



**HAL**  
open science

## A Distributed Architecture for Flexible Multimedia Management and Retrieval

Mihaela Brut, Dana Codreanu, Stefan Daniel Dumitrescu, Ana-Maria Manzat, Florence Sèdes

► **To cite this version:**

Mihaela Brut, Dana Codreanu, Stefan Daniel Dumitrescu, Ana-Maria Manzat, Florence Sèdes. A Distributed Architecture for Flexible Multimedia Management and Retrieval. 22nd International Conference on Database and Expert Systems Applications (DEXA 2011), Aug 2011, Toulouse, France. pp.249-263, 10.1007/978-3-642-23091-2\_22 . hal-03763188

**HAL Id: hal-03763188**

**<https://hal.science/hal-03763188v1>**

Submitted on 31 Aug 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A Distributed Architecture for Flexible Multimedia Management and Retrieval

Mihaela Brut, Dana Codreanu, Stefan Dumitrescu,  
Ana-Maria Manzat, and Florence Sedes

Universite de Toulouse – IRIT – UMR 5505,  
31062 Toulouse, France  
{Firstname.lastname}@irit.fr

**Abstract.** Developing an efficient system that manages distributed multimedia content supposes to minimize resource consumption while providing the most relevant results for a user's query in the shortest time. This paper presents LINDO, a generic architecture framework for distributed systems that acquires efficiency in multimedia indexing and retrieval. Three characteristics particularize it: (1) it differentiates between implicit algorithms executed over all the multimedia content at the acquisition time, and explicit algorithms, executed on demand for answering a specific need; (2) it stores and processes multimedia content and metadata locally, instead of transferring and indexing it on a central server; (3) it selects a set of relevant servers for query execution based on the user query semantic processing and on the system knowledge, including descriptions of distributed servers, multimedia content and indexing algorithms. The paper relies on a concrete implementation of the LINDO framework in order to validate this contribution.

## 1 Introduction

The development of a distributed multimedia system must balance the efficiency principle to minimize resource consumption while providing the most relevant results for a user's query in the shortest time. The system needs to locate relevant multimedia contents in an environment that consists of an increasing number of machines with different capabilities, each hosting large multimedia collection. The efficient content indexation is a key issue for the management and retrieval of relevant information. Indexing is based on a set of algorithms, which generate diverse and heterogeneous multimedia metadata, but which are usually highly resources consuming.

Designing a distributed multimedia system requires a number of choices [1]: indexing based on a fixed or variable set of algorithms, algorithms executed over the entire multimedia collection or only over a filtered sub-collection, the whole set of algorithms being itself filtered or not before their effective execution, indexing in a distributed manner, on the same location as the content, or in a centralized one, by transferring the content to an indexation server (e.g., Web services), a decision regarding the distributed or the centralized placement of the multimedia metadata.

The LINDO<sup>1</sup> project (Large scale distributed INDEXation of multimedia Objects), specifies a generic architectural solution that guides the design and the development of any distributed multimedia information system relying on indexing facilities. The paper illustrates how LINDO differentiates from other multimedia distributed systems by capitalizing and improving the state of the art results concerning the above mentioned decisions. Thus, three characteristics of the LINDO framework are elicited:

- it differentiates between implicit algorithms executed over all the multimedia content at acquisition time, and explicit algorithms, executed on demand for answering a specific need;
- it processes multimedia content locally, instead of transferring and indexing it on a central server;
- it selects a set of relevant servers for query execution based on the user query semantic processing and on the system knowledge, including descriptions of distributed servers, multimedia content and indexing algorithms.

In this way, the solution we adopted in LINDO prevents from executing at once all indexing algorithms by defining a method that determines the relevant set of algorithms for a user's query. These algorithms are executed only on a multimedia sub-collection, which is also selected according to the query. Indexing algorithms will only be run on the multimedia content location. Algorithms and multimedia content filtering is done with respect to a developed centralized knowledge repository. This repository gives an overview of the system, including semantic descriptions of the distributed servers and indexing algorithms functionalities, as well as summaries of the multimedia metadata extracted and stored on each remote server. It also enables the selection of the relevant remote servers where the user's query will be executed.

Similar approaches adopted by distributed multimedia systems are exposed in Section 2. The architecture, as well as the content indexation and retrieval mechanisms of these systems are presented, while emphasizing the characteristics that differentiate them from LINDO. The LINDO framework is described in Section 3 through its generic architecture, as well as through its indexing and querying mechanisms. A testing implementation of the LINDO system is presented in Section 4, also detailing the architecture topology and the indexing and retrieval mechanisms. Finally, conclusions and future work directions are provided.

## 2 Related Work

The requirements to design an information system that manages distributed multimedia contents are:

- R1. *Fixed or variable set of indexing algorithms (IA) for multimedia contents indexation;*
- R2. *Algorithms executed at acquisition time or at user's query;*
- R3. *Selection of algorithms or not, according to the user's query;*

---

<sup>1</sup> <http://www.lindo-itea.eu/>

- R4. *Distributed executing*, in the same location as the multimedia contents storage, or *centralized*, on an indexation server where the multimedia contents are transferred;
- R5. *Filtering multimedia content* or *not* before indexing;
- R6. *Management of multimedia metadata* obtained as results of indexing process in *distributed way*, on each server that stores multimedia content, or *in centralized one*, through a unique metadata collection;
- R7. At the query moment, *selection* or *not of the relevant remote servers (RS)* according to the query, in order to only send the query to these servers.

In the design of the LINDO framework we considered all these aspects after a careful study of the existing state of the art. Systems and approaches in which multimedia contents are distributed adopt various techniques to accomplish content indexing and retrieval. A large part of these approaches addresses only partially the above mentioned issues according to their main objective.

The CANDELA project (Content Analysis and Network DELivery Architectures)<sup>2</sup> proposes a generic distributed architecture for video content analysis and retrieval [2]. Multiple domain specific instantiations are realized (e.g., personal mobile multimedia management [3], video surveillance [4]). The indexation is done on the distributed servers at acquisition time. The resulting metadata can be distributed over the network. However, the indexation algorithms are a priori selected and pre installed.

The KLIMT project (Knowledge InterMediation Technologies) [5] proposes a Service Oriented Architecture middleware for easy integration of heterogeneous content processing applications over a distributed network. The indexing algorithms are considered as web services. The query is limited to pre-defined patterns that match a set of rules for the algorithms' execution sequence. After such a sequence selection, the content is analyzed and the metadata is stored in a centralized database.

The WebLab<sup>3</sup> project proposes an integration infrastructure that enables the management of indexation algorithms as web services in order to be used in the development of multimedia processing applications [6]. These indexing services are handled manually through a graphical interface. For each specific application a fixed set of indexing tools is run. The obtained metadata is stored in a centralized database.

The VITALAS<sup>4</sup> project (Video & image Indexing and retrieval in the Large Scale) capitalizes the WebLab infrastructure in a distributed multimedia environment [7]. The architecture enables the integration of partner's indexation modules as web services. The multimedia content is indexed off-line, at acquisition time, on different indexing servers. No selection of indexing algorithms based on user query is done.

In [8], the authors propose a system that implements a scalable distributed architecture for multimedia content processing. The architecture is service oriented allowing the integration of new indexing tools. The indexation is distributed and the metadata produced are attached to the multimedia document.

In order to avoid the transfer of the multimedia content in the context of a distributed search engine, [9] propose to use mobile agents that migrate from one server to another for indexing the content. The resulted metadata can be either

---

<sup>2</sup> <http://www.hitech-projects.com/euprojects/candela>

<sup>3</sup> <http://weblab-project.org/>

<sup>4</sup> <http://vitalas.ercim.org/>

centralized or distributed over the network. In the latter case, a user’s query is sent to all the remote servers. The authors prove that transferring the indexing algorithms at the content location is more efficient than transferring the content to a central indexing facility.

**Table 1.** A comparative overview of some representative systems

System name	Set of IA	IA execution moment	IA selection	IA execution location	MM content filtering	Metadata management	RS selection
Candela	Fixed	Acquisition time	Not done	Distributed servers	Not done	Distributed DB	Not specified
KLIMT	Variable	Query moment	Done	Indexation servers	Not specified	Centralized DB	Not done
Weblab	Fixed	Query moment	Done manually	Indexation servers	Not specified	Centralized DB	Not done
Vitalas	Variable	Acquisition time	Not done	Indexation servers	Not specified	Distributed DB	Not specified
[8]	Variable	Acquisition time	Not specified	Distributed servers	Not done	Distributed	Not specified

[10] propose to store on a central server a hierarchy of interest concepts that describe the content stored in the distributed servers. This hierarchy is used to select the servers that are relevant to a query. It is a priori constructed and maintained manually. The authors prove that sending the query to some servers only answers faster than sending the query to each server, while the same precision is maintained.

As can be noticed, each information system for distributed multimedia management considers only a part of the above mentioned issues that contribute to the overall system efficiency. This is the reason why the LINDO framework was developed such as to provide solutions for each issue.

Further we present the LINDO framework and explain how it provides support for acquiring efficiency for the mentioned requirements.

### 3 The LINDO Framework Architecture

The LINDO project’s idea was not to define yet another multimedia information indexing solution but rather to reuse existing indexing frameworks into a common architecture. As illustrated latter, this architecture was designed such as to provide efficient solutions to the mentioned issues in order to enable reduced resource consumption and to enhance the context for giving relevant results to the user query.

We have defined the LINDO generic architecture over two main components: (1) *remote servers* (§3.1) which acquire, index and store multimedia contents, and (2) a *central server* (§3.2) which has a global view of the overall system. Even though our proposal is based on this classical approach for distributed systems, it presents two advantages. First, each remote server is independent, i.e., it can perform uniform as well as differentiated indexations of multimedia contents. For instance, some remote servers may index in real time acquired multimedia contents, while others may

proceed to an off-line indexation. Secondly, the central server can send relevant indexation routines or queries to relevant remote servers, while the system is running.

In the following, the role of each framework's component in the fulfillment of the requirements R1 to R7 mentioned in the beginning of Section 2 is presented.

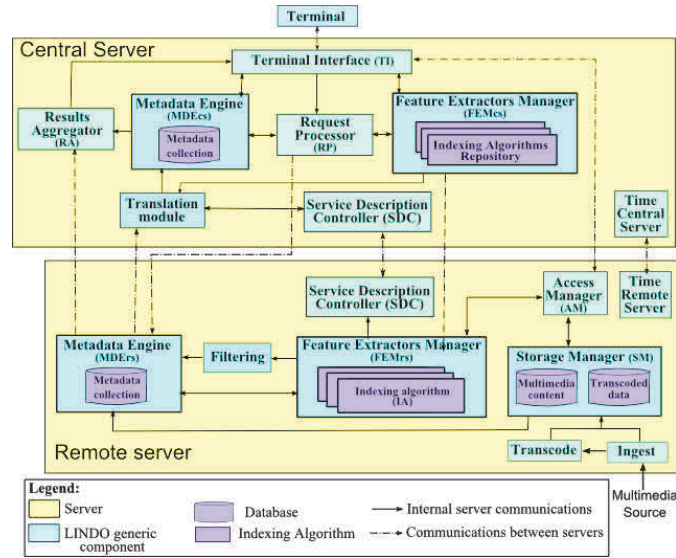


Fig. 1. LINDO Framework Architecture

### 3.1 The Remote Server Components

The remote servers in LINDO-based systems store and index all acquired multimedia contents, to provide answers to user queries. Hence, several modules have been defined and linked together in order to cover all these tasks:

- The *Storage Manager* (SM) stores the acquired multimedia contents. Through the *Transcode* module, acquired multimedia contents can be transcoded into several formats. This allows a user to download different encodings of the desired content.
- The *Access Manager* (AM) provides methods for accessing multimedia contents stored in the SM. Apart from accessing an entire content, different fragments of one multimedia content can be selected (for multimedia filtering, in case of R5).
- The *Feature Extractors Manager* (FEMs) is in charge of managing and executing a set of indexing algorithms over the acquired multimedia contents. At any time, new algorithms can be uploaded into this module, while others can be removed or updated. It can permanently run the algorithms over all the acquired contents or it can execute them on demand only on certain multimedia contents (thus enabling the deployment of the necessary algorithms for a user query for R1 and R2).
- The *Time Remote Server* handles time synchronization with the central server.
- The *Metadata Engine* (MDEs) collects and aggregates all extracted metadata about the multimedia contents stored in the SM. Naturally, the metadata stored in

this module can be queried in order to retrieve some desired information (thus, the distributed management of metadata is enabled for R6).

- The *Service Description Controller* (SCD) stores the remote server description, e.g., its location, its capacities, the acquisition context (useful for enabling the remote servers selection for R5 and R7).

### 3.2 The Central Server Components

The central server can control the remote indexation processes, and it can answer or forward user queries. Thus, a central server is composed of the following components:

- The *Terminal Interface* (TI) enables a user to specify queries and displays the obtained results. Other functionalities are included in the TI, such as visualization of metadata collections and management of indexing algorithms (thus a variable set of indexing algorithms is possible for R1).
- The *Metadata Engine* (MDEcs) gives a global view of the system. It can contain some extracted metadata about multimedia contents, some contextual information about the system, the remote servers' descriptions, the descriptions of the available indexing algorithms, etc. It is a system knowledge repository that enables efficient solutions for multiple issues: algorithm selection according to a user query (for R2 and R3); filtering of the multimedia content for R5; distributed metadata management for R6; the selection of relevant remote servers for R7.
- The *Feature Extractors Manager* (FEMcs) manages the entire set of indexing algorithms available in the system. This module communicates with its equivalent on the remote server side in order to install new indexing algorithms if it is necessary or to ask for the execution of a certain indexing algorithm on a multimedia content, or part of multimedia content. Thus, the management of a variable algorithms set is possible for R1. Their remote deployment and execution is also possible for R4.
- The *Request Processor* (RP) treats some queries on the MDEcs or forwards them to specific remote server metadata engines. Moreover, through the FEMcs, it can decide to remotely deploy some indexing algorithms. Thus, the RP has an essential contribution to the content filtering and to the selection of the relevant algorithms and remote servers for R3, R5 and R7.
- The *Results Aggregator* (RA) aggregates the results received from all the queried metadata engines and sends them to the TI, which displays them.
- The *Translation* module homogenizes the data stored into the MDEcs coming from the MDErs, the remote SDCrs and the FEMcs. Hence, this module unifies all descriptions in order to provide the system global view.
- The *Time Central Server* provides a unique synchronization system time.
- The *Service Description Controller* (SDCs) collects all remote server descriptions (useful for enabling the remote servers selection for R5 and R7). It manages the integration in the system of new remote servers, their removal and the change in their functioning state (e.g., if the server is temporarily down or it is active).



### 3.3 Indexing and Querying Mechanisms

In order to reduce resource consumption, the architecture allows the multimedia contents indexation to be accomplished at acquisition time (i.e., *implicit indexation*) and on demand (i.e., *explicit indexation*). This avoids executing all indexing algorithms at once (thus a solution for R2 issue is available).

When a remote server acquires new multimedia content, the SM stores it and then the FEMrs starts its implicit indexation by executing a predefined set of indexing algorithms. This algorithm set is established according to the server particularities.

Once the execution of an indexing algorithm is achieved, the obtained metadata is forwarded to the Filtering module. The filtered metadata is then stored by the MDErs in its metadata collection. In order to avoid the transmission of the whole collection of metadata computed on the remote servers, the MDErs only sends, at a given time interval, a summary of these metadata to the Translation Module on the central server. [11] (the distributed metadata management is adopted for R6). Once translation is done, the metadata are sent to the MDEcs to be stored and further used in the querying process. Thus, the *implicit indexation* process is achieved.

The *query process* begins with the query specification through the TI. The user's query is sent to the RP module in order to be executed over the metadata collections. In this process, the RP analyses the query in order to select, based on the metadata summaries from MDEcs, the active remote servers that could provide answers to the query. Among the servers that were not thus selected there could be some servers that contain relevant information, but that has not been indexed with the suitable algorithms (the servers' selection relies upon their metadata summary, obtained mainly from the implicit algorithms' metadata; so, maybe among these algorithms there are not the most relevant for the current query). For this reason, our solution detects such supplementary algorithms [12] and starts their execution (i.e., *explicit indexation*) on a sub-collection of multimedia contents (developing thus efficient solution for R3 and R5). The query is sent for execution to all the selected servers. The top-ranked relevant results obtained from these remote servers are sent to the RA, which combines them in order to obtain a global ranked results list that is displayed to the user in the TI.

An important remark is that the two kinds of indexation can be mixed in the LINDO system, i.e., on some remote servers only the implicit indexation can be accomplished, while on others only the explicit indexation is done, and finally on others both indexation processes can be performed. The implementation of these two workflows will be detailed in the next section.

## 4 LINDO System Evaluation

As illustrated before, the LINDO framework was conceived to provide support for efficient handling of all the seven design requirements. The aim of the project was to build a real system, so we could evaluate our framework in different scenarios.

We further present the topology of the system employed in the evaluation. We will also illustrate with some examples how the multimedia indexing and the query processes are flexibly accomplished on this topology:



- (1) Multimedia indexing is performed locally, on each remote server, while being coordinated at the central server level;
- (2) Explicit indexing is employed only when necessary, namely when a query doesn't receive satisfactory results. Thus, for a certain query, all the suitable results are located and retrieved.

We will emphasize, while presenting this concrete implementation, how all the seven issues receive an efficient solution.

#### 4.1 The LINDO System Topology Used for Evaluation

In the development of the testing system architecture, we considered multiple remote servers, located in different countries that instantiate modules of the proposed architecture, and that concern different domains (video surveillance, broadcast).

The topology of the LINDO testing system is composed of a central server and three remote servers, located in Paris and in Madrid. Two remote servers are dedicated to video surveillance and they store, index and query video contents acquired in real time. The third remote server manages multimedia contents for the broadcast domain.

In the following, we detail the particularities of each architecture module instantiation on each one of these servers, either remote or central.

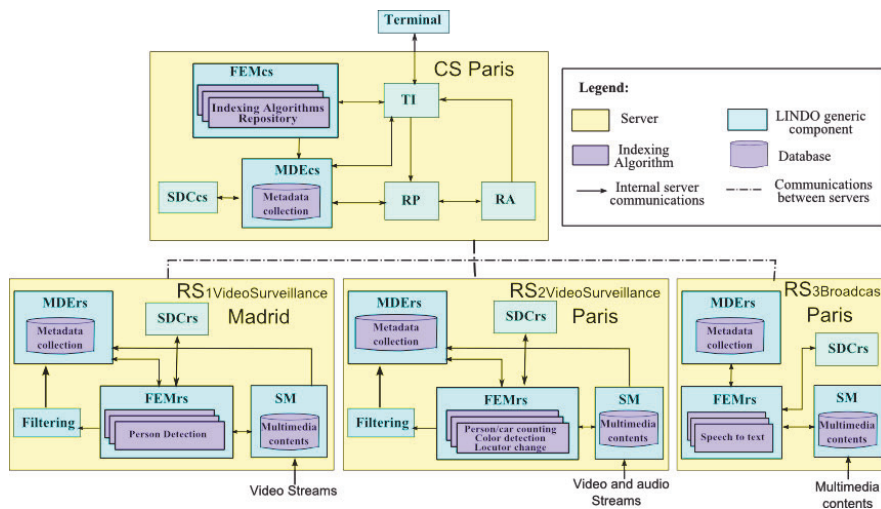


Fig. 2. The LINDO testing system topology

The generic architecture of a remote server was instantiated for each one of the three remote servers. The instantiations maintained the architecture's modules, while adopting a different implementation of their functionalities:

- For the *SM* module, a proprietary software developed in C language by one of the partners was adopted for a video surveillance remote server as well as for the

broadcast remote server; for the other video surveillance remote server, a software produced by another partner was employed. It manages the splitting, naming and storing manner of the multimedia content (Thales CCTV, WiLix).

- Similarly, two different implementations (in Java and C#) of the *FEM* module were adopted for the two video surveillance remote servers, while the broadcast and the central server employed the Java implementation;
- The same *MDE* module was integrated in all the three remote servers. This module was developed in Java and uses the XML native Oracle Berkley DB XML<sup>5</sup> database for storing the metadata;
- The *Filtering* module was not included in the broadcast remote server;
- A Java implementation of the *Service Description Controller* was instantiated on each remote server and on the central server as well.



This topology proves that the LINDO architecture enables each partner to develop his own implementation of each module, while respecting the interfaces and data format requirements.

The characteristics of each remote server in terms of multimedia content and implicit indexing algorithms are presented in the following.

**First video surveillance remote server, installed in Paris:**

- Manages multimedia contents acquired from two video surveillance cameras situated in a train station and watching the main hall and parking.
- Stores audio and video contents acquired in real time in the *SM* module, which in this case is the software developed in C language.
- Contains implicit indexing algorithms managed by the *FEMrs*. The indexing algorithms (executed on Windows and Linux environments) for video content are in charge with person and car counting and intrusion detection for indoor and outdoor environments, as illustrated in Table 1. For audio content, the speaker change detection is available.

**Table 2.** Indexing algorithms for video content on the Paris video surveillance remote server

	<b>Indoor</b>	<b>Outdoor</b>
<b>Intrusion</b>	- Presence of people	- Presence of people & vehicles
<b>Counting</b>	- Number of people - Main color of the upper part of the people	- Number of people, number of vehicles - Main color of the people upper part. - Main color of vehicles
		

<sup>5</sup> <http://www.oracle.com/technology/products/berkeley-db/xml/index.html>

- Handles the metadata provided by the indexing algorithms in a uniform XML data format [13] as well as the descriptions of the installed indexing algorithms. All this information is stored by the *MDErs*. A fragment of the person detection indexing algorithm description is shown in Table 2.

**Table 3.** XML algorithm description

```

<AlgorithmModel AlgoName="Person Detection" MediaType="Video">
  <InputParameters>
    <InputParamFileFormat>xml</InputParamFileFormat>
  </InputParameters>
  <ImageParameters/>
  <Feature>Local Semantic Features</Feature>
</InputParameters>
  <OutputObject Type="Metadata">
    <MetadataObject>
      <MetadataObjectDescription>location of the detected persons</MetadataObjectDescription>
    </MetadataObject>
  </OutputObject></AlgorithmModel>

```

- The *SDC* module contains an XML based description of the specific characteristics and context for this remote server (e.g., the IP address, the deployed indexing algorithms, the spatial topology of the location, the installed cameras and their characteristics). An example of such description is provided in Table 3.

**Table 4.** Remote server description

```

<RemoteServer id="rs1" name="Remote Server 1">
  <localisation>train station, Paris, France</localisation>
  <description>Manages content from cameras located in the main hall of the station and in the parking of the station</description>
  <devices>
    <camera id="c1Paris"> <description>located in the main hall </description> </camera>
  </devices>
  <indexingAlgorithms>
    <indexingAlgorithm id="ia2rs1" name="pedestrian detection" mediaType="video">
      <description>Detects pedestrians in a parking and their predominant color</description>
    </indexingAlgorithm>
  </indexingAlgorithms>
</RemoteServer>

```

**The second video surveillance remote server, installed in Madrid:**

- Manages and stores video contents acquired in real time from a video surveillance camera situated at the entrance into a security control room, using a software produced by a local partner;
- Contains an indexing algorithm for person and color detection in indoor environments. The description of this algorithm and its output follow the same formats as the algorithms installed on the Paris remote server;

- The *SDC* stores locally the description of the remote server, which is similar with the one provided in Table 3.

**The third remote server**, designed for the broadcast domain:

- Stores video content resulted from BBC journals using the same software for the SM as the video surveillance remote server from Paris;
- Contains a speech-to-text indexing algorithm based on Microsoft technology, which processes the audio stream of video files. The output of this algorithm follows the metadata format defined in the project.

The **Central Server** complies with the architecture presented in Figure 1 and has the following characteristics:

- FEMCs manages the indexing algorithm global collection where, alongside with all the implicit indexing algorithms installed on the remote servers, a supplementary set of explicit indexing algorithms are installed (abandoned luggage detection [14], shape detection, color detection, shout detection, etc. [15])
- MDECs manages multiple data: descriptions of each remote server, abstracts of the multimedia metadata from each remote server, descriptions of indexing algorithms.
- Contains also the TI, the RP and the RA modules.

## 4.2 Multimedia Indexing

The indexing algorithms enumerated above for each remote server are implicit indexing algorithms that are selected according to each server's characteristics. These algorithms index all the multimedia contents at the acquisition time. For example, on the remote server from Madrid only the algorithm for person detection in indoor environments is installed because it is a priori enough for processing the video captured with a camera at the entrance of a security control room. On the contrary, all the indexing algorithms presented in Table 1 are necessary on the video surveillance remote server in Paris because this server manages content from the main hall and parking of a train station.

These implicit video indexing algorithms produce metadata for each video frame. In order to reduce the size of the generated metadata, the *Filtering* module aggregates the metadata associated with consecutive frames that refer to the same detection. For example, for the Paris server, Table 4 contains the metadata obtained after the Filtering process was applied on the metadata generated by the person detection algorithm.

**Table 5.** Metadata aggregation result

```
<document src="stream1">
<video capturedBy="cam1_Paris">
<object type="Person" id="0">
<localisation confidence="100">
<period start_time="2010-07-28T11:07:35" end_time="2010-07-28T11:07:55"/>
<area>control room</area>
</localisation>
<property name="color">red</property>
</object>
</video> </document>
```

Periodically, a Web service sends to the central server a metadata summary that contains the essential detected information on each remote server (thus the R6 is handled). In our experiments we send this abstract each hour. Because we are dealing with multimedia contents from two different domains (i.e., video surveillance and broadcast) the metadata summary is built differently, according to the domain:

- for video surveillance: the summary consists in statistics based on the metadata obtained in the last hour of recording;
- for broadcast: the summary will be also accomplished on the metadata that was generated by the indexation of the multimedia content during the last hour, but will consist in the titles and participants for each broadcast content (article, show, etc.).

This summary is concatenated to the other information in the MDEcs, and thus a complete view of the system is obtained on the central server. This overview is the basis for further treatment of the user's query, in the context of explicit indexation, as detailed in the next section.

### 4.3 Query Processing

The user formulates the query in the TI through a graphic interface that enables him to specify five query components: the query itself (as free text), the location (free text), time span (calendar-based), domain (checkbox list) and the media format (video, image, audio or text).

As further detailed, the query is processed in order to select the remote servers (according R7) that are currently in a functional state (active), the sub-set of explicit indexing algorithms (R3), as well as the sub-collection of the multimedia content (R5). Figure 3 shows the logical steps that happen when a user queries the system.

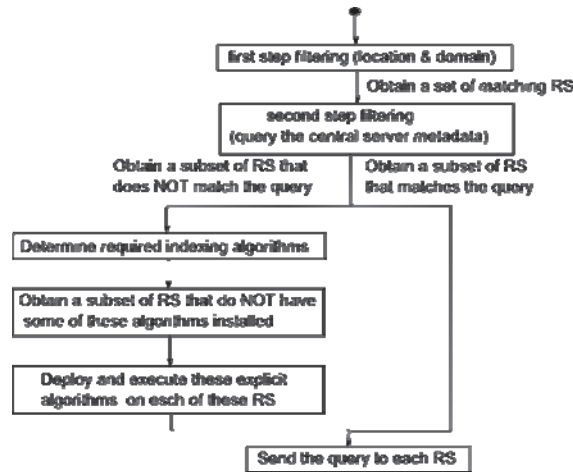


Fig. 3. Query processing diagram

In order to process the query, the first task of the RP is to select a set of active nodes on which the query will be executed. The selection is done into two steps. In

the first step every node that does not match for its location and domain with the user query (location and domain fields) is rejected. The set of remaining nodes then goes into the second filter. In this step, the user's query is applied to the metadata summary stored on the central server that corresponds to all of these remote servers. Two subsets of nodes will result, the first containing nodes that match the query, and the second one containing the remaining nodes that do not match the query.

The query is then directly sent to the first set of matching nodes. For the nodes in the second set, a list of relevant explicit algorithms is first determined. The required algorithms are found by means of similarity between each algorithm description and the user query. For each remote server, only the algorithms that have not been applied will be deployed and executed. This ensures that these nodes will also be able to provide a final answer to the query.

For illustrating these filtering operations during querying processing, we examine in the following the query mechanism on some concrete query examples.

*Q1. Location: Paris; Domain: broadcast; Time: 14 July 2010; Query content: Sarkozy speech.*

The first filtering step will select only the broadcast remote server from Paris (location is matched directly). During the second filtering step, the query content is searched inside the metadata summaries for the date of 14 July 2010 on the central server (the text "Sarkozy speech" is matched over these metadata, based on a semantic processing as will be presented at Q3). Supposing this search is successful, the query is further sent to the metadata collection from the Paris broadcast server. Based on this search, the concrete corresponding audio and video BBC news are located and provided as results.

*Q2. Domain: video surveillance; Time: 8 March 2011; Query content: woman in red.*

The first filtering step will select the two video surveillance remote servers, from Paris and Madrid. During the second filtering step, the query content is searched inside the metadata summaries for the date of 8 March 2011 from the central server (the text "woman in red" is matched over the metadata summaries corresponding to the 8 of March, according the semantic processing described at Q3). Supposing the both remote servers confirm the existence of such information, the query is sent to Paris and Madrid. The returned results are merged by the *RA* and presented to the user via the *TI*. It can be noticed that in both Q1 and Q2, the right branch of the diagram represented in Figure 3 is followed.

*Q3. Location: Paris; Domain: video surveillance; Time: 8 March 2011; Query content: abandoned bag by women in red.*

After first filtering, the Paris server is selected. In the second filtering step, the metadata summaries are queried (using the same technique as described below), but no result is obtained. This means that either no results actually exist, or on the Paris server the algorithms that detect persons and static objects have not been executed.

We have to determine the appropriate algorithms to be run based solely on the user's query. For this purpose, an analysis is first performed on the query to obtain so-called query chunks (usually, a chunk is a noun phrase composed of a noun and its

modifiers). A shallow parsing of the query will obtain two distinct chunks: “abandoned bag” and “women in red”. Then, for each chunk, plural nouns are inflected (“women”->”woman”).

Next, the query chunks are matched to every algorithm description available on the central server, which are themselves pre-processed in the same way. The matching is done separately for every query chunk. For example, the algorithm that has the description “stationary left or abandoned luggage or object” will match query chunk “abandoned bag” due to the exact adjective match “abandoned” and to the fact that “luggage” matches “bag” because both have “container” as hypernym. The chunk-chunk match score is obtained by computing similarity between words, using the JCN - Jiang and Conrath [16] similarity measure between their synonyms and also by matching adjectives’ and adverbs’ synsets using WordNet<sup>6</sup>. JCN was chosen among other similarity measures because of its better performance [17]. The same applies for the person detection algorithm having the description “person and color detection algorithm” that matches to the query chunk “women in red” due to the fact that “woman” is a “person” and “red” is a “color” (direct hypernym relations mean good JCN score). The final score for each algorithm is the sum of the highest similarity scores between algorithm chunks and query chunks. This avoids score imbalance due to variable algorithm description lengths. The top candidate algorithms are chosen.

A check is performed to see if they had already run on the selected remote server. If every candidate algorithm has already run, that means that the initial search in the metadata on the central server yielded correctly no results. In our case, the algorithm that detects stationary objects was not run, and it is deployed for processing on the remote server.

## 5 Conclusions

In this paper we presented a framework that supports the design of an efficient distributed multimedia system by minimizing resource consumption while providing the most relevant results in the shortest time.

This framework was developed in the context of the LINDO project, and acquires efficiency in multimedia indexing and retrieval through three particularities: (1) it differentiates between implicit and explicit indexation; (2) it processes multimedia content locally, instead of transferring and indexing it on a central server; (3) it selects a set of relevant servers for query execution. The paper presented also a concrete implementation of the LINDO framework, which validates this contribution.

In the future, we will study how the LINDO indexing and retrieval mechanisms could be applied on some existing multimedia distributed repositories in order to intelligently handle their knowledge. In order to improve these mechanisms, we also plan to develop semantically enhanced algorithm descriptions that will enable to define better criteria for algorithm selection.

**Acknowledgments.** This work has been supported by the EUREKA project LINDO (ITEA2 – 06011).

---

<sup>6</sup> <http://wordnet.princeton.edu/>



## References

1. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, Hardcover (2011)
2. Petkovic, M., Jonker, W.: Content-Based Video Retrieval, A Database Perspective. *Multimedia Systems and Applications*, vol. 25 (2003)
3. Pietarila, P., Westermann, U., Järvinen, S., Korva, J., Lahti, J., Löthman, H.: CANDELA - storage, analysis, and retrieval of video content in distributed systems. In: *The IEEE International Conference on Multimedia and Expo*. pp. 1557–1560 (2005)
4. Merkus, P., Desurmont, X., Jaspers, E.G.T., Wijnhoven, R.G.J., Caignart, O., Delaigle, J.-F., Favoreel, W.: Candela- Integrated storage, analysis and distribution of video content for intelligent information system. In: *European Workshop on the Integration of Knowledge, Semantics and Digital Media Technology* (2004)
5. Conan, V., Ferran, I., Joly, P., Vasserot, C.: KLIMT: Intermediations Technologies and Multimedia Indexing. In: *International Workshop on Content-Based Multimedia Indexing*, pp. 11–18 (2003)
6. Giroux, P., Brunessaux, S., Brunessaux, S., Doucy, J., Dupont, G., Grilheres, B., Mombroun, Y., Saval, A.: Weblab: An integration infrastructure to ease the development of multimedia processing applications. In: *the 21st Conference on Software & Systems Engineering and their Applications* (2008) (published online)
7. Viaud, M.-L., Thievre, J., Goeau, H., Saulnier, A., Buisson, O.: Interactive components for visual exploration of multimedia archives. In: *The International Conference on Content-based Image and Video Retrieval*, pp. 609–616. ACM Press, New York (2008)
8. Thong, J.M.V., Blackwell, S., Weikart, C., Hasnian, A., Mandviwala, A.: *Multimedia Content Analysis and Indexing: Evaluation of a Distributed and Scalable Architecture*, Technical report, HPL-2003-182 (2003)
9. Roth, V., Peters, J., Pinsdorf, U.: A distributed content-based search engine based on mobile code and web service technology. *Scalable Computing: Practice and Experience* 7(4), 101–117 (2006)
10. Hinds, N., Ravishankar, C.V.: Managing metadata for distributed information servers. In: *The 31st Hawaii International Conference on System Sciences*, pp. 513–522 (1998)
11. Laborie, S., Manzat, A.-M., Sedes, F.: Managing and querying efficiently distributed semantic multimedia metadata collections. *IEEE MultiMedia Special Issue on Multimedia-Metadata and Semantic Management* 16, 12–21 (2009)
12. Brut, M., Laborie, S., Manzat, A.-M., Sèdes, F.: A Framework for Automatizing and Optimizing the Selection of Indexing Algorithms. In: Sartori, F., Sicilia, M.Á., Manouselis, N. (eds.) *MTSR 2009. Communications in Computer and Information Science*, vol. 46, pp. 48–59. Springer, Heidelberg (2009)
13. Brut, M., Laborie, S., Manzat, A.M., Sedes, F.: A Generic Metadata Framework for the Indexation and the Management of Distributed Multimedia Contents. In: *The Third International Conference on New Technologies, Mobility and Security*, pp. 1–5 (2009)
14. Gasteratos, A., Vincze, M., Tsotsos, J., Mieziako, R., Pokrajac, D.: Detecting and Recognizing Abandoned Objects in Crowded Environments. In: *Computer Vision Systems. LNCS*, pp. 241–250. Springer, Heidelberg (2008)
15. Snoek, C.G., Worring, M.: Multimodal video indexing: A review of the state of the art. *Multimedia Tools and Applications* 25, 5–35 (2005)
16. Varelas, G., Voutsakis, E., Raftopoulou, P., Petrakis, E.G.M., Milios, E.E.: Semantic similarity methods in wordNet and their application to information retrieval on the web. In: *the 7<sup>th</sup> International Workshop on Web Information and Data Management*, pp. 10–16 (2005)
17. Jiang, J., Conrath, D.: Semantic similarity based on corpus statistics and lexical taxonomy. In: *International Conference on Research in Computational Linguistics*, pp. 19–33 (1997)