Alma Mater Studiorum
Università degli Studi di Bologna

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea in Sistemi Informativi LA

Progettazione e realizzazione di un benchmark di test per il sistema Shiatsu

Candidato:
Fabrizio Marconi

Relatore:
Prof. Ilaria Bartolini

Correlatore:
Prof. Marco Patella

Anno Accademico 2012/2013 - Sessione I

Resistere è inutile!

Indice

Capitolo 1

Introduzione

Il continuo e sempre più veloce evolversi di internet ha portato cambiamenti radicali nella vita di tutti. Le informazioni vengono ricercate ed acquisite in modi diversi rispetto a quando non esisteva, e perfino rispetto alla sua nascita le cose sono fortemente cambiate. La ricerca all'interno del World Wide Web si pratica quasi sempre attraverso l'uso di keywords. Queste sono spesso efficaci per la ricerca all'interno di testi ma diventano carenti per quanto riguarda le fonti multimediali. Infatti, per far fronte al prepotente espandersi di immagini, audio e video, i motori di ricerca si sono aggiornati ed evoluti. Oggi è possibile trovare facilmente un file musicale attraverso un suo spezzone molto breve, o un'immagine mediante una simile; ma per quanto riguarda i video ci si affida ancora molto spesso al metodo del "tagging", cioè: attribuire dei valori testuali per indicare i contenuti dei video. Benché questo metodo non sia sempre inefficace a volte si rivela inaffidabile. Ciò perché i "tag" devono essere inseriti manualmente per ciascun file multimediale, o magari per gruppi di files. E' evidente che questo lavoro manuale diventa quanto più impreciso o costoso tanto più è grande la quantità di dati da gestire. Inoltre questo sistema può essere impreciso, infatti due video con lo stesso tag possono avere contenuti visivi o uditivi completamente diversi. Proprio per questo il progetto SHIATSU si può considerare un sistema di ricerca all'avanguardia. SHIATSU *(Semantic-Based Hierarchical Automatic Tagging of Videos by Segmentation using Cuts)* è un sistema di tagging video

automatizzato che si propone di trovare e attribuire tag coerenti ai video, e di cercare similarità tra più video. Il principio di base è che il video è un oggetto complesso: divisibile nelle due componenti audio e video; a sua volta la componente video è suddivisibile in scene ed infine immagini. Considerando il video come una sequenza ordinata di immagini, SHIATSU analizza queste ultime, cercando possibili tag ed estendendoli in modo gerarchico alle varie scene ed infine al video stesso. La versione attuale di SHIATSU è uno strumento di ricerca efficiente di video. Tale ricerca è stata recentemente indicizzata ed ottimizzata, aumentando di molto la velocità del sistema. Ad oggi ci si trova di fronte alla necessita di testare quanto questo sistema sia realmente efficiente; tuttavia per fare ciò il database che era nativamente inserito in SHIATSU, atto soprattutto allo sviluppo e ai test di ricerca, risulta inadeguato. Lo scopo di questa tesi è quindi quello di realizzare uno strumento che consenta di aggiungere materiale all'interno del database in modo autonomo e semplificato. In questo modo sarà possibile creare una nuova base di dati, adatta a testare tutte le potenzialità di SHIATSU. Il lavoro di tesi è suddiviso in 6 capitoli strutturati come segue:

1. **Introduzione** in cui si descrive lo scenario e si introducono gli argomenti trattati.
2. **Struttura Shiatsu** in cui si descrive la struttura esistente e se ne illustrano le componenti.
3. **Analisi del problema** in cui si descrive il problema e se ne illustrano gli aspetti.
4. **Realizzazione del CutTask** in cui si descrive la creazione del modulo per l'inserimento di nuovi video.
5. **Progetto del Database** in cui si descrivono le scelte fatte nella realizzazione della base di dati.
6. **Conclusioni** in cui si analizzano i risultati del lavoro e si esaminiamo i punti aperti.

Capitolo 2

Struttura Shiatsu

2.1 Introduzione

SHIATSU è un sistema capace di effettuare interrogazioni efficienti su collezioni di dati multimediali.[1] I suoi compiti comprendono l'annotazione semi-automatica dei video e la ricerca di media a partire sia da tag testuali che da documenti multimediali. Lavora secondo logica gerarchica: un video viene diviso in shots, cioè singole scene, queste poi vengono studiate come collezioni di immagini detti Keyframes; è programmato prevalentemente in Java, e sfrutta ampiamente la libreria Java Media Framework[2]: un modulo che permette la manipolazione di video mediante codice Java. L'architettura di Shiatsu è formata da tre strati software: Interfaccia, Engine Layer e Data Layer. L'interfaccia grafica è efficace e consente sia di fare interrogazioni che di operare sui tag, include una funzione per la riproduzione di video al suo interno. L'Engine Layer comprende i tre componenti chiave: l'estrattore di features visuali che determina le caratteristiche delle immagini, l'annotatore che realizza l'algoritmo di annotazione automatica e il risolutore di interrogazioni, che si occupa di trovare i risultati per le query dell'utente. Il Data Layer è lo strato più basso che si occupa di dialogare con il Database e di mantenere i dati coerenti.
Alcuni moduli sono di particolare rilievo per il lavoro che ci si propone di fa-

re, nello specifico verranno utilizzati soprattutto lo Shot Detection Module, il Data Layer ed infine il Feature Extraction Module che di fatto è realizzato principalmente da Windsurf.

2.2 Windurf

La base del sistema SHIATSU è costituita dalla libreria Windsurf *(Wavelet-Based Indexing of Images Using Region Fragmentation)* che consente la gestione di dati multimediali con struttura gerarchica e dà accesso a modalità flessibili di ricerca e di estrazione dati. Windsurf si colloca prevalentemente nell'Engine Layer nonostante sia dotato di componenti proprie per l'interfaccia al database. La definizione del modello di ricerca di Windsurf viene riportata qui di seguito:

> "Dato un database DB di N documenti, DB= $\{D1, ...,$Dn$\}$, dove ogni documento D è composto da nd elementi, D=$\{R1, ...,$Rnd$\}$. Ogni elemento R di D è descritto dalle proprie caratteristiche che ne descrivano il contenuto. Fornito un documento query D q = $\{Q1, ...,$Qn$\}$ composto da n elementi, una funzione di distanza tra elementi ??Delta che misura, usando le caratteristiche visuali, la dissimilarità tra una coppia di elementi, vogliamo ottenere il set di documenti in DB migliori rispetto a Dq secondo un certo tipo di confronto." [3]

Nel presente caso i documenti sono le immagini: i frame di cui si compongono i video. Questi documenti verranno divisi in modo appropiato e studiati per facilitare le operazioni di ricerca.

2.2.1 Region-Based e Histogram-Based Image Retrival

Nel modello Region-Based ogni immagine viene scomposta in "regioni", cioè aree dell'immagine con caratteristiche visuali simili. Ogni regione è

dotata di caratteristiche proprie: textures, colori, forme. Queste vengono elaborate in forme testuali in modo da poter essere più agevolmente salvate come “features”.

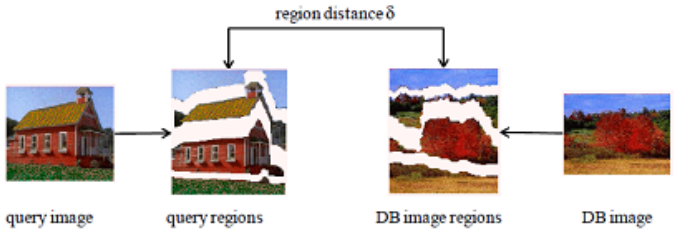


Figura 2.1: Un esempio della divisione in regioni

L’individuazione delle caratteristiche visuali è di fondamentale importanza in quanto queste diventeranno la base per il calcolo della distanza tra più immagini. Prima di effettuare tale calcolo, sarà necessario stabilire la distanza tra le regioni delle immagini; successivamente una funzione andrà ad aggregare i risultati in modo da ottenere la distanza complessiva. Ci sono diverse politiche di similarità tra regioni (Windsurf Distance, Earth’s Mover Distance, Integrated Region Matching, Skyline), che possono essere utilizzate a seconda del caso e dello scopo della ricerca.

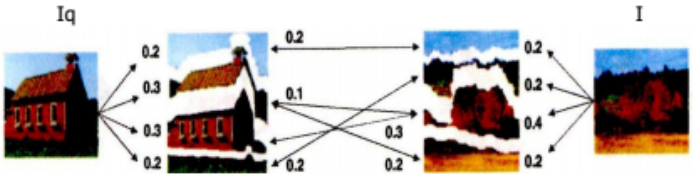


Figura 2.2: Un esempio del confronto fra regioni

Mentre la Region-Based dà importanza anche alla struttura dell’immagine, la tecnica Histogram-Based si occupa principalmente della dominanza dei colori. Essa è basata su istogrammi: l’istogramma di un’immagine digitale è una rappresentazione della distribuzione dei toni in essa contenuti. Nella pratica si definiscono gruppi di colori e vengono analizzati i pixel per definire a quale gruppo essi appartengano; il tutto poi viene rappresentato in forma

grafica. La quantità di colori definiti influenza la precisione della rappresentazione. L'istogramma può essere costruito in molti modi ma gli standard più diffusi sono gli spazi tridimensionali RGB o HSV. Calcolare la distanza tra immagini rappresentate mediante istogrammi è più semplice e quindi più economico in termini di tempo ed elaborazione rispetto alla modalità a regioni, questo a discapito della qualità della ricerca.

2.2.2 Struttura della libreria

La libreria Windsurf è scritta in linguaggio Java, come gran parte del sistema SHIATSU; i suoi package principali sono:

- Document: contiene le classi necessarie alla descrizione delle immagini e delle regioni che le compongono. Include, inoltre, strumenti per il calcolo della distanza tra immagini.
- FeatureExtractor: si occupa di identificare ed estrarre regioni e features dalle immagini.
- QueryProcessor: risolve le query basate su immagini, utilizzando appositi indici, se presenti, per velocizzare la ricerca.
- FeatureManager: gestisce le interrogazioni sul DB MySQL per la ricerca di immagini.
- IndexManager: si occupa di gestire gli indici utilizzati dal QueryProcessor.

2.3 Detection Module

Il Detection Module ha la funzione di individuare gli istanti in cui un video deve essere tagliato, al fine di formare dallo stesso una collezione di scene: gli shots. Il problema della segmentazione è piuttosto complesso: ci sono molti algoritmi con vario grado di profondità per individuare i cambi

di scena; la ricerca, inoltre, è ancora in corso. Inizialmente si era preferito dotare SHIATSU di un sistema di Shot Detection rivolto all'efficienza. Il "ThresholdDetector" che nel sistema era il principale attore del Detection Module lavorava confrontando due frames successivi, utilizzando sia istogrammi HSV sia calcolando gli oggetti che entravano od uscivano dalla scena. Il detector cercava di individuare due fenomeni: *hard cuts* e *gradual transition*. Gli *hard cuts* sono i cambi di scena netti, da un immagine al nero, o da un'immagine a una completamente diversa. Invece la *gradual transition* è la dissolvenza, cioè l'effetto di un'immagine che sfuma più o meno gradualmente in un'altra. Ovviamente questo sistema non poteva garantire l'identificazione perfetta di tutti i cambi di scena, ma inserendo una doppia soglia dinamica è stato possibile adattare la sensibilità per varie tipologie di video. Più recentemente si è deciso di rendere il Detection Module ancor più flessibile, infatti nel lavoro di tesi di Alessandro Colace [4] si è voluto ingegnerizzare questo modulo, consentendo di inserire vari "Detector": moduli di individuazione di cambio scena (vedi figura). In questo modo è possibile scegliere il "Detector" più adeguato a seconda del video che si vuole analizzare. Uno di questi rappresenta il lavoro di tesi di Giovanni Concas[5], che ha scelto di cercare i cambi di scena sfruttando unicamente il canale audio. Esso andrà ad aggiungersi al detector predefinito, il Threshold ed eventualmente ad altri Detector che si voglia realizzare in futuro. L'aspetto interessante di questa struttura è la possibilità di confrontare i tagli individuati dai vari Detector e, analizzando i dati, studiare l'efficienza dei vari algoritmi.

2.4 Java Media Framework

Java Media Framework è una libreria che consente la manipolazione di file video all'interno dell'ambiente Java. Di seguito si riporta la descrizione fornita dal sito ufficiale.

> The Java Media Framework API (JMF) enables audio, video and other time-based media to be added to applications and applets built on Java technology. This optional package, which can

> capture, playback, stream, and transcode multiple media formats, extends the Java 2 Platform, Standard Edition (J2SE) for multimedia developers by providing a powerful toolkit to develop scalable, cross-platform technology.[6]

La libreria offre diversi servizi che consentono di elaborare flussi audio e video: dal player, per riprodurre i video in un interfaccia grafica, a strumenti più complessi che consentono di analizzare o modificare il video stesso.

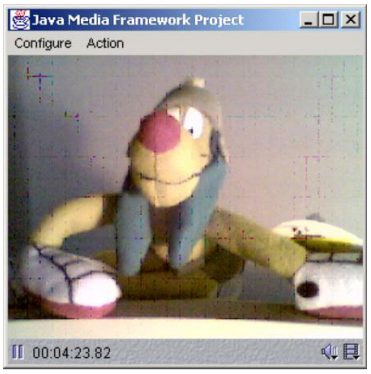


Figura 2.3: La finestra del player di JMF

Viene utilizzata in molti modi all'interno di Shiatsu: un player è inserito nell'interfaccia per visualizzare i video inseriti; inoltre viene utilizzata per estrarre i frame ed i keyframe che devono essere elaborati. La struttura non è di semplice utilizzo anche perché il supporto, insieme all'ultimo aggiornamento, è terminato nel 2004; il tutto è documentato nel sito ufficiale Oracle.

2.5 Data Layer

Il Data layer è lo strato di software che si occupa di registrare e mantenere coerenti i dati generati ed utilizzati da Shiatsu. Esso è composto dal database vero e proprio, su base MySQL; dallo strato software che si occupa delle chiamate a database: estrarre o inserire dati dal database a seconda

delle esigenze; infine, dalla cartella “VIDEO_DB”. Lo strato software è stato realizzato mediante chiamate JDBC[7]. Il tutto è stato modellato a partire da Windsurf ed adattato al fine di lavorare in modo gerarchico per poter estendere i tag delle singole immagini ai video.

2.5.1 Database

La struttura iniziale del database è riportata nel seguente schema.

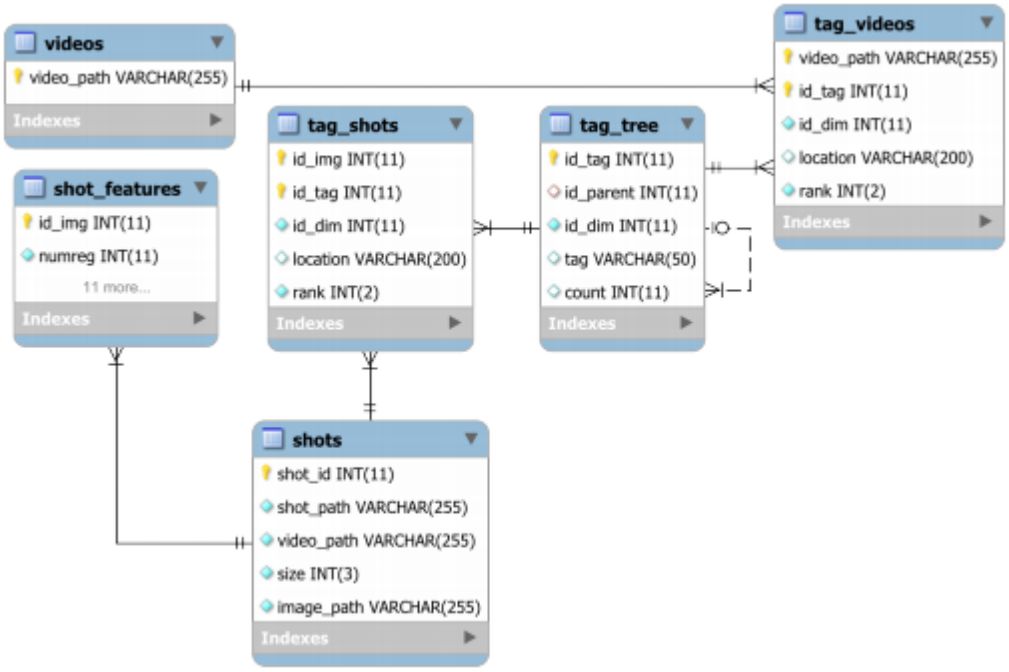


Figura 2.4: Schema ER della struttura originale del database Shiatsu.

Come si può vedere sono presenti 6 tabelle:

videos: è la tabella che contiene l’elenco dei video presenti in Shiatsu, la cui chiave primaria è il path del video, relativo alla cartella di archivio di Shiatsu, DB-VIDEO per default.

shots: contiene l'elenco degli shots presenti in Shiatsu, la chiave primaria è un id numerico, sono inoltre presenti il path dello shot e il path del video da cui questo è stato estratto; è presente anche un'immagine che dovrebbe rappresentare lo shot e il numero dei keyframes estratti.

tag_tree: contiene i tipi di tag che è possibile associare ai keyframe. Ogni tag ha un *id_tag* che lo identifica e un *id_parent* che identifica il genitore recursivamente, così è possibile creare gruppi di tag. Il campo tag invece, è una descrizione del concetto che il tag vuole identificare, ad esempio "sport" o "pepole".

tag_shots: associa agli shots i tag mediante la ForeignKey *id_tag* che si riferisce alla tabella *tag_tree*, la variabile location non è che una ripetizione di *tag* nella tabella *tag_tree*.

tag_video: associa ai video i tag, è strutturata come *tag_shots*.

shot_features: contiene le features dei vari Keyfreams estratte da Windsurf e memorizzate dallo stesso secondo le proprie regole. *id_img* in questo contesto indica una singola immagine, ma essendo ForeignKey riferita alla tabella *shots*, è possibile memorizzare un solo keyframe per ogni shot.

Queste tabelle sono indicizzate in M-tree in modo da migliorare le prestazioni nella ricerca.

2.5.2 CartellaVIDEO_DB

La cartella "VIDEO_DB" contiene i file che sono stati elaborati e segmentati da Shiatsu, sono organizzati secondo un'apposita struttura di directory. E' necessario prestare attenzione in quanto la struttura deve rispecchiare quello che è registrato in database o si rischia di incorrere in errori durante l'esecuzione di Shiatsu.

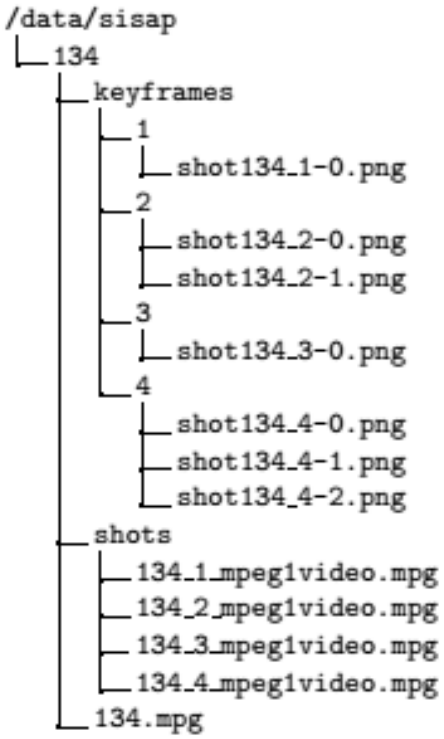


Figura 2.5: La struttura della cartella VIDEO_DB [8]

All'interno della cartella "sisap" si trova una cartella per ogni video presente in shiatsu. All'interno di queste vi è presente il video, una cartella che contiene i vari shots, e una che mantiene organizzati i keyframes. I keyframes sono a loro volta distribuiti in cartelle, una per ogni shot del video.

Capitolo 3

Analisi del problema

A seguito della recente ottimizzazione della ricerca in Shiatsu, il database di cui il sistema era fornito è risultato inadeguato. La collezione di video era stata costruita ed inserita in un primo momento in modo manuale, e per questo motivo era ovviamente di dimensione ridotta. Scopo del database iniziale era soprattutto di testare le potenzialità del sistema e non di misurarne le prestazioni, per tale motivo risultava maggiormente indicato per il nascente sistema Shiatsu perché più semplice da compilare. Ad oggi si conoscono più a fondo le potenzialità di Shiatsu, e si vuole misurarne anche le prestazioni in modo più esteso. Da qui l'obbiettivo del presente lavoro di tesi: compilare un database di grandi dimensioni, adatto a divenire la principale piattaforma per il benchmark di Shiatsu.

3.1 Modulo per l'inserimento di nuovi video

Per raggiungere l'obbiettivo sopra citato il database da compilare dovrà essere ampio, sarà necessario quindi un nuovo modulo software che permetta di inserire nuovi video in modo automatizzato. Si procederà con l'analisi di cosa è necessario fare per inserire un nuovo video. Innanzi tutto, sarà necessario rendere il video compatibile con Java Media Framework, che è

il framework che viene utilizzato per l'analisi dei video. In seguito il video dovrà essere analizzato per stabilire in quali punti dovrà essere tagliato per poter generare gli shots; si dovrà procedere poi al taglio del video in shots e all'estrazione, da questi ultimi, dei keyframes. I keyframes dovranno poi essere analizzati per poter estrarne le features. Di quest'ultima fase si occuperà la libreria Windsurf. E' evidente che questa operazione sarà molto lunga, quindi bisognerà agire tenendone conto.

3.2 Collezione video

L'aspetto principale di questo problema è la costruzione del database vero e proprio. Esso, come si è detto, dovrà essere di grandi dimensioni, ma sarà opportuno scegliere i video con cura in modo che il maggior numero possibile di eventuali test vi sia applicabile. Sarà opportuno considerare sempre quale sistema che dovrà essere testato su questa base di dati: Shiatsu, un sistema per il tagging automatico dei video. Sarà quindi opportuno scegliere video di varia natura, dimensione e contenuto il più possibile completi di metadati, in modo da avere un termine di confronto sui tag che Shiatsu eseguirà. Inoltre, essendo questo un progetto accademico, i video dovranno essere privi di Copyright, in modo da poter essere utilizzati liberamente.

Capitolo 4

Progetto del Database

4.1 Introduzione

La creazione di una base di dati è un lavoro complesso, in questo caso particolarmente. Infatti il nostro database era piccolo e molto mirato. Come si è detto creato ad Hoc con strumenti esterni. Anche lo schema del database rispecchiava quella modalità. Si rendeva quindi necessario, una volta trovati i dati adatti da inserire, riadattare anche lo schema del database cercando di limitare il più possibile di danneggiare il funzionamento di Shiatsu. Dopo alcune ricerche, si è deciso di utilizzare un famoso Benchmark già creato da altri: il TRECVid[9].

4.2 Trecvid

Trecvid è una base di dati nata dalla serie di conferenze Trec:

> "The TREC conference series is sponsored by the National Institute of Standards and Technology (NIST) with additional support from other U.S. government agencies. The goal of the conference series is to encourage research in information retrieval by provi-

> ding a large test collection, uniform scoring procedures, and a forum for organizations interested in comparing their results. In 2001 and 2002 the TREC series sponsored a video "track" devoted to research in automatic segmentation, indexing, and content-based retrieval of digital video. Beginning in 2003, this track became an independent evaluation (TRECVID) with a workshop taking place just before TREC."[10]

Il TREC è una serie di conferenze mirate a incoraggiare la ricerca sull'estrazione di informazioni da video digitali. Per farlo offre ogni anno una grande collezione di video completi di Ground Truth[1] e metadati.

4.2.1 Organizzazione dei dati

I dati video del TREC sono divisi in 2 parti: una a scopo di Development e l'altra a scopo di Test. Per ognuna sono allegati dati diversi, ma per entrambe sono presenti la Ground Truth e la divisione in Shot. Per questo lavoro di tesi è stata ottenuta solo una parte dei dati disponibili per l'anno 2007 e una parte dell'anno 2009. In particolare, per l'anno 2007, si era in possesso di dati quali: annotazioni, divisione in shot, ground truth, keyframe della parte "dev" e 40 video della parte test grazie ai quali è stato possibile creare un buon benchmark per il nostro sistema Shiatsu, adatto soprattutto a testare la divisione in shots e il tagging automatico dei video. I dati relativi all'anno 2009 risultano essere in maggior numero ma ancor più parziali, infatti vi è una notevole quantità di video: 127. Tuttavia non vi è praticamente alcun metadato collegato ad essi; nonostante ciò è possibile utilizzare tali video per creare un data set al fine di testare la scalabilità delle prestazioni di Shiatsu e delle query. Utilizzando questi dati verranno creati 3 database con tre scopi distinti l'uno dall'altro:

[1] Per Ground Truth si intende il set di informazione che viene utilizzato come termine assoluto di paragone per misurare la qualità dei risultati

trecvid_benchmark sarà realizzato a partire dai 40 video del Trecvid 2007, che verranno tagliati seguendo la divisione in shot di Trecvid e importando annotazioni e Ground Truth.

trecvid_shiatsu verrà creato utilizzando gli stessi 40 video del Trecvid 2007, ugualmente tagliati secondo i dati Trecvid ma importando le features visuali secondo le modalità Shiatsu.

big_data per il quale verranno importati i video del Trecvid 2009 seguendo interamente le modalità Shiatsu Il primo Database può essere impiegato per testare l'efficacia del tagging automatico di Shiatsu, mentre è possibile fare uso del secondo per collaudare il nostro algoritmo di divisione in Shots. "big_data", essendo molto grande, può essere adoperato per testare l'efficienza delle query.

4.2.2 Metadati

Come appena riportato, i dati del 2007 erano completi di metadati pur senza troppe descrizioni. E' stato necessario studiare ogni file in relazione con gli altri per capire come interpretare i dati e come utilizzarli. Di seguito viene riportata la spiegazione di ciascun elemento.

Collection2007.xml

E' un file in formato xml contenente l'elenco di tutti i video del Trecvid 2007. Ogni campo contiene:

id un numero da 1 a 217 che indica l'id associato al video per Trecvid.

filename il nome del file video a cui si fa riferimento.

use il pacchetto in cui il video è utilizzato, dev o test.

source l'indicazione di chi ha fornito il video, tipicamente *Sound and Vision* o *BBB rushes*.

filetype la tipologia del video, in questo caso sempre MPEG-1.

Cartella Master Shot Reference

Contiene i file ".msb" e ".sb" che indicano i "master shot bounds" e "shot bounds" di ogni file. Sono file testuali che hanno 2 righe di intestazione in cui si ringrazia Christian Peterson [11] per la realizzazione, dopo di che ci sono 2 valori per ogni riga: il frame di inizio e di fine di ciascuno shot.

Feature definition.txt

In questo file c'è l'elenco dei concetti che Trecvid si propone di trovare in ogni video. E' un file testuale in cui ogni riga indica un *concept* ad esempio:

1. Sports: Shots depicting any sport in action

"1" è l'identificativo associato al concept, "Sports" è il nome del concept e segue una breve descrizione di cosa si vuole cercare.

feature.qrels.tv07

Questo file è la vera e propria Ground Truth per quanto riguarda le annotazioni, segue la descrizione ufficiale:

-TOPIC
-ITERATION
-DOCUMENT#
-RELEVANCY
where TOPIC is the topic number,
ITERATION is the feedback iteration (almost always zero and not used),

> DOCUMENT# is the official document number that corresponds to the "docno" field in the documents, and
>
> RELEVANCY is a binary code of 0 for not relevant and 1 for relevant. [12]

In sostanza, in ogni riga del presente file testuale sono espressi 4 dati separati da spazi, di cui 3 sono molto rilevanti:

Topic è il primo campo, ed è l'identificativo del concept che si vuole individuare.

Document è il nome del keyframe a cui si fa riferimento, si cerca ogni concept in ogni keyframe e se ne verifica la presenza.

Relevancy è l'indicatore di rilevanza del *Topic* nel *Document*: 0 per non rilevante e 1 per rilevante.

4.2.3 Annotazioni

Sebbene anche *feature.qrels.tv07* sia una forma di annotazione, quelle che seguono sono annotazioni fatte da terze parti tutte riferite ai keyframes estratti con la seguente modalità:

> "The keyframes were selected by going to the middle frame of the shot boundary, then parsing left and right of that frame to locate the nearest I-Frame. This then became the keyframe and was extracted. Keyframes have been provided at both the subshot (NRKF) and master shot (RKF) levels."

Files .ann

I file.*ann* prendono il nome del *concept* che riportano. All'interno ogni riga indica la presenza o meno del *concept* nel keyframe indicato, analogamente al *feature.qrels.tv07*.

Annotation_person indica la persona che ha fatto l'annotazione

Organizer chi ha organizzato il processo di annotazione

Concept il concetto che si cerca nei frame

Video Name il nome del video a cui appartiene il keyframe

Keyframe Name il nome del keyframe

Judgement la presenza o meno del concept nel keyframe

Sign indica se l'annotatore era sicuro o no dell'annotazione; è presente S alla fine della riga in caso di incertezza

Sono presenti 2 cartelle con file *.ann*: la prima chiamata *tv2007.mcg.ict.cas.keyframe.annotation* è più completa rispetto alla cartella "ann". Infatti, risulta che qualche elemento del personale abbia dimenticato di salvare parte delle annotazioni nell'elaborare i file .ann. Il gruppo che ha creato *tv2007.mcg.ict.cas.keyframe.annotation* ha corretto questo errore come spiegato nel "readme" allegato

Files .pos.xml

I file ".pos.xml" sono analoghi per struttura agli "ann" ma al loro interno si elencano solo i keyframe "positivi" al concetto ispezionato. Generalmente quando si forniscono questi file si allegano anche i file ".neg" che dovrebbero contenere i match negativi. L'annotazione è stata realizzata secondo il metodo (LSCOM) [13]. Essendo tali file in formato xml sono strutturati: la radice indica il numero di keyframe contenuti nella descrizione e il nome del concept. In successione ogni entità "Img" indica un keyframe e l'attributo "objNum" indica quanti oggetti del dato "concept" sono stati identificati. Se "objNum" è 0 il concept non è identificabile in una sezione dell'immagine; al contrario se objNum è maggiore di 0 sarà presente subito sotto un entità che rappresenta la posizione del concept nel frame.

4.3 Struttura del Database

Alla luce dei dati che si vogliono importare, è risultato evidente che la struttura database presente all'inizio del lavoro e illustrata nel capitolo 2.5.1, sarebbe stata inadeguata. Si sono quindi rese necessarie delle modifiche.

4.3.1 Modifiche alla struttura

Tabella keyframes

La prima modifica che si è deciso di apportare è stata permettere a più immagini di appartenere ad uno stesso shot. Questo non era possibile in precedenza in quanto il campo "id_img" era ForeignKey riferita alla tabella "shots". Le possibili azioni erano 2: creare un nuovo campo nella tabella shot_features da rendere chiave primaria insieme a "id_img", oppure creare una nuova tabella dove associare ad ogni shot più "id_img" e modificare la ForeignKey in "shot_features" in modo che puntasse tale tabella. Nella seguente immagine si può vedere la modifica effettuata.

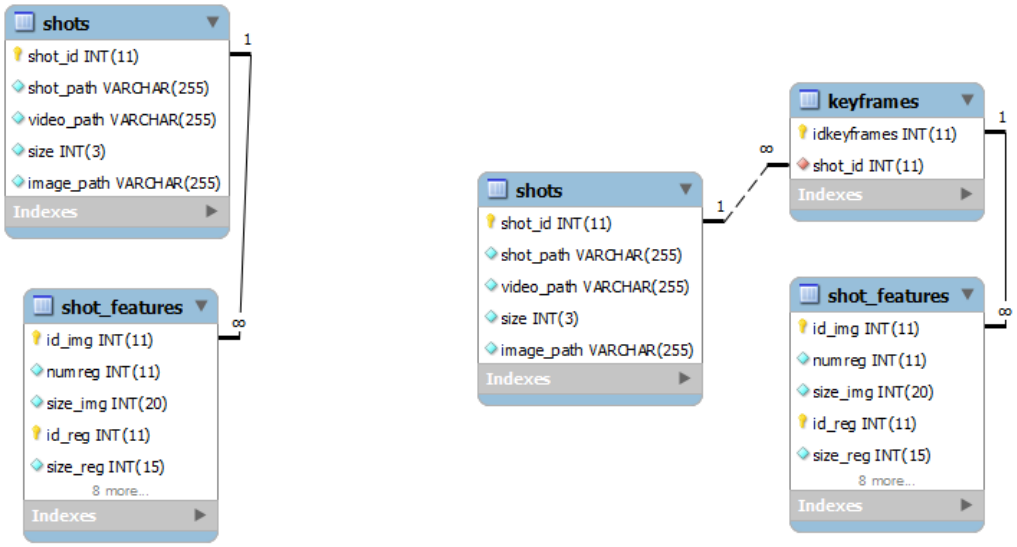


Figura 4.1: La differenza tra il vecchio modello (a sinistra) ed il nuovo modello (a destra).

In seguito è stato necessario eliminare la ForeignKey nella tabella tag_shots che aveva un funzionamento analogo.

Tabella videos

La tabella *videos* aveva un solo campo: il "video_path" che usava per identificare i video. Si è deciso di inserire 2 campi chiamati "original_hash" e "mjpg_format_hash" che memorizzano il checksum di ciascun file da inserire, prima e dopo la conversione. Questo passaggio verrà spiegato in seguito. Questi campi devono essere unici.

Tabella video_meta

La tabella *video_meta* è stata creata per ospitare i metadati di Trecvid. E' stato deciso di creare una tabella apposita invece che aggiungere campi alla tabella *videos*. In questo modo sarà più facile e veloce utilizzare la tabella

videos per individuare i video, mentre la tabella *video_meta* viene letta solo quando occorre estrarre i metadati.

Tabella cutpoints

La tabella *cutpoints* ha il solo scopo di memorizzare gli istanti di taglio di ogni video. Per discriminare ogni shot le chiavi primarie sono il "video_path" e il numero sequenziale dello shot, mentre il campo "frame" è di tipo BIGINT dato che vengono gestiti come *Long*. Ogni riga della tabella indicherà il frame finale di ogni shot riferito al video completo, presumendo che il frame di partenza sia 0.

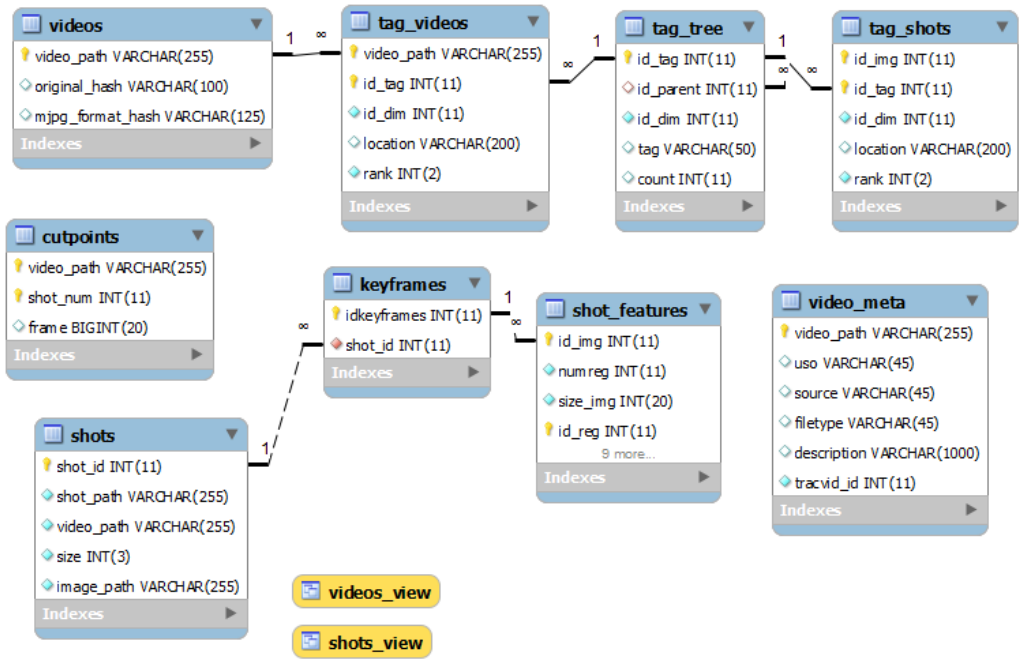


Figura 4.2: Schema ER della struttura modificata del database Shiatsu.

4.3.2 Adattamento dei dati alle tabelle esistenti

Le modifiche sopra descritte erano strettamente necessarie in quanto i dati a cui sono destinate non avevano posto in database. Invece una parte

dei dati da inserire possono essere adattate allo schema preesistente. L'elenco dei *concept* può essere inserito efficacemente nella tabella *tag_tree* che aveva lo stesso scopo. E' stato preso l'accorgimento di inserire i tag in un nuovo ramo della tabella, creando il tag *"Inported Tag"* con *id_tag* = 1000 ed *id_parent* = 1000. Di seguito sono stati creati i tag *"annAnnotation"*, *"posXMLAnnotation"* e *"tv07Annotation"* rispettivamente con *id_tag* 1100, 1200 e 1300, e con lo stesso genitore id_parent 1000. I *concept* a loro volta avranno un *id_tag* a partire da 1100 per le annotazioni prese dai file di tipo "ann". Es.

id_tag 1101, *id_parent* 1100 = Sports (da annotazione "ann"),

id_tag 1201, *id_parent* 1200 = Sports (da annotazione "pos.xml")

id_tag 1322, *id_parent* 1300 =Corporate-Leader (da annotazione "tv07")

E' quindi possibile inserire le annotazioni sotto forma di tag nella tabella *tag_shots* anche se in questo caso non tutte le informazioni sono perfettamente compatibili; si perdono ad esempio quelle relative agli oggetti presenti nelle annotazioni "pos", ma per i nostri fini le informazioni memorizzate sono sufficienti.

Capitolo 5

Realizzazione del Crawler

5.1 Introduzione

L'inserimento di nuovi video, come si è detto, richiede la creazione di un modulo software da inserire in Shiatsu. In questo capitolo verranno esposti i problemi incontrati nella sua realizzazione e come tali problemi sono stati risolti.

5.2 Divisione del Lavoro

Dopo aver constatato la lunghezza eccessiva del processo di inserimento, si è deciso di utilizzare uno strumento Java chiamato SwingWorker che permette di spostare il carico di lavoro su un nuovo Thread, in modo da lasciare il programma principale in esecuzione senza bloccarlo. La classe che implementa *"IImportGui.java"* deve occuparsi di configurare il "Worker" e di mantenere il dialogo con esso. Il protagonista principale del lavoro di inserimento sono le classi *"IWorkHandler.java"* che estendono SwingWorker. In questo lavoro di tesi si sono realizzate 3 implementazioni, ognuna con scopi diversi: *"DefaultWorkHandler.java"*, *"TrecImporterWorkHandler"* e *"PersonalizedWorkHandler"*. Dopo essere stata configurata e avviata da *"Import-*

Gui.java", questa classe genera un nuovo Thread e comincia il lavoro sul file. I vari *WorkHandler* hanno a disposizione svariate altre classi ognuna con un compito specifico da svolgere all'interno del lavoro di inserimento.

5.3 Struttura del Crawler

Si è deciso di strutturare il codice seguendo il più possibile il *"Single Responsability Pattern"*, ovvero di dare ad ogni classe uno ed un solo compito da svolgere. Questo garantisce un'alta riusabilità del codice. Inoltre, si è fatto largo uso di Interfacce per consentire una più agevole estensione delle opzioni di taglio. In questo lavoro di tesi è presente tipicamente una sola implementazione di ciascuna Interfaccia, ma in alcuni casi sono già presenti più implementazioni. Tra queste è possibile scegliere la più adeguata al dato da inserire. Di seguito lo schema UML del codice.

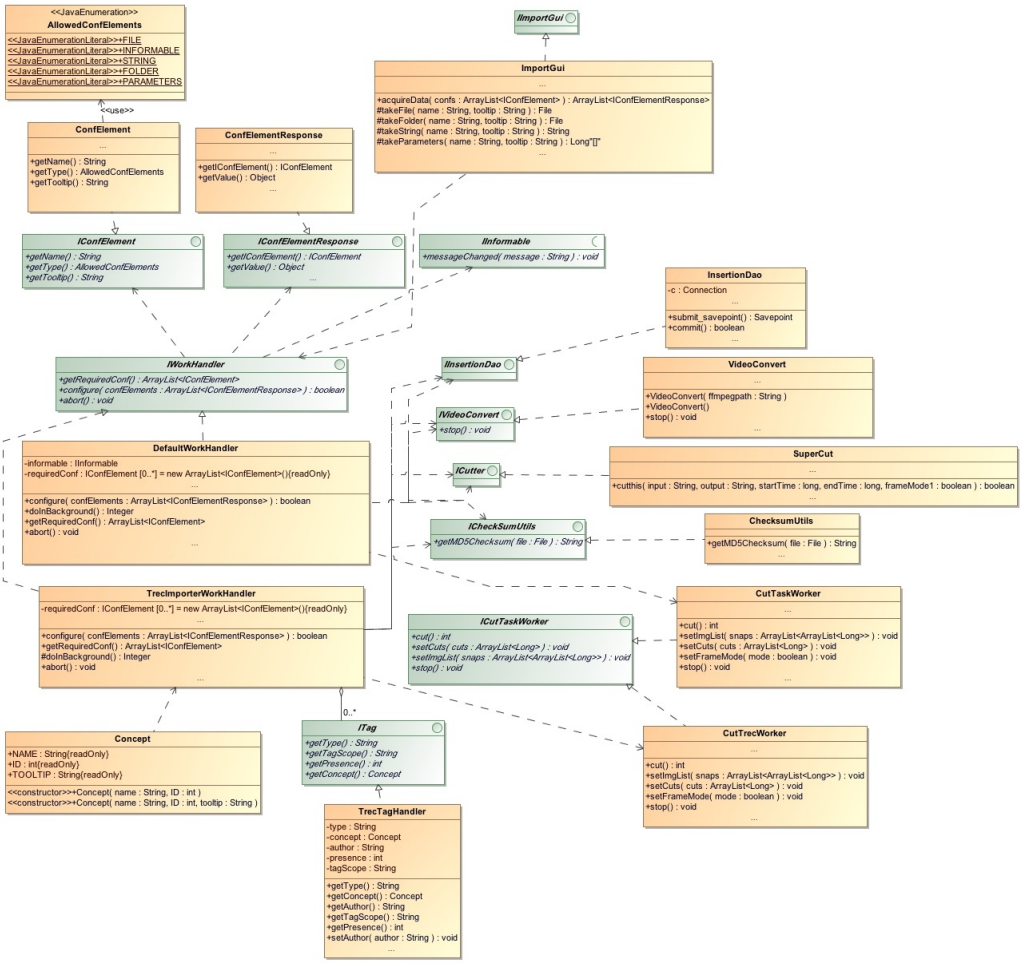


Figura 5.1: L'UML del Crawler

5.3.1 InsertionDao

Si è scelto di adoperare il pattern *"Dao"* per interfacciasi al database. All'interno della classe *"InsertionDao.java"* sono implementati metodi che, sfruttando query JDBC[7], inseriscono, estraggono e manipolano i dati per facilitarne la fruizione.

5.3.2 Interfaccia Grafica

AcquireDataGui

Dal momento in cui l'utente sceglie di inserire un nuovo video nella finestra principale di Shiatsu, sarà istanziata la classe*AcquireDataGui.java* che, tramite un'interfaccia grafica, chiederà all'utente che tipo di inserimento vuole effettuare. In questo ambito di fatto si seleziona quale *"WorkHandler.java"* istanziare. Una volta selezionato il modo dal menù a tendina e premuto il tasto *Next AcquireDataGui.java* istanzierà *"ImportGui.java"* per il passaggio successivo.

ImportGui

"ImportGui.java" ha il compito di configurare e comunicare con il WorkHandler. La struttura delle classi in questa sezione è complessa. Infatti avendo previsto che ogni WorkHandler possa avere una configurazione diversa e diverse necessità, si è deciso di creare una struttura per la configurazione dinamica dei WorkHandler. Allo stesso tempo questo serve per la richiesta dinamica di input. Di seguito si può vedere la sequenza degli eventi:

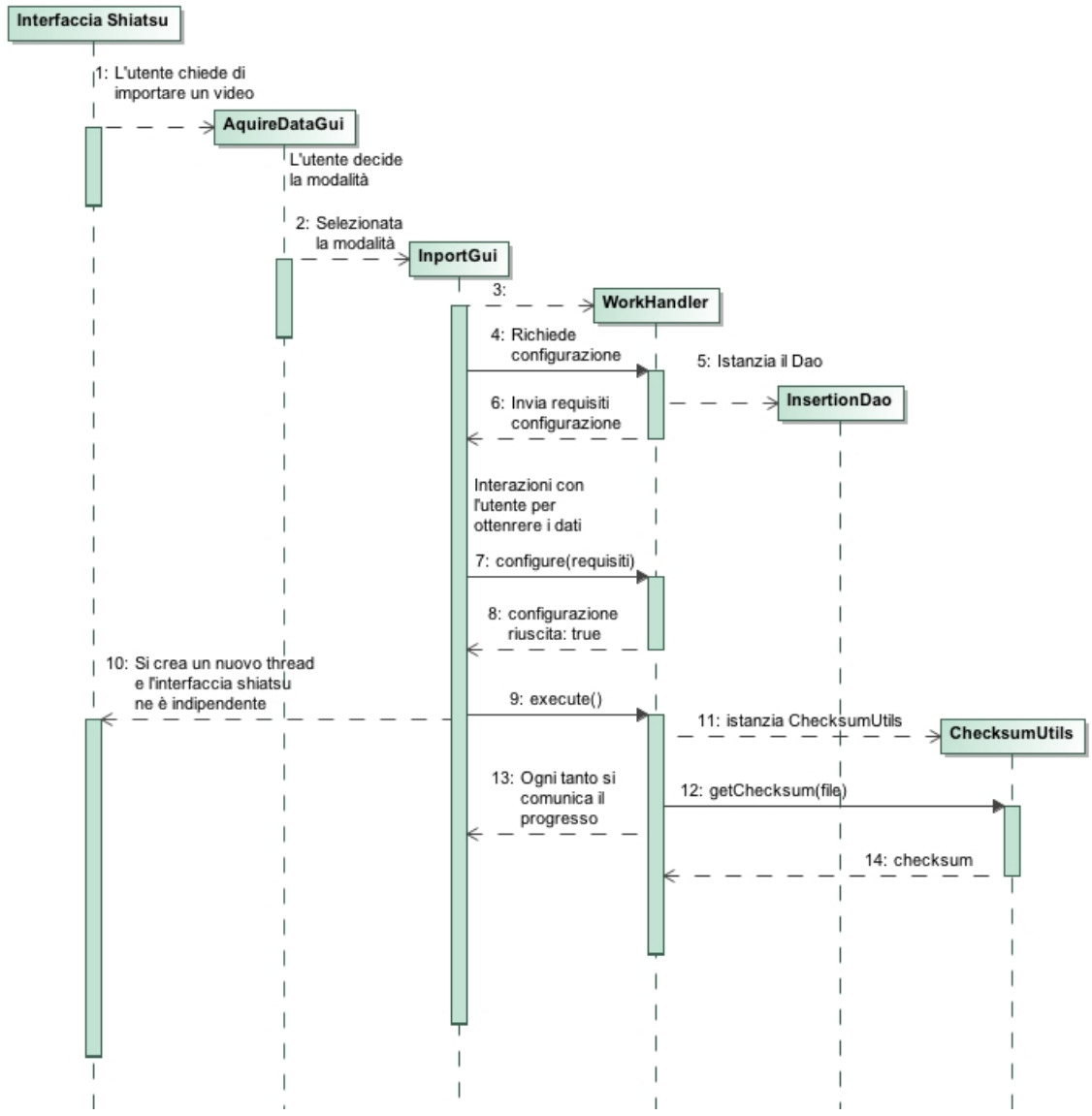


Figura 5.2: La sequenza degli eventi per la configurazione

Ogni WorkHandler deve avere cablato nel codice un ArrayList di *"IConfElement"* cioè ConfigurationElement, formato da un nome, un tipo e un tooltip. Per ogni elemento l'interfaccia grafica fa una richiesta all'utente, andando poi a riempire un array di *IConfElementResponse*. Quindi l'array di risposta viene utilizzato dal WorkHandler per eseguire la configurazione.

Per ora è possibile inserire files, cartelle o stringhe in quanto non risultava necessario inserire altro, tuttavia anche queste opzioni sono facilmente estendibili.

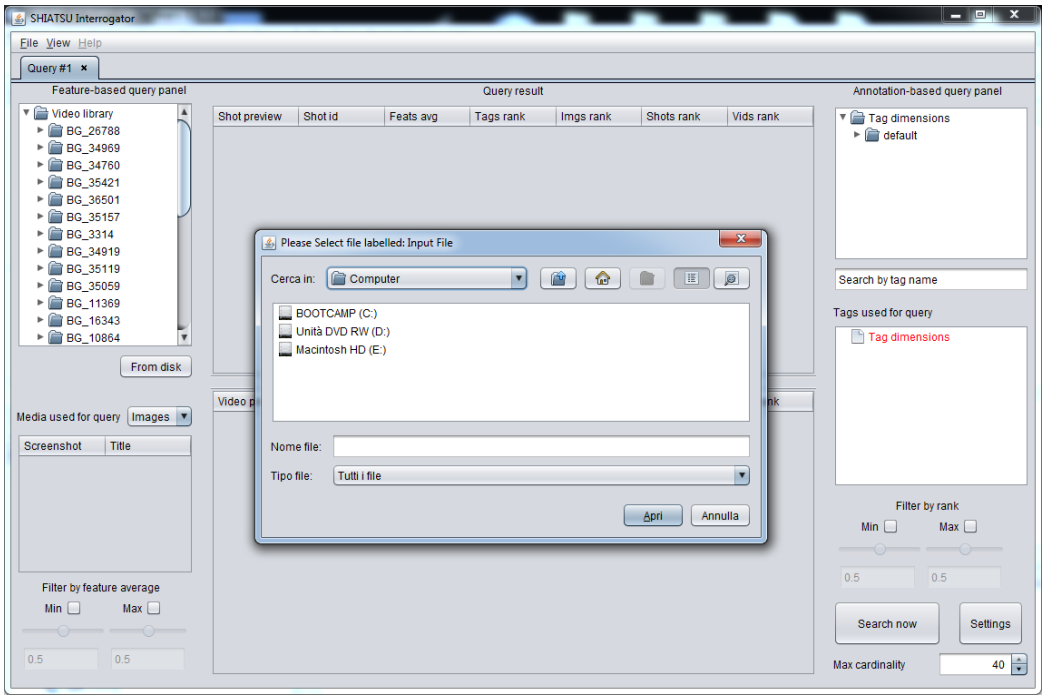


Figura 5.3: Esempio di Requisito: selezione di un file

5.3.3 WorkHandler

Configurazione e controlli iniziali

Prendendo l'array di ritorno *IConfElementResponse*, ciascun WorkHandler si configura secondo le proprie esigenze, inoltre legge dal file di configurazione *Shiatsu.xml* per ottenere i dati di connessione al database e istanzia l'*InsertionDao*, la classe responsabile di dialogare col database. Se la configurazione è andata a buon fine WorkHandler restituisce **true** ad *"ImportGui.java"* che a sua volta lo avvia per il lavoro vero e proprio. Essendo WorkHandler un'estenzione di SwingWorker, quando *"ImportGui.java"* lancia il metodo *"execute()"*, viene generato un nuovo Thread dove proseguirà

l'esecuzione. Appena avviato il WorkHanlder chiederà all'*InsertionDao* di disabilitare l'*autocommit* e di generare un *savepoint* che WorkHandler salverà. In questo modo qualsiasi modifica apportata al database non sarà definitiva finchè non verrà dato il commit. Questo facilita la pulizia dei dati incompleti in caso di errori o interruzioni da parte dell'utente. All'inizio del lavoro, il WorkHandler controlla che il file da elaborare non sia presente nella cartella *DB_VIDEO/data/sisap* né nel database. Se trova un match viene proposto all'utente un cambio di nome che viene operato dallo stesso WorkHandler. Tutte le operazioni atte alla fine del taglio vengono gestite da questa classe, che delega a chi di dovere per le singole operazioni ma raccoglie sempre i risultati e opera i controlli.

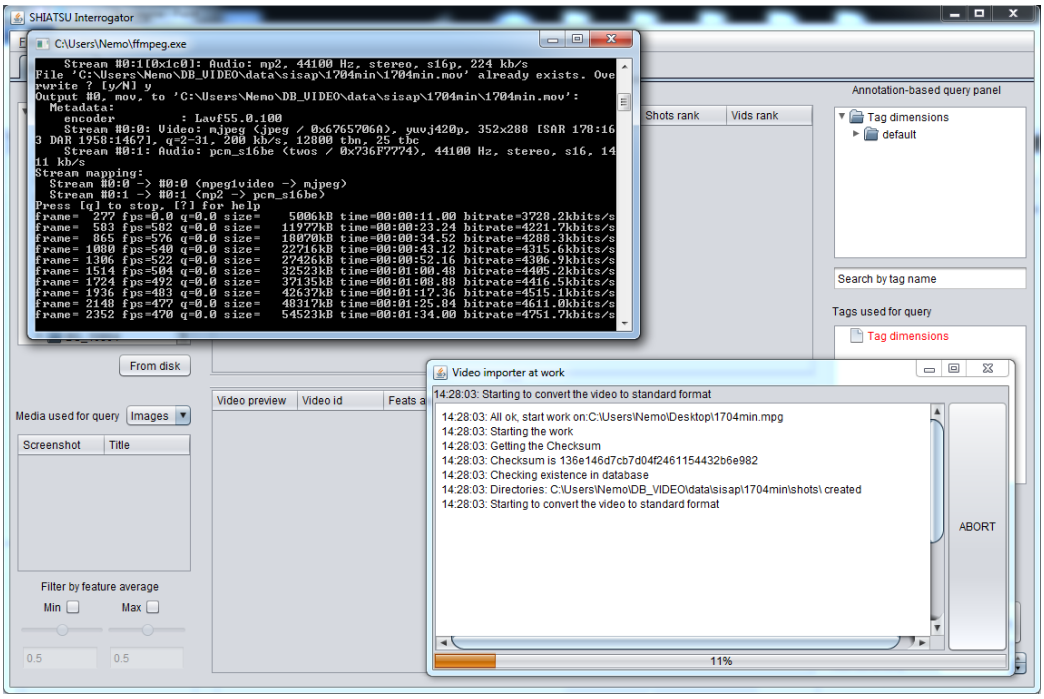


Figura 5.4: Un esempio di come si presenta l'interfaccia durante l'inserimento di un file

Calcolo del checksum: ChecksumUtils.java

Per verificare che uno stesso video non venga inserito più volte se ne calcola il checksum, si controlla che non esista in database un video con lo stesso valore. Questa estrazione viene attuata dalla classe *ChecksumUtils.java* mediante l'algoritmo *MD5*[14] che sta per *Message Digest algorithm 5*; essendo molto veloce, MD5 è perfetto per le nostre semplici esigenze.

Conversione in formato standard Shiatsu: VideoConvert.java

La prima fase nella lavorazione vera e propria del video deve consistere nella sua conversione, questo perché il framework JMF ha una scarsissima compatibilità, specialmente per il flusso video in scrittura. Inoltre nella fase di studio di questo sistema, si è scoperto che era molto difficile far eseguire il Detector di cambi di scena con successo. Questo perché spesso i file video compressi non contengono tutti i frame, ma piuttosto un keyframe ogni *n* frame, e gli altri *n - 1* sono di tipo differenziale o predittivo o simili. Dato che la logica del nostro detector vuole confrontare ogni frame col successivo, emergono svariati problemi. Per convertire il video si è optato per un tool esterno sebbene invocato dal Java. La scelta è ricaduta su "ffmpeg":

> "FFmpeg is a complete, cross-platform solution to record, convert and stream audio and video." [15]

Questo tool permette di lavorare sui video da linea di comando e supporta tutti i principali formati. Questo era quanto serviva. Dopo svariati tentativi per trovare l'ottimale rapporto qualità/compressione è stata creata la classe *VideoConvert.java* in cui si costruisce di volta in volta il comando da far eseguire al *command* di Windows che invoca a sua volta "ffmpeg".

Uno dei requisiti necessari per il suo utilizzo è quello di conoscere con la posizione del file "ffmpeg.exe". Questo path deve essere specificato dall'utente, alla seconda riga del file VideoProperties.txt. VideoConvert costruisce ed invoca il comando:

```
command3 = "cmd /c start/wait " +ffmpegpath+ " -i \"" +inputpath+ "\" -vcodec mjpeg -q 4 "
        + "-acodec pcm_s16be -ac 2 \"" + outputpath + extension+ "\"";
```

Una volta eseguito il comando e atteso il completamento della conversione, il software controlla che il file sia stato correttamente creato e convertito. Di seguito estrae il checksum del nuovo file e verifica nuovamente la sua presenza in database. Se questo processo è stato eseguito correttamente si inserisce il nuovo video in database nella tabella *videos*.

5.3.4 Individuazione dei punti di taglio: Threshold

Al termine della preparazione si ha a disposizione un video che è possibile analizzare e modificare: è il momento di trovare i punti di taglio. All'inizio del lavoro di tesi era presente una classe chiamata TresholdDetector che in teoria avrebbe dovuto elaborare il video per trovare i cutpoints; sfortunatamente questa classe presentava gravi bug ed in definitiva non funzionava. Il tesista Giovanni Concas, avendo anch'egli bisogno della funzione di Treshold, si era adoperato per il fix dei bug, ma nonostante il software sembrasse funzionare adeguatamente, i punti di taglio che individuava erano errati. Questo risultava evidente non appena si andava a riprodurre il video manualmente; si notava che i punti di taglio indicati erano in ritardo di qualche secondo rispetto all'effettivo cambio di scena. Tale comportamento indicava che il sistema individuava effettivamente il cambio di scena, tuttavia erano presenti problemi relativi alla gestione del flusso video. Inoltre, se si analizzava lo stesso filmato ma in formato video ".avi" non compresso, il software funzionava perfettamente. Dopo varie ricerche e test si è giunti alla conclusione che

il problema era nella compressione dei video: “MPEG-2”. In quel momento si è deciso di convertire ogni video da importare in Shiatsu. Il formato scelto con molta attenzione è .mov, con compressione MJPEG. Tale configurazione garantiva massima compatibilità e allo stesso tempo dava una certa compressione, seppur non molto alta, risolvendo in definitiva il problema dell'individuazione dei punti di taglio. Il Tresholding è stato poi ristrutturato dal tesista Alessandro Colace che si è adoperato per offrire flessibilità al codice. Il processo è davvero molto lungo, sui video testati dura mediamente molto più di un'ora. Alla fine può restituire l'elenco dei punti di taglio in frame o nanosecondi, includendo lo 0 e la fine del video. Inoltre, genera i file immagine detti keyframe secondo varie modalità e li inserisce nella cartella adatta.

5.3.5 Taglio del Video ed estrazione delle features: CutTaskWorker

Il Detector di tagli esegue l'analisi del video e restituisce un *Vector<Long>* di punti di taglio che viene subito convertito in un ArrayList<Long>. WorkHandler istanzia il giusto CutWorker, una delle classi che implementa ICutTaskWorker, e gli passa il necessario per l'esecuzione. Nell'importazione di default viene istanziato il CutTaskWorker che ha bisogno di:

il file da tagliare

il path della cartella sisap

un'istanza del InsertionDao

un'istanza dell'interfaccia con l'utente

i punti di taglio

Una volta avviato il CutTaskWorker istanzierà *SuperCut.java* che è la classe che di fatto taglia il video, generando da esso un nuovo video della lunghezza

voluta e comincerà il ciclo di tagli. Ad ogni ciclo si deve ricorrere a *SuperCut.java* per tagliare il video, successivamente, inserendolo in database verrà generato un **ID** che deve essere raccolto ed utilizzato per l'inserimento delle features. A questo punto del lavoro, esplorando la cartella dei Keyframes vengono letti quelli appartenenti allo Shot appena inserito in DB; di ogni immagine se ne calcolano le features utilizzando la libreria Windsurf, si provvede poi ad inserire anche queste in database, sempre stampando a video gli ID delle immagini ed il ciclo in corso. Nel momento in cui Windsurf si trova ad elaborare un'immagine totalmente nera, genererà un'eccezione. questo a causa della sua politica di estrazione features.[16] Anche questa operazione e abbastanza onerosa in termini di tempo seppur neanche vicina al Tresholding, ma riciclando sempre le risorse è possibile portarla a termine anche con file piuttosto pesanti. E' possibile, soprattutto su macchine cariche ed in ogni caso piuttosto raramente, che, durante l'operazione di "Prefetching" del processor, il *SuperCut.java* vada in deadlock. Purtroppo non è stato possibile evitare tale inconveniente dal momento che tutto il processo viene gestito dalla JMF.

5.3.6 Conclusione del lavoro

Finiti i cicli di taglio il controllo torna al WorkHandler, che si occupa di dare il commit al database, in modo che tutte le aggiunte apportate finora siano confermate. Infine, chiude tutte le risorse aperte e stampa a video la conclusione. E' importante cercare di chiudere tutte le risorse aperte perché secondo le politiche di SwingWorker il Thread, nonostante abbia finito di eseguire, rimane aperto in attesa di un altro lavoro da eseguire, quindi è importante lasciarlo il più scarico possibile.

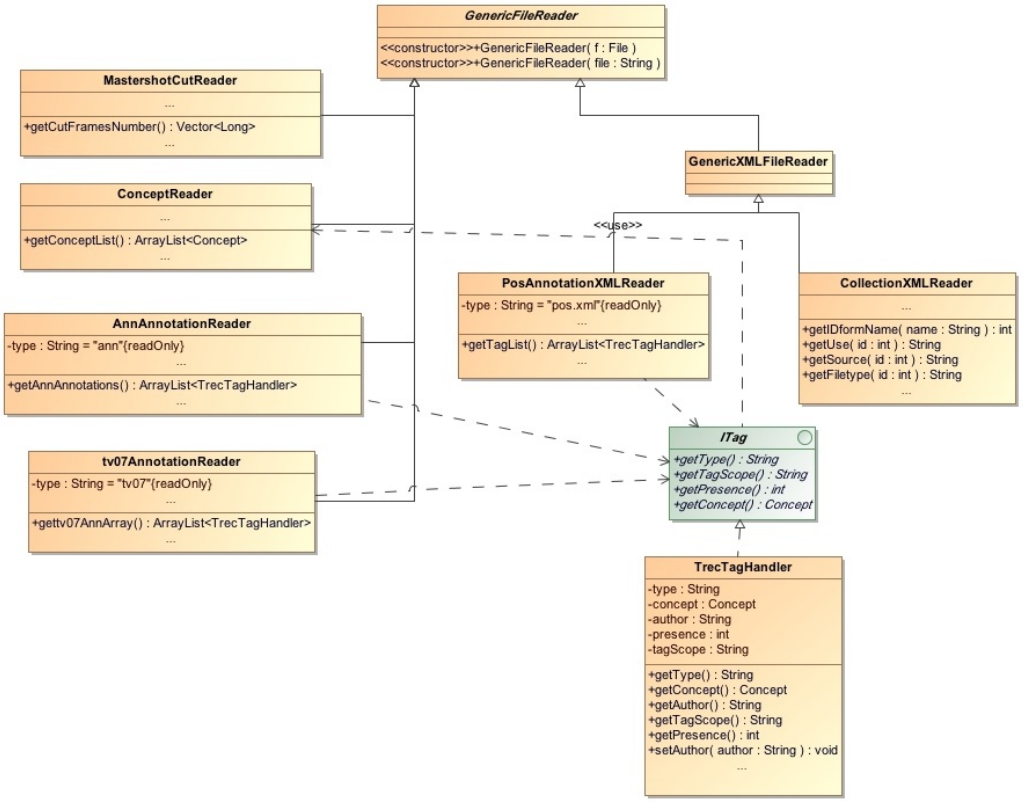
5.4 Implementazioni aggiuntive

5.4.1 WorkHandler alternativi

Le classi descritte fin'ora descrivono il comportamento del DefaultWorkHandler, cioè il normale processo di inserimento di un nuovo video in Shiatsu. Nel processo di lavorazione, col fine di popolare diversi database, è stato implementato anche il *TrecImporterWorkHandler*: una versione modificata ed adattata del DefaultWorkHandler utilizzata per popolare il database "trecvid_benchmark". Questo WorkHalder ha un setup più specifico estrae moltissime informazione dai file contenenti i metadati come verrà descritto nella prossima sezione; esso non invoca CutTaskWorker, ma bensì il CutTrecWorker, una versione a sua volta modificata che non estrae le features dalle immagini ma inserisce altre informazioni in DB.

5.5 File Reader

I metadati allegati a Trecvid sono inseriti in normali file, per importare i dati da questi file si sono resi necessari script specifici per ogni file. Come per il resto del lavoro si è cercato di rendere il tutto il più estendibile possibile, facendo uso di interfacce e classi astratte. Di seguito lo schema UML.



Come si può vedere ogni classe File Reader eredita dalla classe astratta GenericFileReader e i file xml hanno un ulteriore supertipo GenericXMLReader. Come è ovvio non è possibile generalizzare più di tanto uno script che deve essere specifico per un certo lavoro, quindi una volta realizzate queste classi astratte si è passati all'implementazione degli script di lettura dei singoli file.

Collection.xml

Il file "collection.xml" è un elenco strutturato di caratteristiche di file video. Viene letto da CollectionXMLReader.java tramite il quale è possibile effettuare ricerche all'interno dei dati.

Feature list.txt

Feature List.txt contiene un elenco di tutti i tipi di concetto che possono essere presenti nei video. Ad ogni concetto (es. Sport) è associato un numero ed una piccola descrizione. Leggendo questo file la classe ConceptReader.java crea un ArrayList di Concept, classe creata appositamente per contenere un singolo concept. In questo modo una volta letto il file ne siamo indipendenti.

Master Shot Record

La cartella Master Shot Record include un file per ogni video, in cui sono indicati i punti di taglio esatti per quei file. La classe MastershotCutReader si occupa di legger il file e generare un Vector di long contenente i punti di taglio espressi in frame.

TrecTagHandler

La classe TrecTagHandler è adatta a contenere una singola annotazione. Nella costruzione di un TrecTagHandler vanno specificati: il tipo di annotazione (pos, xml, tv07), il "Concept", il keyframe a cui è riferita l'annotazione e la presenza del concept nel frame. Quest'ultimo valore è specificato tramite un int e può presentarsi come:

0 il Concept non è presente.

1 il Concept è presente.

-1 null, non c'è certezza.

2+ sono presenti più oggetti nel frame.

Annotation Readers

Le classi *AnnAnnotationReader.java*, *PosAnnotationReader.java* e *tv07AnnotatioReader.java* hanno lo scopo di leggere i file contenenti le annotazioni. Ciascuna di esser compila un ArrayList di *"TrecTagHandler*.

5.6 Dati prodotti

Una volta completato, testato e debuggato il crawler è stato possibile utilizzarlo per popolare i database che si voleva creare. Sono stati prodotti il database "trecvid_benchmark" è "big_data", che unitamente al database "tracvid_shiatsu" prodotto da Alessandro Colace con questo stesso crawler forniscono una piattaforma completa per il testing di Shiatsu. Di seguito riportiamo un grafico che illustra la quantità di dati inclusa nei vari database

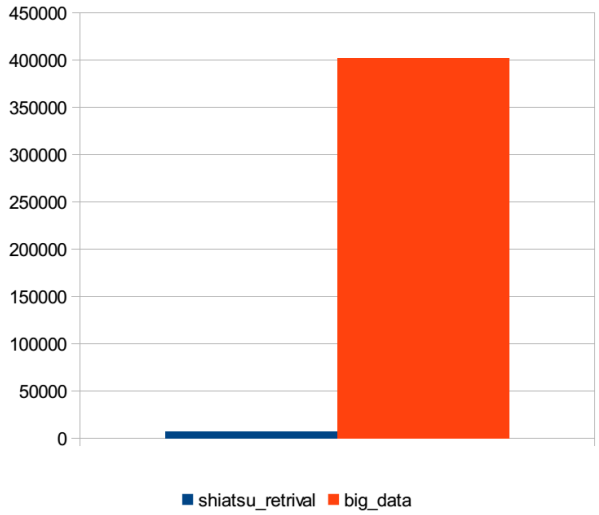


Figura 5.5: Un grafico che indica le righe presenti nel database originale ed in big_data

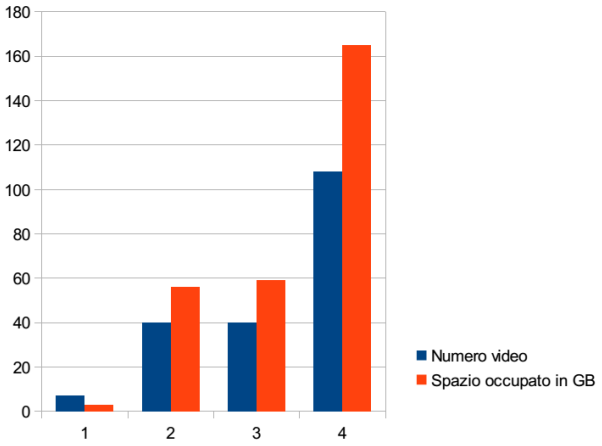


Figura 5.6: Un grafico che indica il numero dei video presenti e la dimensione in GB rispettivamente di: shiatsu_retrival, trecvid_benchmark, trecvid_shiatsu, big_data

Capitolo 6

Conclusioni

L'obiettivo posto inizialmente, cioè la progettazione e la creazione di una piattaforma di benchmark per Shiatsu, è stato pienamente raggiunto. Si sono infatti creati 3 database per diversi scopi di testing. La creazione di uno strato software per l'inserimento di ulteriori video permette il facile ampliamento dei database, anche di tipi diversi. Shiatsu adesso è pronto per essere testato nella sua interezza ed in modo appropriato. Per quanto riguarda gli sviluppi futuri sarebbe conveniente, benché impegnativo, sostituire il framework JMF. Quest'ultimo dà troppe limitazioni e problemi, potrebbe essere sostituito da qualche libreria più moderna, compatibile ed efficiente, come ad esempio Xuggle[17] o Jffmpeg[18]. In questo modo l'inserimento di nuovi video sarebbe ancor più efficiente e il volume occupato dal database si ridurrebbe drasticamente grazie a compressioni più efficaci. In futuro un altro elemento da completare potrebbe essere quello relativo alla creazione di un nuovo database partendo da dati Trecvid aggiornati e completi in modo da ottenere una piattaforma di benchmark pienamente affidabile.

Bibliografia

[1] *SHIATSU: Semantic-Based Hierarchical Automatic Tagging of Videos by Segmentation using Cuts.* **Bartolini, Ilaria, Patella, Marco e Romani, Corrado.** 2010. Proceedings of the 3rd International ACM MM Workshop on Automated Information Extraction in Media Production (AIEMPro10).

[2] The Java Media Framework API (JMF) enables audio, video and other time-based media to be added to applications and applets built on Java technology. This optional package, which can capture, playback, stream, and transcode multiple media formats, extends the Java 2 Platform, Standard Edition (J2SE) for multimedia developers by providing a powerful toolkit to develop scalable, cross-platform technology.

[3] Windsurf: Region-Based Image Retrieval Using Wavelets. **Ardizzoni, Stefania, Bartolini,Ilaria e Patella, Marco.** 1999. Proceedings of the 1st Workshop on Similarity Search (IWOSS 1999 - DEXA 1999).

[4] *"Progettazione e realizzazione di un tool per la segmentazione automatica di collezioni video".* **Alessandro Colace**. 2013

[5] **Giovanni Concas** tesista specialistico per il progetto SHIATSU, indirizzato allo sviluppo di un segmentatore video su dati audio.

[6] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html

[7] Java DB is Oracle's supported distribution of the open source Apache Derby database. Its ease of use, standards compliance, full feature set, and small footprint make it the ideal database for Java developers. Java DB is written in the Java programming language, providing write once, run anywhere portability. It can be embedded in Java applications, requiring zero administration by the developer or user. It can also be used in client server mode. Java DB is fully transactional and provides a standard SQL interface as well as a JDBC 4.1 compliant driver. http://www.oracle.com/technetwork/java/javase/jdbc/index.html

[8] *Progettazione e realizzazione di una interfaccia grafica per l'interrogazione di collezioni video*, **Emanuele Casadio, 2010/2011**

[9] @inproceedings2012trecvidover, author= Paul Over and George Awad and Martial Michel and Jonathan Fiscus and Greg Sanders and Barbara Shaw and Wessel Kraaij and Alan F. Smeaton and Georges Quéenot, title = TRECVID 2012 – An Overview of the Goals, Tasks, Data, Evaluation Mechanisms and Metrics, booktitle = Proceedings of TRECVID 2012, keywords=TRECVid, Video Retrieval, Multimedia Retrieval, IR Evaluation, year = 2012, organization = NIST, USA, pdf=http://www-nlpir.nist.gov/projects/tvpubs/tv12.papers/tv12overview.pdf,

[10] Stéphane Ayache and Georges Quénot, "Video Corpus Annotation using Active Learning", 30th European Conference on Information Retrieval (ECIR'08), Glasgow, Scotland, 30th March - 3rd April, 2008 URL: http://mrim.imag.fr/georges.quenot/articles/ecir08.pdf

[11] Automatic shot boundary detection by Christian Petersohn, Fraunhofer HHI / TU Berlin.

[12] http://trec.nist.gov/data/qrels_eng/

[13] definition on http://en.wikipedia.org/wiki/Large_Scale_Concept_Ontology_for_Multimedia

[14] http://it.wikipedia.org/wiki/MD5 . http://tools.ietf.org/html/rfc1321

[15] FFmpeg is a complete, cross-platform solution to record, convert and stream audio and video. http://www.ffmpeg.org/

[16] *Effcient and Effective Similarity Search in Image Databases*. **Bartolini, Ilaria**. pag.(26-27)

[17] A free open-source library for Java developers to uncompress, manipulate, and compress recorded or live video in real time. http://www.xuggle.com/

[18] The Java Media Framework is the module that handles audio and video files in Java. It is used by many programs to integrate music and movies into a user interface. Included with the programming tools is a simple media player called Java Media Studio (JMStudio). http://jffmpeg.sourceforge.net/