

A GRAPH BASED APPROACH TO AUTOMATICALLY CHAIN DISTRIBUTED MULTIMEDIA INDEXING SERVICES

Bassem Haidar, Philippe Joly, Siba Haidar

IRIT, Institut de Recherche en Informatique de Toulouse
118, route de Narbonne - 31062 Toulouse Cedex 04 - France
{haidar, joly, shaidar}@irit.fr

ABSTRACT

This paper proposes a distributed platform for automatic multimedia indexing; the platform allows the automatic composition of multimedia indexing tools. Our goal is to automate and simplify the production of some complex multimedia description and to provide more flexibility in the collaboration between multimedia indexing research team.

A central server allows the control and collaboration of the distributed indexing tools. Web services are used to access multimedia indexing tools. The communications are based on SOAP-RPC calls.

We propose a chaining algorithm based on the Input/Output data types. This automatic indexing algorithm aims at automatically finding a subset of tools and their interactions to produce an index required by a user. The algorithm produces an execution graph representing the expected chain.

KEY WORDS

Distributed Multimedia Systems, Automatic Multimedia Indexing.

1. Introduction

In the multimedia indexing domain, many indexes need to be produced by a sequence of more or less sophisticated tools in a given order.

Taking for example the indexing of audio documents; in order to make the transcription of the speech in an automatic way, we must first classify the audio content type (in speech, music, Noise and silence), then we can identify the language of a speaker in the case of a speech content, and then we can try to transcribe the speech into text with the transcription tool corresponding to the right language.

Multimedia indexing tools are in general developed and distributed by different research teams with really specialized skills on a single media. This makes the exploitation of these tools for cross-media analysis a challenging issue for potential research activities on multimedia data mining.

The integration of multimedia indexing tools into cooperative applications requires the development of an

intelligent platform in which the interoperability of different indexing tools is defined.

In this paper, we propose a distributed platform for collaborative multimedia indexing that allows automatic tools composition. These tools are modeled as processes with some different potential customized executions according to the client needs. A composite tool is modeled by a graph. The latter defines the order of execution of the different available processes. We implemented on this platform a functionality to generate an index in a completely automatic way. Given an index (i.e. a given type of metadata expected by an end-user), the chaining algorithm finds a subset of tools and determines their interactions to extract the requested information from any type of available sources (audio, video and textual contents). Input data is not defined by the user's query. All the documents available on the platform are potential candidates to be indexed and are selected automatically by the chaining algorithm.

The automatic tool composition has the advantage of reducing the human interaction needed to produce complex indexes. We use SOAP-RPC [1] to access the indexing tools; SOAP has the advantage of posting data over the HTTP protocol. So, it is able to get through firewalls which are a limitation for many distributed applications.

2. State of the art

Collaborative multimedia indexing works have focused, in general, on using a static, manually built, chain of multimedia tools to generate an expected index. Among all the projects developed during this last decade and dealing with that kind of problem, let us mention the KLIMT (Knowledge InterMediation Technologies) project [2] which proposes an open framework that allows the integration of multimedia indexing tools. The framework implements communication and data transfer mechanisms. The KLIMT platform proposes some scenarios for chaining multimedia indexing tools. The chaining process is limited to some predefined scenarios, which are written as scripts. The FERIA project [3] aimed at developing a general and open framework allowing the easy development of applications for editing interactive video documents. The collaboration between tools was here still static and manual.

The Acoi project [4] presents a system that combines independent feature detector programs with a multimedia database technology, to index multimedia objects on the World Wide Web. A grammatical framework called “feature grammars” is used as a description of the execution sequence of the indexing program.

The chaining process is used to compose web services. The order of messages exchanged between services must be described separately in a flow specification. There are many Web services flow specification languages like WSCI [5], BPML [6], BPEL4WS [7], XLANG [8], and WSFL [9].

An open framework is proposed in the DCS [12] (Distributed Software Component). It allows the integration of news component into a distributed framework. The integration is based on a specification file containing the component properties and on a wrapper that allows the component to be linked to the platform.

3. Platform architecture

The platform is composed of distributed services and a central server that implements two principal services: the repository service and the automatic indexing service.

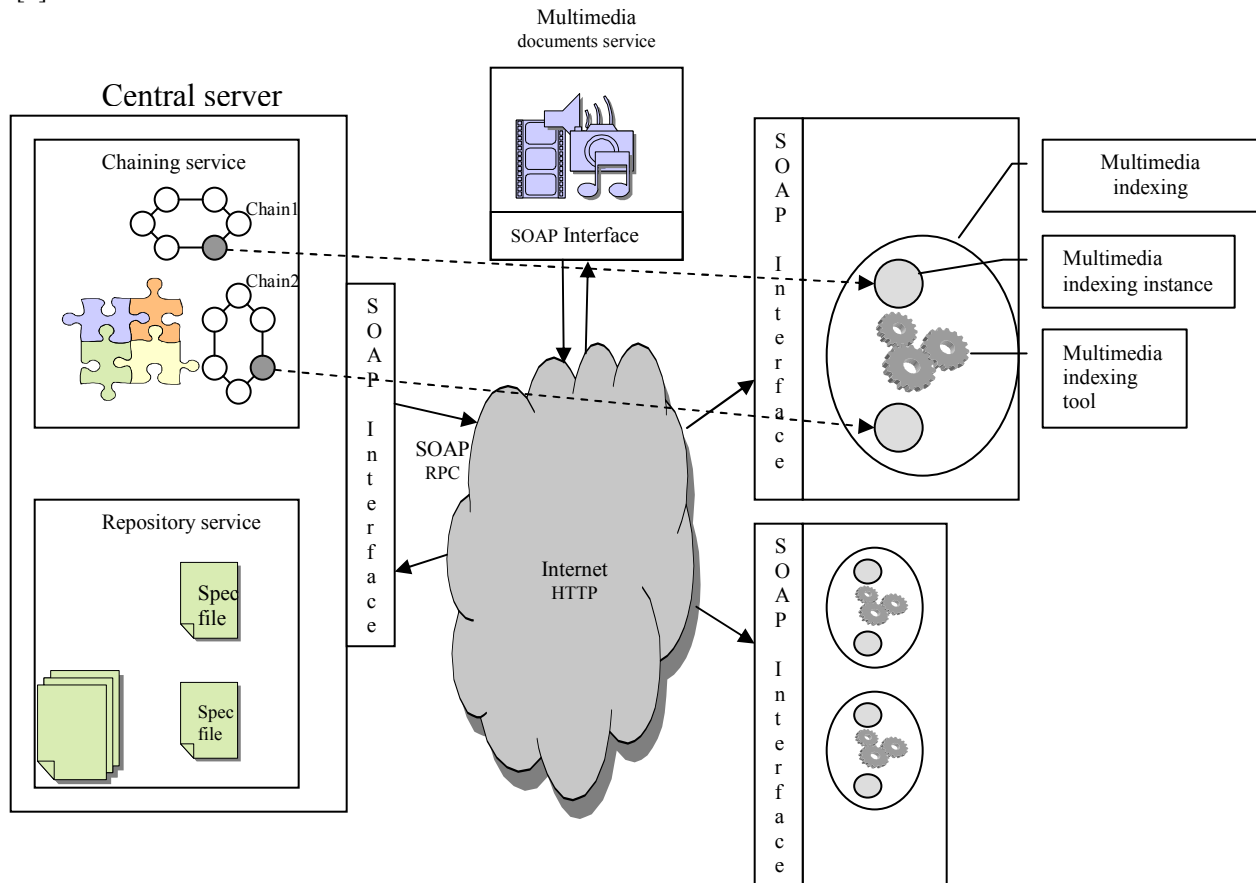


Figure 1 Global architecture

The exact control and the data flow that determines when an operation can be executed are described using a flow composition language. Thus, the composition of the flow is still manually obtained [10].

eFlow [11], a composite service is described as a process schema that composes other basic or composite services. A schema is modeled by a graph (the flow structure), which defines the order of execution among the nodes in the process. The graph can include service, decision, and event nodes. Service nodes represent the invocation of a basic or composite service; decision nodes specify the alternatives and rules controlling the execution flow, while event nodes enable service processes to send and receive several types of events.

3.1 Implementation details

We use the SOAP [1] protocol for the RPC (Remote Procedure Call), the word SOAP stands for Simple Object Access Protocol. SOAP has been developed by Microsoft, IBM, DevelopMentor and UserLand Software and is recommended by the Internet Engineering Task Force (IETF). SOAP is used to communicate between applications via HTTP using XML (or some other Markup Language).

SOAP is used primarily to make remote procedure calls between computers over an open network. It presents the advantage of posting data over the HTTP protocol, which means that the delivery mechanism is widely.

A SOAP message is written using the XML formalism and is sent over HTTP to a remote server. Where the Web

service is located, the Web service processes the SOAP request and returns the value to the client using a SOAP response.

Each RPC invocation is modeled as an outer element which matches the name of the required operation, and containing inner elements used as parameters of this operation [13].

We use Apache Axis [14] for the development; *Axis* (*Apache eXtensible Interaction System*) is a new implementation of the SOAP specification, and is a toolkit for development and creation of web services.

3.2 Repository service

The central server implements the repository service that allows the registration of new services on the platform.

A new tool will be integrated on the platform by sending towards the storage service the information contained in its specification file. This information will be available for the other services connected to the platform. The repository service allows a tool, defined by a set of attributes, to be retrieved by a searching mechanism. This mechanism browses the information located in all the specification files.

Semantic types are inspired from the ones defined by the MPEG-7 standard when it is possible. Other semantic types have to be added to provide pieces of description which are not available in MPEG-7. The composition process is based on a chaining algorithm (see Section 4).

3.3 Automatic chaining service

The automatic composition problem stands in defining and assuming the interoperability between available tools on the platform (i.e. declared in the repository) to build a given index. Our idea is to describe the Input/output data type of each indexing tool with a semantic description. For instance, the goal of an audio analyzer [15] we intend to plug on the platform is to localize segments of speech, music, noise and silence in an audio file. The integration of this tool in an automatic chain is based only on the Input/Output data type, i.e. a .wav file as input and an *AudioSegmentContent* type as the output [Figure 2].

3.4 Multimedia indexing services

3.4.1 Indexing service

An indexing service can interface one or several indexing tools, developed by a single research team or specialized in a single media like (audio segmentation, audio transcription), the indexing service must allow a user or another indexing service to use one of its indexing tool. Each indexing service is identified by a service identifier by the server side, and contains a table of “indexing tool representation” objects. Services have a SOAP-PRC interface that implements the service functionality.

3.4.2 Indexing tool representation

An indexing tool representation is the representation of one indexing tool in the indexing service; it is created by

the central server. More precisely, the scenario of adding new tool to the platform is the following one:

the owner or the developer of the tool ask the repository service to register his tool, by sending towards the repository service the specification of his tool, and the address of the indexing service that will host his tool; then the central server will create an identifier for this tool, contact the specified service, and create an indexing tool representation. The indexing tool representation will be stored in a table by the indexing service side.

3.4.3 Indexing tool instance

As we said in the introduction of this paper, the goal of the platform is to generate new multimedia descriptions by chaining automatically several tools.

An indexing tool may be used by different chains simultaneously. So, for each chain, we create a specific indexing tool instance that allows executing a same indexing tool in different chains. An indexing tool instance has an identifier which is the chain identifier. Figure 1 explains the role of each object in a service.

The indexing tool instance has another principal functionality, it plays the role of a wrapper between the tool and the platform, it allows to launch an indexing tool. This tool is in general an executable program characterized by a specification file written by the owner or the developer of the tool. The specification file contains information needed to execute this tool like the number and the type of input data, the number and the type of the output data, the options needed for the execution, etc.

The indexing tool instance read the specification file of the tool and constructs the command line needed to launch it:

```
Prog_name option1 input1 input2 output1 output2
```

The following example is a specification of an audio segmentation tool:

```
<Component ComponentRole="Index">
<Description>To localize segments of Speech, Music,
Noises and Silences in an audio file</Description>
  <Authors>J.Pinquier</Authors>
  <Version>v123.2b</Version>
  <Input>
    <InputData>
      <FileFormat>wav</FileFormat>
      <StorageInputPath>
        ftp://aljolson.irit.fr/Pidot/Input
      </StorageInputPath>
    </InputData>
  </Input>
  <Output>
    <OutputData>
      <FileFormat>xml</FileFormat>
      <ContentType>AudioSegmentContent
      </ContentType>
      <StorageOutputPath>
        ftp://aljolson.irit.fr/Pidot/output
      </StorageOutputPath>
    </OutputData>
  </Output>
</Component>
```

Figure 2 specification file of an audio segmentation tool

The indexing tool instance plays the role of a generic wrapper. It is independent from the type of the tool, and only the specification file is required.

The figure 1 resume the global architecture of the platform; on the left side we have the central server. Its two principal functionalities are the repository and the chaining service. In the middle, we have a multimedia documents service (refer to section 3.5). On the right side, we have distributed services. Each service interfaces one or several indexing tool. Each tool has an indexing tool representation created by the platform, and finally, each tool can have one or more indexing tool instances, each of which representing the tool in a given created chain

3.5 Multimedia document service

As for multimedia indexing service, a user can add a multimedia document service to the platform; a SOAP-interface is defined to transfer data from and to this service.

We use a `DataHandler` [16] object to transfer the multimedia content. With this `DataHandler`, the Java Activation Framework provides a serialization and deserialization functionality and attaches it automatically to the sent SOAP message. The data file is not directly integrated in the SOAP envelope, but as a reference to an attachment. The entire SOAP request is sent as a multipart MIME-encoded request. The first part is the SOAP envelope, and the second part is the attachment. This is basically the same format that allows sending attachments in e-mail messages. When the server receives the multipart request, it automatically understands how to find the referenced attachments and make them available to the SOAP service.

Transferring data via SOAP can not be achieved for large amounts of data; in a further step of development, we will integrate FTP mechanisms to allow the transfer of large documents such as full-length videos for example.

4. Workflow construction

The composition of web services can be achieved with some composition languages like WSCI [5], BPML [6], BPEL4WS [7], XLANG [8], and WSFL [9]. Each language has in general an XML specification, and there are some commercial and open source workflow engines [17] that allow the execution of a specified workflow. For example, the BPWS4J [18] is a platform upon which can be executed business processes written using the composition language BPEL4WS. For each process, the BPWS4J engine interprets a BPEL4WS document and executes the described workflow. Current workflow languages transform a workflow specification written by an operator or generated by a graphical editor into an execution graph. In our case, the chaining algorithm will construct a DAG (Direct Acyclic Graph). So, we do not need a specific language to describe the workflow. The preceding languages are created to allow the human creation of a chain, while in our case the chain is created automatically by the chaining process.

4.1 Distributed workflow execution

The chaining process produces an abstract chain. Each node represents a distant indexing tool available as a distant indexing service. By the indexing service side, an indexing tool instance is created and has the same identifier as the chain. Its goal is to launch the tool to generate the desired output. For the execution of the chain, we propose a distributed method; we start with the first indexing tool instance at the beginning of the graph. The generated result will be sent to the following indexing tool instances that correspond to the following nodes in the graph. To achieve this, each indexing tool instance knows its predecessors and its successors in the chain. When an indexing tool instance has more than one predecessor, an ending indicator is associated with each predecessor. The current indexing tool instance can't be launched until the ending indicators of all predecessors are true;

```
(Finish_indicator_prec1)&(Finish_indicator_prec2
) &... = true
```

5. Automatic chaining algorithm

A recursive algorithm allows the automatic construction of a composite indexing tool. Based on a given requested index (which is a descriptor asked by the user), the chaining algorithm determines the set of tools and their interactions which will lead to detect or recognize each occurrence of this descriptor. The algorithm stops when a primitive multimedia content (audio, videos files) can be used as an input of the last integrated indexer.

Starting with a given "*Index*", it tries to find the unitary indexer able to generate this descriptor type, using information in the repository service. Once such a tool is found, if the input type of this indexer is a multimedia content, it stops and the index can be generated on all documents corresponding to the input data type. In the other case, if the input type of this indexer is another descriptor type corresponding to data generated by another tool, it repeats the same process recursively, integrating each time a new indexer at the top of the chain, until a multimedia content type is found.

The process is simple in the case of linear chains, i.e. there is only one input and one output for each involved indexer. But it will be much more complicated if the involved tools require several data inputs. In this case, the chains will take the aspect of a graph and then the recursive algorithm is applied to each branch of this graph with the assumption that the graph is devoted to index only one source (i.e. one document at a time). In other words, the graph must have one and only one node at the top. The recursive building of the graph has to converge to a same source.

6. Implementation

6.1 Algorithm simulation

We implemented the chaining algorithm, and tested it by simulating a random number of fictive indexers, each of which has random input and output types. Only the

maximum number of inputs and outputs and the number of data source types are fixed.

First, we have generated fictive indexers called “operators”. Then, we have constructed the dependency graph corresponding to a requested data type. This graph contains all possible paths constructed starting from the root (i.e. the requested type) of the graph.

Each node on the graph corresponds to one operator. Thus, each operator in the platform, which may interfere in the path construction, is represented by a node.

The constructed graph has the structure of AND/OR graph [19], where the AND edges represent the needed connections, and the OR edges represent choices between possible connections. For example an operator which need two different input types, must have two connections (AND edges), and each type can be generated by multiple operators (OR edges). Figure 3 shows the data structure used to represent the constructed graph.

The first column is a vector containing all operator identifiers. For example, the first operator (first cell) has three input types Input1, Input2 and Input3. Each of these types can be generated by multiple operators. For example, Input1 can be generated by operator 2 or 3 or 5.

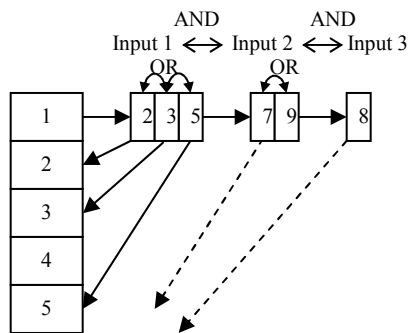


Figure 3 Graph structure in the simulation process

Once the dependency graph has been built, all possible chains must be extracted. Then, these chains are filtered in order to keep only convergent ones.

A chain is considered to be convergent if:

- it is not cyclic (i.e. one node can not be one of its children),
- all nodes in this chain have satisfied the graph AND_edges,
- and finally this chain converges to one and only one data source type.

Here is a result obtained for a given set of the graph parameters.

Max input	Max output	Operator Number	Generated type	Data source	Generated chains
2	2	6	5	2	2

Max input: maximum number of input data of an operator.

Max output: maximum number of generated data by an operator.

Operator Number: number of different operators.

Generated types: number of metadata types (including datasource types).

Data source: number of available data source types.

Generated chains: the average number of generated chains.

This table shows that for one requested output type, the dependency graph generates a mean of 2 different chains which are producing this kind of data with the given parameters. The mean value of the chain number is obtained over 15 randomly generated graphs built with the same parameters. This highlights the fact that, even if the number of potential chains which can be generated by the dependency graph is important, only a very few ones are convergent and so can be applied as indexing services.

6.2 Architecture implementation

The first experience of that approach has been made in the KLIMIT-ITEA [20] project. The general goal of this project was to develop a collaborative technology to allow intermediation between standards services. In this project, we implemented some predefined indexing scenarios. We used the wrapper explained in the indexing tool instance paragraph to launch indexing tools developed by different research teams. This allows a cross-media analysis of multimedia descriptions generated by several tools developed by different research team.

Scenarios were specified using a simple xml specification, and were executed thanks to a specific XML interpreter. The two following sections describe two of the operating scenarios tested on the KLIMIT framework.

6.2.1. Audio visual scenario

The audiovisual indexing scenario [Figure 4] aims at extracting information from an audiovisual document in order to make textual description about the document. Starting from an audiovisual document, an audiovisual splitter splits the document into its two basic components: the audio and the video stream. Then, each of them is to be processed in a specific chain.

A Speech/Music/Noise segmentation tool extracts the segments of speech from the document, then a language identification tool extracts French and English segments, each of them (if they exist) will be processed by the adequate transcription tool in order to extract the textual information included in these segments. Finally, a topic detection tool will search a given topic defined by the user.

For the video documents a visual features extraction tool will extract many descriptions like dominant colours, the activity rate, shot boundaries and keyframes. And then, each keyframe will be processed by two different tools. A text extraction tool extracts textual information existing in the image while a face detection tool determines if there is a human face or not in this frame. Finally all the descriptions will be collected in one xml file.

6.2.2. Video surveillance scenario

This scenario [Figure 5] aims at building an alarm system, in order to detect some predefined gestures. Images are captured by a network camera. Then, a motion detection tool analyses images in order to detect movement. When a movement is detected, images are processed by a gesture recognition tool which launches an alarm if the detected movement corresponds to one of the predefined gesture.

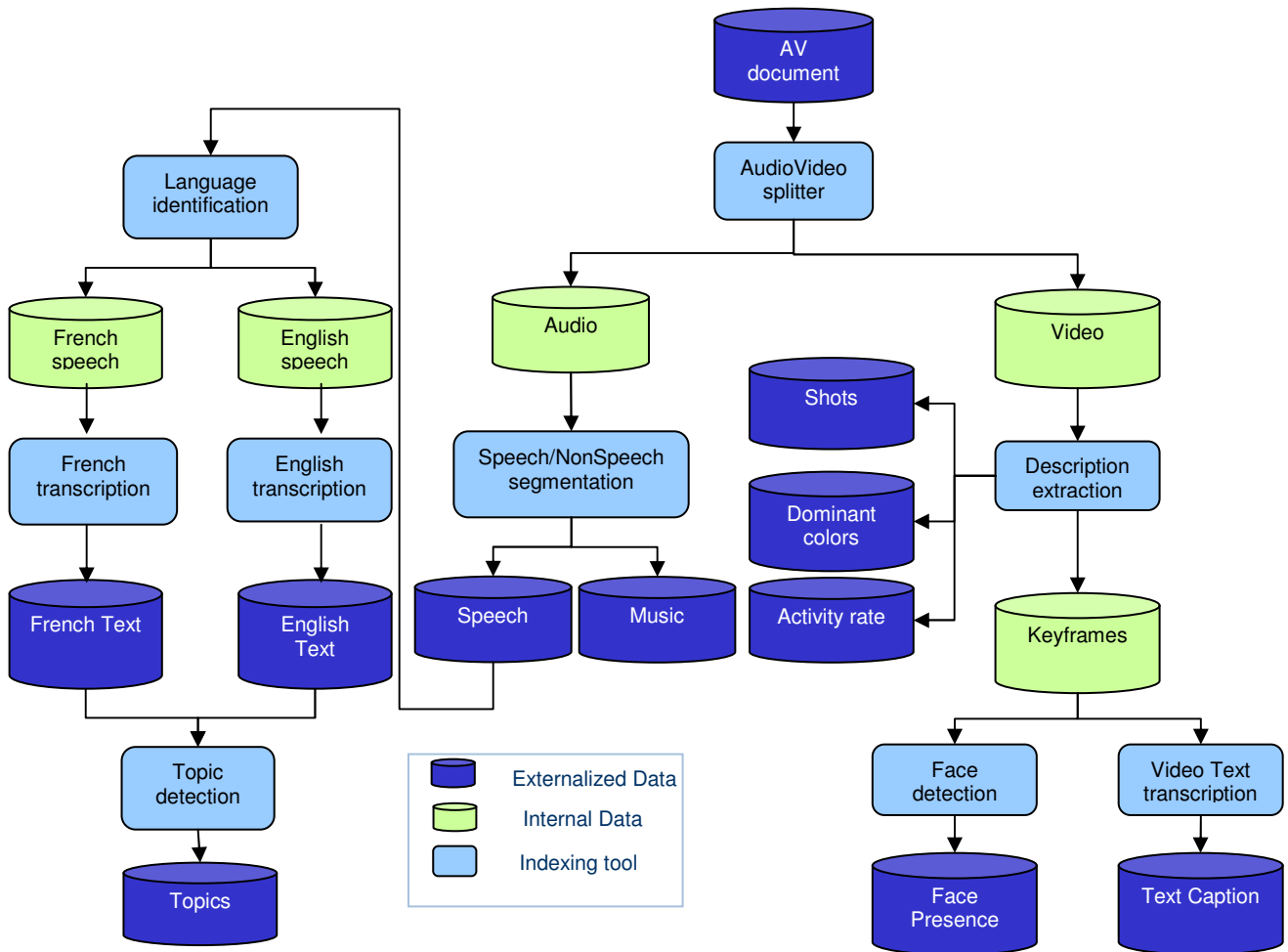


Figure 4 block diagram of the audiovisual indexing scenario

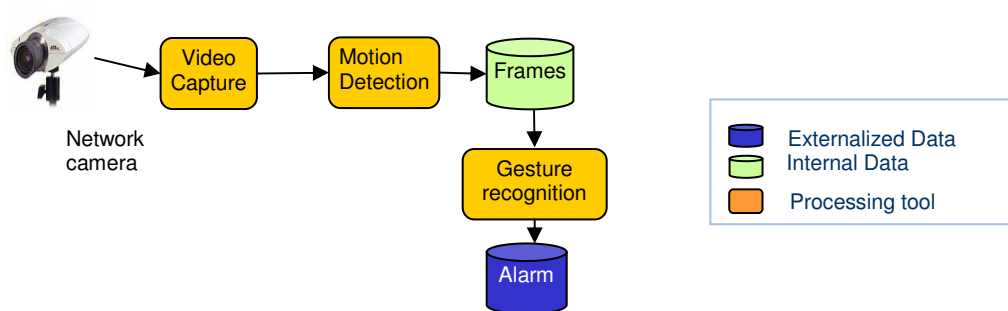


Figure 5 video surveillance scenario

7. Conclusion and future works

In this paper, we have introduced the main concepts of a distributed platform for automatic collaborative multimedia indexing, which supports a automatic policy of multimedia indexation. We have presented the global architecture of the platform, and the services that provides the communication mechanisms. We have proposed a recursive algorithm for automatically chaining indexing tools, based on the compatibility between input and output data types of available tools. We intend to apply this technology for the implementation of the Pidot project, which aims at gathering on a same platform indexing tools developed by five different laboratories.

The validation of the automatic indexing approach will be achieved on the Pidot platform. More than 30 different indexing tools will be plugged on this platform in order to analyze audiovisual contents. This kind of framework should allow us to mine new indexing process, and evaluate a large set a primary tools on official test beds.

Among expected results, we want to identify the best chain between different concurrent solutions to produce one given descriptor or, with the integration of generic aggregation tools, to automatically mine chains which can produce new descriptors.

More research efforts have to be achieved in the future to solve problems related to the automatic indexing approach. The first problem is the consistence of the generated chain. For example, in the case of automatic transcription of French spoken contents, the algorithm will allows to build the chain able to generate a textual description starting from an audio file (.wav), even when some of these audio files contain only spoken English. In this case, time to time, considering the language used in the document, the chain will not generate any result. So we have to define mechanisms able to inform of this inconsistency regarding to the contents and to stop the execution of the chain in that case. The second problem is the choice between different tools or chains that have the same output data type. For the moment the only criteria to select an indexing tool is its output data type. But we can imagine that more than one tool may be able to produce the same output type. So we should define some other criteria to make the best choice. The third problem is the evaluation of the whole chain; we want to know in advance the quality of the result generated by the chain, in order to launch or not its execution. The fourth problem is the monitoring of the chain. As far as the workflow is executed in a distributed manner, we have to know the state of the execution of each indexing service instance at any moment, and we should have to be able to stop the execution if problems occur.

Finally we have to define a chain updating mechanism which would allow to automatically change the chain after the integration or the removal of an indexer.

8. References

- [1] [HTTP://WWW.W3.ORG/TR/SOAP/](http://www.w3.org/TR/soap/)
- [2] V. Conan, I. Ferrané, P. Joly, C. Vasserot. KLIMIT: Intermediations Technologies and Multimedia Indexing. In Third International Workshop on Content-Based Multimedia Indexing (CBMI'03), Rennes, France, septembre 2003. INRIA, 11-18.
- [3] FERIA
<http://www.ina.fr/recherche/projets/encours/feria/>
- [4] M. Windhouwer, A. Schmidt, M. Kersten. Acoi: A System for Indexing Multimedia Objects. In International Workshop on Information Integration and Web-based Applications & Services, November 1999, Yogyakarta, Indonesia.
- [5] WSCI: A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawagushi, D. Orchard, S. Poliglioni, K. Riemer, S. Struble, P. Takacs-Nagy, I. Trickovic, and S. Zimeck. Web Service Choreography Interface 1.0.
- [6] A. Arkin et al. Business Process Modeling Language (BPML), Version 1.0, 2002.
- [7] Business Process Execution Languages for Web Services, Version 1.0. IBM Report.
<http://www.ibm.com/developerworks/library/ws-bpel/>
- [8] S. Thatte. XLANG Web Services for Business Process Design, 2001.
- [9] Web Service Flow Language
www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
- [10] J. Koehler, B. Srivastava, Web Service Composition: Current Solutions and Open Problems, ICAPS 2003 Workshop on Planning for Web Services, 28 - 35.
- [11] Fabio Casati, Ski Ilnicki, Li-jie Jin, Ming-Chien Shan, An Open, Flexible, and Configurable System for Service Composition. WECWIS 2000, 125-132
- [12] H.J. Batteram and H.P. Idzenga, A Generic Software Component Framework for Distributed Communication Architectures, ECSCW 97 Workshop on Object Oriented Groupware Platforms, 1997, Lancaster UK, 68-73
- [13] <http://ws.apache.org/axis/java/user-guide.html>
- [14] <http://ws.apache.org/axis/>
- [15] J. Pinquier, J. L. Rouas, R. André-Obrecht, Robust speech/music classification in audio documents. In International Conference on Spoken Language Processing (ICSLP'2002), Denver, USA, September 2002, 2005 -2008, Vol.3.
- [16] <http://java.sun.com/j2ee/1.4/docs/api/javax/activation/DataHandler.html>
- [17] Open source workflow engine written in Java
http://www.manageability.org/blog/stuff/workflow_in_java/view
- [18] <http://www.alphaworks.ibm.com/tech/bpws4j>
- [19] J. Pearl, *Intelligent search strategies for computer problem solving*, 1985.
- [20] ITEA: Information Technology for European Advancement. <http://itea-office.org>