDF$). Figure 2.11 depicts the construction of a *visual word* vocabulary.

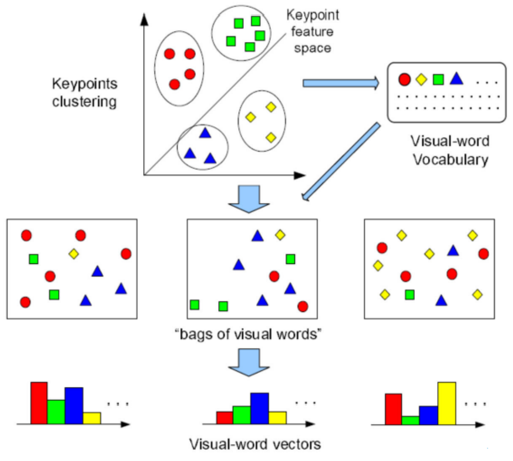


Figure 2.11: The construction of a *visual words* vocabulary [Yang et al., 2007]

2.1.5 Concept-based Image Retrieval

Despite its potential, *Content-based Image Retrieval* (CBIR) presents some limitations and practical issues that still keep keyword-based query the primary input method in multimedia search. The main limitations of *CBIR* are:

- the need of translate information needs into images;
- the inefficiency due to the indexing of high-dimensional features;
- the difficulty to find proper query examples as initial queries.

In *Concept-based Image Retrieval* images are translated into documents composed by semantic concepts and retrieved by matching these semantic concepts and the queries. The semantic concepts aims to fill the gap between user information need and the low-level multimedia content. However, express in words the content of an image is a challenging intellectual effort: for instance, what is the best word to express a picture of a bus? Which one of this words *vehicle*, *truck* or simply *bus*?
The extraction of concepts can be performed manually by humans, asking to persons to provide a set of keywords given a single image or video (i.e., tagging). An example of picture tagging is given by *Flickr*[2], in which millions of users are involved in upload and tag tons of pictures; this scenario is called a *social tagging*. Another peculiar example of concept extraction for multimedia resources is given by image labeling systems realized through *Games With A Purpose* (GWAP) Web applications (see Section 2.2.4).

2.1.6 Multimedia Indexing

When dealing with Multimedia Information Retrieval system and multimedia databases the word *query* assumes a novel meaning. In contrast to traditional database applications, where point, range, and partial match queries are very important, multimedia databases require a search for all objects in the database which are similar (or complementary) to a given search object [Böhm et al., 2001]. Indeed, we refer to those query as *similarity queries*. *Similarity queries* are strictly related to the *similarity measure*. There's no a general definition of similarity measure as it is highly application dependent (e.g., *Euclidean distance*, *cosine similarity*, etc.).
In multimedia databases there are usually two task to de defined in *similarity queries*: *ξ-similarity* and *NN-similarity*. *ξ-similarity* means that the result of the similarity query should be the set of object which similarity to the given object is below the threshold ξ, where ξ is a real number. Instead in *NN-similarity* (i.e., *Nearest-Neighbor similarity*) the result of the query are the N object which are most similar to the given object.
Currently, solution adopted to solve similarity search problems in MIR systems are mostly feature-based:

- extract important information (features) from multimedia object;
- map the features into high-dimensional feature vectors;
- search the database of feature vectors for objects with similar feature vectors.

In such scenario, in order to perform an efficient similarity search, it is necessary to store the feature vectors in an high-dimensional index structure.
Quadtrees, *kd-trees*, *R-trees* and *R*-trees* are indexing structures commonly used for storing high-dimensional feature vectors.

[2]http://www.flickr.com

2.2 Human Computation for Multimedia Information Retrieval

2.2.1 Basic Concepts

Human Computation (HC) can be defined as "*the idea of using human effort to perform tasks that computers cannot yet perform, usually in an enjoyable manner*" [Law and von Ahn, 2009]. There exists some problems that computers are either unable to or are very poor at solving, but they are easy for human to solve (e.g., tag images, determine if a page is relevant, determining song genre, check pages for offensive contents, etc.). The information collected by HC systems is useful for machine learning systems [Yuen et al., 2009]. *Human Computation* is related to, but not synonymous to other terms such as *Crowdsourcing*, *Social Computing* and *Collective Intelligence.*

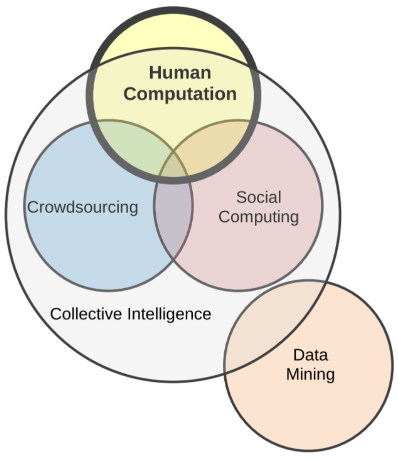


Figure 2.12: *Human Computation* w.r.t. *Crowdsourcing*, *Social Computing* and *Collective Intelligence* [Quinn and Bederson, 2011]

Crowdsourcing refers to "*the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call*" [Howe, 2006]. It is a term derived from outsourcing. Users from the crowd are motivated to participate and accomplish crowdsourced tasks for both intrinsic and extrinsic motivations. A user can be motivated to participate by his, or her, desire to do something good and to help someone, or because his, or her, efforts on the tasks results in a public recognition (i.e., reputation), or because the nature of the tasks engages and entertains him, or her, in accomplishing the task (i.e., tasks designed as computer games, GWAP).
One of the most effective way to recruit users, or workers, is to reward them with money. *Amazon's Mechanical Turk*[3] is an online market for small task (computational or not) that uses monetary payment. Usually the money involved for the completion of a single task is around few dollar cents. Another example of

[3]http://www.mturk.com

the use of monetary motivation is *ChaCha*[4], a search service in which human workers are involved in interpret queries and looking for relevant results. An issue these crowdsourcing platforms has to deal with, is the tendency of workers to cheat to increase their earnings; this tendency is boosted by the fact that often workers are anonymous for the platform.
Social Computing is a term related to communication technologies as blogs, wikis and social networks. Despite this relationship, the purpose of social computing is not a computation. An effective definition of social computing is given in [Parameswaran and Whinston, 2007]: "*applications and services that facilitate collective action and social interaction online with rich exchange of multimedia information and evolution of aggregate knowledge*".
Collective intelligence is a broad term that refers to "*groups of individuals doing things collectively that seem intelligent*" [Malone et al., 2009]. The main distinction between *Human Computation* and *Collective Intelligence* is that the latter one depends on a group of participants, while in Human Computation the task could be performed by just one human worker in isolation.

2.2.2 HC for MIR systems

In this section we briefly present some studies that demonstrate how Human Computation can be exploited in order to validate the outcomes of multimedia information retrieval systems.
In the field of MIR, *Human Computation* is used:

- to improve, extend and validate the outcomes of automatic annotations components in *Content-based* Multimedia Retrieval systems.
- expand or reformulate user queries.
- to generate reliable metadata on multimedia objects, such as tags and annotations, to be used in *Concept-based* Multimedia Retrieval systems.

The MIR systems use automatic technologies to annotate the multimedia objects (e.g., automatic speech recognition, local feature detectors, etc.); such annotations are validated by the crowd, which gives back feedbacks to the systems. The human feedbacks can be used either to establish the quality of an annotation or to enrich to create a new annotation for the multimedia object.

2.2.2.1 Crowdsourcing Rock N' Roll Multimedia Retrieval

Multimedia Retrieval is a topic that arouses interest in research, but up to now does not have many real-world applications. For instance, video retrieval on the Web is still mainly based on textual queries and tags.
[Snoek et al., 2010] presents a real-world video search engine based on advanced multimedia retrieval technology, which allows for user-provided feedback to improve and extend automated content analysis results.
The search engine uses archived video footage of the *Pinkpop*[5] festival.
Humans are asked to provide feedbacks on the visual concepts associated to video fragments by the system. Due to the specificity of the videos selected

[4]http://www.chacha.com
[5]an annual Dutch rock festival

for the experiment, the set of visual concepts is limited to 12 concepts, called *concert concepts*. Human performers interacts with the video search engine by means of a timeline-based video player (see Figure 2.13).
The player enables users to watch and navigate through a single video concert.

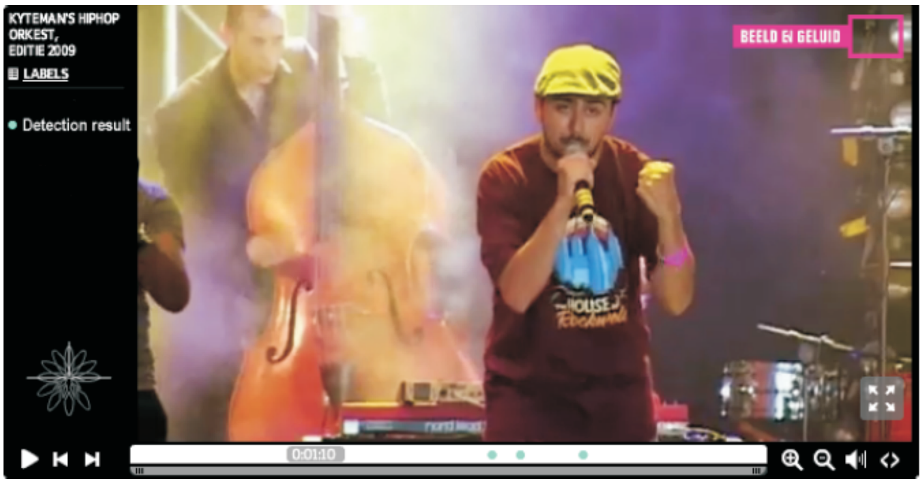


Figure 2.13: The crowdsourcing video player

Colored dots on the timeline mark the location of an interesting fragment corresponding to an automatically derived label. Users can browse the generated labels and indicate that they agree with the automatically detected label for the video fragment. If they disagree, users are asked to correct the label. Within a few clicks the user can select another pre-defined label or create a new label on demand. In addition, users are allowed to indicate whether the start or end of the fragment is inconsistent with the label. If needed, the user can also manually select more concept labels.
Besides providing feedbacks on the automatically detected labels, humans are allowed to comment on the individual fragments, share the fragment through e-mail or *Twitter*, and embed the integrated video player, including the crowdsourcing mechanism, on different websites.
The crucial point of the demonstration is the *motivation* to partecipate of human performers. The demonstration focuses on a dedicated user community of rock n' roll enthusiasts. In order to find a balance between an appealing user experience and a maximized user participation, humans are motivated to participate by providing them with access to a selection of exclusive, full-length concert videos.

2.2.2.2 Crowdsourcing Event Detection in YouTube Videos

The amount of video content published on *YouTube* is constantly growing, up to 48 hours of video uploaded every minute according to official statistics. This huge quantity of data implies the need of advanced search techniques in order to retrieve relevant videos.
In [Steiner et al., 2011] a browser extension that allows the video navigation through events, is presented.

The paper introduces three types of events: *visual*, *occurrence* and *interest-based* ones. Both crowdsourcing and automated components are used to detect these types of events within videos.
Event detection in videos is an ideal candidate for crowdsourcing, as each video is an independent object in itself, i.e., the whole set of all existing *YouTube* videos can be easily split into subtasks by just analyzing one video at a time.
Visual and *occurrence* events are detected on-the-fly in an automated fashion using respectively computer vision (e.g., local histograms) and NLP techniques. For those events types, shots and named entities in the video are detected once by the first *YouTube* user that watches the video. Subsequent viewers can directly profit from the generated annotations.
For *interest-based* events, acknowledging that points of interest within a video might change over time, purposeful navigation events by all users are captured. This allows for the generation of a heat-map-like overlay on top of the video shots, which results in an intuitive representation of popular scenes.
Figure 2.14 depicts the event detection *YouTube* player.

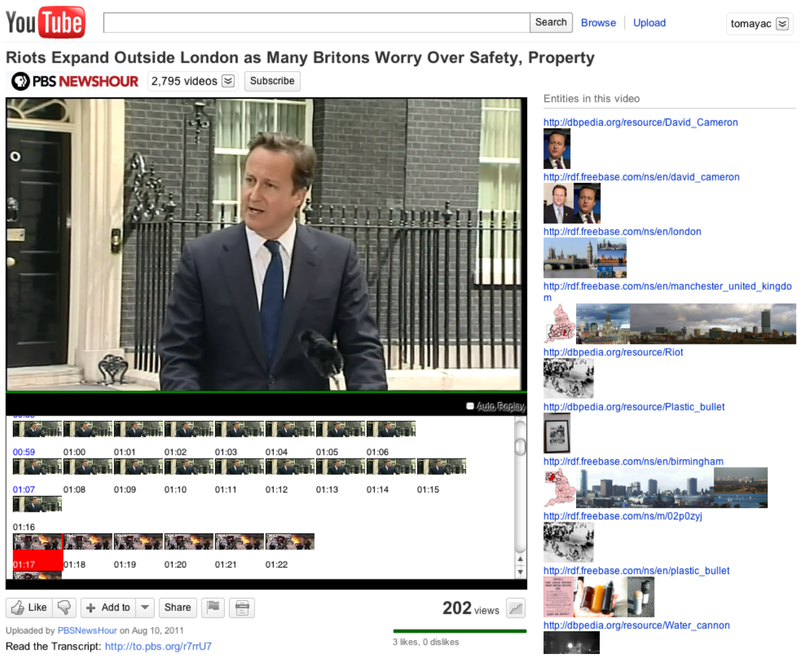

You Tube
Search
Browse
Upload
tomayac
Riots Expand Outside London as Many Britons Worry Over Safety, Property
PBS NEWSHOUR
2,795 videos
Subscribe
Auto Replay
00:59
01:00
01:01
01:02
01:03
01:04
01:05
01:06
01:07
01:08
01:09
01:10
01:11
01:12
01:13
01:14
01:15
01:16
01:17
01:18
01:19
01:20
01:21
01:22
Like
Add to
Share
202 views
Uploaded by PBSNewsHour on Aug 10, 2011
Read the Transcript: http://to.pbs.org/r7rrU7
3 likes, 0 dislikes
Entities in this video
http://dbpedia.org/resource/David_Cameron
http://rdf.freebase.com/ns/en/david_cameron
http://rdf.freebase.com/ns/en/london
http://rdf.freebase.com/ns/en/manchester_united_kingdom
http://dbpedia.org/resource/Riot
http://dbpedia.org/resource/Plastic_bullet
http://rdf.freebase.com/ns/en/birmingham
http://rdf.freebase.com/ns/m/02p0zyj
http://rdf.freebase.com/ns/en/plastic_bullet
http://dbpedia.org/resource/Water_cannon

Figure 2.14: The event detection *YouTube* player extension

2.2.2.3 Visual-based Plant Species Identification from Crowdsourced Data

Another example of a crowdsourced application used to validate the outcomes produced by automated components is reported in [Goëau et al., 2011].
The case of study on which the study focuses is the identification of plants species given pictures of plant leafs. Such task is usually impossible for the general public, and often a difficult task for professionals, such as farmers or

wood exploiters and even for botanists themselves.
The automated part of the application deals with the content-based identification of plant leafs, and is composed by the following steps:

1. local features extraction;
2. local features nearest neighbors search with an efficient hashing-based indexing scheme;
3. spatially consistent matches filtering with a *RANSAC* algorithm using an affine model;
4. basic top-K decision rule as classifier: for each species, the number of occurrences in the top-K images returned is used as its score.

Users query the system, through a Web application, providing a scan or a picture of a leaf. The system returns and displays the top-3 species with the most similar pictures. The user can then either select and validate the suggested species, or he can choose among other species in the list, or even enter a new species name if it is not available. The uploaded image used as query is temporary stored with its associated species name. Then, other users might interact with these new pictures later, after that some professional botanists involved in the project validate the images and theirs species names.
Figure 2.15 draws the Web interface developed for the leaf classification task. Scans of leaves were collected over two seasons, between July and September

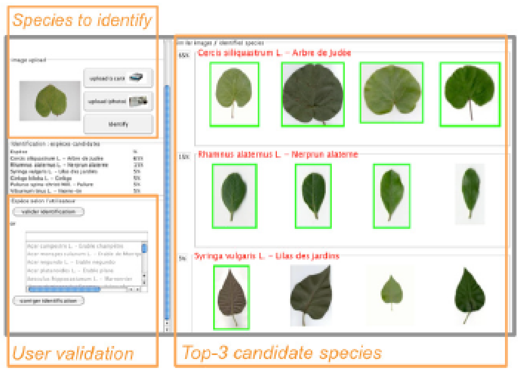

Species to identify
Cercis siliquastrum L. - Arbre de Judée
Rhamnus alaternus L. - Nerprun alaterne
Syringa vulgaris L. - Lilas des jardins
User validation
Top-3 candidate species

Figure 2.15: The leaf classification Web interface

2009 and between June and September 2010 thanks to the work of active contributors from *Tela Botanica*[6] social networks. The first online application did contain 457 validated scans over 27 species and the link was mostly disseminated through *Tela Botanica*. It finally allowed to collect 2228 scans over 55 species.

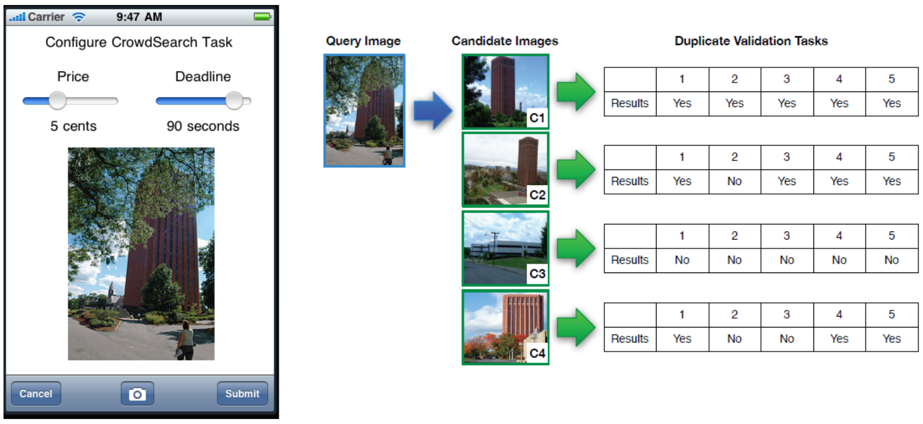
2.2.2.4 CrowdSearch: Exploiting Crowds for Accurate Real-time Image Search on Mobile Phones

[Yan et al., 2010] presents *CrowdSearch*, an accurate image search system for mobile phones. *CrowdSearch* combines automated image search with real-time

[6]a large social botany network: http://www.tela-botanica.org/

human validation of search results. Automated image search uses a combination of local processing on mobile phones and remote processing on powerful servers. For a query image, the system generates a set of candidate search results that are packaged into tasks for validation by humans. Real-time validation uses the *Amazon Mechanical Turk*, where tens of thousands of people are available to work on simple tasks for monetary rewards. Search results that have been validated are returned to the user. Figure 2.16 depicts the *CrowdSearch* mobile interface and an example of the candidate result images including the outcomes of the crowd validation tasks.

The study demonstrates how humans can improve the precision of the im-



	1	2	3	4	5
Results	Yes	Yes	Yes	Yes	Yes

	1	2	3	4	5
Results	Yes	No	Yes	Yes	Yes

	1	2	3	4	5
Results	No	No	No	No	No

	1	2	3	4	5
Results	Yes	No	No	Yes	Yes

Figure 2.16: The CrowdSearch mobile application

age search system. Four different schemes for human validation has been used: *first-response*, *majority(3)*[7], *majority(5)*, and *one-veto* (i.e., complete agreement among validators).

The demonstration used the four different image categories: face, flowers, book covers and buildings.

The results reveal two key observations. First, a considerable improvement in precision is observed whatever the strategy used. All four validation schemes are considerably better than automated search. For face images, even using a single human validation improves precision by 3 times whereas the use of a *majority(5)* scheme improves precision by 5 times. Even for book cover images, *majority(5)* still improves precision by 30%. In fact, the precision using human validators is also considerably better than the top-ranked response from the automatic search engine.

Second, among the four schemes, human validation with *majority(5)* is easily the best performer and consistently provides accuracy greater than 95% for all image categories. *Majority(3)* is a close second, but its precision on face and building images is less than 95%. The *one-veto* scheme also cannot reach 95% precision for face, flower and building images. The use of *first-response* gives the worst precision as it is affected most by human bias and error. Based on the above observation, *majority(5)* is the best validation scheme, for mobile users who care about search precision.

[7]The *majority(N)* rule require each validation task to be duplicated N times, then the final outcome of the task is one which appears the major number of times among the N responses.

Along with providing an implementation of the image search system, the study report a detailed analysis on the system trade-off in terms of: *delay*, *accuracy* and *cost*.
A scheme that optimizes *delay* would post all candidate images to the crowdsourcing system at the same time (i.e., *parallel posting*). While *parallel posting* reduces *delay*, it is expensive in terms of monetary cost. *Parallel posting* is also wasteful since it ignores the fact that images with higher rank are more likely to be better matches than those lower-ranked ones. As a result, if the first candidate image is accurate, the rest of the candidates need not to be posted.
In contrast to *parallel posting*, a scheme that optimizes solely the monetary cost would post tasks serially. A *serial posting* scheme first posts the top-ranked candidate for validation, and waits to see if the majority of validators agree that it is a positive match. If so, the process ends and returns a positive match, otherwise the next candidate is posted, and so on. This scheme uses the least number of tasks to find the first correct match, and thus costs considerably less than the *parallel posting* scheme. However, in cases where top-ranked image is incorrect, *serial posting* incurs much higher delay than *parallel posting*.
CrowdSearch aims to provide a balance between serial and parallel schemes. More precisely, the goal is to minimize monetary cost while ensuring that at least one valid candidate, if present in the ranked list returned from the search engine, is provided to the user within a user specified deadline.

2.2.3 Query expansion and reformulation

A recent study [Harris, 2012] has examined the effects of using students, crowdsourcing, and *YouTube*'s search interface on *user-generated-content* (UGC) searches. Figure 2.17 summarizes the three search approaches followed in the study.
Due to the broad categories used to tag videos on *YouTube* and the inadequacy

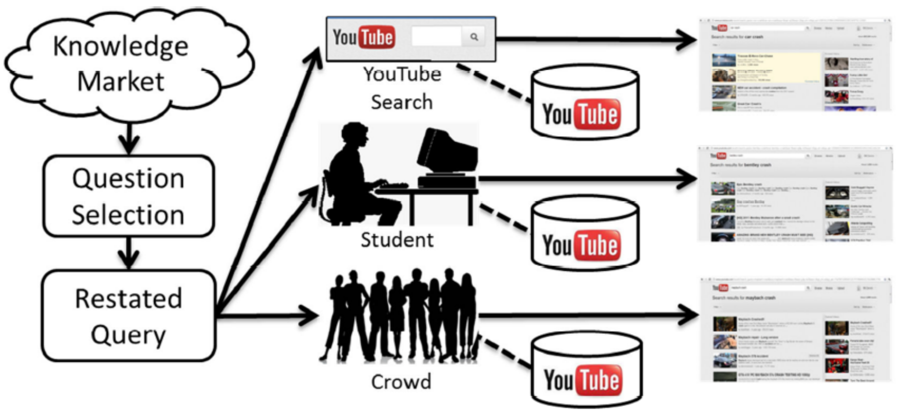


Figure 2.17: The video retrieval process

of many tags in describing the actual video content, many user queries for UGC go unsatisfied. Thus, in order to satisfy their UGC information needs, people often refer to knowledge markets Websites (e.g., *Yahoo! Answers*[8]): as of Jan-

[8]http://answers.yahoo.com

uary 2012, *Yahoo! Answers* had more than 250,000 requests for assistance to locate videos.
The study states that crowdsourcing may provide a viable solution for searching UGC. The crowd introduces diversity of search terms since different members of the crowd will apply different search strategies based on their familiarity with the search topic.
The study compares *YouTube*'s own search interface with a search conducted by students as well as a search approach using crowdsourcing. Results are evaluated using two methods: *mean average precision* (MAP) determined after applying pooling, and a *simple list preference*, where the entire list of videos judged as relevant by each method are compared.
The study experiment was organized as follows:

1. A set of 100 questions were randomly taken from four knowledge market sources (*Yahoo! Answers*, *Answers.com*[9], *Blurtit*[10] and *Allexperts*[11]) containing the terms *find* and *video* and remained either unanswered or partially-answered (i.e., the requestor did not indicate their query had been satisfied).

2. The list was pruned of questions down to 45 by removing those where the requestor's need could not be clearly determined or any candidate videos on *YouTube* could not be found.

3. Then, noisy terms were removed from the original request, to build the so-called *Restated Query*.

4. The requests were then classified in three categories *easy*, *medium* and *difficult*.

5. *Restated Queries* were run using *YouTube*'s own search interface.

6. For the student search method, we asked five university students to perform each search, each student was instructed to provide a list (of up to 40) *YouTube* video links for each *Restated Query*.

7. For the crowdsourced search method, we use the *Amazon Mechanical Turk* platform to list tasks, and provide each worker with *Restated Query* for each question with instructions to return at least 10, but not more than 40, of the most relevant *YouTube* video links.

Results demonstrate that human computation efforts provide better MAP scores than YouTube's own search interface across all categories. The best MAP values are achieved by five students, but if time and cost are to be considered, crowdsourcing achieves additional consideration due to the cost savings it offers over student search.

2.2.4 Games With a Purpose

Games With a Purpose[12](a.k.a. GWAP) come from the observation, that can be easily experienced by everyone, that people around the world spend billions of

[9]http://answers.com
[10]http://www.blurtit.com
[11]http://www.allexperts.com
[12]http://www.gwap.com

hours playing computer games, especially on the Web. As reported in [von Ahn and Dabbish, 2008], more than 200 million hours are spent each day playing computer and video games in the U.S.. The underlying idea of GWAP is to channel these time and effort into useful work. A massive computation can be divided into small tasks, and the tasks can be distributed to the human workers. The entertainment nature of the task is the incentive for humans to participate. *The ESP Game* [von Ahn and Dabbish, 2004], *Peekaboom* [von Ahn et al., 2006], *KissKissBan* [Ho et al., 2009] and *TagATune* [Law et al., 2007] are famous and peculiar examples of GWAPs used to collect tags for multimedia items, such as images or audios.

2.3 CrowdSearcher

Link analysis, the field of research that has shaped Web search technology in the last decade, can be seen as a massive mining of crowd-secured reputation associated with pages. With the exponential increase of social engagement, link analysis is now complemented by other kinds of crowd-generated information, such as multimedia content, recommendations, tweets and tags, and each person can ask for information or advices from dedicated sites. With the growth of online presence, we expect questions to be directly routed to informed crowds. At the same time, many kinds of tasks - either directly used for search or indirectly used for enriching content to make it more searchable - are explicitly crowd-sourced, possibly under the format of games. Many such tasks can be used to craft information (e.g., by naming and tagging data objects and by solving representational ambiguities and conflicts), thereby enhancing the scope of searchable objects. Thus, social engagement is empowering and reshaping the search of Web information.
Crowdsearch is targeted to enabling, promoting and understanding individual and social participation to search [Baeza-Yates et al., 2012].
Crowdsearch uses the crowds as sources for the content processing and information seeking processes; it fills the gap between generalized search systems, which operate upon world-wide information - including facts and recommendations as crawled and indexed by computerized systems – and social systems, capable of interacting with real people, in real time [Baeza-Yates et al., 2012].
Crowdsearching is a brand-new paradigm in Information Retrieval and can be defined as the promotion of individual and social participation to search-based applications and improve the performance of information retrieval algorithms with the calibrated contribution of humans [Bozzon et al., 2012c].

2.3.1 A General Approach for Crowdsourced Multimedia Processing and Querying

In the field of Information Retrieval, *crowdsearching* is usually used in solving problems in which humans out-perform machines. In such approach the results of human computation (e.g., tags, annotations, etc.) replace the output of the automatic tools and algorithms. During the CrowdSearch 2012 workshop at WWW 2012, a different framework for crowdsearching has been proposed.
In the approach proposed by [Bozzon et al., 2012c] humans do not replace automatic feature extractors in the architecture of a MIR system; humans' skills

and efforts are used to improve the performance of such components. In order to do so, it is crucial the selection of the tasks to be performed by the humans executors. As for any other human computation approaches to problem solving, also in the crowdsearching case, a problem has to be mapped into a set of tasks. Then the tasks, according to some criteria (i.e., expected quality of the results, cost of the assignment, etc.), can be assigned either to the crowd or to machine executors.
When a task (i.e., *Crowd Task*) is assigned to the crowd, its modalities of execution need to be defined; this operation is called *Human Task Design*. The *Human Task Design* activity produces the actual design of the *Task Execution GUI*, and the specification of *Task Deployment Criteria*. These criteria can be logically subdivided into two subject areas: *Content Affinity Criteria* (i.e., what topic the task is about) and *Execution Criteria* (i.e., how the task should be executed). The *Execution Criteria* could specify constraints or desired characteristics of task execution including: a time budget for completing the work, a monetary budget for incentivizing or paying workers, bounds on the number of executors or on the number of outputs (e.g., a level of redundancy for output verification) and desired demographic properties (e.g., workers' distribution with respect to geographical position or skill level).

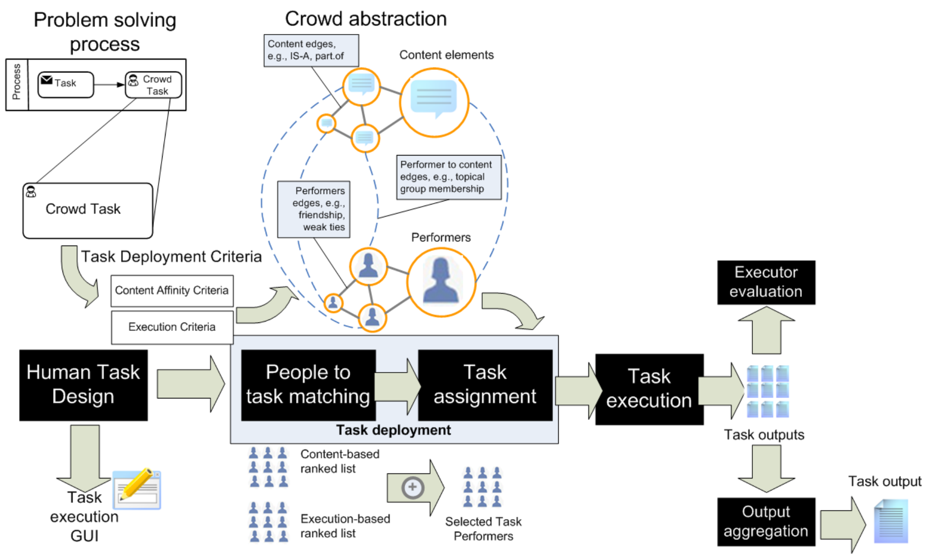


Figure 2.18: The *CrowdSearcher* framework

The framework provides also an abstraction for the crowd and their capabilities. In this abstraction (i.e., the *Crowd Abstraction*), human performers and content elements are represented as nodes connected by edges in a *bi-partite* graph. An edge connecting two performers could denote a friendship, while an edge between two content elements denotes a semantic relationships. Finally, a connection from a performer to a content element may denote an interest.
The subsequent step to the *Human Task Design* is the so-called *Task Deployment*. The goal of Task Deployment is to assign the crowd tasks to the best suited human performers. Thus, this step is composed by two sub-step: *People*

to Task Matching and *Task Assignment*. *People to Task Matching* can be seen as a query matching problem, in which the result is a ranked list of potential candidate workers according to the *Task Deployment Criteria* and the workers' expected suitability in executing the task. Once a set of suitable workers is selected (e.g., the top-k performers), the task is assigned to them. The next step is the *Task Execution*. This step is the actual execution of the *Crowd Task* an its outcome are multiple results. These results are then merged to form the final result of the *Crowd Task*.
The aim of our thesis is to refine this general framework into an actual framework for multimedia search-based application, providing also an actual implementation of the processes and the activities introduced in [Bozzon et al., 2012c].
The paper also explains the design of a logo detection application, as proof-of-concept of the framework. That design is the basis of *logo detection application* demo we designed and implemented in this thesis, that will be discussed in Chapter 5.

2.3.2 A Model-driven Approach for Crowdsourcing Search

People often, when performing a Web search, tend to ask for human help for an opinion on a result or for suggestions about the best query terms to use. For instance, if a user is looking for a good restaurant in Como, it is more likely that he, or she, will trust the opinion of a friend than the review returned by a search engine. This phenomenon, in current Web search system is not considered, hence the user has to search for help independently on social networks or crowdsourcing platforms.
The work presented in [Bozzon et al., 2012b] aims to introduce a model driven approach to the design of Web applications that support crowdsourced search. According to such approach a meta-model for the query task and a model for the user interaction need to defined.

The *Query Task meta-model* is the meta-model to which every query task should conform. The main element in this *Query* submitted by a *User*. The *Query* is defined by a *Question* and by a list of structured information (i.e., relationships), called *CrowdObjects*. The *Question* is the *Query* expressed in the user natural language.
CrowdObjects can be either *Input CrowdObjects* or *Output CrowdObjects*. *Input CrowdObjects* are a set of data attached to the question upon which the responder can apply its response (i.e., list of restaurants); the model of these objects has to defined within a schema. *Input CrowdObjects* can be of different types according to the action that the responder has to perform to answer the question (voting or writing a comment one or more inputs, or adding a new instance of *Output CrowdObject*). *Output CrowdObjects* represent the answers submitted by the crowd.
There may exists relationships between *CrowdObject*. A relation is created when a query is split into sub-queries. These relations can be either *Input-Input* or *Output-Input*.
In an *Input-Input* relation the initial set of inputs is partitioned on several instances of the same query, to reduce the workload of each responder (i.e., a list of 100 restaurants is partitioned in lists of 10 restaurants).
An *Output-Input* relation occurs when the query task is complex, hence it is

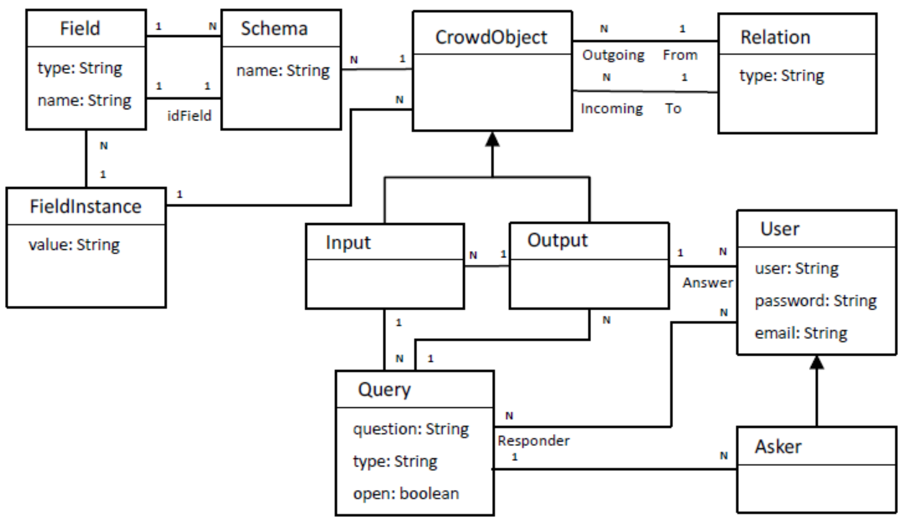


Figure 2.19: The *Query Task meta-model*

divided into a sequence of sub-tasks. Each responder will perform a particular sub-task and his, or her, answers will be the input of the subsequent task.
The user interaction model describes the interface and the navigation aspects of the crowdsearch application. It has to cover three phases of the crowdsearch process:

- the submission of the question by the asker,
- the gathering of the answers from the crowd
- and the analysis of the results to be returned to the asker.

The paper also reports a possible outcome of the user interaction design for the query creation and the query answering on *Facebook*.

2.4 Modeling Search-based Applications

Search-based Applications (SBAs) are data management systems, that have not to be confused with traditional search engines. A search engine is a data management system that deals with the retrieval and the indexing of information items from one or more data sources and allow users to access those items by submitting queries. Indeed, a search-based application is a more complex system, in which search engines, even though being fundamental for the application, are just a part of it. SBAs include further components as, for instance, heterogeneous data source integration, content analysis and user interfaces.
In the next section we will present an overview on the architecture of SBAs.

2.4.1 Search-based Application High-level overview

As well as any IR system, the goal of search-based applications is to satisfy user information needs by searching over the available content collections for information that seems relevant [Middleton and Baeza-Yates, 2007]. To fulfil this, the application is composed by several modules that interact among them. Figure 2.20 provide a coarse overview of the main modules that compose SBAs.

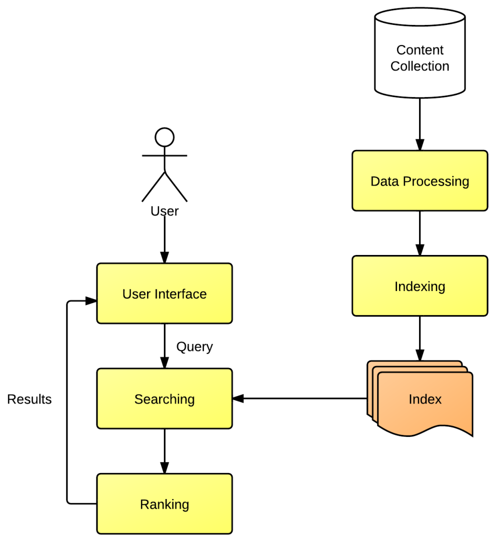


Figure 2.20: The *Search-based Application* coarse processes

This architecture, as discussed in [Middleton and Baeza-Yates, 2007], can be seen as constituted by three main areas: *Indexing*, *Searching* and *Ranking*. The *Indexing* area is in charge of the representation and the organization within an index of the information items, retrieved from the data sources; it should also provide means for a rapid access to the information. The indexing process includes the following tasks:

- the extraction of contents from data repositories (e.g., text documents, images, videos, etc.);
- the analysis of the contents to provide metadata, also called *annotations*, to build a contents' representation;
- the building of an optimized index of the managed information, for efficiently answer to users' queries.

The *Searching* area extracts information from the index which satisfies the user information need.
The *Ranking* area is responsible of sorting the results, according to some heuristics, and to determine which of them better satisfy the user information needs. The user interacts with the application through a user interfaces which provides the means for querying the index and for displaying the results.

2.4.2 Search-based Application Reference Architecture

Performance, *flexibility*, *maintainability*, *reusability*, and *scalability* are non-functional requirements of SBAs. The multi-tier architecture is generally known

as the best suited software architecture to cope those needs.
A SBA is usually characterized by three architectural layers, as depicted in 2.21.

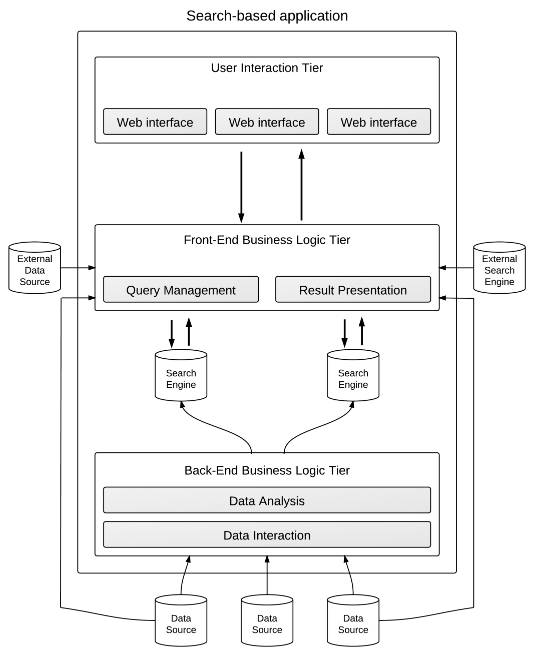


Figure 2.21: The *Search-based Application* architecture

This architecture is very similar to the three-tiers architectures, widely used in Web applications and net-centric information systems. The three components that form the overall architecture are: the *Presentation tier*, the *Front-end tier* and the *Back-end tier*.
The *Presentation tier* is at the top of the SBA architecture and contains all the components that allow users to interact with system for information seeking and social interaction. This tier includes the user interfaces.
The *Front-end tier* is the mid-tier in the architecture, it relates to the business logic of the application. In a SBA, it coordinates the application processes according to the functionalities needed by the *Presentation tier*. Within this tier all the logical and decisions and evaluations are taken. It also manages the movements and the processing of data between the *Presentation tier* and the *Back-end tier*. The business logic can be split in two sub-layers: the *Query Management* and the *Result Presentation*.
Query management layer deals with the orchestration and manipulation of the queries to send to the search components. Queries comes from the *Presentation tier* and are needed to perform the search activity over the application's search engines. A query may need to be transformed (w.r.t. its original format) and analyzed, possibly in the same way as the indexed data; the goal of the *Front-end business logic* layer is to provide a common representation space for both contents and queries; for instance, in a content based information retrieval system, indexed contents are represented as low-level features in a vector space:

in such a case, users can query the system in a query-by-example fashion (e.g., providing a sample image or a music track, etc.); therefore, to compare the user query with the indexed contents, the query itself needs to be transformed to the same vector space as the original content. Additionally, queries may be enriched taking into account contextual information, like the users behavioral, or social profile. Finally, if the search application connects to multiple search engines, queries must be properly partitioned and orchestrated, so to address them to the proper engine.
The *Result Presentation*, indeed, is responsible of the aggregation and manipulation of the results of search operations: results returned by the search engines may need to be fused into a single result-set, transformed and/or enriched. The goal of this layer is to enhance the expressivity of the retrieved information, possibly by adding data coming from other sources. Data sources can be internal to the SBA (e.g., a data repository containing the original contents) or external, like third-party search engines, Web services, etc. *MashUp* applications [Benatallah et al., 2005] are a typical example of result aggregation and manipulation applications. A *MashUp* is an application which uses and composes existing distributed Web services for achieving a complex goal: for instance, a SBA querying an audiovisual search engine may use an external service to enrich the result presentation with information about the location where the video were shot.
The core of a search-based applications are the search engines. In the multi-tier architecture their tasks are in between the *Front-end* and the *Back-end tier*s. First search engines perform the actual query operation on the contents, such task is related to the *Front-end tier* logic. Second, they perform content indexing, managing the creation and the update of indexes, which is are more close to a *Back-end tier* logic. Finally, at the bottom of the architecture, we find the *Back-end business logic tier*. This is the tier that contains all the components that perform the retrieval (also called *Data Interaction*) and the processing (also called *Data Analysis*) of the data managed by the application.
Data interaction components manage all the operations needed to interact with the data sources. Typical examples of such components are crawlers, XML and JSON parsers, database connectors, etc..
Data Analysis components, instead, perform the operations on the content items in order to analyze them, for instance producing annotations.

2.4.3 Reference Architecture Processes

In a search-based application information flows through its layers according to some interaction processes. Figure 2.22 draws an high-level view of the main processes that characterize a SBA.
Within a SBA we can identify three interaction processes:

- *Indexing process*
- *Query and Result Presentation process*
- *User Interaction process*

The *Indexing process* addresses the indexing of the contents retrieved from the external data source; this process includes the actual data retrieval from the

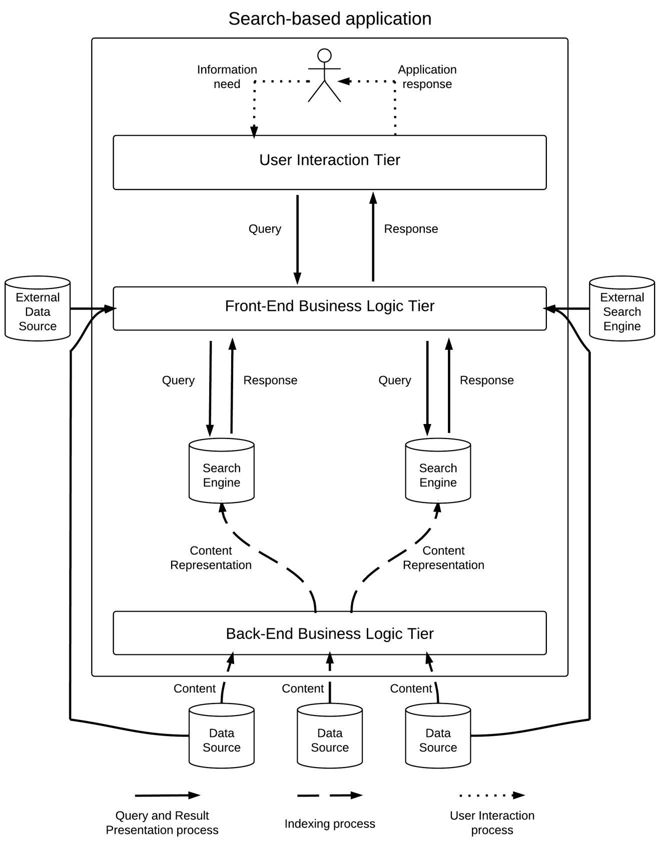


Figure 2.22: The *Search-based Application* processes

data sources, the transformations and the aggregation of the data retrieved and their indexation.
The *Query and Result Presentation* process (also referred as QRP) address the operations related to the query execution, orchestration and result-set composition.
Finally, the *User Interaction process* is related to the way the user interacts with the application.

2.4.4 Design Dimensions

The design of a search-based application is subject to a set of orthogonal dimensions that arise from the domain of the application and affects the design of the application's component and the processing flows. The importance and the prominence of each of the dimension that we are going to list, is highly dependent from the business requirements.

- **Retrieval Policy**
  The data retrieval dimension affect the *Indexing process* and depends on the data source type. Two most common policies are, *pull* and *push. Pull* policy is used when the data source is uncontrolled and is not possible to

track the updates (e.g., the Web), while the *push* policy is more suited for applications that require the data to be continuously refreshed and updated (e.g., news, financial, etc.).

- **Data Homogeneity**
  The data format is another dimension that affects the *Indexing process*. According to the composition of the collection we distinguish between homogeneous data collections (e.g., text document collections, video collections, etc.) and heterogeneous data collections (e.g., collection including both audio and video files). From the data formats in the collections depend the annotators to use in the *Indexing process*.

- **Data Analysis**
  Data Analysis relates to the way information items are analyzed, according to the file type and the content type. We can distinguish two kinds of analysis: *Mono-annotation* data analysis and *Multi-annotation* data analysis. *Mono-annotation* data analysis takes place when a single combination of file type and content type (e.g, the audio track of a video file) is represented by a single annotations set. Conversely, *Multi-Annotation* data analysis provide multiple viewpoints over the same content, in order to produce more valuable annotations.

- **Search Technology**
  The design of a SBA is influenced by the type of the search engines in the architecture (e.g., unstructured, semi-structured, content-based), as well as their number and heterogeneity. The *Search Technology* dimension affects both the *Indexing process* and the *Query and Result Presentation process*.

- **Query Format**
  From the search technology chosen for a SBA directly depends the query format type. The type of queries drives the design of both *User Interaction process* and *QRP process*.
  Textual search engines require queries to be expressed as compounds of textual terms, while content-based search engines expect queries to address the same feature vector space as the indexed contents. Nevertheless, the query type may be inappropriate for users (users cannot express queries in terms of feature vectors), thus the application should provide to users the simplest and the most effective way to pose queries, ignoring the complexity of the underlying search technology. Query formats may be *Mono-modal*, if the application provides a distinct interfaces for each distinct search engine (e.g., Google), or *Multi-modal*, if the application is able to gather, from the same interface, different query terms to different search engines.

- **User Query Interaction**
  Depending on the domain requirements, the *User Interaction process* must be designed in order to support different user interactions styles, like searching, browsing of the indexed collection, push notifications, etc..

- **User Social Interaction**
  The presence of social functionalities influences the design of a SBA and of

all its processes. We can identify three main functions for social interaction support, namely *Clustering*, *Social Relevance*, and *Connections*.
Clustering refers to activities where users collaboratively group together contents in order to provide additional annotations to better understand their properties.
Social Relevance refers to establishing the quality of the contents quality according to user feedbacks. Feedbacks can be provided both as an explicit action (e.g., voting) or as an indirect evaluation derived from user behavior (e.g., the number of times a document has been viewed).
Finally, the *Connections* between users can be exploited in order to enable features like content recommendation.

2.4.5 Conceptual Design of a Search-based Application

In the development of a search-based application the *Conceptual Design* is the core activity. This is macro activity as it involves the main design activities. The first steps in the Conceptual design are the *Process Design* and the *Domain Model Design*, followed by the *Application Design*.
The *Domain Model Design* describes the domain models used by the designed processes. Such models usually include: the *Content model*, the *Usage model*, the *Index model*, the *Query data model*, the *Result data model*, the *User data model* and the *Process data model*.
The *Process Design* defines the base processes of the application, as the *Indexing*, the *Query and Result Presentation* and the *User Interaction* processes.
The *Application Design* is the activity that transforms the functional requirements identified during requirements specification, and the constraints introduced in the *Process Design* activity, into one or more models embodying the needed services for information delivery and data manipulation.

3 Requirements for Human-Enhanced Multimedia Analysis and Search

In this chapter we describe the requirements that a multimedia search-based application (SBA), that benefits of the contribution of human computation tasks, should fulfil. First, we concentrate our attention on the prerequisites that a SBA need to satisfy in order to manage multimedia resources. Then, we will highlight the main challenges that Human Computation introduces in the design of an human-enhanced SBA (hSBA).

3.1 Requirements for Multimedia Search-based Applications

When designing a multimedia search-based application (whether human-enhanced or not) some requirements, such as *execution*, *performance*, *reliability* and *distribution* need to be considered and possibly addressed. Those requirements are generated from the multimedia nature of the data managed from the application; multimedia objects require a different handling w.r.t. classical text document (e.g., video segmentation, feature detection, etc.). In this section we focus on classical SBAs in which tasks are executed by automated components.

Execution In a coarse view, a multimedia SBA is composed by two workflows, one in charge of the retrieval, the processing and the analysis of the multimedia items (i.e., the *Indexing* workflow) and one in charge of the search of the multimedia items stored in the application indexes (i.e., the *Search* workflow).

The design of those workflows is influenced by the types of multimedia items that the system have to manage. The presence of heterogeneous multimedia items (e.g., videos and images at the same time) results in designing different indexing and search workflows according to the item type. For instance, an application that deals with both videos and images has to provide, at least two different processing workflows, which include different analysis operation (e.g., video segmentation, key-frame detection, frame extraction, feature detection, etc.).
Furthermore, we can distinguish the application workflows in two types: *batch* and *triggered* ones. For instance, a video processing workflow that retrieves and processes videos from a collection can run in a batch fashion, while a search workflow is, by definition, triggered by a user query.
The architecture of the SBA should support the simultaneous execution of the several *Indexing* and *Search* workflows. This prerequisite implies that the SBA should act as a *workflow orchestrator* (see Section 6.1).

Analysis performance The particular nature of the multimedia items to be managed (e.g., videos, images, etc.) along with the huge amount of data that these items carry (e.g., the binary file of a video may reach several MB of size), cause the designer to introduce in the architecture components in charge of reducing and transforming such data. These transformations are needed to improve the performances of the overall system, in order to optimize the amount of data to be processed w.r.t. the useful information that can be extracted and queried from them.
For instance, in the multimedia search-based application we present in Chapter 5, each video is processed in order to extract its frames. The frames are the multimedia objects on which we compute local feature descriptors to be used for the image matching. The number of frames within a video depends on the frame rate and on the video length; typically a 2-minutes-long video contains up to 3600 frames (i.e., 30 frames per second). Then, if the application has to process several tens of videos, the number of frame images to be analyzed may grow dramatically. Actually, it can be observed the most of the frame extracted from a video are redundant and they do not contributes to the overall video characterization (e.g., two subsequent frames in a video are very likely to depict the same subjects). Thus, we are just interested in the video frames that denote a sensible variation in the image contents (i.e., the *key-frames*). Key-frame detection reduces the amount of frame to be processed for each video from thousands to tens, corresponding to an impressive boost in the application analysis and indexing performances.

Precision Despite the improvements of the last decades in multimedia analysis and processing, state-of-the-art automatic annotators are not always reliable and may produce unreliable annotations. Those wrong, or low-quality, annotations affects the quality performance of the application. To address and manage such issue, the outcomes of the automatic annotators should also include a value of confidence on the outcomes themselves. According to its value of confidence, and a set of thresholds, an annotation can be treated in different fashions:

- *discarded*: when its confidence is less than the *low-confidence* threshold;

- *validated*: when its value of confidence denotes *uncertainty*;
- *accepted*: when its confidence is greater than the *high-confidence* threshold.

Distribution In order to further improve performances and optimize the hardware resources, the workflows (*Indexing* and *Search* ones) that compose a multimedia SBA application can be executed on several different machines (both physical or virtual ones). For instance, *Indexing* and *Search* workflows can be separated in different machines. A machine executing *Indexing* workflows should provide hardware resources to deals with huge amount of data, conversely a machine that executes *Search* workflows needs to provide fast and optimized means to interact with the application indexes.
Furthermore, we can execute in parallel the single tasks which compose a workflows, distributing them among multiple machines. For instance, we could run multiple instances of the same annotator on multiple machines, in order to annotate multiple resources at the time. This parallelism should result in a speed-up of the indexing performance of the application, but it may introduces trade-offs, especially in case of multimedia resources, due to the need of moving resources across a network from a machine to another (i.e., increment in the bandwidth usage).
The deployment of the workflows on different machines implies that the multimedia items, managed by the application, should be universally referenced. The SBA should provide a service that allows the storage of the multimedia resources, returning a unique reference (i.e., an ID) to the objects. Querying the service with the resource ID, it should be possible to retrieve the multimedia resource from every point in the SBA architecture.
The use of IDs allows also to optimize the exchange of data among the workflows, as it is possible to avoid the exchange of the huge binary files that constitute the multimedia objects.

3.2 Requirements for the Human Enhancement

Thanks to their natural capabilities, humans can be powerful allies for machines in problem solving. We can say that people's and machines' capabilities complement each other. Indeed, humans are good in performing certain tasks, like image and object recognition, while machines are bad at these tasks; conversely, machines are good at certain other tasks, like performing huge numerical calculation, that humans may be not able to perform. This observation lead us to the idea of convey humans' and machines' skills within a hybrid system, for multimedia analysis and search.
As proposed in [Bozzon et al., 2012c] humans' efforts can be used not as a replacement of machines, but as an improvement to the machine work. For instance, crowdsourcing could be used as a validation for the outcomes of automatic components for content analysis (e.g., object detection algorithm, RANSAC matcher, etc.). We call *Human-Enhancement* the addition of Human Computation tasks to a common search-based application.
In the design of a human-enhanced SBA (hSBA) some new challenges arise

w.r.t. a common SBA. In the following paragraphs, we will highlight some of the key ones, also reported in [Franklin et al., 2011].

Performance A great difference between humans and machines is the speed at which they work. A task that a machine can crunch in few milliseconds for a common human being can take several minutes. Such delay may cause performance drawbacks in the search application. The application should address this issue, for instance producing partial result, to be updated just when the outcomes of Human Computation tasks are known.

Asynchrony The time needed for humans to perform a certain task, may be a blocking factor in the application architecture; for instance, a feature extractor component that process a set of human validated images, may have to wait until the validation process is finished. This fact introduces asynchronicity in the information flows of the application, as the execution of some automated tasks may depend on the outcomes of the crowdsourced task, and the time needed to perform the task is not known a-priori. In off-line processes (e.g., the *Indexing process*) the asynchronous fashion of the workflows may affect just the time needed to end the process and thus its performances. Instead, in online processes (e.g., *Query and Result Presentation process* and *User Interaction process*) an asynchronous workflow may ask to change the usual way users interact with the system. For instance in a content-based image retrieval system users can query the system providing a sample image. If the sample image need to be analyzed by humans, users may need to wait for several minutes before the results can be computed and presented. Hence in such scenario, a best suited user interaction could require the application to push the result to the user as soon as they are available (e.g., send an email notification).

Variability Each human being is unique and unrepeatable. In the realm of *crowdsourcing* this means that people may show tremendous variability in performing the same task, both from one individual to another and over time for a particular individual. This differences in human performances are enhanced if results of crowdsourced tasks need to be merged (e.g., the set of annotation for an image). The application should consider this variability and provide a way to manage it (e.g., consider just the outcomes of the quickest human performer vs. collecting results for several performers).

Quality If they cheat or try to sabotage the system, human workers may provide low quality results. The workers may also misunderstand the task directions or simply make mistakes due to personal bias or to a lack of experience in the subject. The application has to take into account that the crowdsourcing outcomes for a task are not gold-plated and, hence, provide some sort of controls on their quality. For instance, the following policy could be applied in the assignment and evaluation of the human computation results: the same task is assigned to different workers and the results are accepted just if there is an agreement between the workers' outcomes (a.k.a. *Output Agreement*).

Task design Crowdsourced tasks have to be presented to workers in human-readable way (i.e., natural language). Unlike programming languages, natural language may be interpreted in several ways, introducing possible ambiguities in the task instructions. The design of the task and the layout of the worker's interface have a direct effect on the speed and the accuracy with which people complete the tasks.

Task affinity Human workers are not fungible as CPUs. Some workers may be more suited in performing certain tasks than others according to their skills or their culture; for instance, in a logo detection application which require humans to provide logo instance images, a worker from the U.S. may be more skilled in providing good logos from an American detergent brand than a worker from Europe.

Workers recruitment In the field of Human Computation, the term *crowdsourcing* implies that the tasks are executed by "an undefined and large group of people" [Howe, 2006]. Thus, one challenge in the design of a hSBA is where to recruit workers. *Amazon's Mechanical Turk* is the online market that may guarantee the set of workers needed to fulfil the crowdsourced task of the application, under monetary payment. Another option is to publish the tasks on social networks, such as *Facebook*, where a huge crowd is online everyday. On social networks the spread and the assignment of the tasks depends on the friendships links associated to the account owned by the SBA and on further shares from the workers themselves in their communities.

Workers motivation As well as the recruitment, motivation is one of the main challenges in human computation systems. Since the computations usually involve small unit tasks that do no benefit the contributors, they will only participate if motivated. One of the most effective motivation factors is the monetary payment (e.g., *Amazon's Mechanical Turk*, *ChaCha*), but there also other ways to recruit workers, as enjoyment (e.g., GWAPs) or reputation (i.e., public recognition).

In the design of our multimedia hSBA *performance*, *asynchrony* and *variability* dimensions force us to think the application's architecture in order to avoid, or at least reduce, stalls in the workflows (e.g., wait until the outcomes of crowdsourced task are known), that may have a dramatic impact on the overall system performances.
Quality dimension, along with *Task design* and *Task affinity* ones, asks us to adopt a framework able to support the design, the deployment and the result aggregation of the crowdsourced tasks.
Workers recruitment and *Workers motivation* dimensions highlight the need of interaction with crowdsourcing platforms and social networks, thus our multimedia search-based application should provide means to interact with them.

4 A Conceptual Framework for Human-Enhanced Multimedia Analysis and Search

In this chapter we introduce a framework for multimedia search-based applications (SBAs) in which Human Computation is used to improve the quality performances of automatic components.
In Section 4.1, we introduce a methodology in order to drive the development of a human-enhanced SBA (hSBA) starting from high-level models until its actual implementation.
The components that will compose the application are founded on a data model, which describes the logical structure of the data handled by the application.
The data model, we introduce in Section 4.2, is composed by three parts: the *Content model*, the *Human Enhancement model* and the *Action model*.
First, the *Content model* provides the concepts in order to characterize the multimedia items (e.g., images, videos, etc.) that the application is managing and the annotations that describe these information items.
Second, the *Human Enhancement model* introduces Human Computation in the conceptual model, describing the tasks and the roles the humans can play within the application.
Third, the *Action model* provides the mean to keep track of the events that occur in the application life-cycle. Such events may be related either to actions performed by automatic components (i.e., *Automatic Actions*) or by humans (i.e., *Human Actions*).
Finally, in Section 4.4, we will focus on the presentation of a suitable model to describe the processes involved in a hSBA.
To give a particular emphasis on the human enhancement of multimedia search-

based application, we will present the *Human Enhancement* data model and the *Human Enhancement* process in a separate section.

4.1 Development Methodology

In the development process of our framework, illustrated in Figure 4.1, the design of data models (i.e., *Domain Models*) is one of the main activities. The *Domain Model Design* activity needs to be properly performed in order to provide a high-level representation of the contents to tackle the problem of multimedia analysis and search; a problem that is made more difficult due to the presence in the application architecture of both automatic and human tasks.
Parallel to the *Domain Model Design* activity, in the *Process Design* activity, we need to identify and design the processes that composes our search-based application.
Both *Domain Model Design* and *Process Design* activities are influenced by a third "cross-activity": the *Human Enhancement Design*, which takes into account the presence of human computation tasks and humans in the loop.
Finally, during the *Application Design* activity we materialize the data models in actual data structures and the processes into actual workflows, tasks, components and data structures throughout a development process.

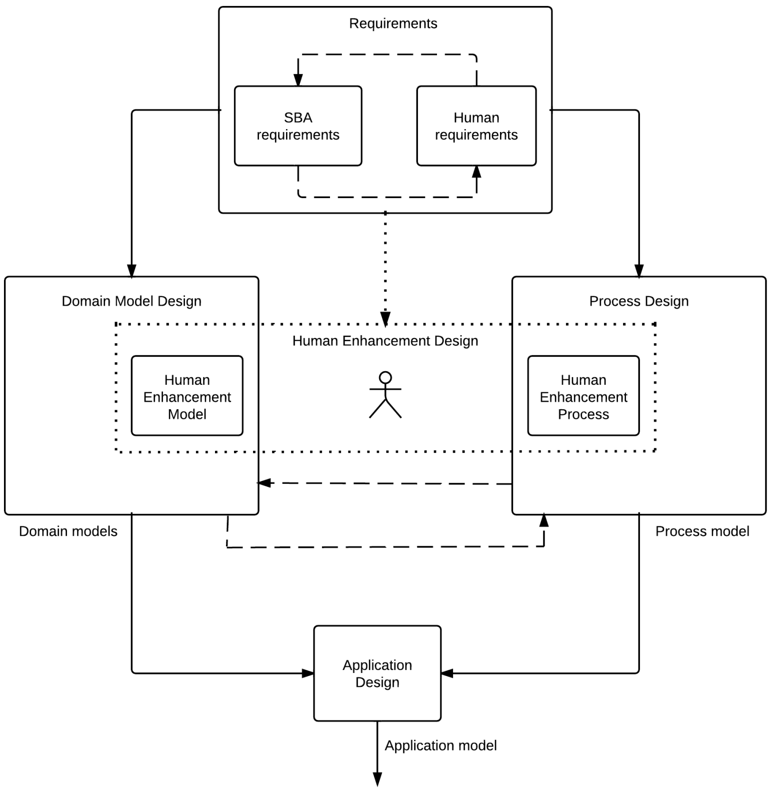


Figure 4.1: The development activities

4.1.1 The Domain Model Design

In the *Domain Model Design* activity we organize the information objects, handled by our multimedia search-based application, into a conceptual data model. The design of the *Domain Models* is influenced by the application requirements (see Chapter 3) and, at the same time, by the *Process Design* activity.
For our human-enhanced multimedia search-based application (hSBA) we identify four models: the *Content model*, the *Human Enhancement model*, the *Action mode*l and the *Content Processing model.*

Content model The *Content model* describes the multimedia items handled by the application, the relationships that may occur between the multimedia objects and, as we deal with analysis process, the produced annotations (e.g., low-level features).

Human Enhancement model See Section 4.1.2

Action model The *Action model* is used to describe the actions (we will further refer to them as *events*) that are performed during the processing of multimedia objects.

Content Processing model The *Content Processing model* describes the processing on the multimedia objects that takes place within the application, the application components (i.e., annotators) and the workflows (i.e., pipelines) that execute the actions presented in the *Action Model.*

4.1.2 The Human Enhancement Design

The *Human Enhancement Design* activity introduces the human factor within the design of a search-based application. It is driven by the requirements that the presence of humans in the loop asks to be satisfied (i.e., *Human requirements*) and influences the other activities of the design phase.
This activity influences the *Domain Model Design* activity through the definition of the *Human Enhancement model.*

Human Enhancement model The *Human Enhancement model* addresses the problem of the need of human computation tasks to process the outcomes produced by automatic components (e.g., enrichment, validation, etc.). The model introduces a format to describe this need, the tasks to be crowdsourced and the outcomes produced by human performers.

The *Process Design* is influenced through the design of the *Human Enhancement process.*

Human Enhancement process The *Human Enhancement process* is the process involved in the management and the resolution of the uncertainty that may arise during the content processing tasks. Automatic components may produce inadequate, unreliable, inconsistent and even missing representations of the content items. Thus, prior to their definitive indexation, the uncertain representations are processed by human performers (i.e., the crowd). The results collected from the human computation tasks are used in the search-based application in order to enrich, validate or correct the indexed content representations. The main tasks to be performed during the *Human Enhancement process* are:

- the design of the tasks to be executed by the human performers, in order to fulfil the content processing tasks;
- the allocation of the task to a set of human performers;
- the support to the execution of the crowdsourced tasks on a proper platform (e.g., *Amazon's Mechanical Turk*, *Facebook*, etc.);
- the aggregation of the results, produced by each crowdsourced task which relates to a specific processing task, into a single outcome.

This process is partially performed by the application in the *Back-end business logic tier*, while the majority of its tasks are executed by an external framework (see Section 2.3), which enables the interaction with human computation and crowdsourcing platforms.

4.1.3 The Process Design

The *Process Design* activity provides a high-level representation for the processes supported by the search-based application (SBA).
In our case, we describe the workflows and the activities of a multimedia human-enhanced SBA (hSBA) through UML activity diagrams.
During *Process Design* activity we should take into account the application requirements and the data model produced in the *Domain Model Design* activity. Indeed, the concepts introduced in the data models are the data objects managed by the designed processes.
In our framework we identify three main coarse processes (as depicted in Figure 4.2), that we will present in the following paragraphs.

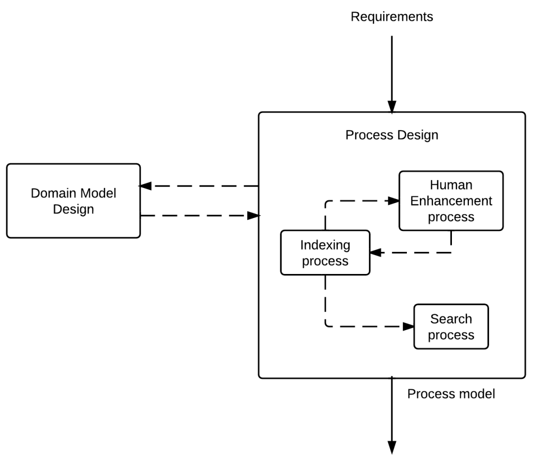


Figure 4.2: The *Process Design* activity

Indexing process The *Indexing process* is the set of activities that the application performs in order to achieve the indexation of the information items consumed. According to the reference SBA architecture introduced in Section 2.4.2, the *Indexing process* is executed by the components belonging to the *Back-end business logic tier.*
The main tasks to be performed are:

- the retrieval, the validation and the registration of the information items from the data sources. Such task is usually executed by the components belonging to the *Data Interaction* layer (i.e., crawlers);
- the analysis of retrieved content items in order to produce a representation suitable to their indexation. Analysis operations are performed by components belonging to the *Data Analysis* layer;
- the construction of an index structure by taking the representation of content produced in the previous task and transforming it into the structure of search engines' indexes.

Human Enhancement process See Section 4.1.2.

Search process The *Search process* manages the information exchange between the search engines, the data sources and the user interfaces. In our framework the *Search process* embodies the activities that in the reference SBA architecture processes are performed during the *Query and Result Presentation process* presented in Section 2.4.3. Such activities includes:

- support the formulation of user queries, by exposing a *query interface.* A *query interface* is an abstraction of the languages offered by the different application's search engines;
- enrich the queries with contextual information in order to improve the retrieval performance of the system;
- translate the queries posed by the user into the search engine's internal format, and orchestrating their execution;
- fuse the results coming from the different search engines;
- enhance the retrieved information with data coming from other sources, both internal to the application (e.g., the repository containing the original contents) or external (e.g., the Web).

The *Search process* is performed by the components belonging to the *Front-end business logic tier.*

4.1.4 The Application Design

The *Application Design* activity is a domain-specific task that allow us to materialize the requirements, the data models and the processes identified during the *Process Design* activity into actual processing workflows and components that will compose the multimedia human-enhanced search-based application (hSBA). Figure 4.3 draws the sub-activities performed during the *Application Design* activity.

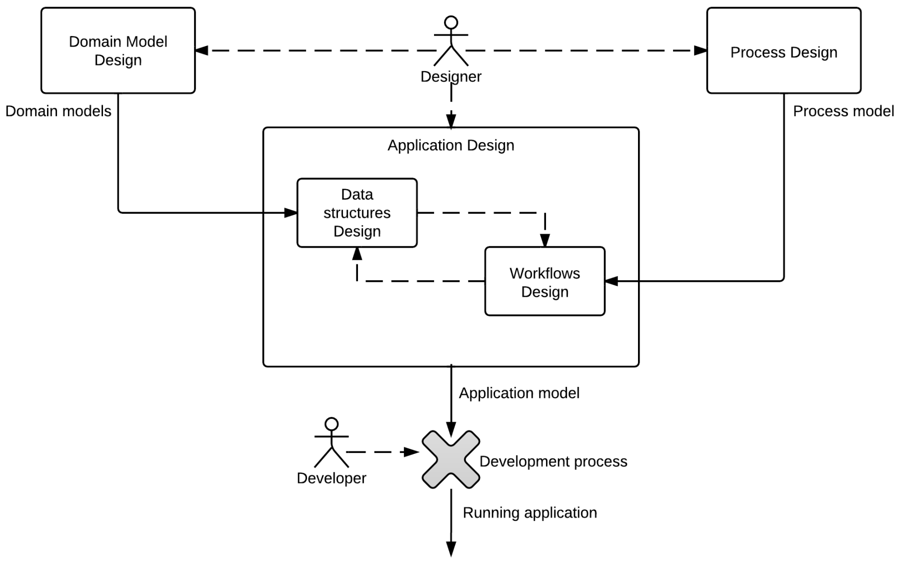


Figure 4.3: The *Application Design* activity

From *Domain Models* to data structures The representation of contents provided by the *Content model* acts as a reference for the design of the actual data structures (e.g., databases, records, etc.) handled and processed in the application. As soon as a content item is registered into the application, a representation of it is produced according to the attributes identified in the data models and the underlying technologies of the working application (e.g., a workflow orchestrator engine). In addition, the *Content Model* reveals the annotations that need to be produced during the processing and the analysis of the content items, according to the application requirements. These information lead us to the design of the inputs and the outputs of the single processing components (e.g., video segmenter, key-frame detector, etc.).
The *Action model* identifies: *(i)* the set of actions (both automated and human ones) that compose the application, *(ii)* the contents they consume and *(iii)* the annotations they produce. In the working application an *action* is materialized into a set of operations performed by a software component or by a human worker.
The *Content Processing model* highlights *(i)* which component is in charge of a certain action and *(ii)* how the single components (or human performers) have to interact each other (e.g., execution order) in order to produce the overall result of the content processing.
Thank to the *Human Enhancement model* we identify *(i)* the tasks to be crowdsourced as well as *(ii)* the content items and the annotations that need to be processed by human workers and *(iii)* design the format of the annotations to be produced during the crowdsourced tasks.
Both requirements and *Content model* drive us to the design of the application indexes. The design of the application indexes should take into account the required querying fashion (e.g., text-based, content-based) and the technology used to materialize the indexes (e.g., Solr, Lucene, etc.).

From *Process Design* to workflows The three reference processes (*Indexing process*, *Human Enhancement process* and *Search process*) that we design during the *Process Design* activity enable us to develop the workflows that the actual search-based application (SBA) manages and executes. The activity diagrams produced during the design phase can be translated in executable representations, such as BPEL workflow specifications. Indeed, in its usual implementation, a SBA acts as a workflow orchestration engines.
According to the single activities identified in the *Process model* and the data structures materialized from the *Domain Models*, we develop the automatic software components (or the software components which support Human Computation) needed to achieve the goal of each process. For instance, in a multimedia SBA, the analysis activities identified during the *Process Design* may result in the development of software components for the detection of key-frames or the temporal segmentation of the videos.
Once the components have been developed, the reference processes give us the guidelines on how to place them within workflow and how each components has to interact with the subsequent one(s).

4.2 The Data Model

4.2.1 The Content Data Model

The *Content model* aims to provide the minimum set of concepts and relationships to address the requirements of a multimedia human-enhanced search-based application (hSBA). The contents managed by our application can be described in terms of:

- *Content Objects*, the abstract representations of the multimedia objects managed by the application (i.e., the *Content Object*) and of the relationships that may occur between multimedia objects (e.g., a frame as part of a video stream).
- *Content Descriptions*, which represent the metadata of each multimedia object.
- *Entities*, which are the representations of objects in the real-world.

4.2.1.1 Content Objects

The main element in the model is the *Content Object*; it is an abstract entity that represents a multimedia information item that can be accessed through a storage system (e.g., a video, a picture, a document, etc.). For the scope of this thesis, the following data model is shaped on audiovisual *Content Objects*, such as *Audio Objects* (e.g., music tracks), *Video Objects* and *Image Objects*.
Each *Content Object* is characterized by the following attributes:

- *ID*: the unique identifier of the multimedia object; it is always required.
- *Media Locator*: the string path that universally identifies the location of *Content Object* in the application (i.e., its URI).

- *Descriptions*: this attribute refers to the set of annotations that describes the *Content Object* (see Section 4.2.1.2).
- *Provider*: the *Content Provider* which owns the Content Object. For instance, a Web provider like *YouTube* and *Google* or a *System Component* (e.g., key-frame detector component).
- *Collections*: this attribute denote the *Content Collection* to which a *Content Objects* belongs. A Content Collection is a meaningful set of *Content Objects* (e.g., the set of match highlights video for a football team in the 2011-2012 season) and it is identified by an ID and a name.
- *MimeType*: specifies the actual file format and encoding of the multimedia object (e.g., MPEG4, JPEG, etc.).

There may exists relationships between *Content Objects*; for instance, the *Image Object* that represents a frame of a football match video is related to the *Video Object* representing the video itself. In our model, relationships are materialized in *Content Relationship* objects.
Relationships may either exist from before the management of the information object within the application or be generated by the application itself. An example of this latter relationship is the one that relates a video to its key-frames (i.e., *Image Objects*) detected during the video processing stage.
Content Relationships are denoted by a type attribute. Here a list of some possible relationships types:

- *Derived Object*: the relationship between the video and its key-frames is an example of this relationship type.
- *Version Of*: when two *Content Objects* represent alternative versions of the same physical object. For instance, the relationships between a video and a down-sampled version of the same video.
- *Duplicate Of*: the relationship between a *Content Object* and its duplicate.
- *Perceptual Duplicate Of*: the relationship between a *Content Object* and its near duplicate.
- *Similar Content Of*: when two *Content Objects* contain instances of the same piece of information (e.g., a real-world object, a monument, a face, etc.). For instance, this relationship exists between two images that both contain a logo for the brand *Coca Cola*. Unlike other relationships, *Similar Content Of* relationships may be characterized by a set of annotations, as any *Content Object*; for instance, the level of similarity between the images low-level features could be attached to the *Similar Content Of* relationship existing between the two images of the previous example.

Figure 4.4 depicts the *Content model* taxonomy introduced so far.

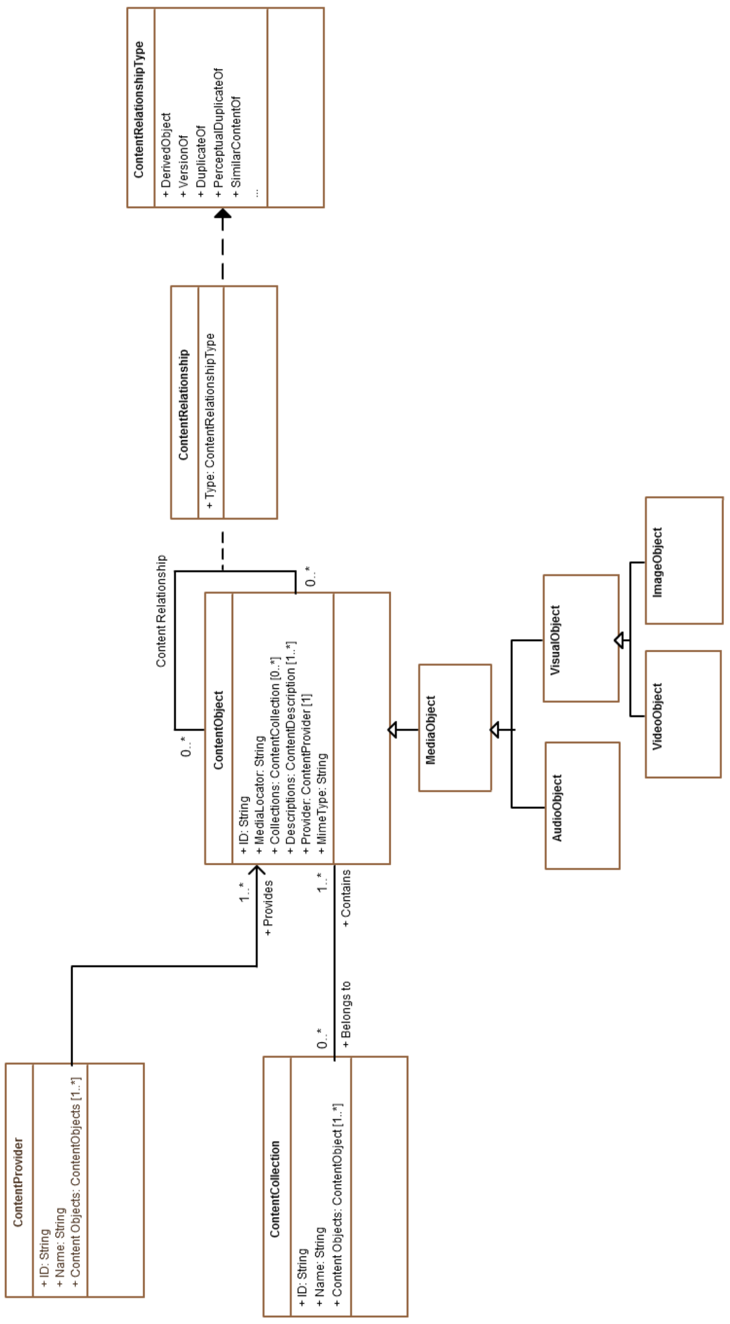


Figure 4.4: The *Content model* taxonomy

4.2.1.2 Content Descriptions

Content Descriptions describe any *Content Object*; they embed information about automatic and manual annotations and about the decomposition of the multimedia objects in *Media Segments*. A *Content Description* is characterized by an ID, its *Annotations*, the *Media Segments* and the related *Content Object*. *Annotations* express metadata that describe a *Content Description* or a *Media Segment Descriptor*. An *Annotation* is created by an *Annotation Action* (i.e., the execution of an *Annotator Component*). Multiple annotations can be created for the same action (e.g., the image tags generated from a GWAP). An *Annotation* is characterized by an *ID*, the activity that generated it (automatic or manual) and the timestamp in which the annotation has been generated. Each annotation has a degree of confidence, called *Annotation Confidence*, that highlights its consistency; such confidence may result from an automatic annotators or an human annotators (e.g., a face detection component may return a value of 0.7).
From the *Annotation* class derives some sub-classes:

- *Text Annotation*, an annotation that contains textual values in a given language (e.g., user comments, tags, etc.).
- *Numeric Annotation*: an annotation that contains numerical values (e.g., frame number, the milliseconds elapsed, etc.).
- *Boolean Annotation*
- *Low-level Features*, an annotation that contains array(s) of numerical values, which, for instance, may represent the result of a numerical analysis on the *Content Object* (color histograms, SIFT descriptors, etc.).
- *Entities*, see Section 4.1.3.
- *Media Segments* are the result of a segmentation process (e.g., temporal segmentation, spatial segmentation, etc.). Each *Media Segment* has its own *ID*, a *Title*, the *Segmentation Criteria* (e.g., temporal, spatial) and a set of annotations. Within the *Content model*, *Media Segments* are described by the *Media Segment Descriptors*.

Figure 4.5 draws the model described so far.

4.2.1.3 Entities

Entities are a specialization of *Annotations*. An *Entity* is an object of the real world which is important to be denoted with a name (e.g., the city of Como). Entities are used to annotate a multimedia object with the place, the person or the event it refers to.
We distinguish entities into two types:

- Basic types:
  - *Location*, an entity for which the spatial dimension is relevant. It refers to spatial objects, real entities occupying regions of space (e.g., regions, cities, boundaries, parcels of land, water bodies, roads, buildings, bridges, etc.).

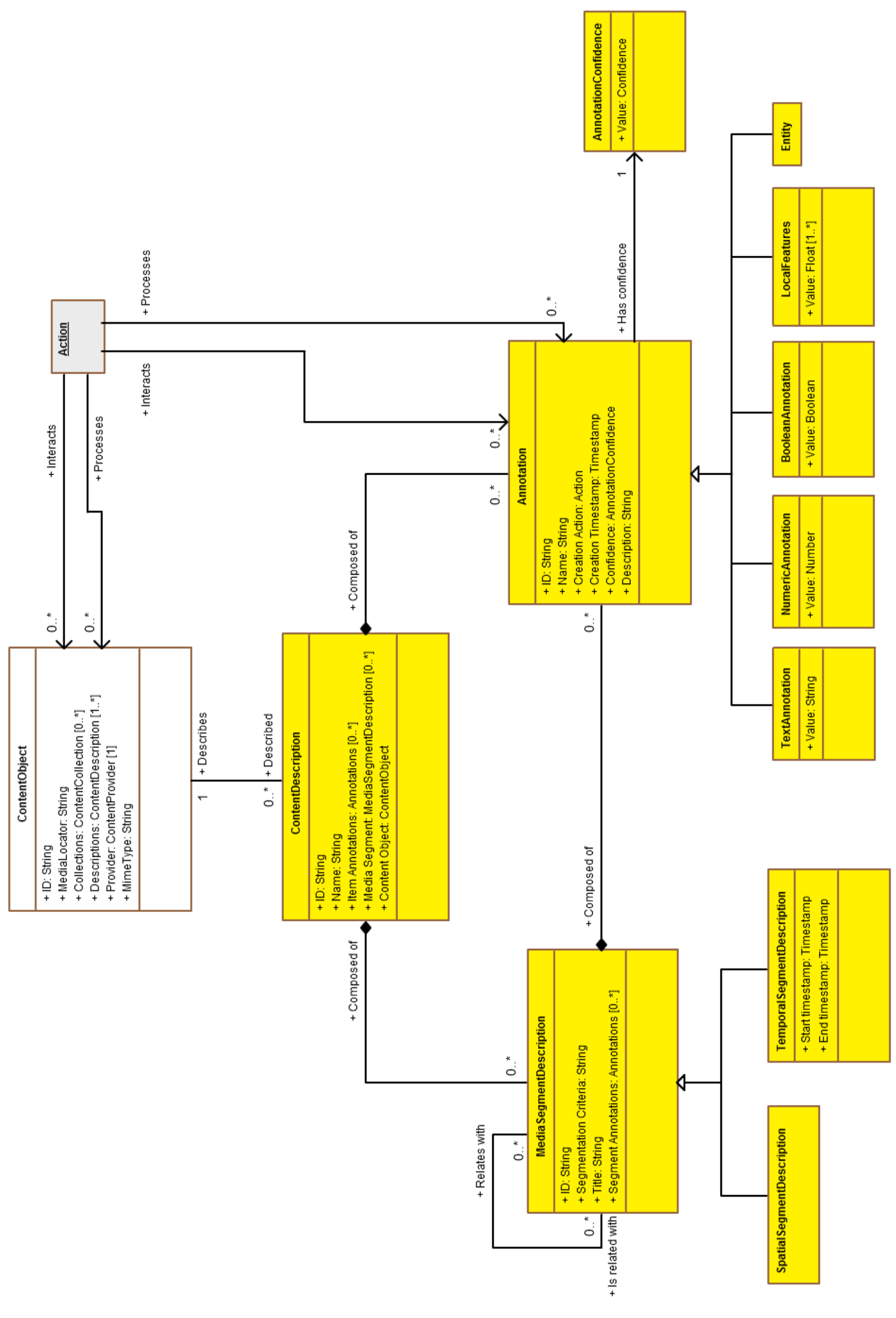


Figure 4.5: The *Content Description* model

  - *Organization*, corporations, agencies, and other groups of people defined by an established organizational structure are all examples of organizations.
  - *Event*, something that happens at a given location and time, for example a conference, a party or a battle.
  - *Person*, a human being, for instance Barack Obama, the president of the USA.

- Media types:
  - *File*, it corresponds to a computer file and is used for storage purposes (e.g., the file containing the SIFT descriptors of an image).
  - *Image file*, a computer file used to store a graphical object (e.g., a photo or a scanned document).
  - *Video file*, a computer file used to store recordings constituted by a sequence of images and audio.

Entities are an open schema ad can be extended with new entity types.

4.2.2 The Action Model

Within our application, an *Action* corresponds to an event that involves the interaction with, the processing, or the creation of, *Content Objects* and *Annotations.* we identify two types of actions:

- *Automatic Actions*, if they are performed by software components (e.g., temporal video segmentation software, object recognition software, etc.).
- *Human Actions*, if they are performed by Performers on a Conflict Resolution Platform (e.g., social networks).

As depicted in Figure 4.6, *Human Actions* are modeled as *Tasks* on the *Conflict Resolution Platform* (see Section 4.3).

4.2.3 The Content Processing Model

In the implementation of our multimedia human-enhanced search-based application (hSBA), described in Chapter 6, all the content analysis components are executed within a framework for the management of unstructured information (i.e., *SMILA*). In the *SMILA*[13] approach the processes are executed within *Pipelines*, which orchestrate the execution of *System Components. System Components* include also the *Annotations Components. System Components* may act as *Content Provider* for *Content Objects* generated within the search application (e.g., the key-frames extracted from a video). The *Content Processing model*, depicted in Figure 4.7 aims to give a description of the processing on the contents that takes place within the application.

[13]http://www.eclipse.org/smila/

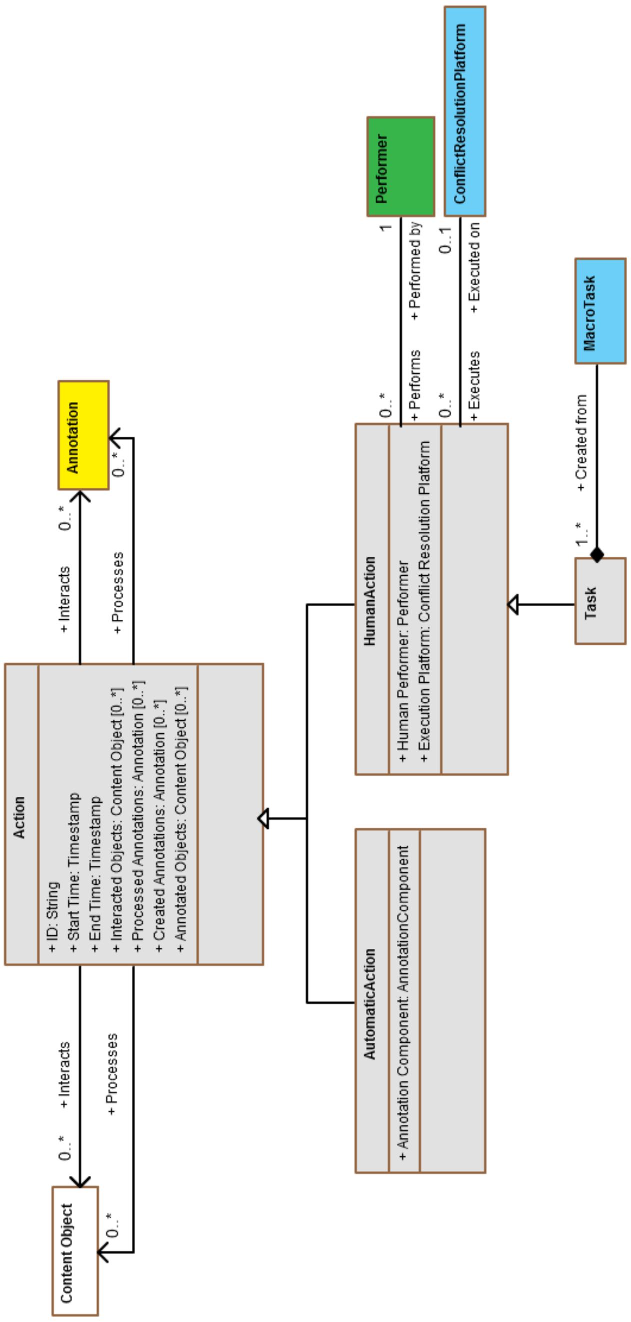

Action
+ ID: String
+ Start Time: Timestamp
+ End Time: Timestamp
+ Interacted Objects: Content Object [0..*]
+ Processed Annotations: Annotation [0..*]
+ Created Annotations: Annotation [0..*]
+ Annotated Objects: Content Object [0..*]
Annotation
Content Object
+ Interacts
+ Processes
0..*
HumanAction
+ Human Performer: Performer
+ Execution Platform: Conflict Resolution Platform
AutomaticAction
+ Annotation Component: AnnotationComponent
Performer
ConflictResolutionPlatform
+ Performs
+ Performed by
+ Executes
+ Executed on
1
0..1
MacroTask
Task
+ Created from
1..*

Figure 4.6: The Action model

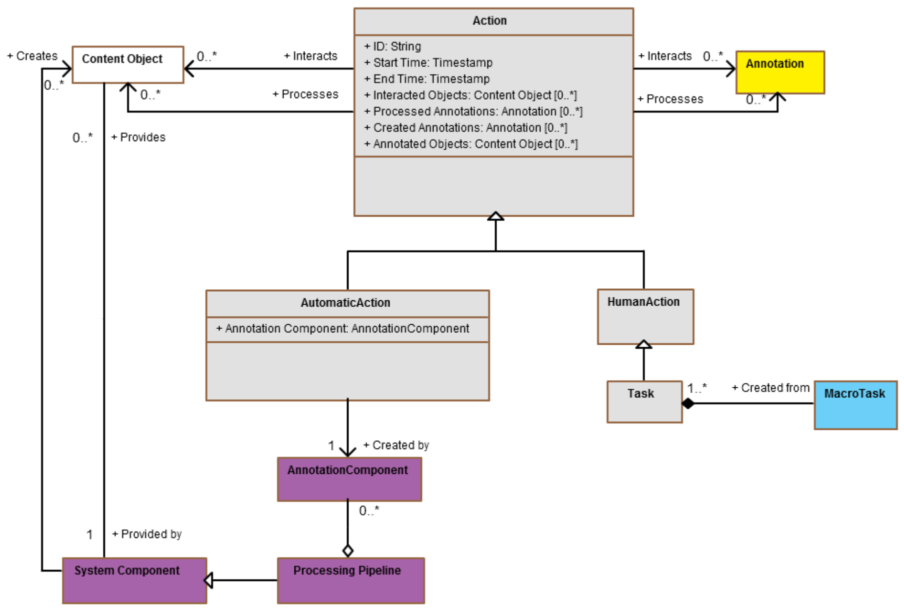


Figure 4.7: The *Content Processing model*

4.3 The Human Enhancement Model

In this section we focus our attention on the enhancement that Human Computation may introduce in a multimedia search-based application (SBA).
The outcomes of the software analysis components are not always reliable and this fact has an impact on the overall quality performances of the application. Especially, when dealing with multimedia contents, software components are more subjected to errors, due to the complexity of the analysis to be performed. Conversely, some multimedia analysis tasks are trivial for humans (e.g., object recognition), hence humans and crowds can be precious allies for software components in order to achieve an higher quality in the results produced.
First, we will provide a suitable data model (i.e., the *Human Enhancement Data model*) to describe the items that humans performers have to process. Then, we will introduce a novel process (i.e., the *Human Enhancement process*) that characterizes human-enhanced SBA w.r.t. to classical SBAs.

4.3.1 The Human Enhancement Data Model

In Section 4.2.1 we presented a data model to describe the multimedia contents managed by our multimedia human-enhanced search-based application. Humans play a central role in our application as they are involved so to improve the quality of content analysis and querying, beyond the limitations of the state-of-the-art content analysis components.
We will refer to the activities that include human computation operations as *Conflict Resolution Tasks*. The name *Conflict Resolution* highlights the usage of humans' capabilities to solve *conflicts* that might arise during the analysis of multimedia contents.

During the analysis of a *Content Object* (i.e., *Annotation Actions*) *conflicts* may arise when there are lacks or contradictions in the object's annotations:

- *Missing Annotation*: the performer of the analysis (either an automatic component or even a human worker) is not able to find a suitable *Annotation* for the analyzed content.

- *Uncertain Annotation*: the *Annotation Confidence* falls in an confidence interval that cause the *Annotation* to be considered uncertain (e.g., the result of a matching between two images).

- *Inconsistent Annotation*: two different annotators produced conflicting annotations on the same *Content Object.*

A *Conflict* is characterized by an *ID*, by either a set of *Annotations* (i.e., the *Conflictual Annotations*) or a set of *Content Objects* (i.e., *Conflictual Objects*) that generated it, and the *Conflict Resolution Task* produced in order to solve it.
When a *conflict* occurs a *Conflict Resolution Task* is triggered. The *Conflict Resolution Task* (e.g., comparing two pictures) is executed on one or more *Conflict Resolution Platforms* (i.e., social network or a human computation platform) and assigned to a human performer.
In case of complex computations a *Conflict Resolution Task* can be organized in a *Macro Task*, which represents a workflow composed by atomic tasks (i.e., *Micro Tasks*), in order to achieve an high-level goal. A *Micro Task* may consist either of a preference task or data manipulation task that typically does not require any particular skill to be performed. Preference tasks correspond to social interactions such as like, dislike, comment or tag, while data manipulation tasks refer to simple human computation activities such as create, order, complete, find or cluster. For instance, the optimal bounding boxes detection *Macro Task* may consist of an object recognition task, that identifies the position of the objects inside an image, and of a bounding box used to re-shape tasks to refine the bounding boxes around the objects.
A *Performer* is a *Person* (see Section 4.2.1.3) that consumes or produces resources when he/she is requested to solve some computation tasks to resolve a *conflict.* The human worker performs *Conflict Resolution Tasks* on a *Conflict Resolution Platform.* A *Performer* can be subscribed to one or more *Conflict Resolution Platforms* and its profiles on each platform are described by the *Conflict Resolution Platform Membership* relationship.
Figure 4.8 illustrates the *Conflict Resolution model.*

4.3.2 The Human Enhancement Process

The *Human Enhancement process* is in charge of managing the *Conflict Resolution Task*, design a set of suitable *Crowd Tasks*, assign the tasks to the human performers and merge the results.
The *Human Enhancement process* specializes and embodies the general framework presented in [Bozzon et al., 2012c]. The process is composed by a set of activities:

- the *Human Task Design*,

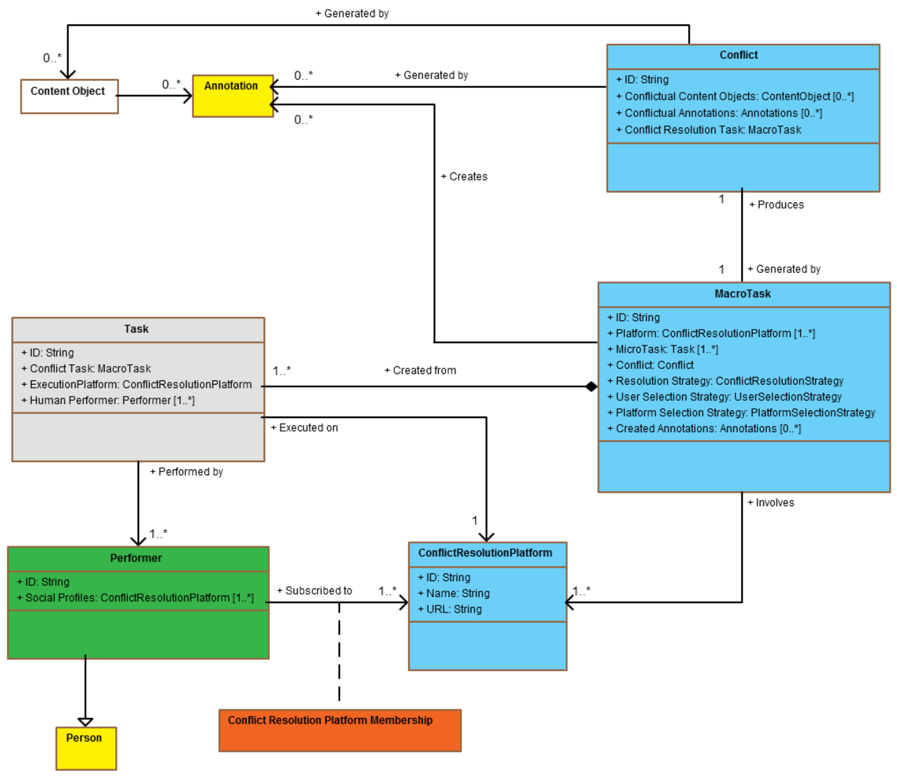

+ Generated by
0..*
Content Object
0..*
Annotation
0..*
+ Generated by
0..*
Conflict
+ ID: String
+ Conflictual Content Objects: ContentObject [0..*]
+ Conflictual Annotations: Annotations [0..*]
+ Conflict Resolution Task: MacroTask
+ Creates
1
+ Produces
1
+ Generated by
MacroTask
+ ID: String
+ Platform: ConflictResolutionPlatform [1..*]
+ MicroTask: Task [1..*]
+ Conflict: Conflict
+ Resolution Strategy: ConflictResolutionStrategy
+ User Selection Strategy: UserSelectionStrategy
+ Platform Selection Strategy: PlatformSelectionStrategy
+ Created Annotations: Annotations [0..*]
Task
+ ID: String
+ Conflict Task: MacroTask
+ ExecutionPlatform: ConflictResolutionPlatform
+ Human Performer: Performer [1..*]
1..*
+ Created from
+ Executed on
+ Performed by
1..*
1
+ Involves
Performer
+ ID: String
+ Social Profiles: ConflictResolutionPlatform [1..*]
+ Subscribed to
1..*
ConflictResolutionPlatform
+ ID: String
+ Name: String
+ URL: String
1..*
Person
Conflict Resolution Platform Membership

Figure 4.8: The *Conflict Resolution model*

- the *Task Deployment*,
- the *Task Execution*
- and the *Output Aggregation*

Figure 4.9 draws the set of activities that forms the *Human Enhancement* process.

4.3.2.1 Human Task Design

Once a *Conflict Resolution Task* is defined, the *Human Task Design* activity is in charge of designing the *Crowd Tasks* (a.k.a. *Macro Tasks*) needed to achieve the resolution of the conflict and the modalities for their execution on the *Conflict Resolution Platform* (i.e., social networks, crowdsourcing platforms). For each created *Crowd Task*, the *Human Task Design* activity builds a *Task Execution GUI*; the GUI and the user interaction design depend on the *Conflict Resolution Platform* (e.g., post on a *Facebook* profile, task published on Amazon's Mechanical Turk, etc.).
Prior to the *Task Deployment* activity, at this stage the *Task Deployment Criteria* are defined. These criteria include the *Content Affinity Criteria* and the *Execution Criteria*. The former ones relates to the topic of the task (i.e., object

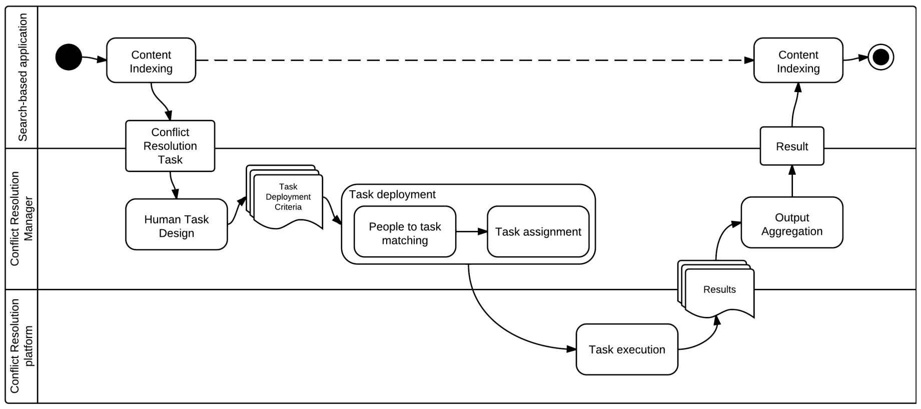


Figure 4.9: The *Human Enhancement process*

recognition) and can be used in the next stages, when defining the *User Selection Strategy* to select the human performers to assign to a task, according to their capabilities and tasks preferences. Indeed, the *Execution Criteria* comprise the *Platform Selection Strategy* (i.e., the methods to choose the best resolution platform on which send the conflict for the resolution) and the *Conflict Resolution Strategy.* the latter strategy dictates the characteristics of the execution of the crowd task, like *(i)* the budget, both in terms of time and money, *(ii)* the desired performers profiles (e.g., language, culture, skills, etc.), *(iii)* the number of performers involved, *(iv)* the result aggregation method (e.g., majority vote) and *(v)* the number of outputs to be returned (e.g., for validation purposes).

4.3.2.2 Task Deployment

This activity is in charge of the assignment to the human performers of the *Crowd Tasks* defined in the *Human Task Design* stage. The *Task Deployment* activity is composed by two sub-activities: the *People to Task Matching* and the *Task Assignment.*

According to the *Conflict Resolution Strategy* defined at the previous stage, the *People to Task Matching* activity selects the best set of performers to execute a given *Crowd Task.* The *Conflict Resolution Platform* is queried in order to retrieve the most suitable set of performers. We define a performer to be suitable for the execution of a *Crowd Task* if he, or she, respects the *Content Affinity Criteria* (i.e., the topic of the task, performer task preferences) and the *Execution Criteria* (i.e., the skills of the performer, the difficulty of the task). The two criteria produce respectively a *content-based* and an *execution-based* performers rankings; the combination of these two rankings gives us the performers ranking among which the top-k performers can be selected for the task assignment.

During the *Task Assignment* activity the *Crowd Task* is assigned to the selected performer.

4.3.2.3 Task Execution

At this stage the performers execute the *Crowd Tasks.* To cope with the uncertainty on the quality of the performers' performances, the same task can be assigned to multiple performers. Thus, for the same task we might have multiple outcomes.

4.3.2.4 Output Aggregation

The *Output Aggregation* activity merge the outcomes of the *Task Execution* activity in order to produce the final task result. At this stage an aggregation policy, such as *Output Agreement* (see *Quality* requirement in Section 3.2), may need to be introduced.

4.4 The Process Model

In this section we define and detail the processes that compose our multimedia human-enhanced search-based application. The model we present is based on the studies reported in [Bozzon, 2009].
First, we introduce the *Indexing process*, which includes several activities, such as the retrieval of the contents to be managed by the application, the analysis of the content items and their indexation within the application. For the purposes of this thesis, we will focus on the *Content Analysis* activity, highlighting the presence of human computation tasks.
Then, we introduce the *Search process* model, which deals with the management of the queries and result presentation to the users.

4.4.1 The Indexing Process

The *Indexing process* in a search-based application is the activity that manages the contents' life-cycle. As depicted in Figure 4.10 is composed by three macro-activities:

- the starting one is the so-called *Content Registration* which deals with the fetching of the raw contents from one or more data sources and their registration into the application;
- the next activity is the *Content Analysis*, in which contents are analyzed by a set of components (automated or human-based) and enriched with a set of annotations;
- finally, the *Content Indexation* activity translates the outcomes of the previous activities in an indexing language, in order to feed the search engine indexes.

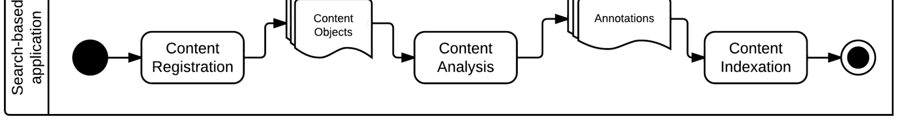


Figure 4.10: The *Indexing process*

Within our application both *Content Analysis* and *Content Indexation* activities may be executed several times in the *Indexing process*; this may happen when the analysis of the content is performed in an incremental fashion. For instance, the annotations of a content object, that has been already indexed, are sent to the crowd for further processing (e.g., validation); when the application receives these outcomes of crowdsourced tasks overtime from the *Conflict Resolution Platform*, thus also the application indexes need to be updated.

4.4.1.1 Content Registration

The *Content Registration* process is the triggering point for the whole *Indexing process.* Within this process, *Content Objects* to be processed by the search-based application are found and retrieved from the data sources. Data sources can be file systems, repositories, the Web, databases and, in a Human Computation view, even users (i.e., *Content Providers*). Once contents are fetched, the content items are registered into the application (i.e., stored in order to be consumed by the application components).
The content items consist of the multimedia file to be indexed (e.g., a video, an image) and, in certain cases, also of the annotations provided by the contents' provider itself (e.g., the textual description of a *YouTube* video given by the owner). Hence, a content item may already hold some annotations before being processed by the application annotators.
The *Content Registration* process can be further split into four main stages:

- the *Content Retrieval* process,
- the *Content Storage* process,
- the *Content Validation*
- and the *Annotation Storage* process

Figure 4.11 draws the steps that compose the *Content Registration* process.

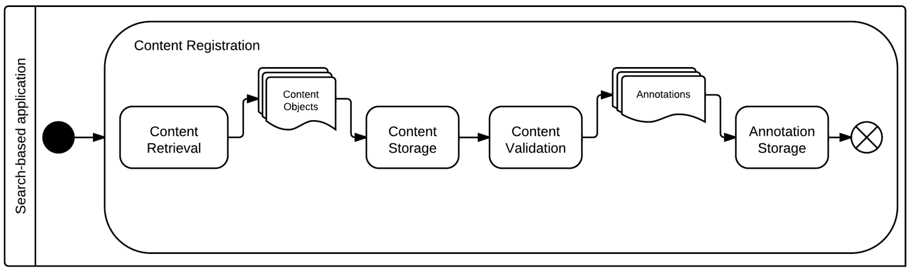


Figure 4.11: The *Content Registration process*

Content Retrieval *Content Retrieval* process relates to the actual fetching of the multimedia resources from the data sources. During the fetching of a single resource, multiple pieces of information can be retrieved; for instance, when fetching a video from an HTML page, the application may retrieve the video object and some other information, like the title, the tags and its description.

Inside the application, these information are attached to the multimedia content object as usual annotations.
The design of this process depends on the *Retrieval Policy* (see Design Dimensions, Section 2.4.4), which itself depends on the type of data source from which we fetch the contents. The two main policies for retrieval are *push* and *pull* modalities.
The *push* mode requires the data sources to notify the application, when a new information item is available. Thus, the *Content Retrieval* process is triggered by the content provider and such operation can be either manual or automatic. In our human-enhanced approach the *push* mode can be used to allow users to provide new multimedia contents to the application, in order to enrich and extend our indexes. For instance, in a trademark logo detection application, users could provide to the system new logo images given a brand name, improving the diversity of the logo instances indexed in the system. Moreover, another example of manual triggering of the *Content Retrieval* process in the logo detection application could occur in the case the system is queried with an unknown brand (i.e., not indexed yet): in addition to the usual "No result found" message, the user may have the chance to start a new instance of the *Indexing process* pushing the name of the brand to be processed.
Usually, the *push* mode is used on limited and controlled data sources, in which the update frequency is low.
Indeed, the *pull* mode, is used when it is impossible for the data source to notify the application about new content, due to its size or heterogeneity (e.g., the Web). Thus, the application should periodically explore (i.e., crawl) the data source in order to catch changes in the data source status (e.g., addition of a resource, update of a resource, etc.).

Content Storage This activity stores the multimedia resource within the application and assigns a unique identifier to it. The goal of the *Content Storage* activity is to make multimedia content items available for the following indexing processes.

Content Validation The *Content Validation* activity has a threefold purpose:

- perform controls on the format of the content items to filter out resources not manageable by the application;
- collect and create annotations coming from the *Content Provider*;
- check the legitimacy of the content items, in order to with respect to the application requirements, by validating the *Content Provider*'s annotations.

At the latter step checks are performed in order to exclude contents that are not indexable, such as duplicates, materials protected by copyright and illicit or offensive contents.

Annotation Storage The annotations created in the *Content Validation* stage are stored in order to make them available to the following *Content Analysis* process.

4.4.1.2 Content Analysis

In the *Content Analysis* process, the content items are analyzed in order to extract and produce annotations that can enrich their representation within the application. In addition, the *Content Analysis* process may post-process the annotations produced in the previous *Content Registration* process.
The activities that take place at this stage depend on the data managed by the application and on the underlying search engines (see *Data homogeneity*, *Data Analysis* and *Search Technology* design dimensions in Section 2.4.4). In our multimedia human-enhanced search-based application (hSBA), we limit our scope on the management of multimedia objects, such as videos and images. The same content item can be processed in several ways, hence the *Content Analysis* process may be not unique and there may be alternative content analysis models, composed by different sets of operations.
Within the *Content Analysis* process, each operation usually depends on the outcomes of another operation, thus they must be performed in a particular order. For instance, the detection of the key-frames within a video must happen before the extraction of low-level feature descriptors from the key-frames themselves.
Furthermore, some operations may be executed in parallel, for instance the classification of the audio track of a video and the low-level feature extraction of the video key-frames can be performed simultaneously.
All the analysis operations that can be carried out at this stage can be aggregated in three main stages:

- *Content Preparation*,
- *Content Processing*
- and *Annotation Storage.*

Figure 4.12 depicts the stages in the *Content Analysis* process.

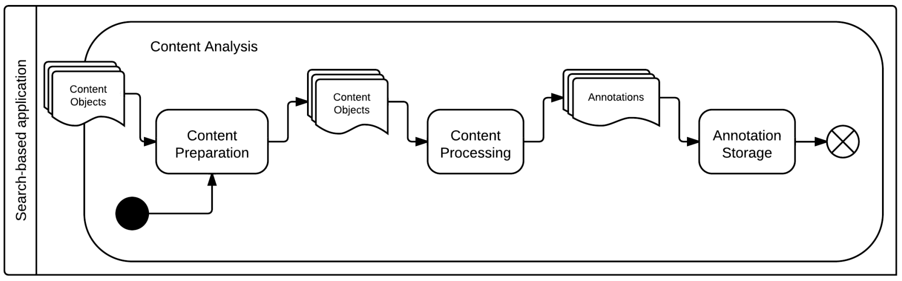


Figure 4.12: The *Content Analysis process*

Content Preparation This activity performs some pre-processing operations on the *Content Objects*, like read them from the storages and transform them into a suitable format. For instance, a video in .AVI format may need to be converted into .MPEG format in order to be processed by the video segmentation component.

Another example is given in the logo detection application (see Chapter 6): the *Content Objects* to be analyzed by the annotators are the video key-frames, hence key-frame detection and extraction are operations that we consider as part in the content preparation process.

Content Processing This activity is the core step in *Content Analysis* process. At this stage all the analysis operations on the multimedia content objects take place and annotations on the content objects are produced by automatic components. Typical analysis operations on multimedia contents are: video temporal segmentation, object detection, audio segmentation, audio classification, image matching. Notice that most of the operations reported require the computation of the low-level features of the multimedia content object (e.g., color histograms, local descriptors).
Automatic analysis components extract annotations and associate to them a value of confidence, that represent the reliability of the outcome. In a human-enhanced application some analysis operations on multimedia contents are performed by human beings. As proposed in [Bozzon et al., 2012c], human computation can enter in the content analysis workflows as a validation of the outcomes of the automatic annotators. For instance, humans could validate the results of an image matching component, which uses SIFT descriptors and RANSAC algorithm to match images. The confidence of the annotation returned by the software analysis component can be used as discriminant for the annotation's quality: annotations having the confidence value above a threshold are considered as reliable, while annotations with a confidence value falling in a lower range are marked as uncertain. As the name of the latter class suggests, within uncertain annotation may be wrongly classified also good annotations, thus humans can help in correctly classifying the annotations and, thus, in improving the annotators' performances in terms of quality.
The main drawback in the addition of human computation tasks in a *Content Analysis* process is the impact on time performances of the application (see Chapter 3) and the blocking effect that these tasks may have in the information flow of the application.
Thus, in our information flow, when the outcome of the *Content Analysis* is uncertain its validation is allocated to the *Human Enhancement process*, that we will discuss in Section 4.3.2.

Annotation Storage Annotations extracted and generated in the *Content Processing* step are merged, associated to the respective *Content Objects* and stored.
Among annotations, the *Content Processing* process may also create new content items for further analysis or later reuse. For instance, the key-frames extracted from a video objects are content items themselves on which image content analysis can be carried out.

4.4.1.3 Content Indexation

After the *Content Analysis* process, each *Content Object* is enriched by a set of annotations. In order to enable the search onto the processed *Content Objects*, annotations must be put in the search engine indexes.
Before annotations are put into the indexes, they need to be to be transformed

into a suitable format, according to the technology of each search engine within the search-based application (plain text, XML, content-based, etc.).
Thus, also the *Content Indexation* process can be split in two steps:

- *Index Entries Preparation*, the activity in charge of transform annotation into a suitable index format.

- *Index Entries Indexing*, the actual update of the search engine's indexes.

Figure 4.13 shows the two activities in the *Content Indexation* process.

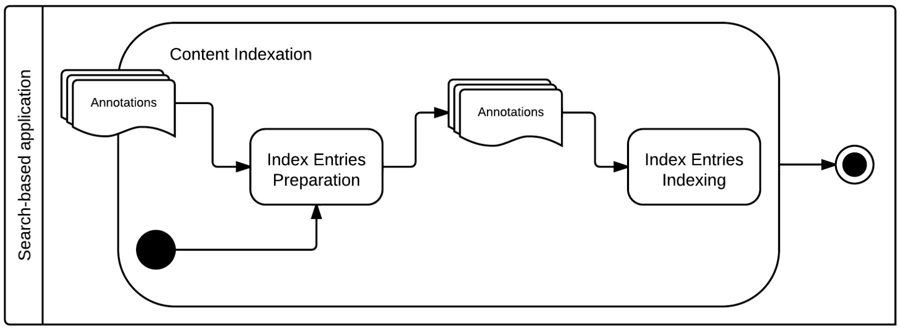


Figure 4.13: The *Content Indexation process*

4.4.2 The Search Process

Parallel to the *Indexing process*, a search-based application has to provide to users the means to query the indexes and to obtain results. According to the framework presented in Section 2.4.3, these tasks are in charge to the *Search process*, which allow the user interface to interact with the underlying search technologies. The *Search process* manages the queries (e.g., reformulates, transforms, expands them) and then forward them to the search engines and viceversa aggregates the results retrieved from the search engine's indexes and return to the user interface for the presentation to the users.
Hence, during the *Search process* several different activities take place; as for the *Indexing process* we can group them three main stages:

- the *Query Management*, that is in charge of the query processing;

- the *Search Orchestration*, which uses the queries to interact with the search engines;

- and the *Result Presentation*, which aggregates and the outcomes of the previous activity.

Figure 4.14 depicts the activities that take place during the *Search process*.

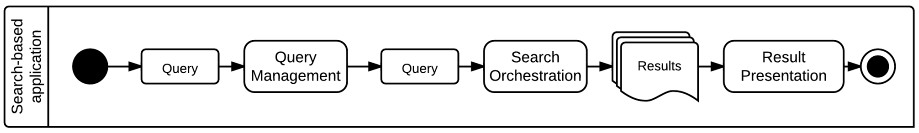


Figure 4.14: The *Search process*

In a human-enhanced search-based application, humans could contribute in the *Query Management* stage.

4.4.2.1 Query Management

The *Query Management* activity, as showed in Figure 4.15, is composed by three sub-activities:

- the *Query Analysis*,
- the *Query Adaption*,
- and the *Query Transformation*

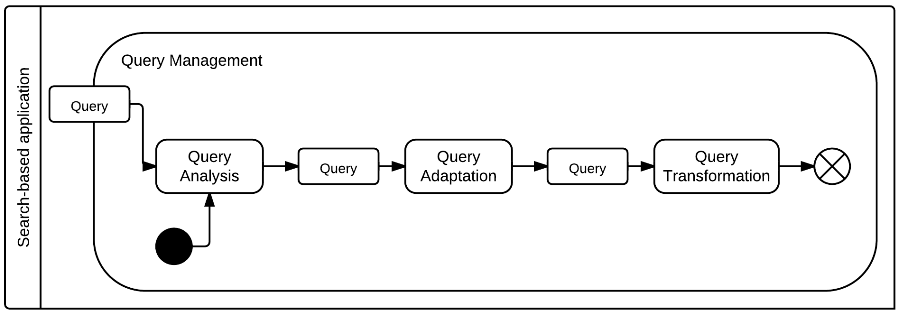


Figure 4.15: The *Query Management process*

Query Analysis In the *Query Analysis* step the queries are analyzed in order to check their correctness and to enable their execution within the application. According to the format of the query (e.g., textual, *content-based*, etc.) different query analysis are performed. For textual queries, query terms are subjected to linguistic processing (e.g., stemming, lemmatization, stop words removal, etc.). *Content-based* queries, instead, are processed in order to extract low-level features from the multimedia query terms to be use in similarity searches.
At this stage humans can help the application in several ways, performing tasks such as:

- check the syntax of the textual query terms (i.e., spell-check);
- provide new query terms to enrich the query terms, both textual and *content-based* queries (i.e., query expansion);

- transform textual query terms into *content-based* query terms and viceversa.

Hence, also during the Search process, the *Human Enhancement process* could be triggered. For instance, humans may be asked to provide a set of images given a concept (e.g., a location, a brand name, etc.), once the set of images is returned to the application, it is used to query the *content-based* search engine.

Query Adaptation As well as the *Query Analysis* step, the *Query Adaptation* step may modify the query terms. Such changes are introduced according to some contextual variables, like the user device or the user profile (e.g., expand the query with the geographical location of the device).

Query Transformation The *Query Transformation* step takes as input the analyzed and adapted queries and transform them into a format suitable for the search engines of the application, according to their *Search Technology* (e.g., textual, *content-based*, etc.).

4.4.2.2 Search Orchestration

The *Search Orchestration* process performs the actual search operations on the search engines. It is composed by two stages: *Query Planning* and *Query Execution* activities.

Query Planning The *Query Planning* activity *(i)* defines the query plan and *(ii)* dispatches the queries to their respective search engine.

Query Execution The query plan triggers the multiple *Query Execution* tasks that send queries to the search engines a collect their outcomes.

4.4.2.3 Result Presentation

Finally, the *Result Presentation* process is in charge of managing the results produced by the queries and making them suitable for the user interfaces. As depicted in Figure 4.16, the process is performed in three steps: the Result Aggregation, the *Result Enhancement* and the *Result Transformation.*

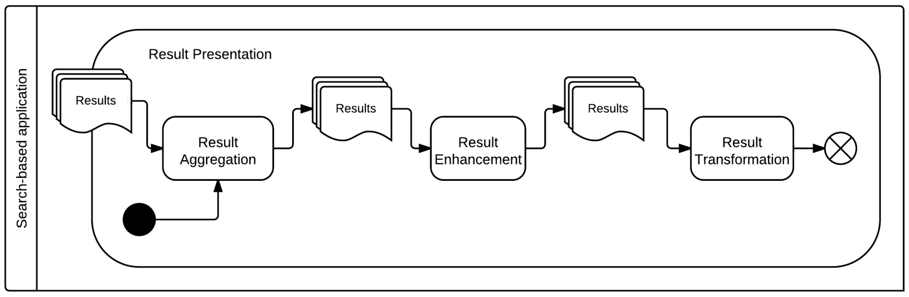


Figure 4.16: The *Result Presentation process*

Result Aggregation If the search-based application contains more than one search engine, the *Result Aggregation* step collects the outcomes of each search engine and merges them to produce a unique and coherent result set. For instance, in a multimedia search-based application user can send a query composed by a word and an image, with the following semantics: retrieve all the images having the word among its tags and that are similar to the sample image.

Result Enhancement The *Result Enhancement* activity attaches to the result set information coming from data sources (different from the search engines' indexes), in order to enrich the representation of the results. For instance, an audio search-based application may enrich the results associating to each song the lyrics, which are retrieved from the Web.

Result Transformation The last step in the *Result Presentation* process is the *Result Transformation* activity in which the format of the result set is adapted (either at design time or at query time) according to user interface.

In Chapter 5 we will describe the case study of a human-enhanced logo detection application and materialize the concepts and the models we have presented in this chapter.

5 Use Case: The Logo Detection

In the previous chapter we introduced a framework for a multimedia search-based application, in which Human Computation is involved in the processing of the contents and the annotations produced by automatic components. In this chapter we present a case of study which materializes the concepts and the models of our framework.
The use case on which we focus on is a search-based application for trademark logo detection within a video collection.

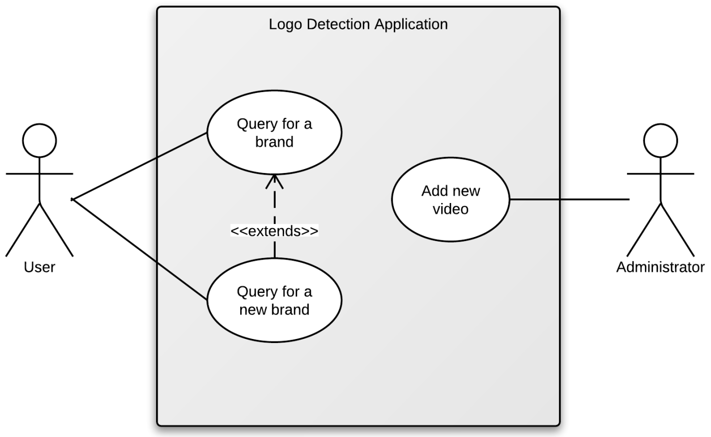


Figure 5.1: The logo detection Use Case Diagram

As depicted in the use case diagram in Figure 5.1, the application should:

- allow users to query with the name of a brand (e.g., Coca Cola) and should produce a report including all the occurrences of the brand logos in a video collection;

- allow users to start the processing of a new brand, in case of the query brand has not been processed yet;

- allow administrators of the application to process new videos.

In the next section we will delve into the processes needed in the logo detection application, then in Section 5.2 we will fit the data managed by the application in the data model we introduced in Chapter 4, finally in Section 5.4 we will review the use case processes according to the process model introduce in Section 4.4.

5.1 The Use Case Activities

In the logo detection application we are presenting in this thesis, the content processing flow is completely synchronous and has no delays due to the uncertain duration of the crowdsourced tasks. Indeed, the human computation tasks (i.e., validate logo instances, validate matches) cause just asynchronous updates on the application index (i.e., the match index).
According to this approach the index is built in an incremental fashion and we can keep track of the changes (i.e., events) that may happen in the index after the human computation tasks.
In order to perform a *content-based* search on the video collection we retrieve top-32 *Google Images*[14] logo instances.
The core logo detection application is the matching task between logo instances and key-frames of the videos. The matching is performed by computing the image similarity on the SIFT descriptors of the logo instances and the video key-frames. In literature [Lowe, 2004] is well-known that image similarity based on SIFT is largely affected by the quality of the images; according to this and to the fact that logo instances retrieved from *Google Images* may contain many irrelevant results (i.e., image not related to the brand), it make sense to introduce a validation task, which exploits crowdsourcing (i.e., a human validation task), to detect the real confidence of the logo instances (according to the brand name) to be used in the *content-based* search. This logo confidence can be used in the *Result Presentation* stage to discard the matches generated from low-confidence logo instances.
Moreover, the image matching task is itself subject to errors, thus human validation can be used also for validating the uncertain outcomes of this task (i.e., the low-confidence matches), providing a judgement on the generated low-confidence matches (i.e., relevant/not relevant).
Parallel to the processing of the logo instances, the logo detection application is in charge of retrieving the videos from the collection and process them in order to detect the video key-frames and to compute the SIFT descriptors of each key-frame to be used later in the matching task.
The logo detection application has been designed to support two modes: *off-line* indexing and *on-line* indexing.
In the *off-line* indexing mode the application processes videos crawled from a collection, simultaneously processes sets of logo instances related to a set of

[14]http://images.google.com

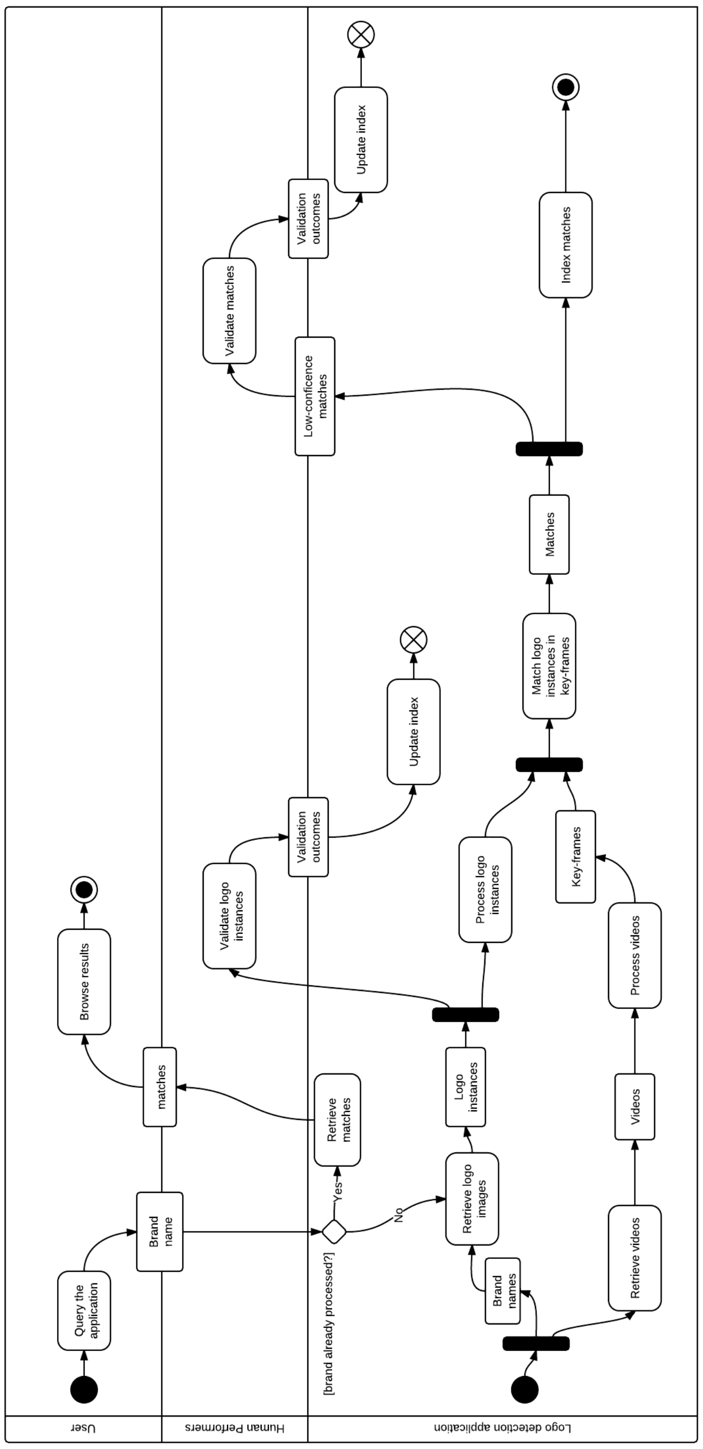


Figure 5.2: The logo detection activities

brand name defined a-priori, collects the crowdsourced tasks outcomes, computes the matches and puts them into the index.
In the *on-line* mode the logo detection application is triggered by a user textual query (i.e., the name of a brand). If the brand name has already been processed (i.e., the top-32 logo instances have been processed and compared with video key-frames), it is used to query the match index and results are presented. On the other hand, if it is the first time that the brand name is send to the application then the logo retrieval process and the subsequent processes are triggered to index on-the-fly new matches. This latter processes may last for several minutes, thus the end user has to wait for a while before receiving results from the application.

5.2 The Data Model

In the next sections we will instantiate the data models presented in Chapter 4 for the scenario of the trademark logo detection application.

5.2.1 The Content Data Model

The multimedia items managed by the application are a set of videos belonging to a collection (e.g., *GroZi-120*[15] multimedia database) and some sets of logo images, each set related to a trademark (i.e., a brand). Hence, according to the *Content Data model* described in Section 4.2.1, the application deals with *Video Objects* and *Image Objects*. During the processing of each video, after the detection of the video key-frames, other *Image Objects* are created, each one associated to a detected key-frame.
The core data object that the application manages and indexes are the *matches*. A *match* is one result of the *Match logo instances in key-frames* activity and it is created just if the logo instance and key-frame have an image-similarity (i.e., match confidence) greater than zero. In our approach a *match* is modeled as a relationship between a key-frame and a logo instance, to which a value of confidence and the bounding box coordinates are associated. According to our model a *match* is a *Similar Content Of* relationship between two *Image Objects* (i.e., the key-frame and the logo instance).
Figure 5.3 shows an instantiation of the *Content Data model* according to the multimedia items managed by the logo detection application. The most important annotations produced within the logo detection application are the low-level features, i.e., SIFT descriptors, for each *Image Object* processed. These features are needed by the matching software component to identify possible *matches* between key-frames an logo instances.
During the *Process videos* activity, the SIFT feature extractor component computes the low-level features descriptors for each key-frame extracted from the videos, and generates the low-level-features annotations. Similarly, in the *Process logo instances activity* the SIFT feature extractor component computes SIFT descriptors for logo instances retrieved from *Google Images*.
In Figure 5.4 we report the data model including the annotations. Once a set of key-frames and a set of logo instances are processed, the SIFT descriptors matching software component, beside creating the match relationships between

[15]http://grozi.calit2.net/grozi.html

the key-frames and the logo instances *Image Objects*, generates the annotations to be attached at the match relationships: the "found match" annotation (i.e., a textual annotation describing the *match*) and the bounding box coordinates. To the former annotation an *Annotation Confidence* value is attached, which corresponds to the image-similarity (i.e., the match score) computed by the matching software component. Such confidence value, given an uncertainty interval, is used as discriminant to identify *Uncertain Annotations* (i.e., uncertain matches) to be validated by the crowd.

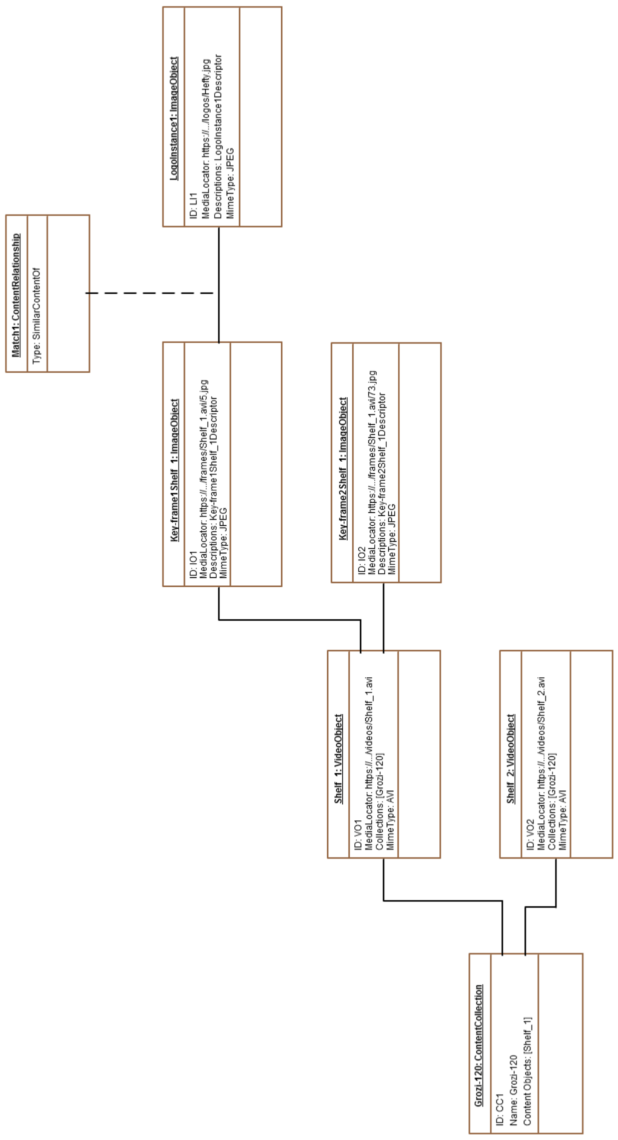

LogoInstance1: ImageObject
ID: LI1
MediaLocator: https://.../logos/Hefty.jpg
Descriptions: LogoInstance1Descriptor
MimeType: JPEG
Match1: ContentRelationship
Type: SimilarContentOf
Key-frame1Shelf_1: ImageObject
ID: IO1
MediaLocator: https://.../frames/Shelf_1.avi/5.jpg
Descriptions: Key-frame1Shelf_1Descriptor
MimeType: JPEG
Key-frame2Shelf_1: ImageObject
ID: IO2
MediaLocator: https://.../frames/Shelf_1.avi/73.jpg
Descriptions: Key-frame2Shelf_1Descriptor
MimeType: JPEG
Shelf_1: VideoObject
ID: VO1
MediaLocator: https://.../videos/Shelf_1.avi
Collections: [Grozi-120]
MimeType: AVI
Shelf_2: VideoObject
ID: VO2
MediaLocator: https://.../videos/Shelf_2.avi
Collections: [Grozi-120]
MimeType: AVI
Grozi-120: ContentCollection
ID: CC1
Name: Grozi-120
Content Objects: [Shelf_1]

Figure 5.3: Instantiation of the *Content Data model* for the logo detection use case

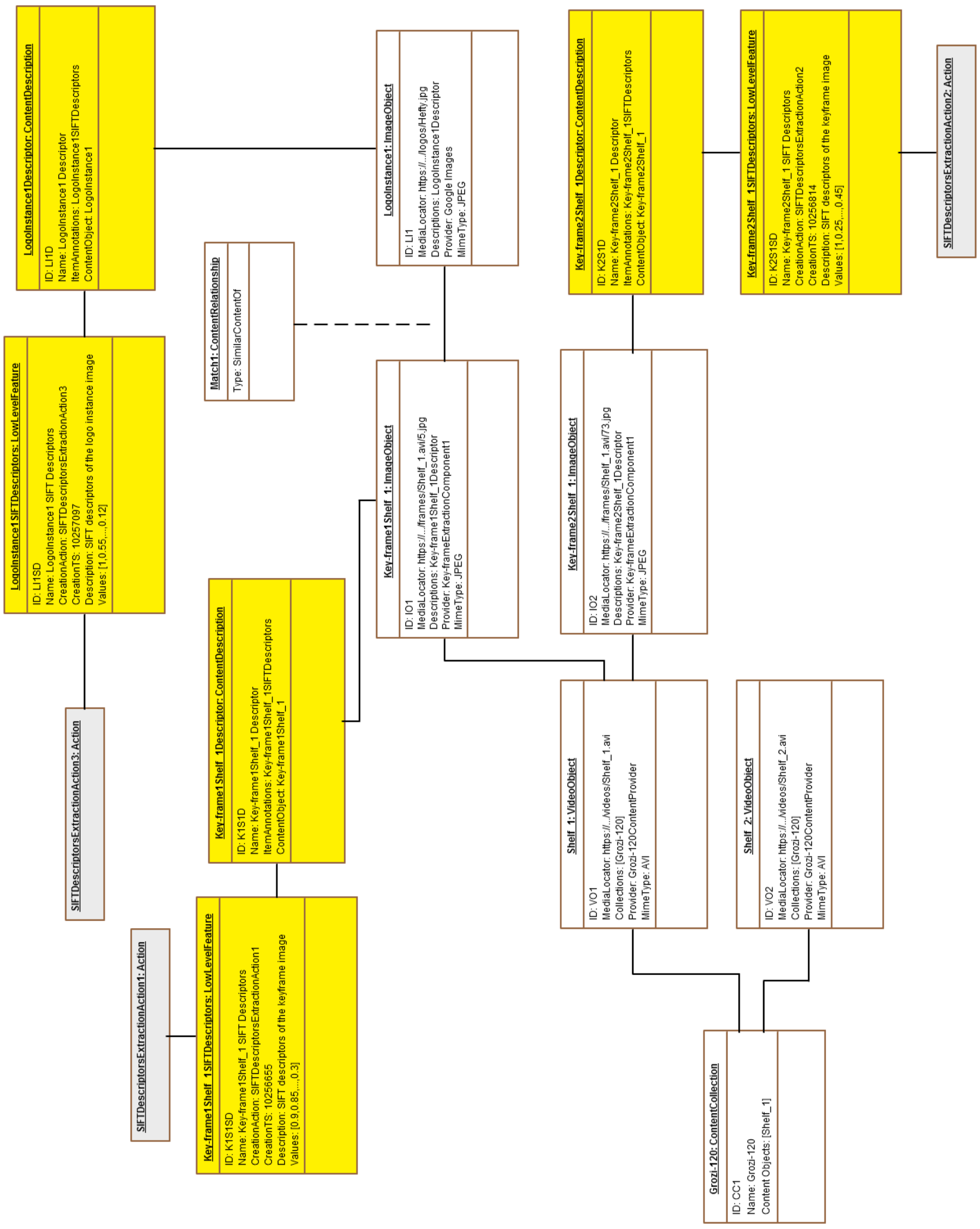

LogoInstance1Descriptor: ContentDescription
ID: LI1D
Name: LogoInstance1 Descriptor
ItemAnnotations: LogoInstance1SIFTDescriptors
ContentObject: LogoInstance1
LogoInstance1: ImageObject
ID: LI1
MediaLocator: https://.../logos/Hefty.jpg
Descriptions: LogoInstance1Descriptor
Provider: Google Images
MimeType: JPEG
Key-frame2Shelf_1Descriptor: ContentDescription
ID: K2S1D
Name: Key-frame2Shelf_1 Descriptor
ItemAnnotations: Key-frame2Shelf_1SIFTDescriptors
ContentObject: Key-frame2Shelf_1
Key-frame2Shelf_1SIFTDescriptors: LowLevelFeature
ID: K2S1SD
Name: Key-frame2Shelf_1 SIFT Descriptors
CreationAction: SIFTDescriptorsExtractionAction2
CreationTS: 10256814
Description: SIFT descriptors of the keyframe image
Values: [1,0.25,...,0.45]
SIFTDescriptorsExtractionAction2: Action
Match1: ContentRelationship
Type: SimilarContentOf
LogoInstance1SIFTDescriptors: LowLevelFeature
ID: LI1SD
Name: LogoInstance1 SIFT Descriptors
CreationAction: SIFTDescriptorsExtractionAction3
CreationTS: 10257097
Description: SIFT descriptors of the logo instance image
Values: [1,0.55,...,0.12]
Key-frame1Shelf_1: ImageObject
ID: IO1
MediaLocator: https://.../frames/Shelf_1.avi/5.jpg
Descriptions: Key-frame1Shelf_1Descriptor
Provider: Key-frameExtractionComponent1
MimeType: JPEG
Key-frame2Shelf_1: ImageObject
ID: IO2
MediaLocator: https://.../frames/Shelf_1.avi/73.jpg
Descriptions: Key-frame2Shelf_1Descriptor
Provider: Key-frameExtractionComponent1
MimeType: JPEG
SIFTDescriptorsExtractionAction3: Action
Key-frame1Shelf_1Descriptor: ContentDescription
ID: K1S1D
Name: Key-frame1Shelf_1 Descriptor
ItemAnnotations: Key-frame1Shelf_1SIFTDescriptors
ContentObject: Key-frame1Shelf_1
Shelf_1: VideoObject
ID: VO1
MediaLocator: https://.../videos/Shelf_1.avi
Collections: [Grozi-120]
Provider: Grozi-120ContentProvider
MimeType: AVI
Shelf_2: VideoObject
ID: VO2
MediaLocator: https://.../videos/Shelf_2.avi
Collections: [Grozi-120]
Provider: Grozi-120ContentProvider
MimeType: AVI
SIFTDescriptorsExtractionAction1: Action
Key-frame1Shelf_1SIFTDescriptors: LowLevelFeature
ID: K1S1SD
Name: Key-frame1Shelf_1 SIFT Descriptors
CreationAction: SIFTDescriptorsExtractionAction1
CreationTS: 10256655
Description: SIFT descriptors of the keyframe image
Values: [0.9,0.85,...,0.3]
Grozi-120: ContentCollection
ID: CC1
Name: Grozi-120
Content Objects: [Shelf_1]

Figure 5.4: Instantiation of the *Content Description model* for the logo detection use case

5.2.2 The Action Data Model

The operations on the multimedia items are modeled as *Actions* in the *Action data model* (see Section 4.2.2). The actions performed are both *Automatic Actions* and *Human Actions*. *Automatic Actions* comprise the processing of the video, their key-frames, the logo instances and the matching process, while *Human Actions* include the two validation tasks (see Section 5.3.1). Figure 5.5 draws an instantiation of the *Action model* and the *Content Processing model* for the key-frame and the SIFT descriptors extraction actions.

5.2.3 The Content Processing Data Model

The logo detection application is implemented in the SMILA framework (see Chapter 6), and its processes are instantiated in the SMILA pipelines. The *Automatic Actions* that occur in the application are created by the *System Components*. For instance, the *SIFT descriptor extraction Action* is created by the *SIFT descriptor extractor Software Component*.
System Components are executed in the *Processing Pipelines* of the logo detection application, as follows:

- *Video processing* pipeline:
  - *Key-frame detection* component
  - *Key-frame extraction* component
  - *SIFT descriptors extraction* component
- *Logo retrieval* pipeline:
  - *Google Images crawler* component
- *Logo processing* pipeline:
  - *SIFT descriptors extraction* component
- *Key-frame matching* pipeline:
  - *SIFT descriptors matching* component
- *Index update* pipeline

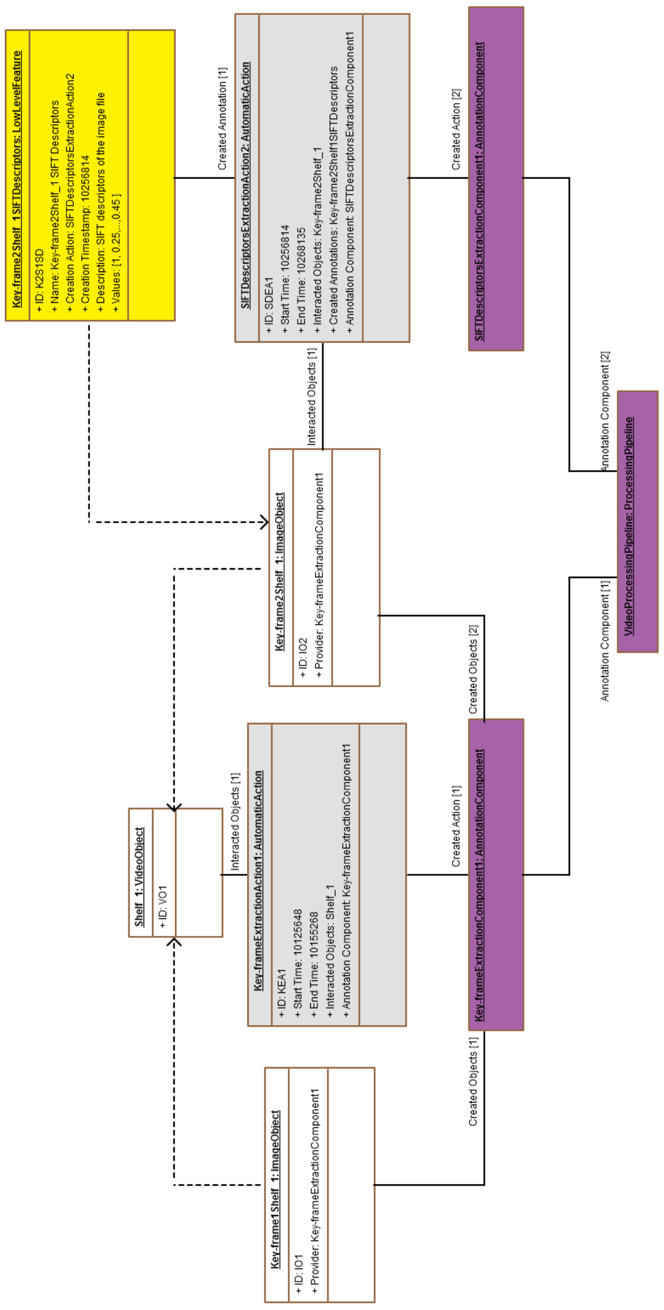

Key-frame2Shelf_1SIFTDescriptors: LowLevelFeature
+ ID: K2S1SD
+ Name: Key-frame2Shelf_1 SIFT Descriptors
+ Creation Action: SIFTDescriptorsExtractionAction2
+ Creation Timestamp: 10256814
+ Description: SIFT descriptors of the image file
+ Values: [1, 0.25,...,0.45]
Created Annotation [1]
SIFTDescriptorsExtractionAction2: AutomaticAction
+ ID: SDEA1
+ Start Time: 10256814
+ End Time: 10268135
+ Interacted Objects: Key-frame2Shelf_1
+ Created Annotations: Key-frame2Shelf1SIFTDescriptors
+ Annotation Component: SIFTDescriptorsExtractionComponent1
Created Action [2]
SIFTDescriptorsExtractionComponent1: AnnotationComponent
Interacted Objects [1]
Annotation Component [2]
VideoProcessingPipeline: ProcessingPipeline
Annotation Component [1]
Key-frame2Shelf_1: ImageObject
+ ID: IO2
+ Provider: Key-frameExtractionComponent1
Created Objects [2]
Shelf_1: VideoObject
+ ID: VO1
Interacted Objects [1]
Key-frameExtractionAction1: AutomaticAction
+ ID: KEA1
+ Start Time: 10125648
+ End Time: 10155268
+ Interacted Objects: Shelf_1
+ Annotation Component: Key-frameExtractionComponent1
Created Action [1]
Key-frameExtractionComponent1: AnnotationComponent
Created Objects [1]
Key-frame1Shelf_1: ImageObject
+ ID: IO1
+ Provider: Key-frameExtractionComponent1

Figure 5.5: Instantiation of both the *Action model* and the *Content Processing model* for the key-frame and the SIFT descriptors extraction actions

5.3 The Human Enhancement Model

This section presents the instantiation of the *Human Enhancement* data model and of the *Human Enhancement* process for the logo detection application use case.

5.3.1 The Human Enhancement Data Model

Among the annotations created by automatic components, like the SIFT descriptors extractor or the matching component, the logo detection application receives also annotations generated by the *Conflict Resolution Tasks.* The crowd is involved in resolving two different type of conflicts:

1. assign a confidence to the logo instances retrieved from *Google Images* (i.e., *Missing Annotation* conflict);
2. determine the effective relevance of matches produced (i.e., *Uncertain Annotations*).

In the former task the annotations generated by human performers are needed to provide reliable level of confidence to the logo instance, as we cannot rely on the rank given by *Google Images.* The logo confidence is used to build an effective logo instance rank in terms of quality.
In the latter task the outcome of the crowdsourced *Action* is simply a Boolean filtering (e.g., relevant/not relevant) on the *matches* identified by the automatic matching component, that have a confidence within the uncertainty interval.
The *Content Resolution Platform* we consider as execution platform for our crowdsourced tasks is the social network *Facebook* and the *Micro Tasks* to be executed by the human performers are simple preferences tasks (i.e., like).
In the logo instances validation case, the *Macro Task* is the validation of the top-32 logo instances retrieved form *Google Images* for a certain brand. This *Macro Task* can be split in several *Micro Tasks*, in which a single performers has to validate just a subset of the 32 logo instances. The activity of the human performer is the selection (i.e., like) of the logo instances images that he/she considers the most related to a given brand name. These subsets should be not exclusive as we wish to collect more than one opinion on a certain logo instance for higher reliability.
Indeed, in the match validation case, the *Macro Task* is the validation of a set of matches (e.g., the matches generated from the top-32 logo instances for the brand “Hefty”), which can be split in several *Micro Tasks.* Each *Micro Task* consists in the validation of a subset of the matches. Given a set of couples key-frame/logo instance, the human performer is asked to vote (i.e., like) the couples in which the logo instance, or a similar one, appears in the key-frame.
In Figure 5.6 we provide an instantiation of the *Conflict Resolution model* for the logo instances validation task.

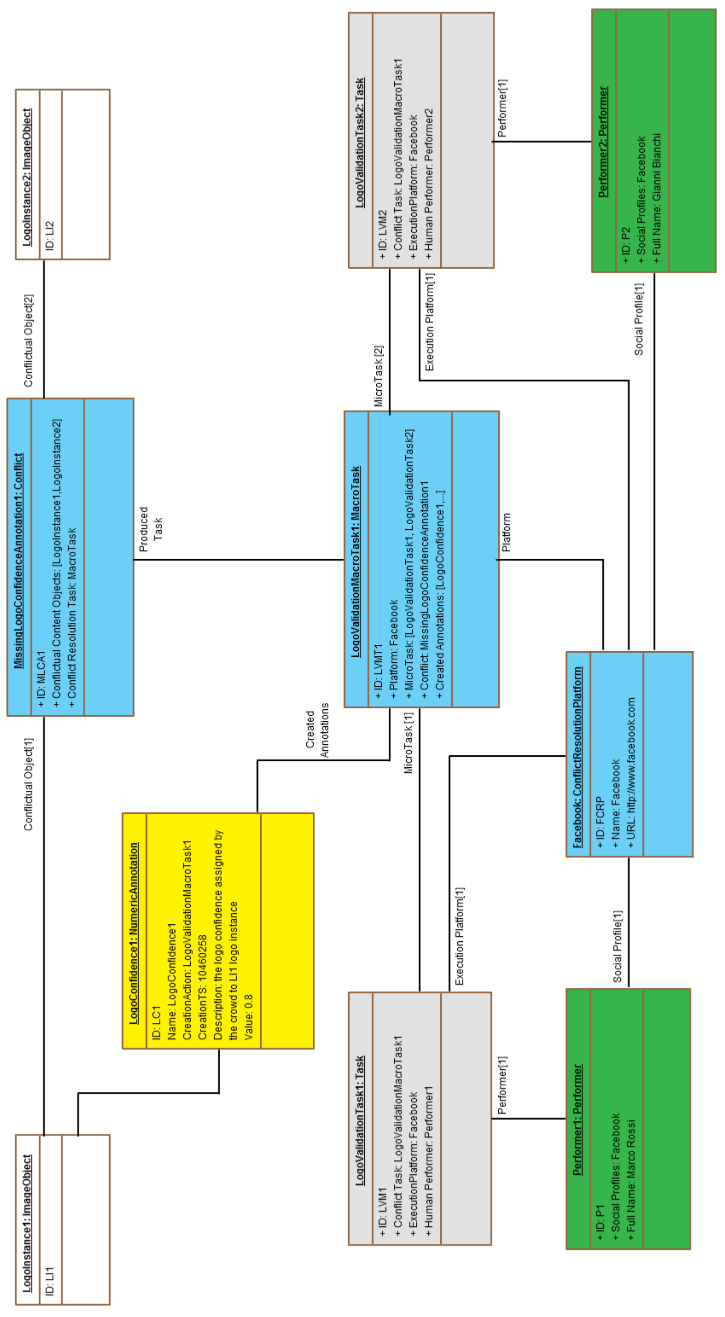

LogoInstance2: ImageObject
ID: LI2
Conflictual Object[2]
MissingLogoConfidenceAnnotation1: Conflict
+ ID: MLCA1
+ Conflictual Content Objects: [LogoInstance1,LogoInstance2]
+ Conflict Resolution Task: MacroTask
Conflictual Object[1]
Produced Task
LogoInstance1: ImageObject
ID: LI1
LogoConfidence1: NumericAnnotation
ID: LC1
Name: LogoConfidence1
CreationAction: LogoValidationMacroTask1
CreationTS: 10460258
Description: the logo confidence assigned by the crowd to LI1 logo instance
Value: 0.8
Created Annotations
LogoValidationMacroTask1: MacroTask
+ ID: LVMT1
+ Platform: Facebook
+ MicroTask: [LogoValidationTask1, LogoValidationTask2]
+ Conflict: MissingLogoConfidenceAnnotation1
+ Created Annotations: [LogoConfidence1,...]
MicroTask [2]
MicroTask [1]
Platform
LogoValidationTask2: Task
+ ID: LVM2
+ Conflict Task: LogoValidationMacroTask1
+ ExecutionPlatform: Facebook
+ Human Performer: Performer2
Performer[1]
Execution Platform[1]
Performer2: Performer
+ ID: P2
+ Social Profiles: Facebook
+ Full Name: Gianni Bianchi
Social Profile[1]
Facebook: ConflictResolutionPlatform
+ ID: FCRP
+ Name: Facebook
+ URL: http://www.facebook.com
Social Profile[1]
Execution Platform[1]
LogoValidationTask1: Task
+ ID: LVM1
+ Conflict Task: LogoValidationMacroTask1
+ ExecutionPlatform: Facebook
+ Human Performer: Performer1
Performer[1]
Performer1: Performer
+ ID: P1
+ Social Profiles: Facebook
+ Full Name: Marco Rossi

Figure 5.6: Instantiation of the *Conflict Resolution model* for the logo instances validation task

5.3.2 The Human Enhancement Process

In our use case, humans enter in the loop to provide a reliable validation on the uncertain annotations (i.e., conflicts) produced by software components in the application. As introduced in Section 4.3.2. the *Human Enhancement process* is in charge of managing the resolution of the conflicts.
Conflict Resolution Tasks are triggered in two distinct moments during the *Indexing process* (see Section 5.3.1), i.e., when:

- the top-32 logo instances are retrieved from *Google Images*;
- uncertain matches are identified by the matching software component.

The two situations trigger respectively the following *Macro Tasks*:

Validate logo instances The logo instances retrieved from *Google Images* may contain irrelevant results (w.r.t. to the query brand name). Thus, we need to annotate each logo instance with a level of confidence (i.e., logo confidence) w.r.t. the brand name, used to retrieve them. As such task is very complex for a software component, we opted to assign it to the human performers on the *Conflict Resolution Platform* (i.e., *Facebook*).

Validate uncertain matches The key-frames/logo instances matching activity is subject to errors. The image-similarities (i.e., match confidences) returned by the matching component range from 0 to 1. Thus, beside the selection of an high confidence and a low confidence threshold, the match confidences can be used to discriminate between GOOD *matches*, UNCERTAIN matches and REJECTED matches. Humans can help the logo detection application in finding good *matches* that the matching component wrongly classified as UNCERTAIN ones.

When a *Conflict Resolution Task* arises, its management is allocated to the *Human Enhancement process* that design a set of suitable *Crowd Tasks*, assign the tasks to the human performers and merge the outcomes. The *Conflict Resolution Platform* chosen for the execution of the *Crowd Tasks* is *Facebook* [Bozzon et al., 2012c].
Due to its specific nature, the *Human Enhancement* process is performed beyond the boundaries of the logo detection application, thus it is delegated to the *Conflict Resolution Manager*: *CrowdSearcher* [Bozzon et al., 2012a]. Figure 5.7 depicts the process for both logo instances validation and match validation tasks. In the following sections we fit the activities of the *Human Enhancement* process in the context of our use case.

5.3.2.1 Human Task Design

The *Human Task Design* is in charge of defining the *Task Execution GUI*, i.e., the interface that is provided to the human performers to fulfil the *Crowd Tasks*. The GUI template is similar in both logo instances and match validation cases. It simply consists in a list of images, each one associated to a checkbox:

- in the logo validation case a subset of the top-32 *Google Images* logo instances is presented to the performers and each performer is asked to select

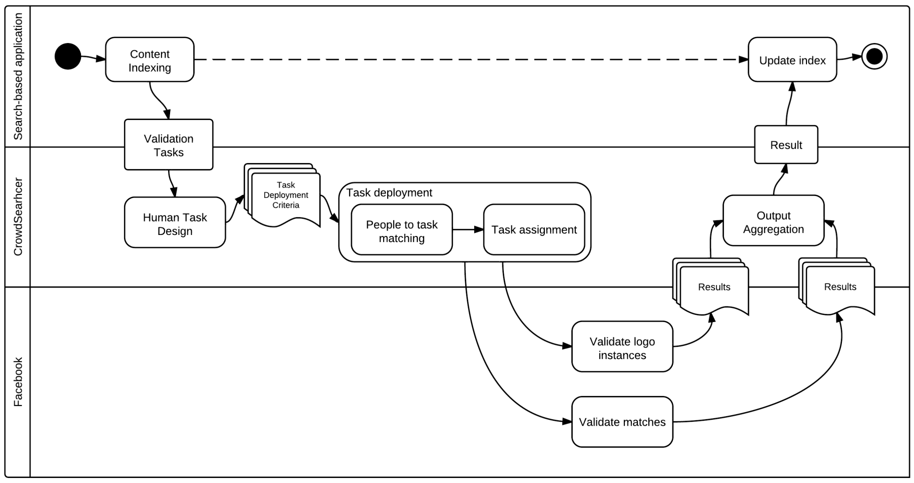


Figure 5.7: The *Human Enhancement process* in the logo detection application

(i.e., checking the corresponding checkboxes) the logo instances he/she consider to be the most relevant w.r.t. a given brand name.

- indeed in the match validation case a subset of uncertain matches (i.e., the couple key-frame/logo instance) identified within the logo detection application is presented to the performers and each performer is asked to select the key-frame/logo instance couples in which the logo instance (or a similar one) actually appears in the key-frame.

The *Human Task Design* activity should also define the *Task Deployment Criteria* in order to select the best performers for each task. For the scopes of this thesis candidate performers are simply selected among the friends of the task owner (e.g., a *Facebook* account associated to the logo detection application) on *Facebook*, with no restrictions on their location, skills and preferences.

5.3.2.2 Task Deployment

In this activity the task owner connects to the *Conflict Resolution Platform* (i.e., *Facebook*) and collects the set of human performers in a worker pool, selecting them from its friends list (i.e., *People to Task matching* step). Once performers are selected the crowd tasks are assigned to them (i.e., *Task Assignment* step).

5.3.2.3 Task Execution

The *Task Execution* activity is implemented as a native *Facebook* application that the performer has to install in order to perform the task, and, if he/she wishes, to re-dispatch it to other friends.

5.3.2.4 Output Aggregation

The *Output Aggregation* activity is in charge to collect the outcomes of the single *Crowd Tasks* assigned to human performers and to collect them in order

to produce a unique result.
In our use case we use two different aggregation policy for the validation tasks:

- in the logo validation case, *CrowdSearcher* collects the number of votes that each logo instance receives, the number of votes is used to compute the confidence value (which ranges from 0 to 1) to the logo instance;
- indeed, in the match validation case, we use a *majority rule* approach. *CrowdSearcher* collects the performers choices (checked or unchecked) for each key-frame/logo instance couple, then a couple is considered relevant if the number of checks (i.e., performers that voted for the couple) is more than the number of unchecks.

5.4 The Process Model

In this section we will fit the logo detection application processes, introduced at the beginning of this chapter, in the search-based application framework presented in Chapter 4. Moreover we will delve into the logo detection application processes providing a detailed description of all the activity involved in the application.
Figure 5.8 depicts the activities of the logo detection application highlighting the coarse search-based application processes to which they are related (i.e., indexing, human validation and search).

5.4.1 The Indexing Process

In our logo detection application the indexing process is in charge of the following tasks:

- retrieve the videos from the *GroZi-120* video collection;
- retrieve the top-32 logo instances from *Google Images*, given a brand name;
- process each video (key-frame detection, key-frame extraction, etc.);
- compute the matches between key-frames and logo instances;
- index the found matches.

In the next sections we associate each one of the listed tasks to the related indexing process activity (i.e., *Content Registration*, *Content Analysis* and *Content Indexation*).

5.4.1.1 Content Registration

As introduced in Section 4.4.1.1 the *Content Registration* activity has to feed the whole indexing process, retrieving the information items to be processed and indexed from the external data sources (e.g., file repositories, the Web, etc.).
As depicted in Figure 5.9, in our logo detection application the *Content Registration* activity comprises the following tasks:

- *Content Retrieval*:

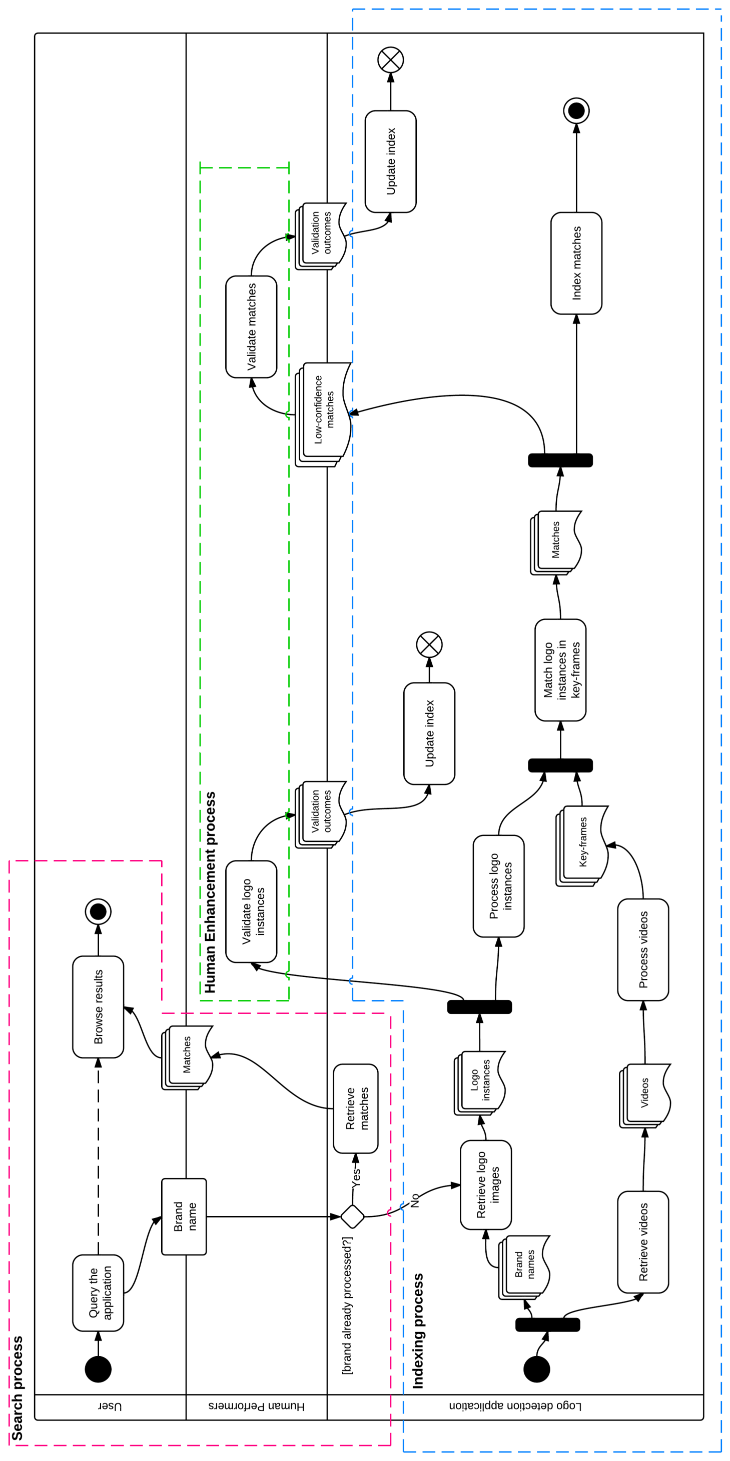


Figure 5.8: The coarse SBA processes in the logo detection application

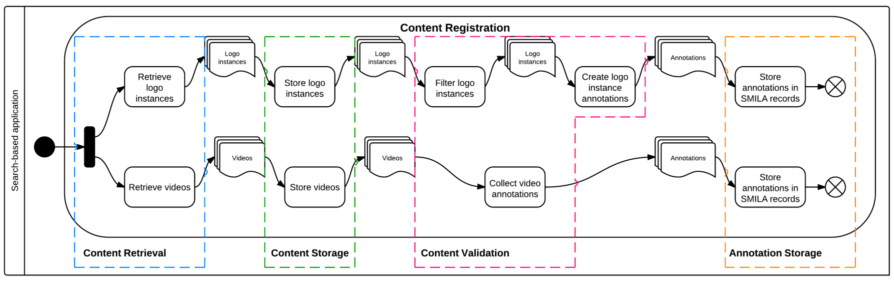


Figure 5.9: The *Content Registration* activity in the logo detection application

- *Retrieve videos*: the *GroZi-120* video collection (29 video files in AVI format) is stored in a directory on a local file system. A crawler is in charge to periodically explore the directory in order to detect changes in the video collection status (i.e., addition of a new video file). The choice of a pull mode for the retrieval of the videos is due to the use of the SMILA technology (see Chapter 6).
- *Retrieve logo instances from Google Images*: a software component (i.e., the *Google Photo crawler*) sends a query to *Google Images* in order to retrieve the top-32 images related to a given brand name. After the retrieval, the *Human Enhancement process* is triggered (see Section 5.3.2), in order to establish the actual confidence logo instances w.r.t the brand name.
- *Content Storage*:
  - *Store videos*
  - *Store logo instances*

  Both video and logo instances objects are uploaded on a *Data Service* (see Section 6.1.2.1) that returns for each object an unique identifier, that makes multimedia objects available for the following indexing processes.

• *Content Validation*

- *Filter logo instances*: logo instances having an image format not compatible with the annotator components (e.g., .gif image files) are discarded from further processing.
- *Collect video annotations*: a set of annotations derived from the corresponding video file is associated to each video object (e.g., video title, format, size, etc.).
- *Create logo instance annotations*: a set of annotations is created for each logo instance image objects (e.g., brand name, URL, confidence, etc.).

• *Annotation Storage*: the annotations collected and created in the previous stages are store in SMILA compliant data structures (i.e., the SMILA records).

5.4.1.2 Content Analysis

The *Content Analysis* process carries out all the operations needed to perform the *content-based* comparison between the videos and the logo instance objects. To perform such comparison we need to compute the SIFT descriptors of the logo instances and the SIFT descriptors of the key-frames of each video. Hence, the videos need to be pre-processed in order to extract key-frames image objects. Notice that this video pre-processing activity, produces new *Image Objects* to be analyzed.
Once the new *Image Objects* have been produced, we can compute their SIFT descriptors and use them to compute the image-similarity w.r.t. to the logo instances image objects.
Figure 5.10 depicts the activities involved in the *Content Analysis* workflow.

As described in Section 4.4.1.2 within the Content Analysis process we can identify four stages; here we report the logo detection application activity performed at each stage:

- *Content Preparation*:
  - *Detect key-frames*: a video analyzer component is used to detect the key-frames in the videos.
  - *Extract key-frames*: for each key-frame detected, a software component is in charge of extracting the corresponding image file (e.g., .JPEG file) from the video file.
- *Content Processing*:
  - *Extract SIFT descriptors from key-frames*: a software component is in charge of detect the local features in the key-frame image object and calculate the SIFT descriptors.
  - *Extract SIFT descriptors from logo instances*: the same software component is used to compute the SIFT descriptors of the logo instances image objects.
  - *Match logo instances in key-frames*: a software component performs the matching of the SIFT descriptors of each key-frame/logo instance couple and returns an image-similarity (i.e., match confidence) and the coordinates of the bounding box surrounding the area in the key-frame image in which the logo instance should have been detected. If the match is classified as UNCERTAIN (see Section 5.3.2) the *Human Enhancement process* is triggered.
- *Annotation Storage*: the annotations collected and created in the previous stages are stored in SMILA compliant data structures (i.e., the SMILA records).

5.4.1.3 Content Indexation

In our logo detection application the *matches* produced during the *Match logo instances in key-frames* activity are the information items to be indexed and searched by end users. According to the indexing technology provided within SMILA, match are put in a Solr index (see Chapter 6). Hence during the

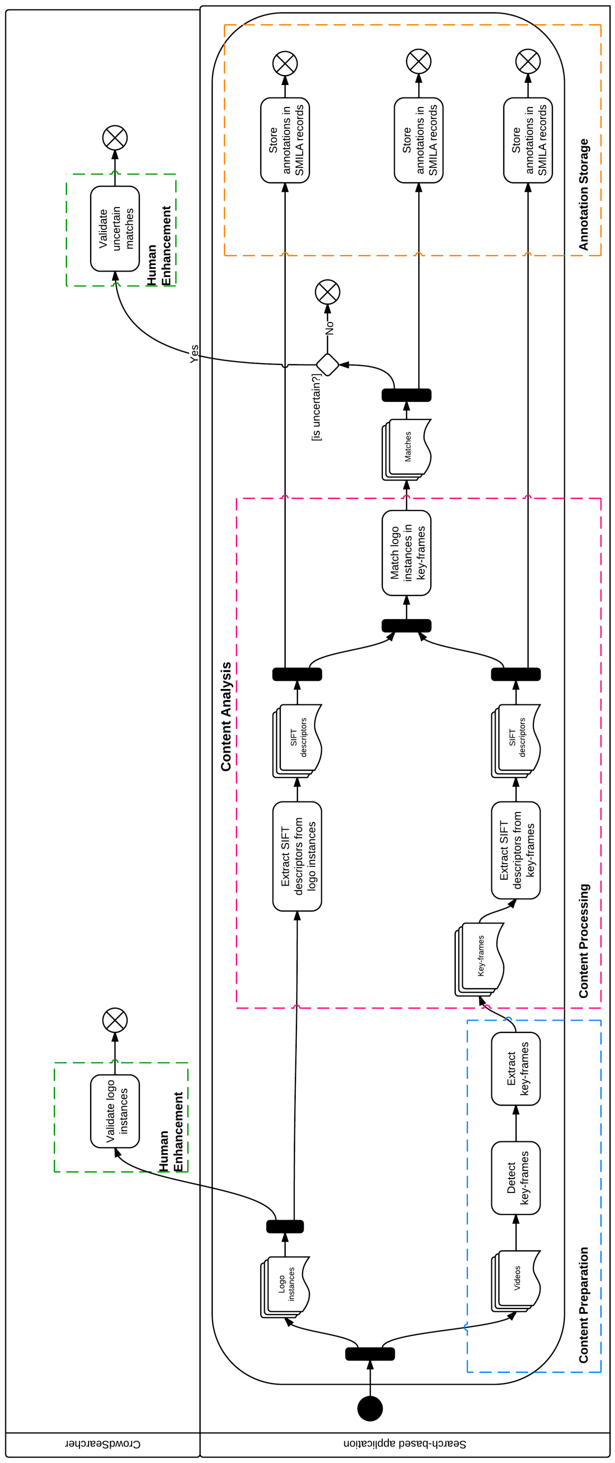


Figure 5.10: The *Content Analysis* activity in the logo detection application

Content Indexation activity, prior to the actual insertion in the index, *match* annotations need to be translated in a format that the index can accept and manage (i.e., *Index Entries Preparation* stage).
Figure 5.11 depicts the use case activities according to the *Content Indexation* activity ones.

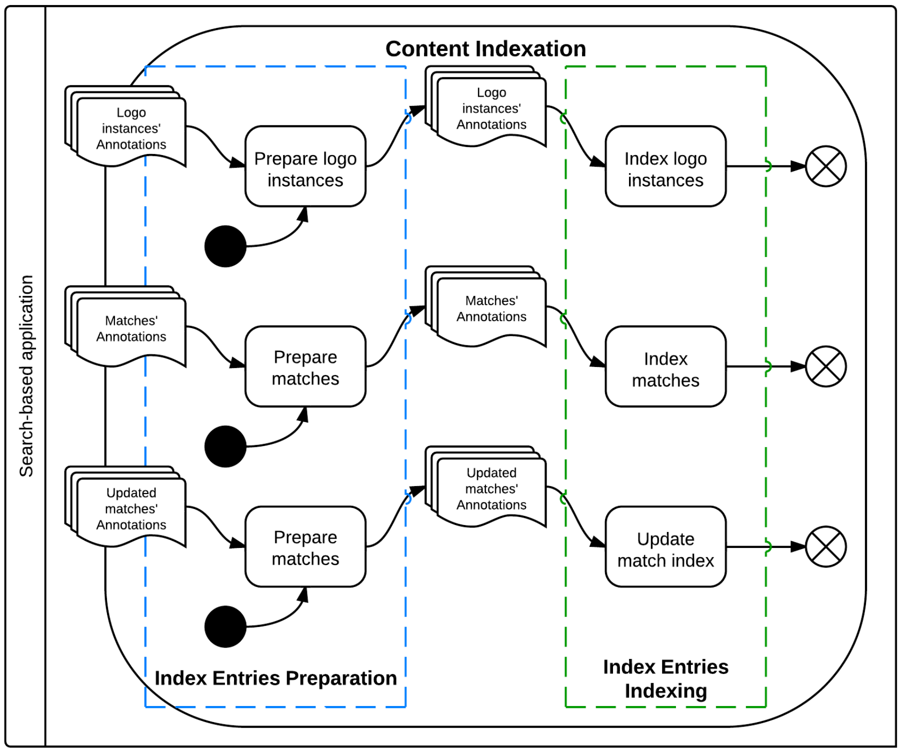


Figure 5.11: The *Content Indexation* activity in the logo detection application

Insertions and updates of *matches* in the index happen in three distinct moment: after the computation of the *matches* and after the validation of logo instances and of uncertain *matches* by the crowd. In addition, to the *match* index, we need to keep track also of the searched brands and of the processed logo instances (see Section 5.3.2), hence we put the processed logo instances in another Solr index (i.e., logo index). The activities needed in the *Content Indexation* activity are associated to the activity's stages as follows:

- *Index Entries Preparation*:
  - *Prepare matches*: *matches* annotations are converted in the index format. For instance, in order to maintain the relationship integrity between the *match* and the couple key-frame/logo instance to which it is related, in the index we need to collapse the annotations of the triple *match*/key-frame/logo instance.
  - *Prepare logo instances*: logo annotations are converted in the index

format. The annotations to index are simply the brand name and the URL of the logo instance (i.e., returned by *Google Images*).

- *Index Entries Indexing*:
  - *Index matches*: after the *matches* have been computed, their annotations are indexed.
  - *Index logo instances*: after the logo instances have been processed, their annotations are indexed.
  - *Update match index*: the *match* index is updated when we receive the validation from the crowd. This can happen after the validation of the logo instances (i.e., *matches* generated by the validated logo instance are affected) and after the validation of a *match*.

5.4.2 The Search Process

Together with the activities needed to collect, annotate and index videos and logo instances, our logo detection application provides to user the means to search for the occurrences of a trademark logo (i.e., a brand) in the *GroZi-120* video collection.
The user expresses his/her query in a textual form, simply providing to the application the name of a brand (e.g., Coca Cola). Then, the logo index is checked:

- IF the brand name is already present in the logo index, the brand name is used to query the *match* index and related *matches* are retrieved and presented in the user interface;
- OTHERWISE, this means that there are also no *matches* to display. Hence, we let the user to decide whether he/she want to start a new *Indexing process* fed with the searched brand name.

Figure 5.12 draws the use case activities within the *Search process*.

5.4.2.1 Query Management

The *Query Management* activity is simply composed by the *Query Analysis* stage, in which the brand name is subjected to linguistic processing (e.g., normalization, stemming).

5.4.2.2 Search Orchestration

The *Query Planning* and *Query Execution* activity are managed by SMILA framework, which provides the services to interact with the index.

5.4.2.3 Result Presentation

This activity is in charge of managing the results retrieved from the index. As the SMILA framework provides a single search engine on top of the *Solr* index, there is no need of *Result Aggregation* stage. The only process to be performed on the results is the transformation into a format that the user interface can consume (e.g., JSON) in order to produce the final presentation of the results.

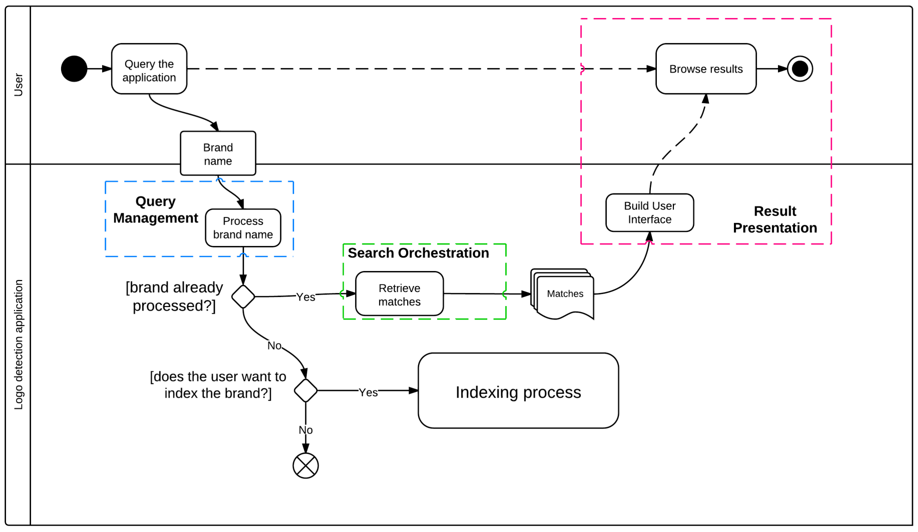


Figure 5.12: The use case *Search process*

In Chapter 6 we will provide an implementation of the logo detection application built on top of the SMILA framework.

6 Implementation and Evaluation

This chapter describes the actual implementation of the logo detection application, instantiating the application processes and the activities described in Chapter 5.
The application has been developed within the *CUbRIK* project[16] and it served as a *blueprint* for the forthcoming projects' applications.
The *Indexing* and the *Search* processes are implemented in the SMILA framework, while the *Human Enhancement process* is allocated to the *CrowdSearcher* framework (see section 2.3).
SMILA "*is a framework for creating scalable server-side systems that process large amounts of unstructured data in order to build applications in the area of search, linguistic analysis, information mining or similar*"[17]. It is an open-source project developed by *Attensity*[18] and part of the *EclipseRT* project[19].
In the following sections:

- we delve into the architecture of the logo detection demo, in the SMILA environment;
- we present the user interface by which end user can interact with the application in order to find the occurrences of a trademark in the *GroZi-120* video collection;
- we present the performance of the application in terms of indexing and searching time on the *GroZi-120* video collection.

[16]CUbRIK is an European research project, partially founded by the European Union, which aims to "introduce real innovative patterns inside the Multimedia search domain, proposing the paradigm of human-enhanced time-aware multimedia search, driven by openness at all levels", http://www.cubrikproject.eu.
[17]http://wiki.eclipse.org/SMILA/Documentation/Architecture_Overview
[18]http://www.attensity.com/home/
[19]http:/www.eclipse.org/eclipsert

6.1 Architecture

Prior to presenting the architecture and the components that implement the application processes (i.e., *Indexing*, *Human Enhancement* and *Search*), we introduce the SMILA architecture and the taxonomy that we will adopt in the presentation of the architecture.

6.1.1 SMILA Architecture and Taxonomy

As depicted in Figure 6.1, SMILA architecture is composed by two main distinguished parts:

- First, data has to be imported into the system and processed to build an search index or extract an ontology or whatever can be learned from the data.
- Second, the learned information is used to answer retrieval requests from users, for instance search requests.

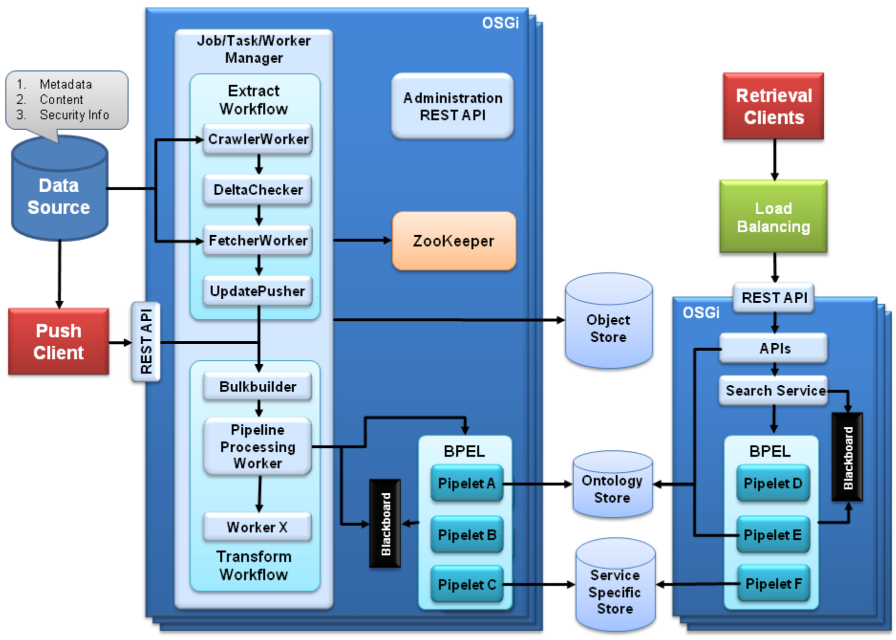


Figure 6.1: The SMILA framework architecture

In the first process data sources are crawled or an external client pushes the data from the source into the SMILA system (e.g., through the HTTP ReST API). Often, the data consist of a large number of documents (e.g., a file system, web site, or content management system) to be processed. In SMILA, each document is represented by a *record* (see Section 6.1.1.1), which describes the metadata of the document (name, size, access rights, authors, keywords, etc.) and its the original content.

To process large amounts of data, SMILA can distribute the work to be done on multiple SMILA nodes (i.e., machines). Therefore, an on-purpose component (i.e., the *bulkbuilder*) separates the incoming data into *bulks* of *records* of a configurable size. For each of these *bulks*, the *Job Manager* (see Section 6.1.1.3) creates a set of *tasks* for the *workers* in order to process the *bulks* and produce other *bulks*, which contain the results of processing.
When a new *worker* is available, it asks the *Task Manager* (see Section 6.1.1.4) for *tasks* to be done, it does the work and finally notifies the *Task Manager* about the result.
Workflows define which *workers* should process a *bulk* in what sequence. Whenever a *worker* finishes a *task* for a *bulk* successfully, the *Job Manager* can create follow-up *tasks* based on the *workflow* definition. In case a *worker* fails processing a *task* (e.g., because the process or machine crashes or because of a network problem), the *Job Manager* can decide to retry the *task* later and so ensure that the data is processed even in problematic conditions. The processing of the complete data set using such a *workflow* is called a *job run*.
Eventually, the final step of the asynchronous processing *workflow* will write the processed data to a target system, for instance, to a database or an index. The target system can be used to process retrieval requests which are being handled by the second part of the system. Such requests are coming from an external client application via the HTTP ReST API. They are usually of a synchronous nature, this means that the user sends a request and then waits until the results are ready.

6.1.1.1 Records

In SMILA, the *record* is the fundamental data structure that contains all the data to process (e.g., contents and metadata of a document or multimedia item). A *record* consists of *metadata* and optional *attachments*.

Metadata consists of typed values (literals) arranged in *Maps* (i.e., key-anything associations) and sequences (lists of anything). Values can be strings, long integers, double precision floating point numbers, booleans, dates (year, month, day) or datetimes (date and time of day, down to seconds). Maps and sequences can be nested arbitrarily, map keys are always strings. Notice that all metadata of one record is arranged in a single *Map*.

Attachments contain any binary content (e.g., the binary stream of a video), possibly of large size.

Figure 6.2 draws the SMILA data model. *Records* are created by the crawlers. The crawler iterates over the elements (e.g., videos) of the data source (e.g., video collection) and creates a *record* for each element and send them to SMILA.

6.1.1.2 Blackboard

The *blackboard* holds the *records* while they are pushed through a *pipeline*.
Pipelets are invoked with a *blackboard* instance and a list of IDs of *records* to process. The *pipelet* can then access the *blackboard* to get *record metadata* and *attachments*.

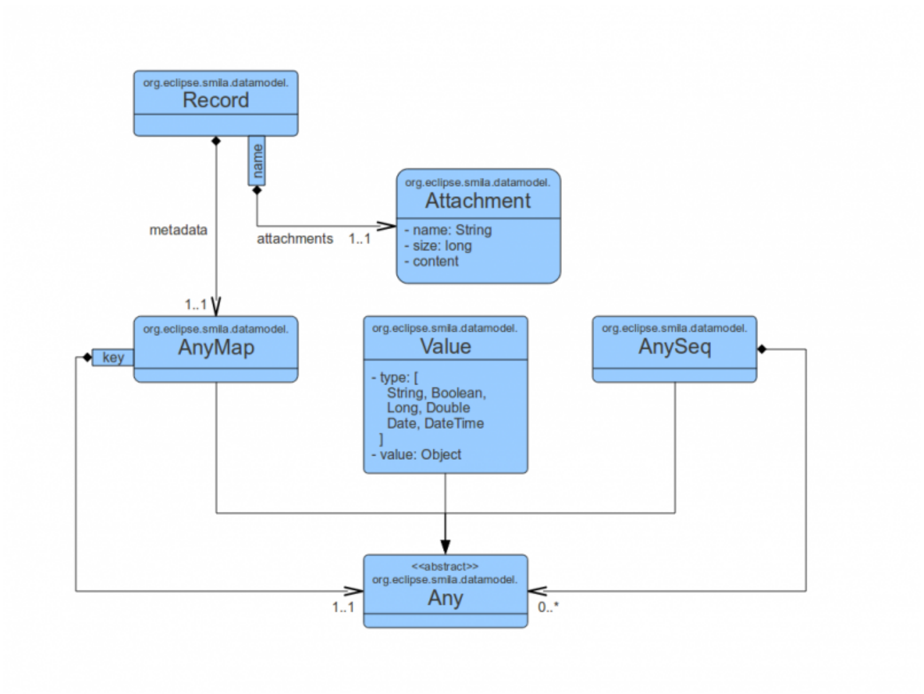


Figure 6.2: The SMILA data model

The *blackboard* hides the handling of *record* persistence from the other SMILA's services. The *blackboard* instance is released after the *pipeline* execution has been finished, for each *pipeline* execution a new *blackboard* instance is created.

6.1.1.3 Job Manager

The *Job Manager* controls the processing logic of *asynchronous workflows* in SMILA, by regulating the *Task Manager*, which in turn generates *tasks* and decides which *task* should be processed by which *worker* and when.

Asynchronous wokflows An *asynchronous workflow* consists of a set of *actions*. Each *action* connects the input and output *slots* of a *worker* to the appropriate *buckets*. A *bucket* is a virtual container of data objects of the same type. The most common data object type in SMILA is the *record bulk*, which is just a concatenated sequence of *records* (including *attachments*). *Record bulks* are created by the *Bulk Builder*.
The *Bulk Builder* is the standard entry *worker* for data to an *asynchronous workflow* in SMILA. It receives single *records* and combines them into one single *bulk* for further processing in an *asynchronous workflow*.
When a new data object arrives in a *bucket* connected to the input *slot* of a *worker*, a *task* is created for the worker to process this object and to produce data object with the results in the *buckets* connected to the output *slots*. Thus the workflow describes a data flow of the data objects through the *workers*. The workflow usually starts with a *worker* that creates data objects from the data which have been extracted from an external data source (e.g., extracted by a

Crawler worker) or by data pulled using the SMILA REST API. The workflow ends either when *workers* do not have output *buckets*, or the output *buckets* are not connected to input *slots* of other *workers*.

Synchronous workflows (BPEL pipelines) A *pipeline* is a *synchronous workflow* composed of components called *pipelets*. *Pipelets* that processes a list of *records* given as the input. Synchronous means that the invoker of the *pipeline* blocks until the execution has finished, and, if the processing is successful, a set of result *records* is returned that represents the result of the processing.
The *pipelets* lifecycle and configuration is managed by the workflow engine. An instance of a *pipelet* is not shared by multiple pipelines, even multiple occurrences of a *pipelet* in the same *pipeline* do not share the same instance. The configuration of each *pipelet* instance is included in the *pipeline* description.
The default SMILA workflow processing engine uses BPEL to describe *pipelines*.

6.1.1.4 Task Manager

The *Task Manager* is the component that administrates and delivers *tasks* during a *job run*. *Tasks* are stored in internal queues and are consumed by *workers* pulling them for processing. They are produced as initial *tasks* or as follow-up *tasks* when *workers* finish their processing of their current *task*.
The *Task Manager* guarantees that *tasks* are delivered at least once, but, in certain cases, *tasks* may be also delivered more than once (e.g., when process crashes or timeouts occur).

6.1.2 Architecture Overview

In this section we present the architecture of the trademark logo detection application implemented in the SMILA framework, in terms of *asynchronous workflows*, *workers* and *BPEL pipelines*.
Figure 6.3 depicts the overall architecture of the logo detection application. The core of the application are the *Indexing workflows*, which are in charge of:

- retrieving both videos and logo instances,
- process them,
- find *matches* between video key-frames and logo instances,
- send to *CrowdSearcher* logo instances and *matches* to be validated,
- put logo instance and *matches* in the *Solr* index;

Indeed, the *Search workflow*, highly based on the SMILA default *Search BPEL pipeline* is in charge of query the Solr index and produce the result set in a JSON format, to be consumed by the user interface.
Two fundamental components in our architecture are the *Data Service* and the SMILA *Record Storage*.

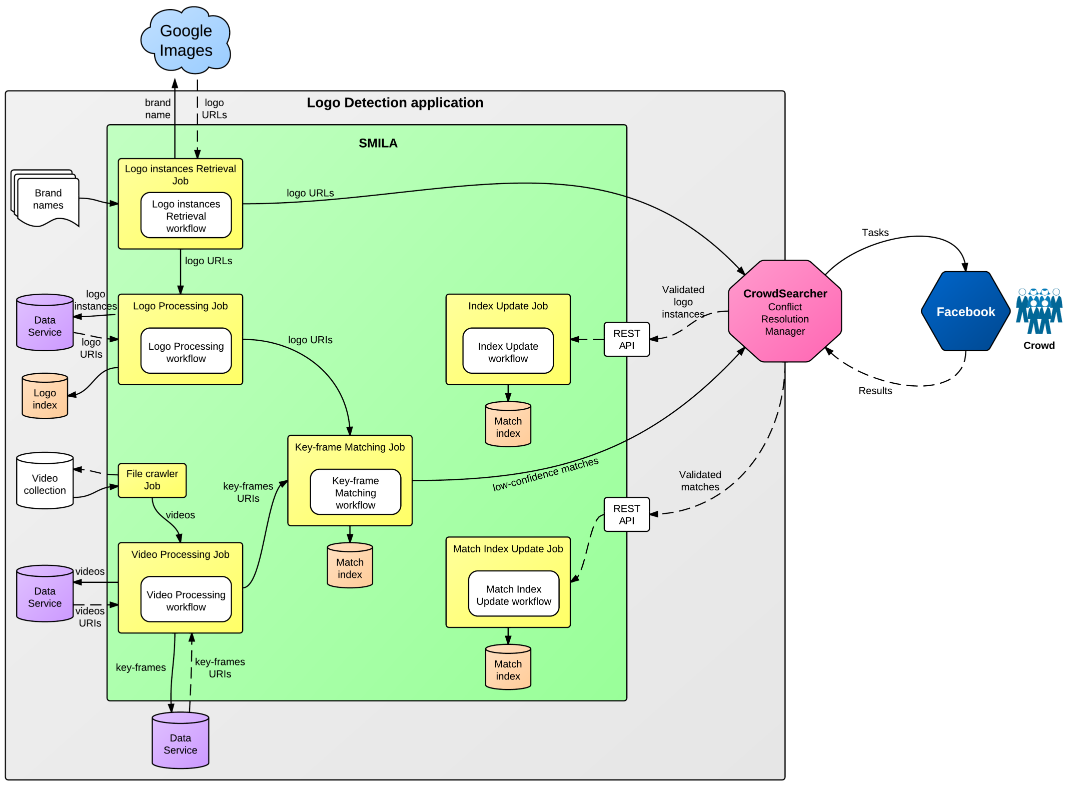


Figure 6.3: The logo detection application architecture

6.1.2.1 Data Service

The *Data Service* is a Web service that allow resources (i.e., videos, key-frames and logos) to be updated. Once a new resource is uploaded, the service returns an unique identifier, the URI (i.e., the *Media Locator* see Section 4.2.1). Querying the *Data Service* with an URI the related resource can be retrieved.
The interaction with the *Data Service* are managed by the *File Utils OSGi service* which provides to the SMILA components (i.e., *workers*, *pipelets*, etc.) the interfaces to upload and download files and to retrieve all the URIs of the resources uploaded in a certain directory.

6.1.2.2 Record Storage

The *Record Storage*[20] is a SMILA service, based on a Derby database, which provides an easy way to store and access *record* objects. It persists only the ID and the metadata of the *records*. Given the *record* ID it is possible to retrieve the corresponding *record*.

6.1.3 Indexing workflows

The *Indexing workflows* include six different workflows:

[20]http://wiki.eclipse.org/SMILA/Documentation/Record_Storage

- the *Logo instances Retrieval* workflow;
- the *Logo Processing* workflow;
- the *Video Processing* workflow;
- the *Key-frame Matching* workflow;
- the *Index Update* workflow;
- the *Match Index Update* workflow.

6.1.3.1 Events tracking

Among the retrieval, the analysis and the indexing of the multimedia *Content Objects* that allow us to perform the actual logo detection, we should also provide the means to track the events that occurs in the application, in order to exploit the benefits introduced by Human Computation. To do so, we insert in the *records* associated to the multimedia objects to be processed (i.e., videos, key-frames and logo instances) a metadata called *events*. Such metadata is a sequence of *event* maps; each *event* map correspond to a main operation in the workflows (e.g., processed video, found matches, etc.) and we store its timestamp, the type of the event and the multimedia item that is processed during the operation.
The *event records* are stored in the *Record Storage* and the multimedia objects *records* just keep the reference to the *event records* (i.e., the ID of the *record*) and their status at the instant of the *event* occurrence (e.g., the logo confidence at that time instant).
The events to be tracked are:

- *New processed video*: SIFT descriptors for all video key-frames have been computed and key-frames nave been uploaded to the Data Service;
- *New user query*: a new brand has been searched by the end user and its logo instances have been sent to the crowd in order to be validated;
- *Validated logo instance*: a logo instance has been validated by the crowd and its confidence has been sent back to SMILA;
- *Processed logo instances*: SIFT descriptors for a set of logo instances have been computed, logo instances have been uploaded on the *Data Service*, and logo metadata are put in the Solr index;
- *Found matches*: a set of relevant *matches* for a new key-frame, or a new logo instance, has been found;
- *Validated match*: a *match* has been validated by the crowd.

At the time of indexing of the *matches* in the Match core of the Solr index all the metadata of the related multimedia objects (video, key-frame and logo instance) are collapsed (see 6.1.3.5) in the *match record* and put in the Solr index. Among the metadata we find also the list of event record IDs associated to each multimedia object, a specific pipelet, the *MergeMatchEventsMetadata pipelet*, is in charge of the retrieval of the *event records* for the *Record Storage*, their ordering and their attachment to the *match record* to be indexed. In Listing 6.1 we report a sample of the *events list* attached to a *match record*.

```
"events" : [ {
     "ID" : "d8703002-08f6-4784-b61d-9e28ef64d608",
     "logoConfidence" : "",
     "matchConfidence" : "",
     "isRelevant" : "",
     "timestamp" : 1346242155868,
     "type" : "Processed video",
     "target" : "https://85.18.109.178:443/logodetection/DATA/nick/videos/avi/Shelf_1.avi",
     "matchEvent" : false,
     "description" : "Processed video: https://85.18.109.178:443/logodetection/DATA/nick/videos/avi/Shelf_1.avi"
   }, {
     "ID" : "8c6d8521-e4af-45cf-b9cc-13167717109c",
     "logoConfidence" : 0.484375,
     "matchConfidence" : "",
     "isRelevant" : "",
     "timestamp" : 1346244088788,
     "type" : "New searched brand",
     "target" : "hefty",
     "matchEvent" : false,
     "description" : "New searched brand: hefty"
   }, {
     "ID" : "1130d77f-c83e-4a26-889e-ccf05302bb7e",
     "logoConfidence" : 0.484375,
     "matchConfidence" : "",
     "isRelevant" : "",
     "timestamp" : 1346244122920,
     "type" : "Processed logos",
     "target" : "hefty",
     "matchEvent" : false,
     "description" : "Processed logos for brand: hefty"
   }, {
     "ID" : "7d906389-80f9-447d-b8d5-777abc3abe9b",
     "logoConfidence" : 0.484375,
     "matchConfidence" : 0.164557,
     "isRelevant" : "",
     "timestamp" : 1346244144062,
     "type" : "Found matches",
     "target" : "hefty",
     "matchEvent" : true,
     "description" : "Found matches"
   } ]
```

Listing 6.1: A sample of the *events* associated to a *match*

6.1.3.2 Logo instances retrieval workflow

The *Logo instances retrieval* workflow is triggered by a REST API call that allow to pull in the workflow a *record*, which contains in its metadata the name of the brand.
Here we report the *job* and workflow definitions in JSON format:

```
{
  "name":"LogoInstancesRetrievalJob",
  ...
  "workflow":"LogoInstancesRetrievalWorkflow"
}
```

Listing 6.2: *Logo instances retrieval* job definition

```
{
"name":"LogoInstancesRetrievalWorkflow",
  ...
"startAction":{
    "worker":"bulkbuilder",
    "output":{
      "insertedRecords":"importBucket"
    }
  },
  "actions":[
  {
    "worker":"pipeletProcessor",
    "parameters":
      {
       "pipeletName" :
        "eu.cubrikprj.pipelet.polmi.RetrieveLogoInstance.RetrieveLogoInstancesFromGooglePipelet",
       "googleContribution" : "0.5",
       "crowdContribution" : "0.5"
      },
    "input":{
       "input":"importBucket"
    },
    "output":{
       "output":"logoURLsBucket"
    }
  },
  {
    "worker":"SubmitLogosToCrowdWorker",
    "input":{
      "input":"logoURLsBucket"
    }
  }
  ]
}
```

Listing 6.3: *Logo instances retrieval* workflow definition

The workflow is composed by three *actions*:

- the *Bulk Builder*;
- a *Pipelet Processor worker*;
- a custom *worker*;

The *Bulk Builder* receives the brand name *record* for the REST API and build a *record bulk* to be processed by the workers within the asynchronous workflow.
The *Pipelet Processor worker* is a *worker* optimized to execute a single *pipelet*, without the overhead of a full pipeline.
The *RetrieveLogoInstancesFroomGoogle pipelet* contains the business logic needed to interact with the *GooglePhotoCrawler OSGi service*, which is in charge of query the *Google Images* search engine and to retrieve the URLs of top-32 images (i.e., the logo instances). The query sent to *Google Images* is simply composed by the brand name and by the term "logos" (e.g., "Coca cola logos"). Figure 6.4 draws the sequence diagram of the interaction between the *pipelet* and *Google Images*. Among the configurations of the *pipelet* we find two float parameters: *googleContribution* and *crowdContribution*.
This values are needed to compute the confidence of each logo instance, as shown in Equation 6.1:

$$\begin{aligned}Confidence =& \\ &= googleContribution * googleConfidence + \\ &+ crowdContribution * crowdConfidence \end{aligned} \tag{6.1}$$

The *googleConfidence* is the confidence related to the position of the logo instance among the top-32 images of *Google Images*.
The *crowdConfidence* is a confidence value proportional according to the votes received by the human performers. Obviously, at this stage such value is unknown and it is always set to zero; it will be updated when the application receives the outcomes of the crowdsourced validation task.
After the retrieval operation is performed, the *pipelet* creates a *record* for each logo instance URL retrieved and put them in the *logoURLs bucket*.
The logo instance record have the following metadata:

- *URL*: the URL returned by *Google Images*;
- *brand query*: the user query string;
- *brand name*: the normalized brand query;
- *confidence*;
- *events*: the list of events IDs associated to the logo instance.

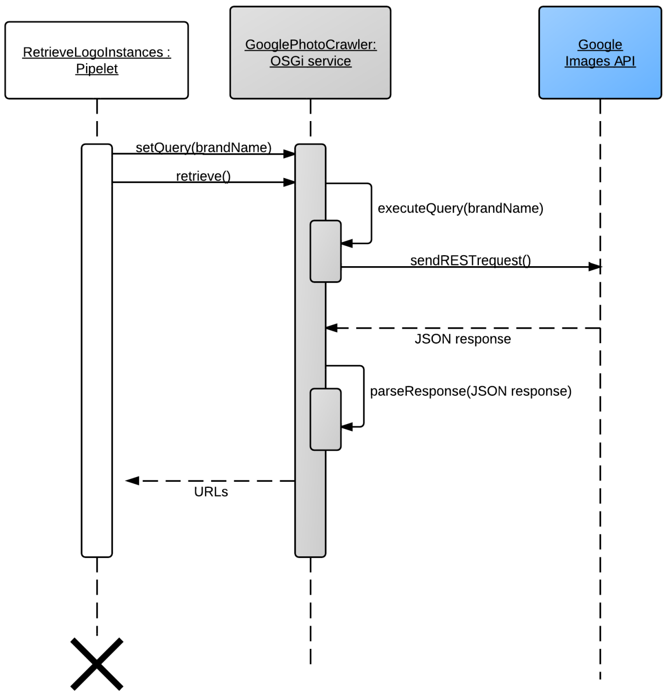


Figure 6.4: The interaction between the *RetrieveLogoInstancesFroomGoogle pipelet* and *Google Images*

As soon as the *pipelet* fills the *logoURLs bucket*, the *SubmitLogosToCrowd worker* is triggered. For each set of 32 logo instances record the worker creates a *Conflict Resolution Task* on the *CrowdSearcher Conflict Resolution Manager*. Together with the *SubmitLogosToCrowd worker*, the *logoURLs bucket* is linked to the *Logo processing* workflow, which we describe in the following section.

6.1.3.3 Logo processing workflow

The *Logo processing* workflow is in charge of the retrieval of the actual image file of the logo instance from the Web, of the computation and the storage of its SIFT descriptors and, finally, of the upload the logo image on the *Data Service* to obtain the universal reference to be further used in the subsequent workflows.

```
{
  "name":"LogoProcessingJob",
  ...
  "workflow":"LogoProcessingWorkflow"
}
```

Listing 6.4: *Logo processing* job definition

```
{
  "name":"LogoProcessingWorkflow",
  ...
  "startAction":{
    "worker":"pipelineProcessor",
    "parameters":
    {
       "pipelineName": "LogoProcessingPipeline"
    },
    "input":{
      "input":"logoURLsBucket"
    },
    "output":{
      "output":"imagesBucket"
    }
  }
}
```

Listing 6.5: *Logo processing* workflow definition

The *Logo processing* workflow is composed by a single *Pipeline processor worker* which executes the *LogoProcessing* BPEL pipeline, depicted in Figure 6.5, defined as follows:

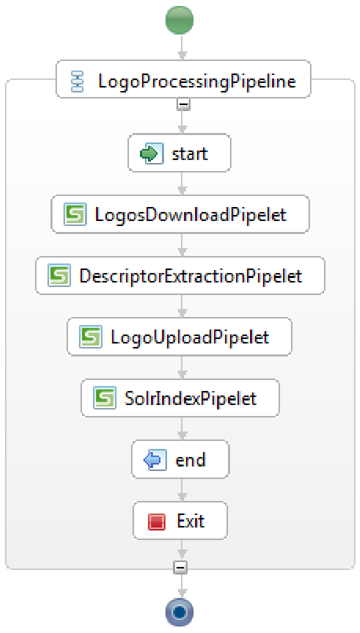


Figure 6.5: The *Logo processing* BPEL pipeline

```
<process name="LogoProcessingPipeline" ... >
...
  <sequence name="LogoProcessingPipeline">
    <receive name="start" partnerLink="Pipeline" portType="proc:ProcessorPortType"
      operation="process" variable="request" createInstance="yes" />

   <extensionActivity>
      <proc:invokePipelet name="LogosDownloadPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.LogosDownload.LogosDownloadPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
            <rec:Val key="outputDir">D:/logos</rec:Val>
         </proc:configuration>
      </proc:invokePipelet>
   </extensionActivity>

   <extensionActivity>
      <proc:invokePipelet name="DescriptorExtractionPipelet">
         <proc:pipelet
          class="eu.cubrikprj.pipelet.polmi.DescriptorExtraction.DescriptorExtractionPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
            <rec:Val key="imagesDir">D:/logos</rec:Val>
            <rec:Val key="descriptorsDir">D:/indexes</rec:Val>
            <rec:Val key="indexerPath">D:/CubrikIndexer/Indexer.exe</rec:Val>
         </proc:configuration>
      </proc:invokePipelet>
   </extensionActivity>

   <extensionActivity>
```

```
      <proc:invokePipelet name="LogoUploadPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.ImageUpload.LogoUploadPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
            <rec:Val key="serverAddress">https://85.18.109.178:443/logodetection/uploadurl.cgi
            </rec:Val>
            <rec:Val key="userID">polmi</rec:Val>
            <rec:Val key="password">logo_detection_2012</rec:Val>
            <rec:Val key="owner">nick</rec:Val>
         </proc:configuration>
      </proc:invokePipelet>
   </extensionActivity>

   <extensionActivity>
     <proc:invokePipelet name="SolrIndexPipelet">
        <proc:pipelet class="org.eclipse.smila.solr.index.SolrIndexPipelet" />
        <proc:variables input="request" output="request"/>
        <proc:configuration>
           <rec:Val key="ExecutionMode">ADD</rec:Val>
           <rec:Val key="CoreName">LogoCore</rec:Val>
           <rec:Seq key="CoreFields">
              <rec:Map>
                 <rec:Val key="FieldName">URI</rec:Val>
              </rec:Map>
              <rec:Map>
                 <rec:Val key="FieldName">brandName</rec:Val>
              </rec:Map>
           </rec:Seq>
       </proc:configuration>
     </proc:invokePipelet>
   </extensionActivity>
   ...
   <exit />
   </sequence>
</process>
```

Listing 6.6: *Logo processing* BPEL pipeline definition

The *Logo processing* pipeline is a sequence of the following *pipelets*:

- ***LogoDownload pipelet***: this *pipelet* is in charge of retrieving the logo instance image from the Web, through the URL returned by *Google Images*. The downloaded file is stored on the local file system and a reference to the local file (i.e., localURI matadata) is attached to the logo instance *record*.

- ***DescriptorExtraction pipelet***: this *pipelet* triggers a binary executable which extracts the SIFT descriptors from the logo instance image file. The SIFT descriptors are stored on the local system and the reference to the file is attached at the logo instance *record* (i.e., *SIFTdescriptorsLocalURI* metadata). The *pipelet* configurations includes the credential to access and modify the resources on the *Data Service* (i.e., *serverAddress*, *userID*, *password*, *owner*).

- ***LogoUpload pipelet***: at this stage the logo instance image is uploaded on the *Data Service* and the URI of the resource is returned. Upload and download operations are managed by the *File Utils OSGi service* (see Section 6.1.2.1). In order to save time we will send to the *Data Service* the URL of the logo instance instead of the image binary stream, a cgi executable on the HTTP service will retrieve the actual image file. The URI returned from the *Data Service* is used as unique reference to the logo instance in the application and also as new *record* ID of the logo instance *record*.

- ***SolrIndex pipelet***: the logo instance *records* are put in the *Logo core* of the *Solr index*. The *Logo core* allow to keep track of the processed logo instances and searched brand. The metadata to be stored are the brand name and the URL associated to the logo instance. The *Logo core* is configured to use as search field the one associated to the brand name metadata.

At the end of the pipeline logo instance *records* are put in the *imagesBucket bucket* to be forwarded to the matching workflow.

6.1.3.4 Video processing workflow

The *Video processing* workflow is in charge of the detection of the video key-frames, the extraction of JPEG images corresponding to the key-frames, the computation of the SIFT descriptors of each key-frame and the upload of the key-frame image files on the *Data Service*.
The workflow is triggered by the SMILA's filesystem crawler worker which pulls the video files in the BPEL pipeline.

```
{
  "name":"crawlFilesystem",
  "workflow":"fileCrawling",
  "parameters":{
    ...
    "dataSource":"file",
    "rootFolder":"D:/videos",
    "jobToPushTo":"VideoProcessingJob"
    }
}
...
{
  "name":"VideoProcessingJob",
  ...
  "workflow":"VideoProcessingWorkflow"
}
```

Listing 6.7: Filesystem crawler job definition

```
{
  "name":"VideoProcessingWorkflow",
  ...
  "startAction":{
    "worker":"bulkbuilder",
    "output":{
      "insertedRecords":"importBucket"
    }
  },
  "actions":[
  {
    "worker":"pipelineProcessor",
    "parameters":{
      "pipelineName": "VideoProcessingPipeline"
    },
    "input":{
      "input":"importBucket"
    },
    "output":{
      "output":"imagesBucket"
    }
  }
  ]
}
```

Listing 6.8: *Video processing* workflow definition

The workflow is composed by two *actions*:

- the *Bulk Builder*: which receives the video records from the *File System Crawler worker* and creates the *record bulks*;
- a *Pipeline processor worker* which executes the *VideoProcessing* BPEL pipeline.

The *VideoProcessing* BPEL pipeline, presented in Figure 6.6, is defined as follows:

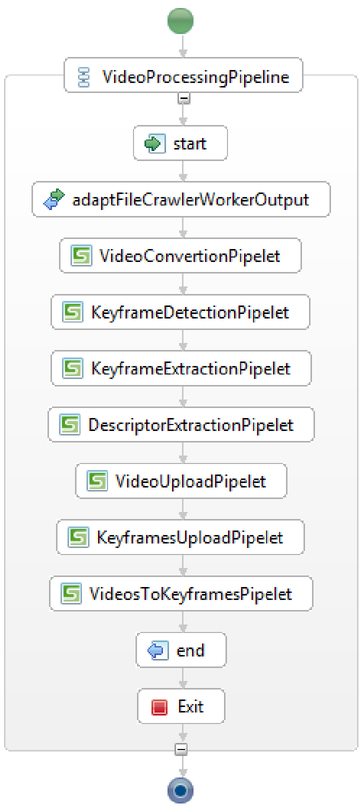


Figure 6.6: The *Video processing* BPEL pipeline

```
<process name="VideoProcessingPipeline"...>
...
  <sequence name="VideoProcessingPipeline">
    <receive name="start" partnerLink="Pipeline" portType="proc:ProcessorPortType"
      operation="process" variable="request" createInstance="yes" />

    <invoke name="adaptFileCrawlerWorkerOutput" inputVariable="request"
    partnerLink="AdaptFileCrawlerWorkerOutput"
      outputVariable="request" operation="process" portType="proc:ProcessorPortType" />

    <extensionActivity>
      <proc:invokePipelet name="VideoConvertionPipelet">
        <proc:pipelet class="eu.cubrikprj.pipelet.eng.video.VideoConvertionPipelet" />
        <proc:variables input="request" output="request" />
        <proc:configuration>
           <rec:Val key="outputDir">D:/videos</rec:Val>
           <rec:Val key="extensions">ogv avi webm mp4</rec:Val>
         </proc:configuration>
      </proc:invokePipelet>
    </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="KeyframeDetectionPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.KeyframeDetection.KeyframeDetectionPipelet" />
         <proc:variables input="request" output="request" />
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="FrameExtractionPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.FrameExtraction.FrameExtractionPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
             <rec:Val key="outputDir">D:/frames</rec:Val>
         </proc:configuration>
       </proc:invokePipelet>
     </extensionActivity>
     <extensionActivity>
       <proc:invokePipelet name="DescriptorExtractionPipelet">
         <proc:pipelet
         class="eu.cubrikprj.pipelet.polmi.DescriptorExtraction.DescriptorExtractionPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
            <rec:Val key="imagesDir">D:/frames</rec:Val>
            <rec:Val key="descriptorsDir">D:/indexes</rec:Val>
            <rec:Val key="indexerPath">D:/CubrikIndexer/Indexer.exe</rec:Val>
         </proc:configuration>
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="VideoUploadPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.eng.video.VideoUploadPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
            <rec:Val key="serverAddress">https://85.18.109.178:443/logodetection/uploadfile.cgi</rec:Val>
            <rec:Val key="userID">nick</rec:Val>
            <rec:Val key="password">logo_detection_2012</rec:Val>
            <rec:Val key="videoBaseURI">http://85.18.109.178:82/logodetection/DATA/videos</rec:Val>
         </proc:configuration>
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="KeyframesUploadPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.ImageUpload.KeyframesUploadPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
             <rec:Val
              key="serverAddress">https://85.18.109.178:443/logodetection/uploadfile.cgi</rec:Val>
             <rec:Val key="userID">polmi</rec:Val>
             <rec:Val key="password">logo_detection_2012</rec:Val>
             <rec:Val key="owner">nick</rec:Val>
         </proc:configuration>
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="VideosToKeyramesPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.ImageUpload.VideosToKeyramesPipelet" />
         <proc:variables input="request" output="request" />
          <proc:configuration/>
       </proc:invokePipelet>
     </extensionActivity>
...
    <exit />
  </sequence>
</process>
```

Listing 6.9: *Video processing* BPEL pipeline definition

Prior to the execution of its *pipelets* the *Video processing* pipeline need to invoke another pipeline, the *AdaptCrawlerWorkerOutput* pipeline: the video records pulled by the *File System Crawler worker* has to be adapted in order to be processed by the other *pipelets*.
Once the adaptation operation is performed the following *pipelets* can be executed:

1. ***VideoConversion pipelet***: the pipelet is in charge to convert the .AVI video into formats suitable for the visualization in a HTML 5 environment (e.g., .MP4, .OGV, .WEBM, .OGG, etc.). the alternative format videos are temporary stored locally and the local URI to each video are appended to the video *record*.

2. ***KeyframeDetection pipelet***: this *pipelet* invokes the *video segmenter* component in order to retrieve the numbers of the key-frames within a video. The list of the key-frames' numbers are appended to the video record in the *Seq* of *Maps* called *keyframes*. Figure 6.7 depicts the interaction between the *pipelet* and the component.

3. ***FrameExtraction pipelet***: the *pipelet* receives the video *records*, each on including the list of the key-frames. For each detected key-frame the *pipelet* invokes the *ffmpeg* component passing the local URI of the video

file and the number of the frame (i.e., the key-frame) to extract. The *ffmpeg* component store the .JPEG image file of the key-frame on the local file system, and the path to the image is appended, in the *keyframes Seq* element, to the *Map* corresponding to the key-frame under the name of *localURI*.

4. ***DescriptorExtraction pipelet***: as well as the same *pipelet* in the *Logo processing* pipeline, this *pipelet* is in charge of invoking the binary executable which extracts the SIFT descriptors for each key-frame image. Figure 6.8 depicts the interaction between the *pipelet* and the SIFT descriptors extractor component.

5. ***VideoUpload pipelet***: the *pipelet* is in charge to upload on the Data Service the original video and its alternative versions in the formats declared in the *VideoConversion pipelet*, for each video uploaded the *Data Service* return a URI to be appended at the video *record*.

6. ***KeyframesUpload pipelet***: as for the *LogoInstanceUpload pipelet*, this *pipelet* uses the *File Utils OSGi service* to upload on the *Data Service* the key-frames. For each key-frame the *Data Service* returns a URI that is appended to the key-frame metadata in the video *record*.

7. ***VideosToKeyframes pipelet***: this *pipelet* has a twofold purpose:

   - create a new SMILA *record* for each key-frame detected and processed;
   - store the video *records* in the SMILA *Record Storage*.

   Each video *record* is parsed and for each key-frame in the key-frames' *Seq* a new *record* is created. The key-frame *record* holds the following metadata:

   - *URI*: the reference to the image returned by the *Data Service*, to be used also as *record* ID.
   - *Video record ID*: the ID of the video *record* associated to the video to which the key-frame belongs to.
   - *Frame number*: the frame number returned by the *video segmenter* component.
   - *Frame instant*: the time instant in the video in which the frame appears.
   - *SIFT descriptors local URI*: the path to the SIFT descriptors file on the local file system.

The new key-frames *records* are put in the *blackboard* and in the workflow *output bucket* (i.e., the *imagesBucket bucket*) in order to be processed in the subsequent matching workflow. On the contrary, video *records* are removed from the *blackboard* and stored in the *Record Storage*.

6.1.3.5 Key-frame matching workflow

The *Key-frame matching* workflow is the application stage where the logo detection task is exploited. The workflow receives in input *records* both from the *Video processing* pipeline and the *Logo processing* pipeline.

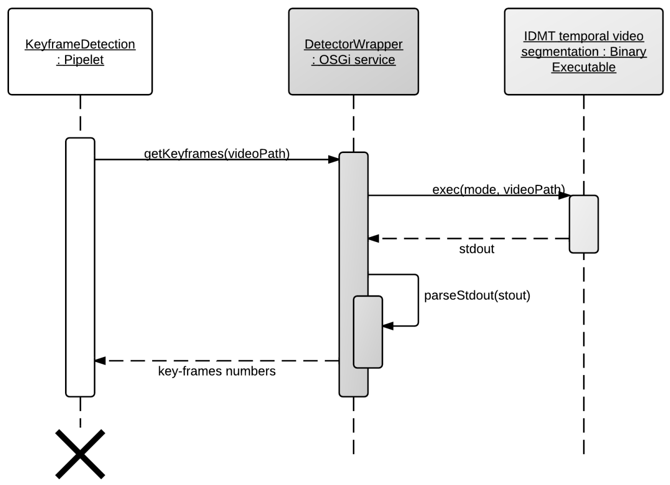

KeyframeDetection : Pipelet
DetectorWrapper : OSGi service
IDMT temporal video segmentation : Binary Executable
getKeyframes(videoPath)
exec(mode, videoPath)
stdout
parseStdout(stout)
key-frames numbers

Figure 6.7: The interaction between the *Key-frameDetection pipelet* and the video segmenter component

```
{
   "name":"KeyframeMatchingJob",
   ...
   "workflow":"KeyframeMatchingWorkflow"
}
```

Listing 6.10: *Key-frame matching* job definition

```
{
   "name":"KeyframeMatchingWorkflow",
   "startAction":{
      "worker":"pipelineProcessor",
      "parameters":{
         "pipelineName":"KeyframeMatchingPipeline"
       },
       "input":{
          "input":"imagesBucket"
       },
       "output":{
          "output":"matchesBucket"
       }
   },
   "actions":[
   {
      "worker":"SubmitMatchesToCrowdWorker",
      "input":{
         "input":"matchesBucket"
      }
   }
   ]
}
```

Listing 6.11: *Key-frame matching* workflow definition

The workflow is composed by a BPEL pipeline, the *KeyframeMatching* pipeline, and a custom *worker*, the *SubmitMatchesToCrowd worker*.
The *KeyframeMatching* pipeline, depicted in Figure 6.9, is defined as follows:

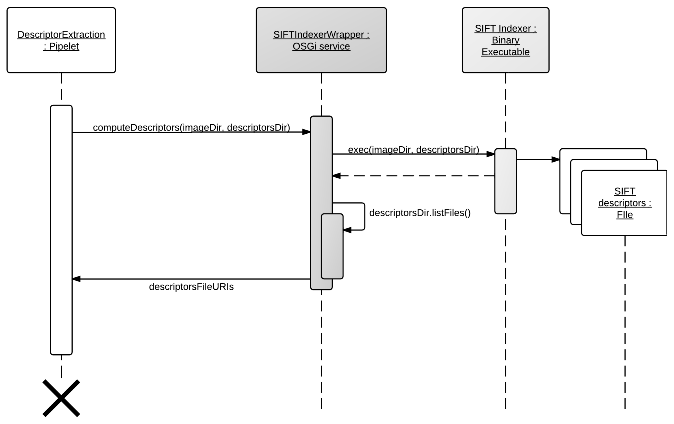

DescriptorExtraction : Pipelet
SIFTIndexerWrapper : OSGi service
SIFT Indexer : Binary Executable
computeDescriptors(imageDir, descriptorsDir)
exec(imageDir, descriptorsDir)
SIFT descriptors : File
descriptorsDir.listFiles()
descriptorsFileURIs

Figure 6.8: The interaction between the *DescriptorExtraction pipelet* and the binary executable

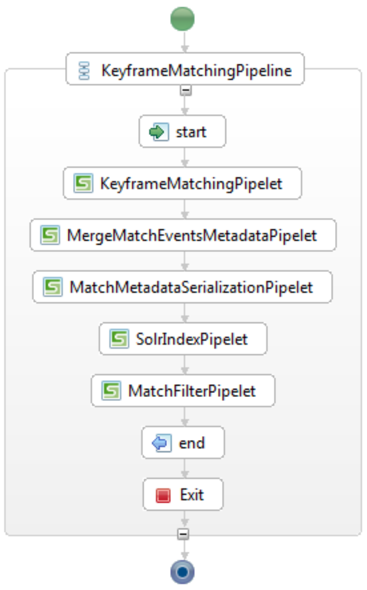


Figure 6.9: The *Key-frame matching* BPEL pipeline

```
<process name="KeyframeMatchingPipeline" ...>
...
    <sequence name="KeyframeMatchingPipeline">
    <receive name="start" partnerLink="Pipeline" portType="proc:ProcessorPortType"
      operation="process" variable="request" createInstance="yes" />

    <extensionActivity>
       <proc:invokePipelet name="KeyframeMatchingPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.KeyframeMatching.KeyframeMatchingPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
             <rec:Val key="matcherPath">D:/Matcher/Matcher.exe</rec:Val>
             <rec:Val key="matcherThreshold">0.001</rec:Val>
             <rec:Val key="serverAddress">https://85.18.109.178:443/logodetection/listfolder.cgi</rec:Val>
             <rec:Val key="userID">polmi</rec:Val>
             <rec:Val key="password">logo_detection_2012</rec:Val>
             <rec:Val key="owner">nick</rec:Val>
         </proc:configuration>
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="MergeMatchEventsMetadataPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MergeMatchEventsMetadataPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration/>
       </proc:invokePipelet>
     </extensionActivity>

      <extensionActivity>
       <proc:invokePipelet name="MatchMetadataSerializationPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MatchMetadataSerializationPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration/>
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
         <proc:invokePipelet name="SolrIndexPipelet">
         <proc:pipelet class="org.eclipse.smila.solr.index.SolrIndexPipelet" />
             <proc:variables input="request" output="request"/>
             <proc:configuration>
                <rec:Val key="ExecutionMode">ADD</rec:Val>
                <rec:Val key="CoreName">MatchCore</rec:Val>
                <rec:Seq key="CoreFields">
                   <rec:Map>
                      <rec:Val key="FieldName">matchID</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">frameURI</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">frameInstant</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">videoURI</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">videoOgv</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">videoMp4</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">videoWebm</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">videoTitle</rec:Val>
                       </rec:Map>
                       <rec:Map>
                      <rec:Val key="FieldName">videoPreview</rec:Val>
                       </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">logoURI</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">logoName</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">brandName</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">boundingBoxCoordinatesSerialized</rec:Val>
                   </rec:Map>
                   <rec:Map>
                      <rec:Val key="FieldName">eventsSerialized</rec:Val>
                  </rec:Map>
               </rec:Seq>
            </proc:configuration>
        </proc:invokePipelet>
    </extensionActivity>

    <extensionActivity>
       <proc:invokePipelet name="MatchFilterPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.KeyframeMatching.MatchFilterPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
             <rec:Val key="lowConfidenceThreshold">0.05</rec:Val>
             <rec:Val key="highConfidenceThreshold">0.1</rec:Val>
         </proc:configuration>
       </proc:invokePipelet>
     </extensionActivity>

    <reply name="end" partnerLink="Pipeline" portType="proc:ProcessorPortType" operation="process"
      variable="request" />
    <exit />
  </sequence>
</process>
```

Listing 6.12: *Key-frame matching* BPEL pipeline definition

The pipeline is the sequence of the following pipelets:

- ***KeyframeMatching pipelet***: this *pipelet* is the core of the logo detection application. It receives in input *record bulks*, from the *imagesBucket bucket*, that can be composed both by key-frame *record* and logo instances *record*. According to the type of each *record* the business logic of the *pipelet* slightly changes: if the current *record* is related to a key-frame the pipelet queries the *Data Service* to retrieve the list of the URI of the logo instances already processed (i.e., the logo upload is performed at the end of the *Logo processing* pipeline). Conversely, if the *record* relates to a logo instance, the pipelet queries the *Data Service* to retrieve the list of the URI of the processed video key-frames.
  Once the set of URIs is downloaded from the *Data Service*, each URI is used to retrieve from the *Record Storage* the corresponding record (e.g., a logo instance record if the *pipelet* is processing a key-frame *record*).
  From the current *record* and the *record* retrieved from the Record Storage the *SIFTdescriptorsLocalURI* metadata (i.e., the path to the SIFT descriptors files) are read and passed, among a minimum accepted threshold (i.e., the matcher threshold), to the binary executable in charge of the SIFT descriptors matching. As depicted in Figure 6.10, the binary returns an image similarity value (i.e., the *match confidence* between 0 and 1) and four coordinates representing the edges of the bounding box that should surround the area in the key-frame that includes the logo instance.
  When the *match confidence* returned is higher than 0 a new match has been found. For each found match a new *record* is created merging the metadata coming from the key-frame, the video and the logo instance. To each match record we also append the list of events *record* IDs associated to the related video and the related logo instance. Finally, the match records are stored in the SMILA *Record Storage* for further updates (see Sections 6.1.3.6 and 6.1.3.7).

- ***MergeMatchEventsMetadata pipelet***: this *pipelet* is invoked to retrieve from the *Record Storage* the event *records* related to each match, and to add the event *records* metadata to the match *record*.

- ***MatchMetadataSerialization pipelet***: some metadata in the match record cannot be indexed in the *Solr index* as they are: the *Solr index*, indeed, accept just plain text or numerical values. Thus, match metadata like the list of the events, which is a sequence of *Maps*, cannot be indexed. To allow the indexing of the events the sequence of *Maps* is serialized into a JSON and appended to the *record* as a textual metadata called *eventsSerialized*. The same problem arises for the four coordinates of the bounding box, which are serialized in a JSON text and put in a new match metadata, called *boundingBoxCoordinatesSerialized*.

- ***SolrIndex pipelet***: the match *records* are put in the *Match core* of the *Solr Index*. The following metadata are stored:
  - *matchID*: the ID of the *record*, to be used to retrieve it from the *Record Storage*;
  - *frameURI*: the URI of the key-frame image on the *Data Service*;

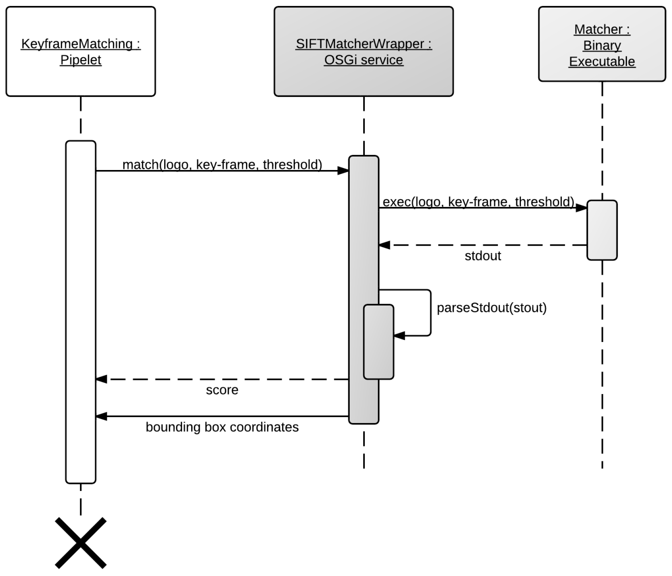

KeyframeMatching : Pipelet
SIFTMatcherWrapper : OSGi service
Matcher : Binary Executable
match(logo, key-frame, threshold)
exec(logo, key-frame, threshold)
stdout
parseStdout(stout)
score
bounding box coordinates

Figure 6.10: The interaction between the *KeyframeMatching pipelet* and the Matcher component

- *frameInstant*: the millisecond in which the key-frame appears in the video;
- *videoURI*: the URI of the video on the *Data Service*;
- *videoOgv*, *videoMp4*, *videoWebm*: the references to the HTML 5 compliant video formats on the *Data Service*;
- *videoPreview*: the URI of the key-frame to be used in the preview of the video;
- *videoTitle*: the title of the video;
- *logoURI*: the URI of the logo instance image on the *Data Service*;
- *logoName*: a name associated to the logo instance (e.g., the file name);
- *brandName*: the brand name associated to the logo instance;
- *boundingBoxCoordinatesSerialized*
- *eventsSerialized*

The *Match core* is configured to use as search field the one associated to the brand name metadata.

- ***MatchFilter pipelet***: this *pipelet* is in charge to select the matches to be forwarded to the *SubmitMatchesToCrowd worker*. The *pipelet* receive as configuration two threshold: the *high-confidence* threshold and the *low-confidence* one.
  Matches having a confidence above the *high-confidence* threshold are considered as GOOD matches, while matches in between the *high-confidence* threshold and the *low-confidence* one are considered *Uncertain* matches.

Finally, matches with a confidence below the *low-confidence* threshold are classified as REJECTED matches (see Section 5.10). Just the UNCERTAIN matches are kept in the *blackboard* and forwarded to the subsequent *worker*.

The *SubmitMatchesToCrowd worker* arises a *Conflict Resolution Task* for each set of UNCERTAIN matches on the *CrowdSearcher Conflict Resolution Manager*.

6.1.3.6 Index update workflow

When the human performers complete the logo instance validation tasks, *CrowdSearcher* triggers this workflow, sending back to SMILA the results. Once crowdsourced tasks results (i.e., the *logo confidences*) are collected, logo instances *records* and match *records* need to be updated both in the *Record Storage* and in the *Solr index*.

```
{
   "name":"IndexUpdateJob",
   ...
   "workflow":"IndexUpdateWorkflow"
}
```

Listing 6.13: *Index update* job definition

```
{
   "name":"IndexUpdateWorkflow",
   "parameters":{
      "pipelineRunBulkSize":"30"
   },
   "startAction":{
      "worker":"bulkbuilder",
      "output":{
         "insertedRecords":"importBucket"
      }
   },
   "actions":[
      {
      "worker":"pipelineProcessor",
      "parameters":{
         "pipelineName": "IndexUpdatePipeline"
      },
      "input":{
         "input":"importBucket"
      }
   }
   ]
}
```

Listing 6.14: *Index update* workflow definition

CrowdSearcher interacts with SMILA workflows through the SMILA REST API, thus the *Bulk Builder* worker is needed to build the *record bulks* to be processed by the BPEL pipeline, as shown in Figure 6.11.

```
<process name="IndexUpdatePipeline" ...>
...
    <sequence name="IndexUpdatePipeline">
    <receive name="start" partnerLink="Pipeline" portType="proc:ProcessorPortType"
      operation="process" variable="request" createInstance="yes" />

    <extensionActivity>
       <proc:invokePipelet name="LogoRecordsUpdatePipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.LogoRecordsUpdatePipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration>
             <rec:Val key="googleContribution">0.5</rec:Val>
             <rec:Val key="crowdContribution">0.5</rec:Val>
         </proc:configuration>
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="MatchRecordsUpdatePipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MatchRecordsUpdatePipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration/>
       </proc:invokePipelet>
     </extensionActivity>

   <extensionActivity>
       <proc:invokePipelet name="MergeMatchEventsMetadataPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MergeMatchEventsMetadataPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration/>
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="MatchMetadataSerializationPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MatchMetadataSerializationPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration/>
       </proc:invokePipelet>
    </extensionActivity>

    <extensionActivity>
      <proc:invokePipelet name="SolrIndexPipelet">
         <proc:pipelet class="org.eclipse.smila.solr.index.SolrIndexPipelet" />
         <proc:variables input="request" output="request"/>
         <proc:configuration>
           <rec:Val key="ExecutionMode">ADD</rec:Val>
           <rec:Val key="CoreName">MatchCore</rec:Val>
            <rec:Seq key="CoreFields">
               <rec:Map>
                 <rec:Val key="FieldName">matchID</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">frameURI</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">frameInstant</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">videoURI</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">videoOgv</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">videoMp4</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">videoWebm</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">videoTitle</rec:Val>
                   </rec:Map>
                   <rec:Map>
                 <rec:Val key="FieldName">videoPreview</rec:Val>
                   </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">logoURI</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">logoName</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">brandName</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">boundingBoxCoordinatesSerialized</rec:Val>
               </rec:Map>
               <rec:Map>
                 <rec:Val key="FieldName">eventsSerialized</rec:Val>
               </rec:Map>
            </rec:Seq>
         </proc:configuration>
      </proc:invokePipelet>
    </extensionActivity>

    <reply name="end" partnerLink="Pipeline" portType="proc:ProcessorPortType" operation="process" variable="request" />
    <exit />
  </sequence>
</process>
```

Listing 6.15: *Index update* BPEL pipeline definition

The *IndexUpdate* pipeline is composed by the following *pipelets*:

- ***LogoRecordsUpdate pipelet***: *CrowdSearcher* returns a *record* for each validated logo instance. Each *record* hold the URL of the logo instance and the logo confidence computed thank to human performers votes (see Section 4.3.2.4). The URL allow us to retrieve the logo instance *record* stored in the *Record Storage* and hence the logo instance URI (if in the meanwhile the logo instance have been processed). The retrieved logo in-

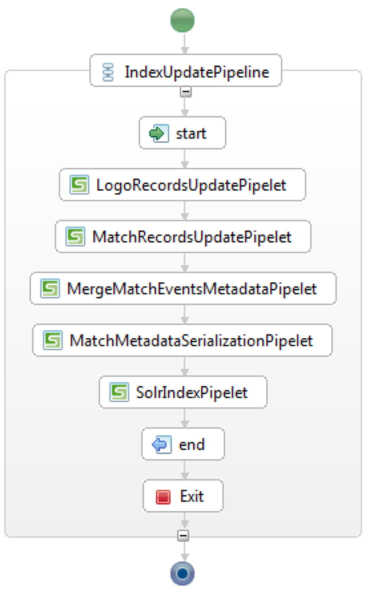

IndexUpdatePipeline
start
LogoRecordsUpdatePipelet
MatchRecordsUpdatePipelet
MergeMatchEventsMetadataPipelet
MatchMetadataSerializationPipelet
SolrIndexPipelet
end
Exit

Figure 6.11: The *Index update* BPEL pipeline

stance record is updated by appending VALIDATED LOGO event ID and the logo confidence metadata couple in the *record* events list. The updated logo instance *records* are put in the *blackboard* and forwarded to the subsequent *pipelets*.

- ***MatchRecordsUpdate pipelet***: this *pipelet* is in charge of updating the match *records* associated to the validated logo instances. The update simply consists in appending to the match events, the VALIDATED LOGO event ID and the *logo confidence*.

- ***MergeMatchEventsMetadata pipelet***: as in the key-frame matching pipeline, this pipelet is used to retrieve the events metadata from the *Record Storage* and adding them to the match events list.

- ***MatchMetadataSerialization pipelet***: before putting the update match records in the *Solr index*, some metadata (i.e., bounding box coordinates and events sequence) need to be serialized in JSON format

- ***SolrIndex pipelet***: the updated match *records* are put in the *Solr index*. The fields stored in the index are the same declared in the pipeline.

6.1.3.7 Match index update workflow

At the end of the *Key-frame matching* workflow, UNCERTAIN matches are sent to *CrowdSearcher* in order to be validated by human performers. When the performers fulfil the tasks, the results are pulled back to SMILA, through the REST API, as well as in the case of the logo instance validation. The result of the crowsourced tasks is a Boolean judgement on the relevance of the match couple, key-frame and logo instance (see Section 4.3.2.4). Hence, the match records stored in the *Solr index*, associated to the matches validated by the crowd, need to be updated.

```
{
  "name":"MatchIndexUpdateJob",
  ...
  "workflow":"MatchIndexUpdateWorkflow"
}
```

Listing 6.16: *Match index update* job definition

```
{
         "name":"MatchIndexUpdateWorkflow",
         "parameters":{
      "pipelineRunBulkSize":"30"
     },
  "startAction":{
    "worker":"bulkbuilder",
    "output":{
        "insertedRecords":"importBucket"
       }
  },
  "actions":[
  {
    "worker":"pipelineProcessor",
    "parameters":
    {
      "pipelineName": "MatchIndexUpdatePipeline"
    },
    "input":{
      "input":"importBucket"
    }
  }
  ]
}
```

Listing 6.17: *Match index update* workflow definition

As always when a SMILA workflow is triggered by the REST API the start action of the workflow has to be the *Bulk Builder worker*.
The subsequent action is a pipeline processor worker in charge of executing the *MatchIndexUpdate* BPEL pipeline presented in Figure 6.12:

```
<process name="MatchIndexUpdatePipeline" ...>
    ...
    <sequence name="MatchIndexUpdatePipeline">
    <receive name="start" partnerLink="Pipeline" portType="proc:ProcessorPortType"
      operation="process" variable="request" createInstance="yes" />

     <extensionActivity>
      <proc:invokePipelet name="MatchIndexUpdatePipelet">
        <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MatchIndexUpdatePipelet" />
        <proc:variables input="request" output="request"/>
        <proc:configuration/>
      </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
       <proc:invokePipelet name="MergeMatchEventsMetadataPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MergeMatchEventsMetadataPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration/>
       </proc:invokePipelet>
     </extensionActivity>

      <extensionActivity>
       <proc:invokePipelet name="MatchMetadataSerializationPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MatchMetadataSerializationPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration/>
       </proc:invokePipelet>
     </extensionActivity>

     <extensionActivity>
      <proc:invokePipelet name="SolrIndexPipelet">
        <proc:pipelet class="org.eclipse.smila.solr.index.SolrIndexPipelet" />
        <proc:variables input="request" output="request"/>
        <proc:configuration>
          <rec:Val key="ExecutionMode">ADD</rec:Val>
          <rec:Val key="CoreName">MatchCore</rec:Val>
          <rec:Seq key="CoreFields">
            <rec:Map>
                <rec:Val key="FieldName">matchID</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">frameURI</rec:Val>
                </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">frameInstant</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">videoURI</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">videoOgv</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">videoMp4</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">videoWebm</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">videoTitle</rec:Val>
                </rec:Map>
                <rec:Map>
                <rec:Val key="FieldName">videoPreview</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">logoURI</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">logoName</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">brandName</rec:Val>
            </rec:Map>
            <rec:Map>
                <rec:Val key="FieldName">boundingBoxCoordinatesSerialized</rec:Val>
            </rec:Map>
            <rec:Map>
               <rec:Val key="FieldName">eventsSerialized</rec:Val>
            </rec:Map>
            </rec:Seq>
        </proc:configuration>
      </proc:invokePipelet>
    </extensionActivity>

    <reply name="end" partnerLink="Pipeline" portType="proc:ProcessorPortType" operation="process"
      variable="request" />
    <exit />
  </sequence>
</process>
```

Listing 6.18: *Match index update* BPEL pipeline definition

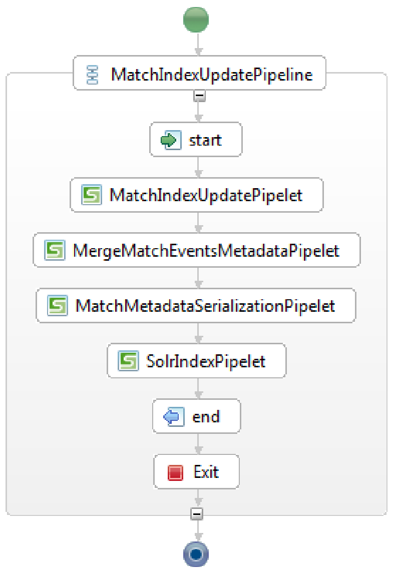


Figure 6.12: The *Match Index update* BPEL pipeline

The BPEL pipeline is the sequence of the following *pipelets*:

- ***MatchIndexUpdate pipelet***: the *pipelet* receives form the *Bulk Builder* the set of matches validated by the crowd. The *records* include the *matchID* and the Boolean judgement of the crowd. The *matchID* is used to retrieve the match *record* from the *Record Storage*.
- ***MergeEventsMetadata pipelet***: see Section 6.1.3.5.
- ***MatchMetadataSerialization pipelet***: see Section 6.1.3.5.
- ***SolrIndex pipelet***: see Section 6.1.3.5.

6.1.4 Search workflow

When the user sends a query (i.e., the brand name) through the logo detection user interface, the request is managed by a servlet executed in the SMILA environment.
The servlet triggers a BPEL pipeline (i.e., the *LogosSearch* pipeline) pulling a single *record*, containg in its metadata the query brand name. The *LogosSearch* pipeline, depicted in Figure 6.13, is in charge of forwarding the query brand name to the *Logo core* in the *Solr index*. This operation is needed to check if at least one logo instance for the query brand name has been already processed or not. If the brand name is not contained in the *Logo core*, then also no matches related to the query brand name can be found, thus the logo detection application returns to the user a message in which asks him/her whether he/she wants to start the processing of the brand name.
Instead, if the brand name is found in the *Logo core* then the query is forwarded to the *Match core* by the *LogoDetectionSearch* pipeline, in order to collect the found matches. The *LogoDetectionSearch* pipeline is drawn in Figure 6.14
The result matches are appended to the result *record* and sent back to the servlet.
In both BPEL pipelines, we use the default SMILA *Solr search pipelet* to query the *Solr cores*. To select the proper *core* (i.e., either *Logo core* or *Match core*) we simply set the *indexname* parameter in the *pipelet* configuration.
When dealing with the match records, we should remember that some metadata have been serialized (see Section 6.1.3.5), thus, before returning the result *records*, we should deserialize those metadata. In order to do so, we invoke the *MatchMetadataDeserialization* pipelet.

```
<process name="LogosSearchPipeline" ...>
 ...
  <sequence name="LogosSearchPipeline">
    <receive name="start" partnerLink="Pipeline" portType="proc:ProcessorPortType"
      operation="process" variable="request" createInstance="yes" />

    <extensionActivity>
      <proc:invokePipelet name="invokeSolrSearchPipelet">
        <proc:pipelet class="org.eclipse.smila.solr.search.SolrSearchPipelet" />
        <proc:variables input="request" output="request" />
        <proc:configuration>
            <rec:Val key="indexname">LogoCore</rec:Val>
        </proc:configuration>
      </proc:invokePipelet>
    </extensionActivity>

    <reply name="end" partnerLink="Pipeline" portType="proc:ProcessorPortType" operation="process"
      variable="request" />
  </sequence>
</process>
```

Listing 6.19: *LogosSearch* BPEL pipeline definition

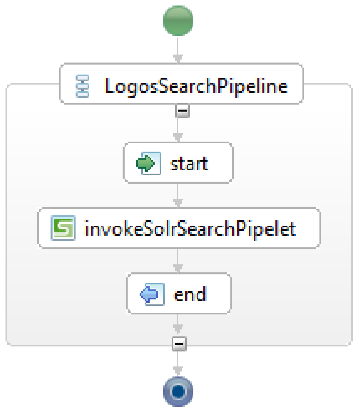


Figure 6.13: The *LogosSearch* BPEL pipeline

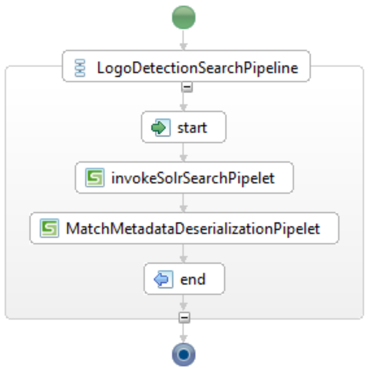
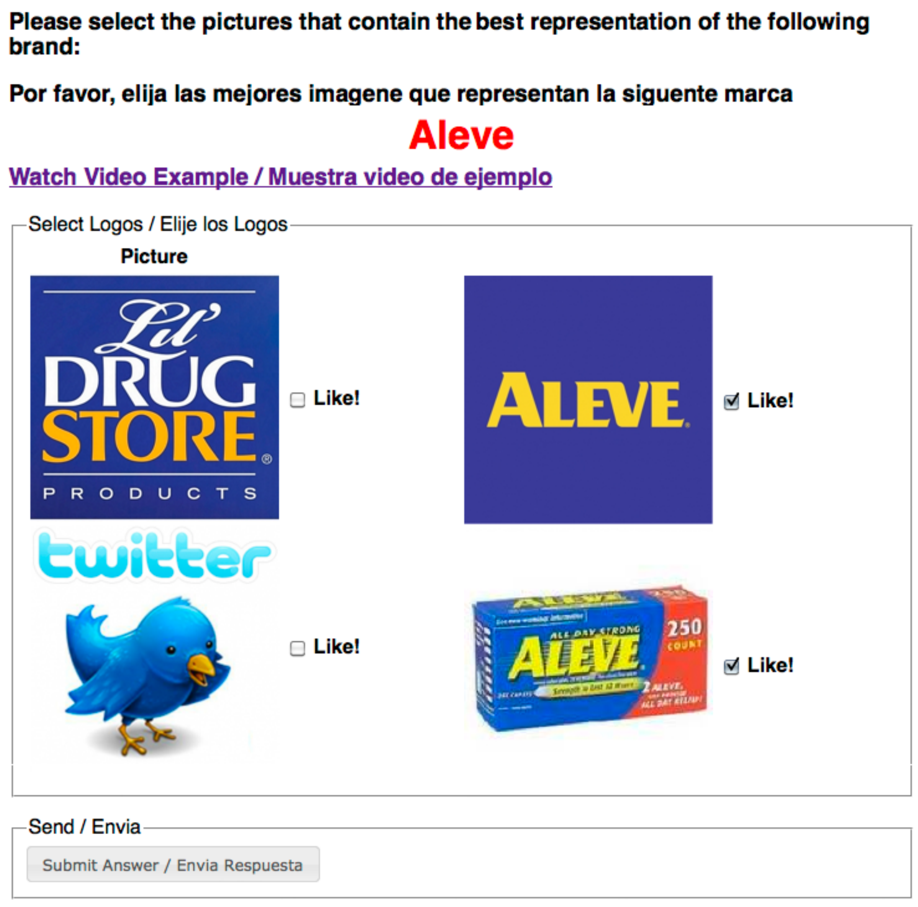

LogoDetectionSearchPipeline
start
invokeSolrSearchPipelet
MatchMetadataDeserializationPipelet
end

Figure 6.14: The *LogoDetectionSearch* BPEL pipeline

```
<process name="LogoDetectionSearchPipeline" ...>
...
  <sequence name="LogoDetectionSearchPipeline">
    <receive name="start" partnerLink="Pipeline" portType="proc:ProcessorPortType"
      operation="process" variable="request" createInstance="yes" />

    <extensionActivity>
      <proc:invokePipelet name="invokeSolrSearchPipelet">
        <proc:pipelet class="org.eclipse.smila.solr.search.SolrSearchPipelet" />
        <proc:variables input="request" output="request" />
        <proc:configuration>
            <rec:Val key="indexname">MatchCore</rec:Val>
        </proc:configuration>
      </proc:invokePipelet>
    </extensionActivity>

    <extensionActivity>
       <proc:invokePipelet name="MatchMetadataDeserializationPipelet">
         <proc:pipelet class="eu.cubrikprj.pipelet.polmi.IndexUpdate.MatchMetadataDeserializationPipelet" />
         <proc:variables input="request" output="request" />
         <proc:configuration/>
       </proc:invokePipelet>
     </extensionActivity>

    <reply name="end" partnerLink="Pipeline" portType="proc:ProcessorPortType" operation="process"
      variable="request" />
  </sequence>
</process>
```
Listing 6.20: *LogoDetectionSearch* BPEL pipeline definition

6.2 Human Enhancement

In this section we describe the implementation of the crowdsourced tasks on the *Conflict Resolution Manager*. The human enhancement implementation is based on the study reported in [Bozzon et al., 2012c].
The tasks to be performed by humans are the logo instances validation and the match validation (see Section 5.3.1).
CrowdSearcher (see Section 2.3) manages the task design, the task distribution and the aggregation of the outputs, while *Facebook* is the social network platform in charge of providing human performers and actually executing the tasks. *CrowdSearcher* acts in the context provided by a given *Facebook* user, who is instrumental to the crowd-sourcing process, being responsible of initiating the tasks which are spawn to the crowd, and by offering friends and colleagues as workers.
The *Facebook* application interacts with *CrowdSearcher* through a platform-specific client. The client has a twofold purpose. On one hand, it allows human performers (i.e., *Facebook* users) to execute the deployed tasks.
On the other hand, the application exploits the native *Facebook Graph API* to



Please select the pictures that contain the best representation of the following brand:

Por favor, elija las mejores imagene que representan la siguente marca

Aleve

Watch Video Example / Muestra video de ejemplo

Select Logos / Elije los Logos

Picture

☐ **Like!**

☑ **Like!**

☐ **Like!**

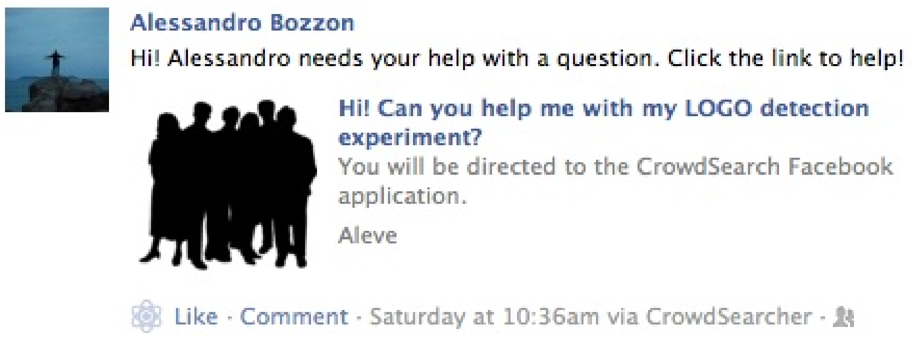
☑ **Like!**

Send / Envia

Submit Answer / Envia Respuesta

Figure 6.15: The user interface for the logo instances validation task

enable a user-defined worker selection, where new workers are explicitly invited by their friends (see Figure 6.16).



Alessandro Bozzon

Hi! Alessandro needs your help with a question. Click the link to help!

Hi! Can you help me with my LOGO detection experiment?

You will be directed to the CrowdSearch Facebook application.

Aleve

Like · Comment · Saturday at 10:36am via CrowdSearcher ·

Figure 6.16: The ask invitation message displayed on the *Facebook* user's wall

Tasks are assumed to have a timeout for completion, that defines how long the system should wait for human execution. When the timeout triggers, the system automatically aggregates the task results.
According to the validation task, (i.e., *logo instance validation* or *match validation*) the system collects the number of preferences for each logo instance, which is next used to establish the *logo confidence* or the number of preferences for each match, to be used to establish whether the match is RELEVANT (i.e., the logo instance is actually present in the video key-frame) or not.

6.3 User Interface

In this section we describe and provide some screenshots of the user interface implemented on top of the logo detection application.
The interface, as well as providing the means to browse the results of the key-frame/logos matching tasks, allow users to explore the intermediate steps in the result formation (see Section 6.1.3.1). The idea behind the *timeline* is to highlight the contribution of the human computation tasks in improving the quality of the result of the application.

6.3.1 User Interaction

User interaction starts when the user sends his query (the name of a brand or a logo instance image) to the logo detection application that queries the *Logo core*.
If the requested brand has already been indexed, the matches for that brand are returned to the user.
Otherwise, if the requested brand has not been indexed yet:

- the SMILA *Logo retrieval* pipeline is started providing the brand name to be query *Google Images*;
- a notification of work in progress is sent to the user;

- as soon as the application updates the *Match core*, the system notifies the Web application that the processing and matching phases are concluded;
- finally the application notifies the user of the search task completion.

6.3.2 User Interface Design

The starting page[21] includes a text-based search form for keyword-based queries.

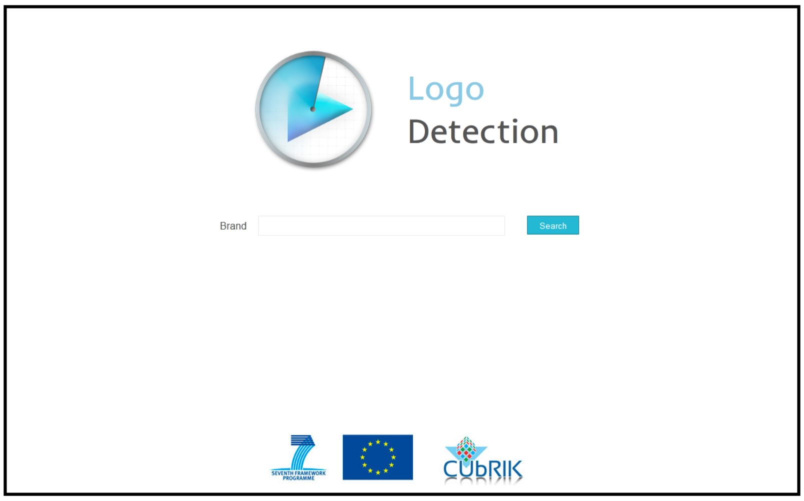


Figure 6.17: The starting page [Preciado Rodríguez et al.]

If the brand name searched by the user is not associated to any of the logo instances in the *Logo core*, the user is asked whether he/she wants to start the processing of the new brand, by clicking on a confirmation button. If the user confirms, then the application replies with a notification in which the user is asked to wait until the process is completed. Figure 6.18 shows the "brand not found" page.

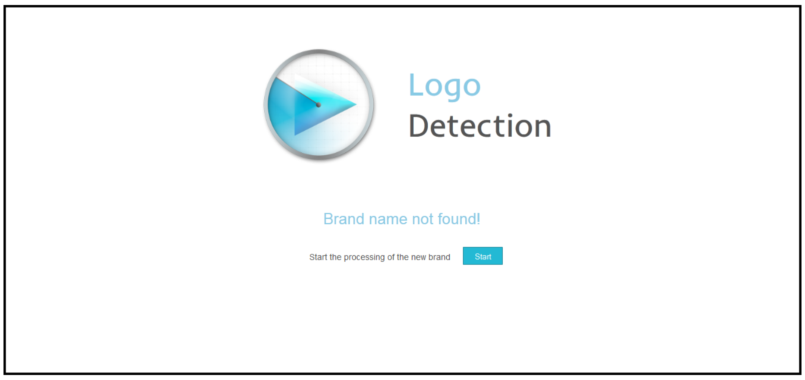


Figure 6.18: The "*brand not found*" page [Preciado Rodríguez et al.]

[21]The screenshots reported in Figures 6.17, 6.18 and 6.19 have been designed and implemented by [Preciado Rodríguez et al.]

Otherwise, if the searched brand has already been indexed (i.e., its logos instances retrieved, processed and matched against the video key-frames), the corresponding matches are shown in the result page.
The result page, depicted in Figure 6.19, is composed of the following sections:

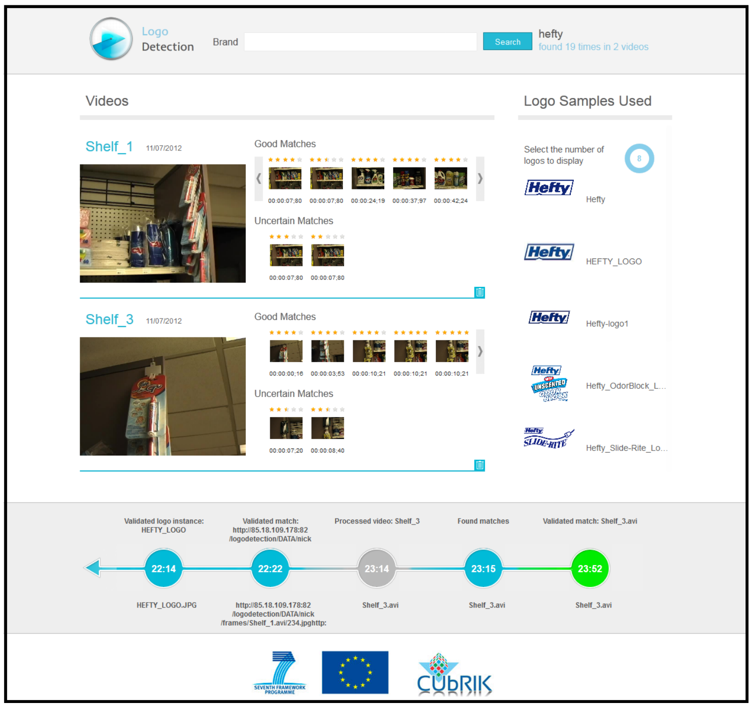


Figure 6.19: The result page [Preciado Rodríguez et al.]

1. ***The search form***: the form, depicted in Figure 6.20, allows the user to send a textual query (i.e., the brand name).

Figure 6.20: The search form [Preciado Rodríguez et al.]

2. ***The statistics***: summarizing some statistics about the search process (e.g., the number of times a brand has been found in the video collection as depicted in Figure 6.21).

Figure 6.21: The statistics [Preciado Rodríguez et al.]

3. ***The logos***: this section contains the top-k logo instances for the query

brand. The logos are ranked according to their *logo confidence*. The value of k can be modified by changing the value of a numeric stepper.

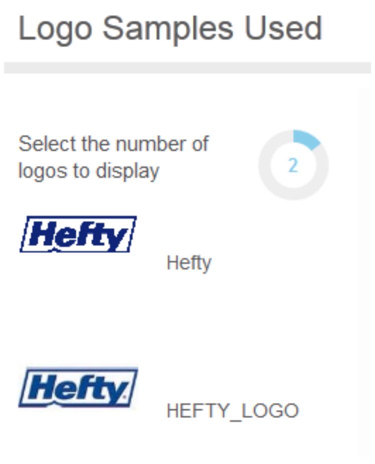


Figure 6.22: The logos section [Preciado Rodríguez et al.]

4. ***The matches***: this section, depicted in Figure 6.23, includes the actual results of the user query. Here videos, in which the top-k logo instances for the query brand have been detected, are listed. For each video, the set of key-frames, in which a match has been found, are shown. Key-frames are divided into three groups, Good (i.e., *high-confidence*), Uncertain (i.e., *low-confidence*) and Rejected ones, according to their score, the high confidence threshold, the *low-confidence* threshold and the crowd judgement. A match is classified as Good if:

   - its score is above the high confidence threshold for the query brand OR
   - the crowd has judged it as Relevant.

   A match is classified as Uncertain if:

   - its score is between the low confidence threshold and the *high-confidence* threshold for the query brand;

   A match is classified as Rejected if:

   - its score is below the *low-confidence* threshold for the query brand OR
   - the crowd has judged it as Not relevant.

   Rejected matches are not displayed in the result page, but a link (e.g., the waste bin icon) is provided in order to show them in a different page. When the mouse pointer passes over a frame image the contained logo instance should be highlighted.
   Similarly, when the pointer passes over a logo instance, the set of key-frames that contains that instance should be highlighted.

When the user clicks on the time instant reported under the key-frame image the video starts playing from that time instant.
When the user clicks on the key-frame image a magnified version of the key-frame is shown in a superimposed frame (depicted in Figure 6.24), showing also the bounding box shape around the logo instance.

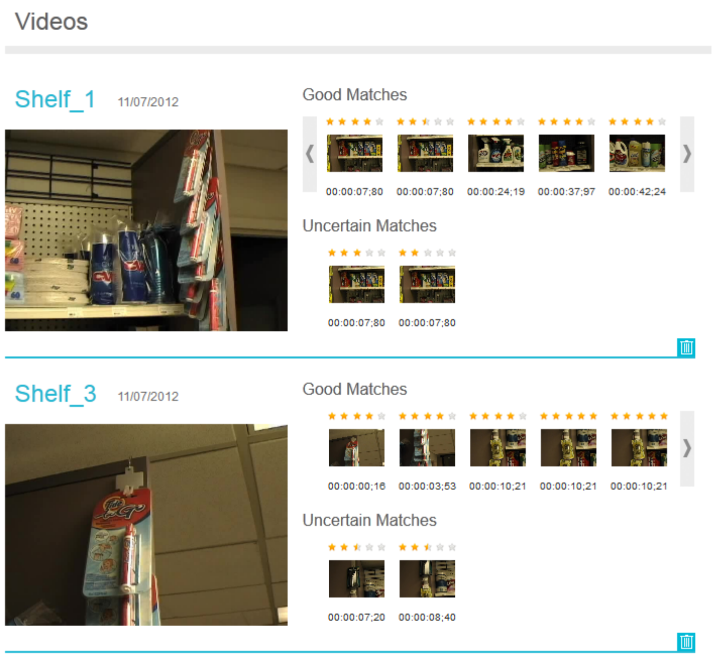


Figure 6.23: The matches section [Preciado Rodríguez et al.]

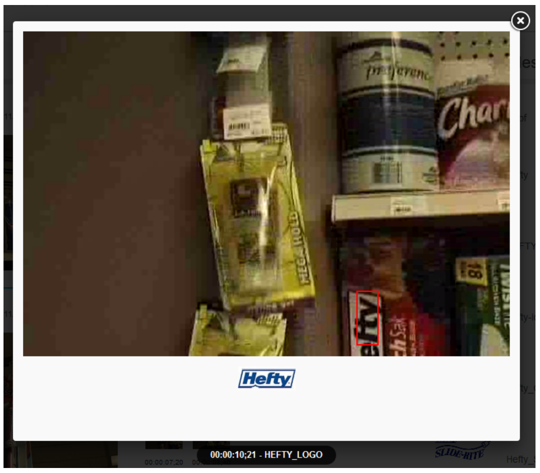


Figure 6.24: An example of match including a bounding box (in red) [Preciado Rodríguez et al.]

5. ***The storyboard***: a set of related events is associated to each result match (e.g., NEW PROCESSED VIDEO, NEW PROCESSED LOGO INSTANCES, FOUND MATCHES, etc.). From the entire result set we can retrieve all the events, that are relevant for the query. The timestamp of each event is known, therefore the events can be ordered using a *timeline*. For each timestamp, matches will be displayed in the status (i.e., *logo confidence* and *match confidence*) they had in that timestamp; if a match has no event for that particular timestamp, the status associated to is the one in which the match was in the maximum of the lesser timestamps. The *timeline* is also enriched with *milestone* timestamps, which correspond to the main system events (see Section 6.1.3.1). Thus, the user can navigate through the results formation story, using the *timeline*. Timestamps in which there are updates in the result set are highlighted with a cue on which the user can click in order to move to that timestamps, on the contrary *milestone* timestamps cannot be navigated by the users. The set of matches shown to the user in the *matches* section is the one corresponding to the latest status, and the corresponding cue on the *timeline* is highlighted. Figure 6.25 draws the storyboard implementation.

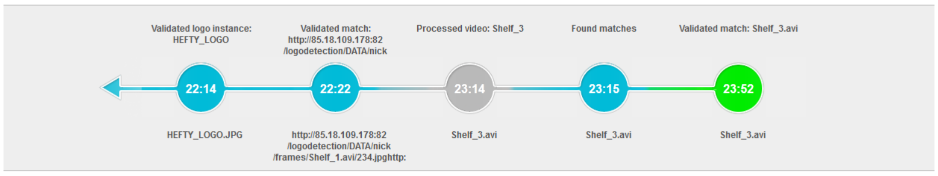

Validated logo instance: HEFTY_LOGO
Validated match: http://85.18.109.178:82 /logodetection/DATA/nick
Processed video: Shelf_3
Found matches
Validated match: Shelf_3.avi
22:14
22:22
23:14
23:15
23:52
HEFTY_LOGO.JPG
http://85.18.109.178:82 /logodetection/DATA/nick /frames/Shelf_1.avi/234.jpghttp:
Shelf_3.avi
Shelf_3.avi
Shelf_3.avi

Figure 6.25: The storyboard section [Preciado Rodríguez et al.]

6.4 Evaluation

In this section we provide an evaluation of the logo detection application, w.r.t. both the *Human Enhancement* process andthe automated ones (i.e., *video processing*, *key-frame matching* and *search*).
In the case of the *Human Enhancement* process we report the experimental results collected in [Bozzon et al., 2012c][22], in which humans were involved in the selection of relevant logo instances w.r.t. a brand name. To evaluate the automated processes, we collected system performance in processing videos (see Section 6.1.3.4), compute matches between video key-frames and logo instances (see Section 6.1.3.5) and retrieving the matches from the *Solr index* (see Section 6.1.4). We observed the time needed to execute the logo application workflows and the time needed to execute single automatic tasks, such as the *key-frame detection* task and the *SIFT descriptors extraction* task, w.r.t. the number of resources (i.e., videos, key-frames, logo instances and matches) to be processed. Finally, in Section 6.4.5 we provide a discussion on the outcomes of each evaluation task.

[22]The experiment had been carried out during the development activities related to this thesis.

6.4.1 Human Enhancement Process Evaluation

The experiments reported in [Bozzon et al., 2012c], involved humans in the validation of the logo instances of three trademark brands (i.e., *Aleve*, *Claritin* and *Chunky*). The logo application demo was tested using the standard measures of *precision* and *recall* on the output of *Key-frame Matching* process.
The *CrowdSearcher* system has been used as *Conflict Resolution Manager* to support the *logo instances validation* crowd task, by allowing users to:

1. select existing logos to improve the *precision* of the application by providing the matching component with correct logo instances
2. add new logos, with the purpose of increasing the overall recall by adding novel logo instance samples.

Around 40 people were involved as workers for the selection or provision of image logos, mostly from the students of Politecnico di Milano, or student's friends who volunteered to be part of the experiment. Some 50 task instances were generated in a time-span of three days, equally distributed on the set of considered logos, resulting in 70 collected answers, 58% of which related to logo images selection tasks.

1. The performance was tested under three experimental settings. No human intervention in the logo instance validation task: here, the top-4 Google Images result set is used as a baseline for the logo search in the video collection; the result set may contain some irrelevant images, since they did not undergo validation.
2. *Logo instances validation* performed by a crowd of domain *experts* (simulation): the top-32 Google Images results are filtered by experts, thereby deleting the non-relevant logos and choosing three images among the relevant ones.
3. Inclusion of the actual crowd knowledge: filtering and expansion of the set of matched logos is done via the *CrowdSearcher* application.

The results are shown in 6.1. For each brand, *precision* and *recall* are evaluated for the three settings of the application.

Brand name	Test	Precision	Recall
Aleve	No Crowd	0.27	0.27
	Experts	0.42	0.54
	Crowd	0.33	0.41
Chunky	No Crowd	0.65	0.19
	Experts	0.70	0.58
	Crowd	0.40	0.21
Claritin	No Crowd	0.31	0.09
	Experts	0.57	0.72
	Crowd	0.36	0.73

Table 6.1: *Precision* and *recall* of the logo detection application in the three considered experimental settings

We can notice that the human contribution, both through the expert and the crowd, brings to an improvement in the system performances w.r.t. the fully automated case. It can be also noticed that the improvement is not evenly distributed over the brands.

6.4.2 Video Processing Evaluation

We considered the *GroZi-120* video collection, composed by 29 .AVI video files. We measured the performances of the system in processing 28 of the 29 videos[23] individually. Then, we plotted the measured values against the number of key-frames detected in each video and through linear regression we get the relationship between the number of key-frames and the time needed to process a video. The mean duration of the videos in the collection is around 57.76 seconds, spanning from a 9 seconds-long video to 1 minute and 50 seconds-long one. The mean number of key-frames detected in a video is 19.
First, we considered the *SIFT descriptor extraction* task, that is the main task performed in the video processing workflow, both for the annotations produced (i.e., the SIFT descriptors files to be used in the subsequent matching workflow) and for the computational efforts. The average measured time needed by the external component to compute the descriptors of a single key-frame image is 3 seconds. For each video in the collection we measured the time needed to extract the SIFT descriptors of all its key-frames; then we plotted the measured time values against the number of key-frames and fitted a line, through the *linear regression* method. The *linear regression* function used to fit the data set is a line without intercept: $y = a * x$.
Figure 6.26 draws the measured values (i.e., the blue triangles) and the fitted line (i.e., the red line).
The parameter a [seconds/key-frame] represents the time needed to process a

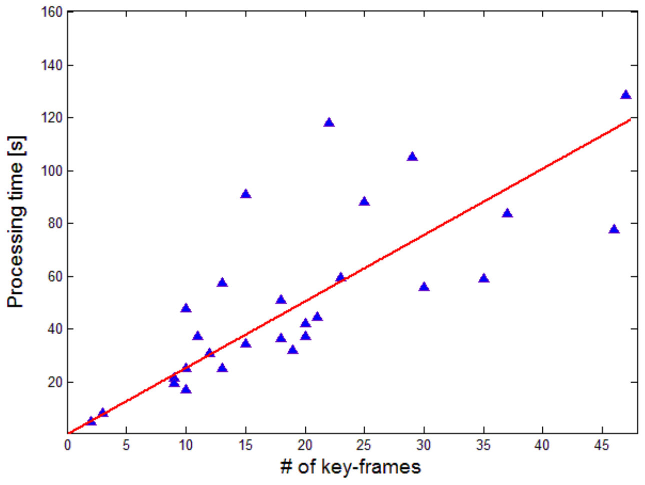


Figure 6.26: The *SIFT descriptors extraction* processing time vs. the number of key-frames

[23]the video named "Shelf_16" had been discarded due to a file corruption

single key-frame and assume the value of 2.426, that is close to the average time needed to process a key-frame image (i.e., 3 seconds).
We can observe that the data set contains some outliers, this is due to the fact that the time needed to extract the SIFT descriptors from the video key-frames, also depends on the image properties of the each key-frame (e.g., image size).
Second, we measured the overall time needed to process a video (i.e., complete the *Video Processing* workflow) and fitted a linear function, (see Figure 6.27).
We can observe that the latter data set is more compact and contains less

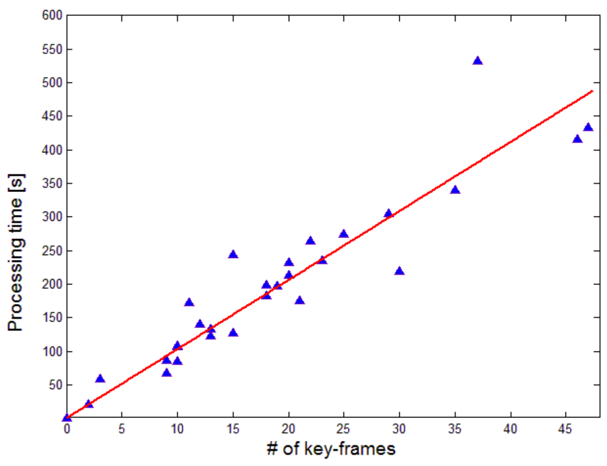


Figure 6.27: The video processing time vs. the number of key-frames

outliers than the one we collected in the case of the SIFT descriptor extraction. Thus, is this case the linear fit well estimates the relationship between the time to process a video and its number of key-frames.

6.4.3 Key-frame Matching Evaluation

We run the *Key-frame Matching* workflow several times, pushing as input always the same logo instance. Before each run, a new video was processed by the application; videos were added one at a time, according to their number of the key-frames. We started from a single video processed (i.e., "Shelf_10" video and its 2 key-frames), to the whole video collection processed (i.e., 28 videos and 542 key-frames).
The mean time needed for the logo detection application to perform the matching operation between a logo instance and a single key-frame is 0.309 seconds.
We plot the measured matching times, and fit a *linear regression* function with no intercept $y = a * x$ (see Figure 6.28). Notice that the x-axis is in logarithmic scale. The value of the parameter a is 0.337 [seconds/key-frame].
Figure 6.29 draws the relationship between matching time and the number of video. The relationships between the matching time and the number of videos is described by a 2-degree curve:

$$y = 0.13 * x^2 + 2.24 * x$$

where x is the number of videos processed.

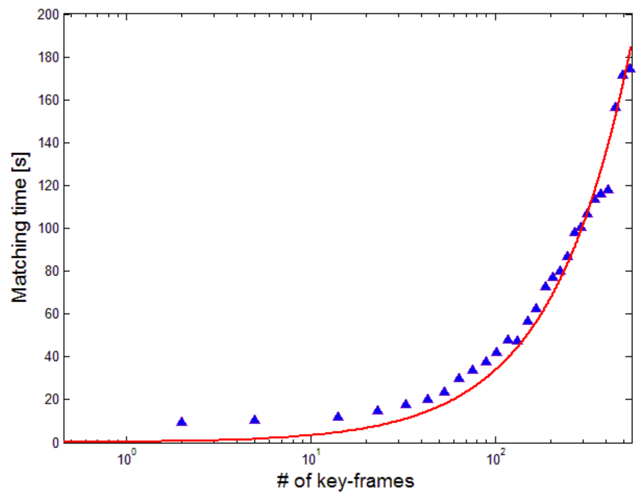


Figure 6.28: The key-frame matching time vs. the number of key-frames

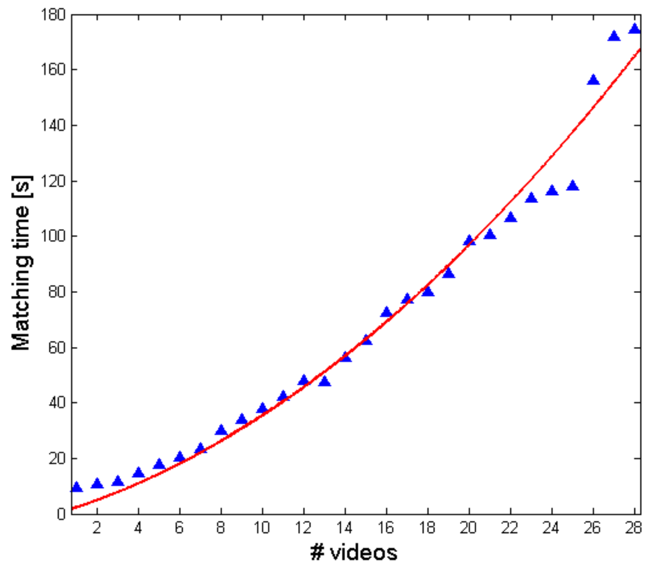


Figure 6.29: The key-frame matching time vs. the number of videos

6.4.4 Search Process Evaluation

We tested the performances of the application in querying and retrieving result from the *Solr index*. In particular, we focused our analysis on the *Match index* (see Section 6.1.3.5).
We started with a system configuration in which 131 logo instances related to 5 brands were processed and indexed, while no videos and no key-frames were processed. Notice that 131 is less than $32 * 5 = 160$ (i.e., the actual number of logo instances retrieved from *Google Images*), because some of the URLs returned by *Google Images* were corrupted or reference to an not-accessible image files. Table 6.2 summarizes the number of logo instances considered for each brand.

Brand name	# logos
Hefty	24
Raid	28
Chunky	20
Halls	29
Campbells	30

Table 6.2: The number of logos processed for each sample brand

We started querying the system with the five sample brands, measuring the response times with cache disabled of the Solr index. Then, we added a video at a time, until we processed the whole *GroZi-120* video collection (i.e., 28 videos), repeating at each step the measurements. Videos were added according to the number of their key-frames, from the one holding the lowest number of key-frames to the largest one. After the addition of a new video, response times were estimated with the following heuristics:

- for each query brand:
  - run some warm-up queries;
  - measure the response time of 10 queries;
  - compute the trimmed mean of the measured response times;
- estimate the overall response time computing the mean w.r.t the query brands.

Figure 6.30 reports the query time measured w.r.t. the number of videos processed, while Figure 6.31 and Figure 6.32 depicts the relationships of the query time, respectively, with the number of key-frames and the number of matches. We can notice that the relationship between the query time and the number of video can be estimated with a quadratic function. In particular the fitting curve we computed has the following form:

$$y = 0.053 * x^2 + 0.45 * x + 6.95$$

where x is the number of videos processed.
Instead, if we consider the number of key-frames processed the relationship turns into a linear one:

$$y = 0.103 * x + 7.69$$

where x is the number of key-frames processed.
Finally, if we consider the number of matches indexed in the *Solr core* the relationship is estimated by the following linear equation:

$$y = 0.011 * x + 6.64$$

where x is the number of matches.

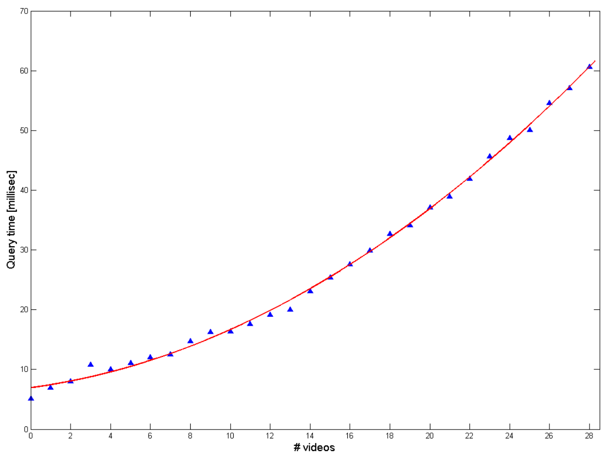


Figure 6.30: The query time vs. the number of videos

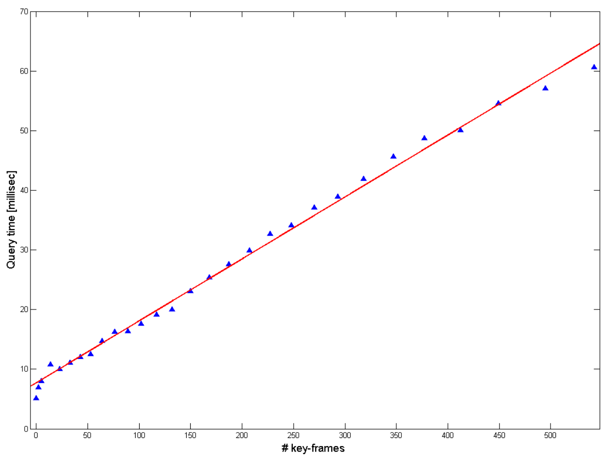


Figure 6.31: The query time vs. the number of key-frames

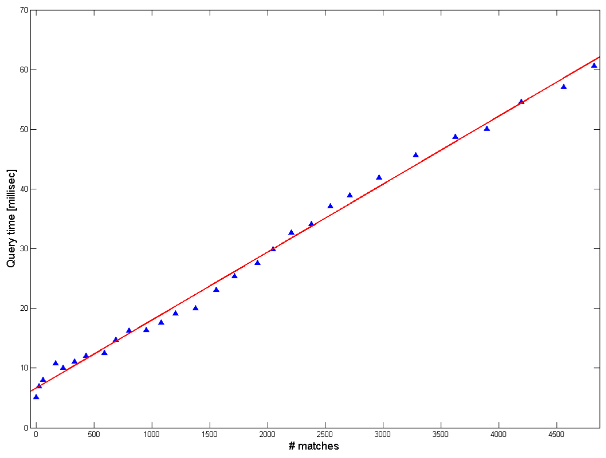


Figure 6.32: The query time vs. the number of matches

6.4.5 Discussion

Human Enhancement process Results reported in Section 6.4.1 show that the *expert validation* increases the application performance in both *precision* and *recall*, with respect to a fully-automated solution. This is due to the fact that the validation process performed on the set of logo instances eliminates irrelevant logos from the query set, and consequently reduces the number of false positives in the result set.
On the other hand, the validation conducted by the crowd showed generally a slight increase in both *precision* and *recall*. It can be noticed that the performance increase is not evenly distributed over all the brands; this could be due to a different user behavior in the choice of the relevant image set. For instance, when validating the *Chunky* brand logo instances, the crowd chose within the top-2 image an irrelevant logo. Consequently, the performance has been affected, with a considerable decrease both in terms of *precision* and *recall* w.r.t. the *expert* evaluation. This observation highlight that importance of the human context of the execution of the crowd tasks: the chances to get good responses depend on the appropriateness of the users' community. Both the geographical location and the expertise of the involved users can heavily influence the outcome of crowdsourcing activities, thus the *People to Task Matching* phase is a crucial phase in the *Human Enhancement* process.

Video Processing workflow Experiments shown that the number of keyframes is not the main factor that influences the time needed to compute the SIFT descriptors for a whole video. The processing time to compute the SIFT descriptors for a single image highly depends on the complexity of the image itself, in terms of structures and details it contains. The more the details an image hold, the more key-points are likely to be detected and the more the number of key-points detected, the more the key-point descriptors to be computed.
Conversely, if we consider the time needed to completely process a video, we can

notice that the number of key-frames affect considerably the whole processing time. Indeed, the time needed to perform the video conversion, the key-frame detection, the key-frame extraction the video upload and the key-frame upload activities clearly depend just on the length of the video and on the number of the key-frames, and the time needed to fulfil these activities represents the 80% of the whole video processing time.

Key-frame Matching workflow From a theoretical point of view the computational complexity of the *Key-frame Matching* workflow is $O(NM)$, where N is the number of logos and M the number of key-frames to match.
The measurement we collected, reported in Section 6.4.3, confirm that the relationship between the time needed to perform the matching between a single logo instance and the whole key-frame set is linear, $O(M)$.
W.r.t. the previous case, the business logic of the *Key-frame Matching* workflow does not vary significantly if we run it pushing a single key-frame. Thus, it can be reasonably assumed that also the relationship between the time needed to perform the matching between a single key-frame and the whole logo instances set is linear, $O(N)$.
Hence, our measurement and observations confirmed that the computational complexity of the workflow is depends from the number of both the logo instances and the key-frames to be processed.

Search process Our measurements show that there is quadratic relationship between the query time and the number of videos. Obviously the form of fitting function we estimated is influenced by the order of addition of a new video in the collection to be processed. But in general, we can observe that it is a always-increasing function and that the change in slope is dependent on the number of key-frames that the new added video holds.
Thus, if we change our point of view and focus on the number of key-frames, we can observe that the query time increase in a linear fashion w.r.t. the number of key-frames. Indeed, the more the new key-frame the system process the more likely the system will find new matches to index.
The number of matches put in the index is the factor that directly affect the query time: the larger the index, the larger the time needed to retrieve contents from it. According to our measurements, the relationship between query time and number of matches is linear.

7 Conclusions and future works

This work introduced a new framework for the design of multimedia search-based applications (SBAs) that comprise human computation and crowdsourced tasks.
The framework considers human as first-class citizens in the search and analysis processes, where performers can, for instance, be exploited in the validation of the outcomes produced by the automated components in SBAs architecture, in order to improve the quality performances of the retrieval system.
We discussed the challenges and the requirements that a human-enhanced multimedia search-based application (hSBA) should address, focusing especially on the requirements driven by the presence of humans in the analysis and search loops.
We proposed a conceptual framework consisting of a development methodology, and data and process models, to support the design and the implementation of hSBAs. In particular, we introduced the *Conflict Resolution model* and *Human Enhancement process*. The former is a data model that allow us to define and characterize the concepts and the actors that are involved in the management and in the resolution of the uncertainty that is exhibited by automated components. Indeed, the *Human Enhancement process* describes all the activities needed to assign the validation tasks to the human workers, such as the design of a suitable user interface and the assignment to the best suited workers, that are peculiar activities w.r.t. to classical *Indexing* and *Search* processes' ones.
We presented the logo trademark detection application, a use case application developed in the context of an European research project which has been designed and realized by using the proposed framework. After having presented and detailed the use case scenario, we proposed an implementation of the logo detection application in the SMILA framework.
We also provided an evaluation of SBA performances in terms of processing time, query time w.r.t. the videos processed, the key-frames processed and the matches found. Thank to preliminary experiments on the *Human Enhancement*

process, we can state that human validation can contribute to a non-negligible improvement in the retrieval performance of the system.

7.1 Future works

This thesis and recent studies on crowdsearching reveal the potential of embedding Human Computation and crowdsourcing in search-based application, but the actual impact on SBAs performances needs to be further studied. For instance, we may be interested in observing how the design of the crowdsourced task affects the quality of the outcomes produced (e.g., number of logos to be validated per single task), or what is minimum number of workers, to which assign the same task, in order to guarantee the consistency of the task outcome. The selection of the most appropriate set of workers is another challenge: workers' profiles, accessible through the crowdsourcing platforms, could be exploited in order to assign a task to the most suited workers (e.g. country, culture, etc.). We consider to continue the evaluation of the logo trademark detection application, in terms of precision/recall, including the results variations after the validation of the matches by the crowd.
To confirm our findings, we also plan to test our application on different video collections.
An interesting improvement of our use case could be the enrichment of the logo detection application with new crowdsourced tasks, such as the addition of new logo instances among the validation of the ones returned by *Google Images.*
Another improvement is the adoption of several crowdsourcing platforms (e.g., *Twitter*, *Google Plus*, etc.) for human computation tasks. Moreover, new use cases could be thought to enforce our conceptual framework and new human tasks could be designed to exploit human capabilities and knowledges.

References

Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 020139829X.

Ricardo A. Baeza-Yates, Stefano Ceri, Piero Fraternali, and Fausto Giunchiglia, editors. *Proceedings of the First International Workshop on Crowdsourcing Web Search, Lyon, France, April 17, 2012*, volume 842 of *CEUR Workshop Proceedings*, 2012. CEUR-WS.org.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

Boualem Benatallah, Marlon Dumas, and Quan Z. Sheng. Facilitating the rapid development and scalable orchestration of composite web services. *Distrib. Parallel Databases*, 17(1):5–37, January 2005. ISSN 0926-8782. doi: 10.1023/B:DAPD.0000045366.15607.67. URL `http://dx.doi.org/10.1023/B:DAPD.0000045366.15607.67`.

Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, September 2001. ISSN 0360-0300. doi: 10.1145/502807.502809. URL `http://doi.acm.org/10.1145/502807.502809`.

Alessandro Bozzon. *Model-driven development of Search Based Web Applications.* PhD thesis, Politecnico di Milano, April 2009.

Alessandro Bozzon, Marco Brambilla, and Stefano Ceri. Answering search queries with crowdsearcher. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 1009–1018, New York, NY, USA, 2012a. ACM. ISBN 978-1-4503-1229-5. doi: 10.1145/2187836.2187971. URL `http://doi.acm.org/10.1145/2187836.2187971`.

Alessandro Bozzon, Marco Brambilla, and Andrea Mauri. A model-driven approach for crowdsourcing search. In Baeza-Yates et al. [2012], pages 31–35.

Alessandro Bozzon, Ilio Catallo, Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. A framework for crowdsourced multimedia processing and querying. In Baeza-Yates et al. [2012], pages 42–47.

Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 61–72, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4. doi: 10.1145/1989323.1989331. URL `http://doi.acm.org/10.1145/1989323.1989331`.

Hervé Goëau, Alexis Joly, Souheil Selmi, Pierre Bonnet, Elise Mouysset, and Laurent Joyeux. Visual-based plant species identification from crowdsourced data. In *MM'11 - ACM Multimedia 2011*, Scottsdale, États-Unis, November 2011. ACM. doi: 10.1145/2072298.2072472. URL `http://hal.inria.fr/hal-00642236`.

Christopher G. Harris. An evaluation of search strategies for user-generated video content. In Baeza-Yates et al. [2012], pages 48–53.

Chien-Ju Ho, Tao-Hsuan Chang, Jong-Chuan Lee, Jane Yung-jen Hsu, and Kuan-Ta Chen. Kisskissban: a competitive human computation game for image annotation. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '09, pages 11–14, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-672-4. doi: 10.1145/1600150.1600153. URL `http://doi.acm.org/10.1145/1600150.1600153`.

Jeff Howe. The rise of crowdsourcing. *Wired*, 14(6), 2006. URL `http://www.wired.com/wired/archive/14.06/crowds.html`.

E. Law, L. von Ahn, R. Dannenberg, and M. Crawford. Tagatune: a game for music and sound annotation. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)*, 2007.

Edith Law and Luis von Ahn. Input-agreement: a new mechanism for collecting data using human computation games. In *CHI*, pages 1197–1206, 2009.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

T.W. Malone, R. Laubacher, and C. Dellarocas. Harnessing crowds: Mapping the genome of collective intelligence. Research Paper No. 4732-09, MIT, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, USA, February 2009. URL `http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1381502#`. Sloan Research Paper No. 4732-09.

B. S. Manjunath and W.Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI - Special issue on Digital Libraries)*, 18(8):837–42, Aug 1996. URL `http://vision.ece.ucsb.edu/publications/96PAMITrans.pdf`.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, UK, 2008. ISBN 978-0-521-86571-5.

Christian Middleton and Ricardo Baeza-Yates. A comparison of open source search engines. Technical report, 2007.

Manoj Parameswaran and Andrew B Whinston. Research issues in social computing. *Journal of AIS*, 8:336–350, 2007.

Miguel A. Preciado Rodríguez, Javier Garcia Morón, and Fernando Sánchez Herrera. Logo detection application user interface. Homeria Open Solutions. URL http://www.homeria.com.

Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 1403–1412, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0228-9. doi: 10.1145/1978942.1979148. URL http://doi.acm.org/10.1145/1978942.1979148.

Cees G. M. Snoek, Bauke Freiburg, Johan Oomen, and Roeland Ordelman. Crowdsourcing rock n' roll multimedia retrieval. In *Proceedings of the ACM International Conference on Multimedia*, Firenze, Italy, October 2010.

Thomas Steiner, Ruben Verborgh, Rik Van de Walle, Michael Hausenblas, and Joaquim Gabarro. Crowdsourcing event detection in youtube videos. In *Detection, Representation, and Exploitation of Events in the Semantic Web (DeRiVE 2011)*, Bonn, Germany, 2011. URL http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-779/derive2011_submission_8.pdf.

Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.

Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Texture features corresponding to visual perception. *IEEE Transactions on System, Man and Cybernatic*, 6, 1978.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Addison-Wesley, 2005. ISBN 0-321-32136-7.

Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proceedings of the 2004 Conference on Human Factors in Computing Systems, CHI 2004, Vienna, Austria, April 24 - 29, 2004*, pages 319–326, 2004. doi: http://doi.acm.org/10.1145/985692.985733.

Luis von Ahn and Laura Dabbish. Designing games with a purpose. *Commun. ACM*, 51(8):58–67, 2008.

Luis von Ahn, Ruoran Liu, and Manuel Blum. Peekaboom: a game for locating objects in images. In *CHI*, pages 55–64, 2006.

Rong Yan and Winston Hsu. Concept-based image retrieval. ACM Multimedia, 2008.

Tingxin Yan, Vikas Kumar, and Deepak Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 77–90, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5. doi: 10.1145/1814433.1814443. URL `http://doi.acm.org/10.1145/1814433.1814443`.

Jun Yang, Yu-Gang Jiang, Alexander G. Hauptmann, and Chong-Wah Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of ACM SIGMM Workshop on Multimedia Information Retrieval*, 2007.

Man-Ching Yuen, Ling-Jyh Chen, and Irwin King. A survey of human computation systems. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 04*, CSE '09, pages 723–728, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3823-5. doi: 10.1109/CSE.2009.395. URL `http://dx.doi.org/10.1109/CSE.2009.395`.